وسط جلسه ۱۷ از دقیقه ۵۵:

Bag of word: در این شیوه مجموعه کلمات متن به صورت برداری خواهد بود یعنی به هر کلمه یک بردار نسبت بدهیم و feature بشود جمع این بردار ها و کارایی ندارد چون ترتیب دنبال متن را حفظ نمیکند.

One-hot: مشکلش این است که ابعاد آن بالا هست به ازای هر کلمه یک بردار به اندازه دیکشنری خودمان داریم که در هر بردار فقط یکی ۱ هست (کلمه متناظر با آن) و بقیه ۰ هستند. sparse هستند یعنی یک مقدار زیادی صفر دارند.

یک روش دیگر این است که بیایم به ازای هر کلمه یک featureهایی در نظر بگیریم و برداری که برای آن کلمه در نظر میگیریم میتواند مفید باشد و بردار ها مفهوم میگیرند و اختلاف مرد با زن مانند اختلاف پادشاه با ملکه هست. به این روش word embedding گفته میشود.

Word embedding: dense هستند یعنی ابعاد کوچولو هست ولی information دارند و قابل یادگیری هستند این feature هایی بردار ها در این روش و relation بین کلمات میتواند یاد گرفته شود مثلا پرتغال دید احتمالا بعدش آب پرتغال هست نه فیل. دقت کن این بردار ها قابل یادگیری هستند و در کنار کار اصلی که مثلا classification روی متن ها هست صورت میگیرد در کنار آن word embedding صورت بگیرد برای بدست آمدن بردار ها. یک روش دیگر استفاده از مدل های قبلی هست که قبلا آموزش داده شده اند. یعنی بردار نسبت به کلمات اتلاق میشود.

برای بدست آوردن این بردار ها مثلا میتوانیم یک پنجره در نظر بگیریم که هر خانه یک کلمه است و رابطه بین هر خانه را بدست میاوریم یک پنجره لغزان است که دارد حرکت میکند و ۲ به ۲ ارتباط بین کلمات را بدست میاوریم. خروجی شبکه به اندازه بردار ورودی هست و از SoftMax classifier استفاده شده است به این معنی که خروجی یکی همیشه ۱ خواهد بود. نقش لایه وسطی چی خواهد بود؟ بر طبق پنجره هر سری یکی از لایه خروجی دارد فعال میشود پس ما یک متن بزرگ بدهیم انگار این لایه وسطی دارد یادگیری انجام میدهد و در نهایت یک feature vector dense به ما میدهد. یعنی اکثر بردار ما غیر صفر است و حاوی اطلاعات است. این لایه مخفی عین look up است یعنی ما ۱۰ هزار تا کلمه داریم به ازای هر کلمه

۳۰۰ تا feature داریم یا نورون که هر کدام یک بردار میشود و بعد اون روش one hot را که یک بردار هست در این ماتریس ضرب میکنیم و خروجی متناظر با one hot میاد و dense آن را میدهد و اطلاعات راجب اون کلمه هست و ارتباط بین کلمات را در خودش نگه میدارد به جای اینکه sparse باشد و کلی مقدار ۰ داشته باشیم.

مشکل اینجا هست که اگر بر اساس کلمات فقط embedding کنیم بعد میانگین مثلا بگیریم بین feature ها و بعد classify کنیم شاید زیاد دقیق نباشد که میانگین بگیریم از feature vector. چون هیچ relation لحاظ نکردیم و صرفا داریم میانگین میگیریم.

Sigmoid میزنیم معمولا binary classification است. یک تعداد لغات پر کاربرد داریم یک تعداد بردار داریم و یک عددی هم داریم برای اینکه مثلا بدانیم چند تا کلمه اول را در نظر بگیریم.

تعداد پارامتر embedding؟ تعداد کلمات پر کاربرد ضربدر مقداری که برای هر بردار در نظر گرفتیم مثلا گفتیم بردار هایی ۸ تایی داشته باشیم یعنی embedding ما چند تایی باشد.

مدل Stanford بیش بردازش شده است مشکل اینجا هست که ما یک سری بردار داریم برای هر کلمه و به عنوان ورودی به شبکه میدهیم و ارتباطی بین آن نیست. یکی از روش ها این است که برای هر کلمه یک con 1بعدی است برای اینکه relation زمانی یا ارتباط آنها را بدست بیاوریم حالا مثلا در اینجا روی یک بعد و فقط روی زمان حرکت بکند به جای ۲ بعدی که عمق ثابت بود، یعنی فقط روی یک محور حرکت میکند.

