

بسمه تعالی



گزارش چهارم

**Meta-Learning**

استاد راهنما: دکتر قدیری

دانشجو: پردیس مرادیکی

تابستان ۱۴۰۲

## آنچه این گزارش پوشش می دهد:

فصل ۱، مقدمه‌ای بر فرا یادگیری، به ما کمک می کند تا بفهمیم متا یادگیری چیست و انواع مختلف یادگیری متا را پوشش می دهد. ما همچنین یاد خواهیم گرفت که چگونه یادگیری متا با یادگیری از چند نقطه داده از یادگیری چند شات استفاده می کند. سپس خواهیم دید که چگونه با نزول گرادیان آشنا شویم. بعداً در این فصل، بهینه سازی را به عنوان مدلی برای تنظیمات آموزش چند شات خواهیم دید.

فصل ۲، تشخیص چهره و صدا با استفاده از شبکه‌های سیامی، با توضیح اینکه شبکه‌های سیامی چیست و چگونه شبکه‌های سیامی در تنظیمات آموزش تک شات استفاده می شوند، شروع می شود. ما به معماری یک شبکه سیامی و برخی از کاربردهای شبکه سیامی خواهیم پرداخت. سپس نحوه استفاده از شبکه های سیامی را برای ساخت مدل های تشخیص چهره و صدا خواهیم دید.

فصل ۳، شبکه‌های اولیه و انواع آنها، توضیح می دهد که شبکه‌های نمونه اولیه چیست و چگونه در سناریوهای یادگیری چند شات استفاده می شوند. خواهیم دید که چگونه می توان یک شبکه نمونه اولیه برای انجام طبقه بندی در مجموعه کاراکترهای omniglot ساخت. بعداً در این فصل، به انواع مختلف شبکه‌های اولیه، مانند شبکه‌های نمونه اولیه گاوسی و شبکه‌های نیمه نمونه اولیه نگاه خواهیم کرد.

فصل ۴، شبکه های ارتباط و تطبیق با استفاده از TensorFlow، به ما کمک می کند تا معماری شبکه رابطه و نحوه استفاده از شبکه رابطه در تنظیمات یادگیری تک شات، چند شات و صفر شات را درک کنیم. سپس خواهیم دید که چگونه با استفاده از TensorFlow یک شبکه رابطه بسازیم. در ادامه با شبکه تطبیق و معماری آن آشنا خواهیم شد. همچنین جاسازی‌های متنی کامل و نحوه ایجاد یک شبکه منطبق با استفاده از TensorFlow را بررسی خواهیم کرد.

فصل ۵، شبکه‌های عصبی تقویت‌شده حافظه، ماشین‌های تورینگ عصبی (NTM) چیست و چگونه از حافظه خارجی برای ذخیره و بازیابی اطلاعات استفاده می‌کنند. ما مکانیسم‌های آدرس دهی مختلف مورد استفاده در NTM ها را بررسی خواهیم کرد و سپس با شبکه‌های عصبی تقویت شده حافظه و تفاوت آنها با معماری NTM آشنا خواهیم شد.

فصل ۶، MAML و انواع آن، به یکی از الگوریتم‌های متا یادگیری معروف، به نام فرا یادگیری مدل-آگنوستیک (MAML) می‌پردازد. ما بررسی خواهیم کرد که MAML چیست و چگونه در تنظیمات یادگیری تحت نظارت و تقویتی استفاده می‌شود. همچنین نحوه ساخت MAML را از ابتدا خواهیم دید. سپس، با یادگیری متا مخالف و CAML آشنا می‌شویم که برای تطبیق سریع زمینه در یادگیری متا استفاده می‌شود.

فصل ۷، Reptile و Meta-SGD، توضیح می‌دهد که چگونه meta-SGD برای یادگیری همه اجزای الگوریتم‌های نزول گرادیان، مانند وزن‌های اولیه، نرخ‌های یادگیری و جهت به روز رسانی استفاده می‌شود. ما نحوه ساخت متا SGD را از ابتدا خواهیم دید. بعداً در این فصل، با الگوریتم خزندگان آشنا خواهیم شد و خواهیم دید که چگونه به عنوان یک بهبود نسبت به MAML عمل می‌کند. همچنین خواهیم دید که چگونه از الگوریتم خزنده برای رگرسیون موج سینوسی استفاده کنیم.

فصل ۸، توافق گرادیان به عنوان یک هدف بهینه‌سازی، نحوه استفاده از توافق گرادیان را به عنوان یک هدف بهینه‌سازی در محیط فرا یادگیری پوشش می‌دهد. ما یاد خواهیم گرفت که توافق گرادیان چیست و چگونه می‌تواند الگوریتم‌های یادگیری متا را افزایش دهد. در ادامه این فصل، نحوه ساخت یک الگوریتم توافق گرادیان را از ابتدا یاد خواهیم گرفت.

فصل ۹، پیشرفت‌های اخیر و گام‌های بعدی، با توضیح متا یادگیری وظیفه‌شناسی شروع می‌شود و سپس خواهیم دید که چگونه فرا یادگیری در یک محیط یادگیری تقلیدی استفاده می‌شود. سپس، یاد خواهیم گرفت

که چگونه می توانیم MAML را در یک تنظیمات یادگیری بدون نظارت با استفاده از الگوریتم CACTUS اعمال کنیم. سپس، یک الگوریتم فرا یادگیری عمیق به نام یادگیری برای یادگیری در فضای مفهومی را بررسی خواهیم کرد.

## فصل دوم:

در فصل آخر با متا یادگیری چیست و انواع تکنیک های فرا یادگیری آشنا شدیم. ما همچنین دیدیم که چگونه می توان نزول گرادیان را با نزول گرادیان و بهینه سازی به عنوان مدلی برای یادگیری چند شات یاد گرفت. در این فصل، یکی از متداول ترین الگوریتم های یادگیری تک شات مبتنی بر متریک به نام شبکه های سیامی را یاد خواهیم گرفت. خواهیم دید که چگونه شبکه های سیامی از نقاط داده بسیار کمی یاد می گیرند و چگونه از آنها برای حل مشکل داده کم استفاده می شود. پس از آن معماری شبکه های سیامی را به تفصیل بررسی خواهیم کرد و برخی از کاربردهای شبکه های سیامی را مشاهده خواهیم کرد. در پایان این فصل، نحوه ساخت مدل های تشخیص چهره و صدا با استفاده از شبکه های سیامی را یاد خواهیم گرفت.

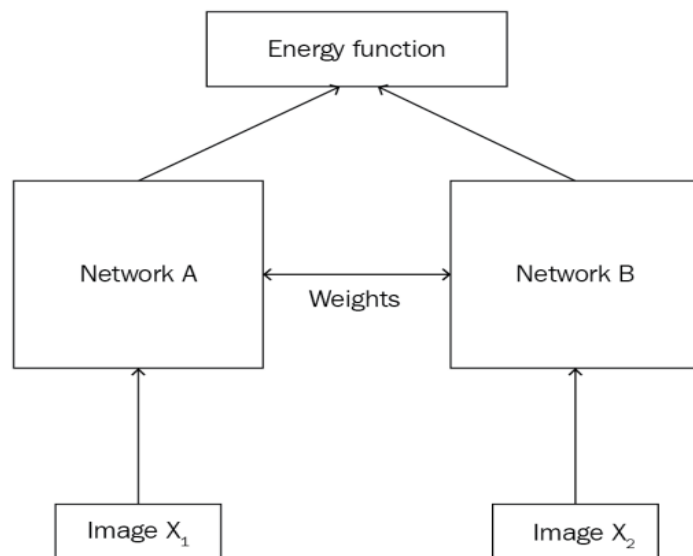
در این فصل موارد زیر را خواهید آموخت:

۱. شبکه های سیامی چیست؟
۲. معماری شبکه های سیامی
۳. کاربردهای شبکه های سیامی
۴. تشخیص چهره با استفاده از شبکه های سیامی
۵. ساخت یک مدل تشخیص صدا با استفاده از شبکه های سیامی

### **شبکه های سیامی چیست؟**

شبکه سیامی نوع خاصی از شبکه عصبی است و یکی از ساده ترین و پرکاربردترین الگوریتم های یادگیری تک شات است. همانطور که در فصل قبل آموختیم، یادگیری تک شات تکنیکی است که در آن فقط از یک مثال آموزشی در هر کلاس یاد می گیریم. بنابراین، شبکه سیامی عمدتاً در برنامه هایی استفاده می شود که در آن نقاط داده زیادی در هر کلاس نداریم. به عنوان مثال، فرض کنید می خواهیم یک مدل تشخیص چهره برای سازمان خود بسازیم و حدود ۵۰۰ نفر در سازمان ما کار می کنند. اگر بخواهیم مدل تشخیص چهره خود را با استفاده از یک شبکه عصبی کانولوشنال (CNN) از ابتدا بسازیم، برای آموزش شبکه و دستیابی به دقت خوب به

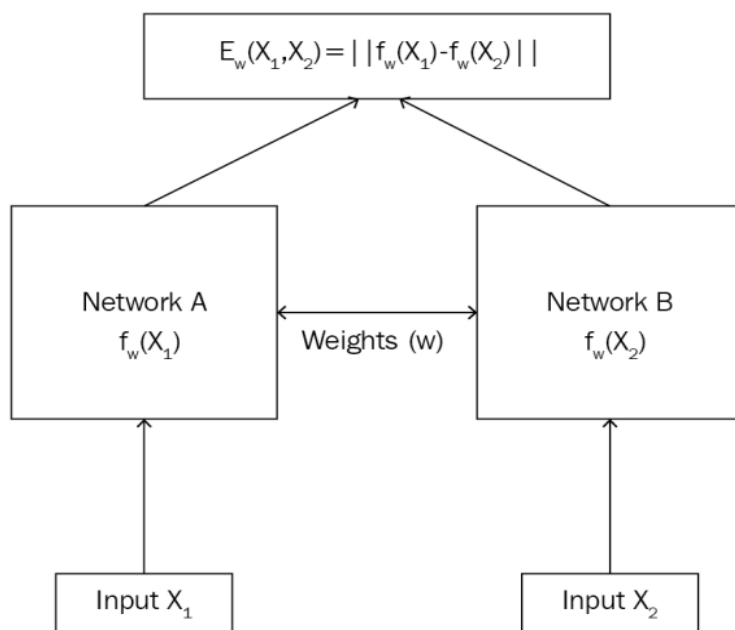
تصاویر زیادی از همه این ۵۰۰ نفر نیاز داریم. اما ظاهراً ما تصاویر زیادی برای همه این ۵۰۰ نفر نخواهیم داشت و بنابراین ساختن یک مدل با استفاده از CNN یا هر الگوریتم یادگیری عمیق امکان پذیر نیست، مگر اینکه نقاط داده کافی داشته باشیم. بنابراین، در این نوع سناریوها، می‌توانیم به یک الگوریتم یادگیری یک‌شات پیچیده مانند شبکه سیامی متوسل شویم که می‌تواند از نقاط داده کمتری یاد بگیرد. اما شبکه های سیامی چگونه کار می کنند؟ شبکه‌های سیامی اساساً از دو شبکه عصبی متقارن تشکیل شده‌اند که وزن‌ها و معماری یکسانی دارند و هر دو در انتها با استفاده از تابع انرژی به هم متصل شده‌اند. فرض کنید دو تصویر  $X_1$  و  $X_2$  داریم و می‌خواهیم بفهمیم که این دو تصویر مشابه یا غیرمشابه هستند. همانطور که در نمودار زیر نشان داده شده است، تصویر  $X_1$  را به شبکه A و تصویر  $X_2$  را به شبکه B دیگر تغذیه می کنیم. نقش هر دوی این شبکه ها ایجاد تعبیه ها (بردارهای ویژگی) برای تصویر ورودی است. بنابراین، ما می توانیم از هر شبکه ای که به ما تعبیه می کند استفاده کنیم. از آنجایی که ورودی ما یک تصویر است، می‌توانیم از یک شبکه کانولوشنال برای تولید جاسازی‌ها، یعنی برای استخراج ویژگی‌ها استفاده کنیم. به یاد داشته باشید که نقش CNN در اینجا فقط استخراج ویژگی ها است و نه طبقه بندی. همانطور که می دانیم این شبکه ها باید وزن و معماری یکسانی داشته باشند، اگر شبکه A ما یک CNN سه لایه است، شبکه B ما نیز باید یک CNN سه لایه باشد و باید از مجموعه وزن های یکسانی برای هر دوی اینها استفاده کنیم. شبکه های. بنابراین، شبکه A و شبکه B به ترتیب تعبیه‌هایی را برای تصاویر ورودی  $X_1$  و  $X_2$  به ما می‌دهند. سپس، این تعبیه‌ها را به تابع انرژی تغذیه می‌کنیم، که به ما می‌گوید این دو ورودی چقدر شبیه هم هستند. توابع انرژی اساساً هر معیار تشابهی هستند، مانند فاصله اقلیدسی و شباهت کسینوس.



شبکه‌های سیامی نه تنها برای تشخیص چهره استفاده می‌شوند، بلکه در برنامه‌هایی که در آن نقاط داده و وظایف زیادی نداریم که باید شباهت بین دو ورودی را بیاموزیم، به طور گسترده استفاده می‌شوند. کاربردهای شبکه‌های سیامی شامل تأیید امضاء، بازیابی سؤالات مشابه، ردیابی اشیاء و موارد دیگر است. شبکه‌های سیامی را در بخش آینده به طور مفصل مطالعه خواهیم کرد.

## معماری شبکه‌های سیامی

اکنون که درک اولیه‌ای از شبکه‌های سیامی داریم، آنها را با جزئیات بررسی خواهیم کرد. معماری یک شبکه سیامی در نمودار زیر نشان داده شده است:



همانطور که در نمودار قبلی مشاهده می‌کنید، یک شبکه سیامی از دو شبکه یکسان تشکیل شده است که وزن و معماری یکسانی دارند. فرض کنید دو ورودی  $X_1$  و  $X_2$  داریم. ورودی  $X_1$  خود را به شبکه A، یعنی  $f_w(X_1)$  و ورودی  $X_2$  خود را به شبکه B، یعنی  $f_w(X_2)$  می‌دهیم. همانطور که متوجه خواهید شد، هر دوی این شبکه‌ها وزن‌های یکسانی دارند،  $w$ ، و جاسازی‌هایی را برای ورودی‌ها،  $X_1$  و  $X_2$  ایجاد می‌کنند. سپس، این تعبیه‌ها را به تابع انرژی  $E$  می‌دهیم، که شباهت بین دو ورودی را به ما می‌دهد.

می‌توان آن را به صورت زیر بیان کرد:

$$E_W(X_1, X_2) = ||f_W(X_1) - f_W(X_2)||$$

فرض کنید از فاصله اقلیدسی به عنوان تابع انرژی خود استفاده می کنیم، اگر  $X_1$  و  $X_2$  مشابه باشند، مقدار  $E$  کمتر خواهد بود. اگر مقادیر ورودی متفاوت باشند، مقدار  $E$  بزرگ خواهد بود.

فرض کنید شما دو جمله دارید، جمله ۱ و جمله ۲. ما جمله ۱ را به شبکه  $A$  و جمله ۲ را به شبکه  $B$  می دهیم. فرض کنید هر دو شبکه  $A$  و شبکه  $B$  ما شبکه های LSTM هستند و وزن های یکسانی دارند. بنابراین، شبکه  $A$  و شبکه  $B$  به ترتیب برای جمله ۱ و جمله ۲ واژه های embedding را ایجاد می کنند. سپس، این تعبیه ها را به تابع انرژی می دهیم که امتیاز شباهت بین دو جمله را به ما می دهد. اما چگونه می توانیم شبکه های سیامی خود را آموزش دهیم؟ داده ها چگونه باید باشد؟ ویژگی ها و برجسب ها چیست؟ کارکرد هدف ما چیست؟

ورودی شبکه های سیامی باید به صورت جفت،  $(X_1, X_2)$ ، همراه با برجسب باینری آنها،  $Y \in (0, 1)$  باشد، که بیان می کند که جفت های ورودی یک جفت واقعی (همان) یا یک جفت غیرمجاز (متفاوت) هستند. همانطور که در جدول زیر می بینید، ما جملات را به صورت جفت داریم و برجسب نشان می دهد که جفت جملات اصل (۱) هستند یا غیرمجاز (۰):

Sentence pairs		Label
She is a beautiful girl	She is a gorgeous girl	1
Birds fly in the sky	What are you doing ?	0
I love Paris	I adore Paris	1
He just arrived	I am watching a movie	0

بنابراین، **loss function** شبکه سیامی ما چیست؟ از آنجایی که هدف شبکه سیامی انجام یک کار طبقه بندی نیست، بلکه درک شباهت بین دو مقدار ورودی است، از **contrastive loss function** استفاده می کنیم. می توان آن را به صورت زیر بیان کرد:

$$\text{Contrastive Loss} = Y(E)^2 + (1 - Y)\max(\text{margin} - E, 0)^2$$

در معادله قبلی، مقدار  $Y$  برچسب واقعی است، که وقتی دو مقدار ورودی مشابه هستند، ۱ و اگر دو مقدار ورودی غیرمشابه باشند، ۰ خواهد بود، و  $E$  تابع انرژی ما است که می تواند هر اندازه گیری فاصله باشد. از عبارت **margin** برای نگه داشتن محدودیت استفاده می شود، یعنی زمانی که دو مقدار ورودی غیرمشابه هستند و اگر فاصله آنها از حاشیه بیشتر باشد، ضرری متحمل نمی شوند.

