

SOFT COMPUTING

استاد درس:

دکتر محمد رضا اکبرزاده توتونچی



F F 6 9 9 8 4 0 8 F 6 9 9 1 5 0 0 F 4 4 6 0 0 3 4 F 4 1 4 3 5 3 0 0 0 1 0 0
B F F 6 9 9 1 0 0 0 0 3 5 4 6 0 0 0 2 5 4 3 4 7 5 6 3 7 0 0 0 0 0 0 6 0 0 6 9 7 0
I 3 6 5 4 6 0 0 0 0 4 6 0 6 0
5 7 7 8 3 5 9 9 0 0 0 0 0 0 0 4 1 1 1 1 1

2 0 0 2



فهرست مطالب

عنوان	صفحه
فهرست	۱
مقدمه	۵
۱. سیستم های هوشمند	۶
۱-۱. چرا محاسبات نرم (soft Computing)؟	۶
۱-۲. مقدمه ای بر محاسبات نرم	۷
۱-۳. تعریف پروفسور زاده از محاسبات نرم	۱۰
۱-۴. آینده محاسبات نرم از دیدگاه پروفسور زاده	۱۱
۱-۵. سیستم های هوشمند چه هستند؟	۱۳
۱-۶. چند کاربرد	۱۴
۱-۶-۱. Fuzzy Neural Network	۱۴
۱-۶-۲. Fuzzy PID System	۱۴
۱-۶-۳. Fuzzy & Genetic System	۱۴
۱-۷. مقایسه فازی، شبکه عصبی و هوش مصنوعی	۱۵
۱-۸. سیستم هوشمند و کنترل هوشمند	۱۵
۱-۹. مدل سازی به روش کلاسیک	۱۶
۱-۹-۱. گام اول: جمع آوری اطلاعات	۱۶
۱-۹-۲. گام دوم: شناسایی ساختار	۱۶
۱-۱۰. کنترل کلاسیک در مقابل کنترل هوشمند	۱۷
۱-۱۱. پیشرفت تاریخی	۱۷
۱-۱۲. فرموله سازی مساله	۱۸
۱-۱۳. مشکلات سیستم های کنترل کلاسیک	۱۹
۱-۱۴. سیستم هوشمند	۲۰
۲. منطق فازی	۲۱
۲-۱. مجموعه های فازی	۲۱
۲-۲. تقسیم بندی فازی (Fuzzy Partitioning)	۲۳
۲-۳. واژه نگاری توابع عضویت	۲۳
۲-۳-۱. هسته (Core)	۲۳
۲-۳-۲. پشتیبان (Support)	۲۳
۲-۳-۳. نقطه تبدیل (Crossover point)	۲۴
۲-۳-۴. برش α (α -cut)	۲۴

۲۴		تحدب	۵-۳-۲
۲۵		زیر مجموعه بودن	۶-۳-۲
۲۵		مکمل	۷-۳-۲
۲۵		اجتماع	۸-۳-۲
۲۵		اشتراک	۹-۳-۲
۲۶		<i>MF Formulation</i>	۴-۲
۲۶		مثلثی	۱-۴-۲
۲۶		دوزنقه ای	۲-۴-۲
۲۶		گوسین	۳-۴-۲
۲۶		تابع بل تعمیم یافته (<i>Generalized bell</i>)	۴-۴-۲
۲۷		مشکل توابع گوسین	۵-۴-۲
۲۷		مزایای توابع گوسین	۶-۴-۲
۲۸		توسعه استوانه ای	۵-۲
۲۸		تصویر	۶-۲
۲۹		مکمل فازی	۷-۲
۲۹		مکمل <i>Sugeno</i>	۱-۷-۲
۲۹		مکمل <i>Yager</i>	۲-۷-۲
۳۰		عملگر اشتراک (<i>T-norm</i>)	۸-۲
۳۱		عملگر اجتماع (<i>S-norm</i>)	۹-۲
۳۲		<i>Associative Set</i>	۱۰-۲
۳۳		اصل توسعه	۱۱-۲
۳۴		رابطه فازی (<i>Fuzzy Relation</i>)	۱۲-۲
۳۴		ترکیب (<i>Max-Min (Composition)</i>)	۱۳-۲
۳۵		ترکیب <i>Max-Product</i>	۱۴-۲
۳۶		متغیرهای زبانی	۱۵-۲
۳۷		قوانین اگر-آنگاه فازی	۱۶-۲
۳۸		استنتاج فازی	۱۷-۲
۳۹		یک قانون با یک مقدم	۱-۱۷-۲
۳۹		یک قانون با چند مقدم	۲-۱۷-۲
۳۹		چند قانون با چند مقدم	۳-۱۷-۲
۴۱		متغیرهای دیگر	۱۸-۲
۴۱		روشهای غیر فازی سازی	۱۹-۲
۴۲		مشکلات روش مرکز ثقل	۱-۱۹-۲

۴۳	 Bisector of area	۲-۱۹-۲
۴۳	 نکات کلیدی در انتخاب غیر فازی سازها	۳-۱۹-۲
۴۳	 سیستم استنتاج فازی (Fuzzy inference system)	۲۰-۲
۴۴	 بخش های مختلف سیستم استنتاج فازی	۱-۲۰-۲
۴۴	 انواع قوانین اگر-آنگاه فازی	۲۱-۲
۴۵	 نحوه تنظیم قوانین	۲۲-۲
۴۶	 تقسیم بندی فضای ورودی (Input Space Partitioning)	۲۳-۲
۴۷	 مدل سازی و کنترل فازی (Fuzzy modeling control)	۲۴-۲
۴۸	۳. شناسایی سیستم ها	
۵۰	 روش های مبتنی بر مشتق	۱-۳
۵۰	 روش حداقل مربعات	۱-۱-۳
۵۴	 روش مبتنی بر گرادیان	۲-۱-۳
۵۵	 روش نیوتن	۳-۱-۳
۵۵	 روشهای بهینه سازی که مبتنی بر مشتق نیستند	۲-۳
۵۶	 جستجوی تصادفی	۱-۲-۳
۵۸	 روش سیمپلکس	۲-۲-۳
۶۰	 الگوریتم ژنتیک	۳-۲-۳
۶۰	 لغات کلیدی	۱-۳-۲-۳
۶۲	 تبرید تدریجی (Simulated Annealing)	۴-۲-۳
۶۴	 دیدگاه کلی	۳-۳
۶۵	۴. الگوریتم ژنتیک و برنامه سازی ژنتیک	
۶۵	 الگوریتم های ژنتیک	۱-۴
۶۵	 چرا از GA استفاده می کنیم؟	۱-۱-۴
۶۷	 فرایند طراحی الگوریتم ژنتیک	۲-۱-۴
۶۸	 مقداردهی اولیه	۳-۱-۴
۶۸	 عملگر انتخاب	۴-۱-۴
۷۰	 عملگرهای ژنتیک	۵-۱-۴
۷۲	 نظریه آجرهای سازنده	۶-۱-۴
۷۳	 تئوری بنیادی الگوریتم ژنتیک	۷-۱-۴
۷۴	 نمایش دوگانه (Dual Representation)	۸-۱-۴
۷۵	 انواع مختلف الگوریتم های ژنتیک	۹-۱-۴
۷۸	 مثالی از کاربرد Fuzzy GA	۱۰-۱-۴
۸۰	 برنامه سازی ژنتیک (Genetic Programming)	۲-۴
۸۶	 طراحی روبات	۳-۴
۹۰	 بهینه سازی دانش فازی	۴-۴

۵. شبکه عصبی

۹۲	۱-۵	ویژگیهای شبکه عصبی
۹۳	۲-۵	کاربردهای شبکه عصبی
۹۷	۳-۵	مبانی شبکه های عصبی
۱۰۱	۴-۵	تاریخچه
۱۰۱	۵-۵	شبکه عصبی تک لایه
۱۰۲	۶-۵	شبکه عصبی دو لایه
۱۰۳	۷-۵	تعریف دینامیک بودن در سیستم های کنترلی
۱۰۵	۸-۵	یادگیری شبکه عصبی
۱۰۵	۱-۸-۵	یادگیری نظارتی (<i>supervised</i>)
۱۰۷	۲-۸-۵	یادگیری بدون ناظر (<i>unsupervised</i>)
۱۰۷	۹-۵	ارائه یک مثال کاربردی از شبکه عصبی به عنوان یک طبقه بندی کننده
۱۱۱	۱۰-۵	الگوریتم آموزش
۱۱۱	۱۱-۵	ترکیب فازی و شبکه عصبی
۱۱۶	۱۲-۵	<i>ANFIS</i>
۱۲۵		



به نام ایزد دانا

مجموعه جزواتی که امروز تقدیم شما می گردد، حاصل تدریس درس سافت کامپوتینگ از پاییز ۷۹ تا به اکنون در دانشگاه فردوسی مشهد می باشد. به همراه این جزوه، مجموعه ای از اسلایدها و فایل های صوتی درس نیز بصورت **CD** تهیه شده است. در اینجا، بر خود لازم می دانم که از اساتید گرانقدری که مرا با مفاهیم ارزشمند رایانش نرم آشنا نمودند و در شکل گرفتن اولیه این درس نیز نقش داشتند مخصوصاً پروفسور محمد جمشیدی، استاد راهنمای دکترای اینجانب و رئیس مرکز پژوهشی سیستم های خودکار ناسا در دانشگاه نیو مکزیکو آمریکا که مرا با مفاهیم مختلف پیچیدگی در سیستم ها و الگوریتم های ترکیبی آشنا نمودند، پروفسور **Timothy Ross** از گروه عمران دانشگاه نیومکزیکو که با تدریس توانای خویش مرا علاقمند به منطق فازی و کنترل فازی نمودند، پروفسور **Andre Titli** از مرکز پژوهشی **LAAS** در **Toulouse** فرانسه، پروفسور **Young Tae Kim** از کره، پروفسور زیلوچیان، از دانشگاه **Florida Atlantic University** و در نهایت پروفسور لطفی زاده از دانشگاه **UC Berkeley** که مرا با مفاهیم عمیق منطق فازی و برخورد مناسب با عدم قطعیت آشنا نمودند سپاسگذاری کنم.

همچنین تکامل این درس بدون مشارکت دانشجویان علاقمندی که مرا در هر ترم در راستای هر چه جذاب تر و پربارتر نمودن آن یاری نمودند امکان پذیر نبوده است. من از کلیه آن دانشجویان عزیز سپاسگذارم. قطعاً چهره های علاقمند و شاداب آن جوانان با استعداد که با سوالهای خویش کلاس را پر بار تر نمودند، هدیه بزرگی برای اینجانب بوده است. شاید نقطه اوج این مشارکت در ترم پاییز ۸۶ پس از بازگشت از فرصت مطالعاتی بود که با کمک چهار دانشجوی علاقمند، خانم ها نسرين اباذری طرقيه، سمیه چشمی، مریم حبیبی پور که در تبدیل فایل صوتی سخنرانی های کلاس به متن و گردآوری این جزوات و خانم نازنین سید حجازی، که در تهیه **CD** همراه آن مرا در تهیه مجموعه حاضر یاری نمودند. از این عزیزان نهایت سپاس را دارم.

با آرزوی توفیق الهی،

محمد رضا اکبرزاده توتونچی

دانشیار گروه های برق و کامپیوتر

دانشگاه فردوسی مشهد



۱. سیستم های هوشمند

۱-۱. چرا محاسبات نرم (soft computing)؟

Soft computing شامل اجزائی همچون الگوریتم های ژنتیک، منطق فازی و شبکه های عصبی می باشد که ممکن است در جاهای مختلف با آن برخورد کرده باشید. این اجزاء را در قالب دیگری هم با نام هوش محاسباتی^۱ می شنوید. این ها از یک دیدگاه خیلی به هم شبیه هستند.

هوش محاسباتی و *soft computing* هر دو از یک سری اجزاء تشکیل شده اند. به عبارت دیگر هر دو اینها منطق فازی، یادگیری تقویتی^۲، روش کلونی مورچه ها، الگوریتم های ژنتیک و شبکه عصبی را شامل می شوند. اما در *soft computing* تاکید خاصی در ترکیب این روش ها وجود دارد.

soft computing دو بعد را ایجاد می کند. یکی بعد نرمی است که نسبت به دنیای واقع وجود دارد یعنی نگرش *soft computing* به واقعیت یک نگرش نرم است؛ بدین معنی که به جای اینکه بگوئیم این چنین باید باشد واقعیت را آنچنان که هست سعی می کنیم تصویر کرده و با آن برخورد کنیم. این دیدگاه اول *soft computing* می باشد. اما *soft computing* دیدگاه دیگری را نیز در بر می گیرد و آن ترکیب بین این پارادیم^۳ ها است.

به عبارت دیگر در *soft computing* با ترکیب شبکه های عصبی و فازی یا الگوریتم ژنتیک و فازی یا الگوریتم ژنتیک و شبکه های عصبی می توان به چیزی دست یافت که اجزاء آنها نمی توانستند به آن برسند.

خیلی جالب است که شبکه های عصبی یک مدل کمی از مغز انسان است و در مقابل آن منطق فازی یک مدل کیفی از دانش انسانی است. پس به نحوی هم شبکه های عصبی و هم الگوریتم های ژنتیک و هم فازی و یادگیری تقویتی که ریشه آن روانشناسی است، از مغز انسان و تعامل آن با محیط نشات می گیرد. پس به نحوی تمام اجزاء *soft computing* از مغز انسان نشات می گیرند.

ما در هوش مصنوعی ادعا می کنیم که واقع گرا هستیم. بنابراین، کاربرد، نقش مهمی در هوش مصنوعی دارد. هوش مصنوعی در دو زاویه جای رشد و کار دارد.

مشکل سیستم های هوشمند یکی آن است که محققین به صورت شبیه سازی کار کرده و اغلب مقالاتی که در کنفرانس ها ارائه می شوند به صورت شبیه سازی می باشد، در حالی که پایه و اساس سیستم های هوشمند، اجرا و

¹ computational intelligence

² Reinforcement learning

³ paradigm

عملی کردن است. همچنین از دیدگاه تئوریک نیز کارهای کمی انجام می شود. بنابراین از دو زاویه کمبود وجود دارد. یکی زاویه اجرا و پیاده سازی عملی و یک زاویه تحلیل تئوریک با ابزارهای ریاضی که در هر کدام به اندازه ϵ بتوانید پیش روید گام بزرگی را در پیشبرد هوش مصنوعی برداشته اید.

۲-۱. مقدمه ای بر محاسبات نرم (Soft computing):

بحث *soft computing* از اواسط دهه ۹۰ توسط پروفسور زاده مطرح شد.

اجزاء *soft computing* همان اجزائی هستند که در هوش محاسباتی استفاده می شوند. *soft computing* و هوش محاسباتی، دو دیدگاه کاملاً متفاوت، به ابزارهایی است که در ظاهر کاملاً مشابه هم هستند.

از دیدگاه پروفسور زاده و پیروان ایشان، *soft computing* یک نگرش واقع بینانه به واقعیت است. *Soft computing* و هوش محاسباتی در بطن کار کاملاً با هم متفاوت هستند. در حقیقت، عقیده زاده این است که هوشمندی صفت یک چیزی است. مثلاً فرض کنید معقول است که بیان شود، آیا این صندلی هوشمند است یا خیر. یا این فرد یا کامپیوتر هوشمند هستند یا خیر. بنابراین می توانند صفت هوشمندی را دارا باشند یا نباشند. ولی *computation* اینگونه نیست. به خاطر این معقول نیست که بیان شود هوشمند است یا خیر. طرفداران زاده معتقدند اصلاً این دو کلمه در کنار هم منطقی نیست.

در هوش مصنوعی یکی از مسائل، محک برای هوشمندی است. طرفداران هوش محاسباتی از عبارت *CIQ* و طرفداران زاده که معتقدند هوشمندی صفت ماشین است از *MIQ* استفاده می نمایند.

در بسیاری از مقالات، *soft computing* و هوش محاسباتی جایگزین هم استفاده می شوند. در حالی که دو دیدگاه کاملاً متفاوت فلسفی به واقعیت دارند.

علت اصلی توجه به سیستم های هوشمند این است که به طور کل، دنیای واقعی بسیار بسیار توأم با عدم قطعیت بوده و ابهام در آن وجود دارد.

آلبرت انیشتن می گوید:

“As for as the laws of mathematics refer to reality, they are not certain, and so far as they are certain, they do not refer to reality.”

یعنی آنجایی که قوانین ریاضی به واقعیت اشاره می کنند این اشاره همراه با قطعیت نیست و آنجایی که قوانین ریاضی قاطع حرف می زنند این به واقعیت اشاره نکرده و از واقعیت خیلی دور است.

امروزه بیش از پیش این مساله اهمیت پیدا کرده است، زیرا ما امروزه می خواهیم سیستم هایی که ساخته دست بشر هستند از توانایی بیشتری برخوردار باشند. انتظار ما از سیستم هایی که می سازیم این است که در عین حالی که پیچیده تر هستند، راندمان بیشتری داشته باشند، دوستانه تر بوده و ارزان تر باشند. این در حقیقت یک پارادوکس را ایجاد می کند که چگونه می توان با پیچیدگی بیشتر به راندمان بالاتر، دوستانه تر و ارزان تر رسید؟

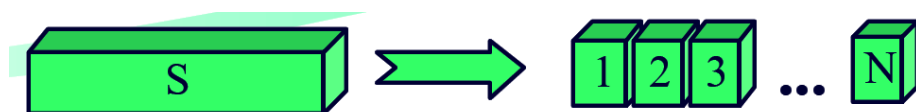
این آن چیزی است که ما در قالب سیستم های هوشمند به دنبال آن هستیم. به عبارت دیگر به جای اینکه سیستم های مصنوع بشر را به گونه ای بسازیم که رفتارشان از پیش تعیین شده باشند، می خواهیم به آنها قدرت یادگیری بدهیم (در حقیقت به آنها ماهیگیری یاد بدهیم) تا بتوانند خودشان با عدم قطعیت هایی که در محیط برایشان پیش می آید برخورد مناسب کنند، به جای اینکه تمام آن چیزی را که ممکن بود پیش بیاید برایشان برنامه ریزی کنیم.

مثلا اگر برای سیستمی ۱۰۰۰۰۰۰ حالت برایش مشخص شود، اگر حالت ۱۰۰۰۰۰۱ پیش آید باز کاری از سیستم بر نمی آمد.

پس علت اینکه ما می خواهیم سیستم ها را هوشمند کنیم دقیقا به این علت است که واقعیت آنگونه است که ما نمی توانیم تمام حالات را از پیش تعیین کنیم. بنابراین باید سیستم، هوشمند شود تا با عدم قطعیتها برخورد مناسب کرده و آن توانایی که از آن انتظار می رود را برآورده سازد.

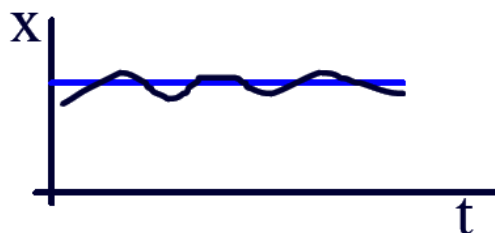
در روش های کلاسیک فرضیاتی در نظر گرفته می شود:

۱. فرض اول: در برخورد با مسائل پیچیده یک مساله بزرگ به مسائل کوچک شکسته می شود. به این امید که اگر نمی توان با سیستم بزرگ یک برخورد مناسب داشت، سیستم های کوچکتر یا زیر سیستم ها^۴ را می توان حل نمود.

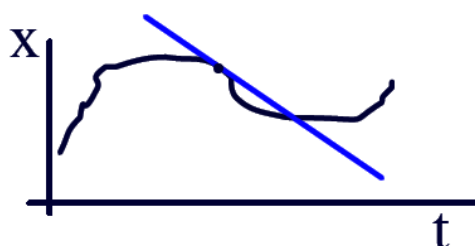


۲. فرض دوم: کار دیگری که انجام می شود این است که فرض می شود پارامترهایی که در سیستم به اندازه کافی آهسته تغییر می کنند، اصلا تغییر نمی کنند.

⁴ sub systems



۳. فرض سوم: به خاطر سهولت، فرض می شود یک سیستم دینامیک^۵ که عموماً غیرخطی است با یک ابزار خطی جایگزین می شود. تبدیل لاپلاس مربوط به سیستم های خطی است در حالی که یک سیستم واقعی خطی در طبیعت وجود ندارد.



۴. فرض چهارم: فرض می شود سیستم کنترل پذیر است. با تمام فرض هایی که تاکنون انجام شد اگر کنترل پذیر نباشد نمی توانی کاری روی آن انجام داد.

۵. فرض پنجم: فرض می شود رویت پذیر است.

۶. فرض ششم: فرض می شود تاخیر زمانی وجود خارجی ندارد. در حالی که همه سیستم های واقعی بعد فیزیکی دارند و وقتی که بعد فیزیکی وجود داشته باشد، تاخیر زمانی وجود دارد. یعنی زمانی طول می کشد بین وقتی که شما نیرو را به سیستم اعمال می کنید تا زمانی که پاسخ را مشاهده می نمایید.

تمام این فرض ها در نظر گرفته شدند تا بتوان با ابزارهای ریاضی ساده ای که وجود دارند مساله را حل کرد. غافل از اینکه تمام این پیش فرض ها ما را از واقعیت مساله دور کرده است. در حقیقت ما مساله ای را حل می کنیم که وجود خارجی ندارد.

این فرضیات باعث از دست دادن راندمان، از دست دادن دقت و افزایش هزینه می شود.

حال چه باید کرد؟

اگر مساله بزرگ و پیچیده ای وجود دارد که نمی توان آن را به مسائل کوچک تر شکسته و حلش کرد پس چه

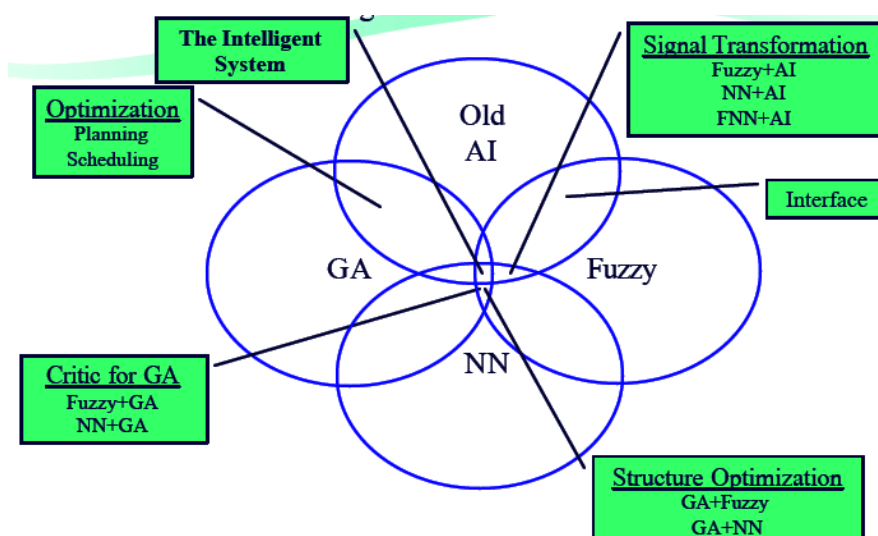
⁵ Dynamic

چیزی غیر از ابزار ریاضی در اختیار ما است ؟

آنچه که در اختیار ما بوده و پروفسور زاده به آن توجه می کند این است که ما بخش عمده ای از دانشمان به صورت دانش انسانی است که تا آن موقع عموماً از آن صرفنظر می شده است. دانش انسانی در کارخانه ها و در اختیار عملگر بوده نه طراح. زاده این نیاز را احساس کرد و این وجود دانش انسانی را حس کرده و می گوید چگونه باید از این استفاده کرد.

زاده مهندس کنترل بوده و کنترل سیستم های فازی از احساس نیاز یک مهندس در حل مسئله اش نشأت می گیرد و می بیند که راهکاری برای حل مساله اش وجود دارد که در قالب سیستم های کلاسیک نمی گنجد.

بخش عمده ای از اطلاعات ما به صورت دانش انسانی موجود است و روش های کنونی نمی توانند از آن استفاده کنند. بنابراین به جای اینکه صرفاً راهکارمان را بر اساس مدل سیستم پایه ریزی کنیم می توانیم از دانش انسانی بهره برداری کنیم و مدل را بر اساس آن بکار ببریم.



۳-۱. تعریف پروفسور زاده از محاسبات نرم:

پروفسور زاده در سال ۱۹۹۲ در تعریف *soft computing* می گوید: *soft computing* یک پدیده نوظهور است که در اثر تعامل بین پارادایم های مختلف پدیدار شده است. بدین معنی که از قبل تعیین شده نیست. *soft computing* سعی می کند به توانایی دست یابد که مانند مغز انسان تفکر کند، محاسبه کرده و بتواند یاد بگیرد و استنتاج کند؛ در محیطی که توام با عدم قطعیت و عدم دقت است.

اجزاء مختلف *soft computing* با هم یک *synergism* دارند.

بنابراین با ترکیب به عنوان مثال الگوریتم ژنتیک و شبکه عصبی یا شبکه عصبی و فازی یا همه این پارادایم ها در یک قالب سودمند، امیدواریم به توانایی برسیم که در تک تک آنها وجود ندارد. بنابراین هدف *soft computing* این است که به ساختاری توانمندتر از هر یک از اعضایش دست یابد.

تک تک اجزاء *soft computing* به گونه ای از طبیعت الهام گرفته اند. مثلاً شبکه عصبی مدل کمی از مغز انسان است. منطق فازی یک مدل کیفی یا یک پایگاه دانش^۶ از مغز انسان است.

پایه گذار الگوریتم های تکاملی، داروین می باشد و پس از وی جان هلند، الگوریتم ژنتیک را پایه گذاری نموده است. جالب است بدانید که یک روانشناس نشان داد که می توانیم از الگوریتم های ژنتیک در مدلسازی تفکر انسانی هم بهره برداری کرد.

اما غیر از این سه پارادایم معروف، پارادایم های دیگری هم وجود دارند. مثل الهام گرفتن از سیستم ایمنی بدن انسان یا کلاً الهام گرفتن از زندگی به صورت *artificial life*، به صورتی که شما هم تکامل و هم یادگیری را در قالب مصنوعی بتوانید داشته باشید.

۴-۱. آینده محاسبات نرم از دیدگاه پروفیسور زاده:

The concept of Soft Computing (SC) made its debut in the early part of the past decade.

Today, soft computing is gaining rapidly growing recognition as an important tool in the conception, design, construction and utilization of information/ intelligent systems. Such systems play a pivotal role in the functioning of modern society.

What is soft computing? Soft computing is not a homogeneous body of concepts, techniques and methods. Rather, it is a consortium of methodologies centering in fuzzy logic (FL), neuro-computing (NC), evolutionary computing (EC), evolutionary programming (EP), probabilistic computing (PC), chaotic computing (CC) and parts of machine learning theory.

What is important is that these methodologies are for the most part complementary and synergistic, rather than competitive. It is this synergism that underlies the ability of soft computing to solve complex problems, which exceed the capability of individual constituents of soft computing when used in a stand-alone mode.

⁶ Knowledge base

Thus the basic premise of soft computing is that, in general, it is advantageous to use the constituents (methodologies of SC in combination).

A combination that today has the highest visibility is that of so called neuro-fuzzy systems. We are beginning to see systems, which are fuzzy-genetic, neuro-genetic, neuro-chaotic and neuro-fuzzy-genetic, among others.

In my view, in coming years such combined hybrid systems will be the rule rather than exception, especially in the realm of high MIQ (Machine IQ) systems.

In the partnership of the constituents of SC, each partner contributes a collection of concepts, techniques and tools.

More specifically, the main contribution of fuzzy logic is the methodology of computing with words and the computational theory of perception.

The principal contribution of neuro-computing are systems which have the capability to learn and identify from input-output data.

The main contribution of EC are methods of systematized random search and optimization.

The main contribution of EP is random optimization of symbolic instructions, e.g. a set of fuzzy rules.

And the principal contribution of probabilistic computing centers on management of uncertainty and decision analysis in an environment of imprecision and uncertainty.

Applications of soft computing in industry are growing in number, visibility and importance.

این دیدگاه زاده همچنان برقرار است و اگر بخواهیم فقط فازی را مثال بزنیم، در سال ۲۰۰۷، فازی بیشترین میزان ارجاع^۷ را در فلسفه داشته و این نشان می دهد که دیدگاه فازی بسیار پایه ای بوده است. چون فلسفه، مادر علوم می باشد.

در ظاهر، شبکه عصبی و فازی هیچ تناسبی با هم ندارند. این عدم تجانس است که توانایی *soft computing* را به ارمغان می آورد.

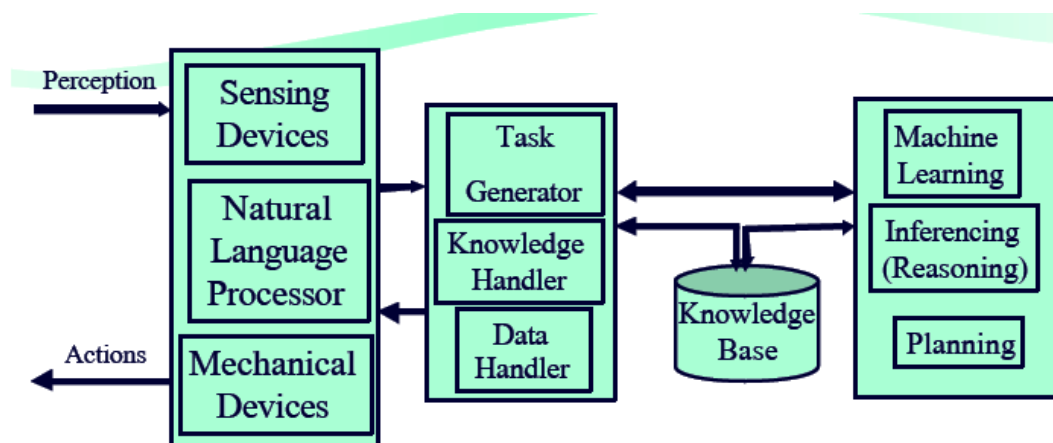
آنچه که مهم است این است که این پارادایم ها بیشتر مکمل هم هستند به جای اینکه رقیب هم باشند.

⁷ Citation

این مکمل هم بودن و *synergism* است که موجب حل مسائل پیچیده می شود؛ به گونه ای که با تک تک اجزاء نمی توان آن مساله را حل نمود. ترکیب های موفق که زاده به آن اشاره می کند *neuro- chaotic, neuro- genetic, fuzzy- genetic, neuro-fuzzy- genetic* هستند.

از دیدگاه زاده، این سیستم های ترکیبی هستند که به عنوان ابزار آینده مورد استفاده قرار خواهند گرفت. مخصوصاً سیستم هایی که قرار است از *high MIQ* بهره مند شوند.

۵-۱. سیستم های هوشمند چه هستند؟



اگر یک سیستم هوشمند را در نظر بگیرید، این سیستم، با محیط بیرون ارتباط برقرار کرده و از طریق تجهیزات مکانیکی^۸ با آن تعامل دارد. در وسط، پردازشگر زبان طبیعی^۹ است که برای محاسبه بکار می رود و بعداً در سطح بالاتر، بخشی است که وظایف را تعیین می کند. بخش بالاتری از هوشمندی، سطحی است که دارای قابلیت استنتاج و یادگیری می باشد.

بنابراین هوشمندی سطوح مختلفی دارد و سؤالی که اینجا مطرح می شود این است که به عنوان مثال الگوریتم ژنتیک را در کجا می توان بکار برد؟ الگوریتم ژنتیک در استنتاج استفاده نمی شود، بلکه موتور استنتاج فازی در استنتاج مورد استفاده قرار می گیرد.

در یادگیری ماشین^{۱۰}، از الگوریتم ژنتیک استفاده می شود. پس قسمت های مختلف سیستم های هوشمند می تواند قسمت های مختلف *soft computing* را در بر بگیرد. اگر بتوان بطور موثری اینها را کنار هم قرار داد، می توان به یک سیستم هوشمندتر رسید.

^۸ Mechanical device

^۹ Natural language processor

^{۱۰} Machine learning

۶-۱. چند کاربرد:

۱-۶-۱. *Fuzzy- Neural Net*:

وقتی می گوئیم *Fuzzy Neural* یعنی یک سیستم عصبی که *Fuzzify* شده است. یعنی لزوماً دیگر ورودی هایش عدد نیست، بلکه می تواند مجموعه فازی باشد و این می تواند با یک مجموعه فازی که عدم قطعیت دارد باز هم محاسباتش را ادامه بدهد. یا سیستم *Neuro Fuzzy* که سیستم های فازی هستند که *Neurofy* شده اند. مزیت سیستم شبکه عصبی این است که آموزش می بیند. حال اگر یک مجموعه فازی در قالب شبکه عصبی آورده شود، می توان از ابزارهایی که در شبکه عصبی وجود دارد استفاده نمود. یکی از ایرادات سیستم های فازی دانش محور بودنش است. یعنی اگر دانش وجود داشته باشد بسیار خوب است، اما اگر دانشی وجود نداشته باشد در این صورت مثلاً باید با شبکه های عصبی ترکیب شود.

۲-۶-۱. *Fuzzy- PID systems*:

در کنترل، هدف این است که رفتار سیستم را با تنظیم ورودی ها طوری تغییر بدهیم که به نحوی بشود که مطلوب ماست. خطا را محاسبه کرده و می گوئیم ورودی به همین نسبت، طوری تنظیم شود که خطا به صفر نزدیک شود. گاهی اوقات *proportional feedback* کافی نیست. نیاز است که از خطا، انتگرال هم بگیریم که به آن *integral feedback* گفته می شود (*PI*). گاهی اوقات انتگرال هم جواب نمی دهد باید از خطا مشتق هم گرفته شود (*PID*).

گاهی اوقات لازم است که *KI, KP, KD* بگونه ای تنظیم شوند که بتوانند با مجموعه قوانین فازی این کار را انجام دهند.

۳-۶-۱. *Fuzzy & Genetic systems*:

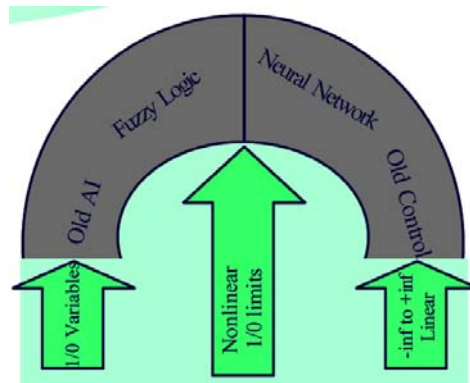
یکی از مشکلات فازی این است که مبتنی بر دانش است که اگر این دانش وجود نداشته باشد هیچ کاری نمی تواند انجام دهد بنابراین با یک روش دیگر باید ترکیب شود.

اجزاء مختلف *soft computing* اجزاء از هم گسسته ای نیستند. در هوش مصنوعی کلاسیک، متغیرها یا یک و یا صفر هستند. منطق فازی را می توان توسعه ای از هوش مصنوعی کلاسیک دانست که در آن متغیرها بین ۰ و ۱ تغییر می کنند.

۷-۱. مقایسه فازی، شبکه عصبی و هوش مصنوعی:

	Math Model	Learning Data	Operator Knowledge	Real Time	Knowledge Representation	Nonlinearity	Optimization	Areas of Strength
Control Theory	O	X	X	O	X	X	X	Rigorous Mathematical Results
Neural Networks	Δ	O	X	O	X	O	◐	Learning and Adaptation
Fuzzy	◐	X	O	O	◐	O	X	Knowledge Representation
AI	Δ	X	O	X	O	Δ	X	Systematic Random Search
GA	X	O	X	Δ	X	O	O	Symbolic Manipulation

O Very Good ◐ Good Δ Weak X Very Poor

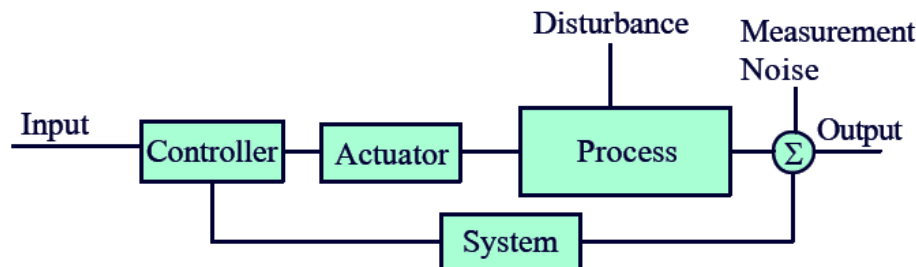


در مقایسه هوش مصنوعی با فازی می توان گفت، هر قاعده اگر- آنگاه فازی شامل بی نهایت قانون اگر- آنگاه هوش مصنوعی است. یعنی گاهی دو قانون اگر- آنگاه فازی می تواند به اندازه میلیون ها قانون اگر- آنگاه هوش مصنوعی بازدهی داشته باشد.

۸-۱. سیستم هوشمند و کنترل هوشمند:

حال می خواهیم از یک دیدگاه کلی به سیستم های هوشمند نگاه کنیم. کلاً ما در سیستم های هوشمند یک جعبه سیاه^{۱۱} داریم که این جعبه سیاه رفتاری را دارد. یک سری ورودی به این جعبه سیاه اعمال می شود و یک سری خروجی از آن دریافت و مشاهده می شود. اغلب آنچه که درون جعبه سیاه وجود دارد را ما یا نمی دانیم یا اینکه اگر می دانیم توام با ابهام است.

^{۱۱} Black box



علاوه بر آن، آنچه که مشاهده می شود، خروجی واقعی سیستم نیست. هنگام مشاهدات، به سیستم نویز اضافه می- شود. به عنوان نمونه می توان به ساخت نمونه آزمایشگاهی دستگاه تقطیر آب اشاره نمود.

یک سری پارامترهای آشوب^{۱۲} وجود دارند که رفتار سیستم را پیچیده می کنند. علاوه بر آن یک سری نویز هنگام مشاهده بوجود می آید. هدف این است که در قسمتی از کار، سیستم شناخته شود و در قسمتی از کار رفتارش با تنظیم یک سری کنترلر آنگونه تنظیم شود که مطلوب است.

۹-۱. مدلسازی به روش کلاسیک (Modeling in conventional model):

۱-۹-۱. گام اول: جمع آوری اطلاعات

در مرحله اول باید یک سری اطلاعات جمع آوری شود.

- گاهی اوقات می توان یک سری داده در ورودی به سیستم اعمال کرده و بر اساس آن ها خروجی را مشاهده نمود. سعی می شود از این اطلاعات به عنوان پایگاه اولیه اطلاعات استفاده شود.
- برای شناسایی رفتار سیستم، می توان از یکی از تکنیک های شناسایی موجود مانند *least square* یا از *Bode plot approximation* استفاده نمود.

۲-۹-۱. گام دوم: شناسایی ساختار

در این مرحله، شناسایی ساختار مهم است. مثلاً درون سیستم چه اجزایی وجود دارد و این اجزاء چگونه با هم ارتباط دارند.

¹² Disturbance

تا اینجا به نظر نمی آید که روش های هوشمند و کلاسیک چندان با هم تفاوتی داشته باشند.

۱-۱۰. کنترل کلاسیک در مقابل کنترل هوشمند:

روش کلاسیک کنترل، یک روش تجویزی^{۱۳} است. یعنی وقتی به پزشک مراجعه می کنید و به وی علائم بیماری خود را می گوئید، پزشک برای شما دارو تجویز می کند. سیستم های تجویزی، مبتنی بر مدلی از سیستم هستند. یعنی بر اساس مدلی از سیستم کار می کنند.

پس در روش کلاسیک، مدلی از سیستم که مبتنی بر *Relational Representation (ODE)* است بدست می آید و بر اساس این مدل از سیستم، تجویز می شود که مثلاً گشتاور ربات چقدر باید باشد. هدف کلی از ساختار توضیح داده شده این است که مساله ای را حل کند که یک سری اهداف دارد. این اهداف از پیش تعیین شده و ثابت هستند.

در مقابل، کنترل هوشمند، دیدگاه کلی اش یک دیدگاه توصیفی^{۱۴} است. به عبارت دیگر مبتنی بر دانش بوده اما این دانش لزوماً مبتنی بر روابط ریاضی نیست بلکه مبتنی بر یک سری قوانین و دانش است که ممکن است این دانش به صورت انسانی مطرح شود.

ابزاری که در کنترل هوشمند در اختیار قرار می گیرد با ابزاری که در کنترل کلاسیک وجود دارد متفاوت است.

در کنترل هوشمند، هوش مصنوعی، منطق فازی، شبکه عصبی و الگوریتم ژنتیک وجود دارد. اینها همه یک سرنخ مشترک دارند و آن این است که همه آنها بر این دیدگاه هستند که دنیای واقعی پیچیده تر از آن است که بتوان توسط یک سری روابط ریاضی آن را تعریف کرد. طبعاً چنین ساختاری که سعی می کند واقع بینانه باشد و به جای تجویز، واقعیت را توصیف کند، هدفش کار روی سیستم های غیرخطی است که مرتبه^{۱۵} بالایی دارند، مرتبه آنها ساده نشده است و متغیر با زمان بوده و تصادفی^{۱۶} هستند و تمامی آن پیچیدگی هایی را که به خاطر ساده سازی و رسیدن به یک ساختار ریاضی فرض شدند دیگر فرض نمی شوند.

چنین ساختاری، دو خصوصیت خاص دارد و آن این است که بتواند تصمیم گرفته و یاد بگیرد.

۱-۱۱. پیشرفت تاریخی (Historical developments):

در کنترل هوشمند، به عدم قطعیت خیلی زیاد اشاره می شود. اولین نمونه صنعتی توربین بخار، کار جیمز وات بوده

¹³ *prescriptive*

¹⁴ *descriptive*

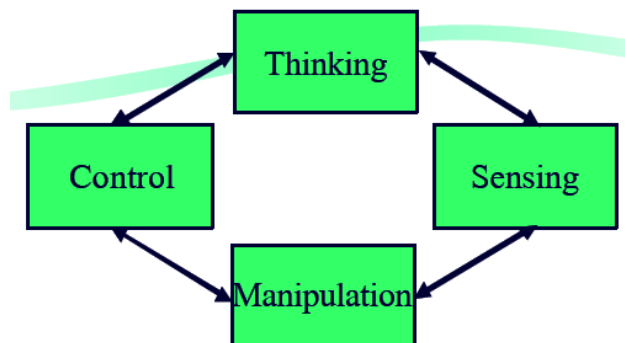
¹⁵ *Order*

¹⁶ *Stochastic*

است. او در آن زمان از بخار برای حرکت لوکوموتیو استفاده کرده و فقط مشکلش این بوده است که حرارت، زمانی که کارگر، زغال می ریخته پس از مدتی زیاد می شده و بنابراین تا مدتی لوکوموتیو حرکت نمی کرده و بعد از آن خیلی سریع حرکت می کرده و کارگر دیگر زغال نمی ریخته است. در حقیقت کنترل انسان محور در آنجا پاسخگو نبوده است. بنابراین وی مفهوم فیدبک را در این زمان مطرح کرد. فیدبک از اولین نمونه های صنعتی برخورد با عدم قطعیت بوده است. برای ایجاد حرکت ثابت، بالای توربین یک چرخ گردان گذاشته که به محور حرارتی توربین متصل بوده و هر چقدر سرعت زیاد می شده، چرخ گردش زیادتر می چرخیده است. به آن دو بازو متصل کرده که هر چقدر گردش بیشتر شود آنها بالاتر می رفتند و آنها را به دریچه گریز بخار وصل کرده است.