VGG مقدار feature را جستجو کن.

عمق conv 1 بعدی به اندازه هم عمق بردار است مثلا ۵۱۲. دقت کن وقتی فیلتر میزنی عدد اسکالر میشود وقتی روی یک بعد میزنی در نتیجه خروجی میشود ٤٠ در ۱ یا یک بردار ٤٠ تایی. اینجا relation ۳ تایی زدیم با کانولوشن یک بعدی بخاطر اون فیلتر و پیچیدگی زیاد ندارد و با این فیلتر در واقع ارتباط اون ۳ تا را بدست آوردیم یعنی relation زمانی بدست آوردیم و دقت کن بعد از این کانولوشن یک relation دیگه میتونی بزنی با کانولوشن یک بعدی میتوانیم الگو ها و ارتباطات بیشتری را بدست بیاوریم. دقت کن فرقی نمیکند ٤٠ تا کلمه باشد یا ۸۰ تا میخواهیم یک feature هم سایز بدست بیاوریم و مسئله این است.

دقت کن feature vector یکتا باید باشد و global باشد و دیگر سایز متن فرقی نمیکند ۴۰ تا کلمه باشد یا ۸۰ تا.

شبکه های عصبی feedforward: در این شبکه ها قابلیت پردازش دنباله ها با طول های متفاوت نداشتیم و ترتیبی در پردازش نداشتیم و هر ورودی به صورت مستقل بدون در نظر گرفتن سایر ورودی ها محاسبه میشد و حالا نیاز داریم relation بین اینها داشته باشیم یعنی خروجی این لحظه در ورودی لحظه بعدی تاثیر گذار خواهد بود. H حالت سیستم هست که با خروجی یکی هست. و یک feedbackیی داریم.

شبکه های RNN:

حالا این feedback در این شبکه ها بدرد میخورد. ماتریس U وزن های اون خروجی هستند در لحظه قبل. یعنی خروجی اولی به همراه ورودی لحظه بعدی خروجی بعدی بدست و این خروجی به همراه ورودی لحظه بعدی خروجی بعدی بدست میاد و این روش برای پردازش سری های زمانی و دنباله ها بسیار مناسب هستند و relation ها را میتوانیم در نظر بگیریم. انتخاب خروجی هم ۲ گزینه داریم: ۱. میتوانیم دنباله همه خروجی را در نظر بگیریم یا صرفا فقط آخرین خروجی را در نظر بگیریم.

One to one: یک ورودی داریم یک feedbackیی هم میگیریم به عنوان state و خروجی محاسبه میشود.

One to many: یک ورودی بهش میدهیم و خروجی ها در لحظه های بعدی وارد میشوند به عنوان ورودی در لحظه بعدی تاثیر گذار هستند یعنی یک ورودی داریم و یک دنباله بدست میاریم.

Many to one: یک سری ورودی میدهیم و فقط یک خروجی میگیریم.

Many to many: یک جمله به یک زبانی بگیرد و ترجمه کند به زبان دیگر حالا میتواند طول ورودی با خروجی برابر باشد یا برابر نباشد اگر برابر نبود یک ورودی به عنوان encoder میسازیم feature بدست میاد بعد به decoder میدهیم.

Many to one مثل همین review رستوران. One to many مثل تولید کننده شعر یا image captioning.

جلسه ۱۸:

با word embedding میخواستیم relation را از دیتا یاد بگیریم. خروجی skip-gram model لایه وسطی آموزش داده میشود و یک بردار dense به ما میدهد. در کنار کانولوشن یک بعدی که relation در نظر میگرفت، اشتراک پارامتر ها هم تاثیر گذار هستند در کنار اینها شبکه های feedforward که feedback دارند پیشنهاد شد و RNN مطرح شد.

خروجی: دنباله باشد مثل ترجمه یک زبان به زبان دیگر. تک خروجی مثل همین review رستوران. Y همان خروجی هست که به صورت feedback برمیگردد در لحظه بعدی پس وزنی که محاسبه به خودش ربط ندارد و به وزن و y لحظه قبل هم بستگی دارد.

Return sequence=true یعنی به ازای هر نورون ما میخواهیم خروجی بگیریم.