### کاربردهای شبکه های سیامی

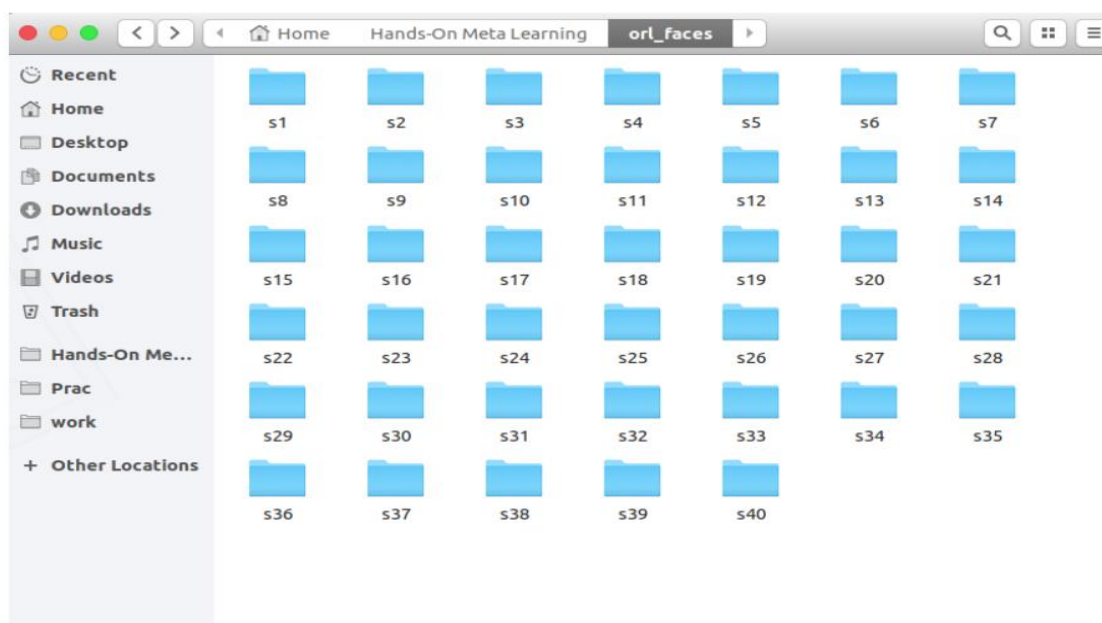
همانطور که فهمیدیم، یک شبکه سیامی با یافتن شباهت بین دو مقدار ورودی با استفاده از معماری یکسان یاد می گیرد. این یکی از متداول ترین الگوریتم های یادگیری چند شات در میان کارهایی است که شامل تشابه محاسباتی بین دو موجودیت است. این شبکه قدرتمند و قوی است و به عنوان راه حلی برای مشکل داده کم عمل می کند. در اولین مقاله ای که شبکه های سیامی در آن منتشر شد (<https://papers.nips.cc/paper/769-signature-verification-using-a-siamese-time-delay-neural-network.pdf>) نویسنده اهمیت شبکه را برای کار تأیید امضا به تصویر می کشد. هدف از کار تأیید امضا، شناسایی صحت امضا است. بنابراین، نویسنده شبکه های سیامی را با جفت های امضای اصیل و تحمیلی آموزش داد و از یک شبکه کانولوشن برای استخراج ویژگی ها از امضا استفاده کرد. پس از استخراج ویژگی ها، آنها فاصله بین دو بردار ویژگی را برای شناسایی شباهت اندازه گیری کردند. بنابراین، وقتی یک امضای جدید وارد می شود، ویژگی ها را استخراج کرده و با بردار ویژگی ذخیره شده امضاکننده مقایسه می کنیم. اگر فاصله کمتر از حد معینی باشد، امضا را معتبر می پذیریم یا امضا را رد می کنیم. شبکه های سیامی نیز به طور گسترده در وظایف NLP استفاده می شوند. مقاله جالبی وجود دارد (<http://www.aclweb.org/anthology/W16-1617>) که در آن نویسندگان از یک شبکه سیامی برای محاسبه تشابه متن استفاده کردند. آنها از شبکه های سیامی به عنوان واحدهای دو جهته و از شباهت کسینوس به عنوان تابع انرژی برای محاسبه شباهت بین متون استفاده کردند. کاربردهای شبکه های سیامی بی



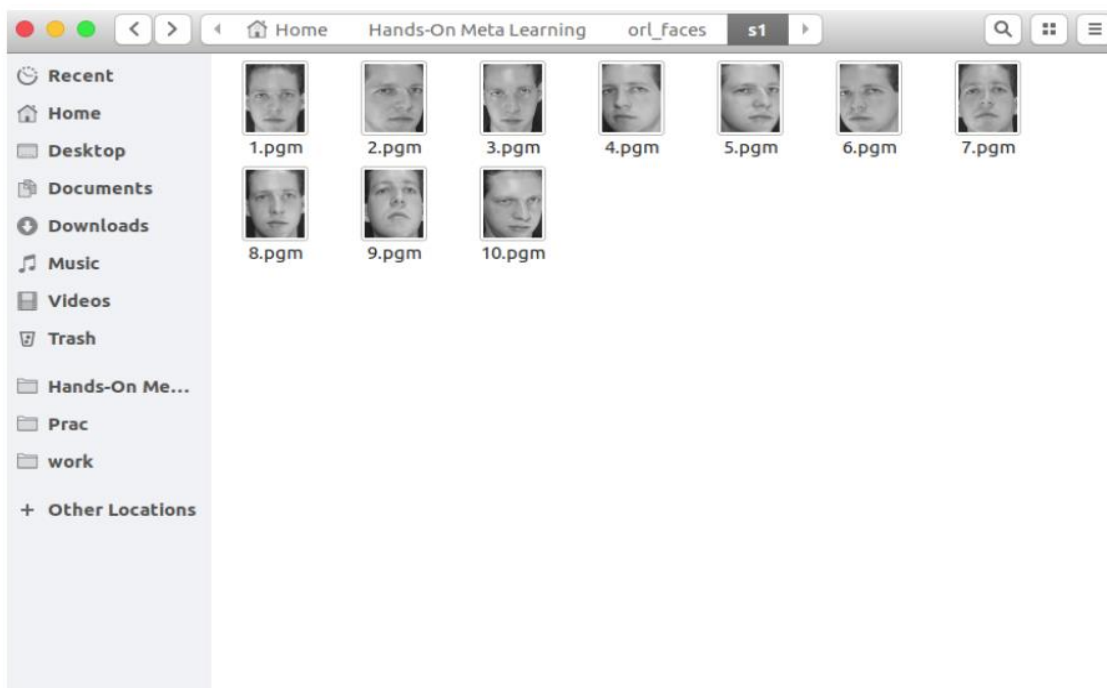
پایان هستند. آنها با معماری های مختلفی برای انجام وظایف مختلف مانند تشخیص اقدامات انسانی، تشخیص تغییر صحنه و ترجمه ماشینی انباشته شده اند.

### تشخیص چهره با استفاده از شبکه های سیامی

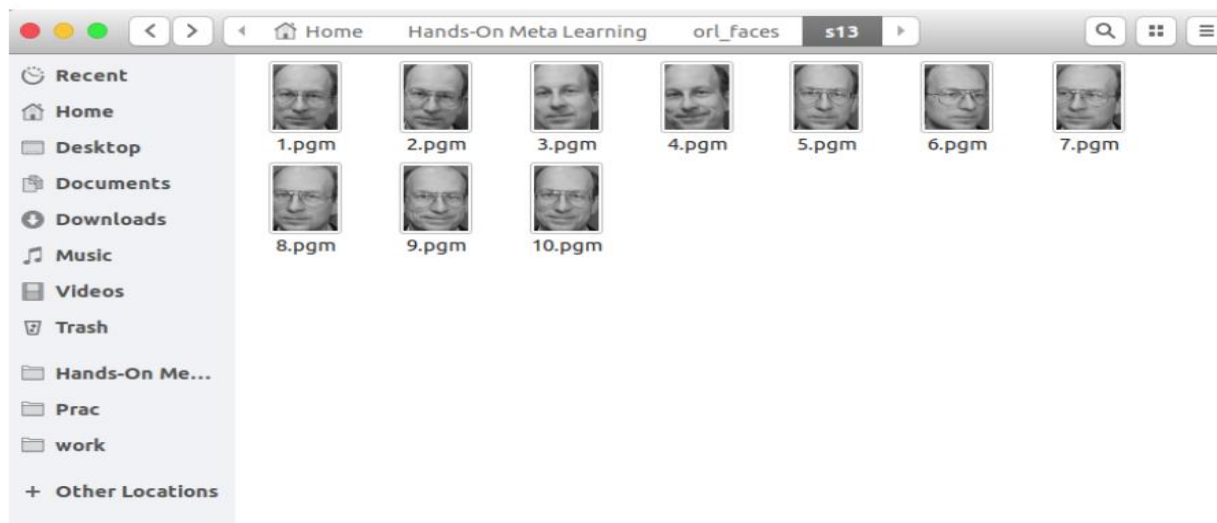
ما شبکه سیامی را با ساخت یک مدل تشخیص چهره درک خواهیم کرد. هدف شبکه ما این است که بفهمیم دو چهره مشابه یا غیرمشابه هستند. ما از پایگاه داده چهره های AT&T استفاده می کنیم که می توانید از اینجا [دانلود کنید](https://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html): <https://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html>. پس از دانلود و استخراج بایگانی، می توانید پوشه های s1، s2، تا s40 را مشاهده کنید، همانطور که در اینجا نشان داده شده است:



هر کدام از این پوشه ها دارای ۱۰ تصویر مختلف از یک فرد است که از زوایای مختلف گرفته شده است. به عنوان مثال، اجازه دهید پوشه s1 را باز کنیم. همانطور که می بینید، ۱۰ تصویر مختلف از یک فرد وجود دارد:




پوشه S13 را باز و بررسی می کنیم:



همانطور که می دانیم شبکه های سیامی به مقادیر ورودی به عنوان یک جفت همراه با برچسب نیاز دارند، باید داده های خود را به این صورت ایجاد کنیم. بنابراین، ما دو تصویر را به طور تصادفی از یک پوشه می گیریم و آنها را به عنوان یک جفت اصلی علامت گذاری می کنیم و از دو پوشه مختلف یک عکس را می گیریم و آنها را به

عنوان یک جفت غیرقابل علامت گذاری می‌کنیم. نمونه ای در تصویر زیر نشان داده شده است. همانطور که می‌بینید، یک جفت واقعی دارای تصاویری از یک شخص است و جفت تحمیلی تصاویری از افراد مختلف دارد:

Input pair		Label
		Genuine
		Imposite
		Genuine
		Imposite

هنگامی که داده های خود را به صورت جفت به همراه برچسب های آنها در اختیار داریم، شبکه سیامی خود را آموزش می دهیم. از جفت تصویر یک تصویر را به شبکه A و تصویر دیگر را به شبکه B می دهیم. نقش این دو شبکه فقط استخراج بردارهای ویژگی است. بنابراین، ما از دو لایه کانولوشن با فعال سازی واحد خطی اصلاح شده (ReLU) برای استخراج ویژگی ها استفاده می کنیم. هنگامی که ویژگی ها را یاد گرفتیم، بردار ویژگی حاصل از هر دو شبکه را به تابع انرژی می دهیم، که شباهت را اندازه گیری می کند. ما از فاصله اقلیدسی به عنوان تابع انرژی خود استفاده می کنیم. بنابراین، شبکه خود را با تغذیه جفت تصویر آموزش می دهیم تا شباهت معنایی بین آنها را بیاموزیم. حالا این را مرحله به مرحله خواهیم دید. برای درک بهتر، می توانید کد کامل را که به عنوان نوت بوک Jupyter با توضیح در اینجا موجود است، بررسی کنید:

<https://github.com/sudharsan13296/Hands-On-Meta-Learning-With-Python/blob/master/02.%20Face%20and%20Audio%20Recognition%20using%20>

Siamese%20Networks/2.4%20Face%20Recognition%20Using%20Siamese%20Network.ipynb.

ابتدا کتابخانه های مورد نیاز را وارد می کنیم:

```
import re

import numpy as np

from PIL import Image

from sklearn.model_selection import train_test_split

from keras import backend as K

from keras.layers import Activation

from keras.layers import Input, Lambda, Dense, Dropout, Convolution2D,
MaxPooling2D, Flatten

from keras.models import Sequential, Model

from keras.optimizers import RMSprop
```

اکنون تابعی برای خواندن تصویر ورودی خود تعریف می کنیم. تابع `read_image` یک تصویر را به عنوان ورودی می گیرد و یک آرایه NumPy را برمی گرداند:

```
def read_image(filename, byteorder='>'):

    #first we read the image, as a raw file to the buffer

    with open(filename, 'rb') as f:

        buffer = f.read()

    #using regex, we extract the header, width, height and maxval of the
image

    header, width, height, maxval = re.search(
```

```

b"(\d+)\s(?:\s*#\s*[\r\n])*"
b"(\d+)\s(?:\s*#\s*[\r\n])*"
b"(\d+)\s(?:\s*#\s*[\r\n])*"
b"(\d+)\s(?:\s*#\s*[\r\n]\s*)", buffer).groups()

```

#then we convert the image to numpy array using np.frombuffer which interprets buffer as one dimensional array

```

return np.frombuffer(buffer,
                      dtype='u1' if int(maxval) < 256 else
                      bytearray+'u2',
                      count=int(width)*int(height),
                      offset=len(header)
                      ).reshape((int(height), int(width)))

```

به عنوان مثال، اجازه دهید یک تصویر را باز کنیم:

```
Image.open("data/orl_faces/s1/1.pgm")
```

وقتی این تصویر را به تابع `read_image` خود وارد می کنیم، به صورت یک آرایه NumPy برمی گردد:

```
img = read_image('data/orl_faces/s1/1.pgm')
```

```
img.shape
```

```
(112, 92)
```

اکنون تابع دیگری به نام `get_data` را برای تولید داده های خود تعریف می کنیم. همانطور که می دانیم، برای شبکه سیامی، داده ها باید به صورت جفت (اصیل و غیرمجاز) با یک برچسب باینری باشند. ابتدا تصاویر

(img1, img2) را از همان دایرکتوری می خوانیم و در آرایه x\_genuine\_pair ذخیره می کنیم و y\_genuine را به ۱ اختصاص می دهیم.

در مرحله بعد، تصاویر (img1, img2) را از دایرکتوری مختلف می خوانیم و آنها را در جفت x\_imposite ذخیره می کنیم و y\_imposite را به ۰ اختصاص می دهیم.

در نهایت، هر دو x\_genuine\_pair و x\_imposite را به X و y\_genuine و y\_imposite را به Y متصل می کنیم:

....

### **Prototypical networks یا شبکه های نمونه اولیه:**







در فصل آخر، یاد گرفتیم که شبکه های سیامی چیست و چگونه برای انجام وظایف یادگیری چند شات استفاده می شوند. ما همچنین نحوه استفاده از شبکه های سیامی را برای انجام تشخیص چهره و صدا بررسی کردیم. در این فصل، الگوریتم یادگیری چند شات جالب دیگری به نام شبکه نمونه اولیه را بررسی خواهیم کرد که توانایی تعمیم حتی به کلاسی را دارد که در یک مجموعه آموزشی وجود ندارد. ما با درک اینکه شبکه های نمونه اولیه چیست شروع می کنیم، پس از آن خواهیم دید که چگونه می توان یک کار طبقه بندی را در یک مجموعه داده omniglot با استفاده از شبکه نمونه اولیه انجام داد. سپس انواع مختلفی از شبکه های نمونه اولیه مانند شبکه های نمونه اولیه گاوسی و شبکه های نیمه نمونه اولیه را مشاهده خواهیم کرد.

در این فصل با موارد زیر آشنا خواهید شد:

۱. شبکه های نمونه اولیه
۲. الگوریتم شبکه های نمونه اولیه
۳. طبقه بندی با استفاده از شبکه های نمونه اولیه
۴. شبکه های نمونه اولیه گاوسی
۵. الگوریتم شبکه نمونه اولیه گاوسی
۶. شبکه نیمه نمونه

### **شبکه های نمونه اولیه**







شبکه های نمونه اولیه، یکی دیگر از الگوریتم های یادگیری ساده، کارآمد و کمی هستند. مانند شبکه های سیامی، یک شبکه نمونه اولیه سعی می کند فضای متریک را برای انجام طبقه بندی بیاموزد. ایده اصلی شبکه های نمونه اولیه ایجاد یک نمایش اولیه از هر کلاس و طبقه بندی یک نقطه پرس و جو (یعنی یک نقطه جدید) بر اساس فاصله بین نمونه اولیه کلاس و نقطه پرس و جو است. فرض کنید یک مجموعه پشتیبانی شامل تصاویر شیر، فیل و سگ داریم که در نمودار زیر نشان داده شده است:

Image ( $x_i$ )	Label ( $y_i$ )
	Lion
	Lion
	Elephant
	Elephant
	Dog
	Dog

Support set

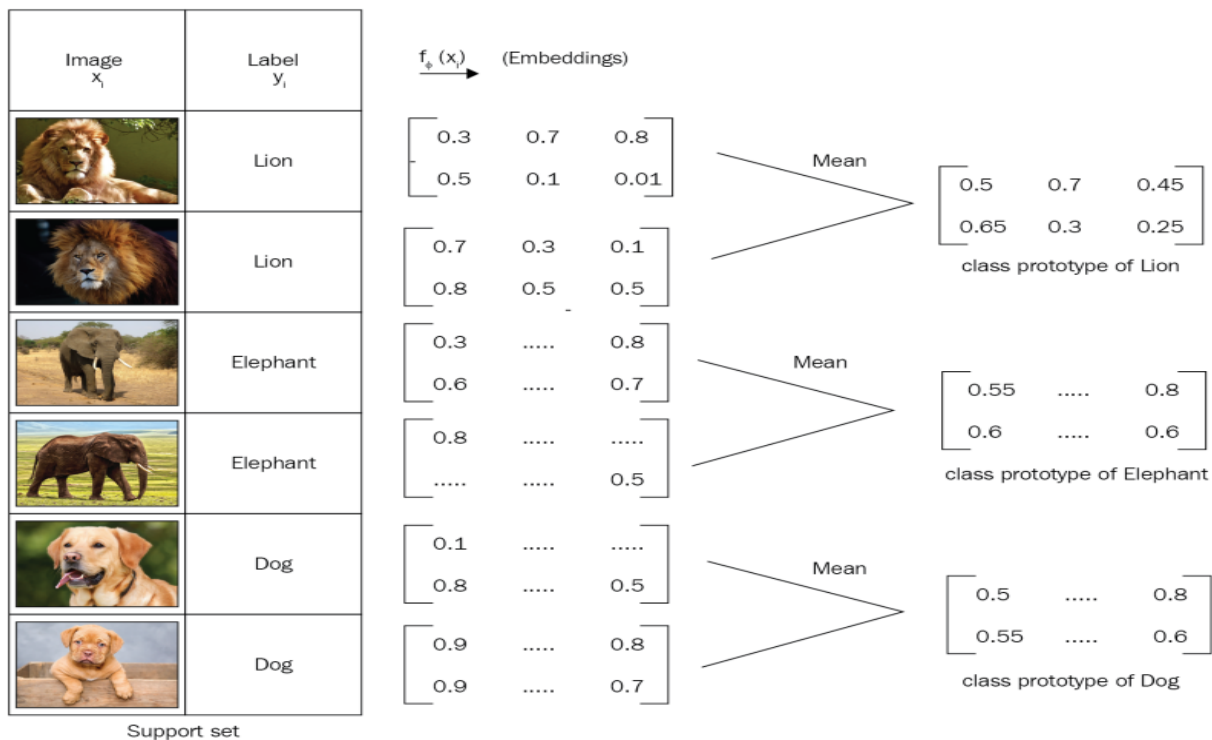
بنابراین، ما سه کلاس داریم: {شیر، فیل، سگ}. حال باید یک نمایش اولیه برای هر یک از این سه کلاس ایجاد کنیم. چگونه می توانیم نمونه اولیه این سه کلاس را بسازیم؟ ابتدا با استفاده از یک تابع جاسازی، جاسازی های هر نقطه داده را یاد می گیریم. تابع **embedding**، می تواند هر تابعی باشد که بتوان از آن برای استخراج ویژگی ها استفاده کرد. از آنجایی که ورودی ما یک تصویر است، می توانیم از شبکه کانولوشن به عنوان تابع جاسازی استفاده کنیم که ویژگی هایی را از تصویر ورودی استخراج می کند:



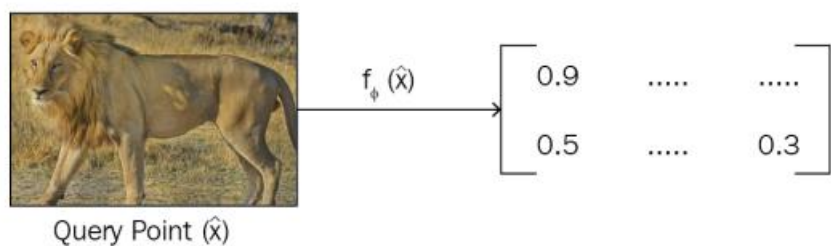
Image ( $x_i$ )	Label ( $y_i$ )	$f_{\theta}(x_i)$	(Embeddings)
	Lion		$\begin{bmatrix} 0.3 & 0.7 & 0.8 \\ 0.5 & 0.1 & 0.01 \end{bmatrix}$
	Lion		$\begin{bmatrix} 0.7 & 0.3 & 0.1 \\ 0.8 & 0.5 & 0.3 \end{bmatrix}$
	Elephant		$\begin{bmatrix} 0.3 & ..... & 0.8 \\ 0.6 & ..... & 0.7 \end{bmatrix}$
	Elephant		$\begin{bmatrix} 0.8 & ..... & ..... \\ ..... & ..... & 0.5 \end{bmatrix}$
	Dog		$\begin{bmatrix} 0.1 & ..... & ..... \\ 0.8 & ..... & 0.6 \end{bmatrix}$
	Dog		$\begin{bmatrix} 0.9 & ..... & 0.8 \\ 0.9 & ..... & 0.7 \end{bmatrix}$

Support set

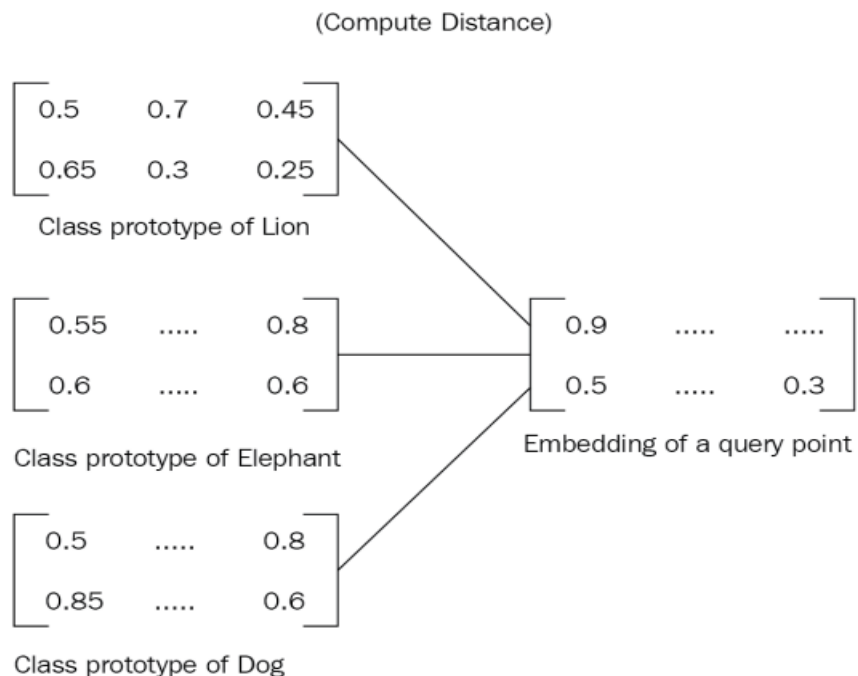
هنگامی که جاسازی های هر نقطه داده را یاد گرفتیم، میانگین جاسازی نقاط داده در هر کلاس را می گیریم و نمونه اولیه کلاس را تشکیل می دهیم، همانطور که در نمودار زیر نشان داده شده است. بنابراین، نمونه اولیه کلاس اساساً میانگین جاسازی نقاط داده در یک کلاس است:



به طور مشابه، وقتی یک نقطه داده جدید وارد می شود، یعنی یک نقطه پرس و جو که می خواهیم برچسب را برای آن پیش بینی کنیم، جاسازی های این نقطه داده جدید را با استفاده از همان تابع جاسازی که برای ایجاد نمونه اولیه کلاس استفاده کردیم، ایجاد می کنیم. این است که ما جاسازی ها را برای نقطه پرس و جو خود با استفاده از شبکه کانولوشن تولید می کنیم:



هنگامی که جاسازی نقطه پرس و جو خود را داشتیم، فاصله بین نمونه اولیه کلاس و جاسازی نقطه پرس و جو را مقایسه می کنیم تا بفهمیم نقطه پرس و جو به کدام کلاس تعلق دارد. همانطور که در اینجا نشان داده شده است، می توانیم از فاصله اقلیدسی به عنوان معیاری برای یافتن فاصله بین نمونه اولیه کلاس و جاسازی نقاط پرس و جو استفاده کنیم:

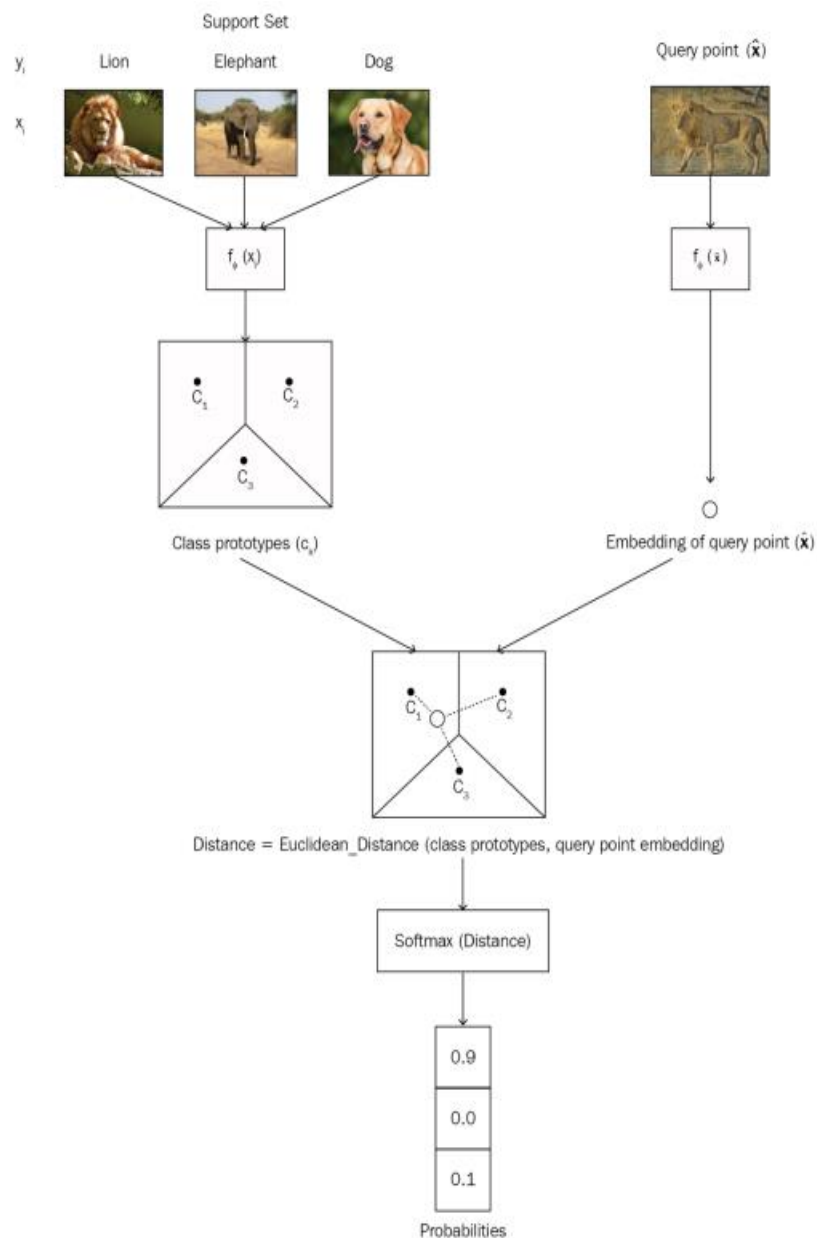


پس از یافتن فاصله بین نمونه اولیه کلاس و جاسازی نقطه پرس و جو، softmax را روی این فاصله اعمال می کنیم و احتمالات را بدست می آوریم. از آنجایی که ما سه کلاس داریم، یعنی شیر، فیل و سگ، سه احتمال به دست خواهیم آورد. بنابراین، کلاسی که بیشترین احتمال را دارد، کلاس نقطه پرس و جو ما خواهد بود.

از آنجایی که می خواهیم شبکه ما از چند نقطه داده یاد بگیرد، یعنی می خواهیم آموزش چند شات را انجام دهیم، شبکه خود را به همین ترتیب آموزش می دهیم.

بنابراین، ما از آموزش اپیزودیک استفاده می کنیم—برای هر قسمت، به طور تصادفی چند نقطه داده از هر کلاس در مجموعه داده مان نمونه برداری می کنیم و آن را یک مجموعه پشتیبانی می نامیم و شبکه را با استفاده از مجموعه پشتیبانی، به جای کل مجموعه داده، آموزش می دهیم. به طور مشابه، یک نقطه از مجموعه داده را به صورت تصادفی به عنوان نقطه پرس و جو نمونه برداری می کنیم و سعی می کنیم کلاس آن را پیش بینی کنیم. بنابراین، به این ترتیب، شبکه ما آموزش می بیند که چگونه از مجموعه کوچکتری از نقاط داده یاد بگیرد.

جریان کلی شبکه نمونه اولیه ما در نمودار زیر نشان داده شده است. همانطور که می بینید، ابتدا جاسازی ها را برای تمام نقاط داده در مجموعه پشتیبانی خود ایجاد می کنیم و با گرفتن میانگین جاسازی نقاط داده در یک کلاس، نمونه اولیه کلاس را می سازیم. ما همچنین جاسازی ها را برای نقطه پرس و جو خود ایجاد می کنیم. سپس، فاصله بین نمونه اولیه کلاس و جاسازی نقطه پرس و جو را محاسبه می کنیم. ما از فاصله اقلیدسی به عنوان اندازه گیری فاصله استفاده می کنیم. سپس softmax را روی این فاصله اعمال می کنیم و احتمالات را بدست می آوریم. همانطور که در نمودار زیر می بینید، از آنجایی که نقطه پرس و جو ما یک شیر است، احتمال شیر بالا است - ۰.۹:



شبکه های اولیه نه تنها برای یادگیری تک شات/چند شات استفاده می شوند، بلکه در یادگیری شات صفر نیز استفاده می شوند. موردی را در نظر بگیرید که هیچ نقطه داده ای در هر کلاس ندارید، اما اطلاعات متا حاوی توضیحات سطح بالایی از هر کلاس دارید. بنابراین، در آن موارد، جاسازی ها را از اطلاعات متا هر کلاس یاد می گیریم تا نمونه اولیه کلاس را تشکیل دهیم و سپس با نمونه اولیه کلاس، طبقه بندی را انجام می دهیم.

...

## شبکه نمونه اولیه گاوسی *Gaussian prototypical network*:

اکنون، ما به گونه ای از یک شبکه نمونه اولیه، به نام شبکه نمونه اولیه گاوسی نگاه خواهیم کرد. ما به تازگی یاد گرفتیم که چگونه یک شبکه نمونه اولیه جاسازی نقاط داده را یاد می گیرد و چگونه نمونه اولیه کلاس را با گرفتن میانگین جاسازی های هر کلاس می سازد و از نمونه اولیه کلاس برای انجام طبقه بندی استفاده می کند. در یک شبکه نمونه اولیه گاوسی، همراه با ایجاد جاسازی برای نقاط داده، یک منطقه اطمینان در اطراف آنها اضافه می کنیم که با ماتریس کوواریانس گاوسی مشخص می شود. داشتن یک منطقه اطمینان به توصیف کیفیت نقاط داده فردی کمک می کند و در مورد داده های پر نویز و کمتر همگن مفید خواهد بود. بنابراین، در شبکه های نمونه اولیه گاوسی، خروجی رمزگذار، جاسازی ها و همچنین ماتریس کوواریانس خواهد بود. به جای استفاده از ماتریس کوواریانس کامل، یک شعاع یا مولفه مورب از ماتریس کوواریانس به همراه جاسازی ها اضافه می کنیم:

**جزء شعاع Radius component:** اگر از مولفه شعاع ماتریس کوواریانس استفاده کنیم، بعد ماتریس کوواریانس ما ۱ خواهد بود، زیرا شعاع فقط یک عدد است.

**مولفه مورب Diagonal component:** اگر از مولفه مورب ماتریس کوواریانس استفاده کنیم، بعد ماتریس کوواریانس ما با بعد ماتریس تعبیه شده یکسان خواهد بود.

همچنین به جای استفاده مستقیم از ماتریس کوواریانس، از معکوس ماتریس کوواریانس استفاده می کنیم. ما می توانیم ماتریس کوواریانس خام را با استفاده از یکی از روش های زیر به ماتریس کوواریانس معکوس تبدیل کنیم. اجازه دهید  $S_{raw}$  ماتریس کوواریانس و  $S$  ماتریس کوواریانس معکوس باشد:

$$S = 1 + \text{Softplus}(S_{raw})$$

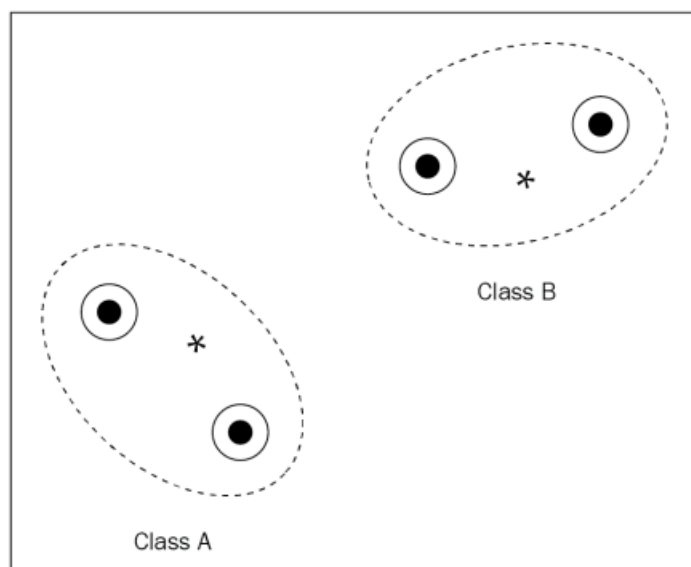
$$S = 1 + \text{sigmoid}(S_{raw})$$

$$S = 1 + 4 * \text{sigmoid}(S_{raw})$$

$$S = \text{offset} + \text{scale} * \text{softplus}(S_{raw}/\text{div}), \text{ where offset and scale are trainable parameters}$$

بنابراین، رمزگذار، همراه با ایجاد جاسازی برای ورودی، ماتریس کوواریانس را نیز برمی گرداند. ما از اجزای قطری یا شعاع ماتریس کوواریانس استفاده می کنیم. همچنین به جای استفاده مستقیم از ماتریس کوواریانس، از ماتریس کوواریانس معکوس استفاده می کنیم.

اما وجود ماتریس کوواریانس همراه با تعبیه ها چه فایده ای دارد؟ همانطور که قبلاً گفته شد، منطقه اطمینان اطراف نقاط داده را اضافه می کند و در مورد داده های نویز بسیار مفید است. به نمودار زیر نگاه کنید. فرض کنید دو کلاس A و B داریم. نقاط تاریک نشان دهنده جاسازی نقطه داده و دایره های اطراف تاریک نشان دهنده ماتریس های کوواریانس هستند. یک دایره بزرگ نقطه چین نشان دهنده ماتریس کوواریانس کلی برای یک کلاس است. یک ستاره در وسط نمونه اولیه کلاس را نشان می دهد. همانطور که می بینید، داشتن این ماتریس کوواریانس در اطراف جاسازی ها به ما یک منطقه اطمینان در اطراف نقطه داده و برای نمونه های اولیه کلاس می دهد:



بیایید با نگاه کردن به کد این موضوع را بهتر درک کنیم. فرض کنید یک تصویر،  $X$  داریم و می خواهیم جاسازی هایی برای تصویر ایجاد کنیم. بیایید ماتریس کوواریانس را با سیگما نشان دهیم. ابتدا، ما انتخاب می کنیم که از چه جزء ماتریس کوواریانس می خواهیم استفاده کنیم – یعنی اینکه آیا می خواهیم از مؤلفه مورب یا شعاع استفاده کنیم. اگر از مؤلفه شعاع استفاده کنیم، بعد ماتریس کوواریانس ما فقط یک خواهد بود. اگر مؤلفه مورب را انتخاب کنیم، اندازه ماتریس کوواریانس با بعد تعبیه شده یکسان خواهد بود:

```
if component == 'radius':
```

```
    covariance_matrix_dim = 1
```

```
else:
```

```
    covariance_matrix_dim = embedding_dim
```

اکنون رمزگذار خود را تعریف می کنیم. از آنجایی که ورودی ما یک تصویر است، از یک بلوک کانولوشن به عنوان رمزگذار خود استفاده می کنیم. بنابراین، اندازه فیلترها، تعدادی فیلتر و اندازه لایه ترکیبی را تعریف می کنیم:

```
filters = [3,3,3,3]
```

```
num_filters = [64,64,64,embedding_dim+covariance_matrix_dim]
```

```
pools = [2,2,2,2]
```

ما embedding ها را به عنوان تصویر خود،  $X$ ، مقداردهی اولیه می کنیم:

```
previous_channels = 1
```

```
embeddings = X
```

```
weight = []
```

```
bias = []
```

```
conv_relu = []
```

```
conv = []
```

```
conv_pooled = []
```

سپس عملیات کانولوشن را انجام می دهیم و جاسازی ها را می گیریم:

...