پس از وی، دیدگاه هایی مطرح شد که چگونه با سیستم هایی که یک ورودی خروجی دارند برخورد شود. بعداً سیستم ها پیچیده شدند (چند ورودی و چند خروجی) و بعد از آن مشخص شده که اینها کافی نیست و باید جعبه سیاه را شکافته شود. در این زمان فضای حالت^{۱۷} معرفی شده است. فضای حالت رفتار اندرون سیستم را تعریف می-کند.

حتی در کنترل مدرن هم پیش فرض این است که یک مدل دقیق را می توان به راحتی بدست آورد و می توان یک معیار عملکرد دقیق را به راحتی فرمول بندی کرد. در مقابل در سیستم های هوشمند همیشه الهام از طبیعت بوده است. در طبیعت هم به صورت خیلی ساده و طبیعی از فیدبک استفاده می شود.



حالا فرض می کنیم مساله ای داریم و می خواهیم آن را حل کنیم. یک جعبه سیاه داریم که می خواهیم برایش یک کنترلر طراحی کنیم.

۱-۱۲. فرموله سازی مساله (Problem formulation):

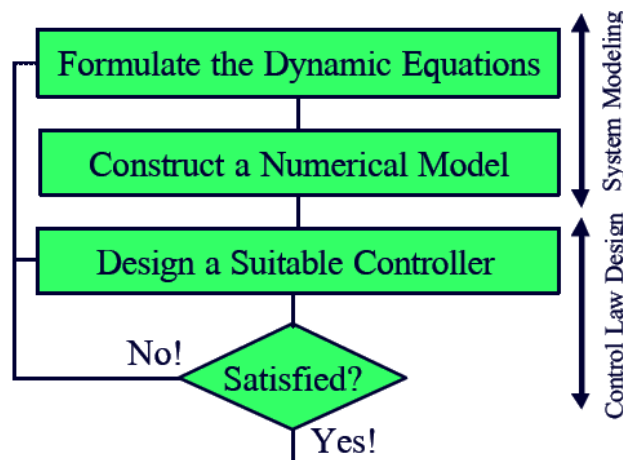
¹⁷ state space

۱. یک مدل دینامیک^{۱۸} بوجود می آوریم.

۲. برای آن یک مدل عددی بوجود می آوریم. گاهی اوقات مدل دینامیک امکانپذیر نیست و گاهی می خواهید هر دو را در کنار هم داشته باشید؛ زیرا آنچه که دینامیک می دهد برای دقتی که لازم است، کافی نبوده و در کنار آن یک مدل عددی نیز لازم است. این بخش مدلسازی سیستم^{۱۹} می باشد.

۳. بعد از اینکه مدلی از سیستم بوجود آمد برایش کنترلر تعریف می شود.

۴. این کار را تا زمانی انجام می دهیم تا راضی شویم وگرنه برمی گردیم و کنترلر یا مدل دینامیک را بازبینی می کنیم.



۱-۱۳. مشکلات سیستم های کنترل کلاسیک:

چنین ساختاری مشکلات زیر را دارد:

۱. *Sensor data overload*: یعنی تعداد اطلاعاتی که برای شما می رسد تا بتوانید به طور معقولی

تصمیم گیری کنید بسیار زیاد می شود.

۲. اگر از بعضی از سنسورها صرفنظر کنیم، سیستم *Robustness* خود را از دست می دهد تا بتواند به صورت موثری عمل کند.

۳. چنین ساختاری نمی تواند از *heuristic* استفاده کند. در اینجا *heuristic* به معنای دانش از پیش می باشد.

¹⁸ Dynamic Model

¹⁹ System Modeling

۴. چنین ساختاری چون محاسبات زیادی را می‌طلبد نمی‌تواند برای کنترل سریع^{۲۰} استفاده شود.

۵. اینکه سنسورها کجا باشند هم مساله ناشناخته‌ای است. معمولاً از پیش، فرض می‌شود که سنسورها در کجا قرار گیرند.

۱۴-۱. سیستم هوشمند:

روش‌های کلاسیک هیچ ابزاری ندارند که سیستم را برای برخورد این‌گونه با مساله مجهز کنند. در مقابل در سیستم هوشمند، هدف این است که تخمینی زده شود که وابسته به یک مدل نباشد و بدون اینکه لزوماً رابطه ریاضی بین ورودی-خروجی را داشته باشیم بتوان به آن تابع بهینه رسید. در سیستم هوشمند، خاصیت تعمیم^{۲۱} وجود داشته و همچنین یک درجه خلاقیت در آن وجود دارد و در نهایت می‌تواند یاد گرفته و خود را تطبیق دهد. یادگیری و تطبیق دو خاصیتی هستند که در کنار هم و توأم با هم هستند. بنابراین ما در سیستم هوشمند به جای اینکه از پیش تعریف کنیم که رفتار یک سیستم اینگونه باید باشد سعی می‌کنیم تا حد امکان کنترلی که می‌سازیم را آنگونه هوشمند کنیم تا خودش بتواند کشف کند که رفتار سیستم چگونه باید باشد.



²⁰ High speed control

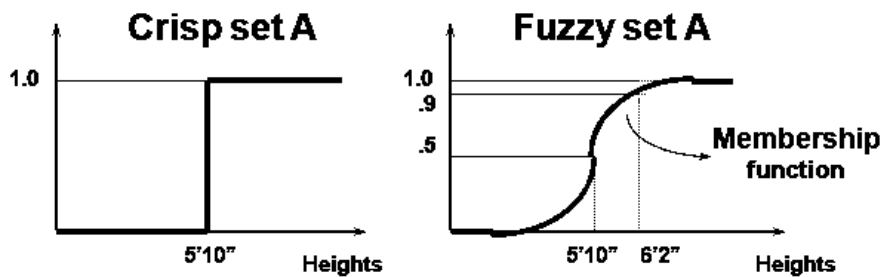
²¹ Generalization

۲. منطق فازی

۱-۲. مجموعه های فازی:

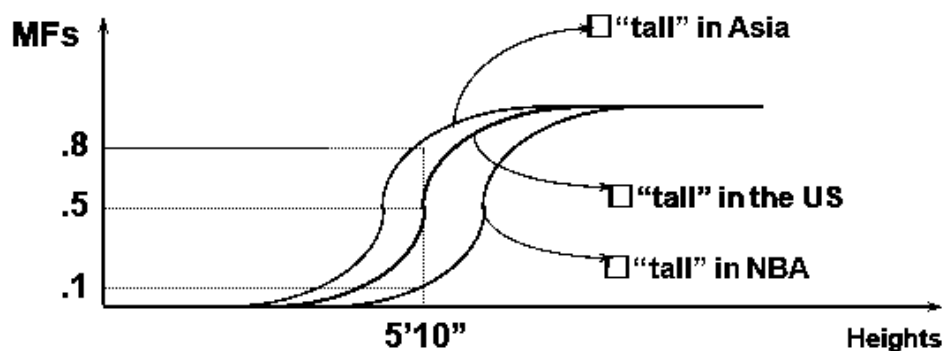
یک مجموعه فازی می تواند مجموعه افراد قد بلند باشد.

$A = \text{Set of tall people}$



مثلا افرادی که قد آنها بیشتر از 5'10 است قد بلند هستند. یکی از خصوصیات دیدگاه فازی این است که توابع عضویت *subjective* هستند و *objective* نیستند و این یکی از ایراداتی بود که در ابتدا بر مجموعه های فازی وارد می شد. در حالی که خود *subjective* بودن نقطه مثبت مجموعه های فازی محسوب می شود، چون این توانایی را دارند که از دانش انسانی بهره برداری کنند و دانش انسانی *subjective* است.

یکی از خواص دیگر این است که تعریف مجموعه های فازی ممکن است در جاهای مختلف با هم فرق کند. برای همین *subjective* است. مثلا مجموعه افراد قد بلند در آمریکا و آسیا، متفاوت است.



یکی دیگر از ایراداتی که در ابتدا مطرح بود این بود که چون گفته می شود فازی برای عدم قطعیت ها بیان شده و توابع عضویتی بین صفر و یک دارد، بنابراین تصور بر این بود که فازی بیان دیگری از احتمالات است که کاملاً اشتباه است.

به عنوان مثال: فرض کنید که در زندانی اسیر هستید و دو لیوان برای شما آورده شده است که حتماً یکی از آنها را باید بنوشید. لیوان سمت راست لیوانی است که به میزان ۰/۱ سمیت دارد و لیوان سمت چپ به احتمال ۰/۱ سم است. شما کدامیک را می نوشید؟

به میزان ۰/۱ سمیت دارد یعنی ۰/۱ آن سم است. اما به احتمال ۰/۱ یعنی یا سم است و یا خوب و سالم است.

و این تفاوت فازی و احتمال است.

در فازی هیچ ابهامی وجود ندارد و اتفاقاً فازی برای رفع ابهام استفاده می شود. مجموعه فازی را با استفاده از *universe* و تابع عضویت می توان به صورت یکتا تعریف نمود. گاهی اوقات، *universe* گسسته^{۲۲} است. مثل شهرهای مطلوب برای زندگی

$$X = \{SF, Boston, LA\} \text{ (discrete and nonordered)}$$

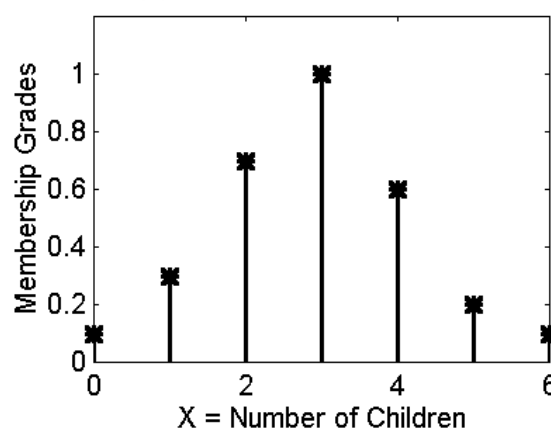
$$C = \{(SF, 0.9), (Boston, 0.8), (LA, 0.6)\}$$

در این مثال، *ordered universe* نیست. به عبارت دیگر هیچ رابطه ای بین *LA*, *SF*, *BS* وجود ندارد (ترتیب معنا ندارد).

اما در مثالی دیگر، تعداد مطلوب فرزندان: هم گسسته و هم *ordered* است. (ترتیب مهم است)

$$X = \{0, 1, 2, 3, 4, 5, 6\} \text{ (discrete universe)}$$

$$A = \{(0, .1), (1, .3), (2, .7), (3, 1), (4, .6), (5, .2), (6, .1)\}$$



²² discrete

همواره علاقمندیم که تابع عضویت نرمال باشد، یعنی حداقل به ازای یک عضو مقداری برابر با ۱ داشته باشد.

گاهی نیز ممکن است *universe* پیوسته باشد.

نحوه نمایش در *universe* گسسته:

$$A = \sum_{x_i \in X} \mu_A(x_i) / x_i$$

نحوه نمایش در *universe* پیوسته:

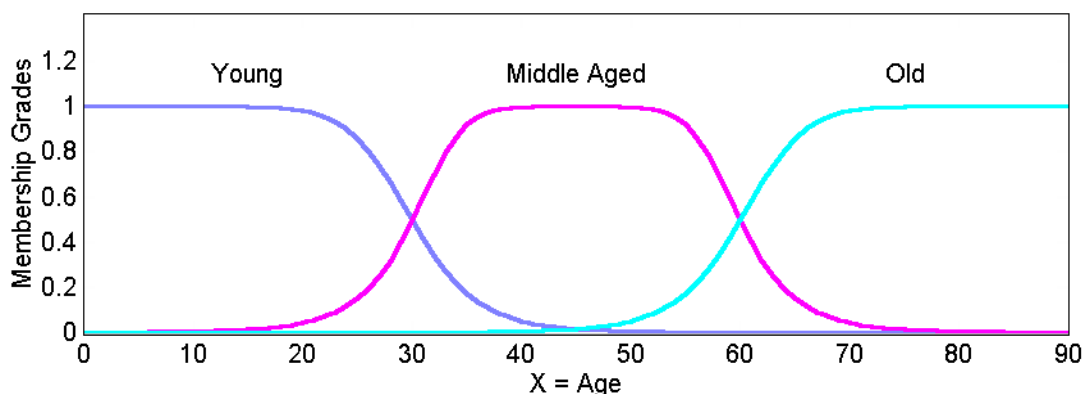
$$A = \int_X \mu_A(x) / x$$

۲-۲. تقسیم بندی فازی (Fuzzy Partitioning):

در حالت *Crisp partitioning* وقتی بخواهیم *universe* را به چند زیر مجموعه تقسیم کنیم، مثلاً می‌گوئیم

افراد بین ۰ تا ۳۰ سال جوان، بین ۳۰ تا ۶۰ سال، میانسال و ۶۰ سال به بالا مسن هستند. ولی در *Fuzzy*

partitioning مرزها به صورت تدریجی^{۲۳} هستند.



۳-۲. واژه نگاری توابع عضویت (MF Terminology):

۳-۲-۱. هسته^{۲۴}: مجموعه ای از x ها که کاملاً به آن مجموعه تعلق دارد.

۳-۲-۲. پشتیبان^{۲۵}: بازه ای از *universe* که به میزانی بیشتر از صفر به A تعلق داشته باشند.

۳-۲-۳. نقطه تبدیل^{۲۶}: آنهایی که هم به A تعلق دارند و هم به A تعلق ندارد. یعنی به همان میزان که به A

تعلق دارد به همان میزان هم به A تعلق ندارد.

²³ gradual

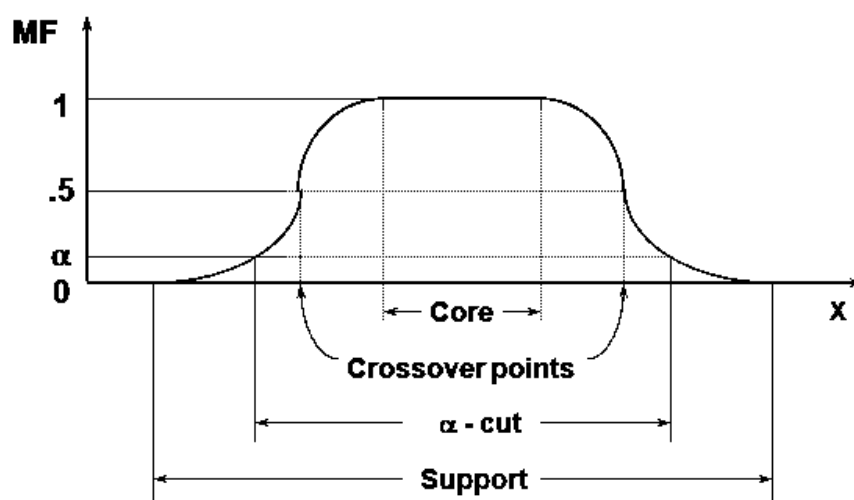
²⁴ core

²⁵ support

²⁶ Crossover point

۲-۳-۴. α -cut :

مجموعه ای است از x ها که به میزان α یا بیشتر به A تعلق داشته باشد. (یک مجموعه *crisp* است)



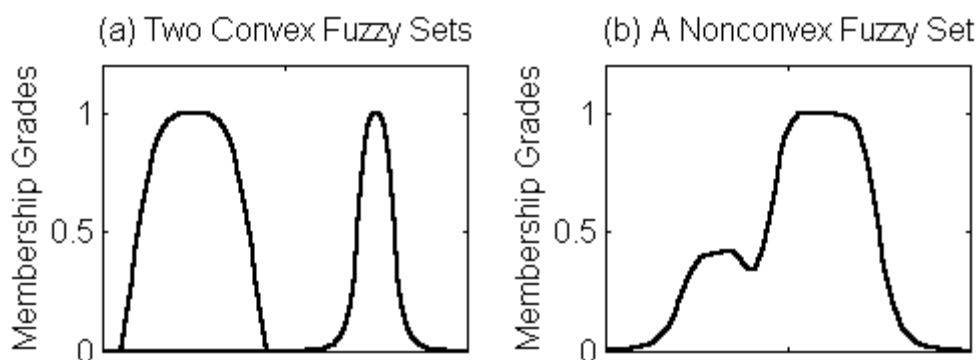
۲-۳-۵. تحدب^{۲۷} :

ساده ترین تعریف این است که مجموعه فازی محدب است، اگر و فقط اگر تمامی α -cut set های آن محدب باشند. چون α -cut set ها *crisp* هستند، بنابراین این تعریف بسیار شفاف است.

تعریف دیگر محدب بودن:

مجموعه فازی A محدب است اگر بازای هر λ در بازه $[0, 1]$ داشته باشیم:

$$\mu_A(\lambda x_1 + (1 - \lambda)x_2) \geq \min(\mu_A(x_1), \mu_A(x_2))$$



²⁷ convexity

۲-۳-۶. زیرمجموعه بودن :

اگر به ازای جمیع x هایی که به A تعلق دارد به مقدار بیشتری به B تعلق داشته باشد، آنگاه می گوئیم A زیرمجموعه B است.

$$A \subseteq B \quad \mu_A \leq \mu_B$$

۲-۳-۷. مکمل^{۲۸} :

آن عضوی که به میزان M_A به A تعلق دارند به میزان $I-M_A$ به A تعلق ندارد.

$$\bar{A} = X - A \Leftrightarrow \mu_{\bar{A}}(x) = 1 - \mu_A(x)$$

۲-۳-۸. اجتماع^{۲۹} :

از ماکزیمم استفاده می کنیم. اجتماع باید هم A و هم B را در بر بگیرد، به گونه ای که A باید زیرمجموعه $A \cup B$ باشد. مقادیر دیگری هم غیر از ماکزیمم می تواند باشد.

$$C = A \cup B \Leftrightarrow \mu_c(x) = \max(\mu_A(x), \mu_B(x)) = \mu_A(x) \vee \mu_B(x)$$

۲-۳-۹. اشتراک^{۳۰} :

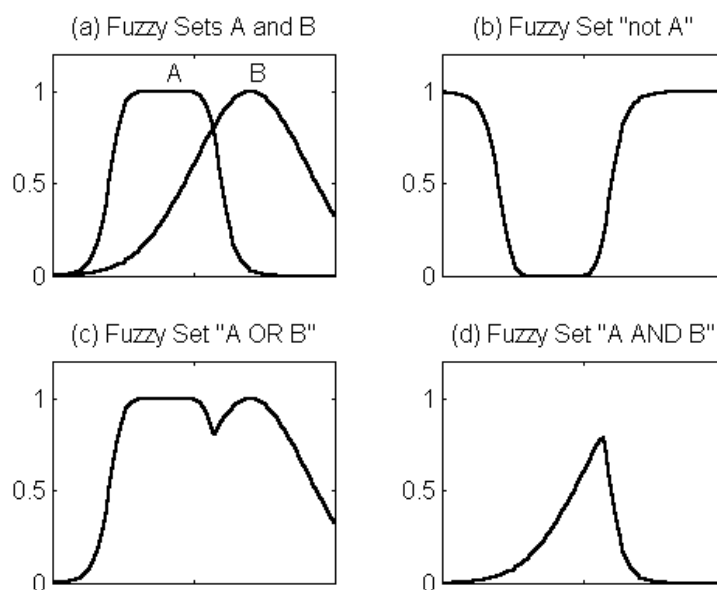
از مینیمم استفاده می کنیم.

$$C = A \cap B \Leftrightarrow \mu_c(x) = \min(\mu_A(x), \mu_B(x)) = \mu_A(x) \wedge \mu_B(x)$$

²⁸ complement

²⁹ union

³⁰ intersection



۴-۲. *Mf-formulation*

۱-۴-۲. مثلثی^{۳۱}: سه نقطه لازم است که این تابع تعریف شود. (a, b, c)

$$\text{trimf}(x; a, b, c) = \max\left(\min\left(\frac{x-a}{b-a}, \frac{c-x}{c-b}\right), 0\right)$$

۲-۴-۲. دوزنقه ای^{۳۲}: چهار نقطه نیاز دارد. (a, b, c, d)

$$\text{trapmf}(x; a, b, c, d) = \max\left(\min\left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right), 0\right)$$

۳-۴-۲. گوسین^{۳۳}: دو پارامتر دارد. مرکز c و بازه a را تعریف می کنید. a در حقیقت *standard deviation*

نیست. a میزان باز یا بسته بودن تابع عضویت است.

$$\text{gaussmf}(x; a, b, c) = e^{-\frac{1}{2}\left(\frac{x-c}{a}\right)^2}$$

۴-۴-۲. تابع بل تعمیم یافته^{۳۴}: سه تا پارامتر لازم دارد.

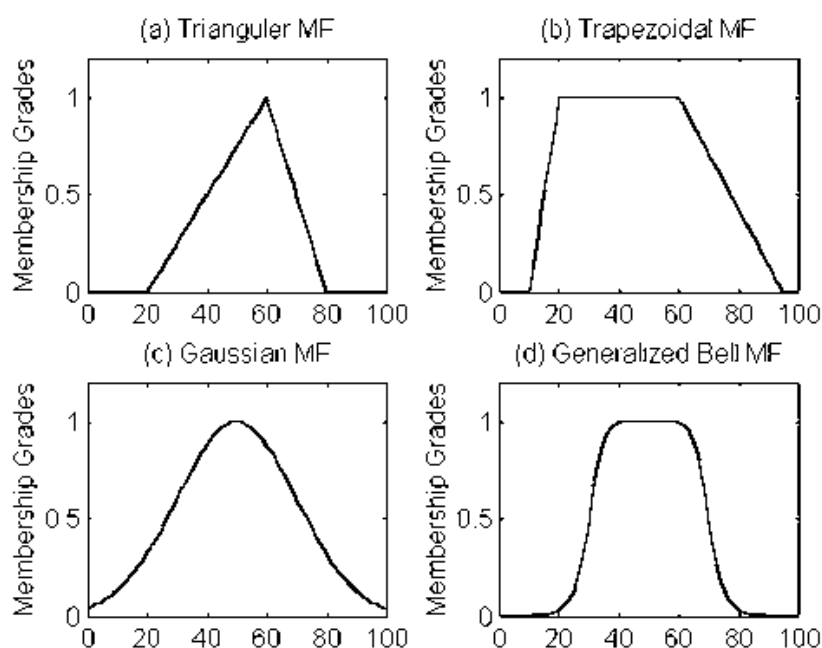
$$\text{gbellmf}(x; a, b, c) = \frac{1}{1 + \left|\frac{x-c}{b}\right|^{2a}}$$

³¹ Triangular

³² Trapezoidal

³³ Gaussian

³⁴ Generalized bell



مثلا اگر بخواهید ساختاری داشته باشید و بخواهید از آن برای بهینه سازی مشتق بگیرید، آن موقع باید از تابع عضویت گوسین استفاده شود.

۵-۴-۲. مشکل تابع عضویت گوسین :

هیچگاه به صفر نمی رسد؛ در حالی که وقتی بخواهیم دمای گرم را تعریف کنیم، بازه خاصی مدنظر است و می-خواهیم در یک حالتی اصلاً این قانون مورد توجه نباشد. مثلاً در زمانی که دما $20^{\circ}C$ است نمی خواهیم به میزان ۰/۰۰۱ قطعیت داشته باشد.

در جاهایی که مبتنی بر دانش انسان است (پایگاه داده مبتنی بر دانش انسان) بیشتر از ساختارهای مثلثی یا دوزنقه ای استفاده می کنیم و در جاهایی که مبتنی بر دانش انسانی نیست (به صورت زیاد) از تابع عضویت گوسین و G -bell استفاده می کنیم.

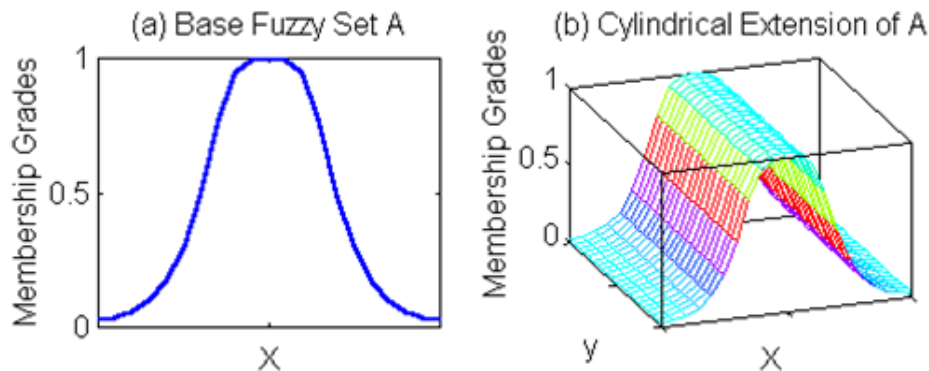
۶-۴-۲. مزایای تابع عضویت گوسین:

مزیت تابع عضویت گوسین آن است که مشتق آن ساده بوده و فرم خودش را دارد. به روش های دیگری هم می توان توابع عضویت را ایجاد کرد مثل *sigmoidal*.

$$\text{sigmf}(x; a, b, c) = \frac{1}{1 + e^{-a(x-c)}}$$

۵-۲. توسعه استوانه ای (Cylindrical Extension):

توسعه استوانه ای این امکان را می دهد که از فضای m بعدی به فضای n بعدی بروید. قابلیت بسیار مهمی می باشد.



گاهی اوقات از شما خواسته می شود دمای داغ را تعریف کنید چه می گوئید؟

مثلا تابع عضویتی تعریف می کنید که مرکز آن $40^{\circ}C$ است.

زمانی از شما خواسته می شود، هوای مطلوب را تعریف کنید. کار مشکل است چون از ۲ پارامتر تشکیل می شود:

۱. دمای مطلوب

۲. رطوبت مطلوب.

در حقیقت در جایی شما نیاز دارید از فضای یک بعدی به فضای چند بعدی بروید.

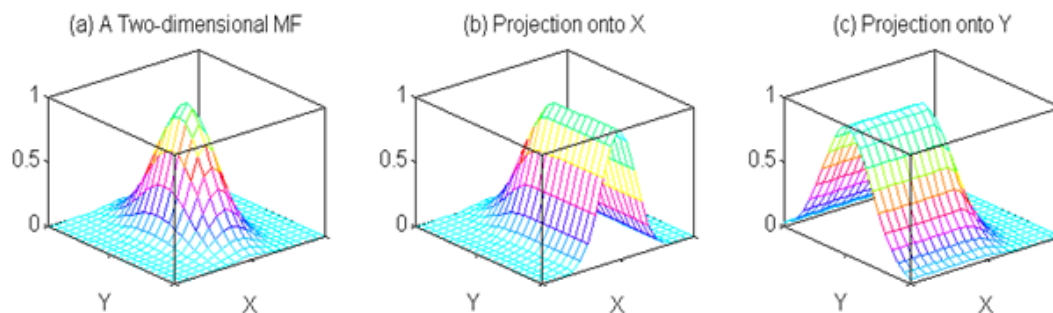
۶-۲. تصویر (projection):

عملگر عکس توسعه استوانه ای عملگر $projection$ است. در $projection$ از یک فضای دو بعدی به یک فضای

یک بعدی می روید. از لحاظ ریاضی خیلی ساده می شود:

$$\mu_A(x) = \max_y \mu_R(x, y)$$

$$\mu_B(y) = \max_x \mu_R(x, y)$$



در *project*، فرض کنید یک منبع نور پشت آن قرار گرفته و تصویر روی محور ایجاد شده است. از این دو برای رفتن از فضای m بعدی به n بعدی و بالعکس استفاده می شود.

هنگامی که دمای مطلوب و رطوبت مطلوب را تعریف کردیم، آنگاه تک تک آنها را *cylindrical extend* کرده و بین آنها اجتماع می گیریم (یا دمای مناسب یا رطوبت مناسب). یا بالعکس بین آنها اشتراک می گیریم.

۷-۲. مکمل فازی (Fuzzy complement):

سه شرط برای مکمل بودن وجود دارد.

۱. *boundary*:

$$N(1) = 0 \quad \text{and} \quad N(0) = 1$$

۲. *Monotonicity*:

$$N(a) > N(b) \quad \text{if} \quad a < b$$

۳. *Involution*:

$$N(N(a)) = a$$

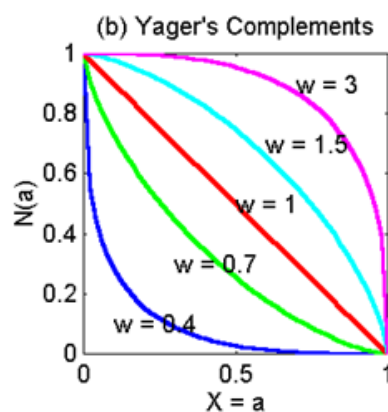
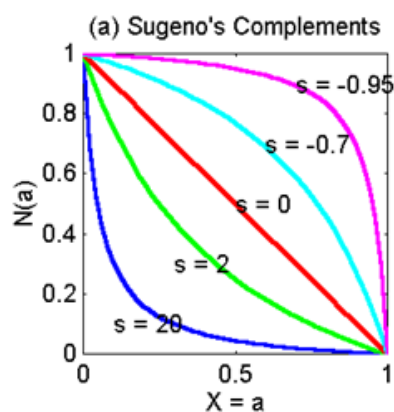
۷-۲-۱. مکمل Sugeno:

سوگنو مکمل را پارامتریک بیان نمود. بنابر آن پارامتر رفتارش تغییر می کند.

$$N_s(a) = \frac{1-a}{1+sa}$$

۷-۲-۲. مدل Yager:

$$N_w(a) = (1-a^w)^{1/w}$$



۸-۲. عملگر اشتراک (T -norm):

T -norm نیز دارای چهار شرط می باشد.

۱. $Boundary$:

$$T(0,0) = 0, \quad T(a,1) = T(1,a) = a$$

۲. $Monotonicity$:

$$T(a,b) < T(c,d) \text{ if } a < c \text{ and } b < d$$

۳. $Commutativity$:

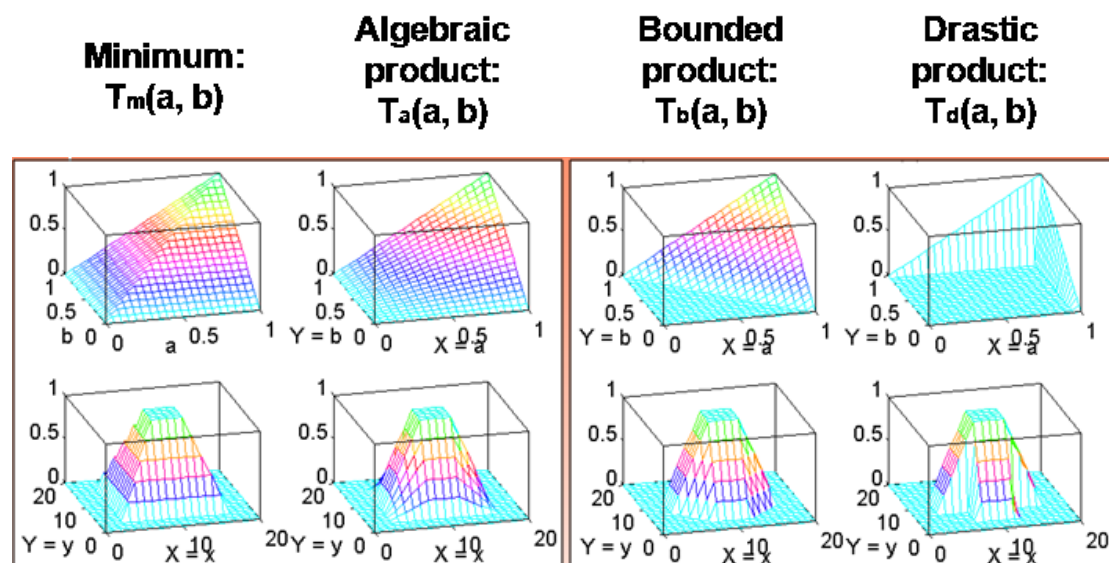
$$T(a,b) = 0, \quad T(b,a)$$

۴. $Associativity$:

$$T(a, T(b,c)) = T(T(a,b), c)$$

چند نوع از انواع T -norm به شکل زیر می باشند.





بزرگترین T -norm، $\min$ و کوچک ترین T -norm، $Drastic product$ می باشد.

۹-۲. عملگر اجتماع (S -norm):

برای S -norm نیز خواصی مانند T -norm تعریف می گردد.

۱. $Boundary$:

$$S(1, 1) = 1, S(a, 0) = S(0, a) = a$$

۲. $Monotonicity$:

$$S(a, b) < S(c, d) \text{ if } a < c \text{ and } b < d$$

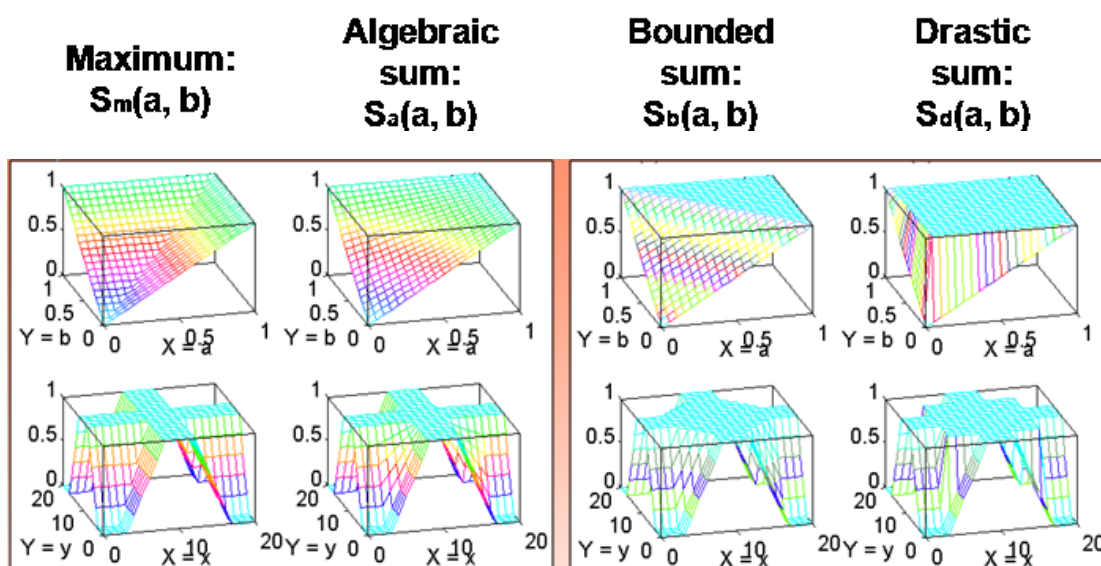
۳. $Commutativity$:

$$S(a, b) = S(b, a)$$

۴. $Associativity$:

$$S(a, S(b, c)) = S(S(a, b), c)$$

چند نوع از انواع S -norm به شکل زیر می باشند.



به همین ترتیب برای S -norm ها داریم:

کوچک ترین S -norm، $\max$ و بزرگ ترین S -norm، $Drastic Sum$ می باشند.

۲-۱. Associative set :

اگر برای مجموعه ای از T -norm، S -norm و مکمل بتوان خاصیت دموگران را ثابت کرد، آنگاه گفته می شود که

این مجموعه انجمنی^{۳۵} است. می توان ثابت کرد که $basic complement, max, min$ این خاصیت را دارند.

$T_m(a, b)$	$\longleftrightarrow$	$S_m(a, b)$
$T_a(a, b)$	$\longleftrightarrow$	$S_a(a, b)$
$T_b(a, b)$	$\longleftrightarrow$	$S_b(a, b)$
$T_d(a, b)$	$\longleftrightarrow$	$S_d(a, b)$

کلا T -norm ها و S -norm ها و $Fuzzy complement$ در طبقه ای از دانش با نام $meta knowledge$ قرار

می گیرند.

$Meta knowledge$ یعنی ورای دانش یا دانش در مورد دانش. در مقابل $Meta knowledge$ ، $Domain$

$knowledge$ وجود دارد. $Domain knowledge$ دانش خودآگاه می باشد.

در مجموعه های فازی قواعد اگر آنگاه جزء $Domain knowledge$ است.

³⁵ Associative

۱۱-۲. اصل توسعه (Extension principle):

اصل توسعه این امکان را می دهد که با اعدادی که توام با عدم قطعیت هستند بتوان محاسبات را انجام داد.

به عنوان مثال برای ساخت خانه هزینه ها شامل هزینه کارگر + هزینه آهن + هزینه ... می باشند. که هیچ یک از اینها قطعی نیستند. می توان برای تمام آنها تابع فازی نوشت. حال اگر بخواهیم اینها را با هم جمع کنیم از اصل توسعه استفاده می نماییم.

اگر A یک مجموعه به صورت زیر فازی باشد.

$$A = \mu_A(x_1)/x_1 + \mu_A(x_2)/x_2 + \dots + \mu_A(x_n)/x_n$$

و داشته باشیم $y=f(x)$ ، $f(A)$ به صورت زیر تعریف می شود:

$$B = \mu_B(y_1)/y_1 + \mu_B(y_2)/y_2 + \dots + \mu_B(y_n)/y_n$$

where $y_i = f(x_i)$, $i = 1$ to n

اگر نگاشت یک به یک باشد که مشکلی وجود ندارد و خیلی ساده می توان گفت که $f(x)$ به همان میزان تعلق دارد که x تعلق دارد. (نگاشت یک به یک است).

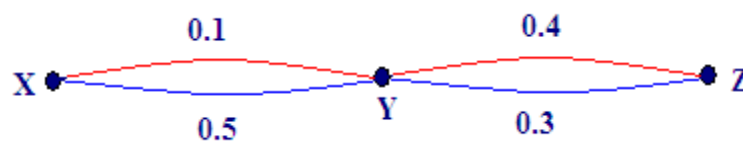
اگر نگاشت چند به یک باشد، به صورت زیر این کار انجام می شود.

$$\mu_B(y) = \max_{x=f^{-1}(y)} \mu_A(x)$$



برای توجیه اینکه چرا از ماکزیمم استفاده می شود می توان گفت: اگر از X به Y دو ریسمان وجود داشته باشد که ریسمان اولی قوت ۰/۱ و دیگری قوت ۰/۵ را دارد، در صورت کشش، ریسمان زمانی پاره می شود که قوت ماکزیمم از بین برود.

حالا اگر یک نگاشت دیگری داشته باشیم که از Y به Z برود با چه قدرتی می رود؟



چون سری است، ریسمانی حاکم است که از همه ضعیف تر باشد، بنابراین از مینیمم استفاده می شود. این $Fuzzy$

Relation است.

مثال:

$$1 \underset{\sim}{=} \frac{0.1}{-1} + \frac{0.3}{0} + \frac{1}{1} + \frac{0.5}{2}$$

$$1 \underset{\sim}{+} 1 \underset{\sim}{=} ?$$

$$1 \underset{\sim}{+} 1 \underset{\sim}{=} \frac{0.1}{f(-1,0)} + \frac{0.1}{f(-1,1)} + \frac{0.1}{f(-1,2)} + \frac{0.1}{f(-1,-1)} + \frac{0.1}{f(0,-1)} + \frac{0.3}{f(0,0)} + \frac{0.3}{f(0,1)} + \frac{0.3}{f(0,2)} + \frac{0.1}{f(1,-1)} + \frac{0.3}{f(1,0)} + \frac{1}{f(1,1)} + \frac{0.5}{f(1,2)} + \frac{0.1}{f(2,-1)} + \frac{0.3}{f(2,0)} + \frac{0.5}{f(2,1)} + \frac{0.5}{f(2,2)}$$

$$\Rightarrow 1 \underset{\sim}{+} 1 \underset{\sim}{=} \frac{0.1}{-1} + \frac{0.1}{-2} + \frac{0.3}{0} + \frac{0.3}{1} + \frac{1}{2} + \frac{0.5}{3} + \frac{0.5}{4}$$

۱۲-۲. رابطه فازی :

R یک رابطه فازی می باشد.

$$R = \{((x, y), \mu_R(x, y)) \mid (x, y) \in X \times Y\}$$

مثال:

- مثلاً x نزدیک y است.
- x وابسته به y است. (x و y دو واقعه هستند)
- x و y شبیه هم هستند. (x و y فرد یا شیء هستند)

۱۳-۲. ترکیب ($Max-Min$ composition):

$Max-Min$ Composition دو رابطه فازی R_1 و R_2 به صورت زیر تعریف می گردند.

$$\mu_{R_1 \circ R_2}(x, z) = \bigvee_y [\mu_{R_1}(x, y) \wedge \mu_{R_2}(y, z)]$$

ویژگی ها:

۱. انجمنی:

$$R \circ (S \circ T) = (R \circ S) \circ T$$

۲. توزیع پذیری بر روی اجتماع

$$R \circ (S \cup T) = (R \circ S) \cup (R \circ T)$$

۳. توزیع پذیری ضعیف بر روی اشتراک

$$R \circ (S \cap T) \subseteq (R \circ S) \cap (R \circ T)$$

۴. *Monotonicity*:

$$S \subseteq T \Rightarrow (R \circ S) \subseteq (R \circ T)$$

Max-Min فقط یک نوع *composition* است، بطور کلی می توانیم *Max-Star* داشته باشیم.

هر *T-norm* ای می تواند یک *Star* باشد، اما به جای *max* نمی توان از هر *S-norm* دیگری استفاده نمود. زیرا اگر از *max* استفاده نشود دیگر شرایط *composition* برقرار نمی گردد.

۲-۱۴. ترکیب *Max-product*:

$$\mu_{R_1 \circ R_2}(x, z) = \bigvee_y [\mu_{R_1}(x, y) \mu_{R_2}(y, z)]$$

و بطور کلی برای *Max-Star* داریم:

$$\mu_{R_1 \circ R_2}(x, z) = \bigvee_y [\mu_{R_1}(x, y) * \mu_{R_2}(y, z)]$$

مثال:

$$R_x = \begin{bmatrix} 0.1 & 0.2 & 0.5 \\ 0 & 1 & 1 \\ 2 & 0.6 & 0.3 \end{bmatrix} \quad R_y = \begin{bmatrix} 0.9 & 0 \\ 0.3 & 1 \\ 0 & 0.2 \end{bmatrix}$$

R_x یک *relation* بین فضای x, y و R_y نگاشتی بین y و z می باشد. می خواهیم رابطه x, z را بدست آوریم.

$$MAX - MIN \Rightarrow R_{x \rightarrow z} = \begin{bmatrix} 0.2 & 0.3 \\ 0.3 & 1 \\ 0.3 & 0.6 \end{bmatrix}$$

$$MAX - Product \Rightarrow R_{x \rightarrow z} = \begin{bmatrix} 0.09 & 0.2 \\ 0.3 & 1 \\ 0.18 & 0.6 \end{bmatrix}$$

مثلا دوری شهرها از یکدیگر را می توان به این طریق محاسبه نمود.