تعداد لایه ها بیشتر شود feature قوی تری بدست میاد.

یک مشکل RNN برای backpropagation through time یعنی در حین زمان این محاسبه میشود اگر دنباله های ما خیلی طولانی باشد باعث vanishing gradient میشود یا exploding gradient میشود ما دوست داریم مقدار وزن یک چیز متوسط باشد و بخاطر طول بلند این مشکلات برای گرادیان رخ میدهد. پس اگر طول بلند شد RNN خوب آموزش نمیبیند. و اصلا relation ها را نمیبیند و لحاظ نمیکند.

راهکارها:

Truncate backpropagation: در جاهایی که طول خیلی بلند میشود شما بیا یک پنجره در نظر بگیر.

LSTM: RNN ها وابستگی هایی long term را نمیتوانند در نظر بگیرند و فراموش میکنند برای حل این، LSTM مطرح شده است. این شبکه از skip connection استفاده میکند یا residual ها که میگفتند این اطلاعاتی که میخواهی اونجا استفاده کنی نگه دار ولی اگر این لایه اطلاعاتی خاصی اضافه نمیکند skip کن و خروجی همان ورودی بشود هم از این دیدگاه و هم از مکانیزم توجه

استفاده میکند. داخل آن گیت ها مشخص میکنند چه حرکتی صورت بگیرد. در این شبکه ۲ تا خروجی داریم یکی که h همان خروجی ما هست مثل RNN معمولی. گیت ها اینجا میگویند چه قدر اطلاعات از گیت ها رد بشوند و به cell state برسند، این همان حافظه بلند مدت هست که ممکن است یک چیز هایی بهش اضافه یا کم شود h مثل حافظه کوتاه مدت میماند برعکس c. چرا مثل RESNET هست؟ چونکه skip connection دارد و میتواند همان چیز که وارد شده است بدون تغییر خارج شود، که حالت سیستم هم گفته میشود c حافظه بلند مدت هست و state مشخص میکند چه قدر باید اطلاعات رد شود توسط گیت ها یعنی گیت ها مشخص میکنند. به اولین تابع فعالیت سیگموید forget gate گفته میشود که مشخص میکند چه مقدار از اطلاعات را میتوانیم فراموش بکنیم و چه قدر حفظ شود. در گام بعدی مشخص میکنیم چه اطلاعاتی به حالت بعدی اضافه شود یا input gate layer که ترکیب سیگموید و تانژانت هست و حالت جدید به روز میشود. در نهایت خروجی بر اساس ورودی و حالت به روز شده محاسبه میشود. و انگار مکانیزم توجه داریم چون میگوییم همه را با یک وزن و یک دید نگه ندار و بعضی چیزها را نگه میدارد و بیشتر توجه میکند و بعضی چیز هارا کمتر.

در نمونه های دیگر در نمونه دوم میگوید اون چیزی که فراموش کرده ایم را میتوانیم در حالت جدید لحاظش بکنیم. نمونه سوم GRU نام دارد که پیچیدگی خیلی پایینی دارد و اون cell state را ندارد.

دنباله–دنباله:

معمولا در ترجمه ماشینی مورد استفاده قرار میگیرد. در ساختار آن هم ورودی encode شده را داریم هم خروجی decode شده قبلی را داریم تا خروجی جدید را محاسبه بکنیم و decode کنیم. بعد از محاسبه encode شده عین همون RNN های خودمان است و ساختار مشابه دارد. فرق مدل زبانی و ماشین ترجمه: در مدل زبانی میخواهد بعد از حدس زدن کلمه اول به کمک آن کلمه دوم را حدس بزند و حدس آن توزیع احتمال روی کلمات هست و کلماتی را میخواهیم حدس بزنیم که بیشترین احتمال را داشته باشد که در ماشین ترجمه این شکلی نیست ما ورودی را encode میکنیم بعد در مرحله بعد میخواهیم decode کنیم به همراه خروجی قبلی برای انتخاب خروجی با احتمال بیشتر. در مدل زبانی که میخواهیم احتمال بیشینه شود یک احتمال شرطی هست یعنی بر اساس ورودی که در نظر میگیرد خروجی را محاسبه میکند.