## الگوریتم

اکنون، با مرور مرحله به مرحله شبکه نمونه اولیه گاوسی را بهتر درک خواهیم کرد:

۱. فرض کنید یک مجموعه داده داریم،  $D = \{(x_1, y_1), (x_2, y_2), \dots (x_i, y_i)\}$  که  $x$  ویژگی و  $y$  برچسب است. فرض کنید یک برچسب باینری داریم، یعنی فقط دو کلاس داریم، ۰ و ۱ و ما نقاط داده را به صورت تصادفی و بدون جایگزینی از هر یک از کلاس های مجموعه داده خود،  $D$ ، نمونه برداری می کنیم و مجموعه پشتیبانی خود،  $S$  را ایجاد می کنیم.
۲. به طور مشابه، از نقاط داده به صورت تصادفی در هر کلاس نمونه برداری می کنیم و مجموعه پرس و جو،  $Q$  را ایجاد می کنیم.
۳. مجموعه پشتیبانی را به تابع embedding خود،  $f()$  منتقل می کنیم. تابع جاسازی، جاسازی های مجموعه پشتیبانی ما را به همراه ماتریس کوواریانس ایجاد می کند.
۴. معکوس ماتریس کوواریانس را محاسبه می کنیم.
۵. نمونه اولیه هر کلاس را در مجموعه پشتیبانی به صورت زیر محاسبه می کنیم:

$$Prototype(\vec{p}^c) = \frac{\sum_i \vec{s}_i^c \cdot \vec{x}_i^c}{\sum_i \vec{s}_i^c}$$

۶. پس از محاسبه نمونه اولیه هر کلاس در مجموعه پشتیبانی، جاسازی های مجموعه پرس و جو را یاد می گیریم،  $Q$ . فرض کنید  $x'$  جاسازی نقطه پرس و جو است.
۷. فاصله جاسازی های نقطه پرس و جو تا نمونه های اولیه کلاس را به صورت زیر محاسبه می کنیم:

$$distance = \sqrt{(\vec{x}' - \vec{p}_c)^T \vec{s}_i^c \cdot (\vec{x}' - \vec{p}_c)}$$

۸. پس از محاسبه فاصله بین نمونه اولیه کلاس و جاسازی مجموعه پرس و جو، کلاس مجموعه پرس و جو را به عنوان کلاسی که حداقل فاصله دارد، به صورت زیر پیش بینی می کنیم:

$$\hat{y} = \operatorname{argmin}_c(distance)$$

### شبکه های نیمه نمونه *Semi-prototypical networks*

اکنون، نوع جالب دیگری از شبکه های نمونه اولیه به نام شبکه نیمه اولیه را خواهیم دید. این به رسیدگی به نمونه های بدون برچسب می پردازد. همانطور که می دانیم، در شبکه نمونه اولیه، نمونه اولیه هر کلاس را با در نظر گرفتن میانگین تعبیه هر کلاس محاسبه می کنیم و سپس با یافتن فاصله بین نقاط پرس و جو تا نمونه های اولیه کلاس، کلاس مجموعه پرس و جو را پیش بینی می کنیم. موردی را در نظر بگیرید که مجموعه داده ما حاوی برخی از نقاط داده بدون برچسب است: چگونه نمونه های اولیه کلاس این نقاط داده بدون برچسب را محاسبه کنیم؟

...

## فصل چهارم:

### **روابط و شبکه های تطبیق با استفاده از TensorFlow:**

در فصل آخر، در مورد شبکه های نمونه اولیه و نحوه استفاده از انواع شبکه های نمونه اولیه، مانند شبکه های نمونه اولیه و نیمه نمونه اولیه گاوسی، برای یادگیری تک شات، یاد گرفتیم. ما دیده ایم که چگونه شبکه های نمونه اولیه از جاسازی ها برای انجام وظایف طبقه بندی استفاده می کنند. در این فصل با شبکه های ارتباطی و شبکه های تطبیق آشنا می شویم. ابتدا خواهیم دید که شبکه رابطه چیست و چگونه در تنظیمات یادگیری تک شات، چند شات و صفر شات استفاده می شود و پس از آن نحوه ساخت شبکه رابطه با استفاده از TensorFlow را خواهیم آموخت. بعداً در این فصل، با شبکه های تطبیق و نحوه استفاده از آنها در یادگیری چند شات آشنا خواهیم شد. همچنین انواع مختلفی از توابع جاسازی مورد استفاده در شبکه های تطبیق را خواهیم دید. در پایان این فصل، نحوه ساخت شبکه های تطبیق را در تنسورفلو خواهیم دید.

در این فصل با موارد زیر آشنا خواهیم شد:




۱. شبکه های ارتباطی
۲. شبکه های ارتباطی در تنظیمات تک شات، چند شات و صفر شات
۳. ساخت شبکه های رابطه با استفاده از TensorFlow
۴. شبکه های منطبق
۵. توابع جاسازی یک شبکه منطبق
۶. معماری شبکه های تطبیق
۷. شبکه های تطبیق در تنسورفلو

### **شبکه های ارتباطی Relation networks:**

اکنون الگوریتم یادگیری تک شات جالب دیگری به نام شبکه رابطه را خواهیم دید. این یکی از ساده ترین و کارآمدترین الگوریتم های یادگیری تک شات است. ما چگونگی استفاده از شبکه های رابطه در تنظیمات یادگیری تک شات، چند شات و صفر شات را بررسی خواهیم کرد.

### **شبکه های ارتباط در یادگیری تک شات Relation networks in one-shot learning:**

یک شبکه رابطه از دو تابع مهم تشکیل شده است: تابع **embedding** که با  $F$  علامت گذاری می شود و تابع رابطه که  $G$  با نشان داده می شود. تابع **embedding** برای استخراج ویژگی ها از ورودی استفاده می شود. اگر ورودی ما یک تصویر باشد، می توانیم از یک شبکه کانولوشنال به عنوان تابع جاسازی استفاده کنیم که بردارهای ویژگی/جاسازی های یک تصویر را به ما می دهد. اگر ورودی ما یک متن باشد، می توانیم از شبکه های **LSTM** برای دریافت جاسازی های متن استفاده کنیم. همانطور که می دانیم، در آموزش تک شات، در هر کلاس فقط یک نمونه داریم. برای مثال، فرض کنید مجموعه پشتیبانی ما شامل سه کلاس با یک مثال در هر کلاس است. همانطور که در نمودار زیر نشان داده شده است، یک مجموعه پشتیبانی شامل سه کلاس  $\{lion, elephant, dog\}$  داریم:

Image ( $x_i$ )	Label ( $y_i$ )
	Lion
	Elephant
	Dog

Support Set

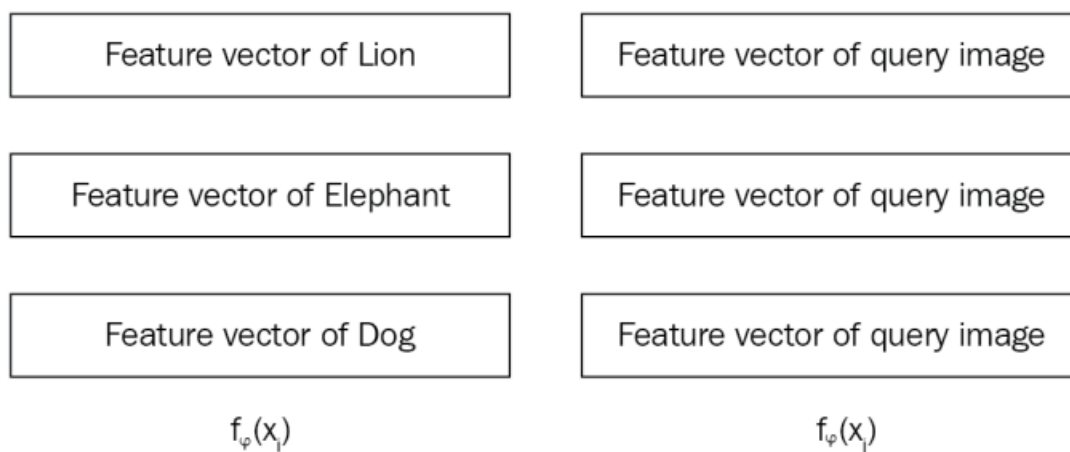
و فرض کنید همانطور که در نمودار زیر نشان داده شده است، یک تصویر **query**،  $x_q$  داریم، و می خواهیم کلاس این تصویر **query** را پیش بینی کنیم:



ابتدا هر تصویر  $X_i$  را از مجموعه پشتیبانی می گیریم و برای استخراج ویژگی ها به تابع  $F.embedding$  منتقل می کنیم. از آنجایی که مجموعه پشتیبانی ما دارای تصاویر است، می توانیم از یک شبکه کانولوشن به عنوان تابع جاسازی برای یادگیری جاسازی ها استفاده کنیم. تابع  $embedding$  بردار ویژگی هر یک از نقاط داده در مجموعه پشتیبانی را به ما می دهد. به طور مشابه، جاسازی های تصویر پرس و جو خود،  $X_j$  را با ارسال آن به تابع جاسازی،  $F$ ، یاد خواهیم گرفت.

بنابراین، هنگامی که بردارهای ویژگی مجموعه پشتیبانی،  $FX_i$  و مجموعه پرس و جو،  $FX_j$  را داریم، آنها را با استفاده از عملگر  $Z$  ترکیب می کنیم. در اینجا  $Z$  می تواند هر عملگر ترکیبی باشد. ما از الحاق به عنوان عملگر برای ترکیب بردارهای ویژگی و مجموعه پرس و جو استفاده می کنیم - یعنی  $Z(FX_i, FX_j)$ .

همانطور که در شکل زیر نشان داده شده است، بردارهای ویژگی مجموعه پشتیبانی  $FX_i$  و مجموعه پرس و جو  $FX_j$  را با هم ترکیب می کنیم. اما ترکیب اینجوری چه فایده ای دارد؟ این به ما کمک می کند تا بفهمیم که چگونه بردار ویژگی یک تصویر در مجموعه پشتیبانی با بردار ویژگی یک تصویر پرس و جو مرتبط است. در مثال ما، به ما کمک می کند تا بفهمیم که چگونه بردارهای ویژگی تصاویر یک شیر، یک فیل و یک سگ با بردار ویژگی تصویر پرس و جو مرتبط است:



$$Z ( f_{\varphi}(X_i) , f_{\varphi}(X_j) )$$

Feature concatenation

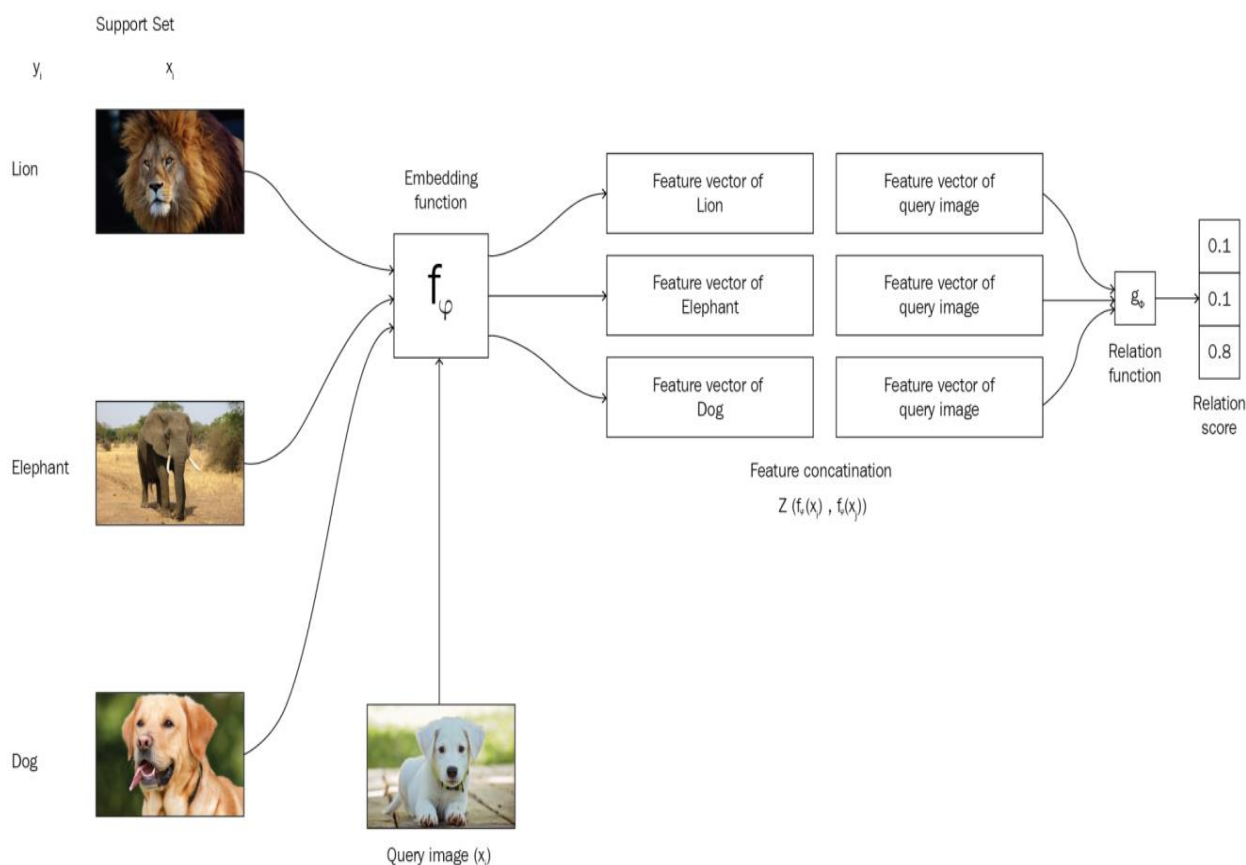
اما چگونه می توانیم این ارتباط را اندازه گیری کنیم؟ به همین دلیل است که ما از یک تابع رابطه استفاده می کنیم،  $G$ . این بردارهای ویژگی ترکیبی را به تابع رابطه می دهیم، که امتیاز رابطه بین ۰ تا ۱ را ایجاد می کند، که نشان دهنده شباهت بین نمونه ها در مجموعه پشتیبانی،  $x_i$  و نمونه ها در مجموعه پرس و جو،  $x_j$ .

معادله زیر نشان می دهد که چگونه امتیاز رابطه را در یک شبکه رابطه محاسبه می کنیم:

$$r_{ij} = g_{\phi}(Z(f_{\phi}(x_i), f_{\phi}(x_j)))$$

در این معادله،  $r_{ij}$  امتیاز رابطه را نشان می دهد که نشان دهنده شباهت بین هر یک از کلاس ها در مجموعه پشتیبانی و تصویر پرس و جو است. از آنجایی که ما سه کلاس در مجموعه پشتیبانی و یک تصویر در مجموعه پرس و جو داریم، سه امتیاز خواهیم داشت که نشان می دهد چگونه هر سه کلاس در مجموعه پشتیبانی شبیه به تصویر پرس و جو هستند.







نمایش کلی یک شبکه رابطه در یک محیط یادگیری تک شات در نمودار زیر نشان داده شده است:



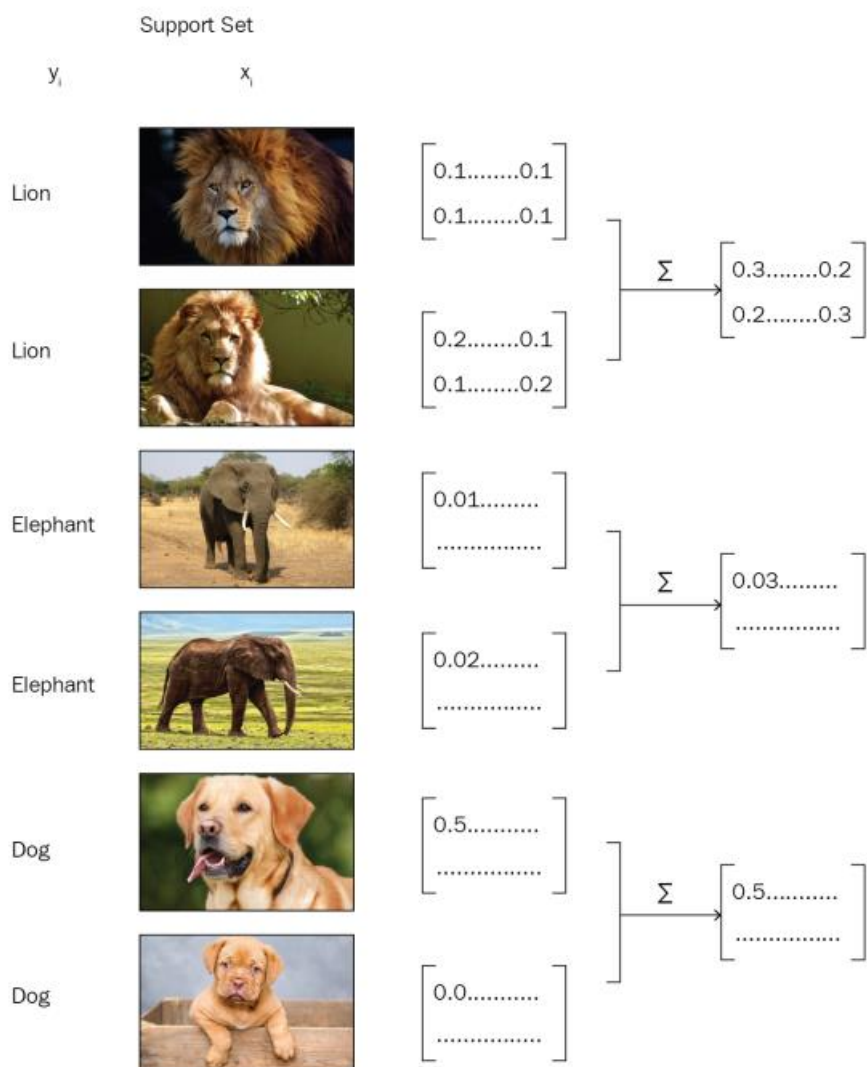
## شبکه های ارتباط در یادگیری چند شات *Relation networks in few-shot learning*:

ما دیده ایم که چگونه یک تصویر واحد متعلق به هر یک از کلاس های مجموعه پشتیبانی می گیریم و رابطه آن ها را با تصویر موجود در مجموعه پرس و جو در تنظیمات یادگیری تک شات شبکه رابطه مان مقایسه می کنیم. اما در تنظیمات یادگیری چند شات، ما بیش از یک نقطه داده در هر کلاس خواهیم داشت. چگونه با استفاده از embedding function خود، نمایش ویژگی را در اینجا یاد بگیریم؟

همانطور که در نمودار زیر نشان داده شده است، فرض کنید ما یک مجموعه پشتیبانی داریم که حاوی بیش از یک تصویر برای هر یک از کلاس ها است:

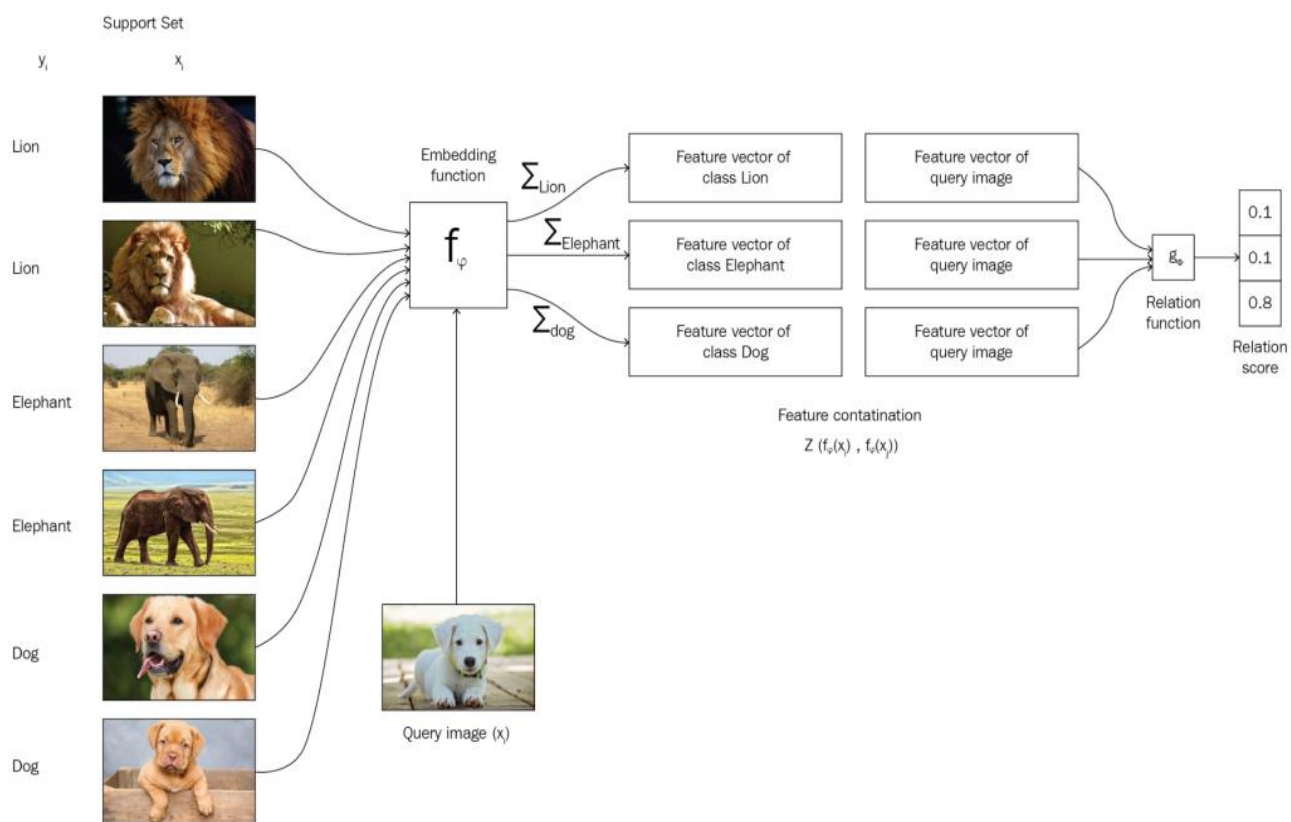
Support Set	
$y_i$	$x_j$
Lion	
Lion	
Elephant	
Elephant	
Dog	
Dog	

در این مورد، جاسازی هر نقطه در مجموعه پشتیبانی را یاد می گیریم و به صورت عنصری جاسازی همه نقاط داده متعلق به هر کلاس را انجام می دهیم. بنابراین، ما برای هر یک از کلاس ها جاسازی هایی خواهیم داشت، که از نظر عناصر، جاسازی های جمع بندی شده همه نقاط داده در آن کلاس است:



ما می توانیم بردار ویژگی تصویر پرس و جو خود را با استفاده از تابع embedding به طور معمول استخراج کنیم. سپس، بردارهای ویژگی مجموعه های پشتیبانی و پرس و جو را با استفاده از عملگر الحاق،  $Z$  ترکیب می کنیم. ما الحاق را انجام می دهیم، سپس بردارهای ویژگی الحاقی خود را به تابع رابطه می دهیم و امتیازهای رابطه را می گیریم، که نشان دهنده شباهت بین هر یک از کلاس ها در مجموعه پشتیبانی و مجموعه پرس و جو است. نمایش کلی یک شبکه رابطه در یک محیط یادگیری چند شات در شکل زیر نشان داده شده است:





### شبکه های رابطه در یادگیری شات صفر :Relation networks in zero-shot learning

اکنون که نحوه استفاده از شبکه ارتباطی را در تکالیف یادگیری تک شات و چند شات فهمیدیم، نحوه استفاده از شبکه های رابطه را در یک محیط یادگیری صفر شات که در آن هیچ نقطه داده ای در زیر هر کلاس نخواهیم داشت، خواهیم دید. با این حال، در یادگیری شات صفر، ما متا اطلاعاتی خواهیم داشت که اطلاعاتی در مورد ویژگی های هر کلاس است و در بردار معنایی، که زیرنویس C نشان دهنده کلاس است، کدگذاری می شود.

...

ساخت شبکه های رابطه با استفاده از TensorFlow

تابع رابطه بسیار ساده است، درست است؟ ما شبکه های رابطه را با پیاده سازی یکی در TensorFlow بهتر درک خواهیم کرد.

...

## شبکه های همسان

شبکه های تطبیق یکی دیگر از الگوریتم های ساده و کارآمد یادگیری تک شات است که توسط تیم DeepMind گوگل منتشر شده است. حتی می تواند برچسب هایی برای کلاس مشاهده نشده در مجموعه داده تولید کند.

فرض کنید یک مجموعه پشتیبانی داریم،  $S$ ، که شامل  $K$  مثال هایی به عنوان  $(x_1, y_1)$ ،  $(x_2, y_2)$ ،  $(x_3, y_3)$ ، ...،  $(x_K, y_K)$  است. هنگامی که یک نقطه پرس و جو داده می شود (یک مثال نادیده جدید)، شبکه تطبیقی کلاس  $y$  را با مقایسه آن با مجموعه پشتیبانی پیش بینی می کند.

ما می توانیم این را به صورت  $p(y|x, S)$  تعریف کنیم، جایی که  $p$  شبکه عصبی پارامتری شده است،  $y$  کلاس پیش بینی شده برای نقطه پرس و جو،  $x$  و مجموعه پشتیبانی است  $p(y|x, S)$ . احتمال تعلق  $x$  به هر یک از کلاس های مجموعه داده را برمی گرداند. سپس کلاس  $x$  را به عنوان کلاسی که بیشترین احتمال را دارد انتخاب می کنیم. اما این دقیقاً چگونه کار می کند؟ این احتمال چگونه محاسبه می شود؟ حالا بیایید آن را ببینیم.

خروجی،  $y$ ، برای نقطه پرس و جو،  $x$ ، به صورت زیر قابل پیش بینی است:

$$\hat{y} = \sum_{i=1}^k a(\hat{x}, x_i) y_i$$

بیایید این معادله را رمزگشایی کنیم.  $x_i$  و  $y_i$  ورودی و برچسب های مجموعه پشتیبانی هستند.  $x$  ورودی پرس و جو است - ورودی که می خواهیم برچسب را به آن پیش بینی کنیم.  $a$  مکانیسم توجه بین  $x$  و  $x_i$  است. اما چگونه توجه را انجام دهیم؟ در اینجا ما از یک مکانیسم توجه ساده استفاده می کنیم، که تابع softmax در فاصله کسینوس بین  $x$  و  $x_i$  است - یعنی:

$$a(\hat{x}, x_i) = \text{softmax}(\text{cosine}(\hat{x}, x_i))$$

ما نمی توانیم فاصله کسینوس بین ورودی خام،  $x$  و  $x_i$  را مستقیماً محاسبه کنیم. بنابراین، ابتدا تعبیه های آنها را یاد می گیریم و فاصله کسینوس بین جاسازی ها را محاسبه می کنیم. ما از دو جاسازی مختلف،  $f$  و  $g$ ، برای

یادگیری تعبیه‌های ورودی query، x و ورودی مجموعه پشتیبانی، xi، استفاده می‌کنیم. ما خواهیم دید که این دو تابع جاسازی، f و g، دقیقاً چگونه جاسازی‌ها را در بخش آینده یاد می‌گیرند.

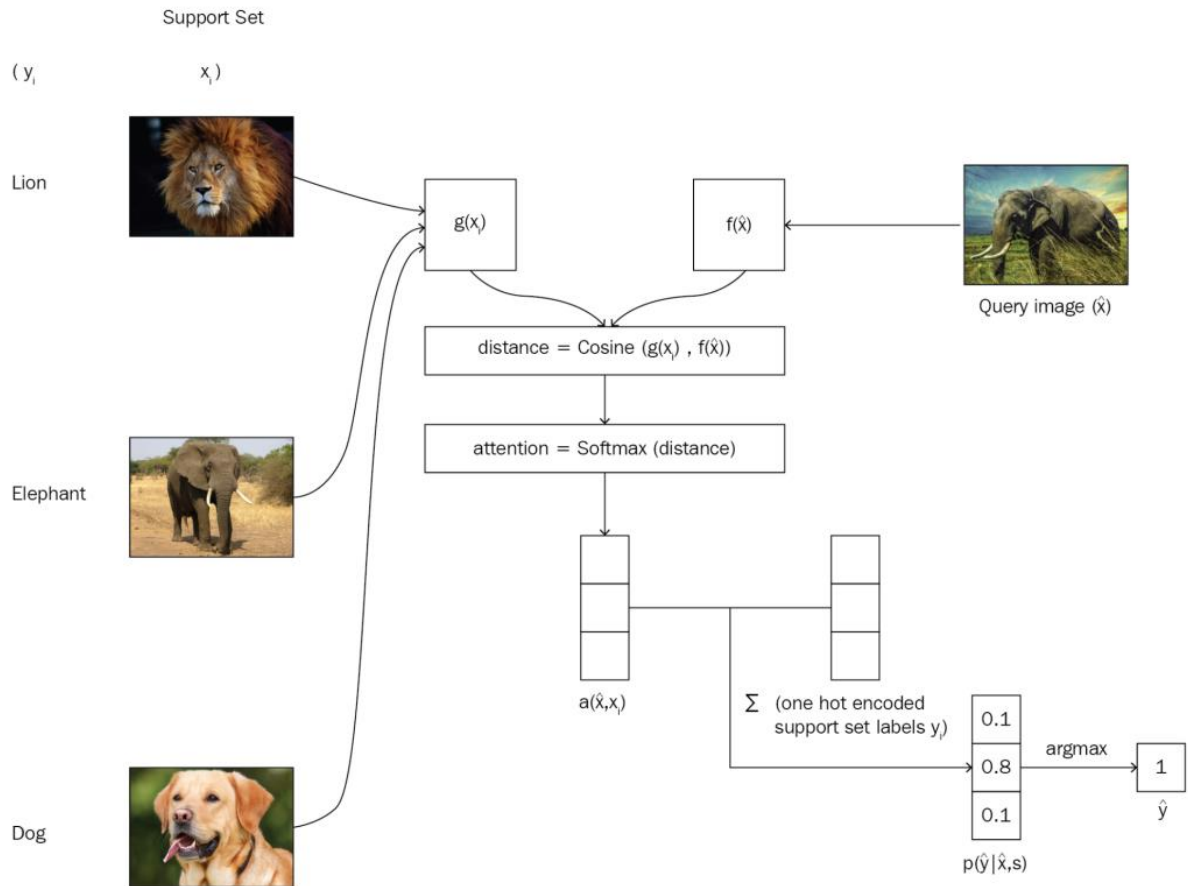
بنابراین، ما می‌توانیم معادله توجه خود را به شرح زیر بازنویسی کنیم:

$$a(\hat{x}, x_i) = \text{softmax}(\text{cosine}(f(\hat{x}), g(x_i)))$$

ما می‌توانیم معادله قبلی را به شرح زیر بازنویسی کنیم:

$$a(\hat{x}, x_i) = \frac{e^{\text{cosine}(f(\hat{x}), g(x_i))}}{\sum_{j=1}^k e^{\text{cosine}(f(\hat{x}), g(x_j))}}$$

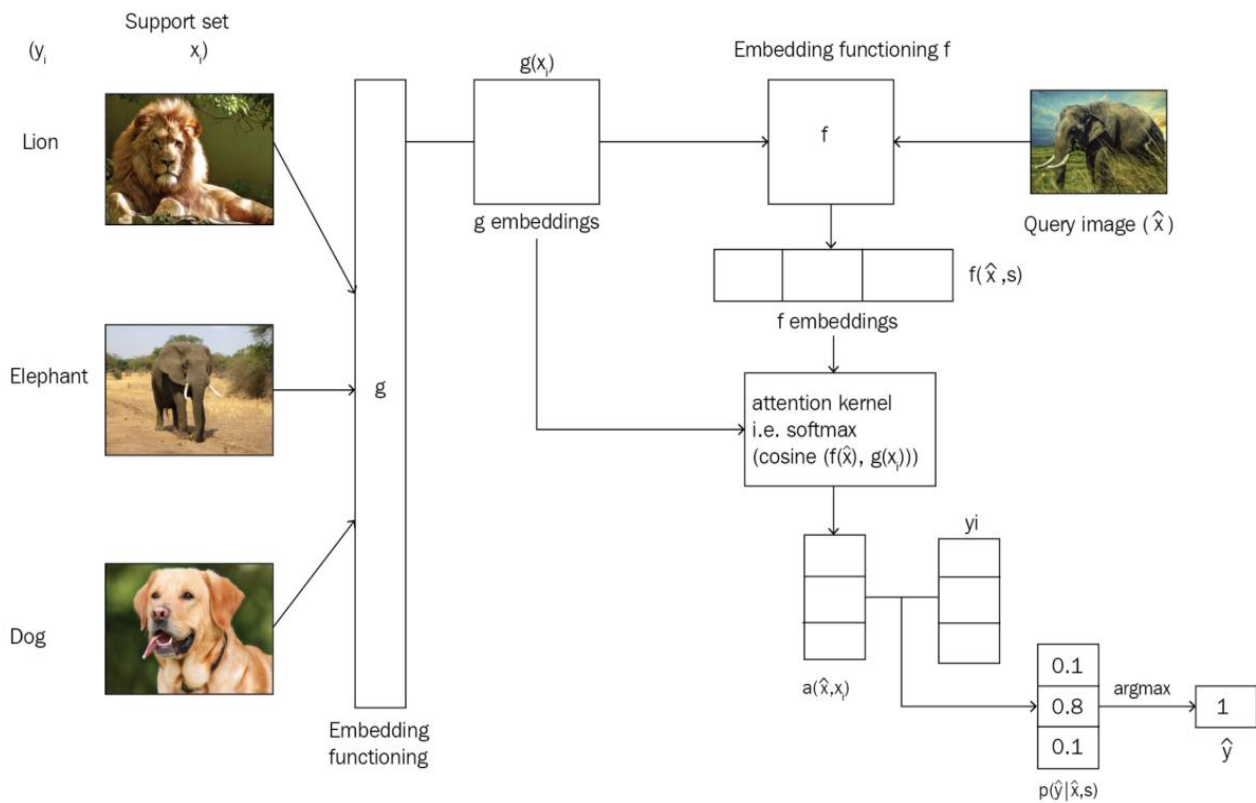
بنابراین، پس از محاسبه ماتریس توجه،  $a(x, xi)$ ، ماتریس توجه خود را با برچسب‌های مجموعه پشتیبانی، yi ضرب می‌کنیم. اما چگونه می‌توانیم برچسب‌های مجموعه پشتیبانی را با ماتریس توجه خود ضرب کنیم؟ ابتدا برچسب‌های مجموعه پشتیبانی خود را به مقادیر رمزگذاری شده تکی تبدیل می‌کنیم و سپس آنها را با ماتریس توجه خود ضرب می‌کنیم و در نتیجه، احتمال تعلق y به هر یک از کلاس‌های مجموعه پشتیبانی را به دست می‌آوریم. سپس، argmax را اعمال می‌کنیم و y را که دارای حداکثر مقدار احتمال است انتخاب می‌کنیم. آیا هنوز در مورد تطبیق شبکه‌ها واضح نیستید؟ به نمودار زیر توجه کنید؛ همانطور که می‌بینید، ما سه کلاس در مجموعه پشتیبانی خود داریم، {شیر، فیل و سگ}، و یک تصویر query جدید، x داریم. ابتدا، مجموعه پشتیبانی را به تابع embedding، g، و تصویر query را به تابع embedding، f می‌دهیم، و تعبیه‌های آنها را یاد می‌گیریم و فاصله کسینوس بین آنها را محاسبه می‌کنیم. سپس، توجه softmax را در این فاصله کسینوس اعمال می‌کنیم. سپس، ماتریس توجه خود را با برچسب‌های مجموعه پشتیبانی کدگذاری شده یک گرم ضرب می‌کنیم و احتمالات را بدست می‌آوریم و سپس y را به عنوان بیشترین احتمال انتخاب می‌کنیم. همانطور که در نمودار زیر مشاهده می‌کنید، تصویر مجموعه پرس و جو یک فیل است و ما در شاخص ۱ احتمال بالایی داریم، بنابراین کلاس y را به صورت ۱ (فیل) پیش‌بینی می‌کنیم:



...

### معماری شبکه های تطبیق *The architecture of matching networks*:

جریان کلی شبکه تطبیق در نمودار زیر نشان داده شده است و با تصویری که قبلاً دیدیم متفاوت است. می توانید متوجه شوید که چگونه مجموعه پشتیبانی،  $X_i$  و مجموعه پرس و جو،  $X$ ، به ترتیب از طریق توابع جاسازی،  $g$  و  $f$  محاسبه می شوند. همانطور که می بینید، تابع  $f$ ، مجموعه پرس و جو را به همراه جاسازی های مجموعه پشتیبانی به عنوان ورودی می گیرد:



### شبکه های تطبیق در تنسورفلو *Matching networks in TensorFlow*:

اکنون نحوه ساخت یک شبکه تطبیق در TensorFlow را مرحله به مرحله خواهیم دید. کد نهایی را در انتها خواهیم دید.

...