۱۵-۲. متغیرهای زبانی (Linguistic variable):

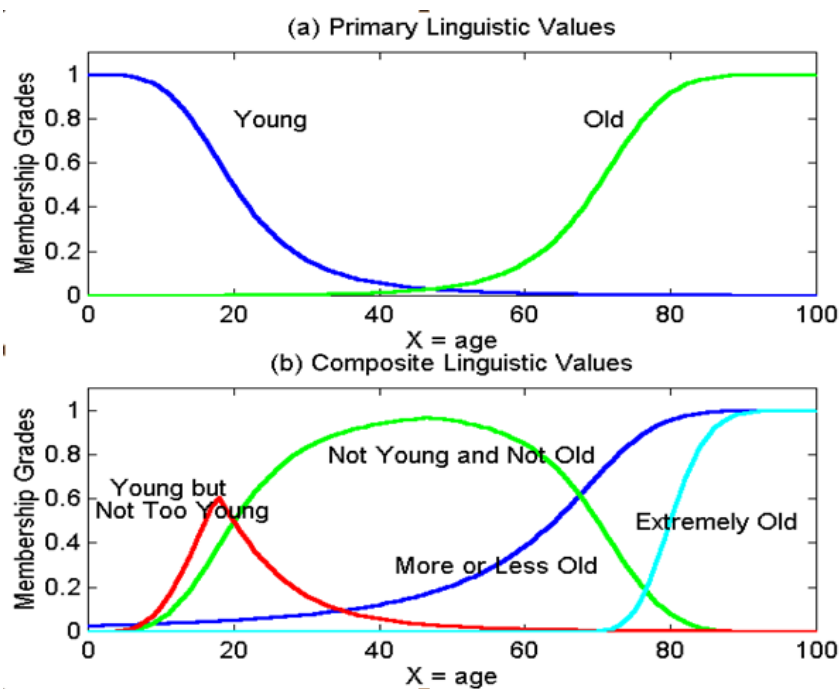
وقتی از شما این سوال پرسیده می شود که هوا امروز چطور است؟ به عنوان مثال پاسخ می دهید کمی سرد یا کمی گرم است اما هیچگاه پاسخ نمی دهید $29^{\circ}C$.

خیلی جالب است که بتوان با کلام محاسبه انجام داد و محاسبه با کلمات^{۳۶} همان چیزی بود که زاده در دو دهه اخیر مطرح نمود. یعنی اینکه بتوان با کلمات محاسبه انجام داد.

در متغیرهای کلامی مثل دما، متغیر یک کلمه به خود اختصاص می دهد. مقادیر زبانی، مجموعه های فازی هستند. به عنوان مثال اگر سن را در نظر بگیریم:

All linguistic values form a term set:

$T(\text{age}) = \{\text{young, not young, very young, ...}$
middle aged, not middle aged, ...
old, not old, very old, more or less old, ...
not very young and not very old, ...}\}



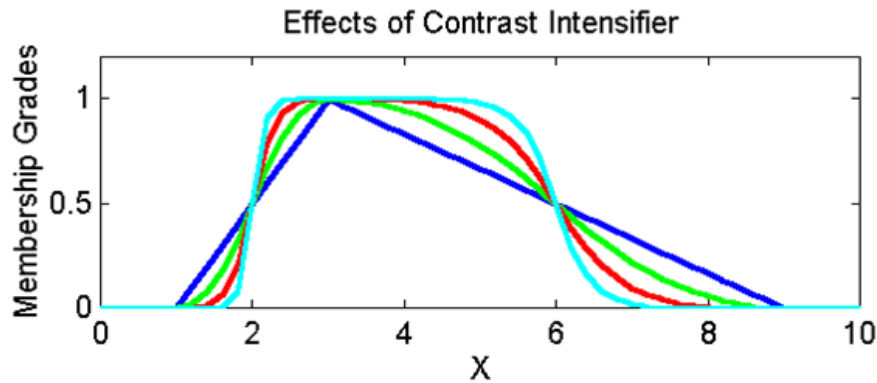
Concentration: $CON(A) = A^2$

³⁶ Computing with words

Dilation: $DIL(A) = A^{0.5}$

Contrast

$$intensification: INT(A) = \begin{cases} 2A^2, & 0 \leq \mu_A(x) \leq 0.5 \\ -2(\neg A)^2, & 0.5 \leq \mu_A(x) \leq 1 \end{cases}$$



۱۶-۲. قوانین اگر-آنگاه فازی:

دو نگرش برای قوانین فازی وجود دارد: نگرش کلاسیک، نگرش مدرن

۱. **نگرش کلاسیک:** می گوید که کلا منطق فازی تعمیم یافته منطق کلاسیک است. با این دیدگاه

علاقه‌مندیم هر آنچه در منطق کلاسیک داریم را تعمیم دهیم. مثل مودس پوننس، مودس تولنس و در

این دیدگاه $p \rightarrow q$ یا حقیقت دارد یا باطل است.

۲. **نگرش دوم:** می گوید، وقتی می گوییم $p \rightarrow q$ ، منظور این است که اگر p بود آنگاه q ولی اگر p

نبود این قانون را فراموش کن. فقط $p \rightarrow q$ زمانی معتبر است که p وجود داشته باشد در جاهای دیگر

انگار وجود نداشته یا *don't care* است.

هر دو دیدگاه در منطق فازی مطرح است.

اگر داشته باشیم $x=a, y=f(x)$ آنگاه برای بدست آوردن مقدار y چه می کنیم؟

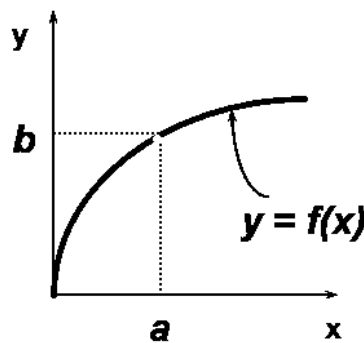
یک خط عمودی در a کشیده و در نقطه ای که $f(x)$ را قطع کرد، یک خط افقی می کشیم و می بینیم که کجا

محور دیگر را قطع می کند. به عبارت دیگر A را ابتدا *Cylindrical extend* کرده و وجه اشتراک آن را با منحنی

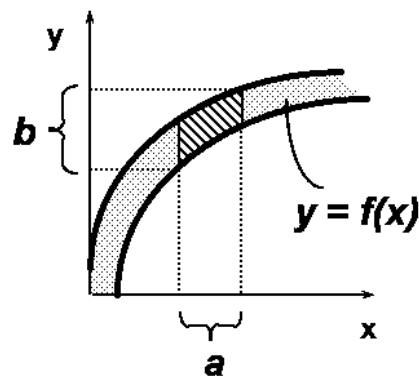
بدست می آوریم و آن وجه اشتراک را بر روی y ، تصویر می کنیم.

حال اگر به جای یک عدد یک بازه داشته باشیم و f به جای اینکه منحنی باشد یک *Fuzzy-interval value* باشد باز هم مانند قبل عمل می کنیم.

Derivation of $y = b$ from $x = a$ and $y = f(x)$:



a and b : points
 $y = f(x)$: a curve



a and b : intervals
 $y = f(x)$: an interval-valued function

A دیگر یک *interval* فازی است و F نیز یک *Fuzzy relation* است.

۱۷-۲. استنتاج فازی (Fuzzy Reasoning):

در مودس پوننس کلاسیک اگر داشته باشیم $A \rightarrow B$ و A موجود نباشد، اگر A' به اندازه ε با A فرق داشته باشد هیچ چیزی نمی توان گفت.

قابلیت استنتاج تقریبی^{۳۷} در فازی مطرح می شود. یعنی به میزانی که A' به A نزدیک باشد B' نیز به B نزدیک است. این یک قانون بسیار ارزشمند است. چرا؟ زیرا در طبیعت امکان ندارد بتوان تمام حالات ممکن را پیش بینی کرد. در حقیقت به میزانی که A و A' با هم تطبیق دارند B آتش می شود.

این همان قدرت فازی است که با دو قانون می توان به *resolution* ای رسید که با یک میلیون قانون در سیستم خبره نمی توان به آن دست یافت.

از هر T -norm ای می توان در استنتاج استفاده کرد مانند مینیمم.

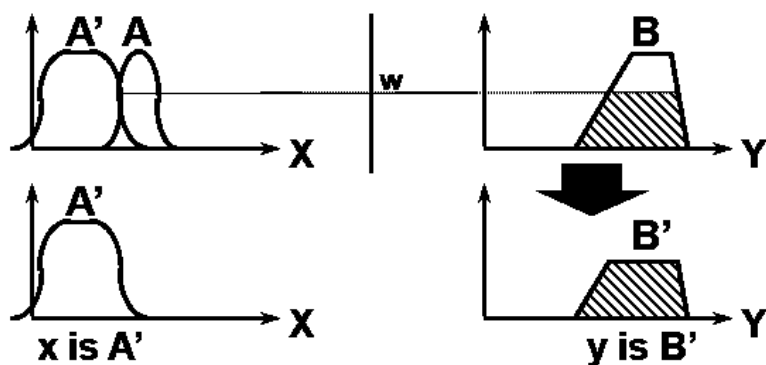
³⁷ Approximate Reasoning

۱-۱۷-۲. یک قانون با یک مقدم^{۳۸}:

Rule: if x is A then y is B

Fact: x is A'

Conclusion: y is B'

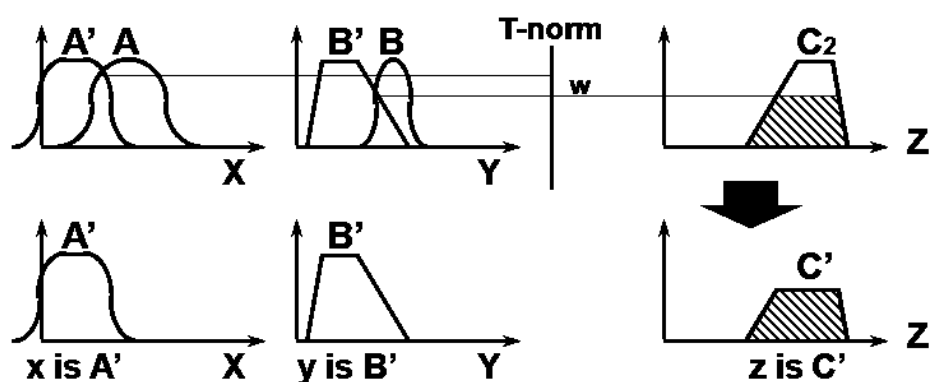


۲-۱۷-۲. یک قانون با چند مقدم:

Rule: if x is A and y is B then z is C

Fact: x is A' and y is B'

Conclusion: z is C'



۳-۱۷-۲. چند قانون با چند مقدم:

if x is A_1 and y is B_1 then z is C_1

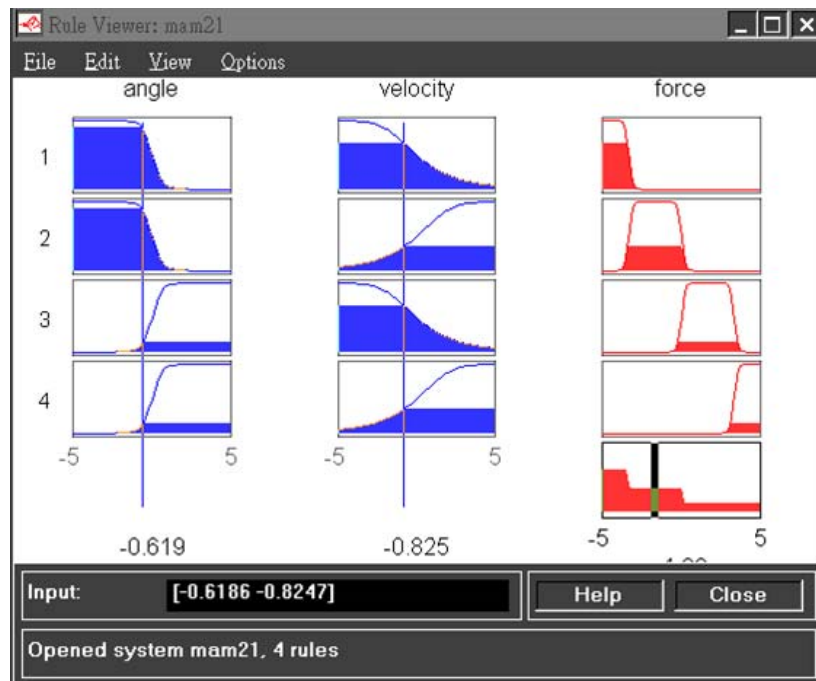
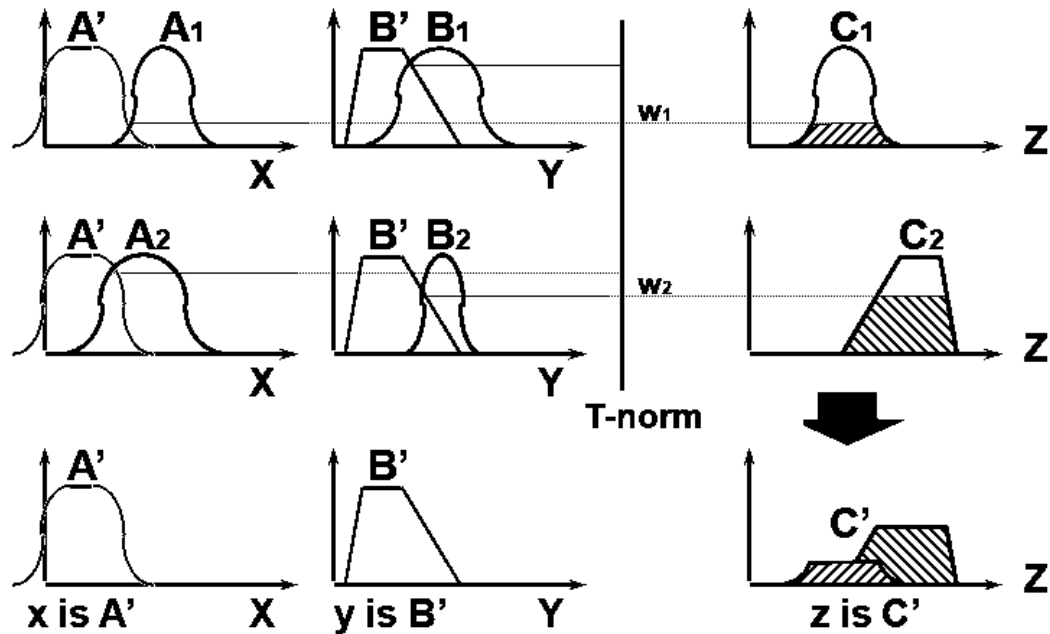
³⁸ Antecedent

if x is A_2 and y is B_2 then z is C_2

Fact: x is A' and y is B'

Conclusion: z is C'

در صورتی که بیش از یک قانون وجود داشته باشد، برای هر یک، خروجی را همانند بالا بدست آورده و در نهایت اجتماع آنها را بدست می آوریم.



۱۸-۲. متغیرهای دیگر:

درجه تطابق^{۳۹}: یعنی هر چقدر که A' به A نزدیک تر باشد به همان میزان بیشتر قطعیت دارد و به همان میزان

بیشتر قانون آتش شده و به آن اعتبار داده می شود.

Firing strength: به معنای میزان آتش است.

Qualified (induced) MF: آن توابع عضویتی است که به میزانی آتش شده و قطعیت دارند.

حال اگر مثلاً نتیجه اجتماع حاصل از مراحل قبل بخواهد روی یک کولر اعمال شود، کولر مجموعه فازی را نمی-

فهمد؛ بنابراین باید از این نمودار به نحوی عدد استخراج شود که به این فرایند غیر فازی سازی^{۴۰} گفته می شود.

۱۹-۲. روش های غیر فازی سازی:

۱. *Mean of Maximum*

۲. *Smallest of Maximum*

۳. *Largest of Maximum*

۱. *Bisector of area*

۲. *Centroid of area* (مرکز ثقل)

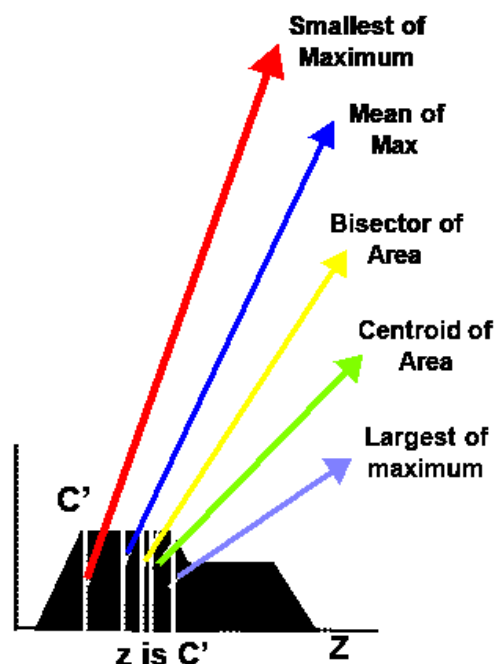
$$y^* = \frac{\int_V y \mu_{B'}(y) dy}{\int_V \mu_{B'}(y) dy}$$

۳. *Center Average*

$$y^* = \frac{\sum_{l=1}^M \bar{y}^l w_l}{\sum_{l=1}^M w_l}$$

³⁹ Degrees of compatibility

⁴⁰ Defuzzification



۱-۱۹-۲. مشکلات روش مرکز ثقل:

۱. اگر یک خروجی به خاطر نویز ایجاد شود، و نویز ایجاد شده در سیستم در دوردست به میزان 0.05 قطعیت داشته باشد، مرکز ثقل بسیار جابه جا شده و در جایی قرار می گیرد که هیچ شباهتی با جایی که باید داشته باشد ندارد.

یک راه حل این است که به میزانی که فکر می کنیم در سیستم نویز وجود دارد، به همان میزان سیستم را فیلتر کنیم. یعنی مثلاً اگر فرض کنیم به میزان α نویز در سیستم وجود دارد، یک α -cut روی مجموعه خروجی فازی تعریف کرده و مرکز ثقل را صرفاً داخل α -cut بدست می آوریم. به این ترتیب سیستم از نویزهای محیطی مصون می ماند.

۲. برای بدست آوردن آن لازم است دو بار انتگرال گرفته شود. این انتگرال خیلی زمان بر بوده و در بسیاری از سیستم ها نمی توان از آن استفاده نمود.

در *Center Average* کل سیستم فازی را با یک *Fuzzy singleton* تقریب می زنیم. معمولاً تالی را یک *singleton fuzzifier* در نظر می گیریم. این *singleton fuzzifier* را به هیچ وجه نباید روی ورودی ها اعمال نمود. چرا؟

زیرا مفهوم فازی را از دست می دهیم. توابع عضویت ورودی هم اگر *singleton* باشند دیگر مسئله *crisp* می شود. با این کار از توانایی سیستم فازی به عنوان *universal approximator* کاسته نشد. این ساختار، مشتق پذیر نیز

می باشد.

۲-۱۹-۲. *Bisector of area*

سعی می کند مشکل روش مرکز ثقل را حل کند. بنابراین نقطه ای را پیدا کرده که طرف راست سطح زیر نمودار و طرف چپ سطح زیر نمودار با هم برابر باشند. اما مشکل زمان بر بودن را حل نمی کند.

گاهی اوقات فضای خروجی گسسته است. آن وقت از max ها استفاده می کنیم.

۲-۱۹-۳. نکات کلیدی در انتخاب غیر فازی سازها :

۱. مشتق پذیر بودن

۲. *universe approximate* بودن

۳. میزان محاسبات.

۲-۲۰. سیستم استنتاج فازی:

به *Fuzzy inference system* به دید نگاشت، نگاه می شود که در مدلسازی فازی برای مدلسازی سیستم های پیچیده یا پیش بینی بازار بورس و یا پیش بینی آب و هوا کاربرد دارد.

در شبکه عصبی، حافظه انجمنی^{۴۱} بدین معناست که ما اطلاعات را تنها در یک نرون ذخیره نمی کنیم. یعنی اگر یک نرون از بین برود دیگری جبران کار آن را می کند.

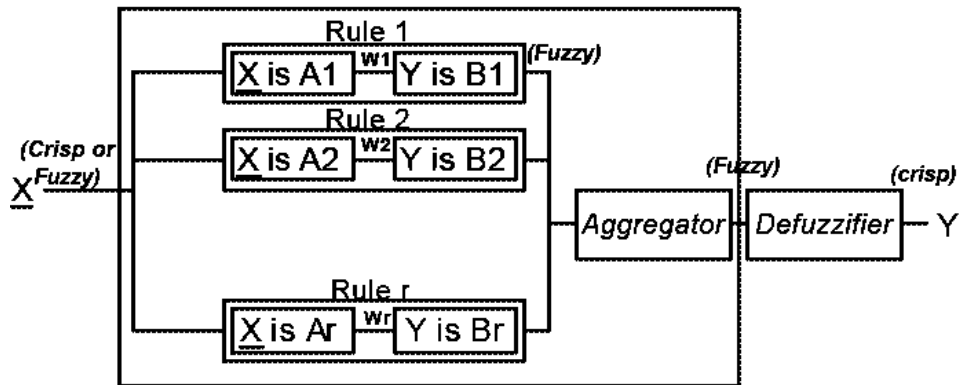
در حافظه انجمنی فازی^{۴۲} قابلیت تعمیم پذیری وجود دارد. یعنی خیلی بیشتر از آن اطلاعات خامی که به آن می - دهیم اطلاعات در آن حفظ می شود.

به عنوان مثال اگر قانونی داریم که افراد مسن یک سری خصوصیات دارند و افراد جوان یک سری خصوصیات دیگری دارند، آنگاه می دانیم که فردی که سن وی ۵۰ است باید چه خصوصیتی داشته باشد. در حقیقت تعاملی بین مقداری که جوان و مقداری که مسن است وجود دارد. بنابراین حافظه انجمنی فازی می تواند هم در مدل ها و هم در کنترلرها بکار رود. در بعضی مقالات به آن *FAM BANK* گفته می شود.

⁴¹ *Associative memory*

⁴² *Fuzzy associative memory*

۲-۲۰-۱. بخش های مختلف سیستم استنتاج فازی:



ورودی x به مجموعه فازی هم می تواند فازی و هم *crisp* باشد.

گاهی اوقات با اینکه اطلاعات ما با اندازه گیری^{۴۳} بدست آمده باز می خواهیم آنها را *Fuzzify* کنیم چرا؟

زیرا این اطلاعات توام با نویز هستند و ما به آنها اطمینان نداریم.

برای اینکه یک سیستم فازی کامل داشته باشیم باید موارد زیر را تعریف کنیم:

۱. *T-norm : And operator*

۲. *S-norm : Or operator*

۳. *Implication*

۴. *Aggregation*

۵. *Defuzzification*

۲-۲۱. انواع قوانین اگر-آنگاه فازی:

قوانین اگر-آنگاه فازی دو نوع هستند:

۱. *Mamdani*: قسمت مقدم و تالی آن فازی است.

۲. *Sugeno*: قسمت تالی آن فازی نیست بلکه یک رابطه عددی است.

• نوع صفر

• نوع یک

$$y = \bar{y}_i * 5 + x_i$$

⁴³ measurement

• نوع دو

$$y = \bar{y}_i + 5 * x_i^2$$

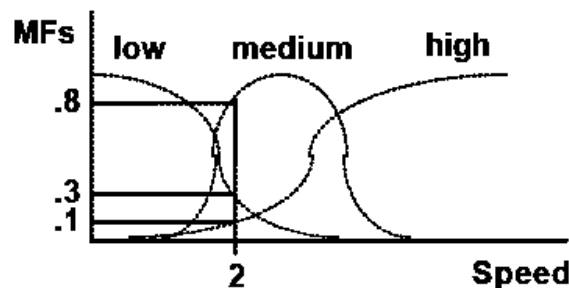
به مدل سوگنو TS یا TSK هم گفته می شود. قسمت تالی رابطه ای بین ورودی و خروجی می باشد.

مثال:

If speed is low then resistance = 2

*If speed is medium then residence = 4*speed*

*If speed is high then resistance = 8*speed*



Rule 1: $w1=0.3$; $r1=2$

*Rule 2: $w2=0.8$; $r2=4*4$*

*Rule 3: $w3=0.1$; $r3=8*2$*

$$\rightarrow \text{Resistance} = \sum (w_i * n_i) / \sum w_i = 7.12$$

ممدانی از نوع *center average defuzzifier* معادل با TSK نوع صفر می باشد.

۲-۲۲. نحوه تنظیم قوانین:

هر چقدر توابع عضویت زیادتر باشد، دقت زیادتر شده اما مشکل این است که هر چه توابع بیشتر شود تعداد قوانین

هم رشد می کند. که این رشد یک رشد خطی نبوده بلکه رشد نمایی می باشد. یعنی اگر ما فضایی داشته باشیم که

$n=2$ باشد و برای هر متغیر ۳ تابع عضویت تعریف کنیم کلاً ۳^۲ قانون باید داشته باشیم.

اگر ۳ تابع عضویت به ۴ تبدیل شود، تعداد قوانین بسیار زیادتر می شود. در اینجا مسئله نفرین (طلسم) ابعاد مطرح

می شود.

در سیستم های فازی هم این مشکل بوجود می آید در عین حالی که می گوئیم فازی برای حل مسائل پیچیده بکار

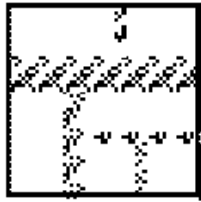
می رود.

۲-۲۳. تقسیم بندی فضای ورودی:

Grid partitioning



Tree partitioning



Scatter partitioning

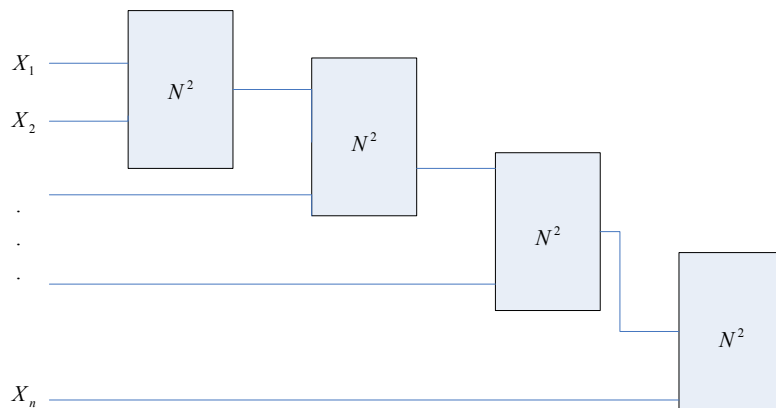


۱. **Grid partitioning**:

به علت های ذکر شده در بالا وقتی ورودی ها و خروجی ها افزایش می یابد با مشکل مواجه می گردد.

۲. **Tree partitioning**:

به صورت سلسله مراتبی این کار انجام می شود. چگونه سلسله مراتب را تعریف کنیم؟



تعداد قوانین برابر با $N^2 \times (n-1)$ می شود. در حقیقت n از نما به مضرب می آید. رشد قوانین به مراتب کاهش می یابد (چون N^2 ثابت است).

۳. **Scatter partitioning**:

بر اساس توزیع واقعی داده ها، توابع عضویت را پارتیشن کنیم. پارتیشن را بر اساس داده هایی که داریم انجام می - دهیم.

مزیت: تعداد قوانین را کاهش می دهد.

مشکل: همه قسمت ها را پوشش نمی دهد.

۲-۲۴. مدلسازی و کنترل فازی:

برای اینکه مسئله ای را حل کنیم، چه در مدلسازی و چه در کنترل سعی می کنیم هر چه *Domain knowledge* در مورد آن وجود دارد از انسان کشف کنیم. غیر از *Domain knowledge* دسترسی به یک مقدار داده عددی داریم. بنابراین در ابتدا

۱. منابع اطلاعاتی را شناسایی می کنیم که شامل موارد زیر می باشد:

- *Fuzzy Domain knowledge* : دانش انسانی

- *Numerical data*

۲. شناسایی ساختار^{۴۴}

- انتخاب نوع موتور استنتاج فازی

- مشخص نمودن وابستگی داده های ورودی و خروجی

- تعداد *linguistic term* هایی که می خواهیم *associate* کنیم.

۳. شناسایی عمیق ساختار^{۴۵}

- انتخاب توابع عضویت و پارامترهای مطلوب.

- استفاده از *Regression* یا روش های دیگر برای بهینه سازی *MF* ها استفاده می کنیم.

⁴⁴ *Surface structure identification*

⁴⁵ *Deep structure identification*

۳. شناسایی سیستم ها :

فرایند تعیین یک مدل ریاضی از یک سیستم ناشناخته توسط مشاهده ورودی ها و خروجی های آن را شناسایی سیستم گویند. شناسایی سیستم ها از دیدگاه های مختلفی ممکن است انجام شود که در چند نمونه از آنها را ذکر می نماییم.

۱- به منظور پیش بینی رفتار سیستم ها : اگر بتوانیم بر اساس داده های ورودی و خروجی مدل سیستم را بدست آوریم سپس قادر خواهیم بود رفتار آن سیستم را پیش بینی نماییم. این موضوع در مسائل متعددی نظیر پیش بینی آب و هوا و یا پیش بینی بازار بورس بسیار کاربرد دارد

۲- تغییر سیستم: شناسایی کنش ها و واکنش های بین ابزار سیستم و چگونگی ارتباط ورودی ها و خروجی ها به یکدیگر کمک می کند تا بتوانیم بفهمیم که اجزای مختلف سیستم چگونه با یکدیگر در تعاملند و به عبارتی در داخل سیستم چه می گذرد. در اینصورت شاید بتوانیم تغییراتی در سیستم ایجاد نماییم تا رفتار سیستم آنگونه باشد که ما می خواهیم

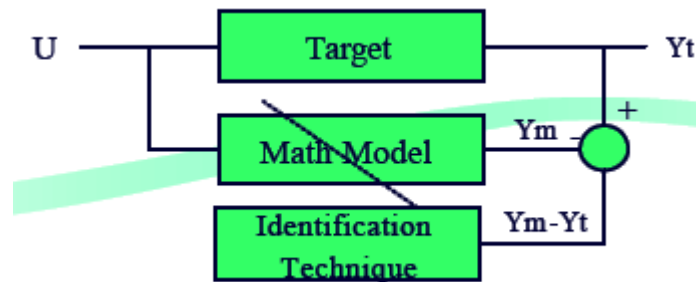
۳- شناسایی به منظور شبیه سازی: در این دیدگاه به دنبال شناسایی سیستم ها هستیم تا بتوانیم آن ها را کنترل نماییم. از شناسایی سیستم^{۴۶} برای طراحی کنترلر استفاده می نماییم تا بتوانیم رفتار سیستم را پیش بینی نماییم. بطور کلی سه علت اصلی وجود دارد که ما را وادار به شناسایی سیستم می نماید.

بطور معمول شناسایی سیستم در دو سطح انجام می شود :

۱. شناسایی ساختار: ساختار تعامل بین اجزای مختلف سیستم را شناسایی می نمایم. مثلاً شبکه عصبی چند لایه دارد، هر لایه چند نمره دارد. آیا سیستم درجه دو است یا درجه ۳ و شناسایی ساختار از اهمیت بسیار بالایی برخوردار است . بطوریکه اگر ساختار در آن قاب پاسخی نداشته باشد هر چقدر به دنبال جواب بگردیم طبیعی است که آن را پیدا ننماییم اما معمولاً مسئله ای است که از آن صرف نظر می شود و معمولاً در اکثر الگوریتم های بهینه سازی و شناسایی سیستم ها بحث اصلی بحث دوم و شناسایی پارامترها است.

۲. شناسایی پارامترها: در این سطح فرض بر این است که ساختار سیستم را می شناسیم و نقطه کلیدی ما این است که حال چه پارامترهایی بایستی ترکیب شوند تا این ساختار متناسب با مدل هدف تنظیم شود.

⁴⁶ System Identification



بطور معمول در اکثر سیستم ها ما یک هدف داریم و در زیر آن یک مدل ریاضی ، یک ساختار برای آن تنظیم می شود. از تفاوت میان مدل ریاضی و ساختار خطا محاسبه می شود و این خطا چیزی است که موتور تنظیم را آتش می کند.

بنابر این مرحله شناسایی پارامترها بخشی است که مدل را مانند شکل فوق تنظیم می نماید.

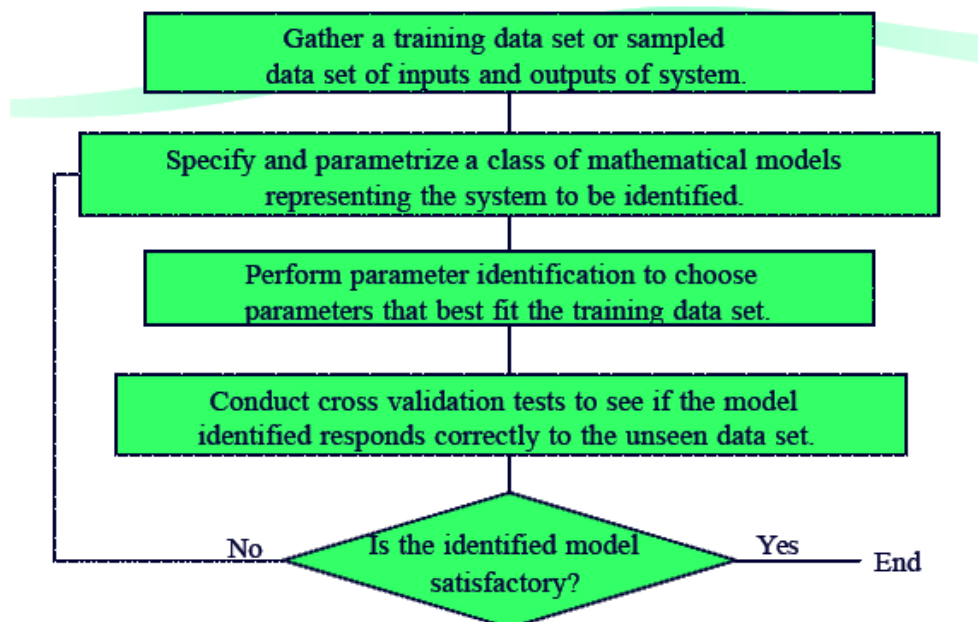
در مسائل اگر مدل *crisp* را داشته باشیم معمولا دلیلی ندارد بخواهیم مدل دیگری را بوجود آوریم. بدست آوردن مدل سیستم زمانی بکار می رود که مدل دقیق یا در دسترس نیست و یا اگر در دسترس هست خیلی زمان می برد و محاسبات آن امکان پذیر نیست.

مراحل کار برای مدل کردن یک سیستم یا پیش بینی رفتار سیستم، (مراحل کار در شناسایی سیستم):

۱. جمع آوری داده های آموزش توسط نمونه برداری از ورودی ها و خروجی های جعبه سیاه
۲. طراحی یک ساختار برای آن (سیستم خطی است- غیرخطی است، شبکه عصبی استفاده می شود، فازی استفاده می شود و ...)
۳. شناسایی پارامترها: برای تنظیم پارامترهای ناشناخته مدل تاکنون تا بتواند یک معیار خطا را کمینه نماید.
۴. اعتبار سنجی^{۴۷}: با یک سری داده جدید سیستم را تست کنیم اگر عملکرد حفظ شد و یا حداقل خطا خیلی افزایش پیدا نکرد در این صورت کار تمام شده و در غیر این صورت برمی گردیم و ساختار جدیدی را بررسی می کنیم.

⁴⁷ Cross validation

فلوچارت زیر مراحل شناسایی یک سیستم و مدل کردن آن را نمایش می دهد.



نمونه از روشهای معمول که برای شناسایی پارامترها^{۴۸} بکار می روند در ادامه توضیح داده می شوند .

۱-۳. روشهای مبتنی بر مشتق

۱-۱-۳. روش حداقل مربعات^{۴۹}

پیش از توضیح این روش نیاز به تعریف چند اصطلاح^{۵۰} می باشد :

گرادیان: گرادیان یک تابع برداری مانند f که تابعی از x_I تا x_n می باشد، برداری است که هر عضو آن یک مشتق f

نسبت به هر یک از x_I تا x_n های مختلف می باشد.

⁴⁸ parameter identification

⁴⁹ Least Squares Estimator

⁵⁰ Terminology

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}$$

Hessian: اگر دو مرتبه روش انجام شده برای گرادیان را تکرار نماییم یعنی از گرادیان مشتق بگیریم *Hessian*

بدست می آید. *Hessian* در حقیقت گرادیان گرادیان است و مشتق دوم f می شود که یک بردار $n \times n$ می باشد.

$$J_f = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \dots & \frac{\partial f_1(x)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n(x)}{\partial x_1} & \dots & \frac{\partial f_n(x)}{\partial x_n} \end{bmatrix}$$

Jacobian: برای بردار هایی از توابع برداری تعریف می شود و مشتق این توابع می باشد چنانکه اگر یک تابع n

بعدی داشته باشیم و بخواهیم مشتق آن را در فضای n بعدی x_1 تا x_n بدست آوریم حاصل ژاکوبین خواهد شد که یک ماتریس $n \times m$ می باشد.

$$H_f(x) = \nabla^2 f(x) = \begin{bmatrix} \frac{\partial}{\partial x_1} \left(\frac{\partial f}{\partial x_1} \right) & \dots & \frac{\partial}{\partial x_n} \left(\frac{\partial f}{\partial x_1} \right) \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_1} \left(\frac{\partial f}{\partial x_n} \right) & \dots & \frac{\partial}{\partial x_n} \left(\frac{\partial f}{\partial x_n} \right) \end{bmatrix}$$

استفاده از روش *LSE* به منظور شناسایی پارامترها برای یک سیستم خطی که به شکل زیر تعریف شده است.

$$y = \theta_1 f_1(u) + \theta_2 f_2(u) + \dots + \theta_n f_n(u)$$

f لزوماً خطی نیست و می تواند هر چیزی باشد. اما تا زمانی که θ به صورت خطی در فرمول ظاهر شود سیستم خطی است. چون تا زمانی که θ به صورت خطی ظاهر می شود برای ما مهم نیست f چه هست.

در رابطه فوق فرض بر این است که $f_1, f_2, \dots, f_n$ توابع شناخته شده و معلوم در $u = [u_1, u_2, \dots, u_p]$ می باشند، می خواهیم پارامترهای ناشناخته $\theta_1, \theta_2, \dots, \theta_n$ را محاسبه نماییم. بنابراین f یک بردار n بعدی است و θ هم یک بردار n بعدی است که ما باید θ بهینه را محاسبه نماییم.

به نظر می رسد که اگر m داده ورودی خروجی^{۵۱} داشته باشیم می توانیم آن ها را به شکل زیر کنار هم بچینیم و یک دستگاه معادلات را نتیجه بگیریم :

$$\left. \begin{aligned} \theta_1 f_1(u_1) + \theta_2 f_2(u_1) + \theta_n f_n(u_1) &= y_1 \\ \theta_1 f_1(u_2) + \theta_2 f_2(u_2) + \theta_n f_n(u_2) &= y_2 \\ \cdot \\ \cdot \\ \theta_1 f_1(u_m) + \theta_2 f_2(u_m) + \theta_n f_n(u_m) &= y_m \end{aligned} \right\} A\theta = y$$

در رابطه فوق داریم:

$$A = \begin{bmatrix} f_1(u_1) & \cdot & \cdot & f_n(u_1) \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ f_1(u_m) & & & f_n(u_m) \end{bmatrix}, \theta = \begin{bmatrix} \theta_1 \\ \cdot \\ \cdot \\ \theta_m \end{bmatrix}, y = \begin{bmatrix} y_1 \\ \cdot \\ \cdot \\ y_m \end{bmatrix}$$

در رابطه فوق اگر تعداد ورودی خروجی ها حداقل به تعداد پارامترها باشد ، محاسبه θ به درستی انجام می شود و می توان از رابطه فوق θ بهینه را محاسبه نمود.

اما رابطه فوق چند مشکل بوجود می آورد.

۱. تعداد ورودی ، خروجی ها معمولا به تعداد پارامترها نیست و بسیار بیشتر از آن است. مثلا برای آب و هوا

ورودی ، خروجی ها ممکن است به ۶۰ سال برسد. این یعنی باید به تعداد 60×365 تا پارامتر داشته باشیم. علاوه بر این ممکن است سیستم غیر تکین^{۵۲} باشد.

۲. ممکن است خطا در مشاهدات وجود داشته باشد به عنوان مثال ممکن است با اجبار کردن یک سیستم

365×60 پارامتری داریم آن را وادار می کنیم به آن شکلی باشد که ما می خواهیم در حالی که ممکن

است در خود مشاهدات و در آن نقاط خطا داشته باشیم.

کلا در مدل سازی سیستم ها *Robustness* ناشی از این است که ما بتوانیم با تعداد پارامترهای کم سیستم را

تقریب بزنیم . بنابر این اگر تعداد پارامترها کم باشد می گوییم سیستم *Robust* است.

اگر قبول کنیم که ممکن است در ورودی ، خروجی ها نویز وجود داشته باشد باید θ های بهینه را بگونه ای تنظیم

⁵¹ measurement

⁵² non singular

کنیم که یک معیار خطا را به حداقل برساند. بنابر این یک تابع انرژی از خطا را به شکل زیر تعریف می کنیم.

$$\theta = A^{-1}y$$

$$e = y - A\theta$$

$$E(\theta) = e^T e = (y - A\theta)^T (y - A\theta) = \sum_{i=1}^m y_i - a_i^T \theta$$

این ساختار حداقل صفر است و کمتر از صفر نمی باشد و فقط زمانی صفر است که $(y - A\theta)$ صفر باشد. اگر الگوریتمی داشته باشیم که $e^T e$ را به حداقل برساند می توانیم امیدوار باشیم θ هایی که بدست می آیند θ های مطلوبی هستند.

تابع انرژی ارائه شده در فوق یک تابع درجه دو از خطا است بنابر این برای بدست آوردن θ کافی است تا از این تابع درجه دو مشتق بگیریم. اگر تابع انرژی فوق را باز کنیم به رابطه زیر می رسیم

$$E(\theta) = (y - A\theta)^T (y - A\theta) = y^T y + \theta^T A^T A \theta - 2\theta^T A^T y$$

و اگر از رابطه فوق مشتق بگیریم خواهیم داشت :

$$\nabla E_{\theta} = 2A^T A \theta - 2A^T y$$

از حل رابطه فوق برای θ بهینه خواهیم داشت.

$$\nabla E_{\theta} = 0 = 2A^T A \theta - 2A^T y$$

$$\hat{\theta} = (A^T A)^{-1} A^T y$$

اگر تعداد ورودی خروجی ها خیلی بیشتر از تعداد پارامترها باشد رابطه فوق θ بهینه را نتیجه خواهد داد. این رابطه در حقیقت نوعی مصالحه انجام می دهد زیرا در حالاتی که تعداد ورودی ها خیلی بیشتر از پارامترها باشد تعداد m معادله وجود خواهد داشت در حالی که تعداد پارامترها تنها n عدد ($n < m$) می باشد بنابراین لزوما نمی توان تمام معادلات را به صفر رساند . روش LSE بسیار خوب است اما از نظر بعد محاسباتی بسیار پیچیده است زیرا ماتریس $e^T e$ از بعد $m \times n$ می باشد که m می تواند بی نهایت بزرگ باشد بنابراین نیاز به حجم محاسباتی زیادی برای بدست آوردن $\hat{\theta}$ دارد .