ما باید خروجی را برداریم که بیشترین احتمال را دارد پس باید arg max را انتخاب بکنیم یا بیشینه آرگومان را. انتخاب کلمه اگر فعل متداول تر باشد احتمال انتخاب آن بیشتر است اگر بخواهیم برای هر کلمه فقط بیشترین احتمال را انتخاب بکنیم شاید کامل مناسب نباشد و رویکرد حریصانه باشد یک رویکرد دیگر این هست که خود کلمه شاید بیشینه احتمال نباشد ولی جمله ای که ساخته میشود دارای بیشینه احتمال است. به این beam search میگوید که b را میتوانیم یک عدد بگیریم و میایم b تا بزرگترین را انتخاب میکنیم این مرحله اول بعد در مرحله دوم بعد از انتخاب اون b تا دوباره دیکشنری را باز نویسی میکنیم دوباره b تا که هر جمیع b تا را بیشینه میکند انتخاب میکنیم. دفعه بعد هم دوباره همینطور یعنی دیکشنری را مینویسیم و بعد b تا را برمیداریم بر اساس همین بالاترین ها و کلمات قبلی که در دفعه قبلی انتخاب کرده ایم. هر چی b بزرگتر جمله بهتر ولی پیچیدگی هم بالاتر.

Blue score: Gt میشود ترجمه بهتر نسبت به خروجی ترجمه انسانی و ترجمه ماشین. و کلا با دنباله بزرگ مشکل داریم. با وجود LSTM ها باز هم مشکل دنباله داریم پس transformer ها پدید آمدند تا مشکل حافظه حل شود. یک مسئله دیگر این است که ما در روش های قبلی موازی سازی نداشتیم و چون خروجی گام به گام هست با افزایش طول حجم محاسبات هم میرود بالا تازه ترجمه خوبی هم نداریم و blue score میاد پایین و hardware friendly نیست.

یک روش دیگر bidirectional RNN هست یعنی دو جهته باشد یعنی برای حدس زدن Y2، Y3 هم داشته باشیم و میتواند تا حدودی جواب بهتری فراهم کند.

یک نکته مقاله مهم دیگر مکانیزم توجه خواهد بود. Intra یعنی داخل خودش مثل self attention.

کلا توجه میگوید مثلا در یک تصویر همه پیکسل ها اهمیت یکسانی ندارند و بعضی قسمت ها مهم تر هستند و برای دنباله های زمانی پیشنهاد شدند چرا از sotmax استفاده میکنیم؟ تا یک توزیع احتمالاتی بدهیم و بگیم همه این ورودی ها به اندازه یکسان اهمیت ندارد و بعضی از آنها با توجه به ضریبی که دارند و scale شده اند را در نظر میگیریم.

ضرب داخلی یا dot product تشابه را بیان میکند. بعد از sotmax رد میکنیم و احتمالاتی میشود تشابه بعد جمع وزن دار میکنیم خروجی بدست میاد و وزن ها مشخص میشوند و وزن از ضرب داخلی یک

کلمه با در نظر گرفتن همه بدست میاد. یعنی یک ورودی را گرفتی نسبت به بقیه ورودی ها میسنجی و خروجی را تولید میکنی. یعنی دنبال ارتباط یک کلمه را با کلمات دیگری حساب میکنیم و اونهایی که فقط برای ما مهم است را نگه میداریم. Conv 1 بعدی برای بدست آوردن ارتباطات است.

جلسه ۱۹:

میزان ارتباط کلمات با هم در لایه self attention محاسبه میشود نسبت به خود کلمه و نسبت به سایر کلمات آن جمله، و بعد به شبکه feedforward میدهیم تا خروجی بهتری تولید شود. امکان موازی سازی نیز داریم.

Query, key and value با استفاده از وزن هایی که در جریان آموزش بدست آمده است محاسبه میشوند برای هر کلمه این ۳ تا بردار را داریم و با استفاده از این ۳ attention محاسبه میشود.

Mask: در decoder هستی کلمات قبلی آمده اند اما کلمات بعدی نیامده است این رو اعمال میکنیم چون هنوز از encoder نیومده اند منفی بینهایت میگذاریم در آن پنجره و اگر از softmax رد شوند صفر میشوند یعنی میگوید اونجا ها را لحاظ نکن چون اطلاعاتی بعدی را که نداریم و ارتباط آن ها را. از SoftMax استفاده کردیم تا ارتباط بین اینها احتمالاتی شود. انگار داریم value های مختلف را با وزن های مختلف در نظر میگیریم. در یک ماتریس ۲*۲ ما ۲ کلمه را در نظر گرفتیم یکی اولی با خودش بعد اولی با دومی بعد دومی با اولی و بعد دومی با دومی. Z خروجی self-attention است.