**شبکه های عصبی تقویت شده با حافظه Memory-Augmented Neural Networks:**

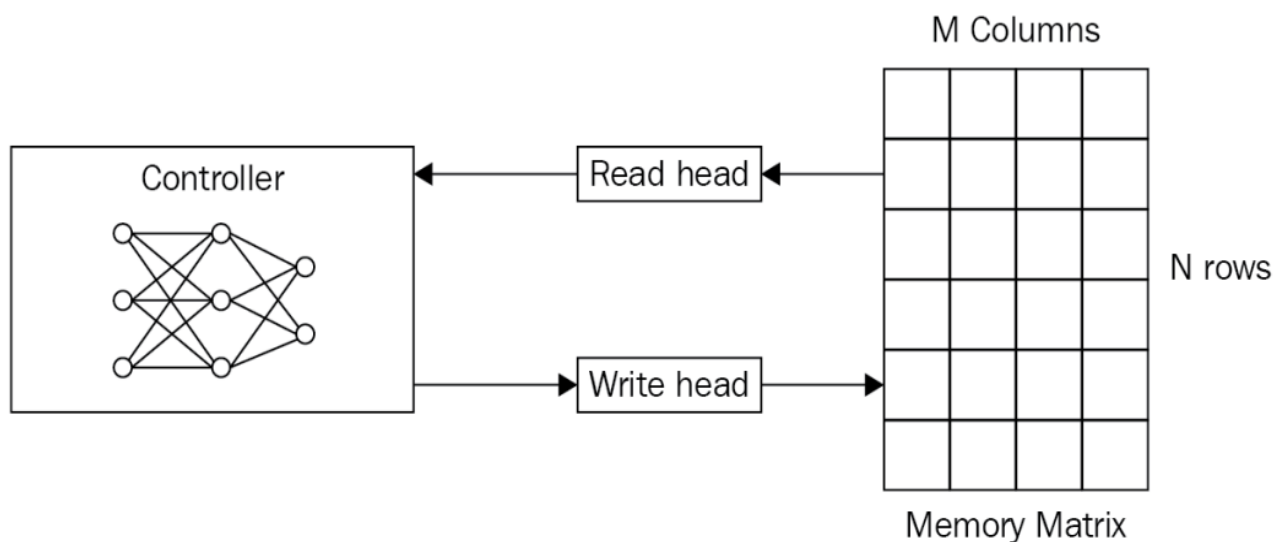
تاکنون در فصل های قبل چندین الگوریتم یادگیری متریک مبتنی بر فاصله را یاد گرفتیم. ما با شبکه های سیامی شروع کردیم و دیدیم که چگونه شبکه های سیامی یاد می گیرند بین دو ورودی تمایز قائل شوند، سپس به شبکه های نمونه اولیه و انواع شبکه های نمونه اولیه، مانند شبکه های نمونه اولیه گوسی و شبکه های نیمه نمونه اولیه نگاه کردیم. در ادامه، شبکه های تطبیق و شبکه های ارتباطی جالبی را بررسی کردیم. در این فصل، با شبکه های عصبی تقویت شده حافظه (MANN) آشنا می شویم که برای یادگیری تک شات استفاده می شود. قبل از غواصی در MANN، ما در مورد سلف آنها، ماشین های تورینگ عصبی (NTM) خواهیم آموخت. ما یاد خواهیم گرفت که چگونه NTM ها از حافظه خارجی برای ذخیره و بازیابی اطلاعات استفاده می کنند و همچنین خواهیم دید که چگونه از NTM برای انجام وظایف کپی استفاده کنیم.

در این فصل با موارد زیر آشنا خواهیم شد:

۱. NTM
۲. خواندن و نوشتن در NTM
۳. مکانیسم های آدرس دهی
۴. وظایف را با استفاده از NTM کپی کنید
۵. MANN
۶. خواندن و نوشتن در MANN

**:NTM**

NTM الگوریتم جالبی است که قابلیت ذخیره و بازیابی اطلاعات از حافظه را دارد. ایده NTM این است که شبکه عصبی را با یک حافظه خارجی تقویت کند، یعنی به جای استفاده از حالت های پنهان به عنوان حافظه، از یک حافظه خارجی برای ذخیره و بازیابی اطلاعات استفاده می کند. معماری NTM در شکل زیر نشان داده شده است:



اجزای مهم NTM به شرح زیر است:

کنترل کننده: این اساساً یک شبکه عصبی پیشخور یا شبکه عصبی بازگشتی است. از حافظه می‌خواند و می‌نویسد.

حافظه: ماتریس حافظه یا بانک حافظه یا به سادگی حافظه جایی است که اطلاعات را ذخیره می‌کنیم. حافظه اساساً یک ماتریس دو بعدی است که از سلول‌های حافظه تشکیل شده است. ماتریس حافظه شامل  $N$  ردیف و  $M$  ستون است. با استفاده از کنترلر به محتوا از حافظه دسترسی پیدا می‌کنیم. بنابراین، کنترلر ورودی را از محیط خارجی دریافت می‌کند و با تعامل با ماتریس حافظه، پاسخ‌هایی را منتشر می‌کند.

سر خواندن و نوشتن: سر خواندن و نوشتن، نشانگرهایی هستند که حاوی آدرس‌هایی از حافظه هستند که باید از آن بخواند و بنویسد.

خوب، اما چگونه می‌توانیم به اطلاعات از حافظه دسترسی پیدا کنیم؟ آیا می‌توانیم با تعیین شاخص سطر و ستون به اطلاعات حافظه دسترسی پیدا کنیم؟ بله ما می‌توانیم. اما مشکل این است که، اگر به اطلاعات بر اساس شاخص دسترسی داشته باشیم، نمی‌توانیم NTM خود را با استفاده از گرادیان نزول آموزش دهیم، زیرا نمی‌توانیم گرادیان را برای یک شاخص محاسبه کنیم. بنابراین، نویسندگان NTM عملیات تری را برای خواندن و نوشتن با استفاده از یک کنترلر تعریف می‌کند. عملیات تری با تمام عناصر موجود در حافظه تا حدی تعامل خواهد داشت. اساساً یک مکانیسم توجه است که به شدت بر روی یک مکان خاص در حافظه متمرکز می‌شود.

که خواندن/نوشتن مهم است، در حالی که تمرکز روی مکان دیگر را نادیده می گیرد. بنابراین، ما از عملیات خواندن و نوشتن ویژه استفاده می کنیم تا تعیین کنیم که روی کدام مکان از حافظه تمرکز کنیم. در بخش آتی در مورد عملیات خواندن و نوشتن اطلاعات بیشتری خواهیم داشت.

### خواندن و نوشتن در NTM

اکنون نحوه خواندن و نوشتن در ماتریس حافظه را خواهیم دید.

### عملیات خواندن

عملیات خواندن یک مقدار را از حافظه می خواند. اما از آنجایی که ما بلوک های حافظه زیادی در ماتریس حافظه خود داریم، کدام یک را برای خواندن از حافظه باید انتخاب کنیم؟ که توسط بردار وزن تعیین می شود. بردار وزن مشخص می کند که کدام ناحیه در حافظه مهمتر از سایرین است. ما از مکانیزم توجه برای بدست آوردن این بردار وزن استفاده می کنیم. ما در بخش بعدی بیشتر در مورد نحوه محاسبه دقیق این بردار وزن بررسی خواهیم کرد. بردار وزن نرمال شده است، به این معنی که مقدار آن از صفر تا یک متغیر است و مجموع مقدار برابر با یک است. نمودار زیر بردار وزن طول  $N$  را نشان می دهد:

0.3	0.1	0.3	0.1	0.2
-----	-----	-----	-----	-----

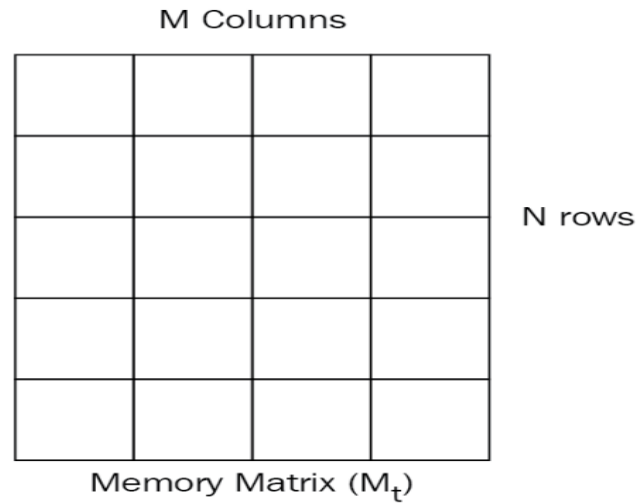
Weight Vector ( $w_t$ )

بیایید این بردار وزن نرمال شده را با  $w_t$  نشان دهیم، جایی که زیرنویس  $t$  دلالت بر زمان دارد و  $w_{t(i)}$  عنصری در بردار وزن را در شاخص  $i$  و زمان  $t$  را نشان می دهد:

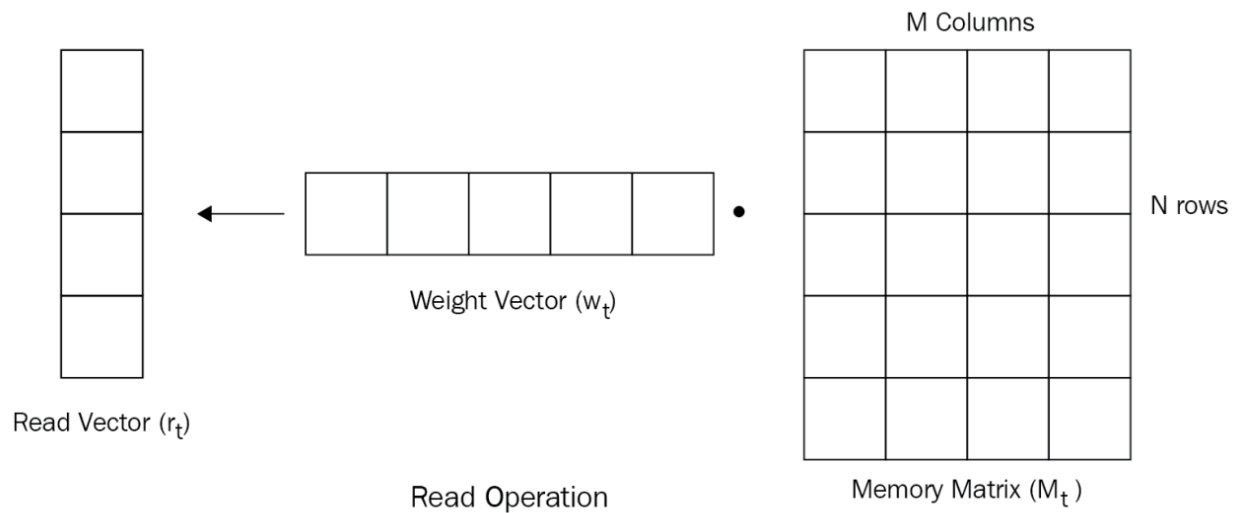
$$\sum_i w_t(i) = 1, \quad 0 \leq w_t(i) \leq 1, \quad \forall i$$



ماتریس حافظه ما از  $N$  ردیف و  $M$  ستون تشکیل شده است، همانطور که در نمودار زیر نشان داده شده است. بیایید ماتریس حافظه خود را در زمان  $t$  به صورت  $M_t$  نشان دهیم:



اکنون که بردار وزن و ماتریس حافظه را داریم، ترکیب خطی ماتریس حافظه  $M_t$  و بردار وزن  $w_t$  را انجام می دهیم تا بردار خوانده شده  $r_t$  را به دست آوریم، همانطور که در شکل زیر نشان داده شده است:



...

## **MAML و انواع آن: MAML and Its Variants:**

در فصل قبل با ماشین تورینگ عصبی (NTM) و نحوه ذخیره و بازیابی اطلاعات از حافظه آشنا شدیم. ما همچنین در مورد نوع NTM به نام شبکه عصبی تقویت شده حافظه، که به طور گسترده در یادگیری تک شات استفاده می شود، آشنا شدیم. در این فصل یکی از جالب ترین و پرکاربردترین الگوریتم های یادگیری متا به نام Model Agnostic Meta Learning (MAML) را خواهیم آموخت. ما خواهیم دید که یادگیری متا آگنوستیک چیست و چگونه از آن در تنظیمات یادگیری تحت نظارت و تقویتی استفاده می شود. همچنین یاد خواهیم گرفت که چگونه از ابتدا MAML بسازیم و سپس با یادگیری متا (ADML) آشنا خواهیم شد. خواهیم دید که چگونه از ADML برای یافتن یک پارامتر مدل قوی استفاده می شود. پس از آن، نحوه پیاده سازی ADML را برای کار طبقه بندی خواهیم آموخت. در نهایت، ما در مورد سازگاری زمینه برای یادگیری متا (CAML) خواهیم آموخت.

در این فصل با موارد زیر آشنا خواهید شد:

۱. MAML
۲. الگوریتم MAML
۳. MAML در تنظیمات یادگیری تحت نظارت و تقویتی
۴. ساخت MAML از ابتدا
۵. ADML
۶. ساخت ADML از ابتدا
۷. CAML

## **:MAML**

MAML یکی از الگوریتم های متا یادگیری اخیراً معرفی شده و پرکاربرد است و پیشرفت بزرگی در تحقیقات فرا یادگیری ایجاد کرده است. یادگیری برای یادگیری تمرکز اصلی فرا یادگیری است و ما می دانیم که در فرا یادگیری، ما از کارهای مرتبط مختلف که فقط تعداد کمی از نقاط داده را شامل می شود، یاد می گیریم و

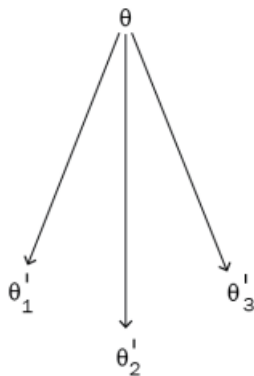
فراگیرنده یک یادگیرنده سریع تولید می کند که می تواند به خوبی روی یک کار مرتبط جدید حتی با تعداد کمتری از نمونه های آموزشی تعمیم دهد.

ایده اصلی MAML یافتن یک پارامتر اولیه بهتر است تا با پارامترهای اولیه خوب، مدل بتواند به سرعت کارهای جدید را با مراحل گرادیان کمتر یاد بگیرد. بنابراین، منظور ما از آن چیست؟ فرض کنید در حال انجام یک کار طبقه بندی با استفاده از یک شبکه عصبی هستیم. چگونه شبکه را آموزش دهیم؟ ما با مقداردهی اولیه وزن های تصادفی شروع می کنیم و با به حداقل رساندن تلفات، شبکه را آموزش می دهیم. چگونه ضرر را به حداقل برسانیم؟ ما این کار را با استفاده از نزول گرادیان انجام می دهیم. خوب، اما چگونه از شیب نزول برای به حداقل رساندن تلفات استفاده کنیم؟ ما از شیب نزول برای یافتن وزن های بهینه استفاده می کنیم که کمترین ضرر را به ما می دهد. ما چندین مرحله گرادیان را برای یافتن وزن های بهینه انجام می دهیم تا بتوانیم به همگرایی برسیم.

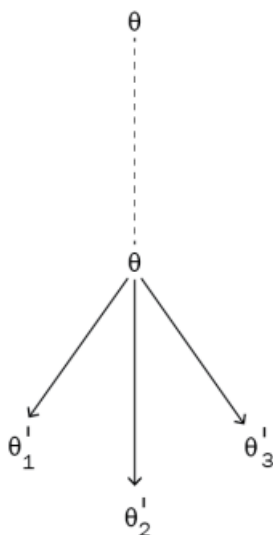
در MAML سعی می کنیم با یادگیری از توزیع وظایف مشابه، این وزن های بهینه را پیدا کنیم. بنابراین، برای یک کار جدید، لازم نیست که با وزن های اولیه تصادفی شروع کنیم - در عوض، می توانیم با وزن های بهینه شروع کنیم، که گام های گرادیان کمتری برای رسیدن به همگرایی می برد و به نقاط داده بیشتری برای آموزش نیاز ندارد.

بیایید MAML را به زبان ساده درک کنیم. فرض کنید سه کار مرتبط داریم  $T_1$ ،  $T_2$  و  $T_3$ . ابتدا، ما به طور تصادفی پارامتر مدل خود،  $\theta$  را مقداردهی اولیه می کنیم. ما شبکه خود را در وظیفه  $T_1$  آموزش می دهیم. سپس، سعی می کنیم اتلاف  $L$  را با نزول گرادیان به حداقل برسانیم. با یافتن پارامتر بهینه،  $\theta_1$ ، تلفات را به حداقل می رسانیم. به طور مشابه، برای وظایف  $T_2$  و  $T_3$ ، ما با یک پارامتر مدل به طور تصادفی اولیه،  $\theta$ ، شروع می کنیم و با یافتن مجموعه ای از پارامترها با نزول گرادیان، تلفات را به حداقل می رسانیم. فرض کنید  $\theta_2$  و  $\theta_3$  به ترتیب پارامترهای بهینه برای وظایف،  $T_2$  و  $T_3$  هستند.

همانطور که در نمودار زیر مشاهده می کنید، ما هر کار را با پارامتر  $\theta$  به طور تصادفی اولیه آغاز می کنیم و با یافتن پارامترهای بهینه  $\theta_1$ ،  $\theta_2$  و  $\theta_3$  به ترتیب برای هر یک از وظایف  $T_1$ ،  $T_2$  و  $T_3$ ، تلفات را به حداقل می رسانیم:



با این حال، به جای مقداردهی اولیه  $\theta$  در یک موقعیت تصادفی - یعنی با مقادیر تصادفی - اگر  $\theta$  را در موقعیتی که برای هر سه کار مشترک است مقداردهی اولیه کنیم، نیازی به برداشتن گام‌های گرادیان بیشتر نیست و زمان کمتری برای آموزش از ما می‌گیرد. MAML سعی می‌کند دقیقاً این کار را انجام دهد. MAML سعی می‌کند این پارامتر بهینه  $\theta$  را که در بسیاری از وظایف مرتبط مشترک است، پیدا کند، بنابراین ما می‌توانیم یک کار جدید را نسبتاً سریع با نقاط داده کم بدون نیاز به برداشتن مراحل گرادیان زیاد آموزش دهیم. همانطور که در نمودار زیر نشان داده شده است،  $\theta$  را به موقعیتی تغییر می‌دهیم که برای همه مقادیر بهینه  $\theta'$  مشترک است:



بنابراین، برای یک کار مرتبط جدید، مثلاً، T4، لازم نیست که با یک پارامتر به طور تصادفی اولیه،  $\theta$  شروع کنیم. در عوض، می‌توانیم با مقدار بهینه  $\theta$  شروع کنیم تا گام‌های گرادیان کمتری برای رسیدن به همگرایی انجام شود. بنابراین، در MAML، ما سعی می‌کنیم این مقدار  $\theta$  بهینه را که برای کارهای مرتبط مشترک است،

پیدا کنیم، به طوری که به ما در یادگیری از نقاط داده کمتر و به حداقل رساندن زمان آموزش کمک می کند .  
MAML مدل آگنوستیک است، به این معنی که ما می توانیم MAML را برای هر مدلی که با گرادیان نزول  
قابل آموزش هستند اعمال کنیم. اما دقیقاً MAML چگونه کار می کند؟ چگونه پارامترهای مدل را به موقعیت  
بهینه تغییر دهیم؟ در بخش بعدی به تفصیل آن را بررسی خواهیم کرد.