مشکل LSE و بدست آوردن $\hat{\theta}$ یا θ بهینه حجم محاسباتی سنگین در هر بار اجرا می باشد. برای رفع مشکل فوق از روابط بازگشتی استفاده می شود . بدین ترتیب که به جای انجام تمام محاسبات در هر مرحله از ابتدا ، نتیجه محاسبات قبلی را همراه با محاسبه فعلی بکار می بریم و بنابراین رابطه محاسبه θ به صورت بازگشتی به شکل زیر

بدست می آید.

$$\hat{\theta}_{k+1} = \left(\begin{bmatrix} A \\ a_{K+1}^T \end{bmatrix} \begin{bmatrix} A \\ a_{k+1}^T \end{bmatrix} \right)^{-1} \begin{bmatrix} A \\ a_{K+1}^T \end{bmatrix} \begin{bmatrix} Y \\ y_{k+1} \end{bmatrix}$$

...

$$\hat{\theta}_k = \hat{\theta}_k + P_{k+1} a_{k+1} (y_{k+1} - a_{k+1}^T \hat{\theta}_k)$$

$$P_{k+1} = P_k - \frac{P_k a_{k+1} a_{k+1}^T P_k}{1 + a_{k+1}^T P_k a_{k+1}}$$

در رابطه فوق p تابعی است که هم تراز با $\hat{\theta}$ ، $update$ می شود. بنابراین p_0 و $\hat{\theta}_0$ باید مقداردهی اولیه شوند.

برای محاسبه $\hat{\theta}_0$ می توانیم از روش LSE استفاده کنیم و به تعداد پارامترهای θ ، ورودی - خروجی در نظر بگیریم و توابع را برای بدست آوردن $\hat{\theta}$ حل نماییم نتیجه بدست آمده را می توانیم به عنوان $\hat{\theta}_0$ قرار دهیم و از آن به بعد با رابطه فوق ادامه دهیم. در روش ساده تری نیز می توانیم آن را به صورت تصادفی مقداردهی نماییم.

p در حقیقت نرخ یادگیری^{۵۳} را تنظیم می نماید. هر چقدر p بیشتر باشد در حقیقت نرخ یادگیری افزایش می یابد و از طرفی موجب می شود که بیشتر تابع خطای فعلی باشیم تا چیزی که قبلاً محاسبه نموده ایم. بنابراین منطقی به نظر می رسد که در ابتدا p را خیلی بزرگ در نظر بگیریم (چون اطمینانی به دانسته های فعلی نداریم و مشاهدات زیادی هم نداریم) و به مرور آن را کاهش دهیم (چون مشاهدات زیاد شد و قصد داریم مقاومت نسبت به خطا را افزایش دهیم). p مرتب رو به کاهش است تا زمانی که به نزدیک صفر برسد. بنابراین LSE بازگشتی، دانسته های قبل را با دانسته های جدید تکمیل می نماید تا بتواند جواب مطلوب را بیابد.

۳-۱-۲. روشهای مبتنی بر گرادینان^{۵۴}

روش دیگری برای تنظیم پارامترهای θ که مبتنی بر خطا می باشد متد مبتنی بر گرادینان است. در این روش از تابع خطا مشتق می گیریم و بر حسب نوع مسئله به عنوان مثال اگر مسئله کمینه سازی باشد در جهتی حرکت می کنیم که خلاف جهت افزایش مشتق باشد.

$$\theta_{k+1} = \theta_k - \eta g$$

$$g(\theta) = \nabla E(\theta)$$

η طول گام را تعیین می کند ، اگر η خیلی بزرگ و یا خیلی کوچک باشد الگوریتم دچار مشکل می شود . بطوریکه

⁵³ learning rate

⁵⁴ Gradient Based Methods

اگر خیلی کوچک انتخاب شود، گام ها خیلی کوچک شده و به مقصد رسیدن می تواند خیلی طول بکشد. از طرفی اگر خیلی بزرگ انتخاب شود، گام ها خیلی بزرگ شده و ممکن است سبب واگرا شدن سیستم شود.

بنابراین تخمین گام در روش گرادیان بسیار مهم است و به همین خاطر است که η را در ابتدا بزرگ انتخاب می کنند تا به سرعت به سمت هدف حرکت کند و به نزدیکی آن برسد و سپس آن را کوچک می کنند تا، زمانی که به نزدیکی هدف رسید با دقت بیشتر به سمت هدف حرکت کند. مشکلی که برای این روش وجود دارد این است که خیلی راحت در بهینه محلی^{۵۵} گرفتار می شود.

۳-۱-۳. روش نیوتون

در مقابل روش گرادیان، روش دیگر روش نیوتون است. در این روش تابع هدف را با یک تابع درجه دو تقریب می زنیم. بنابراین برای اینکه بتوانیم این تابع را بدست آوریم ابتدا در آن نقطه از تابع هدف مشتق می گیریم و بعد از آن $Hessian$ می گیریم با داشتن مشتق و $Hessian$ می توانیم در آن نقطه یک تابع درجه دو را تخمین بزنیم. چون خودمان آن را بوجود آورده ایم می دانیم که min و max کجا قرار دارند و سپس از آن نقطه برای شروع $step$ بعدی استفاده می کنیم.

$$E(\theta) \approx E(\theta_k) + g^T(\theta - \theta_k) + \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k)$$

$$0 = g + H(\hat{\theta} - \theta_k)$$

$$\hat{\theta} = \theta_k - H^{-1}g$$

روش نیوتون همچنان مشکل گرفتار شدن در نقطه بهینه محلی را دارد اما می تواند سریع تر به نقطه بهینه برسد. این روش ممکن در ۲ یا ۳ مرحله^{۵۶} به جایی برسد که گرادیان در عرض ۱۰۰۰ مرحله نرسیده است. در این روش فرض بر این است که تابع خطا با یک تابع درجه ۲ قابل تقریب است.

بطور کلی روش هایی که بر اساس مشتق کار می کنند عموماً مشکل $local\ mini$ را دارند.

۳-۲. روش های بهینه سازی که مبتنی بر مشتق نیستند:

این روش ها مستقل از مشتق^{۵۷} هستند و سعی می کنند از تابع هزینه مشتق نگیرند و عموماً در دسته ی روشهای

⁵⁵ local optima

⁵⁶ step

⁵⁷ Derivative free

تصادفی^{۵۸} قرار می گیرند.

چرا از مشتق گرفتن پرهیز می کنیم؟

الف) عمل مشتق گیری نويز را تشديد می کند. بنابراین هرگاه مشتق می گیریم باید قبول کنیم که این سیستم حتی اگر مشتق پذیر باشد کوچک ترین نويز را تشديد می کند.

ب) روش های مبتنی بر مشتق و گرادیان بعد محاسباتی بسیار زیادی دارند و خود مشتق گرفتن ابعاد محاسبات را زیاد می کند.

ج) در برخی مسائل ممکن است محاسبه مشتق امکان پذیر نباشد

د) این روش ها رسیدن به نقاط بهینه عمومی را تنظیم نمی کنند و مشکل گرفتار شدن در نقطه بهینه محلی را دارند

ه) روشهای مبتنی بر مشتق بسیار به شرایط اولیه حساس می باشند

چهار روش بهینه سازی که از مشتق استفاده نمی کنند عبارتند از :

۱- *Random search*

۲- *Downhill simplex search*

۳- *Genetic algorithm*

۴- *simulated annealing*

در ادامه هر یک از روشهای فوق را بررسی می نماییم.

۳-۲-۱. جستجوی تصادفی (*Random search*):

از ویژگی های این روش میتوان سادگی ، طبیعی و شهودی بودن آن را نام برد . روند کار در این الگوریتم بدین صورت است که در ابتدا بطور تصادفی یک نقطه از فضای مسئله را انتخاب نموده و برای رسیدن به یک نقطه جدید یک گام به یک جهت تصادفی بر میدارد یا این نقطه جدید شیب به پایین دارد و یا بالا، اگر به پایین بود تشویق می شود و یک گام دیگر برمی دارد اگر باز هم خوب بود در همان جهت یک گام بلندتر برمی دارد. اما اگر به جایی رسید که هر چقدر به جلو حرکت می کند بالاتر می رود حال جهت عکس آن را امتحان می کند اگر عکسش رو به

⁵⁸ stochastic

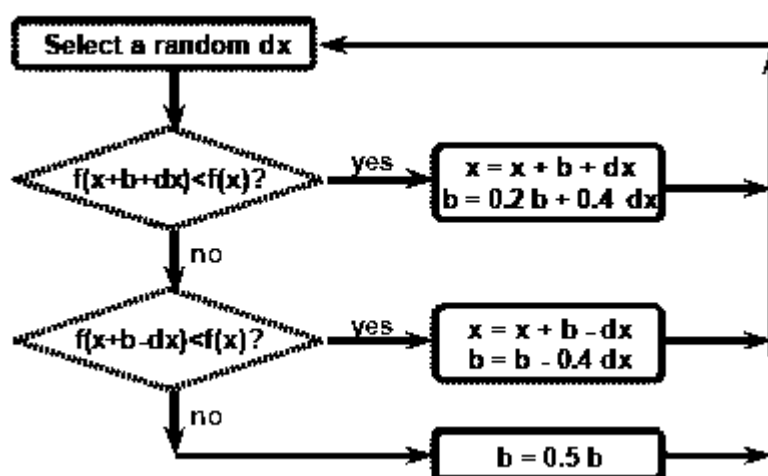
کاهش بود می پذیریم و در آن جهت حرکت می کنیم اما اگر عکس آن هم در حال افزایش بود یک جهت جدید انتخاب می کند و در آن جهت حرکت می کند. این ها *Heuristic* های این روش هستند.

بطور کلی *Random search* دارای دو *heuristic* اصلی بصورت زیر می باشد .

Reverse step : اگر گامی که برمی داریم درست نبود، یک گام جدید در یک جهت جدید انجام می دهد (جهت عکس آن را انتخاب می کند).

Bias direction : اگر گامی که برمی داریم درست بود و باعث کاهش f بود آن را می پذیریم و بایاس را در آن جهت تقویت نموده و گام بعدی را بلندتر در همانجهت برمی داریم.

یک جهت را انتخاب می کند و اگر آن جهت باعث کاهش f بود آن را می پذیرد و بایاس را افزایش می دهد اما اگر آن جهت باعث افزایش f بود آن را رد نموده و جهت عکس آن را انتخاب می کند. اگر این جهت موفق بود در جهت عکس بایاس را افزایش می دهد اما اگر هیچ یک از آنها موفق نبودند آن وقت بایاس را کاهش داده و یک جهت تصادفی دیگر را انتخاب می کند. فلوچارت زیر مراحل این الگوریتم را نمایش می دهد .

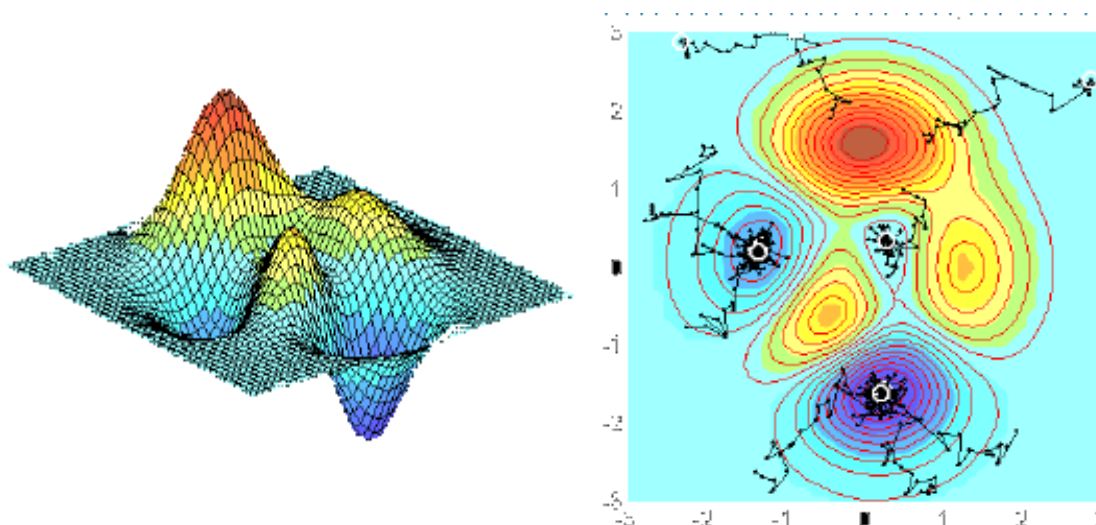


مشکل اصلی روش *Random search* در این است که بسیار به حالت اولیه حساس است و ممکن است به نقطه بهینه محلی همگرا شود.

به عنوان مثالی از این روش می توان تابع زیر را در نظر گرفت که در آن هدف یافتن نقطه مینیمم آن می باشد.

$$f(x, y) = 3(1 - x^2)e^{-x^2 - (y+1)^2} - 20\left(\frac{x}{5} - x^3 - y^5\right)e^{-x^2 - y^2} - e^{-(x+1)^2 - y^2}$$

شکل زیر نمایشی سه بعدی از این تابع و تصویر دو بعدی از اجرا های مختلف الگوریتم را نشان می دهد.

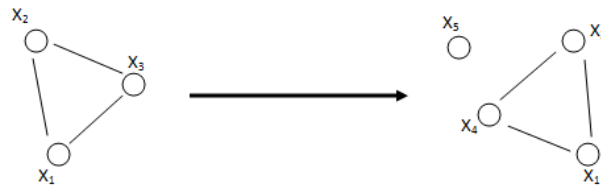


۲-۲-۳. روش سیمپلکس:

این روش کمی با قبلی متفاوت است. در این روش به جای یک عامل بیش از یک عامل فضای مسئله را برای یافتن نقطه بهینه کاوش می نمایند و می توانند در فضای مسئله حرکت کنند. بدین ترتیب که اگر یک فضای n بعدی داشته باشیم $n+1$ نقطه یا عامل را در این فضا در نظر می گیریم. به عنوان مثال در یک فضای دو بعدی به سه نقطه و در یک فضای سه بعدی به ۴ نقطه نیاز داریم. این نقاط در هر فضایی که باشند با کمک یکدیگر یک تپه را تشکیل می دهند.

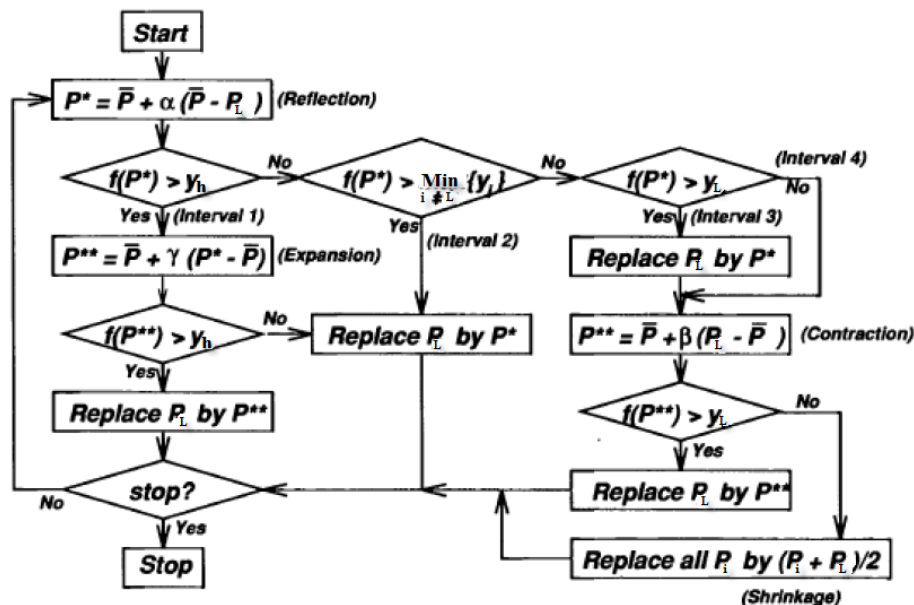
در حقیقت $simple x$ شکل تغییر یافته ای از $Random search$ است چون $Random$ عمل می کند اما یک $Random search$ هدف دار است و از طرفی می تواند شکلی از گرادینت نیز قلمداد شود چون اگر مثلاً سه نفر یا سه نقطه در فضای مسئله باشند و ارتفاع این سه نقطه با هم مقایسه شود، در این صورت تصویری از آن سطح بوجود می آید.

این صورت الگوریتم بطور مثال در یک مسئله مینیمم سازی می تواند به این صورت عمل نماید که همواره آن نقطه ای که مرتفع تر است به یک نقطه پست تر حرکت کند. اگر فرض کنیم فضا دو بعدی است بنابراین سه نقطه داریم که بطور تصادفی در سه نقطه فضا توزیع شده اند و هر کدام از آنها یک ارتفاع دارند و از ارتفاع هم باخبرند.



همانطور که در شکل فوق نشان داده شد، چون x_3 بالاترین ارتفاع را دارد بنابراین باید به یک نقطه پست تر حرکت کند و معقول به نظر می رسد که این نقطه در جهت خلاف موقعیت فعلی اش باشد. یعنی بهتر است برعکس حرکت کند اما اینکه چقدر حرکت کند بستگی به تجربیات گذشته اش دارد. اینکه آیا گام های قبلی که برداشته موفق بوده، اگر موفق بوده گام را بلندتر بدارد اما اگر موفق نبوده گام را کوتاه می نماید.

فلوچارت این روش در زیر ارائه شده است

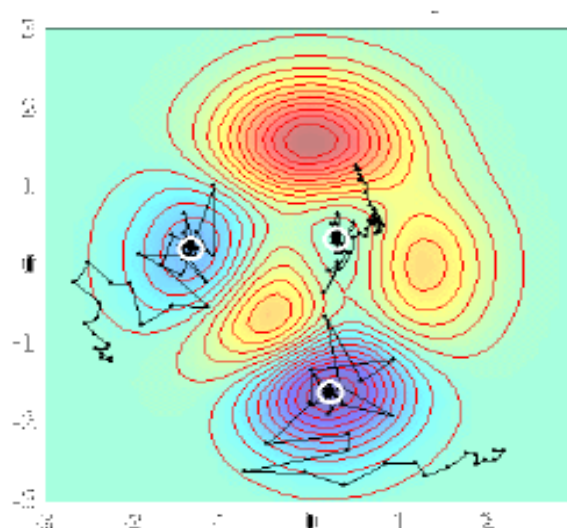


روش *simplex* چند *Hueristic* اصلی دارد که عبارتند از:

۱. بطور مرتب بالاترین نقاط را با نقاط پایین تر جایگزین می کند.
۲. هر چقدر موفقیت ها بیشتر باشد *simplex* نیز بزرگ تر می شود زیرا گام ها بلندتر شده و مثلث بزرگ تر می شود.
۳. هر چقدر عدم موفقیت ها بیشتر باشد *simplex* کوچک تر شده زیرا گام ها کوچک تر می شوند.
۴. مشکل *simplex* این است که می تواند به *local minima* همگرا شود. اما یک مقدار هوشمندانه تر

عمل می کند چون بدون اینکه مشتق بگیرد دارد از مفهوم مشتق استفاده می کند.

شکل زیر نتایج حاصل از اجراهای مختلف این روش را برای تابع ارائه شده در قسمت قبل را نمایش می دهد.



۳-۲-۳. الگوریتم ژنتیک:

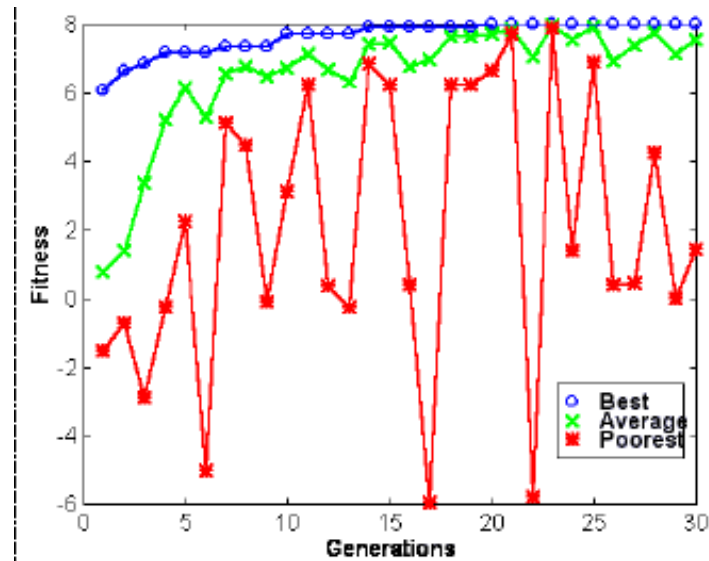
این روش بر خلاف روشهای قبل یک روش *population base* است. یعنی به جای اینکه بر مبنای یک نقطه باشد ، جمعیتی از نقاط را در آن واحد مورد ارزیابی قرار می دهد. از این دیدگاه ممکن است مشابه با روش *Simplex* بنظر برسد در حالی که در روش *simplex* درست است که از سه نقطه استفاده کنیم اما به هر حال باز هم در آن واحد تنها یک نقطه مورد ارزیابی قرار می گیرد و از این جهت نمی توان آن را در دسته روشهای مبتنی بر جمعیت دانست.

مزیت اصلی الگوریتم های ژنتیک موازی پذیری آنها است و این یکی از دلایل مورد توجه قرار گرفتن این الگوریتم ها است. چون در بیشتر الگوریتم های بهینه سازی بخش اصلی پردازش مربوط به محاسبه تابع هزینه است که در این روش می تواند به خوبی موازی سازی شود. اما دلیل عمده محبوبیت آن بدین جهت است که این الگوریتم از طبیعت الگو گرفته شده و حال آنکه دنیای واقعی یک دنیای بسیار نویزی و بسیار پیچیده است و در عین حالی که با این دنیای پیچیده رو به رو هستیم تکامل در طبیعت مزایای بسیار خوبی را به ارمغان آورده است .

پیش از تشریح کامل این الگوریتم و چگونگی عملکرد آن لازم است تا ابتدا برخی لغات و عملگر های کلیدی این الگوریتم را تشریح نماییم.

۳-۲-۳-۱. لغات کلیدی :

۱. *Gene*: کوچک ترین بخشی از یک کروموزوم که مفهوم دارد
 ۲. *Fitness function*: تابع برازندگی
 ۳. *Population*: جمعیت اولیه
 ۴. *Encoding scheme*: مکانیزم تعریف یک صورت مسئله واقعی به فضای *GA*
 ۵. *Selection*: این عملگر در هر نسل با این دیدگاه که کروموزوم هایی که برازندگی بیشتری دارند باید امکان بقای بیشتری برای حضور در نسل های بعدی بیابند کروموزوم ها را برای تولید جمعیت میانی انتخاب می نماید.
 ۶. *Crossover*: این عملگر الگوریتم را قادر می سازد تا جواب های متفاوت را با یکدیگر ترکیب نماید و به آن فرصت را می دهد تا به راه حل های بهینه تر دست یابد. در حقیقت در *crossover* ما به دنبال تعمق بر دانسته ها هستیم.
 ۷. *Mutation*: در این عملگر ما به دنبال تجسس هستیم برای نادانسته ها (بدست آوردن دانش جدید)
 ۸. *Mutation rate*: معمولا احتمال خیلی کمی دارد و مقدار کمی دارند مثلا ۰/۰۵ یا ۰/۰۷ برای *mutation* مقدار معقولی است. اما در *crossover* بدین گونه نیست و می تواند بیشتر باشد مثلا ۰/۷ و یا ۰/۶ برای آن معقول است. یعنی t در زوج انتخاب شده به احتمال ۰/۷ با هم *crossover* انجام می دهند و یا به احتمال ۰/۳ مستقیما به نسل بعد منتقل می شوند.
 ۹. *Elitism*: نخبه گرایی، این عملگر به دنبال حفظ نخبه ها در نسل های بعد می باشد و به دنبال آن است که به نخبگان امکان بقای بیشتری دهد.
- تفاوت *selection* با *elitism* این است که در *selection* امکان بقای بهترین ها افزایش می یابد اما در *elitism* امکان حذف آنهایی که از همه قوی تر هستند را از بین می بریم.
- در *elitism* نخبه مستقیما وارد نسل بعدی می شود و بدین جهت در *selection* و *crossover* شرکت داده نمی شود تا هم از بین نرود و هم بقیه را نیز تحت تاثیر قرار ندهد و موجب همگرایی زودهنگام و غیرواقعی سیستم نشود.
- معمولا برای الگوریتم های ژنتیک جمعیت های اولیه به صورت کاملا *random* انتخاب می شوند. تصویر زیر روند همگرایی و مقدار برازندگی های بهترین ، میانگین و بدترین را در هر نسل نمایش می دهد.



خط آبی : نمودار بهترین ها، مرتبا مسیر صعودی دارد چون ما هیچ وقت نخبه را از دست نمی دهیم مگر اینکه نخبه بهتری را جایگزین آن نماییم. بنابراین در الگوریتم های ژنتیک *fitness* بهترین ها هیچ گاه پایین نمی آید.

خط سبز : متوسط برازندگی^{۵۹} جمعیت را نشان می دهد، مشاهده می شود که وارد به یک مقدار دائم و وضعیت پایدار (*steady state*) همگرا نمی شود و صاف صاف نمی شود و این به دلیل وجود عملگر جهش^{۶۰} است.

خط قرمز : نمودار ضعیف ترین ها، ضعیف ترین ها باید باشند چون ممکن است در آن الگوهایی پیدا شود که در نهایت در کل الگوریتم منجر به موفقیت می شوند.

در نمودار این صفحه اگر نمودار واریانس^{۶۱} نیز نمایش داده شود خوب است. چون اگر واریانس در برازندگی ها زیاد باشد نشان دهنده تنوع در جمعیت می باشد. معمولا در ابتدا واریانس زیاد است و اگر مراقب نباشیم خیلی زود کاهش می یابد. بنابراین حفظ واریانس برای حفظ قابلیت اکتشاف^{۶۲} مهم است تا بتوانیم قابلیت جستجو داشته باشیم و بهتر جستجو کنیم. بنابراین حفظ تنوع^{۶۳} در جمعیت و حفظ بهترین ها خیلی مهم است.

بطور کلی الگوریتم های ژنتیک با توجه به اینکه مبتنی بر جمعیت هستند انتظار می رود با امکان و احتمال بیشتری به نقطه بهینه همگرا شوند.

۴-۲-۳. تبرید تدریجی (*Simulated annealing*):

⁵⁹ fitness

⁶⁰ mutation

⁶¹ variance

⁶² exploration

⁶³ diversity

Annealing: فرآیند خنک سازی در فلزات را گویند. زمانی که فلزات از کوره در می آیند اگر به تدریج خنک شوند مولکول هایشان فرصت پیدا می کنند تا به حالت کریستالی در بیایند و این استحکام فلز را زیاد می نماید

در SA نیز فرآیندی داریم به نام دما که به تدریج کم می شود و فرآیندی داریم به نام پارامتر که همان مولکول ها هستند و این پارامترها به تدریج تغییراتشان کم می شود. هر از گاهی ممکن است دما را افزایش دهیم و بعد دوباره به تدریج آن را کم کنیم، این تا زمانی که پارامترها جای خودشان را پیدا کنند ادامه می یابد. در این الگوریتم هم ابتدا لازم است تا با برخی اصطلاحات کلیدی بکار رفته در آن آشنا شویم:

۱. *objective function*: تابع هدف برخلاف *objective fitness* مجبور نیست حتما ماکزیمم سازی باشد می تواند مینیمم سازی هم باشد.

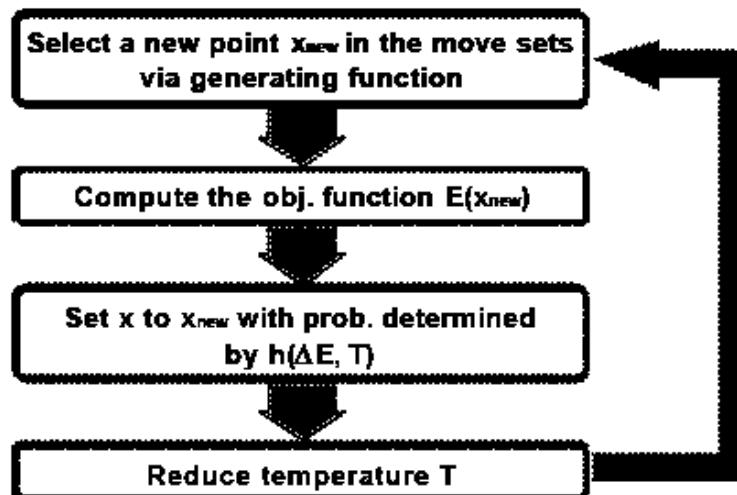
۲. *Move set*: مجموعه مکان هایی که مولکول می تواند به آنها حرکت نماید یا مجموعه مقادیری که آن پارامتر می تواند بپذیرد.

۳. *Generating function*: پیشنهاد مکان بعدی، این پیشنهاد بر اساس مکان فعلی، مکان هایی که وجود دارد و دما می باشد. بنابراین هر چقدر دما بیشتر باشد به الگوریتم امکان می دهد تا فواصل بیشتری را حرکت نماید.

۴. *Acceptance function*: تعیین اینکه آیا این مکان جدید پذیرفته شود و یا خیر، که این تابعی از ΔE (اختلاف انرژی بوجود آمده به خاطر حرکت از نقطه قبلی به نقطه فعلی) و دما می باشد. ممکن است در شرایطی که دما خیلی بالا باشد حاضر به رفتن به مکانی شویم که ΔE آن خیلی مطلوب نباشد چون می خواهیم *Diversity* را افزایش دهیم.

۵. *Annealing schedule*: برنامه زمانبندی که برای سرد کردن دما در نظر گرفته ایم. T را در ابتدا خیلی بالا می گیریم و بعد به تدریج کاهش می دهیم.

تصویر زیر فلوجارت این روش را نمایش می دهد.



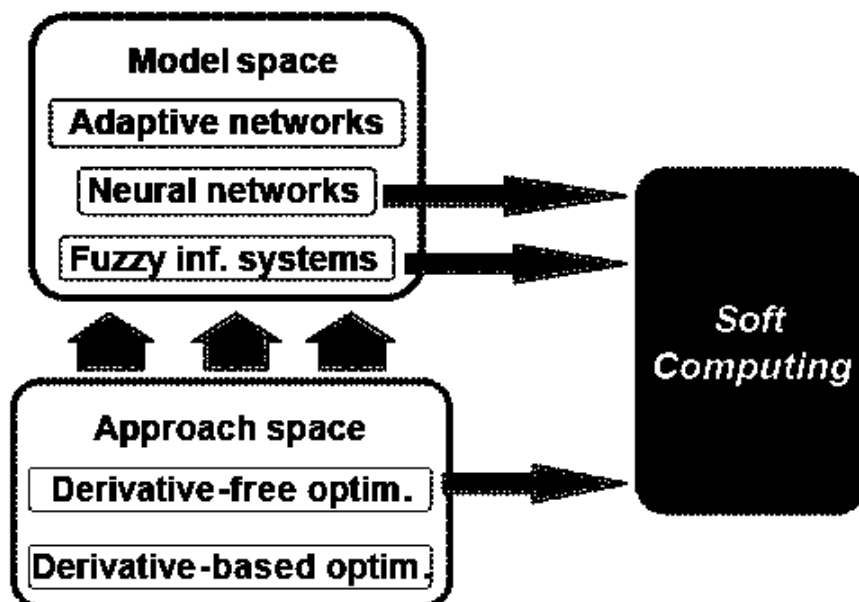
۳-۳. دیدگاه کلی:

آنچه که تا به حال با آنها آشنا شدیم شامل دو پارادایم کلی است.

۱. الگوی دانش: الگوی دانشمندی و حفظ دانش و اطلاعات به عنوان مثال سیستم های فازی

۲. الگوی روش: زاویه ای که نیاز داریم فقط دانش را حفظ نکنیم بلکه نیاز داریم بتوانیم آن را بهینه نماییم

و با شرایط محیطی تطبیق دهیم مانند روش های بر اساس مشتق و روش های مستقل از مشتق.



۴. الگوریتم ژنتیک و برنامه سازی ژنتیک:

۴-۱. الگوریتم های ژنتیک:

از روش های مبتنی بر ژنتیک استفاده می شود چون در مسائل پیچیده خوب عمل نموده و می توانند به جواب بهینه همگرا شوند در حالی که روش های کلاسیک ممکن است در مینیمم های محلی گرفتار شوند، در رابطه با روش های ساده تر مانند *Random search* نیز ممکن است در مینیمم محلی گیر کنید و از طرفی حتی اگر در مینیمم محلی هم گیر نکنند خود محاسبه تابع هزینه بسیار دشوار و زمان بر است. علاوه بر این الگوریتم های ژنتیک این قابلیت را دارند که خیلی خوب موازی سازی شوند.

۴-۱-۱. چرا از *GA* استفاده می کنیم:

سیستم های طبیعی بسیار پیچیده، غیریوسته و نویزی هستند و در این گونه مسائل روش های مبتنی بر *GA* بسیار خوب عمل نموده و می توانند به جواب بهینه همگرا شوند. از طرفی روش های کلاسیک و روش های مبتنی بر مشتق مشکل گرفتار شدن در مینیمم های محلی را دارند و روش های دیگری مانند *Random search* نیز علاوه بر اینکه ممکن است در مینیمم محلی گرفتار شوند، ناکارآمد نیز هستند چون ارزیابی تابع هزینه در این روش ها بسیار زمان بر است و حجم محاسباتی بالایی دارد. بطور کل روش هایی که از مفهوم جمعیت استفاده نمی کنند اینگونه هستند.

الگوریتم های ژنتیک این قابلیت را دارند که موازی سازی شوند و این مسئله به آنها کمک می کند تا در حل مسائل پیچیده بسیار خوب عمل نمایند.

GA : یک الگوریتم بهینه سازی تصادفی^{۶۴} است که از تکامل در طبیعت الگو برداری شده و بدست آمده است

دو هیوریستیک اصلی در *GA* عبارتند از:

۱. *Survival of the fittest*: (بقای بهترین ها) در عملگر *selection* مهم است بدین ترتیب که آن که از

همه قوی تر است فرصت بقای بیشتری می یابد و امکان بیشتری برای جفت گیری دارد.

۲. *Recombination* (جفت گیری): با ترکیب در پاسخ می توان امیدوار بود به پاسخ های بهتر برسیم.

از ویژگی های *GA* این است که کور است و جلوی راهش را نمی بیند و تنها چیزی که نیاز دارد این است که به آن

⁶⁴ stochastic

امکان ارزیابی عملکردش را دهیم چراکه معتقدیم اگر بتواند عملکردش را ارزیابی نماید می تواند راهش را پیدا کند. ویژگی دیگر آن این است که مسئله را ساده می نماید و رشته ای از کدها را برای تصمیم گیری بکار می گیرد که این کدها عموماً به صورت باینری هستند اما انواع دیگری نیز دارند.

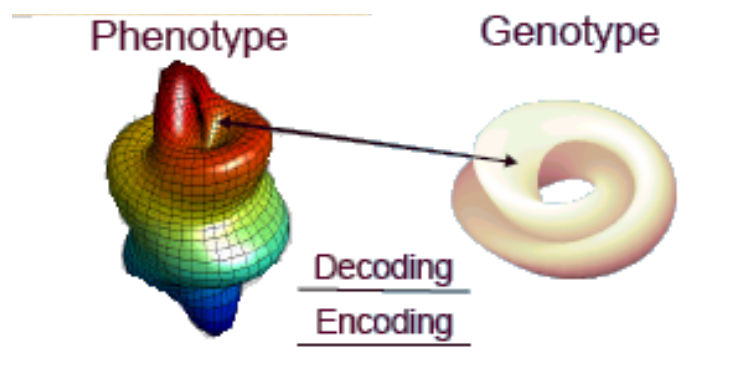
GA از شباهت ها استفاده می کند^{۶۵} (بهره برداری می کند) و همین می تواند دلیل کارآمدی آن نسبت به یک $Random search$ معمولی باشد. بنابر این اگر بخواهیم دلیل کارآمدی آن را بدانیم باید به این $similarity$ ها پرداخته شود که منظور از $similarity$ همان شباهت ها یا الگوها می باشد.

یکی از نکات مهم GA جستجو از طریق یک جمعیت به جای یک نقطه در آن واحد می باشد ، بنابر این هر چقدر جمعیت بزرگ تر باشد و تنوع بیشتر باشد در نتیجه امکان رسیدن به نقطه بهینه افزایش می یابد.

در GA دو عمل اصلی وجود دارد که تحت عنوان تابع تفسیر^{۶۶} از آنها یاد می شود.

Encoding : که موجود واقعی را از صورت پارامترهای مسئله واقعی به آنچه که در GA وجود دارد تبدیل می کند

Decoding : عکس عمل فوق را انجام داده و مسئله را از فضای GA به فضای واقعی می برد.



برخی اصطلاحاتی که در الگوریتم های ژنتیک بکار می روند عبارتند از:

۳. $Phenotype$: نمایش مسئله در فضای واقعی

۴. $Genotype$: نمایش مسئله در فضای GA

^{۶۵} exploit similarity

^{۶۶} Interpretation function

۵. *Evaluation* : (فرآیند ارزیابی) زمانی که یک *individual* از فضای *GA* به فضای واقعی مسئله برده می شود ارزیابی می گردد.

۶. *Chromosome* : یک رشته که نمایش *phenotype* در فضای *GA* است.

۷. *Genes* : کوچک ترین واحد کروموزوم که مفهوم دارد.

۸. *Alleles* : مقدارهایی که یک ژن می تواند بپذیرد.

۹. *Locus* : مکان یک ژن در کروموزوم

۱۰. *Population* : جمعیتی که در هر مقطع از نسلی به نسل بعدی ارتقاء پیدا می کند.

۱۱. *Generation* : فرآیند عبور از یک نسل به نسل بعدی

۴-۱-۲. فرآیند طراحی یک الگوریتم ژنتیک:

۱. انتخاب یک فضای محدود (فضایی که پارامترها می توانند در آن تغییر کنند) و گسسته سازی آن

۲. *fitness function*: طراحی تابع برازندگی. (*interpretation function*)

۳. *initialization of population* : چگونه یک جمعیت اولیه را تولید نماییم.

۴. *Genetic operator*: عملگرهای ژنتیک تعریف می شود.

۵. *stopping criterion* : چه زمانی متوقف شویم.

آنچه که در جمعیت *GA* خیلی مهم است برازندگی آن *individual* نسبت به دیگر کروموزوم ها و یا *individual* ها است و مقدار *fitness* خود کروموزوم چندان مهم نیست. بنابر این ، آن *fitness function* ای بهتر است که بتواند بین جمعیت تمایز بیشتری قائل شود نه اینکه مقدار *fitness* بیشتری نتیجه دهد. در طراحی تابع برازندگی باید توجه داشت که خطا نمی تواند تابع برازندگی باشد و یک تابع هزینه است : برازندگی خطا/۱ می باشد و یا یک مقداری به اضافه آن و یا منفی آن.

در هوش مصنوعی برای تعریف یک *agent* نیاز به تعریف دو مقدار داریم:

۱. تعریف مسئله

۲. تعریف هدف مسئله.

Fitness scaling: جدا کردن، گاهی اوقات جمعیت خیلی به هم نزدیک می شود بطوریکه تمایز قائل شدن بین کروموزوم های جمعیت خیلی سخت است. در این زمان *fitness scaling* جمعیت را می کشد و از هم جدا می کند.

نکته اصلی در *GA* این است که چگونه فضای تابع تفسیر را انتخاب نماییم که بتواند با عملگرهای ژنتیکی که وجود دارند، سریع تر ما را در مسیر بهینه شدن قرار دهند و در کنار آن *fitness function* چگونه انتخاب شود که بتواند بین آنکه بسیار خوب، عالی، ضعیف و نسبتاً ضعیف است تمایز قائل شود، دیگر اینکه چه زمانی الگوریتم را خاتمه دهیم.

الگوریتم های ژنتیک خاصیتی دارند به نام: *epochal behavior* بدین معنا که در ابتدا با یک جمعیت اولیه شروع می نمایند و معمولاً یک مسیر صعودی را طی می کند تا به نقطه ای برسد که به نظر می رسد همگرا شده اند اما اگر مدتی صبر کنیم به یکباره در آن جهشی بوجود می آید و یک پله می زند و باز مدتی افزایش ادامه می یابد و به جلو حرکت می کند. دوباره به نظر می رسد همگرا شده است ولی باز یک جهش می زند و دوباره افزایش می یابد و بنابراین خیلی مهم است که بدانیم کی باید بایستیم.

۴-۱-۳. مقدار دهی اولیه^{۶۷}

ساده ترین بدیهی ترین نوع تابع تفسیر چیدن پارامترهای مسئله کنار هم است. این پارامترها اگر درست کنار هم چیده شوند می توانند بسیار مطلوب باشند. انواع دیگری هم وجود دارند مانند درخت و لیست به عنوان مثال در مسئله *TSP* لیست نسبت به رشته می تواند خیلی خوب باشد.

۴-۱-۴. عملگر انتخاب^{۶۸}

روشهای مختلفی برای انتخاب کروموزوم ها برای شرکت در عمل های *Crossover* و *mutation* و تولید نسل بعد وجود دارد که از جمله معروف ترین این روش ها می توان چرخه رولت^{۶۹} و رتبه بندی^{۷۰} را نام برد.

چرخ رولت: در این روش به هریک از کروموزوم ها بر اساس میزان برازندگی اش امکان بقا برای نسل بعد داده می شود و سپس از بین این کروموزوم ها بطور تصادفی، هر بار دو کروموزوم بصورت تصادفی برای انجام بازترکیب و جهش انتخاب می شوند. مشکل این روش در این است که ممکن است کروموزوم های ضعیف را حذف نموده و یا

⁶⁷ Initialization

⁶⁸ Selection

⁶⁹ Roulette wheel

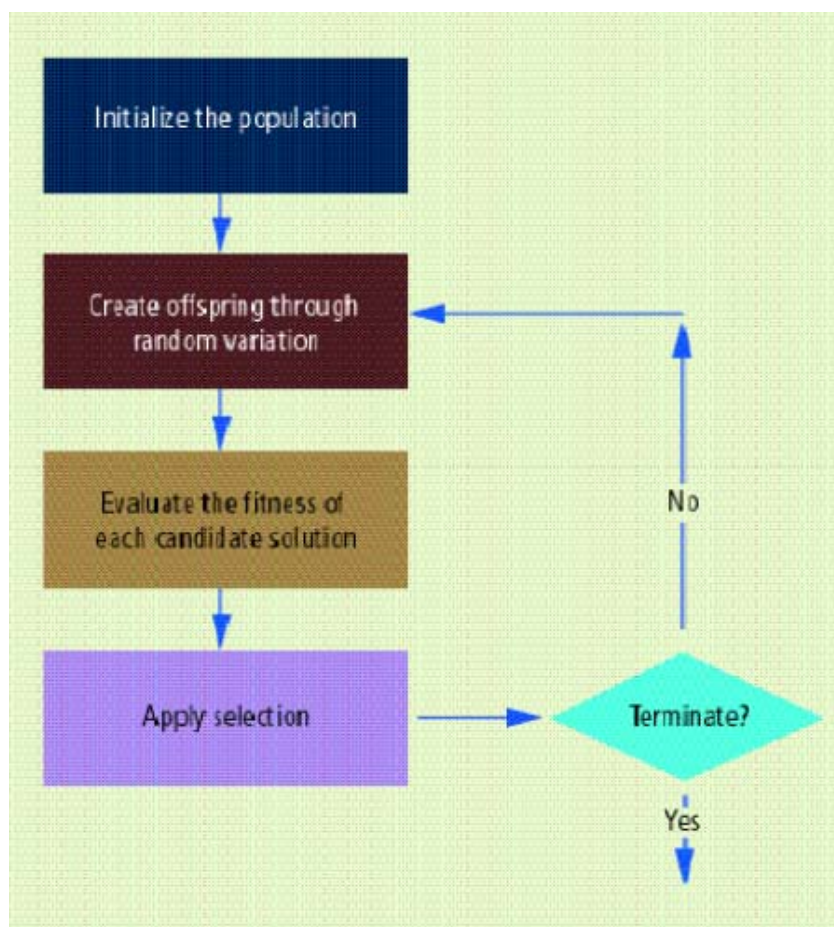
⁷⁰ Fitness ranking

اصلا در نظر نگیرد.