هدف transformer چیست؟ فرآیند training را کوتاه میکند به علت موازی سازی که دارد. دقت کن وزن ها و ارتباطات در جریان train بدست میاد. موازی سازی به جای استفاده از بردار های بزرگ میشکند و به صورت موازی محاسبه میکند. X ورودی attention اولی و برای لایه های بعدی r ورودی attention های لایه بعدی خواهد بود.

Position encoding: اگر positionزوج بود سینوس در نظر میگیریم. میتوانیم به جای این ۲ تابع از طریق learning بدست بیاوریم.

Residual: اگر لایه اضافه کردی ورودی را با چیزی که رد میشد جمع میکرد اگر یاد میگرفت که هیچی اگر نه ورودی را مستقیم به خروجی میداد.

ورودی decoder ۲ چیز است یک اون چیزهایی که predict میکند علاوه بر relationهایی که در encoder بدست آورده ایم و برداری که encode شده است. دقت کن بعد از decode شدن i این دوباره به decoder برمیگردد تا بعدی را decode کنیم. در decoder کلمات جلوتر را نداریم به جز یک mask و دقت کن ما relation قبلی ها را داریم و decode هایی هم که قبلا شده وارد آن میشود. بعد از این مرحله میگیم دیکشنری ما هزار تا کلمه دارد ما خروجی decoder را میدهیم به linear و بر اساس سایز دیکشنری یک عددی بدست میاد یا یک logit بدست میاد، و یکی از اون کلمات را به عنوان پیشنهاد بهتر، پیشنهاد میدهد و بعد از SoftMax رد میکنی میگی هر کدام احتمالش بالاتر شد انتخاب کن. همه حرف transformer پیدا کردن یک شباهت هست و این شباهت ها را یاد میگیرد یعنی میفهمد بعد یک کلمه چه کلمه ای را باید پیشنهاد بدهد.

Autoencoder:

با استفاده از encoder نگاشت میشویم به یک فضای دیگه مثلا صفحه شطرنج ورودی ماست ولی میخواهیم مپ کنیم به یک فضای دیگه. Output و ورودی باید یکسان باشد و از بردار بدست آمده که سایزش کوچکتر از ورودی هست feature extractor میتوانیم استفاده کنیم و با استفاده از این داده ما میتوانیم ورودی را بسازیم پس representation خوبی برای ورودی است. این شبکه عصبی ورودی را به خروجی منتقل میکند. حاصل یک تابع همانی است یعنی ورودی را میگیرد و همان را به عنوان خروجی تحویل میدهد. autoencoder برای کاهش ابعاد مناسب هستند. برای data augmentation هم مورد استفاده قرار میگیرد برای ایجاد داده هایی که شبیه به داده های آموزشی هستند که بهش generative model گفته میشود.

PCA: وقتی میزنیم دنبال چی هستیم؟ دنبال principal component ها را میخواهیم یا مولفه های اصلی آن را. ستون های ماتریس A متعامد هست پس حاصل آن در transpose شده خودش I

خواهد شد. و داده ما که ۲ بعدی باشد اگر مولفه اصلی را بکنیم که ۱ بعدی هست انگار بیشترین اطلاعات را بدست آوردیم و کاهش ابعاد دادیم.

PCA خطی است و autoencoder غیر خطی هست و representation بهتری دارد.

PCA حالت خاصی از autoencoder است.

Stacked autoencoder وزن هایی که قبلا بدست آورده ایم را میتوانیم بیاوریم و زمان training و پیچیدگی ما کاهش پیدا میکند در واقع توی ۲ فاز داریم انجام میدهیم و بهتر است و سریعتر است. لایه های مخفی اولی چون ورودی را دارد همان خروجی تحویل میدهد بهتر هستند برای feature extraction و پارامتر ها را در فاز ۲ کپی میکنیم حالا label data بهش میدهیم و چون از label نخورده feature extract کردیم خوب جواب میدهد و بعد label data را برای آموزش classifier استفاده میکنیم که بهش پیش آموزش بی نظارت گفته میشود.