## الگوریتم MAML

اکنون که درک اولیه ای از MAML داریم، آن را با جزئیات بررسی خواهیم کرد. فرض کنید ما یک مدل  $f$   
داریم که با  $\theta$  یعنی  $f(\theta)$  پارامتر شده است و یک توزیع روی وظایف،  $p(T)$  داریم. ابتدا، پارامتر  $\theta$  خود را با  
مقادیری تصادفی مقداردهی اولیه می کنیم. در مرحله بعد، دسته ای از وظایف  $T_i$  را از توزیع بر روی وظایف  
نمونه برداری می کنیم - یعنی  $T_i \sim p(T)$ . فرض کنید از پنج وظیفه نمونه برداری کرده ایم،

$T = \{T_1, T_2, T_3, T_4, T_5\}$ ، سپس برای هر وظیفه  $T_i$ ،  $k$  نقطه داده را نمونه برداری کرده و مدل را آموزش  
می دهیم. ما این کار را با محاسبه تلفات انجام می دهیم و با استفاده از گرادیان نزول تلفات را به حداقل می  
رسانیم و مجموعه بهینه پارامترهایی را پیدا می کنیم که تلفات را به حداقل می رساند:

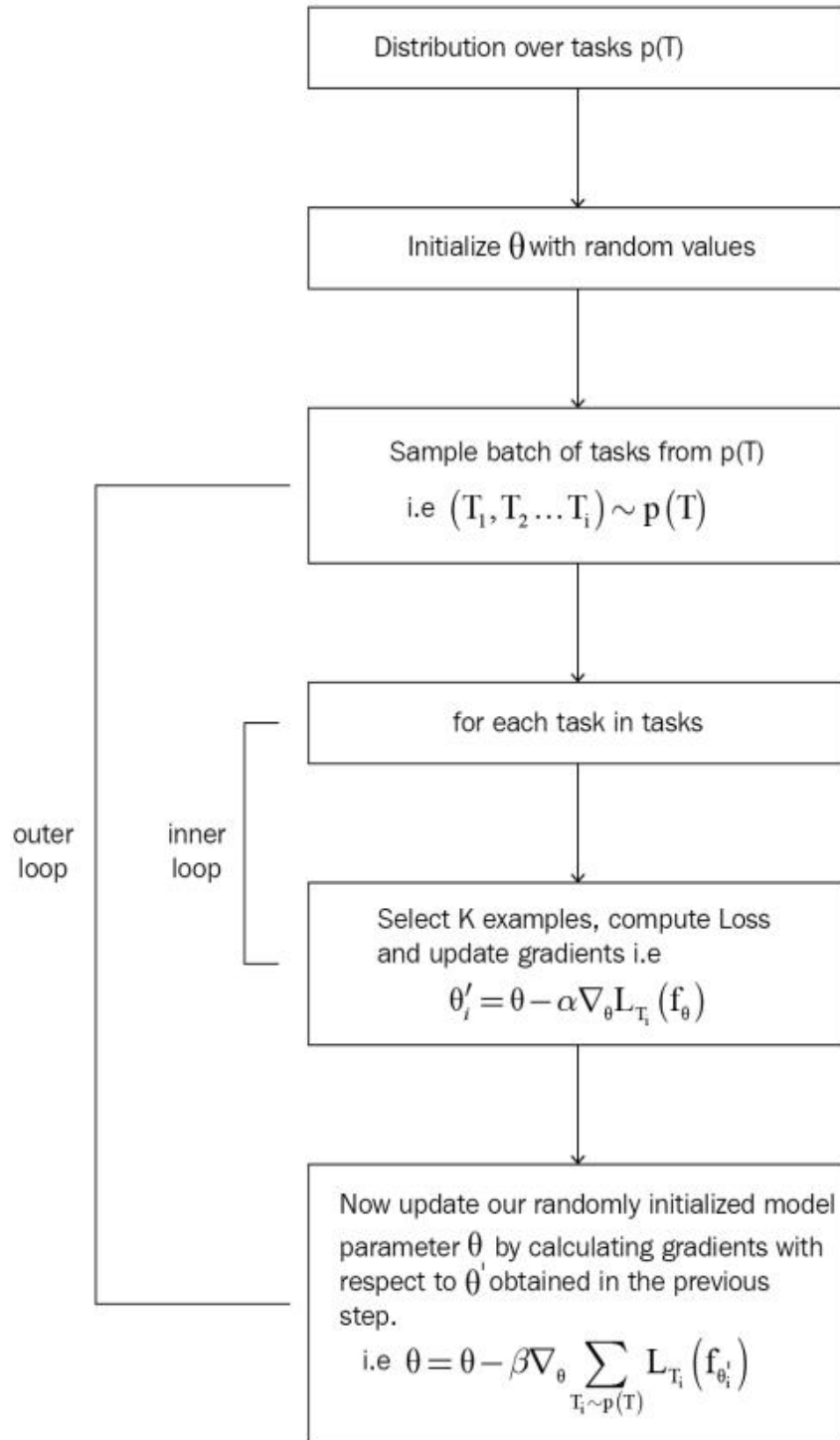
$$\theta'_i = \theta - \alpha \nabla_{\theta} L_{T_i}(f_{\theta})$$

In the previous equation, the following applies:

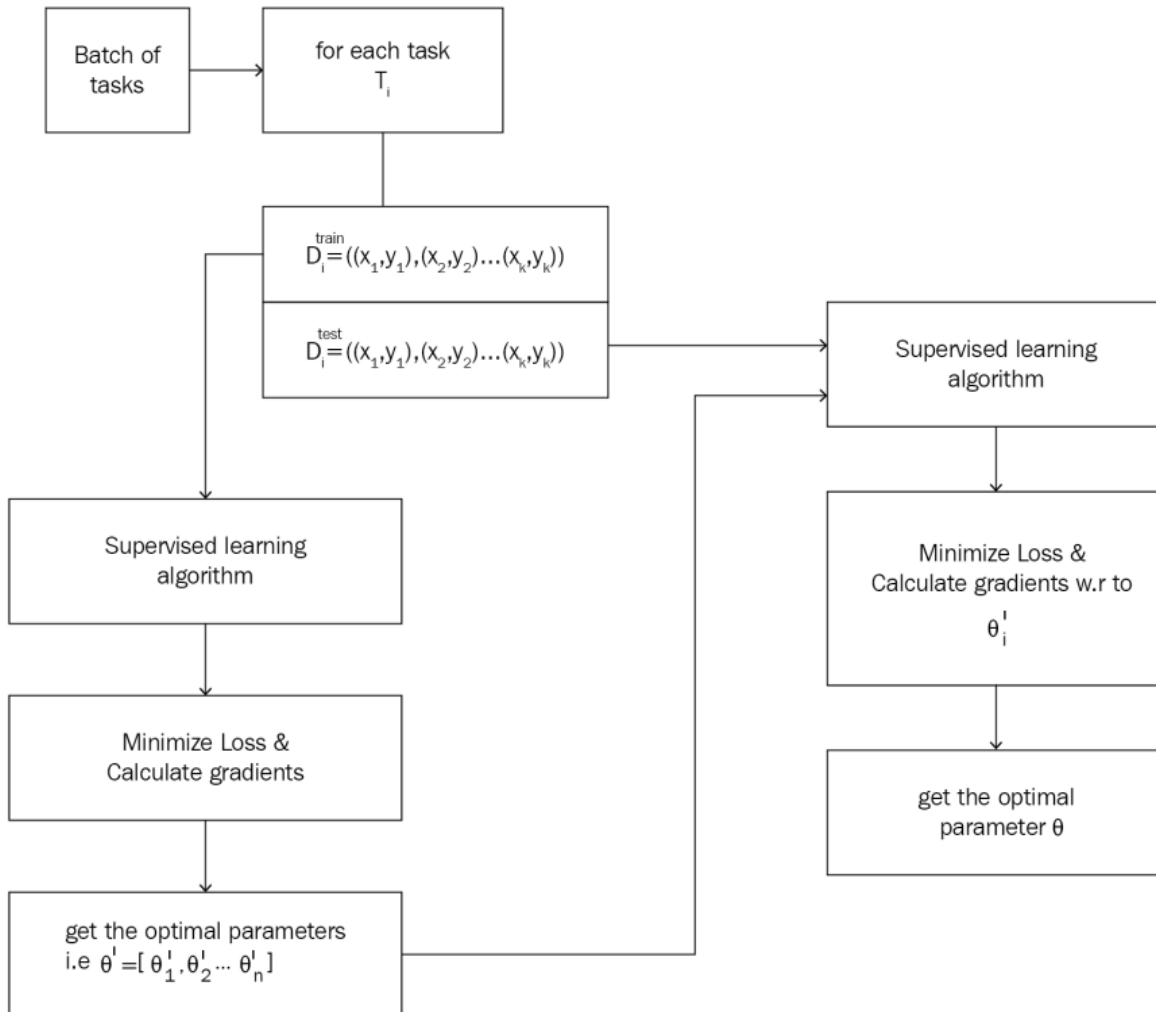
- $\theta'_i$  is the optimal parameter for a task  $T_i$
- $\theta$  is the initial parameter
- $\alpha$  is the hyperparameter
- $\nabla_{\theta} L_{T_i}(f_{\theta})$  is the gradient of a task  $T_i$

...

الگوریتم کلی MAML در نمودار زیر نشان داده شده است. الگوریتم ما از دو حلقه تشکیل شده است - یک  
حلقه داخلی که در آن پارامتر بهینه  $\theta_i$  را برای هر یک از وظایف  $T_i$  پیدا می کنیم و یک حلقه بیرونی که در آن  
پارامتر مدل اولیه  $\theta$  به طور تصادفی را با محاسبه گرادیان ها با توجه به پارامترهای بهینه  $\theta_i$  در یک مجموعه  
جدید به روز می کنیم. از وظایف  $T_i$ :



## ***:MAML in supervised learning***



## ***:Adversarial meta learning مخالف یادگیری متا***

ما دیدیم که چگونه از MAML برای یافتن پارامتر بهینه  $\theta$  که قابل تعمیم در بین وظایف است استفاده می شود. اکنون، یک نوع از MAML به نام ADML را خواهیم دید که از نمونه های تمیز و متخاصم برای یافتن پارامتر مدل اولیه بهتر و قوی  $\theta$  استفاده می کند. قبل از ادامه، بیایید بفهمیم که نمونه های متخاصم چیست. نمونه های خصمانه در نتیجه حملات خصمانه به دست می آیند. فرض کنید یک تصویر داریم. حمله خصمانه شامل تغییر جزئی این تصویر به گونه ای است که برای چشم ما قابل تشخیص نباشد و به این تصویر اصلاح

شده، تصویر خصمانه می گویند. وقتی این تصویر متخاصم را به مدل تغذیه می کنیم، آن را به درستی طبقه بندی نمی کند. چندین حمله خصمانه مختلف برای به دست آوردن نمونه های دشمن استفاده می شود. یکی از روش های رایج به نام روش نشانه گرادیان سریع (FGSM) را خواهیم دید.

## :FGSM

فرض کنید در حال انجام یک طبقه بندی تصویر هستیم. به طور کلی، ما مدل را با محاسبه تلفات و تلاش برای به حداقل رساندن تلفات با محاسبه گرادیان های تلفات با توجه به پارامترهای مدلمان، مانند وزن ها، و به روزرسانی پارامتر مدلمان آموزش می دهیم. برای به دست آوردن نمونه رقیب تصویر خود، به جای پارامتر مدل، گرادیان های از دست دادن را با توجه به پیکسل های ورودی تصویر خود محاسبه می کنیم. بنابراین، نمونه خصمانه یک تصویر اساساً شیب از دست دادن نسبت به تصویر است. ما فقط یک مرحله گرادیان برمی داریم و بنابراین از نظر محاسباتی موثر است. بعد از محاسبه گرادیان ها علامت آن را می گیریم.

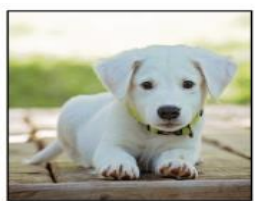
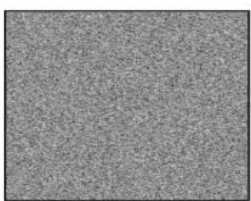

An adversarial image can be calculated as follows:

$$X_{adv} = x + \epsilon \operatorname{sign}(\nabla_x J(x, y_{true}))$$

In the previous equation, the following applies:

- $x_{adv}$  is the adversarial image
- $x$  is the input image
- $\nabla_x J(x, y_{true})$  is the gradient of loss with respect to our input image

همانطور که در نمودار زیر می بینید، ما یک تصویر ورودی  $x$  داریم و با اضافه کردن علامت گرادیان از دست دادن خود نسبت به تصویر خود، به تصویر واقعی، تصویر رقیب را دریافت می کنیم:

	+	$\epsilon$	$\times$		=	
Actual image (x)				$\operatorname{sign}(\nabla_x J(\theta, x, y))$		Adversarial image ( $x_{adv}$ )



## :ADML

اکنون که دیدیم نمونه های متضاد چیست و چگونه نمونه های متضاد را تولید می کنیم، خواهیم دید که چگونه از این نمونه های مخالف در یادگیری متا استفاده کنیم. ما مدل فرا یادگیری خود را با نمونه های تمیز و نمونه های مخالف آموزش می دهیم. اما نیاز به آموزش مدل با نمونه های متضاد چیست؟ این به ما کمک می کند تا پارامتر مدل قوی  $\theta$  را پیدا کنیم. هر دو نمونه تمیز و متخاصم در حلقه های داخلی و خارجی الگوریتم استفاده می شوند و به طور مساوی به روز رسانی پارامتر مدل کمک می کنند. ADML از این همبستگی متفاوت بین نمونه های تمیز و متخاصم برای بدست آوردن مقداردهی اولیه بهتر و قوی از پارامترهای مدل استفاده می کند تا پارامتر ما برای نمونه های متخاصم قوی شود و به خوبی به وظایف جدید تعمیم یابد. بنابراین، هنگامی که ما یک توزیع وظیفه ای  $p(T)$  داریم، دسته ای از وظایف  $T_i$  را از توزیع وظیفه نمونه برداری می کنیم و برای هر کار،  $k$  نقطه داده را نمونه برداری می کنیم و مجموعه های  $train$  و  $test$  خود را آماده می کنیم.

In ADML, we sample clean and adversarial samples for both train and test sets as  $D_{clean_i}^{train}$ ,  $D_{adv_i}^{train}$ ,  $D_{clean_i}^{test}$ , and  $D_{adv_i}^{test}$ .

اکنون تلفات را در مجموعه  $train$  خود محاسبه می کنیم، تلفات را با نزول گرادیان به حداقل می رسانیم و پارامتر بهینه  $\theta'$  را پیدا می کنیم. از آنجایی که ما مجموعه های  $train$  تمیز و متخاصم داریم، در هر دوی این مجموعه ها نزول گرادیان را انجام می دهیم و پارامتر بهینه را برای هر دو مجموعه تمیز و متخاصم پیدا می کنیم.

adversarial sets as  $\theta'_{clean_i}$  and  $\theta'_{adv_i}$  respectively:

$$\theta'_{clean_i} = \theta - \alpha_1 \nabla_{\theta} L_{T_i}(f_{\theta}, D_{clean_i}^{train})$$

$$\theta'_{adv_i} = \theta - \alpha_2 \nabla_{\theta} L_{T_i}(f_{\theta}, D_{adv_i}^{train})$$

Now, we go to the meta training phase where we find the optimal parameter  $\theta$  by minimizing loss on the test set by computing gradient of our loss with respect to optimal parameter  $\theta'$  obtained in the previous step.

So, we update our model parameter  $\theta$  by minimizing loss on both clean  $D_{clean_i}^{test}$  and adversarial  $D_{adv_i}^{test}$  test sets by computing gradient of loss with respect to an optimal parameter  $\theta'_{clean_i}$  and  $\theta'_{adv_i}$ :

$$\theta = \theta - \beta_1 \nabla_{\theta} \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta'_{clean_i}}, D_{clean_i}^{test})$$

$$\theta = \theta - \beta_2 \nabla_{\theta} \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta'_{adv_i}}, D_{adv_i}^{test})$$

## :CAML

ما دیدیم که چگونه MAML پارامتر اولیه بهینه یک مدل را پیدا می کند تا بتواند به راحتی با یک کار جدید با مراحل گرادین کمتر سازگار شود. اکنون، یک نوع جالب از MAML به نام CAML را خواهیم دید. ایده CAML بسیار ساده است، مانند MAML. همچنین سعی می کند پارامتر اولیه بهتر را پیدا کند. ما یاد گرفتیم که چگونه MAML از دو حلقه استفاده می کند. در حلقه داخلی، MAML پارامتر مخصوص کار را می آموزد و سعی می کند با استفاده از گرادین نزول تلفات را به حداقل برساند و در حلقه بیرونی، پارامتر مدل را به روزرسانی می کند تا تلفات مورد انتظار را در چندین کار کاهش دهد تا بتوانیم از مدل به روز شده استفاده کنیم. پارامتر به عنوان مقداردهی اولیه بهتر برای وظایف مرتبط.

در CAML، ما یک تغییر بسیار کوچک در الگوریتم MAML انجام می دهیم. در اینجا، به جای استفاده از یک پارامتر مدل، پارامتر مدل خود را به دو قسمت تقسیم می کنیم:

پارامتر زمینه Context parameter: این پارامتر مختص کار است که در حلقه داخلی به روز می شود. آن را با  $\theta$  نشان می دهند و برای هر کار خاص است و نشان دهنده جاسازی یک کار فردی است.

پارامتر مشترک Shared parameter: در بین وظایف به اشتراک گذاشته می شود و در حلقه بیرونی به روز می شود تا پارامتر مدل بهینه را پیدا کند. با  $\theta$  نشان داده می شود.

بنابراین، پارامتر زمینه در حلقه داخلی برای هر کار تطبیق داده می شود و پارامتر مشترک در بین وظایف به اشتراک گذاشته می شود و برای آموزش متا در یک حلقه بیرونی استفاده می شود. قبل از هر مرحله تطبیق، پارامتر زمینه را صفر می کنیم. باشه؛ اما واقعاً چه چیزی در تقسیم پارامتر ما به دو پارامتر مختلف مفید است؟ برای جلوگیری از تطبیق بیش از حد با توجه به وظایف خاص استفاده می شود، یادگیری سریعتر را ترویج می کند و حافظه کارآمد است.

# CAML algorithm

Now, let's see how CAML works steps by step:

1. Let's say we have a model  $f$  parameterized by a parameter  $\theta$  and we have a distribution over tasks  $p(T)$ . First, we randomly initialize the model parameter  $\theta$ . We also initialize our context parameter  $\phi_0 = 0$ .
2. Now, we sample some batch of tasks  $T_i$  from a distribution of tasks, that is,  $T_i \sim p(T)$ .
3. **Inner loop:** For each task ( $T_i$ ) in tasks ( $T$ ), we sample  $k$  data points and prepare our train and test datasets:

$$D_i^{train} = \{(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)\}$$

$$D_i^{test} = \{(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)\}$$

Now, we set our context parameter to 0:

$$\phi_0 = 0$$

Then, we calculate loss on  $D_i^{train}$ , minimize the loss using gradient descent, and learn the task specific parameter  $\phi_i$ :

$$\phi_i = \phi_0 - \alpha \nabla_{\phi} L_{T_i}(f_{\phi_0, \theta})$$

4. **Outer loop:** Now, we perform meta optimization in the test set—that is, here we try to minimize the loss in the test set  $D_i^{test}$  and find the optimal parameter:

$$\theta = \theta - \beta \nabla_{\phi} \sum_{T_i \sim p(T)} L_{T_i}(f_{\phi_i, \theta})$$

5. Repeat steps 2 to step 4 for  $n$  number of iterations.

## **Meta-SGD و Reptile :**

در فصل آخر، یاد گرفتیم که چگونه از MAML برای یافتن یک پارامتر بهینه که در چندین کار قابل تعمیم است استفاده می شود. ما دیدیم که MAML چگونه این پارامتر بهینه را با محاسبه متا گرادینان و انجام بهینه سازی متا محاسبه می کند. ما همچنین متا یادگیری متا مخالف را دیدیم که با افزودن نمونه های متخاصم به عنوان یک پیشرفت برای MAML عمل می کند و به MAML اجازه می دهد بین نمونه های تمیز و رقیب دست و پنجه نرم کند تا پارامتر بهینه را پیدا کند. ما همچنین شاهد CAML یا سازگاری زمینه برای فرا یادگیری بودیم. در این فصل، ما در مورد Meta-SGD، یکی دیگر از الگوریتم های یادگیری متا که برای انجام سریع یادگیری استفاده می شود، یاد خواهیم گرفت. برخلاف MAML، Meta-SGD نه تنها پارامتر بهینه را پیدا می کند، بلکه نرخ یادگیری بهینه و جهت به روز رسانی را نیز پیدا می کند. نحوه استفاده از Meta-SGD را در تنظیمات یادگیری تحت نظارت و تقویتی خواهیم دید. همچنین خواهیم دید که چگونه Meta-SGD را از ابتدا بسازیم. در ادامه، با الگوریتم Reptile آشنا خواهیم شد که باعث بهبود MAML می شود. خواهیم دید که Reptile چه تفاوتی با MAML دارد و سپس نحوه استفاده از Reptile در وظایف رگرسیون موج سینوسی را پوشش خواهیم داد.

در این فصل با موارد زیر آشنا خواهید شد:

۱. Meta-SGD
۲. Meta-SGD in supervised learning
۳. Meta-SGD in reinforcement learning
۴. Building Meta-SGD from scratch
۵. Reptile
۶. Sine wave regression using Reptile

## **: Meta-SGD**

فرض کنید یک وظیفه داریم،  $T$ . از یک مدل  $f$  استفاده می کنیم که با پارامتر  $\theta$ ، پارامتر شده است، و مدل را برای به حداقل رساندن تلفات  $\text{train}$  می دهیم. ما تلفات را با استفاده از گرادینان نزول به حداقل می رسانی و پارامتر بهینه را برای مدل پیدا می کنیم.

بیایید قانون به روز رسانی یک نزول گرادینان را به یاد بیاوریم:

$$\theta = \theta - \alpha \nabla_{\theta} L_{T_i}(f_{\theta})$$

So, what are the key elements that make up our gradient descent? Let's see:

- Parameter  $\theta$
- Learning rate  $\alpha$
- Update direction

**آموزش یادگیری در فضای مفهومی *Learning to learn in concept space*:**

اکنون خواهیم دید که چگونه یادگیری را در فضای مفهومی با استفاده از یادگیری متا عمیق یاد بگیریم. اول، چگونه یادگیری متا را انجام دهیم؟ ما مجموعه ای از وظایف مرتبط و چند  $K$  نقطه داده را در هر کار نمونه برداری می کنیم و فراگیرنده خود را آموزش می دهیم. به جای آموزش با استفاده از تکنیک متا یادگیری وانیلی، می توانیم قدرت یادگیری عمیق را با یادگیری متا ترکیب کنیم. بنابراین، وقتی دسته ای از کارها و تعدادی  $k$  نقطه داده را در هر کار نمونه می گیریم، نمایش هر یک از  $k$  نقطه داده را با استفاده از شبکه های عصبی عمیق می آموزیم و سپس فرا یادگیری را روی آن نمایش ها انجام می دهیم.

چارچوب ما از سه جزء تشکیل شده است:

۱. Concept generator

۲. Concept discriminator

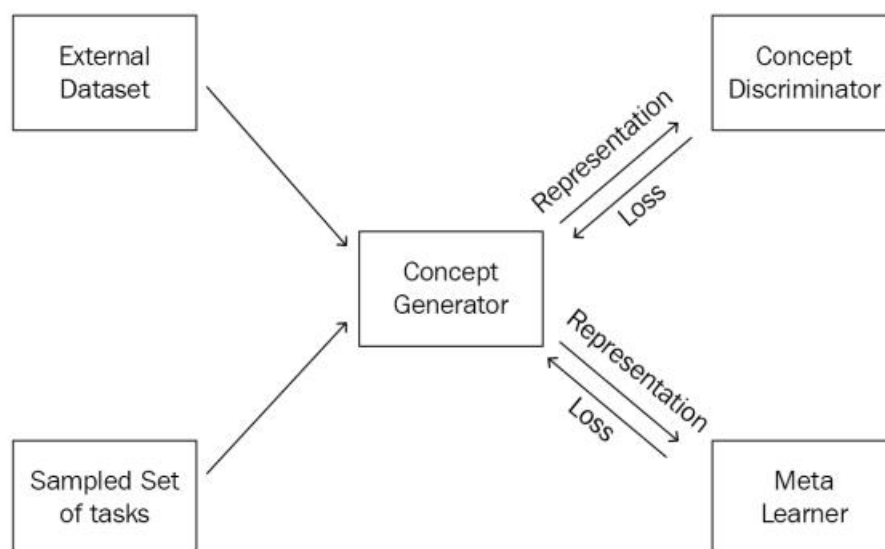
۳. Meta learner

نقش مولد مفهوم استخراج نمایش های ویژگی هر یک از نقاط داده در مجموعه داده ما است، و مفهوم سطح بالای آن را به تصویر می کشد، و نقش تمایزکننده مفهوم شناسایی و طبقه بندی مفاهیم تولید شده توسط مولد مفهوم است.

در حالی که فراگیرنده مفاهیم تولید شده توسط مولد مفهوم را یاد می گیرد. همه مؤلفه های قبلی - یعنی مولد مفهوم، تفکیک کننده مفهوم و فراگیرنده - با هم یاد می گیرند. بنابراین، ما یادگیری متا وانیلی را با ادغام یادگیری متا با یادگیری عمیق بهبود می بخشیم. مولد مفهوم ما با داده های ورودی جدید تکامل می یابد تا بتوانیم چارچوب خود را به عنوان یک سیستم یادگیری مادام العمر مشاهده کنیم.

اما واقعا اینجا چه خبر است؟ به نمودار زیر توجه کنید؛ همانطور که می بینید، مجموعه ای از کارها را نمونه برداری می کنیم و آنها را به مولد مفهوم خود می خوریم، که مفاهیم را یاد می گیرد - یعنی جاسازی ها - و سپس آن مفاهیم را به فراگیرنده می دهد، که بر اساس این مفاهیم می آموزد و ضرر را به عقب می فرستد. مولد مفهوم در همین حال، ما همچنین مقداری مجموعه داده خارجی را به مولد مفهوم تغذیه می کنیم، که مفاهیم مربوط به آن ورودی ها را یاد می گیرد و آن مفاهیم را به تفکیک کننده مفهوم می فرستد. تمایزکننده مفهوم،

برچسب‌های آن مفاهیم را پیش‌بینی می‌کند، ضرر را محاسبه می‌کند و ضرر را به مولد مفهوم باز می‌فرستد. با انجام این کار، ما توانایی مولد مفهوم خود را برای تعمیم مفاهیم افزایش می‌دهیم:



اما هنوز چرا این کار را می‌کنیم؟ به جای اجرای متا یادگیری بر روی یک مجموعه داده خام، فرا یادگیری را در فضای مفهومی انجام می‌دهیم. چگونه این مفاهیم را یاد بگیریم؟ این مفاهیم توسط مولد مفهوم با یادگیری تعبیه‌های ورودی تولید می‌شوند. بنابراین، ما سازنده مفهوم و فراگیرنده را در مورد وظایف مختلف مرتبط آموزش می‌دهیم. همراه با این، ما مولد مفهوم را از طریق تفکیک کننده مفهوم با تغذیه یک مجموعه داده خارجی به مولد مفهوم بهبود می‌دهیم تا بتواند مفاهیم را بهتر یاد بگیرد. این فرآیند آموزشی مشترک به مولد مفهوم ما اجازه می‌دهد تا مفاهیم مختلف را بیاموزد و در وظایف مرتبط بهتر عمل کند. ما مجموعه داده خارجی را فقط برای افزایش عملکرد مولد مفهوم خود تغذیه می‌کنیم، که به طور مداوم وقتی مجموعه جدیدی از ورودی‌ها را تغذیه می‌کنیم، یاد می‌گیرد. بنابراین، این یک سیستم یادگیری مادام‌العمر است.

# Assessments

## Chapter 1: Introduction to Meta Learning

1. Meta learning produces a versatile AI model that can learn to perform various tasks without having to be trained from scratch. We train our meta learning model on various related tasks with a few data points, so for a new but related task, the model can make use of what it learned from the previous tasks without having to be trained from scratch.
2. Learning from fewer data points is called **few-shot learning** or **k-shot learning**, where  $k$  denotes the number of data points in each of the classes in the dataset.
3. In order to make our model learn from a few data points, we will train it in the same way. So, when we have a dataset  $D$ , we sample some data points from each of the classes present in our dataset and we call it the support set.
4. We sample different data points from each of the classes that differ from the support set and call it the query set.
5. In a metric-based meta learning setting, we will learn the appropriate metric space. Let's say we want to find out the similarities between two images. In a metric-based setting, we use a simple neural network, which extracts the features from the two images and finds the similarities by computing the distance between the features of those two images.
6. We train our model in an **episodic fashion**; that is, in each episode, we sample a few data points from our dataset  $D$ , and prepare our support set and learn on the support set. So, over a series of episodes, our model will learn how to learn from a smaller dataset.



---

## Chapter 2: Face and Audio Recognition Using Siamese Networks

1. A siamese network is a special type of neural network, and it is one of the simplest and most commonly used one-shot learning algorithms. Siamese networks basically consist of two symmetrical neural networks that share the same weights and architecture and are joined together at the end using an energy function,  $E$ .
2. The contrastive loss function can be expressed as follows:

$$\text{Contrastive Loss} = Y(E)^2 + (1 - Y)\max(\text{margin} - E, 0)^2$$

In the preceding equation, the value of  $Y$  is the true label, which will be 1 when the two input values are similar and 0 if the two input values are dissimilar, and  $E$  is our energy function, which can be any distance measure. The term **margin** is used to hold the constraint; that is, when two input values are dissimilar and if their distance is greater than a margin, then they do not incur a loss.

3. The energy function tells us how similar the two inputs are. It is basically any similarity measure, such as Euclidean distance and cosine similarity.
4. The input to the siamese networks should be in pairs,  $(X_1, X_2)$ , along with their binary label,  $Y \in (0, 1)$ , stating whether the input pairs are genuine pairs (the same) or imposite pairs (different).
5. The applications of siamese networks are endless; they've been stacked with various architectures for performing various tasks, such as human action recognition, scene change detection, and machine translation.

---

## Chapter 3: Prototypical Networks and Their Variants

1. Prototypical networks are simple, efficient, and one of the most popularly used few-shot learning algorithms. The basic idea of the prototypical network is to create a prototypical representation of each class and classify a query point (new point) based on the distance between the class prototype and the query point.
2. We compute embeddings for each of the data points to learn the features.
3. Once we learn the embeddings of each data point, we take the mean embeddings of data points in each class and form the class prototype. So, a class prototype is basically the mean embeddings of data points in a class.
4. In a Gaussian prototypical network, along with generating embeddings for the data points, we add a confidence region around them, which is characterized by a Gaussian covariance matrix. Having a confidence region helps to characterize the quality of individual data points, and it is useful with noisy and less homogeneous data.
5. Gaussian prototypical networks differ from vanilla prototypical networks in that in a vanilla prototypical network, we learn only the embeddings of a data point, but in a Gaussian prototypical network, along with learning embeddings, we also add a confidence region to them.
6. The radius and diagonal are the different components of the covariance matrix used in a Gaussian prototypical network.

## Chapter 4: Relation and Matching Networks Using TensorFlow

1. A relation network consists of two important functions: the embedding function, denoted by  $f_\varphi$ , and the relation function, denoted by  $g_\phi$ .
2. Once we have the feature vectors of the support set,  $f_\varphi(x_i)$ , and query set,  $f_\varphi(x_j)$ , we combine them using an operator,  $Z$ . Here,  $Z$  can be any combination operator; we use concatenation as an operator to combine the feature vectors of the support set and the query set—that is,  $Z(f_\varphi(x_i), f_\varphi(x_j))$ .

- 
3. The relation function,  $g_\phi$ , will generate a relation score ranging from 0 to 1, representing the similarity between samples in the support set,  $x_i$ , and samples in the query set,  $x_j$ .
  4. Our loss function can be represented as follows:

$$\varphi, \phi < -\operatorname{argmin}_{\phi, \varphi} \sum_{i=1}^m \sum_{j=1}^n (r_{i,j} - 1(y_i == y_j))^2$$

5. In matching networks, we use two embedding functions,  $f$  and  $g$ , to learn the embeddings of the query set  $\hat{x}$  and the support set  $x_i$ , respectively.
6. The output,  $\hat{y}$ , for the query point,  $\hat{x}$ , can be predicted as follows:

$$\hat{y} = \sum_{i=1}^k a(\hat{x}, x_i) y_i$$

## Chapter 5: Memory-Augmented Neural Networks

1. NTM is an interesting algorithm that has the ability to store and retrieve information from memory. The idea of NTM is to augment the neural network with external memory—that is, instead of using hidden states as memory, it uses external memory to store and retrieve information.
2. The controller is basically a feed-forward neural network or recurrent neural network. It reads from and writes to memory.
3. The read head and write head are the pointers containing addresses of the memory that it has to read from and write to.
4. The memory matrix or memory bank, or simply the memory, is where we will store the information. Memory is basically a two-dimensional matrix composed of memory cells. The memory matrix contains  $N$  rows and  $M$  columns. Using the controller, we access the content from the memory. So, the controller receives input from the external environment and emits the response by interacting with the memory matrix.
5. Location-based addressing and content-based addressing are the different types of addressing mechanisms used in NTM.

- 
6. An interpolation gate is used to decide whether we should use the weights we obtained at the previous time step,  $w_{t-1}^c$ , or use the weights obtained through content-based addressing,  $w_t^c$ .
  7. Computing the least-used weight vector,  $w_t^{lu}$ , from the usage weight vector,  $w_t^u$ , is very simple. We simply set the index of the lowest value usage weight vector to 1 and the rest of the values to 0, as the lowest value in the usage weight vector means that it is least recently used.

## Chapter 6: MAML and Its Variants

1. MAML is one of the recently introduced and most commonly used meta learning algorithms, and it has lead to a major breakthrough in meta learning research. The basic idea of MAML is to find better initial parameters so that, with good initial parameters, the model can learn quickly on new tasks with fewer gradient steps.
2. MAML is model agnostic, meaning that we can apply MAML for any models that are trainable with gradient descent.
3. ADML is a variant of MAML that makes use of both clean and adversarial samples to find the better and robust initial model parameter,  $\theta$ .
4. In FGSM, we get the adversarial sample of our image and we calculate the gradients of our loss with respect to our image, more clearly input pixels of our image instead of the model parameter.
5. The context parameter is a task-specific parameter that's updated on the inner loop. It is denoted by  $\varnothing$  and it is specific to each task and represents the embeddings of an individual task.
6. The shared parameter is shared across tasks and updated in the outer loop to find the optimal model parameter. It is denoted by  $\theta$ .

## Chapter 7: Meta-SGD and Reptile Algorithms

1. Unlike MAML, in Meta-SGD, along with finding optimal parameter value,  $\theta$ , we also find the optimal learning rate,  $\alpha$ , and update the direction.
2. The learning rate is implicitly implemented in the adaptation term. So, in Meta-SGD, we don't initialize a learning rate with a small scalar value. Instead, we initialize them with random values with the same shape as  $\theta$  and learn them along with  $\theta$ .

- 
3. The update equation of the learning rate can be expressed as
 
$$\alpha = \alpha - \beta \nabla_{\theta} \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta_i})$$
  4. Sample  $n$  tasks and run SGD for fewer iterations on each of the sampled tasks, and then update our model parameter in a direction that is common to all the tasks.
  5. The reptile update equation can be expressed as  $\theta = \theta + \epsilon(\theta' - \theta)$ .

## Chapter 8: Gradient Agreement as an Optimization Objective

1. When the gradients of all tasks are in the same direction, then it is called gradient agreement, and when the gradient of some tasks differ greatly from others, then it is called gradient disagreement.
2. The update equation in gradient agreement can be expressed as
 
$$\theta = \theta - \beta \sum_i w_i \nabla L_{T_i}(f_{\theta_i})$$
3. Weights are proportional to the inner product of the gradients of a task and the average of gradients of all of the tasks in the sampled batch of tasks.
4. The weights are calculated as follows:

$$w_i = \frac{\sum_{j \in T} (g_i^T g_j)}{\sum_{k \in T} |\sum_{j \in T} (g_k^T g^j)|}$$

5. The normalization factor is proportional to the inner product of  $g_i$  and  $g_{average}$ .
6. If the gradient of a task is in the same direction as the average gradient of all tasks in a sampled batch of tasks, then we can increase its weights so that it'll contribute more when updating our model parameter. Similarly, if the gradient of a task is in the direction that's greatly different from the average gradient of all tasks in a sampled batch of tasks, then we can decrease its weights so that it'll contribute less when updating our model parameter.



---

## Chapter 9: Recent Advancements and Next Steps

1. Different types of inequality measures are Gini coefficients, the Theil index, and the variance of algorithms.
2. The Theil index is the most commonly used inequality measure. It's named after a Dutch econometrician, Henri Theil, and it's a special case of the family of inequality measures called **generalized entropy measures**. It can be defined as the difference between the maximum entropy and observed entropy.
3. If we enable our robot to learn by just looking at our actions, then we can easily make the robot learn complex goals efficiently and we don't have to engineer complex goal and reward functions. This type of learning—that is, learning from human actions—is called imitation learning, where the robot tries to mimic human action.
4. A concept generator is used to extract features. We can use deep neural nets that are parameterized by some parameter,  $\theta_G$ , to generate the concepts. For examples, our concept generator can be a CNN if our input is an image.
5. We sample a batch of tasks from the task distributions, learn their concepts via the concept generator, perform meta learning on those concepts, and then we compute the meta learning loss:

$$L_T(\theta_M, \theta_G)$$