رتبه بندی : در این روش برخلاف روش چرخه رولت به جای اینکه به هر یک از عناصر جمعیت ، بر اساس مقدار نسبی *fitness* اش امکان بقا دهیم ، به هریک از آنها بر اساس میزان برازندگی شان یک رتبه می دهیم و بدین ترتیب آن کروموزومی که ضعیف ترین *fitness* را دارد کمترین رتبه را می گیرد ، اما باز هم امکان بقا دارد هر چند کم. در حالی که در چرخه رولت ممکن است کروموزومی را انتخاب کرده و *fitness* آن را بدست آوریم و مثلاً ۰/۵ باشد و کروموزوم دیگری نیز دارای *fitness* ۰/۰۵ باشد طبق قاعده چرخه رولت، امکان انتخاب شدن کروموزوم یک ۱۰ برابر کروموزوم ۲ است و اگر کروموزوم دو مثلاً ۰/۰۱ باشد احتمال انتخاب کروموزوم یک ۵۰، برابر می شود و دیگر کم کم احتمالی برای انتخاب کروموزوم دو آن باقی نمی ماند.

در چرخه رولت اتفاقی که در نهایت می افتد این است که الگوریتم خیلی زود همگرا می شود و این مطلوب نیست. بنابراین برای جلوگیری از این مسئله یک راه این است که تنوع را در جمعیت حفظ نماییم. راهکاری که به ما این امکان را می دهد استفاده از روش رتبه بندی است.

فلوچارت کلی ژنتیک الگوریتم در زیر نشان داده شده است.



۴-۱-۵. عملگرهای ژنتیک:

۱. *Initialization*: این مرحله از *GA* خیلی کم مورد توجه قرار گرفته چرکه مدعی هستیم *GA* می تواند مسائل پیچیده را با کم ترین دانش در رابطه با مسئله حل نماید. و بدین ترتیب اکثر روش های مبتنی بر ژنتیک جمعیت اولیه را کاملاً *Random* بوجود می آورند و دلیل این عمل را هم بدین صورت عنوان می کنند که زمانی که جمعیت اولیه *Random* باشد *diversity* بالا می رود و بنابراین امکان رسیدن به حل بهینه عمومی افزایش می یابد.

۲. *mutation*: این عملگر بخشی از یک کروموزوم را انتخاب نموده و با احتمالی تحت عنوان احتمال جهش آن را دچار جهش می نماید. این احتمال می تواند شکل های مختلفی داشته باشد. مثلاً اینکه p_m را برای تک تک بیت ها بدست آوریم و یا اینکه p_m را برای کل کروموزوم حساب شود. اما روش معمول روش اول است که برای تک تک بیت ها تعیین می شود بنابراین در یک کروموزوم ممکن است یک بیت، چند بیت و یا حتی ۰ بیت دچار جهش شوند.

۳. *Reproduction*: این عملگر توسط *fitness* تعریف می شود. اونی که برازندگی بالاتری دارد امکان تولید به مثل بیشتری خواهد داشت.

۴. *Crossover*: دارای احتمالی است که این احتمال تعیین می کند برای دو کروموزوم انتخاب شده امکان بقا عیناً به هر یک داده شود و یا اینکه برای تولید نسل جدید تر با هم ترکیب شوند. (توجه شود که این دو به هر حال به نسل بعدی منتقل می شوند، حال یا خودشان و یا فرزندانسان).

نمایشی از نحوه عمل چند عملگر این الگوریتم در زیر ارائه شده است

مقدار دهی اولیه :



$$P(0) = \begin{array}{cc} 0011011 & 0101011 & \text{Individual1} \\ 1100100 & 1001010 & \text{Individual2} \\ 0101011 & 1011111 & \text{Individual3} \\ 1001010 & 1010001 & \\ 0010111 & 0101000 & \end{array}$$

Reproduction و تولید نسل میانی :

Reproduction					
Fitness					
0011011	0101011	71	0101011	1011111	98
1100100	1001010	44	0101011	1011111	98
0101011	1011111	98	0101011	1011111	98
1001010	1010001	16	0011011	0101011	71
1111010	0000011	23	0011011	0101011	71

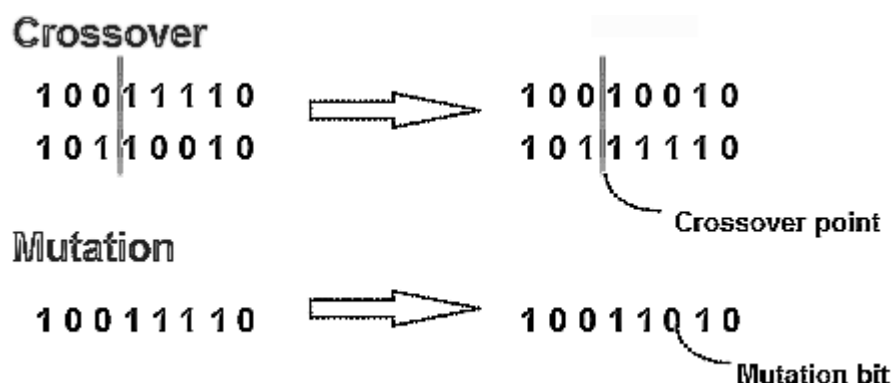
Selection یا انتخاب کروموزوم ها برای شرکت در باز ترکیب و جهش :

Chromosomes should be selected for mating

0101011	1011111	98	→	0101011	1011111	98
0101011	1011111	98		0101011	1011111	98
0101011	1011111	98		0101011	1011111	98
0011011	0101011	71		0011011	0101011	71
0011011	0101011	71		0011011	0101011	71
				0101011	1011011	44

پس از مراحل فوق و انتخاب جفت ها ی دوتایی از کروموزوم ها در این مرحله هر یک از جفت کروموزوم های انتخاب

شده بصورت تصادفی به عملگر *Crossover* داده می شوند در این مرحله بخش هایی از هر دو کروموزوم انتخاب شده و با یکدیگر ترکیب می شوند نمونه ای از چگونگی انجام این عمل بر روی یک کروموزوم باینری در زیر نمایش داده شده است. در انتها پیش از اینکه کروموزوم های بدست آمده توسط عملگر بازترکیب به نسل بعد منتقل شوند ابتدا عملگر جهش بر روی آنها عمل می نماید. بدین صورت که بخشی از کروموزوم را انتخاب نموده و با احتمالی آن را دچار جهش می کند. این عملگر می تواند یک بیت و یا چند بیت از یک کروموزوم را دچار جهش نماید. نمونه ای از چگونگی انجام این روش نیز در زیر نشان داده شده است.



۴-۱-۶. نظریه آجرهای سازنده (*Buildin Block*):

الگوریتم های ژنتیک توسط کنار هم چیدن آجرهای محکم می توانند بنای محکم و موفقی بسازند. به عبارت دیگر توسط کنار هم چیدن *Gene* های مطلوب است که یک انسان برارنده ساخته می شود. در این نظریه عنوان می شود که الگوها یا ژن های ریز، با طول کم و برارندگی بالا امکان بقای بیشتری را در نسل های بعد پیدا می کند. آنچه که *GA* آن را رشد می دهد همان الگوها و آجرهای سازنده هستند.

الگو^{۷۱}: طرحی از شباهت است که زیرمجموعه ای از یک رشته را تعریف می کند. اگر رشته باینری باشد، الگو می تواند از سه پارامتر $\{1, 0, *\}$ تولید شود. به عنوان مثال برای کروموزومی با طول ۵ الگوی *0000 عنوان می کند که تمام موقعیت های آن به جز اولی صفر می باشند

نکته: در یک الگو هر چقدر حالات *don't care* آن بیشتر باشد جمعیت بیشتری را می تواند شامل شود.

بنابر این طبق نظریه اجر های سازنده ، الگوهایی که طول کوتاه و مرتبه کمی دارند به صورت نمایی امکان بقای بیشتری در نسل های بعد پیدا می کنند. این پایه *GA* نیز گفته می شود

⁷¹ Schemata

برای تعریف الگو به شکل فوق دو خصوصیت می تواند تعریف شود:

۱. مرتبه^{۷۲}: مرتبه یک الگو برابر است با تعداد نقاط fix (ثابت) در آن الگو $O(*111x)=3$

۲. طول^{۷۳}: بیشترین فاصله بین نقاط ثابت در یک الگو را شامل می شود $L(*1*1*1)=6$

در یک کروموزوم با طول L و به شکل باینری تعداد (3^L) الگو می توانیم تعریف نماییم.

۴-۱-۷. تئوری بنیادی ژنتیک الگوریتم :

این موضوع همان پایه GA است و تعیین می کند که چرا $crossover$ یا $mutation$ ها را به جواب نزدیک می کند. این مفهوم و تئوری بیان می کند که آن الگوهایی که از برازندگی نسبی بالاتری برخوردارند و مرتبه کمتری دارند و طول کوتاه تری دارند امکان بقای بالاتری دارند در نسل های آینده.

$$m(H, k+1) \geq m(H, k) \frac{f(H)}{\bar{f}} \left[1 - P_c \frac{\delta(H)}{l-1} - P_m O(H) \right]$$

در رابطه فوق داریم :

$M(H, k+1)$: تعدادی که الگوی H در جمعیت $k+1$ ام ظاهر می شود

$F(H)/f$: نسبت برازندگی الگوی H نسبت به متوسط برازندگی کل جمعیت ضرب

$\delta(H)/l-1$: نسبت طول الگوی H به یک منهای طول کروموزوم

P_c : احتمال باز ترکیب

P_m : احتمال جهش

$O(H)$: مرتبه الگوی H



از رابطه فوق چنین نتیجه می شود که هر چه احتمال $mutation$ بالاتر باشد با فرض درست بودن نظریه بلوک های سازنده عبارت داخل آکولاد کوچک تر می شود و بنابراین رشد m کاهش می یابد. هر چقدر مرتبه H هم کوچک باشد باز عبارت داخل آکولاد کوچک تر شده و رشد m کاهش می یابد. این بدین معناست که هر قدر تعداد نقاط ثابت در H بیشتر باشد (مرتبه H بزرگ تر باشد) امکان بقای آن در نسل بعد کمتر می شود. این موضوع برای طول نیز وجود دارد بدین ترتیب که هر قدر طول H بزرگتر باشد و یا احتمال $crossover$ افزایش یابد امکان بقای

⁷² order

⁷³ length

آن الگو در نسل بعد کاهش می یابد. پس نشان داده می شود الگوهایی که طول و مرتبه کمتری و برازندگی بالاتری دارند، در نسل های بعد امکان بقای بیشتری می یابند.

GA یک روش بهینه سازی موازی است. در الگوریتم های ژنتیک نشان داده می شود که به ازای هر n بار که تابع هزینه ارزیابی می شود در حقیقت (n^3) الگو پردازش می شود. به عنوان مثال اگر ۱۰ بار یک تابع هزینه را ارزیابی کرده باشیم مقدار اطلاعاتی که در این مدت پردازش شده ۱۰۰۰ است و این دلیلی بر کارآمدی *GA* است که نشان می دهد به ازای ارزیابی تعداد کمی از توابع^{۷۴} تا چه حد اطلاعات زیاد و مختلفی در اختیار آن قرار می گیرد. این در حالی است که می دانیم آنچه در الگوریتم های ژنتیک و دیگر روش های *stochastic* زمان بر است همین ارزیابی تابع هزینه می باشد. تابع هزینه چیزی است که باید با استفاده از آنچه که در اختیار داریم آن را کاهش دهیم. به عنوان مثال الگوریتم های ژنتیک از الگوها بدین منظور استفاده می نمایند. *GA* الگوها را بارزتر نموده و از آنها بهره برداری می کند. الگوریتم ژنتیک تمایل دارد به الگوهایی که مرتبه و طول کمتری دارند و در کروموزوم هایی با برازندگی بالا ظاهر شده اند در نسل های بعد امکان بقای بیشتری دهد.

Simple GA زمانی کارآمد است که از کنار هم چیدن جزء های با ارزش بالا بتوان به حل بهینه رسید در این صورت است که قانون *Building Block* صادق بوده و رابطه *fundamental theorem* وجود خواهد داشت. به مسائلی که شرایط فوق را نداشته باشد اصطلاحاً *GA Deception* گفته می شود (یا فریب *GA*). اینگونه مسائل *GA* را فریب می دهند چون *GA* تلاش می کند اجزای خوب مسئله را کنار هم بچیند تا مسئله را حل کند اما به جواب نمی رسد چون مسئله این شرایط را ندارد.

GA یک روش بهینه سازی *stochastic* است و به هر حال هر چند طولانی بالاخره می تواند به جواب برسد. اما صرف اینکه این روش از طبیعت الهام گرفته شده و طبیعت با این روش توانسته مسائل پیچیده را حل نماید نمی توان ادعا نمود که این روش قادر است تمامی مسائل را حل نماید چون اولاً خود موضوعی که ما شبیه سازی می نماییم واقعاً آن چیزی نیست که در طبیعت اتفاق می افتد. و ثانیاً ما دو توانایی، زمان بی نهایت و منابع بی نهایت که در طبیعت وجود دارد را در اختیار نداریم

۴-۱-۸. نمایش دوگانه (*Dual Representation*):

نمایش دوگانه یا چند گانه، این موضوع یکی از مسائلی است که الگوریتم های ژنتیک با آنها مواجه می شود. تعبیر دیگری برای آن *competing conventions* (یا قراردادهایی که با هم رقابت می کنند) می باشد. این مسئله

⁷⁴ function evaluation

زمانی بوجود می آید که در فضای ژنتیک^{۷۵} کروموزوم^{۷۶} هایی داریم که با هم متفاوت هستند ولی وقتی اینها را به فضای واقعی^{۷۷} نگاشت می کنیم یک موجودیت منحصر به فرد را نتیجه می دهند.

یعنی نگاشت ما یک نگاشت چند به یک است. به عبارت دیگر اگر در فضای *genospace* دو *individual* متفاوت داشته باشیم که دارای برازندگی بیشتری می باشند ولی به خاطر اینکه اینها در دو *convention* یا *dual representation* متفاوت قرار دارند عملگر باز ترکیب (*crossover*) یک نتیجه کاملاً نامناسب را نتیجه می دهد. یعنی ممکن است دو کروموزوم با برازندگی بالای انتخاب شده از *genospace* فعلی در فضای *phenospace* به یک *individual* نگاشت شوند در این صورت اجرای عمل *crossover* بین این دو نتیجه مطلوبی نخواهد داشت. بنابراین تا حد ممکن باید سعی کنیم از این موضوع پرهیز نماییم. به عبارتی این *multi representation* به نوعی ایجاد تپه های مجازی در مسئله می نماید.

موضوع فوق ناشی از نگاشت چند به یک است و ما یا باید این مسئله را حذف و یا خودمان را برای آن آماده نماییم. مشکل *Dual Representation* در نتیجه ساده سازی نیز بوجود می آید و ما نمی توانیم از آن در *GA* صرف نظر نماییم. چون مسائل موجود و واقعی بسیار پیچیده هستند و در این مسائل افزونگی^{۷۸} بسیار زیاد است.

این موضوع در سیستم های فازی نیز وجود دارد. در این سیستم ها دانش خود به دو قسمت دانش ناخودآگاه^{۷۹} و خود آگاه^{۸۰} تقسیم می شود و خود دانش خودآگاه نیز خود به دو قسمت توابع عضویت و قوانین تقسیم می شود خیلی ساده می تواند نشان داده شود که دو مجموعه *mf*^{۸۱} متفاوت و دو مجموعه قوانین متفاوت کاملاً به یک نگاشت غیرخطی نگاشت پیدا می کنند (یک سطح عیناً یکی را تولید می کنند یا نتیجه می گیرند). بنابراین سیستم های فازی نیز می دانیم که این مشکل *multi representation* را دارند.

۴-۱-۹. انواع مختلف الگوریتم های *GA* :

۱. *Elite GA* : یا *GA* نخبه گرا، الگوریتمی است که تاکنون توضیح داده شد به همراه توانایی و عملگر

Elitism

۲. *Simple GA* : مانند *Elite* است با این تفاوت که دیگر عملگر *Elitism* را ندارد.

⁷⁵ geno space

⁷⁶ individual

⁷⁷ pheno space

⁷⁸ Redundancy

⁷⁹ meta knowledge

⁸⁰ Domain knowledge

⁸¹ Membership function

۳. *Steady state GA* : در این الگوریتم جمعیت میانی وجود ندارد. بنابراین مقدار زیادی از حافظه مورد نیاز کاسته خواهد شد. در این حالت عملگر *selection* عملی است که بر روی همان نسل اولیه انجام می شود. بنابراین در هر نسل یک جفت از کروموزوم ها انتخاب شده و ممکن است به احتمالی *crossover* انجام دهند و یا انجام ندهند و سپس به همان جمعیت برگردانده می شوند. علاوه بر این در این الگوریتم می توان مفهوم مسن شدن را نیز گنجانند. بدین ترتیب که برخی از *individual* ها که انتخاب نمی شوند کم کم یا حذف شده و یا *fitness* آنها مقداری کم می شود. بنابراین با مفهوم *steady state* ، هم به حافظه کمتری نیاز داریم و هم می توان در آن مفهوم مسن شدن^{۸۲} بگنجانیم. در این روش یک کروموزوم بعد از اینکه چند بار فرصت بدان داریم اگر انتخاب نشد از بین می رود. از طرفی به دلیل بوجود نیامدن نسل مبانی نیاز به فضای کمتری دارد.

۴. *Incremental GA* : تفاوت اش با روش های قبلی در این است که در هر آن فقط یک *individual* را ارزیابی می کند. این می تواند هم یک محدودیت و هم یک مزیت برای آن تلقی شود. گاهی اوقات در هر زمان مجبوریم که در هر آن فقط یک *individual* را مورد ارزیابی قرار دهیم چون فقط یک *platform* آزمایش داریم مثلاً برای سیستم های *Real time*. طبیعتاً این روش می تواند ساده باشد و به فضا و حافظه کمی نیاز دارد و از لحاظ محاسباتی می تواند بهتر باشد. اما از آنجایی که در هر آن *search* موازی را به یک *search* که در آن واحد فقط یک نقطه جدید اطلاعاتی دارد تغییر داده ایم ، احتمال همگرایی الگوریتم به راه حل بهینه محلی افزایش می یابد.

۵. *Parallel GA* : *GA* ذاتاً موازی و موازی پذیر است اما اگر به صورت موازی هم اجرا شود به آن *parallel GA* می گویند این دسته از الگوریتم ها در هر آن یک تعداد از زیرجمعیت ها را حفظ و ارزیابی می کنند و از اینرو برای مسائل *multi model* بسیار مناسب هستند. چون تپه ها را شناسایی می کنند و این امکان را می دهند که *individual* هایی که در اطراف آن تپه هستند با هم عمل بازترکیب را انجام دهند و بنابراین مسائل *multi modality* در این حالت حل خواهند شد. *Niched GA* یک نمونه از این گونه الگوریتم ها می باشد این الگوریتم ها خیلی خوب موازی سازی می شوند و بر روی پرسورهای متفاوت اجرا می شوند و بنابراین هر پرسور مسئول یک *Niche* می شود که هر کدام از آنها یک *sub population* دارد و در هر آن و هر از گاهی امکان مهاجرت از یک *Niche* یا یک *sub*

⁸² ageing

migration GA به دیگری داده می شود (برای یک *individual*). به این الگوریتم *migration GA*

گفته می شود. یعنی *GA* ای که برای یک *individual* امکان مهاجرت از یک تپه به تپه دیگر وجود

دارد. اما عمل مهاجرت خیلی کمتر اتفاق می افتد و اصلا بدین دلیل این امکان پیش بینی شده چون

ممکن است بین دو تپه یک تپه دیگری نیز وجود داشته باشد.

در حقیقت مسئله در روش های *GA, stochastic* آنقدر پیچیده است که ما هیچ دیدی نسبت به آن نداریم و

نسبت به آن نابینا هستیم و با این الگوریتم ها سعی می کنیم بینش خود را افزایش دهیم. سوال کلا در رابطه با این

الگوریتم ها این است که با این اطلاعاتی که در حال حاضر موجود است چگونه می توانیم هوشمندانه گام بعدی را

انتخاب نماییم. بنابراین در *Niche* با فرض اینکه می دانیم مسئله *multi modal* است و به خاطر اینکه به مسئله

competing convention گرفتار نشویم این امکان را در نظر می گیریم که تپه هایی داشته باشیم و هر تپه برای

خود زیر جمعیتی داشته باشد.

بطور کلی در این حالت مجموعه کروموزوم ها را بر اساس معیارهایی که به عنوان مثال می تواند *fitness* باشد در

کلاسترهایی دسته بندی نماییم و *crossover* را در میان کروموزوم های قرار گرفته در یک خوشه انجام دهیم با

این امید که الگوهای مشترک می توانند بدین ترتیب پررنگ تر شوند. به عنوان مثال برای نقاطی که روی یک تپه

قرار دارند امیدواریم انجام *crossover* در این کروموزوم ها ما را به قله برساند از طرفی می توانیم *crossover* را

بین خوشه های متفاوت انجام دهیم با این امید که بتوانیم به راه حل های بهینه نیز دست یابیم و دچار مینیم

های محلی نشویم. به عنوان مثال زمانی که چند *min/max* وجود دارد.

۶. *GA: Meta level GA* روی یک *GA* دیگر، *GA* ای است که پارامترهای *GA* را بهینه می نماید. این

پارامترها می توانند احتمال *mutation*، احتمال *crossover* یا هر چیز دیگر باشد. این روش یک مقدار

ثابت و معمولا پارامتر بهینه (مثلا احتمال *crossover* بهینه) را می دهد

۷. *Fuzzy GA*: نوعی از *GA* است که در آن یک *fuzzy* سیستم پارامترهای *GA* را تنظیم می کند. مثلا

اگر احساس می کنیم در جمعیت *diversity* کم شده است و الگوریتم دارد به جایی همگرا می شود که

زیاد مطمئن نیستیم بهینه باشد عاقلانه این است که احتمال جهش را افزایش دهیم تا *diversity*

افزایش یابد و برعکس اگر حس کنیم الگوریتم *GA* رفتار و جهت خاصی ندارد بهتر است احتمال

mutation را کاهش داده و *crossover* را افزایش دهیم. به عبارت دیگر از الگوریتم *GA* بخواهیم با

عمل *crossover* به دنبال *exploit* کردن دانسته ها باشد به جای اینکه نقاط جدید نادانسته را کشف

کند. *Fuzzy GA* یک نوع *adaptive GA* است یعنی پارامترهای *GA* را در طول مسیر بهینه سازی با توجه به آنچه که مشاهده می نماید بهینه می کند. مثلاً نسبت به توزیع جمعیت چقدر بهینه است، چقدر *diversity* وجود دارد. بنابراین گاهی اوقات ممکن است p_m را زیاد کند و یا کم کند و این اعمال را بر اساس دانش انسانی انجام می دهد.

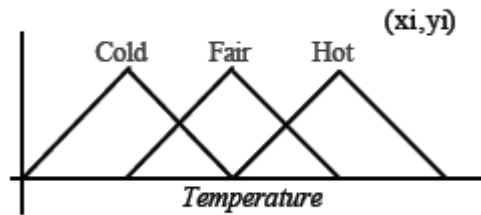
مثلاً کاری که برای تنظیم p_m انجام می دهند چون در ابتدا جمعیت اولیه به صورت کاملاً تصادفی تولید می شود و بنابراین *Random search* در آن فراوان است p_m مقدار کمی دارد. این روند به عنوان مثال تا نسل ۵۰ ام ادامه می یابد و تا این زمان الگوریتم تقریباً همگرا می شود و در این زمان یکباره به *mutation* یک پالس می دهند و در این حالت غیر از *Elite* ها دیگر کروموزوم ها بهم ریخته و در حقیقت الگوها را از هم پاشیده می شوند. در این زمان الگوریتم دوباره شروع می کند به بهینه شدن و دوباره همگرا می شود. سپس دوباره یک پالس به *mutation* داده می شود و این روند آنقدر ادامه می یابد تا به نظرمان به حل بهینه دست یابد.

یکی از ایراداتی که به *GA* از ابتدا وارد بود این است که به حل های نامعقول و غیریابوری دست می یابد.

۴-۱-۱۰. مثالی از کاربرد *Fuzzy GA*

در طراحی یک کنترلر، خروجی ها که به عنوان مثال می تواند زوایا برای بازوی یک ربات باشند، باید خروجی های مطلوب یا زوایای مطلوب را دنبال کنند. بنابراین کنترلر فازی آنچه که اتفاق می افتد را با آنچه که واقعیت است مقایسه می کند و ورودی سیستم را یا گشتاورها را به گونه ای انتخاب می کند که آنچه اتفاق می افتد بر واقعیت منطبق باشد. اما تنظیم پارامترها برای این کنترلر بسیار دشوار است حتی با اینکه این کنترلر، بر اساس دانش انسانی کار می کند اما تنظیم دقیق پارامترهای عضویت باز هم بسیار دشوار است. قواعد را می شود تنظیم کرد اما توابع عضویت دشوار است.

به عنوان مثال در تعریف توابع عضویت اگر دما را در نظر بگیریم در یک مثال به خصوص می توان سه بازه برای دماهای مختلف تعریف کرد و بر روی این بازه ها از توابع تعلق مثلثی استفاده نمود. بنابراین سه تابع عضویت بر روی این بازه ها تعریف می شود و با توجه به اینکه توابع عضویت مثلثی هستند برای هر یک از آنها نیاز به سه نقطه داریم، بنابراین در کل ۹ نقطه وجود خواهد داشت که هر نقطه توسط دو پارامتر x, y مشخص می شود.



از کنار هم قرار دادن این مختصات ها و نقاط یک کروموزوم به شکل زیر تولید می شود.

|x1|y1|x2|y2|x3|y3|x4|y4|x5|y5|x6|y6|x7|y7|x8|y8|x9|y9|

کروموزوم طراحی شده در این حالت می تواند توابع عضویت را بهینه نماید. اما ما همیشه مجبور نیستیم صرفاً یک کروموزوم را با یک رشته تعریف نماییم. گاهی اوقات ممکن است یک ساختار داشته باشیم مثل یک *Rule* اگر در سیستم های فازی توابع عضویت را ثابت بگیریم و بخواهیم قوانین را بهینه نماییم. در این شرایط بهتر است که از یک ساختار دو بعدی مثل ماتریس استفاده نماییم. در این حالت برای *crossover* یک نقطه و یک حاشیه اطراف آن انتخاب می شود و آنچه را که در این محدوده قرار می گیرد با زوج دیگر آن مبادله می شود.

Parents

Offspring

p_{11}	p_{12}	p_{13}	p_{14}	p_{15}
p_{21}	p_{22}	p_{23}	p_{24}	p_{25}
p_{31}	p_{32}	p_{33}	p_{34}	p_{35}
p_{41}	p_{42}	p_{43}	p_{44}	p_{45}
p_{51}	p_{52}	p_{53}	p_{54}	p_{55}

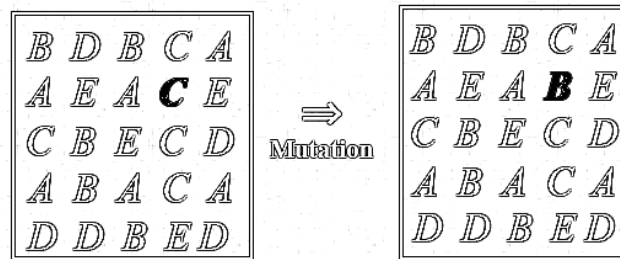
p_{11}	p_{12}	p_{13}	p_{14}	p_{15}
q_{21}	q_{22}	q_{23}	q_{24}	q_{25}
q_{31}	q_{32}	q_{33}	q_{34}	q_{35}
p_{41}	q_{42}	p_{43}	p_{44}	p_{45}
p_{51}	p_{52}	p_{53}	p_{54}	p_{55}

⇒
Crossover

q_{11}	q_{12}	q_{13}	q_{14}	q_{15}
q_{21}	q_{22}	q_{23}	q_{24}	q_{25}
q_{31}	q_{32}	q_{33}	q_{34}	q_{35}
q_{41}	q_{42}	q_{43}	q_{44}	q_{45}
q_{51}	q_{52}	q_{53}	q_{54}	q_{55}

q_{11}	q_{12}	q_{13}	q_{14}	q_{15}
q_{21}	p_{22}	q_{23}	q_{24}	q_{25}
p_{31}	p_{32}	p_{33}	q_{34}	q_{35}
q_{41}	p_{42}	q_{43}	q_{44}	q_{45}
q_{51}	q_{52}	q_{53}	q_{54}	q_{55}

برای *mutation* نیز به همین ترتیب است و یک خانه از این ماتریس به احتمالی می تواند دچار جهش شود



تمامی انواع GA موازی پذیر هستند جز $incremented GA$ ، از این جهت به سایر روش ها $parallel GA$ نمی گویند مگر اینکه حتما به صورت موازی بکار گرفته شوند و پیاده سازی شوند. ذاتا موازی بودن GA به دلیل استفاده از مفهوم جمعیت و بر اساس جمعیت بودن آن است.

$Fuzzy GA$: ماهیت آن GA است و از فازی برای بهینه سازی استفاده می کند.

$GA Fuzzy$: ماهیت آن فازی است و از GA برای بهینه سازی استفاده می کند.

$Scaling$ در GA : بازه ای از تابع $fitness$ را که جواب در آن قرار دارد می کشد تا تفاوت میان بهترین جواب و جواب های دیگر بیشتر شود و یا ممکن است تابع را فشرده نماید تا تفاوت بیش از حد را حذف کند و از بین رفتن بقیه جلوگیری کند. در جایی از فشردن استفاده می شود که بهبود یکی باعث حذف و از بین رفتن نشود و تنوع از بین رود.

۴-۲. Genetic programming

چرا به برنامه سازی ژنتیک می پردازیم؟

از زمانی که $programming$ شروع شد، رشد و توسعه و تولید $program$ ها تاکنون به صورت نمایی رو به افزایش بوده است به حدی که اگر امروز برنامه ساز یا برنامه نویس ها یک نرم افزار را تولید نماید. حتی قبل از اینکه آن نرم افزار به دست مصرف کننده و یا کاربر برسد می توان گفت از نظر تکنولوژیکی دیگر معتبر نیست و تاریخ مصرف آن گذشته است و می توان گفت نرم افزارها به این سرعت در حال تغییر هستند.

در گذشته مکانیزم هایی برای برخورد با این رشد اختراع شدند تا بتوانیم سریع تر برنامه سازی نموده و به تولید نرم

افزار بپردازیم . برای مثال برنامه نویسی شی گرا^{۸۳} یکی از این نمونه ها است. مکانیزم دیگری که در اختیار است برنامه نویسی ساخت یافته^{۸۴} می باشد و نوع دیگر نیز *object libraries* می باشند که نمونه ای از این مکانیزم زبان های ویژوال (*visual*) را می توان نام برد، مکانیزم دیگری هم وجود دارد به نام *Rapid Proto typing* و همه اینها این امکان را دارند تا مهندسين کامپیوتر بتوانند سریع تر نرم افزار را تولید نمایند.

اما باز هم مشکل حل نشده است یعنی به همان میزان که توانایی ما برای تولید کدهای بیشتر و طولانی تر و توانا تر افزایش پیدا کرده است از آن طرف انتظار ما برای آنچه که نرم افزار باید انجام دهد هم افزایش یافته است. بنابراین در محیط نرم افزاری هم ، مطلوب به نظر می رسد که بتوانیم نرم افزاری بنویسیم تا بتواند خودش را برنامه نویسی کند. این موضوع در حقیقت آن *methodation* است که پشت *Genetic programming* قرار دارد.

یک برنامه نرم افزاری که بتواند از خودش با استفاده از اصول ژنتیک برنامه بنویسد.

تفاوتی که *Genetic programming* با روش های قبل از آن دارد در همین موضوع است که روش های قبلی مثل *object programming* و غیره اگرچه در راستای افزایش سرعت برنامه نویسی بودند ولی آن محور اصلی را که انسان باشد از این حلقه تولید بر نداشته بودند که در *Genetic programming* ما به دنبال این موضوع هستیم. یعنی برنامه ای را بنویسیم و راهکاری را بیاندیشیم که بدون نیاز به انسان بتواند به صورت هوشمندانه نرم افزار بنویسد.

Genetic programming صرفا به برنامه نویسی منحصر نمی شود و بعد کاری آن بسیار فراتر از برنامه نویسی می باشد. از دیدگاه نرم افزاری هدف ما در *Genetic programming* این است که بتوانیم کدی را بنویسیم که قبل از اینکه غیرقابل استفاده^{۸۵} شود بتواند خودش را *update* نماید. در حال حاضر شرکت *Microsoft* از طریق *internet* و *update* کردن نرم افزارهای ساخت این شرکت این عمل را انجام می دهد.

مسئله برنامه نویسی یا برنامه سازی ژنتیک مسئله جدیدی نیست به عنوان مثال

- در سال ۱۹۶۳ کسی به نام *Samuel* به این مسئله پرداخت او *machine learning* را به این ترتیب تعریف می کند که یادگیری ماشین یعنی: کامپیوترها بتوانند خودشان را برنامه سازی یا برنامه ریزی کنند^{۸۶}

- در سال ۱۹۹۶، *Michel*، یادگیری ماشین را بدین ترتیب تعریف می کند. برنامه های کامپیوتری که بطور

⁸³ object oriented programming

⁸⁴ structured programing

⁸⁵ obsolete

⁸⁶ computer programming themselves

خودکار و از طریق تجربه می توانند خودشان را بسازند.

Genetic programming چیست؟ برنامه یادگیری است که تاکیدش روی تکامل ساختارهای درختی است به عبارت دیگر اغلب روش های بهینه سازی که تاکنون به آن پرداختیم و یا وجود دارد به بهینه سازی پارامترها می پردازد. اما *Genetic programming* به بهینه سازی ساختار می پردازد.

- بنابراین تفاوت اصلی میان *GA* و *GP* در این است که در *GP* این ساختار و در حقیقت این کنش ها و واکنش های بین *Object* هاست که دارد بهینه می شود. اما در *GA* این پارامترها هستند که دارند بهینه می شوند. ما در *GP* پارامتر هم داریم ولی این پارامتر نیست که مورد تاکید *GP* است و آنچه که *GP* بر روی آن تاکید می کند بهینه سازی ساختار است و اینکه کدام *object* با کدام *object* در تعامل باشد و چگونه تعامل نماید. بنابراین *GP* در حقیقت پاسخ *koza* است به مسئله و پرسشی که *Samuel* مطرح کرد اینکه بتوانیم به کامپیوترها بیاموزیم که چگونه خودشان را برنامه سازی کنند.

- مسئله *program induction* از دیرباز وجود داشته در حقیقت برای مثال *Friedberg* سعی کرد یک مسئله ۶۴ بیتی را از طریق *Random mutation* حل کند و توانست با امکانات آن زمان صرفا با عمل *mutation* یک ترکیب ۶۴ بیتی را پیدا کند (*64 combination*) که مسئله مورد نظرش را بهینه نماید و در نهایت چنین عنوان می کند که :

«اگر ما امید داشته باشیم که بتوانیم ماشینی بسازیم که بتواند صحبت کند یا بفهمد، زبان های انسانی را ترجمه کند و مسائل ریاضی را حل کند که مبتنی بر تخیل باشند و یا اینکه بتواند یک تخصص را که نیاز به دانش خاص و یا تصمیم گیری دارد را بهینه و اداره نماید بسازیم ، یا می بایست بتوانیم آنها را به آنچنان علم دقیقی ساده نماییم (یعنی حاشیه ها را برداریم و آنچه را که دقیقا اصل است را برداریم) که بتوانیم آن را به شکلی که از بیرون مشاهده نشود که به سیستم فقط یک سری اگر آنگاه داده ایم به خورد ماشین دهیم ، که این در حال حاضر یک هدف نیافتنی است و یا اینکه بایستی ماشینی ساخته شود که بدون اینکه به آن بگوییم چگونه بتواند عمل را انجام دهد. در مسئله فوق در حقیقت دو نگرش اصلی در هوش مصنوعی هستند.

نگرش اول *expert system* است. نگرش دوم *machine learning* است و در همین دیدگاه است که این دو موضوع از هم جدا می شوند. در ابتدا این *expert system* ها بودند که بکار گرفته می شوند و رواج یافتند و از جمله آنها می توان *PROSPECTOR* ، *DESIGN ADVESOR* ، *MOLE* و *MYCIN* را نام برد اما این سیستم ها در مقابل نادانسته ها بسیار شکننده می باشند و فقط بر اساس یک سری قواعد اگر آنگاه که از خبره

گرفته می شوند عمل می کنند و بنابراین در حالاتی که دانشی برای آن ندارند دچار شکست می شوند و در این زمان است که ما به *learning* نیاز پیدا می کنیم.

Learning چه فرآیندی است؟

- *learning* معمولاً با یک *Domain* شروع می شود و در اینجا صورت مسئله مشخص می شود.

- پس از انتخاب *Domain* باید یکسری داده ورودی و خروجی از مسئله وجود داشته باشد. معمولاً مسائلی که ما در حوزه هوش مصنوعی به آنها می پردازیم مسائلی هستند که مدل ریاضی دقیقی برایشان موجود نیست. بنابراین برای مدل سازی مسئله نیازمند مجموعه ای از ورودی ها و خروجی های آن مسئله می باشیم.

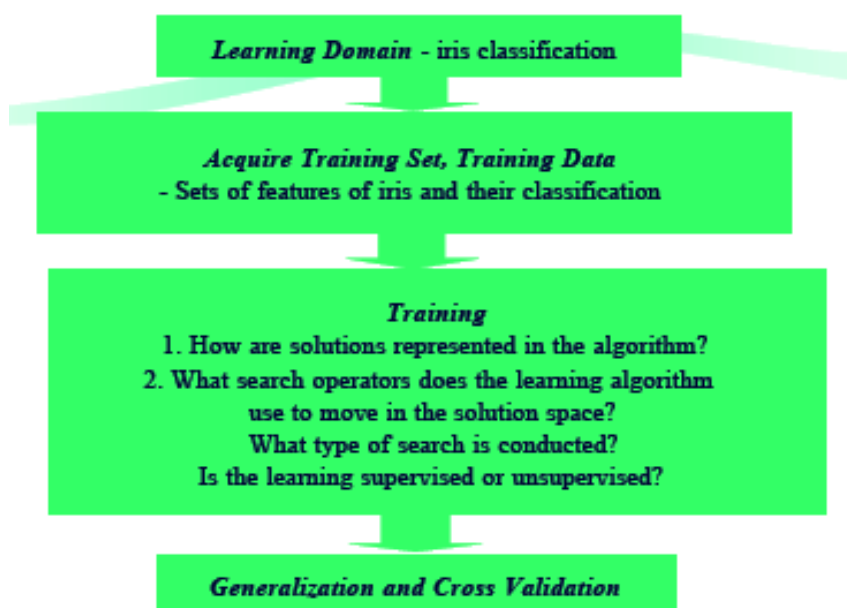
- حال باید آموزش ببینیم: در این مرحله سوالاتی مختلفی مطرح می شود:

۱. چگونه پاسخ ها در این الگوریتم نمایش داده می شود.

۲. از چه عملگرهای جستجویی برای اینکه یادگیری انجام شود استفاده نماییم.

۳. آیا یادگیری باید با سرپرستی باشد یا بدون سرپرستی باشد و یا صرفاً بر اساس یکسری بازخوردها از محیط باشد که این پایه ی یادگیری تقویتی^{۸۷} است.

فلوچارت این الگوریتم در زیر نشان داده شده است



⁸⁷ Reinforcement learning

بعد از اینکه یادگیری انجام شد امیدواریم که بتوانیم آن را تعمیم دهیم و به وسیله *cross validation* آزمایش کنیم که این فرآیند یادگیری تا چه حد قابلیت تعمیم دارد.

- *Genetic programming* کجای این فرآیند یادگیری قرار دارد؟

می توانیم آن را یکی از روش های یادگیری بنامیم و اگر دقیق تر بخواهیم عنوان کنیم آن را یکی از روش های بهینه سازی می توانیم بنامیم.

- تفاوت میان یادگیری، بهینه سازی و *learning* در چیست؟

در درازمدت یادگیری به بهینه سازی همگرا می شود. به عبارت دیگر یادگیری آن *step* ها و گام های کوچک است که ما در آنها به طرف بهینه حرکت می کنیم. بنابراین در اغلب روش ها مانند *Random search* و غیره که در قسمت بهینه سازی مطرح شدند می توانند به عنوان یک روش *learning* قلمداد شوند. تفاوت عمده ای که بین *learning* و *optimization* می توانیم ببینیم در این بازه زمانی است ، بدین صورت که در *learning* به دنبال این هستیم که از گامی فعلی و حرکت فعلی چه چیزی می توانیم یاد بگیریم و موقعیت خودمان را بهبود دهیم. بنابراین این می توان چنین عنوان کرد که *Learning* در یک *step* انجام می شود حال آنکه *optimization* روی کل فرآیند مطرح می شود یعنی می تواند در یک بازه زمانی طولانی پس از گذراندن گام های بسیار آن وقت بحث *optimization* مطرح شود.

- *Genetic programming* یک روش *optimization* است یا *Learning*؟

پاسخ ساده این است که یک روش *optimization* است یعنی زمانی که ما معمولاً از *GA* یا *GP* نام می بریم. یک تابع برازندگی داریم که اگر آن را *min* یا *max* نماییم آن فرآیند را بهینه می نماید. بنابراین *GP* یک روش بهینه سازی است.

- آیا می توان از *GP* به عنوان یک روش *learning* هم استفاده کرد؟

بله مثلاً اگر ما از *GP* استفاده نماییم دیگر مرز بین *optimization* و *Learning* دیگر خیلی کدر می شود.

تفاوت دیگر بین *GA* و *GP* در نوع *coding* آنها برای شکل دهی کروموزوم هایشان می باشد. بدین ترتیب که چون در *GP* تاکید بر روی بهینه سازی ساختار است. بنابراین ساختار کروموزوم می تواند از نوع *Tree* باشد و یا هر نوع دیگری باشد. این در حالی است که تمامی اصول *GA* نظیر *selection, mutation* و غیره همچنان وجود

دارند و تفاوت تنها در *coding* مسئله و نحوه نمایش و شکل گیری کروموزوم می باشد. با توجه به اینکه *coding* تغییر کرده در عملگر *crossover* هم باید یک بازنگری انجام شود.

مثال:

نمونه ای از کاربرد این مسئله طراحی یک *mobile Robot* به شکلی است که بتواند برای تحقیقات به مریخ فرستاده شود. مشکلی که برای این *Robot* وجود دارد این است که محیط مریخ یک محیط کاملاً ناشناخته است و اینکه نمی دانیم این روبات دقیقاً کجا فرود می آید و علاوه بر این نمی دانیم عناصر مورد نیاز را کجا خواهد یافت. دیگر اینکه به دلیل فاصله زیاد نمی تواند برای هر عملش با زمین تبادل اطلاعات نماید و دستور بگیرد چون در این صورت انرژی اش خیلی زود تمام می شود. بنابراین باید به روبات خود مختاری^{۸۸} و قدرت تصمیم بیشتری دهیم (*learning*).

و بدین منظور یکی از ابزارهای بسیار قدرتمندی که در اختیارمان است *Behavior basis Robotics* یعنی روباتیک رفتارگرا می باشد. بدین معنی که رفتارهای روبات را جداگانه برای آن تعریف نموده و به روبات اجازه می دهیم در شرایط مختلف یکی یا ترکیبی از این رفتارها را از خود به نمایش بگذارد.

بنابراین تصمیم بر این شد که از روباتیک رفتاری در ترکیب با منطق فازی برای طراحی این روبات استفاده شود. استفاده از منطق فازی این امکان را می دهد که همه چیز و همه موانع را یکسان نگاه نکنیم و بنابراین با مانعی که خیلی بزرگ است به اندازه یک مانع خیلی بزرگ و با یک مانعی که خیلی کوچک است به اندازه یک مانع خیلی کوچک برخورد شود. علاوه بر این منطق فازی اجازه می دهد که تعامل بین رفتارها به تدریج انجام شود بدین صورت که اگر مانعی به میزانی از روبات دور است روبات هم به همان اندازه رفتار پرهیز از مانع را از خود نشان می دهد. بنابراین ترکیب بین فازی و روباتیک رفتارگرا^{۸۹} یک ترکیبی خیلی موفق است که به آن ژنتیک هم برای بهبود بیشتر آن اضافه شده است.

بنابراین در این طراحی برای *Mobile Robot* یک ساختار تصمیم گیرنده فازی طراحی شد که قوانینش توسط برنامه سازی ژنتیک بهینه می شود و این یک ترکیب کاملاً معقول است.

منطق فازی یا کنترل فازی (*Fuzzy control*): ابزاری برای تصمیم گیری و انجام رفتارهای هوشمند است.

Genetic programming: رفتارهای هماهنگ فازی را نتیجه گیری می کند و قوانین فازی را بهینه می نماید.

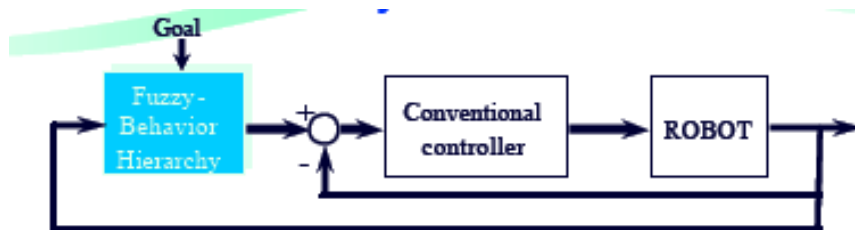
⁸⁸ autonomy

⁸⁹ Behavior basis Robotics

Behavior-based control: نوابری و هدایت خودکار و خودرفتار روبات را فراهم می کند.

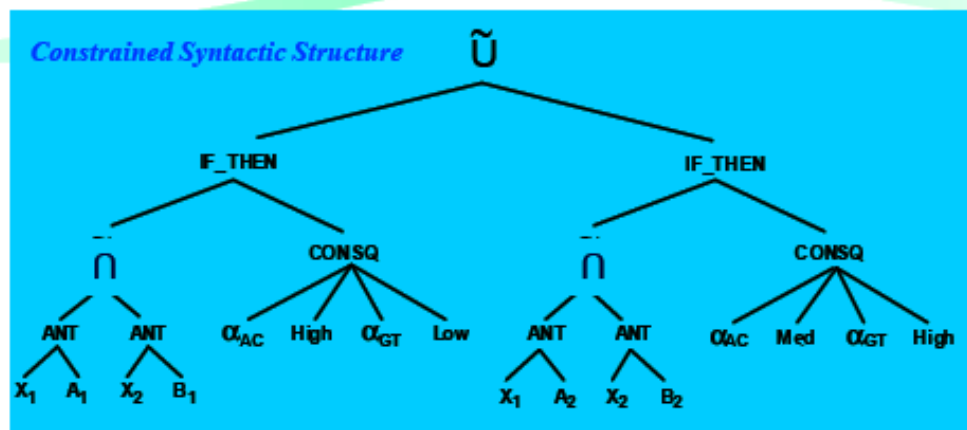
۳-۴. طراحی روبات:

برای طراحی این روبات از یک سری کنترلر استفاده می شود.



کنترل *PID*: یک کنترلر ساده که در اغلب سیستم *Dynamic* کاربرد دارد این مسائل را حل می کند و از مشتق و انتگرال و فیدبک مستقیم برای کنترل استفاده می کند.

ناظر یا *Fuzzy controller* بدین صورت عمل می نماید که سعی می کند رفتارها را به صورت یکسری رفتارهای ابتدایی (*primitive behavior*) و یکسری رفتارهای مرکب (*composite behavior*) تعریف نماید. به عبارت دیگر یکسری رفتارهای سازه داریم که یکسری رفتارهای مرکب از آنها بوجود آمده است.

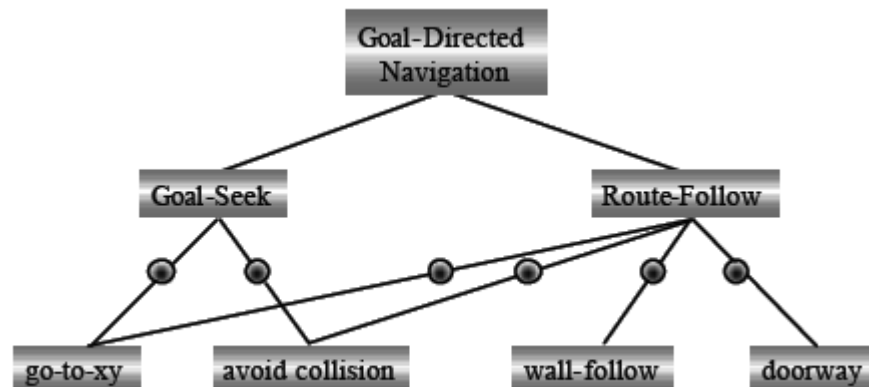


Rule1: IF x_1 is A_1 and x_2 is B_1
 THEN α_{AC} is *High*; α_{GT} is *Low*
Rule2: IF x_1 is A_2 and x_2 is B_2
 THEN α_{AC} is *Med*; α_{GT} is *High*

در *GP* هدف بهینه کردن یکسری قوانین فازی بود و تاکیدمان بر روی میزان *Real* پارامترها نیست. فرض می

کنیم Mf ها تعریف شده هستند. تاکیدمان بر روی این است که کدام Mf در کدام مقدم^{۹۰} و با کدام تالی^{۹۱} قرار بگیرد. در ساختار درختی رسم شده بین دو قانون یک «یا» OR وجود دارد. یعنی بین قوانین اجتماع می گیریم چون تک تک این قوانین در $local\ area$ خودشان متغیر هستند

مثالی کاربردی با استفاده از ساختار فوق برای بخشی از کنترلر فازی رفتار گرای طراحی شده برای یک ربات در زیر نشان داده شده است.:



با توجه به رفتارهای شکل قبل می توان نتیجه گرفت که رفتارهای متفاوت با ترکیبشان یک رفتار فازی یا یک $Fuzzy\ behavior\ hierarchy$ را بوجود می آورند. نکته ای که در رابطه با این کنترلر فازی وجود دارد این است که رفتارها قدم به قدم و تدریجی هستند ($gradual$) و ناگهان بروز نمی کنند به عنوان مثال به یکباره $door$ روشن و $wall-follow$ خاموش نمی شود. همچنین به در نزدیک می شود رفتار $doorway$ پررنگ تر شده و به آن تاکید بیشتری می شود و بالعکس تاکید بر رفتار $wall-follow$ کمتر می گردد.

شکل و ساختار درختی نشان داده شده تصویر قبل در حقیقت یک پایگاه قانون است با دو $Rule$ فازی.

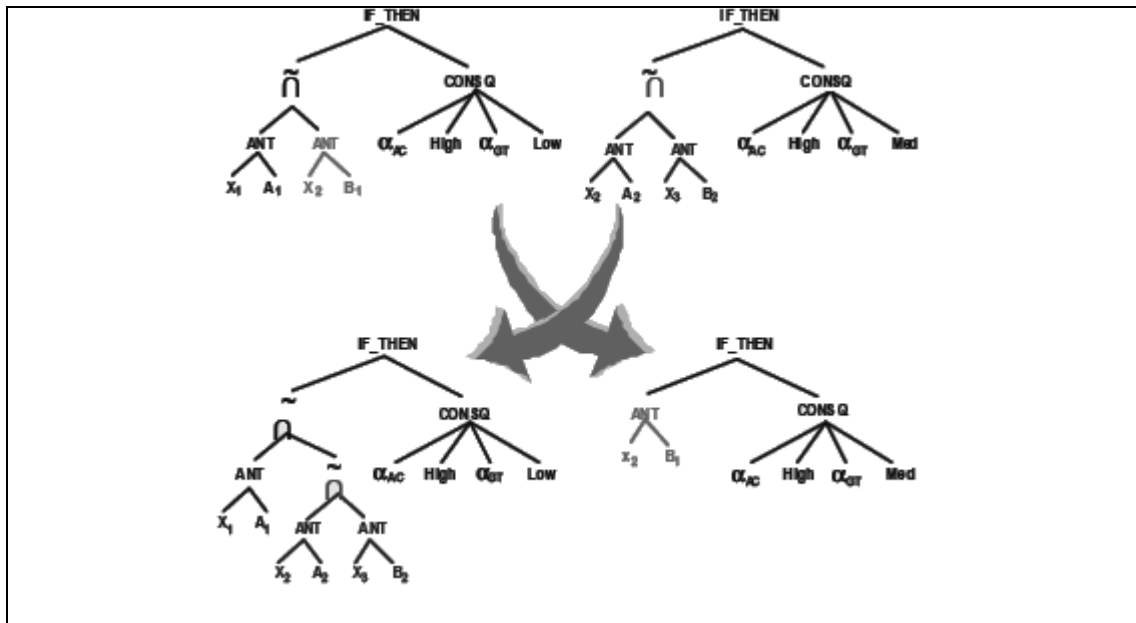
این شکل درختی در حقیقت همان کروموزوم در GP است. بنابراین برای این شکل از کروموزوم باید عملگر $crossover$ را اصلاح نموده و یا عملگر جدیدی تعریف نماییم که با این ساختار جدید سازگار باشد. در شکل درختی نمایش داده شده قانون اول نقش $individual1$ و قانون دوم نقش $individual2$ را بازی می کند و $crossover$ بین این دو انجام می شود.

در این ساختار از بین نودهای معتبر یک نقطه و یا یک نود برای $crossover$ انتخاب می شود دقت شود که این

^{۹۰} antecedent

^{۹۱} consequence

نقطه باید در جایی انتخاب شود که *valid* باشد. به عنوان مثال ما نمی توانیم تالی را در مقدم و یا مقدم را در تالی قرار دهیم بلکه در نتیجه *crossover* قسمت های تالی و مقدم در قانون فقط می توانند با همانند خود عمل باز ترکیب را انجام دهند.



همانطور که در تصویر فوق نمایش داده شده است تمام قسمت مقدم از قانون دوم جایگزین بخشی از قسمت مقدم قانون اول شده است.

برای جهش نیز همینطور عمل می شود بدین صورت که بطور *Random* یک نقطه انتخاب می شود و از میان مقادیری که می تواند بپذیرد یک نقطه جایگزین آن می شود.

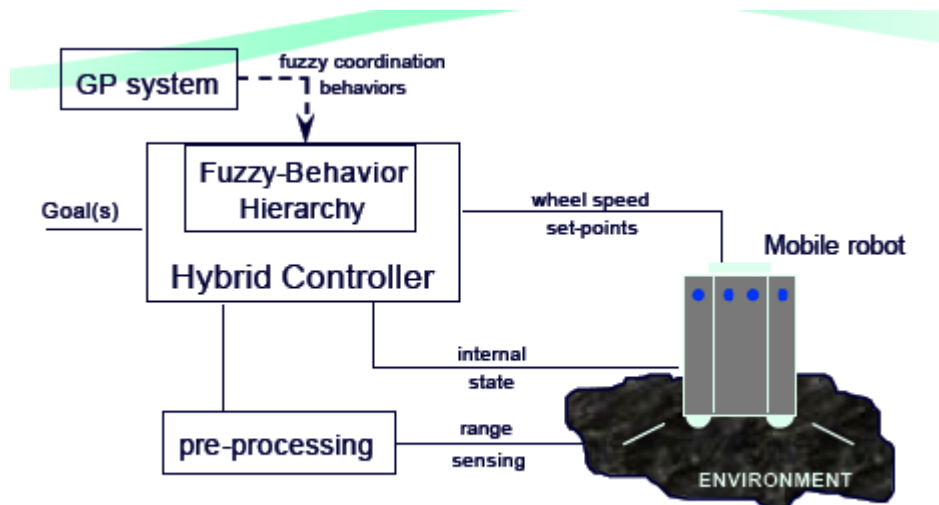
LoBot نمونه ای از یک *mobile robot* است که از یک پردازشگر *Pentium 75 MHz* برای کنترل آن استفاده می شود علاوه بر این از تعدادی سنسور *ultrasonic* نیز برای *sense* محیط و شناسایی موانع در این روبات استفاده می شود (۵ تا در جلو برای موانع و ۳ تا در کنار برای شناسایی دیوارها) از آنجایی که سنسورهای *ultrasonic* بسیار نویزی هستند معمولاً به یک سنسور اکتفا نمی کنند و بیش از یک سنسور را بکار می گیرند. این روبات از دو رفتار *wall-following* و *doorway* را در موتور تصمیم گیری فازی^{۹۲} خود پشتیبانی می کرد و بنابر این قادر بود تا در یک محیط با عدم قطعیت و با موانع متحرک عمل کند.

این روبات در سطح پایین یک *hybrid controller* بود و در سطح *mc68Hc11* تصمیم گیری شدت جریان برای موتورهای انجام می شد و در سطح بالاتر هم یک کنترلر رفتارگرای فازی برای اینکه روبات بالاخره به کجا حرکت

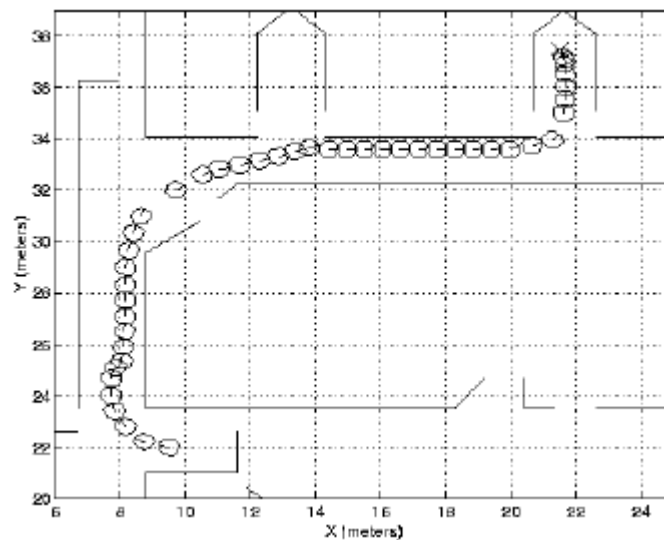
^{۹۲} fuzzy decision making

کند تصمیم می گرفت و این رفتار در نهایت توسط GP بهینه می شد.

تصویری از ساختار این روبات و نحوه تعامل اجزای آن با یکدیگر در زیر نمایش داده شده است.



این روبات زمانی که نزدیک به یک دیوار می رسد و مانعی بر سر راه آن وجود ندارد رفتار *wall following* را دنبال می کند یعنی مسیر دیوار را دنبال می کند و در جهت دیوار حرکت می کند و وقتی که به هدف نزدیک می شود رفتار *goal fitting* در آن تقویت می شود.



به این روبات هیچ نقشه ای از محیط داده نشده و فقط هدف داده شده بنابراین رفتار این روبات کاملاً واکنشی است یکی از ویژگی های مدل رفتار گرا (*behavior based modeling*) همین رفتار واکنش گرا است چون یک رفتار منطقی گرا نمی تواند با سرعت کافی به شرایط محیط پاسخ دهد.

Functional accuracy (درست بودن عملگرها): یعنی اینکه مهم نیست درست باشد مهم این است که رفتار به موقع درست باشد. مثلاً برای بازار بورس اینکه یک نتیجه خوب در زمان طولانی بدست آید اصلاً مطلوب نیست و به درد نمی خورد. بنابراین مدل های رفتارگرا و به این خاطر طراحی می شوند که مدل بتواند *functionally accurate* باشد این می تواند بهینه نباشد اما مهم است که بتواند در زمانی که *decision making* هنوز معتبر است پاسخ را بیابد.

۴-۴. بهینه سازی دانش فازی

بهینه سازی سیستم های فازی دو قسمت دارد:

۱-توابع عضویت :

برای توابع عضویت مثل هر روش استاندارد دیگر می توانید یک رشته ای از پارامترها را به عنوان کروموزم در نظر بگیرید و آن را بهینه نمایید.

۲- قوانین:

برای قوانین می توان یک ماتریس را در نظر گرفت که هر عضو از آن مربوط به یک قانون است. به عنوان مثال اگر فضای ورودی دو تا متغیر دارید و هر کدام را با ۳ تا MF ، قسمت بندی کرده اید، کلاً ۹ قانون می توانید داشته باشید که این ۹ تا قانون را در قالب یک ماتریس 3×3 می توانید نمایش دهید.

<i>Rule1</i>	<i>Rule2</i>	<i>Rule3</i>
<i>Rule4</i>	<i>Rule5</i>	<i>Rule6</i>
<i>Rule7</i>	<i>Rule8</i>	<i>Rule9</i>

ولی مسئله ای که ایجاد می کند این است که توابع عضویت را به صورت رشته ای و قوانین فازی را به شکل ماتریس چگونه در کنار هم قرار دهیم؟

یک روش به نام *co-operative co-evolutionary* وجود دارد که در این روش مسئله را به ۲ قسمت کوچک تر تقسیم می کند:

(۱) GA : بهینه سازی توابع عضویت

مسئله بزرگ را به ۲ مسئله کوچک تر تقسیم کردیم و در ظاهر مشکل را حل کردیم ولی مشکل دیگری را ایجاد نمودیم، مشکلی که ایجاد کردیم این است که مجموعه قوانین فازی بدون توابع عضویت مفهومی ندارند. یعنی نمی توانیم برای آنها معیار خطا تعریف کنیم. بنابراین مسئله ای که در این جا مطرح شد مسئله *credit assignment* است که یکی از مسائل هوش مصنوعی می باشد و به این معنا است که اگر دو عامل با همکاری یکدیگر کاری را انجام دهند به کدامیک پاداش این کار تعلق می گیرد. به عنوان مثال اگر ۲ نفر یک پروژه را به صورت مشترک انجام دهند، آیا باید به صورت مساوی به آنها پاداش داده شود یا خیر؟ در این بحث نیز توابع عضویت و قوانین را می توان به عنوان دو عامل در نظر گرفت که خطایی را به دست می آورند و این خطا همان پاداششان است. بنابراین سنوالی که بوجود می آید این است که با چه مکانیزمی به این توابع عضویت و قوانین فازی پاداش دهیم؟

یکی از کارهایی که در این راستا انجام شد معرفی مفهومی به نام دوستی است. که دوستی را به این صورت تعریف می کنند. در جوامع انسانی بهترین ما از نظر فیزیکی و فکری اگر در محلی دور از جوامع انسانی قرار بگیریم نمی توانیم حداقل امکانات را برای خودمان بوجود آوریم. بنابراین در انسان ها، این دوستی و برخورد با اجتماع در نحوه زندگی آنها تاثیرگذار است. که مشابه آن نیز در برخی از موجودات دیگر مانند، رفتار پرندگان، *Ant colony*، ... هم مشاهده می شود اما دوستی که در انسان ها قرار دارد مفهوم عمیق تری است. البته در مفهوم دوستی که استفاده نمودیم برازندگی من، متوسط رفتار من با افراد کره زمین نمی باشد بلکه برازندگی من بیشتر تابع رفتار من با اطرافیانم مانند پدر، مادر، برادر و ... است. بعد از آن مفهوم دوستی، مفهوم دیگری به نام *aging* را در این فرآیند پیاده سازی نمودیم. یعنی عامل ها نه تنها با بخش کوچکی از جامعه شان می توانستند رابطه برقرار کنند بلکه مسن هم می شدند و در این فرآیند نشان دادیم که می توانیم جواب های بهتری داشته باشیم گرچه سرعت همگرایی را به شدت کاهش دادیم. در حقیقت *diversity* را با *co-operative co-evolution* افزایش دادیم ولی زمانی که *diversity* را افزایش می دهیم با توجه به اینکه جستجوی بهتری می توانیم انجام دهیم، اما سرعت همگرایی کاهش پیدا می کند.



۵. شبکه عصبی:

شبکه عصبی یکی از ساختارهایی است که در آن، دانش هوشمند می تواند قرار گیرد. در حقیقت شبکه عصبی یک حافظه هوشمند است که مبنای آن یک حافظه انجمنی (*associated memory*) است.

ازجمله دلایلی که به شبکه های عصبی، حافظه های هوشمند می گویند این است که دارای قابلیت تعمیم پذیری می باشند و در مقابل نویز مقاوم می باشند. به عنوان مثال یک شبکه عصبی را که آموزش داده شده تا یک صندلی را بشناسد با توجه به اینکه شبکه با صندلی دسته دار آموزش داده شده حال اگر یک صندلی بدون دسته به آن داده شود باز هم آن را به عنوان یک صندلی شناسایی می کند. به عبارت دیگر با توجه به اینکه اطلاعات ناقص به آن می دهیم می تواند اطلاعات کامل را بازیابی نماید. این هوشمندی به خاطر قابلیت تعمیم پذیری و مقاوم بودن شبکه در مقابل نویز می باشد (زمانی که یک قسمت از تصویر را به آن نمی دهیم در حقیقت یک نویزی به آن سیستم اضافه شده است و سیستم در مقابل آن نویز مقاومت نشان می دهد).

نوع دیگری از *Robustness* هم وجود دارد که از طبیعت الهام گرفته است مانند ویژگی که در مغز انسان وجود دارد. که هر روزه تعدادی از نرون ها را از دست می دهد ولی باز هم خاطرات دوران کودکی را از یاد نمی برد. بنابراین در مقابل از دست دادن تعدادی از نرون ها باز هم توانای خودش را حفظ می کند این *Robustness* به خاطر ویژگی *associated memory* است که در مغز وجود دارد.

اگر در سیستم های هوشمند هم بتوانیم این *Robustness* را داشته باشیم خیلی عالی می شود. چرا این *Robustness* برای سیستم های مصنوعی مهم است؟ دلیل آن را این گونه می توان بیان کرد که اگر یک ساختاری قدرتمند وجود داشته باشد که نسبت به از بین رفتن برخی از نودها در شبکه، خودش را می تواند حفظ کند در آن صورت می تواند در مقابله با خطاهایی که وزن های آن عصب ها دارند خودش را حفظ نماید. به عبارت دیگر در یک شبکه ما همیشه می توانیم تحمل کنیم که برخی از نودها وزن هایشان، وزن های ایده ال نباشند و در عین حال سیستم کارایی خودش را از دست ندهد. و ما به دنبال این *Robustness* در سیستم های خود می باشیم.

بنابراین شبکه عصبی یک ساختار *Robust* است نسبت به نویز و هم قابلیت تعمیم پذیری (*Generalization*) دارد و این دو تا از خاصیت های مهم در شبکه های عصبی است. ویژگی های شبکه عصبی را می توان به چند دسته تقسیم نمود که عبارتند از:

۵-۱. ویژگی های شبکه عصبی:

:Handle noise Well

خاصیت اول شبکه عصبی که قبلا هم گفته شد خاصیت *handle noise well* است که هنگامی که آموزش داده می شود می تواند الگوها را حتی اگر بخشی از آنها را از شبکه مخفی کنیم بتوانند آن را بشناسند.

:Robust

خاصیت دوم شبکه *Robustness* آن نسبت به وزن هایشان است یک بعد این *Robustness* این است که چون در شبکه های عصبی، اطلاعات توزیع شده است (*distributed*) است شبکه های عصبی خیلی بهتر می توانند بقا پیدا کنند. به عنوان مثال اگر فرض کنیم که یک شبکه عصبی داشته باشیم که در کل ایران پخش شده باشد حال اگر یک کشور دیگر ایران را بمباران کرد این شبکه هنوز قابلیت خودش را می تواند حفظ کند توزیع پذیر بودن اطلاعات این خاصیت ها را دارد. یا اگر یک *big Processor* را در نظر بگیریم که شامل مجموعه ای از کامپیوترها است که شبکه عصبی روی آنها کار می کند و هر کدام از آنها یک بخشی از شبکه عصبی هستند. حال اگر یکی از کامپیوترها *down* شود اگر یک مسئله خیلی بزرگی را داشته حل می کرده هنوز هم می تواند بخشی عمده ای از توانائی هایش را حفظ کند. بنابراین ۲ مزیت دارد خاصیت برخورد مناسب با نویز و خاصیت مقاوم بودن در مقابل خطاها.

:Parallelism

خاصیت سوم، خاصیت موازی پذیری شبکه های عصبی است. به عبارت دیگر اگر ما یک شبکه عصبی داشته باشیم که ابعادش خیلی وسیع باشد آن را می توانیم در سطح بالایی با یک *Parallelism* افزایش کنیم. البته شرط آن این است که موازی پذیر باشد یعنی باید بتوانیم آن ها را به بخش های کوچکتر بشکنیم. بنابراین سرعت محاسبات هم به همان نسبت پائین می آید. هر چه بتوانیم آنها را بیشتر بشکنیم سرعت هم به همان اندازه افزایش می یابد. اما باید در نظر داشت که بعضی از مسائل را نمی توانیم به بخش های کوچک تر بشکنیم.

:learn new patterns

خاصیت چهارم آنها این است که می توانند الگوهای جدید را به خوبی یاد بگیرند. یک سری از الگوها را یاد دارد و از آن می خواهید که یک سری از الگوهای جدید را هم یاد بگیرد.

:provide a model of intelligent mechanism

خاصیت بعدی شبکه عصبی این است که با الگوبرداری از طبیعت ساختاری را برای مکانیزم های هوشمند به ما معرفی می کند. ما در موجودات زنده شبکه عصبی را داریم با الگوبرداری از این موجودات می توانیم امیدوار باشیم در قالب این ساختار در نهایت به ساختاری برسیم که هوشمندی مصنوعی را بتواند اجرا کند.

:success in areas like vision

در برخی از زمینه ها مخصوصا، شناسایی الگو و پردازش تصویر، شبکه های عصبی خیلی موفق بوده اند و در جاهایی که روش های کلاسیک به نتایج خوبی نرسیده اند شبکه های عصبی به موفقیت هایی رسیده اند.

:promising model of associative memory

یک مدل امیدوار کننده از حافظه مشترک (*associated memory*) است. اگر بخشی از آن اشتباه کند و یا خراب شود بخش های دیگر جبران می کنند. در سیستم های فازی هم حافظه مشترک را هم داشتیم، به خاطر همپوشانی توابع عضویت، قوانین می توانند حذف شوند و الگوریتم کارایی خود را خیلی از دست ندهد. (

:provide tool for modeling and exploring brain function

اگر این سیستم از طبیعت الگوبرداری شده شاید دوباره بتواند برگردد و به ما در شناخت طبیعت کمک کند. بنابراین این گونه به ما کمک می کند که با توجه به توسعه آن و با توجه به اینکه در شبکه های عصبی مصنوعی موفق بوده آیا مانند این هم در سیستم های طبیعی هم وجود دارد یا خیر؟ آیا می تواند سرنخ هایی برای شناختن طبیعت و هوش طبیعی باشد.

:Universal Approximation Probability

یکی خاصیت های شبکه های عصبی، قابلیت تقریب گیری عمومی (*universal approximation*) است. به عبارت دیگر می توانیم امیدوار باشیم که یک شبکه عصبی بتواند یک رابطه غیرخطی را پیش بینی کند. و این خاصیت خیلی مهمی است. به عبارت دیگر اگر یک مسئله ای داریم که برای آن یک قالب شبکه عصبی تعریف می کنیم خیلی مهم است که بدانیم در آن قالب و آن ساختار حداقلی پا سخی وجود دارد (قبل از اینکه ما انرژی و وقت خود را برای پیدا کردن پارامترهای آن پاسخ بگذاریم) بنابراین یک سوال خیلی مهمی را با عنوان *universal approximation prob* می تواند پاسخ دهد.

در کنار این مزایایی که گفته شد شبکه های عصبی دارای معایبی هم هستند:

۱- آنچه که به آن باید توجه کنیم این است که شبکه های عصبی با همه این امیدها و توانائی ها هیچ وقت خود مغز نیستند. در مغز فرآیندهایی وجود دارد که در شبکه های عصبی مصنوعی وجود ندارد. به عنوان مثال در مغز هر روز نرون هایی از بین می رود و نرون های جدیدی بوجود می آیند اما در شبکه های عصبی که ما با آنها سروکار داریم دارای ساختار ثابت (*fix*) هستند و چند لایه هستند. در مغز مجبور نیستند لایه ها به صورت سلسله به سلسله باشند. آنچه که در شبکه *MLP* وجود دارد ساده ترین شبکه عصبی و ساده ترین برداشت ها است که از شبکه عصبی پشت چشم الگو برداری شده است. که به صورت لایه لایه اطلاعات را پردازش می کند و هر لایه یک عملی را انجام می دهد. در حالیکه بخش اصلی شناسایی که در بخش *cortex* مغز صورت می گیرد خیلی مسئله پیچیده تری است.

۲- شبکه های عصبی که ما می سازیم گرفتار نفرین ابعاد هستند (در سیستم های فازی هم مشکل نفرین ابعاد وجود دارد). نفرین ابعاد در شبکه عصبی به این صورت است که وقتی تعداد ورودی ها از ۲ و تعداد خروجی ها از یک بیشتر شود می بینیم که پیچیدگی مسئله یک دفعه به صورت نمایی می شود. و ما هنوز با توجه به افزایش توانائی های کامپیوترهایمان نتوانسته ایم آن را حل کنیم.

۳- در شبکه های عصبی ما هنوز با این مشکل مواجه هستیم که در عین حال که توانایی خوبی برای خورد با داده های عددی و خام داریم در عین حال برای مدل سازی مفهوم های بالاتر مانند توجه (*attention*)، نمادها (*symbols*)، تمرکز، احساس مشکل داریم.

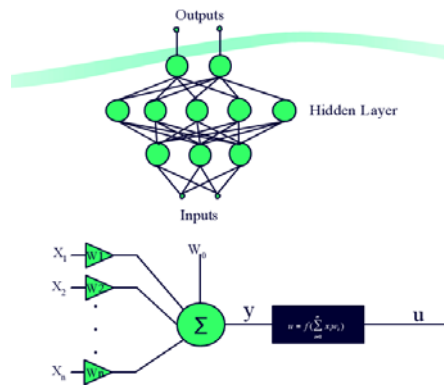
۴- زمانبر بودن، خستگی پذیر بودن و طولانی بودن الگوریتم های آموزش (*long- tedious*)

۵- لزوما همیشه نمی توانیم همگرا شویم (*convergence*)

البته این مشکلات نباید باعث شود که ما روی شبکه های کار نکنیم زیرا امید بسیاری وجود دارد که به عنوان یک مدل *low level* (پائین به معنای این که با داده های خام کار می کند) در زمینه ادراک و حافظه مشترک پیشرفت های خوبی داشته باشیم.

یک شبکه عصبی را می توان به طور کلی به صورت زیر نشان داد:

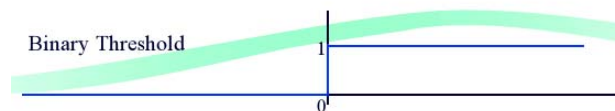
The Multi-layer Perceptron



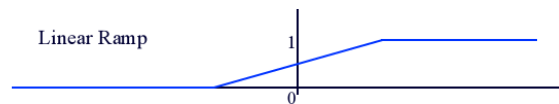
همان طور که در شکل بالا مشاهده می کنیم یک سری ورودی به یک سری خروجی وصل می شوند و ورودی ها در یک وزنی ضرب می شوند و با یک بایاس جمع می شوند و میزان آتش یک عصب را تعیین می کنند اگر این میزان روشن شدن (هیجان)، از یک حدی افزایش پیدا کرد آن وقت یک عصب یک پالس را به خروجی می دهد.

این که عصب نسبت به *excitation* ای (هیجان - تحریک) که در آن وجود دارد که حاصل جمعی از ورودی های مختلفش است چگونه برخورد کند و به بیرون آتش کند بستگی به طراح مسئله دارد که از چه جداساز یا تفکیک کننده ای (*Discriminators*) استفاده کند چند نمونه از این تفکیک کننده ها عبارتند از:

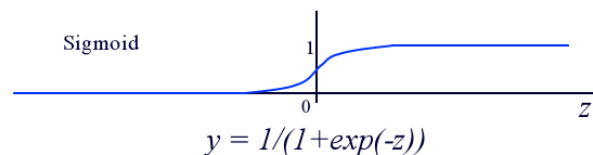
آستانه بانیری (*Binary Thresholding*): که اگر از آستانه بیشتر بود آتش کند (یا خاموش باشد یا روشن)



Linear Ramp: یک عصب به تدریج آتش می کند تا به یک حدی برسد. یک بازه تعریف می شود که در این بازه به صورت خطی پاسخ می دهد.



sigmoid: در یک بازه ای صفر و ۱ است و در بازه های دیگر به تدریج بین ۰ و ۱ حرکت می کند.



نکته: اگر بخواهیم یک روش بهینه سازی در شبکه عصبی استفاده کنیم که مبتنی بر مشتق باشد مانند آنچه که در سیستم های فازی صحبت شد از کدام موردهای بالا استفاده کنیم؟

پاسخ: *sigmoid* زیرا مشتق پذیر است.

هر شبکه عصبی یک لایه ورودی و یک لایه خروجی دارد لایه های میانی را لایه های پنهان می گویند که می تواند وجود داشته باشد یا نداشته باشد و یا چند تا باشد. معمولا تعداد لایه های پنهان شبکه عصبی را به عنوان تعداد لایه های شبکه عصبی می گویند. به عنوان مثال زمانی که می گویند شبکه یک لایه، یعنی یک لایه پنهان دارد (علاوه بر لایه ورودی و خروجی که همیشه وجود دارند)

لایه پنهان به این معنا است که از ورودی به خروجی به طور مستقیم اثرگذار نیست. از طریق لایه پنهان قابل اثر است.

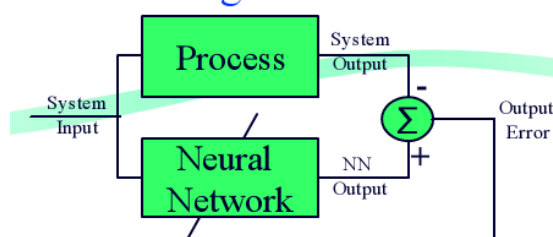
۵-۲. کاربردهای شبکه عصبی:

۱- Modeling/ Identification:

یکی از کاربردهای شبکه عصبی این است که می تواند به عنوان مدلسازی فرآیند استفاده شود. به عنوان مثال یک سری زمانی از آب و هوا داریم یا یک سری زمانی از سیگنال حیاتی یک موجود زنده داریم و می خواهیم آن را یاد بگیرد. به عنوان مثال می خواهیم مقایسه اش کنیم با موقعی که خطا ندارد یا می خواهیم آب و هوا را برای سال آینده پیش بینی کنیم تا با استفاده از آن استفاده از سدها را بهینه کنیم.

بنابراین یک فرایند داریم که می خواهیم با شبکه عصبی آن را یاد بگیریم تا بتوانیم مدلسازی کنیم. یک سری ورودی و یک سری خروجی داریم. ورودی را به شبکه عصبی و فرآیند می دهیم خروجی را محاسبه می کنیم اگر خطایی وجود داشته باشد وزن های شبکه عصبی را به گونه ای تنظیم می کنیم که خطا به حداقل برسد که این استفاده از شبکه عصبی برای مدل سازی است.

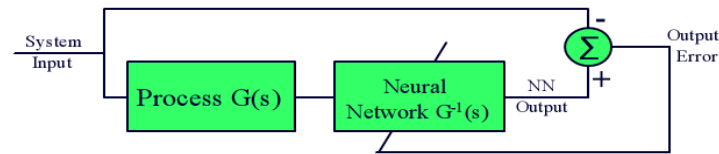
Modeling/Identification



۲- Inverse Modeling :

از شبکه عصبی می توانیم برای بدست آوردن مدل معکوس یک فرآیند هم استفاده کنیم.

Inverse Modeling



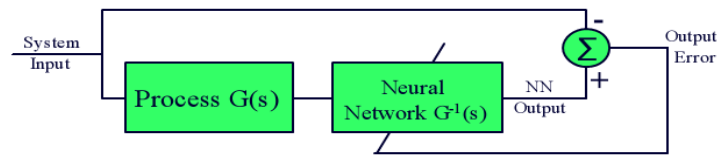
به عنوان مثال اگر یک ربات را در نظر بگیریم ممکن است *kinematic* آن را راحت بتوانیم بدست آوریم. معمولاً در رباتیک اگر یک ربات با ۲ درجه آزادی داشته باشیم اگر θ ها را داشته باشیم به راحتی می توانیم x, y آن را بدست آوریم. این می شود مسئله *kinematic* - برعکس آن یک مسئله مشکلی است زیرا از مسیرهای مختلف (x, y) می توان به یک θ خاص رسید یعنی یک راه حل ندارد- گاهی اوقات این راه حل ها هم از مسیرهای عبور می کند که *single variety* وجود دارد *single variety* یعنی اگر من بخواهم به اندازه یک ϵ حرکت کنم باید بی نهایت گشتاور اعمال کنم. بنابراین خیلی جالب می شود که معکوس این مدل را داشته باشیم یعنی به این فرآیند θ ها را بدهیم و آن (x, y) دهد سپس این (x, y) ها را به شبکه عصبی بدهیم و شبکه عصبی به ما θ را بدهد این می شود بدست آوردن مدل معکوس یک فرآیند که این می تواند مدلسازی *inverse kinematic* باشد یا مدلسازی دینامیک خود ربات باشد.

پویا است- به خاطر *conventional* که دارد با توجه به خطایی که در سیستم ایجاد می شود تغییر می کند.

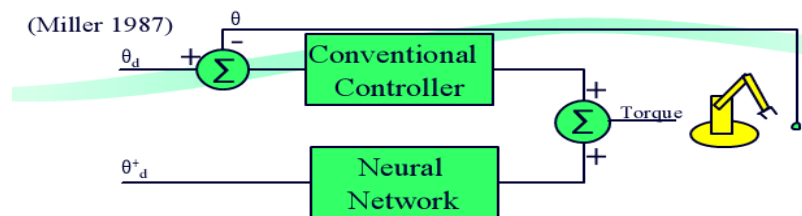
اگر بتوانیم مدل معکوس دینامیک دقیق یک ربات را داشته باشیم در آن صورت اگر بخواهم ربات در یک نقطه خاص بایستد اگر مدل معکوس آن را داشته باشم می گویم که چه گشتاورهای داشته باش تا در این نقطه قرار بگیری- این خیلی عالی است و این کاری است که در این ساختار انجام می دهیم.

اگر یک سیستم داشته باشیم که برای ربات *inverse dynamic* آن را محاسبه کرده باشد در آن صورت ما θ مشهور را به آن می دهیم به ما گشتاور می دهد. بنابراین کافی است مدتی این شبکه عصبی را گذاشته باشم و به ربات گشتاورهای مختلف بدهم و θ های مختلف را مشاهده کنم حالا این شبکه عصبی را بگیرم تا معکوس آن را یاد بگیرد. حالا اگر θ به آن بدهم گشتاور به من می دهد. و این خیلی ایده ال است.

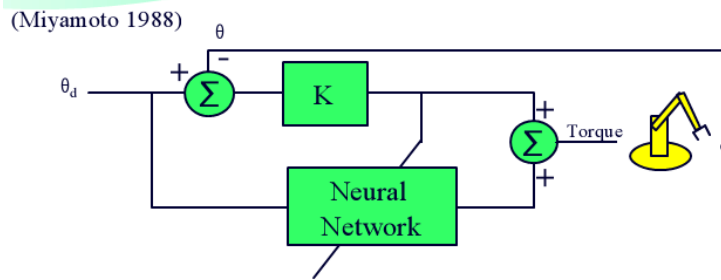
Inverse Modeling



اما همیشه خطا وجود دارد همیشه این گشتاوری که من محاسبه می کنم معکوس دقیقی نیست اگر بود مسئله حل بود. بنابراین چون دقیق نیست مجبورم برای جبران این خطا در محاسبه معکوس، یک کنترلر معکوس بگذارم. کارم خیلی ساده شد چون که کنترلر کلاسیک فقط می خواهد خطای من را جبران کند و مابقی را شبکه عصبی برای آن انجام می دهد.



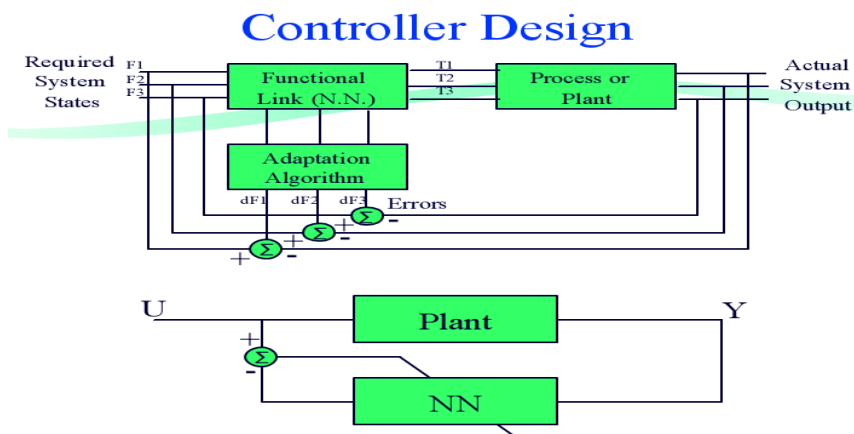
یک حالت دیگر استفاده از شبکه عصبی است که اینجا هم دقیقاً از معکوس استفاده می کند. θ_d را می گیریم و گشتاور را می خواهد بدهد. فرض کنید می خواهیم بروید کنترل (تلویزیون) را بردارید در ذهن خودم یک مدل از این ساخته ام تا چه نیرو و گشتاورهایی بر روی بازوهای خودم باید اعمال کنم تا بتوانم این کنترل را به صورت موفق بردارم می دانم که اگر نیرو به اندازه کافی استفاده نکنم نمی توانم آن را بردارم و اگر نیرو بیش از اندازه استفاده کنم ربات ممکن است آن را له کند. بنابراین یک چیزی در ذهن دارم که از تجربیات گذشته سرچشمه گرفته است. بنابراین مدل معکوس کور است فرض کنید من قبلاً این کنترل را برداشته باشم ولی مشابه آن را قبلاً برداشته ام. می آئیم می بینیم این هم شبیه قبلی است اگر بخواهم آن را بردارم باید گشتاور بیشتری وارد کنم. بنابراین نیروی بیشتری وارد می کنیم این وظیفه *gain* است تا اینجا هم مثل قبلی است و فرقی نکرده در این جا با یک *conventional controller* رفتار خود را تصحیح می کنم اما چیزی که این جا فرق کرده این است که اینجا بر اساس این خطا، شبکه عصبی را بیشتر آموزش می دهم به عبارتی دفعه بعد شبکه عصبی که با این کنترل (تلویزیون) برخورد کرد می داند که آن نیروی که برای برداشتن آن اعمال کرد چند است. و برای برداشتن این کنترل (تلویزیون) جدید باید نیروی بیشتری وارد کنیم.. تفاوتی که این مدل با مدل قبلی دارد این است که شبکه عصبی اینجا پویا است با توجه به خطایی که در سیستم حاصل می شود تغییر می کند همان *conventional controller* که دارد باعث می شود پویا باشد علاوه بر آن خودش را هم تنظیم می کند.



اگر بتوانیم مدل معکوس دینامیک را داشته باشیم خیلی خوب است. مثلاً اگر بخواهیم روبات در یک مکان خاصی بایستد می‌گوییم چه گشتاورهایی داشته باشد تا به این مکان برسد.

کنترل کلاسیک فقط خطا را کنترل می‌کند. همیشه خطا وجود دارد. همیشه گشتاوری که محاسبه می‌کنیم معکوس دقیقی نیست و مجبوریم برای جبران خطا در محاسبه معکوس، یک کنترلر کلاسیک بگذاریم که فقط خطا را جبران می‌کند و بقیه کار را شبکه عصبی انجام می‌دهد.

اما گاهی اوقات به یک شبکه عصبی به عنوان یک مجموعه داده ورودی-خروجی نگاه می‌کنیم به عبارت دیگر ما می‌خواهیم به آن نمونه‌های مطلوب (*case*) را بدهیم از آن گشتاور بگیریم و در این میان کنترلر می‌خواهیم طراحی کنیم نمی‌خواهیم مدل معکوس سیستم را طراحی کنیم. در این صورت فرآیند را داریم کنترلر را به آن اعمال می‌کنیم ورودی‌های مطلوب را می‌گیریم مقایسه می‌کنیم با ورودی‌های واقعی سیستم، خطایی را که حاصل می‌شود را به عنوان ورودی به یک الگوریتم تطبیقی می‌دهیم الگوریتم تطبیقی وظیفه اش این است که پارامترهای شبکه ما را تقریب بزند. و به این وسیله ما می‌توانیم یک سیستمی داشته باشیم که حتی اگر فرآیند در طول زمان تغییر کند خودش را با آن تطبیق دهد.

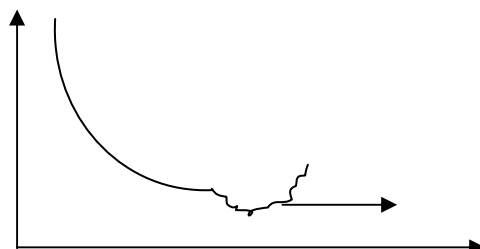


نکته: شبکه‌های عصبی بازگشتی هستند زمانی که شبکه خیلی آموزش ببیند دچار *over training* می‌شود. بنابراین داده‌ها را به ۳ قسمت تقسیم می‌کنیم. در این صورت بخش کوچکی از داده‌های *test* را به عنوان

checking به شبکه عصبی می دهیم تا به ما بگوید که آیا شبکه یاد می گیرد یا دچار *overtraing* شده است.

می خواهیم آنقدر یاد بگیریم که بتوانیم تعمیم دهیم و دچار فراموشی در اثر یادگیری زیاد نشویم.

اگر در آموزش خطا افزایش می یابد و در آزمایش خطا کاهش می یابد دچار *overtraing* شده ایم.



خطا رو به افزایش است توقف آموزش
آموزش کافی است و ادامه نده وگرنه دچار *overtraing* می
شویم زمانی دچار *overtraing* می شویم که در ۳ مرحله

۳-۵. مبانی شبکه های عصبی:

در این بخش جزئیات بیشتری از شبکه عصبی بیان می شود.

Axon: آنچه که از نرون منشعب می شود تا سیگنال از طریق آن به عصب های دیگر برود

Dendrites: سیگنال را از نرون های دیگر دریافت می کند.

محل اتصال *Axon* و *Dendrites*، که به آن *Synapses* می گویند می تواند *inhibitory* (گین منفی)

excitatory (گین مثبت) باشد اثری که روی *activation* نرون دارد می تواند گین منفی یا گین مثبت باشد.

شبکه های عصبی مصنوعی با مغز انسان و شبکه های عصبی طبیعی خیلی فاصله دارند. در مغز انسان 10^{11} نرون

داریم که هر کدام از آنها با 10^4 نرون دیگر در ارتباط است. به عبارت دیگر در مغز انسان 10^{15} تا اتصال وجود دارد.

شاید امروزه داشتن 10^{11} نرون مشکل باشد ولی مشکل اصلی که وجود دارد نحوه یادگیری و آموزش این شبکه ها

است که چگونه می توان این حجم زیاد اتصال (*connection*) را آموزش داد.

۴-۵. تاریخچه:

شاید بشود گفت توسعه شبکه های عصبی مصنوعی، از حالت مصنوعی شروع نشد بلکه در سال ۱۹۴۳ دو نفر

محقق، شبکه عصبی طبیعی را خلق کردند بعد یک ریاضی دان به نام جان وان نیومن به این نتیجه رسید که با

تکنولوژی که موجود است از قبیل *vacuum tubes* و *relays* می توان شبکه های عصبی مصنوعی را ساخت. ولی

یک مشکل باقی ماند که این بود که چگونه می توان شبکه عصبی را آموزش داد. فردی به نام *Hebb* الگوریتمی

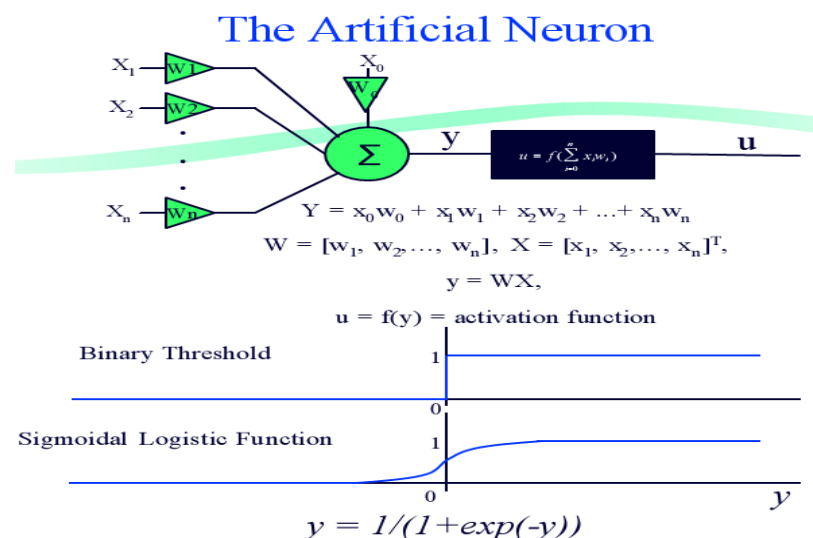
ساده برای آموزش شبکه عصبی ابداع کرد که وقتی که دو تا نرون همزمان آتش می کنند اتصال بین آنها تقویت می

شود که این یکی از روش های کلاسیک در یادگیری شبکه های عصبی بود ولی روش محبوبی نیست. سپس

widrow که مهندس بود آنچه را که *Hebb* پیشنهاد کرده بود را اجرا کند و ماشین *adoline* را اختراع کرد سپس بعد از او، *Rosenblatt* ماشینی را ابداع کرد که ۵۱۲ تا *relay* داشت که می خواست الگویی را شناسایی کنند. در کل ۸ تا الگو به ماشین آموزش داده شده بود و این به نوعی شروعی از شبکه عصبی بود. *Minsky* یک محقق بود که یک مقاله داد و در آن بیان کرد و ثابت کرد که شبکه های عصبی که یک لایه هستند قابلیت تقریب گری عمومی را ندارند حتی یک سیستم غیرخطی *Xor* را نمی توانند درست *classify* کنند (*Xor* فقط ۴ حالت دارد).

اما بعدها ثابت شد که، اگر شبکه های عصبی یک لایه پنهان داشته باشند می توانند *universal approximator* (تقریب گر عمومی) باشند.

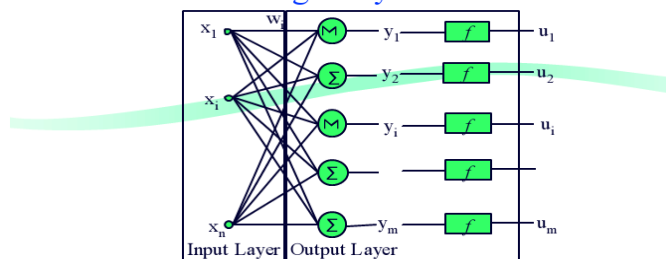
دهه ۸۰ و تا اواسط دهه ۹۰ دوران طلایی شبکه عصبی بود و خیلی رشد کرد.



همانطور که قبلا هم گفته شد در شبکه عصبی هر ورودی در وزن w_i ضرب می شود سپس با هم جمع شده و این حاصل با یک بایاس جمع شده و در نهایت باعث تعیین خروجی نرون می شوند که آتش بکند یا نکند که میزان آتش می تواند *Binary threshold* باشد یا یک *activation function* نرم تری مانند *sigmoidal* باشد. *sigmoidal* برای این که پیوسته است بیشتر مورد توجه قرار گرفته است. (اما گاهی اوقات باید بین چند تا الگو یکی را انتخاب کنیم، آن را که بیشتر است باید انتخاب کنیم در این صورت از *Binary threshold* استفاده می کنیم).

۵-۵. شبکه عصبی تک لایه (*Single layer ANN*)

Single Layer ANN



$$y_i = NN_i = \sum_{j=1}^n w_{ij} x_j$$

$$U = [u_1, u_2, \dots, u_n]^T, \text{ and}$$

W is an $m \times n$ weight matrix.

$$u_i = f(y_i) = f(NN_i)$$

$$U = \Gamma [WX]$$

where Γ is a diagonal matrix.

x_1	w_{11}	w_{12}	$\dots$	w_{1n}	$f_1(\cdot)$
x_2	w_{21}	$\cdot$	$\cdot$	$\cdot$	$f_2(\cdot)$
$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$
x_n	w_{n1}	$\cdot$	$\cdot$	w_{nn}	$f_m(\cdot)$

وزن ها ارتباط بین ورودی و خروجی ها را تعیین می کنند. طبیعی است که u را باید بتوانیم به صورت حاصلضرب WX نشان بدهیم.

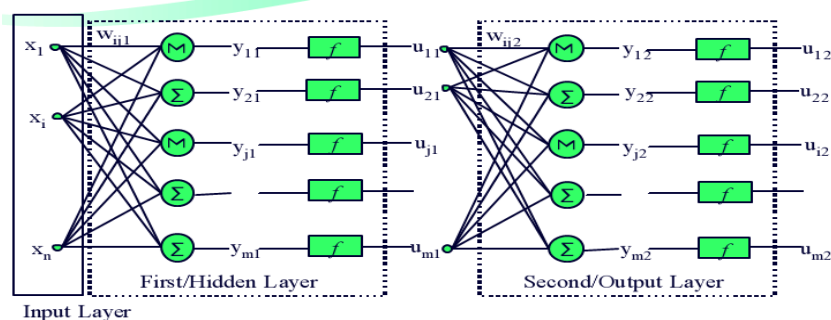
W : ماتریس وزن ها

X : بردار ورودی

۵-۶. شبکه عصبی دو لایه (Two Layer ANN)

اگر مسئله، *linear discriminator* داشته باشیم شبکه های عصبی *single layer* کاربرد دارند. اگر بخواهیم *nonlinear discriminator* داشته باشیم باید یک لایه پنهان داشته باشیم. بنابراین از *Two layer Ann* ها استفاده می کنیم تا قابلیت تقریب گری عمومی را داشته باشیم.

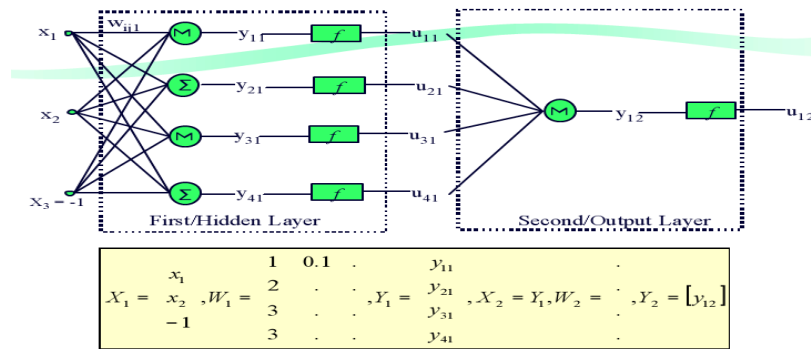
Two Layer ANN



در این جا یک لایه پنهان داریم. بنابراین از ورودی به خروجی کاملاً قابلیت رؤیت نیست. و به این ترتیب شبکه های عصبی قابل بسط هستند.

مثال ۱) در این مثال یک شبکه عصبی با ۳ ورودی و یک خروجی داریم که ماتریس وزن های ورودی یک ماتریس 3×4 است و بردار وزن خروجی یک بردار 1×4 است (ماتریس نیست).

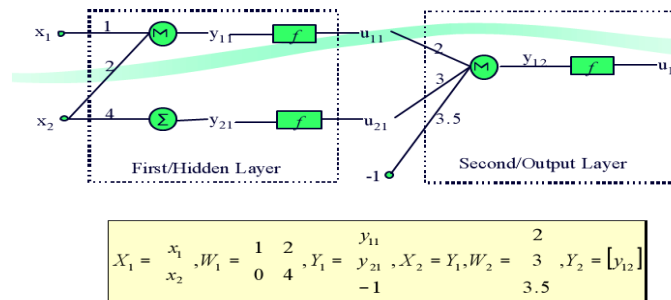
Example 1



بایاس به صورت ورودی در نظر گرفته شده است که می تواند ۱+ باشد.

مثال ۲) در این مثال دو ماتریس وزن W_1, W_2 داریم که در شکل زیر نشان داده شده است:

Example 2

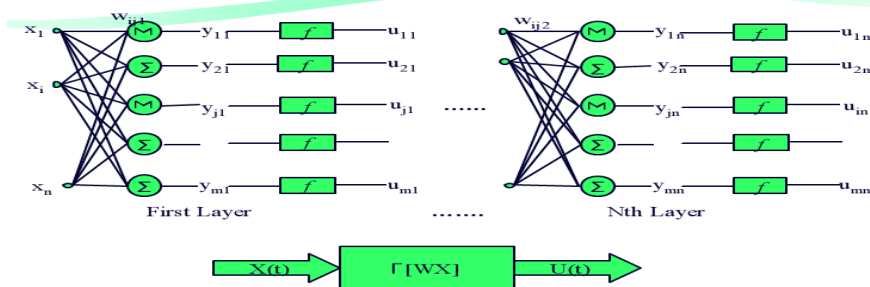


ساختاری که تا اینجا به آن اشاره کردیم به طور دقیق تر *Feed Forward NN* نام دارد زیرا اطلاعات از لایه اول

لایه به لایه به سمت جلو می رود.

Neural Network Configurations

A. FeedForward Networks



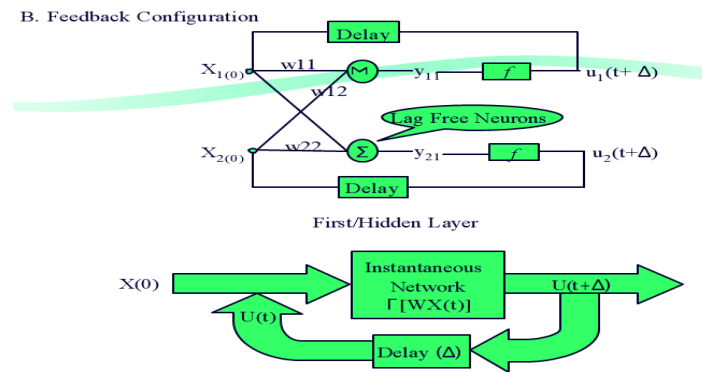
در مقابل آن *Feed back configuration* هم داریم. خروجی شبکه عصبی به ورودی آن برگشته است و یک

Feed back ایجاد کرده است. در این مثال در حقیقت شبکه هیچ ورودی ندارد به عبارتی یک *Feed forward*

NN است که بخشی از خروجی های آن با یک تاخیر به ورودی آن اضافه می شود (علاوه بر ورودی های که می

تواند داشته باشد) این ساختار معمولاً برای سیستم های دینامیک استفاده می شود. سیستم های دینامیک در مقوله

سیستم ها با آنچه در سیستم های هوشمند تعریف می کنیم فرق می کند.



در مثال هیچ ورودی اولیه نداریم و ورودی از خروجی با بخشی از تاخیر گرفته می شود (برای سیستم های دینامیک استفاده می شوند).

۵-۷. تعریف دینامیک بودن در سیستم های کنترلی:

سیستم دینامیک یعنی سیستمی که حافظه دارد. باید ساختاری داشته باشیم که حافظه داشته باشد یعنی بتواند از خروجی در ورودی استفاده کند. این تعریف دینامیک بودن در سیستم های کنترلی است.

در شکل قبلی اگر ورودی نداشته باشیم چه می شود؟

در مدل سازی یا فرض می کنیم که حالت اولیه را داریم که مشکل نداریم یا اینکه در مدل سازی الگوریتم به گونه ای نوشته می شود که اگر حالت اولیه نداریم به آن همگرا شود در طول زمان $state$ های سیستم به $state$ های واقعی همگرا شوند ($adaptive$ است).

حالت اولیه را می توان $Random$ بگیریم و یا از صفر شروع کنیم- در این جا هر بار که ورودی جدید از خروجی دریافت می کند الگوریتم $adaptation$ وزن های شبکه را تغییر می دهد.

۵-۸. یادگیری شبکه عصبی

یکی از معضلات و مشکلات شبکه عصبی این است که بر خلاف دانش انسانی که ما حداقل یک ایده ای داریم، ما در سیستم های فازی تا حدی می دانیم که قوانین را چگونه بنویسیم و توابع عضویت را چه طوری تعریف کنیم ولی در شبکه های عصبی هیچ مکانیزم مشخصی وجود ندارد که در ابتدا بگوید که وزن باید چه باشد، وزن را باید شبکه عصبی بدست آورد چون که $Learning$ در شبکه های عصبی یا تفکیک پذیری در شبکه های عصبی یکی از خصوصیات مهم در شبکه های عصبی است. هدف $Training$ در شبکه های عصبی این است که بر اساس یک

سری ورودی- خروجی ها سعی می کند یاد بگیرد که می تواند به صورت *supervised* (مانند *Backpropagation* و *Least Mean square*) یا *un supervised* (مانند کوهنن و هاپفیلد) باشد.

در *supervised* ما یک سری ورودی- خروجی داریم سپس از الگوریتم های *BP* و *MLS* می توانیم استفاده کنیم و شبکه را آموزش دهیم. در *un supervised* هیچ نظارتی وجود ندارد یعنی کسی نیست که به ما بگوید چقدر خطا کرده ایم. (*un supervised* خیلی سخت است) بنابراین کسی نیست که بگوید خطا چقدر است و این چه است که شبکه یاد می گیرد. مغز انسان این ویژگی را دارد که کلاستر (یا خوشه بندی) می کند. در فکرمان، افکار را کلاستر می کنیم. بنابراین در *unsupervised learning* ما به دنبال چیز دیگری هستیم که یاد بگیریم و آن روابط بین داده هایی است که در فضای ورودی به ما داده شده است و این یعنی این که داریم کلاستر می کنیم.

نکته: دو دیدگاه متفاوت در مورد سیستم های *Dynamic* وجود دارد

در تعریف سیستم به سیستم های متغیر با زمان *time varying* گفته می شود اما در *AI* به سیستم های متغیر با زمان دینامیک (*Dynamic*) گفته می شود.

سیستم دینامیک: دارای حافظه هستند $x(k+1)=x(k)$

سیستم متغیر با زمان: با زمان تغییر می کند $x(k+1)=t x(k)$

هر سیستم دینامیکی متغیر با زمان نیست (مانند $x(k+1)=x(k)$)

به تعداد حافظه ای که نیاز دارد تا نقطه بعد را پیش نماید به آن *order* آن سیستم دینامیک گفته می شود. به عنوان مثال برای پیش بینی $x(k+1)$ فقط نیاز به $x(k)$ داریم *order* آن یک است. این *order* باید به گونه ای انتخاب شود که سیستم بتواند یاد بگیرد.

شبکه عصبی

شبکه های عصبی یک سری نرون های خیلی ساده هستند که با یکدیگر در ارتباط هستند. همان طور که قبلا هم گفته شد در مغز انسان 10^{11} نرون وجود دارد که هر کدام با 10^4 نرون دیگر در ارتباط هستند. بنابراین 10^{15} تا اتصال در مغز انسان وجود دارد. در شبکه های عصبی معمولی هم ما یک همچنین چیزی داریم یک سری نرون های ساده که هوشمندی در تعامل بین نرون های ساده تعریف می شود. به عبارت دیگر این که چگونه چه وزن هایی بین نرون های شبکه تعریف شود رفتار این نرون ها را به صورت جمعی تعریف می کند. این وزن ها را باید توسط آموزش بدست آوریم چون که یک سری وزن هایی داریم که نمی دانیم چگونه این وزن ها را بدست آوریم دو نوع آموزش

داریم:

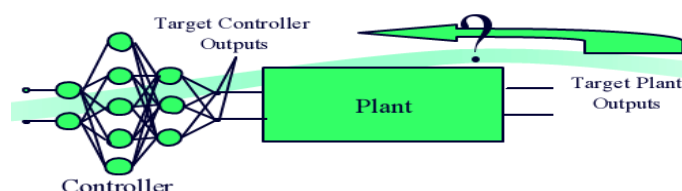
۵-۸-۱. یادگیری نظارتی (supervised):

زمانی است که ما یکسری خطا از سیستم مشاهده می کنیم و بعد بر اساس آن تصمیم گیری می کنیم که وزن ها را چگونه تنظیم کنیم. مثلاً در LMS وزن ها را آن گونه تغییر می دهیم تا مربع خطاها به حداقل خود برسد یا در BP ، خطا را در خروجی به سمت عقب انتشار می دهیم به طوریکه وزن ها را آن گونه تنظیم کنیم که در نهایت به صفر میل کند در این روش ناظر داریم.

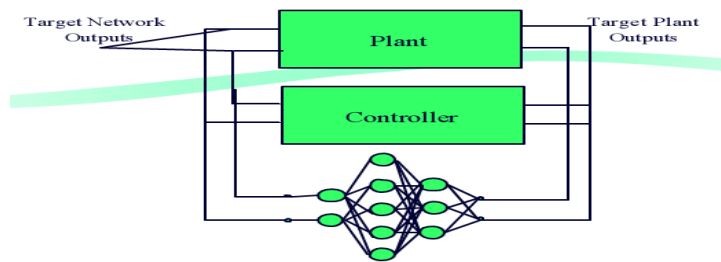
۵-۸-۲. یادگیری بدون ناظر (Unsupervised):

در این روش ناظر وجود ندارد شاید عجیب باشد که به ما بگویند یاد بگیر بدون اینکه بگویند چقدر خطا می کنیم در این روش می خواهیم که روابط بین داده ها را پیدا کند.

فرض می کنیم یک سیستم و یک کنترلر داریم در این جا ما خروجی مطلوب شبکه عصبی را نداریم. اگر می دانستیم توسط *provide training* این کار را انجام می دادیم اما چون نداریم یکی از سؤالاتی که وجود دارد این است که چگونه می توانیم از فیدبکی که از خروجی وجود دارد استفاده کنیم تا وزن ها را تنظیم کنیم؟ که این می شود مسئله آموزش شبکه عصبی برای طراحی کنترلر.



یکی از راه ها این است که ما یک کنترلر مانند انسان داریم و یک شبکه عصبی را به طور موازی با کنترلر بگذاریم تا آن را یاد بگیرد- طراحی کنترلر به این صورت است که از یک کنترلی که در طبیعت وجود دارد الهام بگیریم. چرا این کار را می خواهیم انجام دهیم؟ به خاطر اینکه اگر ما بتوانیم انسانی که مجرب است و سیستم را کنترل می کند توسط شبکه عصبی جایگزین کنیم، در آن صورت این چیزی که جایگزین انسان کرده ایم دیگر خسته نمی شود دیگر خوابش نمی برد، اشتباه نمی کند. بنابراین خیلی مطلوب است.



بنابراین این راه حل برای استفاده از شبکه عصبی برای مدل سازی رفتار انسان یا یک کنترلر موجود می باشد. مدل سازی رفتار انسانی خودش یک مقوله جالبی است چون رفتارهای انسانی ماهیت پیچیده ای دارند که در مرز بین کیاس و *Randomness* قرار می گیرد. به عبارت دیگر کیاس ساده در حد نهائی خود می تواند خیلی پیچیده باشد ولی کیاس *probabilistic* نیست.

(کیاس خیلی به حالت اولیه حساسیت دارد کوچک ترین تغییرات می تواند منجر به تفاوت هایی زیادی در درازمدت بشود. رفتار انسانی یک همچنین ماهیتی دارد یعنی ما به میزانی رفتارمان کیاتیک است و به میزانی رفتارمان *Random* است. بنابراین برای مدلسازیهای پیچیده می توانیم از شبکه عصبی استفاده کنیم گرچه اگر بخواهیم رفتار *Randomness* را هم در رفتار انسانی داشته باشیم) که گاهی اوقات می خواهیم و گاهی اوقات نمی خواهیم) در آن صورت می توانیم از شبکه های عصبی که توانایی *Random selection* دارند استفاده کنیم – چه موقعی ما نمی خواهیم *Randomness* را در مدلی که هست در کنترلر طراحی کنیم؟ وقتی که مثلاً می خواهیم کنترلر یک ربات را طراحی کنیم یا کنترلر یک نیروگاه اتمی را طراحی کنیم در یک چنین موقعیت هایی مانند دنبال آن رفتار نرم هستیم رفتار معمولی که انسان از خودش نشان می دهد نه به دنبال رفتار *Random property* که ممکن است اتفاق بیافتد. در چه مواقعی ما می خواهیم *Randomness* را در انسان مدل کنیم؟

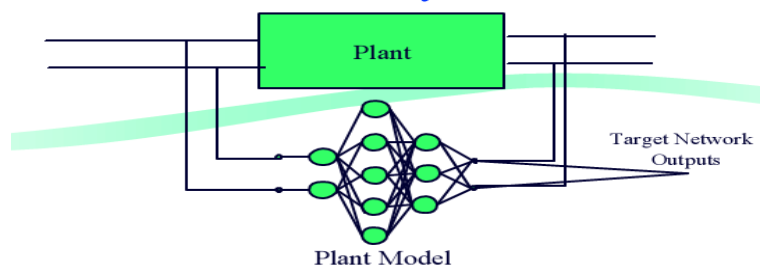
اغلب روش های مدل سازی یک روش مدل سازی استاتیک هستند یعنی یک فرآیند کاملاً *Random* را می گیریم و آن را تبدیل به مدل استاتیک می کنیم (نرم از آن می گیریم). در چنین مواقعی این نرم گرفتن باعث می شود که ماهیتی را از دست دهیم که ما نمی توانیم آن را از دستش بدهیم؟ در محیط هایی که شامل مقابله کننده است ممکن است رفتار احتمالاتی در حقیقت بهترین رفتار باشد. به عنوان مثال در مسئله شکار و شکارچی، اگر شکار به جای رفتار هوشمند و قطعی صرفاً یک رفتار *Random* را از خودش نشان دهد کار برای شکارچی خیلی سخت می شود. بنابراین گاهی اوقات در مقابل مقابله کننده شما می خواهید *Random* رفتار کنید و شاید هم دلیل اینکه طبیعت تکاملی ما هم بقا پیدا کرده شاید به همین دلیل باشد.

وقتی که شما می خواهید محیطی را شبیه سازی کنید که در آن *Randomness* مهم است به عنوان مثال اگر

شما بیائید یک محیط استاتیک شهری را مدل سازی کنید که همه راننده ها همیشه نرم عمل می کنند چه اتفاقی می افتد؟ هیچ اتفاقی نمی افتد. شما می خواهید مثلا میزان تصادفات را محک بزنید ولی رانندگان هیچ وقت تصادف نمی کنند چون نرم رفتار می کنند در حالی که منشا اکثر تصادفات، رفتار *stochastic* رانندگان است که ما گاهی اوقات این رفتار نرم خود را دنبال نمی کنیم. بنابراین مدلسازی رفتار انسانی، بستگی به صورت مسئله ای دارد که برخورد می کنید. مثلا بخواهید فقط پدیده کیاس را در آن در نظر بگیرید. یا نه بخواهید ماهیت *Randomness* را هم در آن در نظر بگیرید.

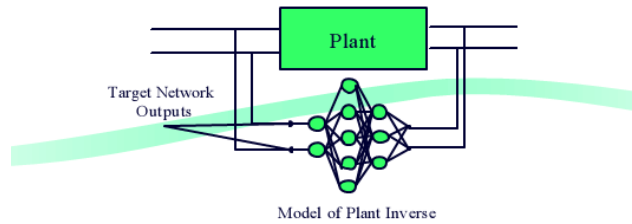
شبکه عصبی می تواند به عنوان یک *system identification* استفاده شود. ورودی های سیستم را داریم خروجی های سیستم را هم مشاهده می کنیم می توانیم توسط *LMS* یا *BP* وزن های سیستم را آموزش دهیم.

II. Neural network for System Identification



اما سؤالی که باقی می ماند این است که اگر ما توانستیم مدلی از سیستم را بوجود بیاوریم آیا این سیستم را می توانیم کنترل کنیم یا خیر؟ و سؤال هایی که در خود مدل سازی پیش می آید این است که چه ساختاری باید داشته باشد؟ آیا این ساختار باید *static* باشد؟ آیا باید از *Feed Forward* استفاده کنیم یا می توانیم از *Feed back* استفاده کنیم؟ این که از ساختار *Feed back* استفاده کند منظور این نیست که می خواهیم پیچیدگی مسئله را زیاد کنیم. یعنی اگر ما ساختار *recursive* یا *Feed back* ای در سیستم داریم به علت نیاز است، چون سیستم ما دینامیک است و نیازه حافظه دارد. اگر سیستم را به هر ترتیب مدل سازی کردیم آیا می توانیم *Inverse* را هم انجام دهیم؟ *Inverse* را هم می توانیم انجام دهیم اما چگونه؟ می توان به این صورت انجام داد، ورودی شبکه، خروجی سیستم و خروجی شبکه، ورودی سیستم باشد این هم در حقیقت *inverse modeling* می شود که ممکن است *exist* بکند یا نکند.

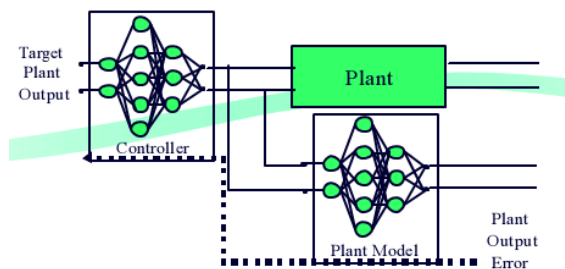
II. Neural Network as Inverse of the System



اگر بتوانیم یک *inverse modeling* در سیستم داشته باشیم آن وقت خیلی عالی می شود. اگر واقعا *inverse* دقیق باشد کافی است آن را در ورودی *Plant* قرار دهیم و به ورودی های آن بگوییم که خروجی های مطلوب را می خواهیم چون یک *inverse model* است به من ورودی مطلوبی که *plant* نیاز دارد را می دهد تا به خروجی مطلوب برسیم.



البته این در صورتی است که *inverse* سیستم امکان پذیر باشد. بنابراین ممکن است *inverse* سیستم وجود نداشته باشد. دوم اینکه اگر هم وجود داشته باشد ما به خاطر خطایی که در مدلسازی وجود دارد ممکن است نتوانیم دقیقا آن را مدل کنیم. بنابراین همیشه یک خطایی وجود خواهد داشت، برای این که این خطا را کاهش دهیم راه حل آن این است که علاوه بر کنترلر، یک *plant model* باید داشته باشیم. در این ساختاری که در این جا توضیح داده شده خروجی واقعی سیستم با خروجی *plant model* مقایسه می شود هر چقدر که بین این دو تا تفاوت وجود داشته باشد *back propagate* می کند به یک ورودی دیگر.

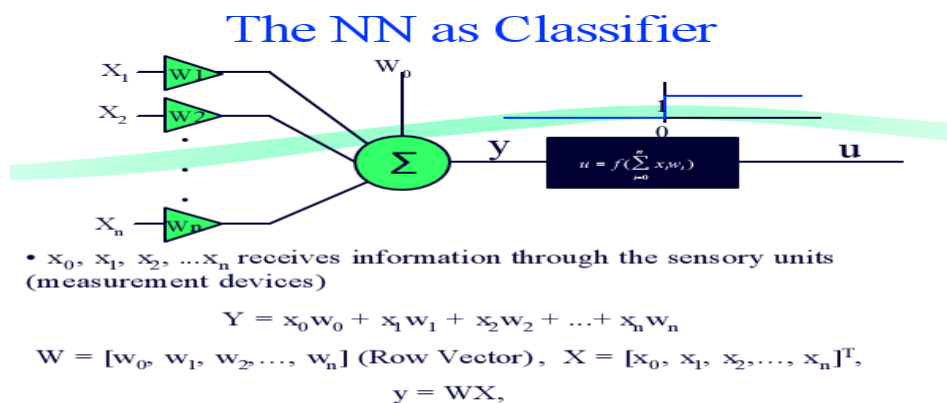


بنابراین سؤال اول که در طراحی کنترلر مطرح شد که ما بدون داشتن خروجی مطلوب کنترلر چگونه کنترلر را طراحی کنیم با این روش حل می شود- خروجی واقعی سیستم را با خروجی *plant model* مقایسه می کنیم چون *plant model* کار خودمان است و خودمان آن را بوجود آورده ایم می توانیم خطا را *back propagate* کنیم به ورودی *inverse* مدل تا بدانیم چقدر باید کنترلرمان را تنظیم کنیم تا خروجی سیستم به آن خروجی

مطلوب همگرا شود. که این هم یکی دیگر از کاربردهای شبکه های عصبی می باشد.

۹-۵. ارائه یک مثال کاربردی از شبکه های عصبی به عنوان یک طبقه بندی کننده

(classifier)



$u = f(y) = \text{activation function}$

• Pattern Observed by sensors belongs to:

1. Class C1 if $y > 0$
2. Class C2 if $y < 0$

در این مثال بخصوص می خواهیم بین دو تا کلاس تفاوت قائل شود. در این مثال خروجی شبکه عصبی حاصل جمع، ضرب ورودی ها به اضافه بایاس است که y می شود و u را هم بر اساس یک *discrivative function* بدست می آوریم. چون در این مثال خاص می خواهد صرفاً بین دو تا کلاس تفاوت قائل شود اگر خروجی بیش از صفر باشد مربوط به کلاس $c1$ است در غیر این صورت مربوط به کلاس $c2$ است.

1. class c1 if $y > 0$

2. class c2 if $y < 0$

۱۰-۵. الگوریتم آموزش (training):

به عنوان مثال فرض کنید الگوریتم می خواهد وزن ها را تنظیم کند. در ابتدا وزن ها را مقدار دهی اولیه می نماییم (*initialize*). که می تواند بین مقادیری که ما مورد نظرمان است به صورت *Random* تنظیم شود و یا مقدار صفر را به آنها دهیم. سپس بر اساس هر نمونه ورودی جدید اگر $x(k) \in C1$ بود و $x(k) w(k) < 0$ بود در این صورت این نمونه درست *classify* نشده است. بنابراین باید وزن ها را تنظیم کنیم.

If $X(k) \in C_1$ and $W(k)X(k) \leq 0$ then , $W(k+1) = W(k) + \delta_0.X(k)$

همچنین اگر $X(k) \in C_2$ بود و $W(k)X(k) \geq 0$ بود باز هم این نمونه درست دسته بندی نشده و باید وزن ها را تنظیم کنیم.

$$W(k) = W(k) - \delta_0 X(k)$$

به این ترتیب یکی یکی زوج های ورودی- خروجی دریافت می شود. و تا زمانی که تمام داده ها درست طبقه بندی شوند این کار ادامه پیدا می کند. البته این کار همیشه این کار امکان پذیر نیست که همه داده های ما درست دسته بندی شوند. گاهی اوقات داده ها آنقدر همپوشانی دارند که نمی شود همچنین کاری کرد ولی در این مثال به خصوص می توانیم این کار را انجام دهیم.

مثال: در این جا ۴ تا داده داریم که هر ۲ تا به یک کلاس تعلق دارند. دو کلاس دایره و مربع داریم. می خواهیم وزن های شبکه عصبی را به گونه ای تنظیم کنیم که به طور خودکار *classifier* انجام شود و بتواند x_1 , x_2 را از x_3 , x_4 شناسایی کند. در ابتدا وزن های اولیه را به صفر تنظیم می کنیم.

$$\delta=1 \quad \text{و} \quad w(k) = [0 \ 0]$$

طبق الگوریتم ابتدا داده x_1 را محاسبه می کنیم چون برابر صفر است یعنی درست *classific* نشده است بنابراین w را باید *update* بکنیم.

$$(1) \quad W(1)X(1) = \begin{bmatrix} 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \end{bmatrix} = 0$$

بنابراین w را باید *update* بکنیم.

$$W(2) = W(1) + X(1) = \begin{bmatrix} 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 2 \end{bmatrix} = \begin{bmatrix} 0 & 2 \end{bmatrix}$$

در مرحله بعدی زوج ورودی- بعدی را محاسبه می کنیم.

$$(2) \quad W(2)X(2) = \begin{bmatrix} 0 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 2 > 0$$

چون بزرگ تر از صفر است درست دسته بندی شده و نیاز نیست که w را تغییر دهیم.

$$W(3) = W(2) = \begin{bmatrix} 0 & 2 \end{bmatrix}$$

نقطه سوم را محاسبه می کنیم که صفر شده بنابراین نادرست تشخیص داده و w را دوباره تنظیم می کنیم.

$$(3) \quad W(1)X(1) = \begin{bmatrix} 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \end{bmatrix} = 0$$

$$W(2) = W(1) + X(1) = \begin{bmatrix} 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 2 \end{bmatrix} = \begin{bmatrix} 0 & 2 \end{bmatrix}$$

این کار را برای چهارم هم تکرار می کنیم.

$$(4) \quad W(4)X(4) = \begin{bmatrix} -1 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1$$

$$W(5) = W(4) - X(4) = \begin{bmatrix} -1 & 2 \end{bmatrix} + \begin{bmatrix} 1 & 1 \end{bmatrix} = \begin{bmatrix} -2 & 1 \end{bmatrix}$$

آیا کار تمام شده ؟ خیر باید برگردیم و این وزن را برای داده ها امتحان کنیم تا ببینیم آیا درست تشخیص می دهد یا خیر؟

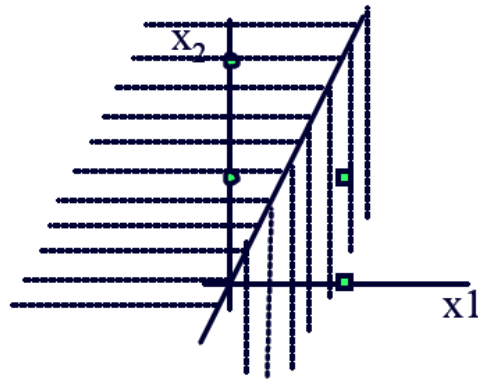
$$(5) \quad W(5)X(5) = \begin{bmatrix} -2 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \end{bmatrix} = 2 > 0 \quad \text{درست}$$

$$(6) \quad W(6)X(6) = \begin{bmatrix} -2 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 1 > 0 \quad \text{درست}$$

$$(7) \quad W(7)X(7) = \begin{bmatrix} -2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = -2 < 0 \quad \text{درست}$$

$$(8) \quad W(8)X(8) = \begin{bmatrix} -2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = -1 < 0 \quad \text{درست}$$

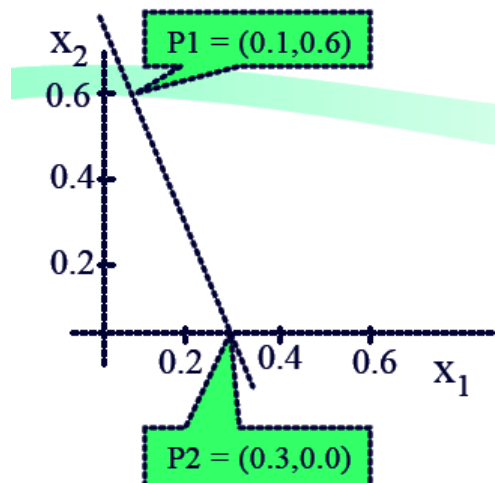
خوشبختانه درست *classific* شده اند بنابراین با ۸ تا *step* توانستیم این ۴ داده را در طبقه های مناسب قرار دهیم. این یک نمونه آموزش دستی برای شبکه های عصبی می باشد اما حالا که این کار را انجام دادیم آیا یک روش ای وجود دارد که من بتوانم این را ببینم ؟ هر خطی که من بکشم که این دو فضا را بتواند از هم جدا کند می تواند آن خط به عنوان *linear discriminator function* باشد؟ این خط منحصر بفرد نیست.



بنابراین یکی از سؤالاتی که ممکن است پیش بیاید این است که آیا لازم بود الگوریتم *Training* را انجام دهیم؟ و یا با معادلات خطی که وجود دارد شبکه عصبی ام را طراحی کنم؟ می توانستیم این کار را بکنیم اما این لزوماً بهینه نیست. بنابراین پشت این الگوریتم های آموزش یک فرض خیلی ساده نهان است ولی در نهایت برای اینکه مطمئن شویم این خط همان خط بهینه است ما به روش های آموزش نیاز داریم علاوه بر این مسائل واقعی خیلی پیچیده تر هستند و به راحتی نمی توانیم یک همچنین خطی را بکشیم.

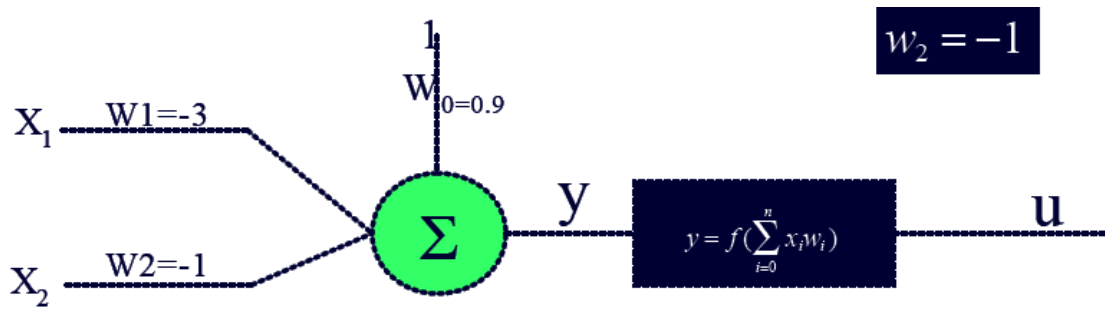
در مثال دوم می خواهیم + را از - جدا کنیم اگر آنها را در فضا تفکیک کنیم می توانیم توسط خط زیر آنها را از هم جدا کنیم. بنابراین با توجه به دانستن فرمول بندی خط ، مقادیر w_0 و w_1 و w_2 به صورت زیر می شوند :

$$w_0=0/9, w_1=-3, w_2=-1$$

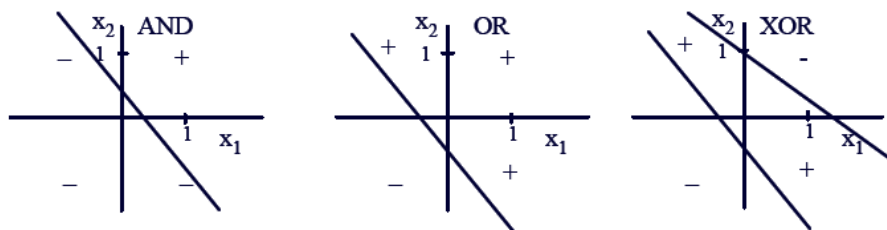


و این یک *discriminator* برای + ها و - ها می شود و یا به طور کلی فرمول خط آن به صورت زیر است

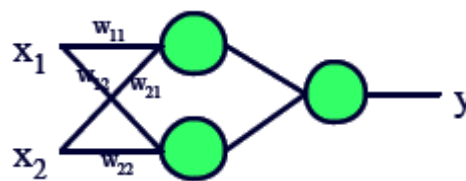
$$y=-3x_1-x_2+0/9$$



بنابراین در مثال های قبلی توانستیم یک سری *discriminator* بدست آوریم و بقیه مرکبی از این ها است. در *classification* برای *XOR* از آنجایی که ما ترکیب مرکب لازم داریم برخلاف *and* و *OR* ما نمی توانیم با یک خط + را از - ها جدا کنیم و به ۲ خط احتیاج داریم.