اگر gaussian noise استفاده کنی برای training بهتر است یا به صورت تصادفی یکی سری نورون را خاموش بکنی از روی بقیه که روشن هستند یا نویز ندارند حدس بزند اون تیکه هایی را که ندارد و به overfitting کمک میکند و در لایه مخفی useful feature یاد میگیرد که اونهایی هستند که به نویز کاری ندارد. کلا dropout برای چی بود؟ جلوگیری از overfit شدن که یک سری از نورون ها را خاموش میکردیم و از روی نورون های بعدی عملکرد داده را حدس بزنند.

Sparse autoencoder: قیدی بزاریم که ویژگی های متناسب و تنک بازنمایی بدست آید و ویژگی های گزیده بدست میاد.

VAE: بهش generative autoencoder گفته میشود؟ در latent space or coding space که در لایه های میانی هست ما یک توزیع گوسی داریم، یا توزیع داریم که یک توزیع تحویل میدهد ما یک نمونه از توزیع میگیریم به وسیله آن داده را در خروجی میسازیم بعد از decode شدن. پس چون میو و سیگما داریم نشان از توزیع است. وقتی نمونه از توزیع برمیداریم اگر از روی دقیقا میو برداریم دقیقا در خروجی همان توزیع ساخته میشود و اگر از میو اونور تر باشد یک مقدار تغییر پیدا میکند.

پس دقت کن ما یک discrete value و یک توزیع احتمالاتی داریم که مثلا هر چه قدر سمت راست برداریم این ناراحتی تبدیل به خندان تر شدن میشود.

Multi-head attention عین ensemble learning ها هستند که از وجوه مختلف بررسی میکنیم و بعد اینها را concate میکند.

جلسه ۲۰:

VAE: وسط در لایه های معمولی autoencoder داریم یا به نام latent attribute که مثلا یک سری feature vector داریم و هر سری از یک feature بررسی میکنیم، توزیع معمولا توزیع گوسی خواهد بود و یک discrete value داریم بعد به توزیع تبدیل میکنیم و میشود latent space که میتوانیم اینها را مثلا بالا پایین کنیم و اشکال مختلف ساخته شود در decoder باید چیکار کنیم؟ sample بگیریم به مرکزیت همین گوسی ها دقیقا همین شکل خواهد بود. اگر sample را روی گوسی ها تغییر بدیم انتظار داریم تصویری که درست میشود متفاوت باشد و تولید شود مثلا خندان تر باشد پس با sample گیری میتوانیم در decoder تصویر تولید کنیم. بعد sample گیری حالا یک vector داریم که از روی اون vector میتوانیم در decoder تصویر جدید بسازیم. پس در این VAE یک ورودی داریم میدهیم به encoder بعد از این توزیع بدست میاوریم این میتواند بعد های بیشتری داشته باشیم و هر کدام یک گوسی باشند به اصطلاح multivariative باشند(چرا گوسی میگیریم؟ اگر مرکز را بگیری چون حالت discrete دارد دقیقا به خروجی میرسیم حالا میگیم به مرکزیت این سنتر با یک واریانسی یک گوسی در نظر میگیریم و از هر جای این توزیع میتوانی sample بگیری و از اون توزیع نمونه گیری میکنیم بعد decode میکنیم بعد خروجی بدست میاد.

Information چطوری بدست میاد؟ مثلا اگر یک جمله ای وقوع آن حتمی باشد چه قدر information دارد؟ صفر است چون اطلاعاتی ندارد و محتمل است مثلا در تابستان هوا گرم است. حالا مثلا در تابستان هوا سرد است این اطلاعات بالاتری دارد.

Entropy میخواهد بگوید از هر الگوریتم فشرده سازی هم که استفاده کنی از یه حدی دیگه کمتر نخواهد شد.

Cross entropy: h(p,q) چون به حالت بهینه هنوز نرسیدیم که q به p برسد از h(p) بزرگتر خواهد بود.

KL-DIVERGENCE: برای مقایسه ۲ تا توزیع استفاده میشود به عنوان loss function استفاده میشود. انگار یک طوری تابع خطا ما هست. و سعی میکند دو تا توزیع را به هم نزدیک کند یعنی q را میخواهد به p نزدیک کند. KL کی منفی میشود؟ هیچوقت منفی نمیشود. و هدف این کمینه کردن cross entropy است.