XOR یکی از مسائلی است که *minsky* نشان که یک شبکه عصبی یک لایه نمی تواند آن را انجام دهد و بعدا نشان داد که اگر یک لایه مخفی به سیستم اضافه شود آن وقت سیستم می تواند این کار را انجام دهد. (تعداد نرون ها = تعداد خطوط)



به طور کلی اگر دو کلاس داشته باشیم که با یک خط بتوان آنها را از هم جدا کرد با جدا ساز خطی می توان آنها را از هم تفکیک نمود. اماگاهی اوقات این کار امکان پذیر نیست و به چند خط نیاز داریم تا آنها را از هم جدا نماییم.

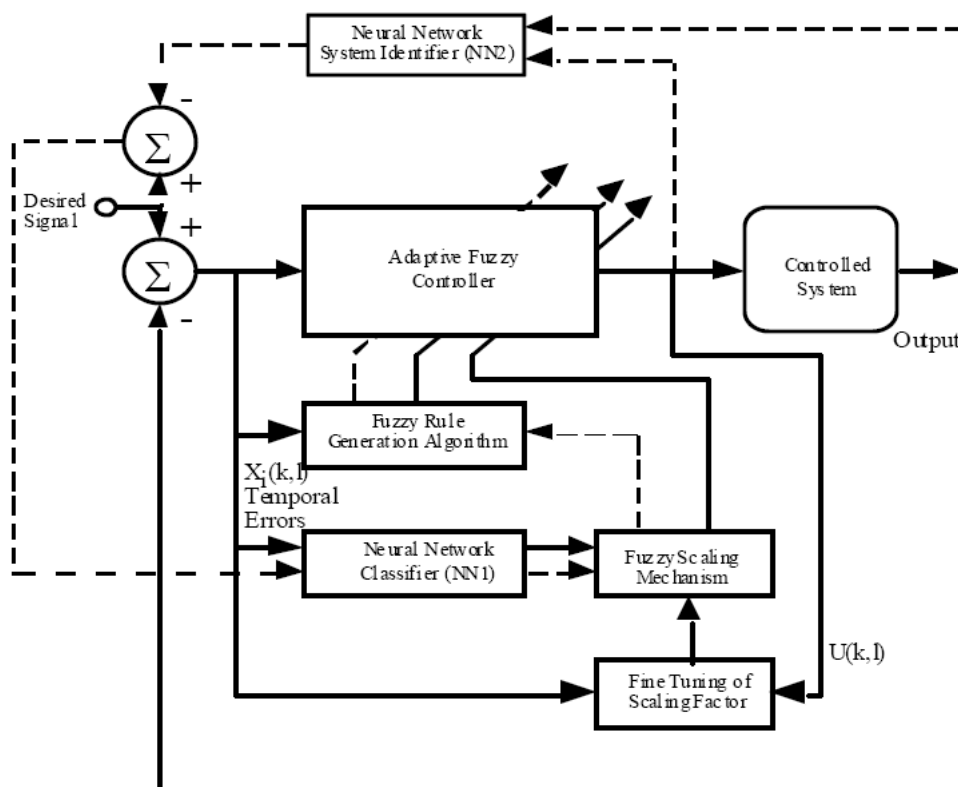


یکی از مسائلی که خیلی کم در شبکه عصبی به آن پرداخته شده تنظیم ساختار شبکه عصبی است چون انتخاب های زیادی برای ساختار شبکه وجود دارد. به عنوان مثال اینکه شبکه چند لایه میانی داشته باشد و هر لایه چند نرون داشته باشد انتخاب های زیادی را ایجاد می کند. روش هایی مانند *BP* و *LMS* صرفاً وزن ها را بهینه می کنند ولی خود ساختار را بهینه نمی کنند در صورتیکه با اصلاح کردن ساختار می توان به شبکه های عصبی خیلی ساده و توأمند رسید.

۵-۱۱. ترکیب فازی و شبکه عصبی

در این قسمت ترکیب فازی و شبکه عصبی مطرح می شود. قسمتی از این کار مربوط به کار آقای گیشان می باشد که بر روی کنترل *disk drive* کار می کرد.

کار اول ترکیب سیستم های *NN* و *Fuzzy Logic* بود اولین کاری که گیشان انجام داد این بود که من چگونه می توانم از شبکه های عصبی و سیستم های فازی استفاده کنم در جایی که ما می دانیم هر کدام از آنها به تنهایی قدرت خودشان را ثابت کرده اند. آنچه که گیشان به آن رسید این بود که از شبکه عصبی به عنوان *classifier* برای طبقه بندی رفتار سیستم ها استفاده نماید. سپس بر اساس اطلاعاتی که از شبکه عصبی بدست آورد توانست یک شبکه *self organizing Neuro-Fuzzy* طراحی نماید. در این ساختار قوانین بوجود می آمدند و حذف می شدند. بنابراین ساختار خود سازمان ده بود. یعنی یکی از خصوصیاتش این بود که قوانین را بوجود می آورد اگر استفاده نمی کرد حذف می کرد. بنابراین هم *adept* می کرد و هم قوانین جدید را *generate* می کرد. این یک نمونه از *self organizing controller* است که از مفاهیم *NN* و *Fuzzy Logic* استفاده می کند. گیشان این ساختار را روی چند *tester* مانند فرآیند تصفیه آب (*Desalination* یعنی جداسازی نمک از آب) و *adept two robot* که یک ربات *Direct Drive* بود امتحان کرد. (ربات هایی که شفت موتور مستقیماً به بازو وصل هستند).



در ابتدا به نظر می آید که این کار ساده و طبیعی است ولی در قالب سیستم های رباتیک این کار (*Direct Drive*) اتفاق نمی افتد بلکه شفت موتور وصل به یک چرخ دنده است. چرخ دنده سرعت را کاهش و گشتاور را افزایش می دهد. سپس این چرخ دنده به بازوی ربات وصل می شود.

ربات هایی که نیاز به دقت خیلی بالای دارند معمولا *Direct Drive* طراحی می شوند *adaptive* یک همچنین رباتی است. این کار دومشکل ایجاد می کند مشکل اول این است که موتور بایستی بزرگ باشد تا بتواند همچنین گشتاوری را اعمال کند یعنی این گشتاور باید مستقیما روی بدنه موتور ایجاد شود. اگر موتور سنگین شد ربات هم سنگین می شود. مشکل دوم این است که کوچک ترین اغتشاشی در انتهای بازو، در طول بازویر گشتاور اعمال می شود. بنابراین خطا را خیلی سخت می شود کنترل کرد. کنترل موتورهای *Direct Drive* یک مسئله تحقیقاتی است که برای ربات هایی که نیاز به دقت بالایی دارند استفاده می شود. به عنوان مثال در تولید مواد نیمه رسانا رباتی که استفاده می شود باید خیلی دقیق باشد. اما در رنگ رزی ماشین از ساختار پویا که چرخ دنده دارد استفاده می کنند که خطایش بیشتر است ولی در عوض خیلی مناسب ترو صنعتی تر است.

در مثالی که بر روی شبکه عصبی *Neuro-Fuzzy* مطرح می شود گیشان آن را روی یک موتور *Direct Drive* که مسئله خیلی جالبی است و به صورت *Real Time* است پیاده سازی نمود.

در این ساختار خروجی موتور *Direct Drive* با ورودی مطلوب مقایسه و خطای آن محاسبه می شود تا یک کنترلر فازی را هدایت کند تا معین شود چه گشتاوری را به سیستم اعمال نماید. (تا اینجا کنترل حلقه بسته معمولی است).

در ابتدا این کنترلر فازی هیچ قانونی ندارد و کلیه قوانین در حین اجرا بوجود می آیند. چگونه در حین اجرا بوجود می آیند؟

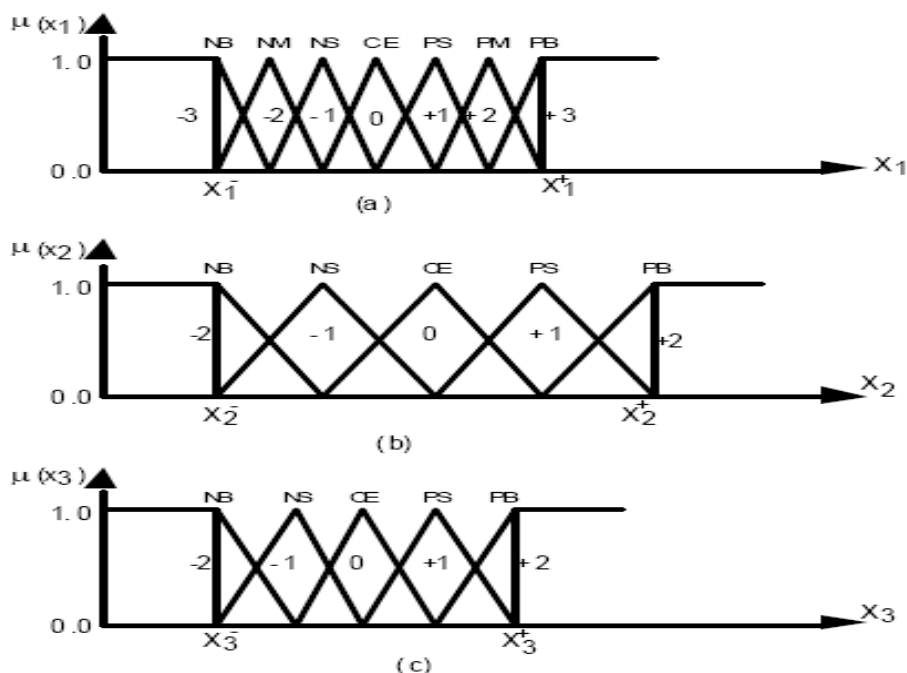
اولین کاری که گیشان با آن روبرو بود این بود که یک شبکه عصبی طراحی کند که بتواند رفتارهای مختلف سیستم را مشاهده و شناسایی کند. اگر یک شبکه عصبی داشته باشیم که توانای انجام این کار داشته باشد، آن وقت می توانیم از رفتاری که مشاهده شده برای تعریف و تنظیم قوانین استفاده نماییم. به عبارت دیگر کاربرد دو مرحله انجام شد. ۱- *scale* کردن فضای *universal* و ۲- ایجاد قوانین جدید.

گام اول این است که ساختار کلی *adeptive Fuzzy control* را در نظر بگیریم. *Universal approximation problem* یک ساختاری است که می تواند یک رابطه غیرخطی را تقریب بزند. این ساختار کلی است که گیشان از آن استفاده نمود. (برای غیرفازی ساز از *center average* استفاده کرد).

ولی کنترلر فازی بدون قانون ارزشی ندارد لذا باید قانون بوجود بیاوریم، قانون را چگونه بوجود می آوریم؟ نکته ای که گیشان به آن رسید این است که اگر بیاییم و تاریخچه رفتار سیستم را نگاه کنیم البته نه تاریخچه طولانی اش را، بلکه *L* تا *step* قبلی اش را، آن وقت در *L* تا *step* قبلی نگاه کنیم و ببینیم که رفتار سیستم به کدامیک از توابع عضویت بیشتر تعلق داشته است.

فضای *universe* برای یک *PIP* را به صورت زیر تعریف نمودیم.

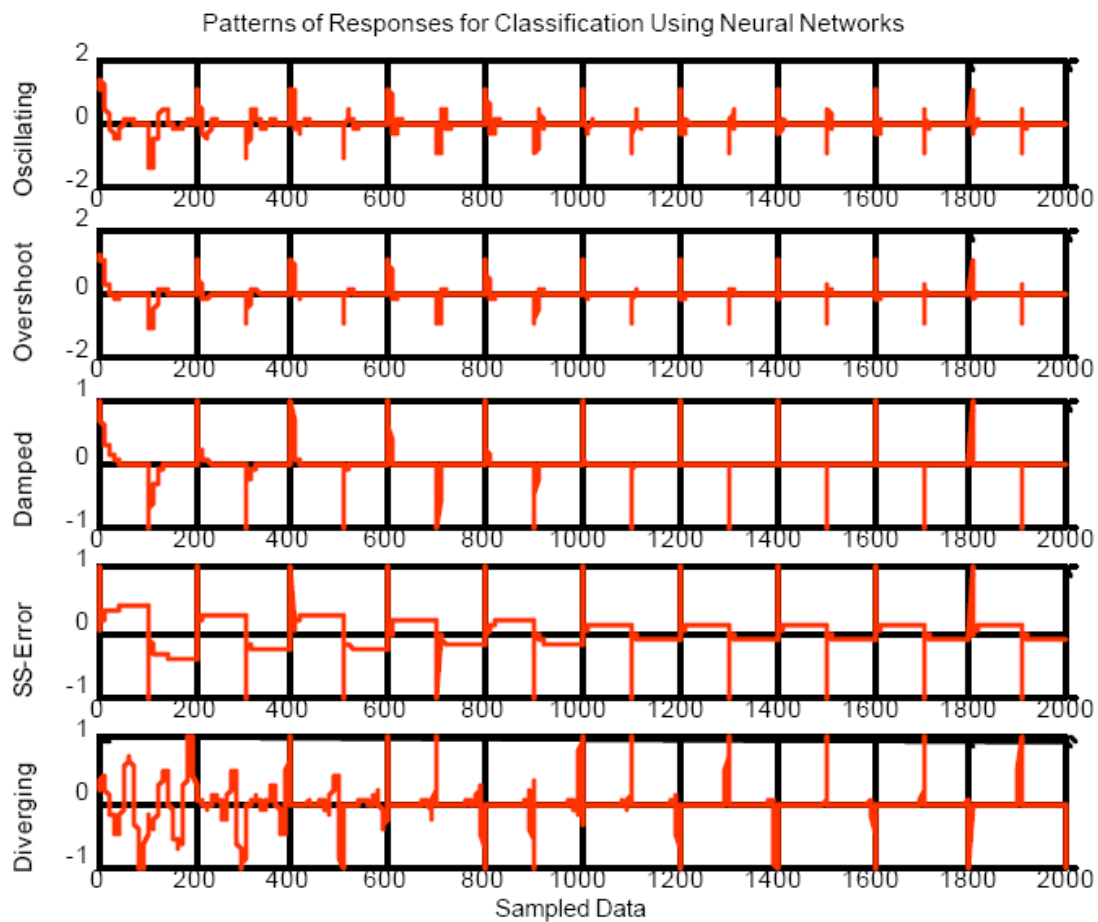




که P را به ۷ تا I و D را به ۵ تا قسمت تقسیم نمودیم. در اینجا خطای صفر، خطای کوچک، خطای بزرگ، خطای منفی داریم. در ضمن کیشان یک عدد هم *assign* کرد، به عبارتی غیر از توصیف کلامی یک عدد هم کنارش گذاشت. این کار را برای این انجام داد که می خواست مفهومی به نام *mean effective value* را معرفی کند. با توجه به اینکه تاریخچه متغیر را در L مرحله قبل داریم بررسی می کنیم که در L مرحله قبل این متغیر به کدام تابع تعلق بیشتر داشته، که یک سری عددی بدست می آید و این اعداد متوسط می گیریم. به عبارت دیگر برای P ، *time* گذشته از آن متغیر متوسط می گیریم که یک عددی تولید می شود که بر اساس آن می توانیم قوانین را بدست آوریم.

کیشان از یک شبکه *Feed Forward* همراه با الگوریتم *BP* برای *Classification* استفاده کرد و الگوهایی را هم که برای آموزش استفاده نمود به صورت زیر بودند. (رفتار نوسانی - *overshot*، خوب، ایده آل - خطای دائمی - رفتار واگرا)





در این مثال $50 = 10 \times 5$ تا الگو را به شبکه عصبی داده شد تا ۵ کلاس را بتواند شناسایی کند. سپس گیشان بر اساس این رفتار *scaling mechanism* اش را به این صورت تعریف نمود که اگر سیستم *Diverts* یا *converge* می کند *universe* را باز یا کوچک کند. و به این ترتیب اگر هر کدام از این کلاس ها به میزانی تحقق داشته باشند *univers* ها و δ ها را بزرگ یا کوچک می کنیم.

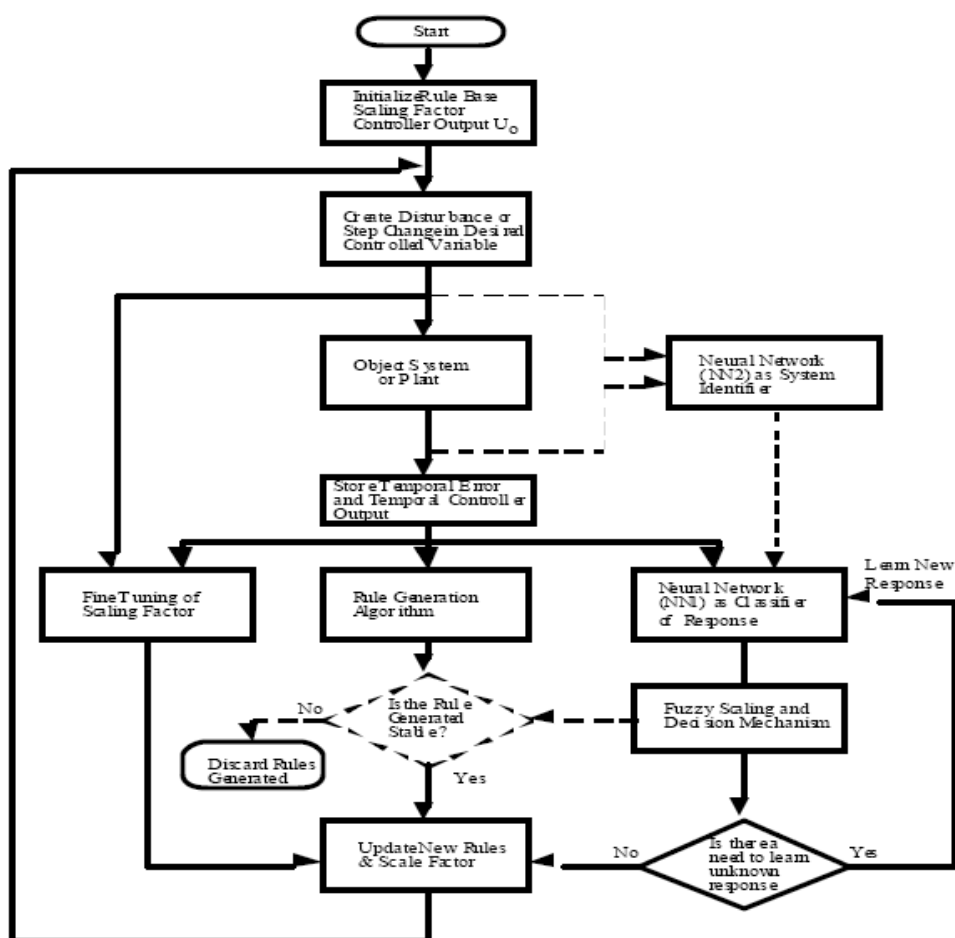
در ابتدا هیچ قانونی نداریم. در مرحله بعد ۲ تا قانون تولید شد سپس ۹ تا ۱۷ و ... قانون تولید شد. بعد از مدتی قوانین بکار نمی روند و تعدادی از آنها را حذف می شدند تا اینکه به ۲۳ تا قانون همگرا شد و در واقع سیستم یاد می گیرد تا موتور *Direct Drive* را کنترل کند.

بحث ما ترکیب *Fuzzy Logic* با *NN* بود در این روشی که این جا معرفی شد از شبکه های عصبی برای *classifier* و شناسایی سیستم استفاده کردیم. شبکه عصبی نقشش اینجا این است که رفتارهای سیستم را مشاهده کند و به ما بگوید که این رفتار یک رفتار واگراست یا واگرا، همگراست همراه با نوسان، *overshot*، ایده ال، خطای دائمی و ... است و کار سیستم فازی این است که سیستم را اجرا کند و قوانین را بوجود آورند. در این ساختار شبکه های عصبی در حقیقت ورودی *scale encatible* ارائه می کنند یعنی در حقیقت با این که رفتار

سیستم *asolitory* هست یا اینکه همگرا است یا واگرا است و یا ... است فضای مسئله را ما می توانیم باز یا کوچک کنیم. در تعریف قوانین فازی دو چیز مهم است یکی قوانین یکی توابع عضویت.

در این روش قوانین از طریق *Fuzzy Generation algorithm* بوجود آمده اند. و برای توابع عضویت به صورت تک تک، پارامترهایشان را بهینه نمی کنیم بلکه کل این فاز را باز یا بسته کنیم. به همین دلیل به آن *Fuzzy scaling mechanism* می گوئیم.

آنچه که گیشان انجام داد به این صورت است که در L مرحله قبل تاریخچه سیستم بررسی می شود تا مشخص شود که به طور متوسط خطای سیستم فرضا صفر است یا مثبت و یا منفی است و به آن یک عدد تعلق می گیرد. به عبارتی دیگر به هر MF اعدادی مانند ۰ و ۱ و ۲ و ... تخصیص داد. و آنها را برای زمانهای مختلف با هم متوسط گرفت و در نهایت با هم جمع کرد. این در کنار *Scaling mechanism* خیلی می تواند کارا تر شود. در *Scaling mechanism* رفتار زمانی سیستم مشاهده و برای آن یک ورودی به *Scaling mechanism* وارد می کنیم.



قوانین هم به این ترتیب تولید می شوند که به عنوان مثال اگر در بین متغیرهای مختلفی که داریم فقط *classifier* ای که مربوط به الگوی *oscilating* می گوید که سیستم *oscilating* است و سیستم *stady state error* ندارد. نشان می دهد که فضای *integral error* را نباید تغییر دهیم و فضای *Direcy Feedback* را باید بزرگ تر کنیم. در حقیقت باید *gain* اش را زیادتر یا کوچک تر کنیم. (اگر بزرگ ترمی کنیم *gain* اش را کوچک تر و اگر کوچک تر می کنیم *gain* اش را بزرگتر کرده ایم).

به عبارت دیگر حساسیت سیستم را نسبت به بزرگ یا کوچک شدنش زیاد یا کم کرده ایم.

$$\begin{aligned} \text{IF } A \text{ is } R_a^{Nj} \text{ and } B \text{ is } R_b^{Nj} \text{ and } C \text{ is } R_c^{Nj} \text{ and } D \text{ is } R_d^{Nj} \\ \text{and } E \text{ is } R_e^{Nj} \text{ and } F \text{ is } R_f^{Nj} \text{ then} \\ \Delta\alpha_1 \text{ is } I_{\alpha_1}^{Nj} \text{ and } \Delta\alpha_2 \text{ is } I_{\alpha_2}^{Nj} \text{ and} \\ \Delta\alpha_3 \text{ is } I_{\alpha_3}^{Nj} \text{ and } \Delta\beta \text{ is } O_{\beta}^{Nj} \end{aligned}$$

$$\begin{aligned} \alpha_1(T+1) &= \alpha_1(T) + \Delta\alpha_1(T) \\ \alpha_2(T+1) &= \alpha_2(T) + \Delta\alpha_2(T) \\ \alpha_3(T+1) &= \alpha_3(T) + \Delta\alpha_3(T) \\ \beta(T+1) &= \beta(T) + \Delta\beta(T) \end{aligned}$$

این در حقیقت *Fuzzy scaling mechanism* است که در نهایت در *New cycle* های مختلف فرآیند، کنترل سیستم را یاد می گیرد. در *New cycle* اول تعداد قوانین ۲ تا است و با گذشت زمان تعداد قوانین زیاد می شود سپس به یک حدی می رسد که به دلیل اینکه بعضی از الگوهای رفتاری برای آن پیش نمی آید فراموش می شود و تعداد قوانین کمتر می شود و در نهایت به یک حالت *stady state* می رسد.

در ابتدا رفتار سیستم چیزی جزء حرج و مرج نیست ولی صرفاً با ۲۰ تا *cycle* که زمان کوتاهی است سیستم می تواند بدون هیچ قانونی، خودش را یاد بگیرد. بنابراین این یک *Fuzzy Neural Network* می باشد (ترکیب فازی و *NN*) به ترتیبی که هر دو ماهیت خودشان را جداگانه دارند فقط در یک ساختار در کنار هم قرار می گیرند تا از مزیتشان استفاده شود.



Tuning Cycle	Number of Rules
1	2
2	4
3	9
4	13
5	17
6	12
7	16
8	22
9	26
10	29
11	32
12	11
13	16
14	19
15	22
16	22
17	23
18	23
19	23
20	22

حالت بعدی ترکیب *Fuzzy* و *NN* است به ترتیبی که در هم آمیخته می شوند مثلاً در *Fuzzy Neural Network*، فازی یک صفت *NN* می شود، نرون هایی که فازی عمل می کنند یا در *Neuro Fuzzy system*، *Neuro* یک صفت برای *Fuzzy* می شود سیستم فازی که مثل نرون عمل می کند که هر یک مزایایی دارند.

به عنوان مثال اگر بتوانیم یک سیستم *Neuro Fuzzy* داشته باشیم که ماهیت فازی اش را حفظ کند ولی مثل شبکه عصبی عمل کند، مزایایی می تواند داشته باشد. مزیت آن این است اگر سیستم فازی ماهیتش را حفظ کند (فازی باشد) بنابراین برای تعریف آن می توانیم از دانش انسانی استفاده نماییم و اگر مثل نرون عمل می کند حال شاید بتوانیم از الگوریتم های آموزش شبکه عصبی که بسیار هم هستند برای آموزش آن استفاده نماییم.

وقتی که *Jang* در اواسط دهه ۹۰ این کار را انجام داد روش های بهینه سازی سیستم های فازی خیلی محدود و اندک بودند به عنوان مثال آنچه که ما و تعدادی از محققین روی آن کار می کردیم روی ترکیب الگوریتم ژنتیک با سیستم فازی بود چون که سیستم فازی، لزوماً یک سیستم بهینه نیست به دلیل اینکه بر اساس دانش انسانی است و دانش انسانی هم هیچ اثباتی نیست که بهینه باشد. بنابراین علاقمند بودیم که سیستم فازی را با الگوریتم ژنتیک ترکیب کنیم تا خاصیت بهینه بودن را به آن اضافه نماییم. این کار انجام شد و *GA* توانست فازی را بهینه کند ولی این یک *offline optimazation* بود (بهینه سازی خارج از خط) اگر ما می خواستیم یک بهینه سازی روی خط (*online optimazation*) داشته باشیم به یک روش *adeptive* نیاز داشتیم که سیستم را بتواند تطبیق دهد.

در سیستم های *Anfis* این خاصیت وجود که شبکه های عصبی می توانند به صورت *adeptive* هم عمل کنند گرچه اکثر پیاده سازی های *Anfis* به صورت ساده آن است که همان حالت *offline optimazation* را در بر می گیرد. *Anfis* بیشتر برای *offline classification* استفاده می شود ولی ماهیت *Neuro Fuzzy* سیستم ها یک ماهیت *adeptation* و یک ماهیت *online learning* دارد.

Anfis این مزیت را دارد که چون *Neuro Fuzzy* است، ماهیت *Fuzzy* است که مثل *Neuro* عمل می کند بنابراین هم از دانش انسانی می توان استفاده کرد وهم می توان آن را *adeptive* کنیم اما *Fuzzy Neural Network* شبکه عصبی است که صفت فازی می گیرد.

Fuzzy Neural Network:

شبکه های عصبی ساختارهای موازی (*parallel*) هستند و ترکیبی هستند از عناصر خیلی کوچک تر که هوشمندی در زائده تعامل بین اینها بوجود می آید. می دانیم که شبکه های عصبی خاصیت یادگیری تقریب گر عمومی را دارند.

چگونه می توان صفت فازی را به یک شبکه عصبی اضافه نمود و در کجا می توان از این ساختار استفاده کرد؟

۱- در شبکه عصبی می توان به جای وزن از تابع عضویت استفاده نمود به عبارت دیگر ورودی های شبکه می توانند ورودی فازی باشند. به جای یک عدد دقیق آن را تخمین بزنیم. (گاهی اوقات عدد دقیق را نداریم مجبوریم این کار را انجام دهیم).

۲- همچنین *aggregation* ها می توانند فازی باشند به عبارت دیگر نحوه ترکیب بین اطلاعات، فازی باشند. در شبکه های عصبی معمولی برای ترکیب از حاصل جمع استفاده می شود در حالیکه می توانیم از عملگرهایی مثل اجتماع (*union*) و اشتراک (*intersection*) برای *aggregate* کردن بین اطلاعات استفاده نماییم. فقط بستگی دارد به این که اطلاعات چه هستند، اطلاعات چه مفهومی دارند و چگونه می توانیم ترکیب کنیم.

اگر ورودی های شبکه عصبی مجموعه های فازی (*Fuzzy set*) باشند و داده ها (*Data*) اعتبار آن مجموعه ها (*Fuzzy set*) و وزن ها باشند در آن صورت منطقی است که برای ترکیب بین اطلاعات از *Fuzzy union* یا *Fuzzy intersection* استفاده کنیم. بنابراین می توانیم یک نرون را فازی فای نماییم.

۳- وزن ها فازی باشند.

این که در کدام بخش از کدام توانایی استفاده کنیم به کاربرد مسئله برمی گردد که چه چیزی لازم است؟ آیا ورودی ها با عدم قطعیت توأم است؟ یا اینکه مفهوم *Fuzzy set* را دارد؟ وقتی که با هم ترکیب می شوند آیا *union* آنها را می خواهیم یا *intersection* آنها را می خواهیم؟

اما نحوه ترکیب بین آنها مشخص است، *aggregation* از *Fuzzy operator* ها می تواند استفاده کند یا *weighting operation* ها می توانند فازی باشند یا خود ورودی اگر توأم با عدم قطعیت است می تواند فازی باشد.

در شبکه های عصبی معمولی، ۹۵ درصد *learning* هایی که انجام می شود در قالب *synoptic optimazation* قرار می گیرد یعنی می خواهیم وزن ها را بهینه کنیم در حالیکه یکی از سوالاتی که پیش می آید این است که ما چگونه می توانیم *somatic operation* داشته باشیم؟ ترکیب شبکه عصبی و فازی این باب را هم شاید باز کند که ما بتوانیم در کنار *somatic adaptation, synaptic optimazation* را هم داشته باشیم.

Somatic operation ها شامل مواردی مانند تغییر قوانین، تغییر μ هایی که به هر قانون مربوط می شود یا تغییر نحوه ای که قوانین را اجرا می کنیم می باشد. (تغییرات در ساختار)

synaptic operation: در این روش تعدادی از وزن ها یا تعدادی از پارامتر و اعداد را تغییر می دهیم بنابراین شامل مواردی مانند *shifting*، حرکت دادن تابع عضویت، تغییر بازوی یک تابع عضویت یا تغییر شکل تابع عضویت می باشد.

۵-۱۲. *Anfis*:

Anfis یک نوع ترکیب بین *Neuro* و فازی است به طوریکه این *Neuro* یک صفت برای سیستم فازی است ماهیت هنوز فازی است. به عبارت دیگر *Anfis* هنوز یک سیستم فازی است فقط در قالب *Neuro* نشان داده می شود که مخفف عبارت زیر می باشد:

Adaptive Network Fuzzy inference system

افرادی هم آن را مخفف عبارت زیر می دانند:

Adaptive Neuro Fuzzy inference system

دلیل اینکه روی *Network base* تاکید می کنیم به خاطر ساختار شبکه ای است که در شبکه های عصبی وجود دارد.

Adaptive همان *Online optimazation* و *online learning* است.

تفاوت *adapt learning* و *optimazation* چیست؟

Optimization عموماً *offline* است و *learning* ، *online* است.

تشابه *adapt learning* و *optimazation* چیست؟

optimazation بهینه سازی تدریجی است و به تدریج سعی می کند با تطبیق خود را بهینه کند.

Learning بهینه سازی *online* است یک دفعه خود را دفعه بهینه می کند.

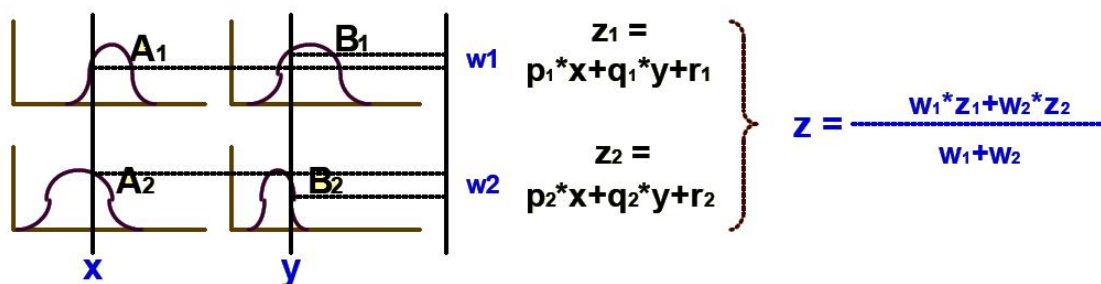
Neuro Fuzzy می تواند *adaptive* یا *Nonadaptive* باشد. بنابراین *Neuro Fuzzy* ، *adaptive Neuro*

Fuzzy راهم در بر می گیرد. به عبارت دیگر بحث *Anfis* یک بحث جزئی تری از *Neuro Fuzzy* است.

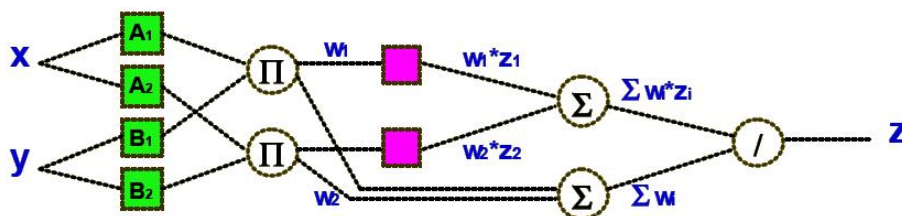
تفاوت *Neuro Fuzzy* و *Anfis* چیست؟

در *Anfis* ، سیستم فازی و شبکه عصبی با یکدیگر ترکیب ویکی شده اند اما در *Neuro- Fuzzy* در عین حال که هر دو تاسیستم فازی و شبکه عصبی را در یک ساختار داریم ولی ماهیتشان جداگانه است.

مثال :



■ ANFIS (Adaptive Neuro-Fuzzy Inference System)



اگر مثال بالا را در نظر بگیرید یک نوع قانون فازی از نوع *sugeno type 1* است زیرا قسمت تالی آن یک فرمول ریاضی است در این مثال دو ورودی x, y داریم میزان مطلوب آنها را بررسی کرده و *and* شان را بدست می آوریم سپس به آن میزان خروجی ها را وزن می دهیم و در نهایت خروجی سیستم فازی جمع وزنی تک تک خروجی های می شود.

یکی از موارد جالب این است که آقای *jang*، *adaptive Neuro Fuzzy* را به صورت شبکه ای نوشته است که قوانین کاملاً مشخص هستند.

قانون اول: $\text{if } x1 \text{ is } A1 \text{ and } y \text{ is } B1 \text{ then } w1$

قانون دوم: $\text{if } x1 \text{ is } A2 \text{ and } y \text{ is } B2 \text{ then } w2$

برای Min از $w1, w2$ استفاده کرده و خروجی $w1, w2$ شده است.

بنابراین $w1z1, w2z2$ هم به راحتی محاسبه می شود تا اینجا اگر ضرب (*) داشته باشیم همه علائم وجود دارند کافی است دو تای صورت را با هم جمع کرده تقسیم بر مخرج نماییم وقتی $Jang$ قانون Tsk را به این صورت نشان داد گفت این شبیه شبکه عصبی است و من می توانم BP برای آن استفاده کنم؟

اما برای BP باید خطا را در خروجی بگیرد و آن را $Back Propagate$ یا انتشار به عقب کند، ولی این کار را نتوانست انجام دهد چون در قسمت آخر تقسیم داریم (در شکل صفحه قبل) و بنابراین به مشکل برمی خورد. لذا اگر این کار را نمی توان انجام داد شاید کار دیگری بتوان انجام داد گفت بیایم قسمت جلو را با $least square, optimz$ کنیم و خطا را بگیریم و این را پس انتشار نماییم و بنابراین الگوریتم $Anfis$ ترکیب بین $least square$ و BP شد. در حقیقت از همان الگوریتم های آموزش شبکه عصبی استفاده نمود اما این آموزش فقط بایستی در ۲ مرحله انجام شود. و به این ترتیب روشی به نام $Anfis$ را معرفی کرد که خیلی ها از آن استفاده می کنند.

Fuzzy Neural Network and Neuro-Fuzzy Modeling

مثال کاربردی: در سال ۹۴ برای شناسایی حروف تاییپ کاری انجام شد. که از یک متن بتواند حروف را شناسایی کند کاری که انجام داد یک شبکه $Fuzzy-Neuron$ را معرفی کرد- این شبکه ۴۱۰ تا لایه دارد ۲ تا لایه اولیه ۲ بعدی هستند چون تصویر ۲ بعدی است i index و j index دارد هرورودی شبکه یک پیکسل از تصویر می باشد.

کار لایه اول: این است که اطلاعات را، تصویر را نرمالایز کند یعنی بنا بر ماکزیمم قدرتی که توی $intensity$ وجود دارد همه پیکسل ها را بر آن تقسیم کنیم تا همه پیکسل ها بین صفر و یک قرار بگیرند. این کار را می کند چون $universe$ ها با هم متفاوت هستند باید بتواند سیستم در مقابل نورهای مختلف $Robust$ باشد.

کار لایه دوم: ورودی نرمالایز شده را با الگوهای از پیش آموزش داده شده، $Match$ می کند. ما به ازای هر پیکسل و به ازای هر الگویی که در سیستم مورد آموزش قرار گرفته، یک نرون داریم و آن نرون کارش این است که هر وقت ورودی با آن نرون تطابق پیدا کرد به عبارت دیگر ما دو تا MF داریم یکی برای نرون $Max Fuzzy Neuron$

است $(Max-fn)$ input function هم حالا یک بازه $single ton$ باشد به ازای هر پیکسل با اندیس i, j, k = تعداد پیکسل های افقی j = تعداد پیکسل های عمودی k = تعداد الگوها) ما یک همچنین تابع عضویتی را آموزش می دهیم آموزشش خیلی ساده است فقط با مقادیرمان $assign$ می کنیم. بنابراین این شناختش- حالا یک تصویر

آمده نرمالایز شده و تک تک این نرون ها می گویند این با ما با این میزان تطابق دارد. فرض کنید اگر یک تصویر 3×3 را در نظر بگیریم هر ۹ نرون میزان تطابق را نشان می دهد من یک الگویی را دارم برای آن در نظر می گیرم آن الگویی رو که این نرون ها دارند اعلام تطابق می کنند رو چگونه بدست بیاورم؟ به چه میزان آیا این با آن الگو تطابق دارد. آقای *Min kwan* گرفته، و می گوید به میزان فرضاً 0.3 این الگو با آن تطابق دارد. بنابراین *Min* به تعداد الگوهایش است. بنابراین خروجی لایه سوم میزان تطابق آن تصویر است با آن الگویی که از پیش تعریف شده است. حالا کدامیک از این الگوها برنده هستند؟ آن نرون یا الگویی که بیشترین قطعیت را دارد- خیلی ساده است- آموزشش هم خیلی ساده است. یعنی هر الگویی را که برای شما بوجود می آید چه کار می کنید؟ کافی است فقط بر روی لایه وسط *MF* تعریف کنید.

حالا این که چقدر این *discrimet* بتواند بکند چقدر آیا باید تصویر متفاوت باشد و هنوز این شبکه عصبی فازی فای ما بتواند تشخیص بدهد برمی گردد به این که این بازه *MF* ما چقدر باز یا بسته باشد اگر خیلی بسته باشد ممکن است که نتواند درست تشخیص دهد اگر خیلی باز باشد باز هم ممکن است که نتواند درست تشخیص دهد. ولی برای این درست تشخیص نمی دهد که همه (نرون ها) می گویند قبلاً این تصویر را ندیده ایم و دومی به خاطر اینکه همه (نرون ها) می گویند که ما این را قبلاً دیده ایم. بنابراین این مقدار خیلی مهم است که بهینه تعیین شود آموزش این خیلی راحت است شما کافی است الگوهای جدید را بیاورید و نرمالایزش کنید و بعداً بیایید بر اساس آن چند تا نرون بوجود بیاورید؟ به تعداد پیکسل هایش- اگر $n * m$ است به تعداد $n * m$ تا پیکسل برایش تعریف کنیم. این می شود آموزشش و او بعد از آن می تواند یاد بگیرد. سرعت آموزش خیلی بالا است زیرا *error* ای نداریم که بیائیم هر دفعه *error* را بهینه کنیم. درجه الگو را به آن می دهید کاملاً مشخص است که چگونه باید باشد.

(نرون ها در لایه دوم ۳ تا اندیس i, j, k دارند $i =$ تعداد پیکسل های افقی $j =$ تعداد پیکسل های عمودی $k =$ تعداد الگوها) بنابراین لایه اول سیگنال xi, j را می گیرد xij را تقسیم می کند بر $Prmax$ (بزرگ ترین تابع *intensity* *light*) در آن تصویر- و آنچه که بدست می آید yi, j است که نرمالایز xi, j از لحاظ *intensity* است در *step* بعدی این را فازی فای کرده و مقایسه می کنیم یک تابع *exp* برای وزن تعریف می کنیم. به این ترتیب این را فاز فای می کنیم و *match* اش می کنیم با yp, q, m - اینجا *MF* را به جای گوسی مثلی گرفته است و هر چقدر α بزرگ تر باشد کمتر می توانند *DISCRIMINIT* کنند فکر می کنند همه چیز مثل خودشان هستند. و در نهایت *min layer* کارش این است که میزان تطابق الگو را شناسایی کند. بنابراین بین *index i* و *index j*، *minimom* می گیرد (یا p, q اینجا توضیح داده شده است) و در نهایت *competition layer* به آن نرونی که بیشترین قطعیت را دارد را برنده اعلام می کند. این روش *self organizing* است چون که خودش به طور

خودکار نرون بوجود می آورد وقتی می خواهید آموزشش بدهید یک روش *self organazing* است ساختارش را خودش تعیین می کند.