P(z|x) توزیع latent space ما هست مثل همون گوسی ها میخواهیم توزیع آن را بدست بیاوریم. دقت کن ما توزیع latent space در لایه z رو میخواهیم و هدف ما این است و میدانیم این یک توزیع گوسی است. چون ۲ تا توزیع را میخواهیم به هم نزدیک و بدست بیاوریم و به هدف برسیم از KL میتوانیم استفاده کنیم.

در همان صفحه دقت کن که توزیعی که داریم نرمال است و واریانس و میانگین توابعی از X هستند که همان ورودی ها هستند و میخواهیم به P(Z|X) برسیم توسط بازی کردن با Q(X).

KL نمیتواند منفی شود و ما باید این را کمینه کنیم پس باید LOG P را بیشینه کنیم.

Autoencoder عادی قطعی و VAE احتمالی است. Z را میدهیم و X هت را میسازد.

در autoencoder ما یک نقطه داریم که به عنوان ورودی میدهیم و خروجی آن را بازسازی میکند.

چرا regularization میکنیم؟ به پیوستگی و تمامیت کمک میکند. پیوستگی یعنی دو تا نمونه که خیلی شبیه بهم هستند باید در latent space شباهت بین آنها وجود داشته باشد و ۲ تا خروجی کاملا متفاوت نباشد. به نوعی از regularization استفاده میکنیم تا حرکت نمونه ها و تولید خروجی به صورت آرام و تدریجی باشد.

اسلاید بهترین انتخاب از جلسه های امسال:

پس کلمه فعلی هم بر اساس ورودی فعلی هم بر اساس خروجی های قبلی انتخاب میشود و احتمال هر کدام بیشتر بود همان انتخاب میشود.

پس برای کلمه فعلی همه s ها و همه کلمات قبلی و c استفاده میشود. c تمامی اطلاعات encoder ها است. s همان state ما است که گام های decoding است.

دقت کن ما در نهایت کلمات را انتخاب میکنیم ما ترکیبات مختلف را نگه میداریم اون ترکیباتی را نگه میداریم که بیشترین احتمال را دارد مثل beam search که بین اونها در آخر وقتی به EOS رسیدیم میایم بهترین ترکیب را برمیداریم.

در transformer از s0 عبور میدهیم بعد میایم attention آنها را نسبت به همه کلمات محاسبه میکند.

برای هر کلمه جدید یک information جدید داریم یعنی یک c جدید داریم. و کلمه قبلی هم مورد استفاده قرار میگیرد. alignment score از softmax رد بشود میشود attention که نشان دهنده ارتباط هر خروجی encoder با گام فعلی decoder است.

با استفاده از attention میشود image captioning انجام داد بر اساس CNN یک سری feature map بدست آوردیم و میتوانیم بگیم اجزای این که در کنار هم هستند با همدیگر ارتباطی هم دارند. ما عکس را به یک سری patch تقسیم میکنیم و به جای اینکه بگیم relation کلمات به چه شکل هست میگوییم relation، patch ها به چه شکل است. e1,1,1 دو تا ۱ اول position و سومی یعنی step 1 است بعد از softmax رد میکنیم بعد وزن هر patch را بدست میاوریم بعد با feature map ضرب نظیر به نظیر میکند. این میشود c ما بعد با کمک s0 ما مثلا خروجی smiling decode میشود. بعد به سراغ position بعدی و گام بعدی میرویم و وزن های جدید بدست میاد با استفاده از s1 و اون alignment score مقدار c2 جدید ما بدست میاد.

دقت کن EOS در طی آموزش شدن بدست میاد. یعنی در طی آموزش میفهمیم اینجا انتهای جمله است.

ما اینجا از probability استفاده کردیم ما با کمک blue score ارزیابی میکنیم خروجی را.

جلسه ۲٤ امسال:

ما یک encoding داشتیم که یک c تولید میشد در حاصل از encoding ما. C اطلاعات ما از encoder هست و در هر گام با گام قبلی فرق میکند. این c های مختلف که هر سری وارد میشوند بر اساس informationیی که در هر گام داریم بدست میاد یعنی برای کلمه سوم وجود کلمه دوم اهمیت بیشتری دارد از بقیه و در جریان یادگیری این مورد یاد گرفته میشود.

جلسه ۲۵ امسال:

جلسه ۲٦ امسال: