به نام اول معلم گیتی

محمد علی آبادی

[redacted]

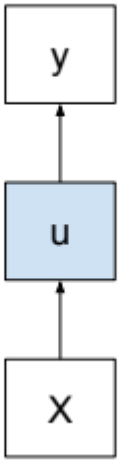
روزهای آخر پاییز 1400):

سوال اول :

کلا چهار مدل عمده برای پیش بینی دنباله ها داریم که در اینجا برای مرور خودم، همه ی این ها رو بررسی میکنیم.

One to one model :

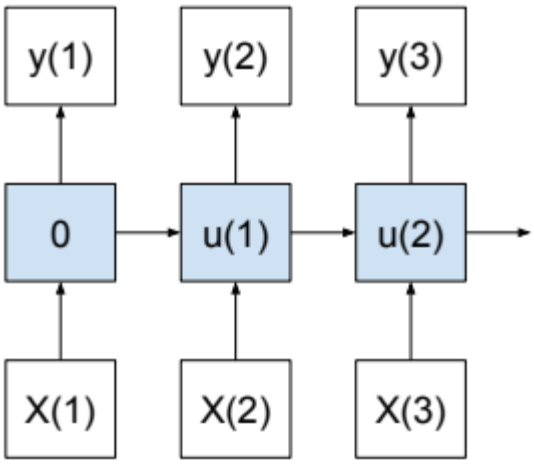
این مدل برای هر ورودی یک خروجی تولید میکنه .



یعنی در واقع میتونیم عملکرد این مدل رو به صورت زیر بنویسیم

```
y(1) = f(X(1))
y(2) = f(X(2))
y(3) = f(X(3))
...
```

هیدن استیت داخلی برای تایم استپ اول برابر صفر است و معمولا و به مرور زمان با دادن ورودی این ورودی های بعدی مقادیر مختلف این هایدن استیت روی هم انباشه میشه.



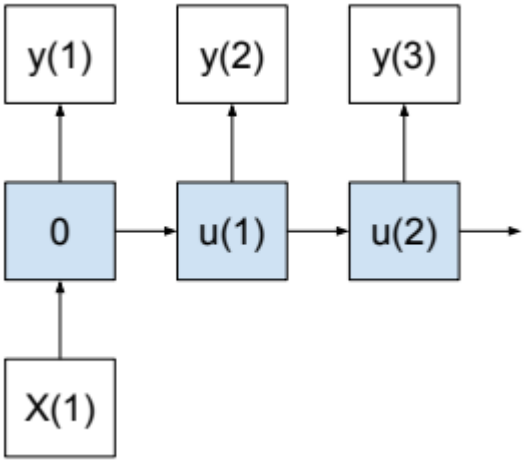
از این مدل وقتی استفاده میکنیم که مثلا بخوایم یک تایم استپ رو بر حسب یک ورودی پیش بینی کنیم. یعنی مثلا یک مقدار حقیقی در سری زمانی رو پیش بینی کنیم یا کلمه بعدی در جمله رو پیش بینی کنیم

One to many model :

این مدل چندین خروجی رو برای یک ورودی تولید میکنه

```
y(1),y(2) = f(X(1))
```

در واقع اگر زمان رو گسسته در نظر بگیریم و اومدن هر ورودی رو یک تایم استپ در نظر بگیریم. میتونیم شکل زیر رو برای این مدل رسم کنیم :



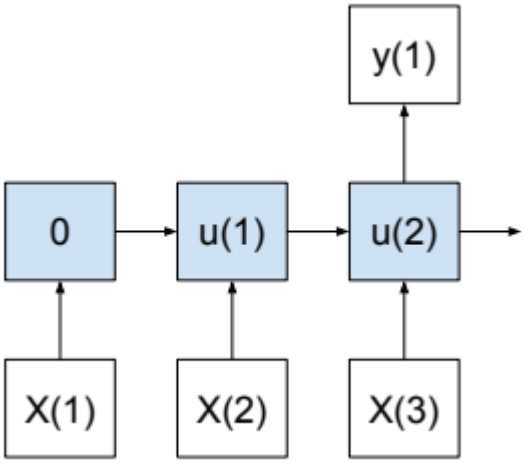
هایدن استیت های داخلی هم مقادیرشون زمانی که یک خروجی در دنباله خروجی ها تولید میشه ... انباشه میشن و آپدیت میشن. و این مدل برای تسک هایی خوب هست که برای یک ورودی، ما یک دنباله خروجی رو بخوایم تولید کنیم. مثلا بخوایم برای یک عکس، دنباله ای از کلمات رو تولید کنیم که همون تسک image captioning میشه. و یا بخوایم دنباله ای از مشاهدات رو بعد از اتفاق افتادن یک رویداد، پیش بینی کنیم.

Many to one model :

برای دنباله ای از ورودی ها، یک خروجی تولید میکنه. یعنی مثلا :

```
y(1) = f(X(1), X(2))
```

و در واقع اگر زمان رو بر دریافت ورودی یا تولید خروجی، گسسته کنیم، میتونیم شکل زیر رو رسم کنیم :

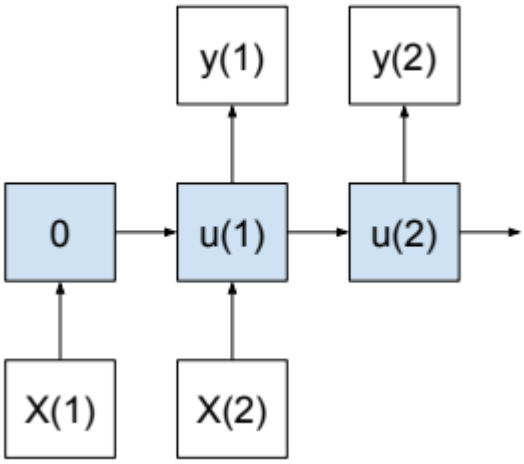


و در این جا هایدن استیت های داخلی بعد از دریافت هر ورودی و قبل از تولید خروجی، انباشه و آپدیت میشن. این مدل برای مسائل sequence prediction ای مناسب است که مثلا برای تولید یک خروجی نیاز به اطلاعات دنباله ای از ورودی ها داشته باشیم. . مثلا در مسائل سری های زمانی برای تولید یک مقدار حقیقی، به چند مشاهده ی قبلی نیاز داشته باشیم. یا مثلا در تسک تحلیل احساسات، دنباله ای از کلمات یک جمله رو بدیم و در نهایت یک خروجی که همان لیبل احساس هست رو دریافت کنیم.

Many to many model :

این مدل بعد از دریافت چندین ورودی، شروع به تولید چندین خروجی میکند که در واقع میتونیم به صورت ضابطه زیر، این مدل رو بنویسیم.

```
y(1),y(2) = f(X(1), X(2))
```



و همانند مدل many to one ، هایدن استیت های ما انباشه و آپدیت میشن تا زمانی که به تولید اولین خروجی برسیم. و در واقع نیازی هم نیست که تعداد خروجی های تولید شده با ورودی های تولید شده برابر باشه. این مسئله که یکی از تیپ های پرکاربرد دنیای یادگیری عمیق هست رو sequence to sequence یا seq2seq میگیم. و مثلا اگر بخوایم یک داکیومنتی رو که حاوی دنباله ای از کلمات است رو به یک داکیومنت کوچکتری تبدیل کنیم که در واقع همان تسک summarization هست. یا مثلا بخوایم دنباله ای از ورودی صوتی رو به دنباله از کلمات متنی تبدیل کنیم. یا دنبال ای از کلمات انگلیسی رو به دنباله ای از کلمات فارسی تبدیل کنیم. در واقع شاید بشه گفت که این مدل اومده توانایی های مدل های many to one و one to many رو با همدیگه ترکیب کرده.

قسمت ب : مدل های خواسته شده در یک نوت بوک نوشته شده. و سعی کردیم که از variant های lstm برای پیاده سازی این مسئله کمک بگیریم و این که هم داده ی یک فیچره تولید کنیم و هم داده چند فیچره و برای هر کدوم مدل هایی رو پیاده سازی کنیم.

سوال دوم : در این مسئله از اپتیمایزرهای sgd و adam و توابع فعالسازیlinear و sigmoid و relu استفاده شده است. به صورت کامل در نوت میتونید خروجی ها رو مشاهده کنید و نیازی

به ران کردن هم نیست. و مقادیری هم که مثلا برای متغیرهای اپتیمایزرها و سایر هایپرپارامترها دیده میشه، بعد از چندین ران به اون مقادیر رسیدیم و تا حد زیادی hyper parameter optimization هم انجام دادیم.

سوال سوم :

توی سوال اول گفتیم که ما یک مدل many to many داریم که دنباله ای از ورودی ها رو میگیره و دنباله ای از خروجی ها رو تولید میکنه. بعد ها از روی همین معماری، یک معماری دیگری به نام اینکدر-دیکدر ساختند. که در واقع قسمت اینکدر قضیه میومد یک تعدادی ورودی مثلا کلمات یک جمله رو میگرفت و اونا رو در یک برداری با سایز ثابت اینکد میکرد و مثلا تمام اطلاعات جمله در این بردار ثابت گذاشته میشد. بعدش دیکدر میومد از این بردار ثابت که در واقع بهش encoding vector میگن، استفاده میکرد و شروع به تولید خروجی ها یا مثلا کلمات میکرد. اما در این معماری یک مشکلی که وجود داشت این بود که ما نمی تونیم کل اطلاعات دنباله ورودی یا مثلا جمله ورودی رو در این بردار جاگذاری کنیم و در واقع دیکدر نمیتونه از تمام اطلاعاتی که در جمله ورودی داره استفاده کنه و جمله خروجی رو تولید کنه. و حتی این مشکل برای ورودی هایی که طول بیشتری دارن مثلا جملات طولانی یا پاراگراف ها، خیلی بیشتر میشد. در واقع در سال 2014 یک مقاله ای برای تسک ماشین های ترجمه منتشر شد(معمولا همیشه ایده های ناب برای تسک ماشین ترجمه میاد بعدش بسطش میدن به تسک های دیگه) که در واقع در اون به کمک برای حل این مشکل پرداخته بود و مفهومی به نام اتنشن رو مطرح کرده بود که در ادامه به اون میپردازیم. و در واقع هدف کلی اتنشن اینه که مثلا در پردازش زبان، به قسمت هایی از جمله یا کلماتی، اهمیت بیشتری بده و اطلاعات اونا رو بیشتر در encoding vector ذخیره کنه. در این مکانسیم اول ما باید 3 چیز رو محاسبه کنیم که در ادامه به اون میپردازیم

Alignment scores : این مدل، هایدن استیت اینکد شده رو میگیره که در اینجا h اونو مینامیم و خروجی قبلی دیکدر رو هم میگیره که بهش s میگیم و با کمک این ها یک امتیازی به نام e تولید میکنیم. که نشون میده که ایتم های دنباله ورودی با خروجی فعلی چقدر خوب به

اصطلاح تراز بندی شدن. و alignment model با یک تابعی به نام a() بیان میشه که مثلا میتونه یک شبکه فیدفوروارد باشه.

$$e_{t,i} = a(s_{t-1}, h_i)$$

weights : وزن ها که اون ها رو آلفا مینامیم با اعمال تابع سافت مکس بر روی امتیازی که در مرحله قبل بدست آوردیم، بدست میان

$$\alpha_{t,i} = \mathrm{softmax}(e_{t,i})$$

Context vector : یک بردار بافتاری منحصر به فرد به نام c که در هر تایم استپ به دیکدر داده میشه. و این بردار از طریق جمع وزن دار تمام هیدن استیت هایی که از اینکدر بدست اومده، بدست میاد

$$c_t = \sum_{i=1}^{T} \alpha_{t,i} h_i$$

بعد ها از روی همین مقاله یک مدل کلی تری رو ارائه کردن که در واقع نیازی نیست که مثلا داده های ورودی ما به صورت یک دنباله از ورودی ها باشه و مثلا میتونه ورودی اینکدر ما یک عکس باشه.

در این مدل کلی تر ما از 3 مفهوم استفاده میکنیم که اون ها رو queries : q و keys : k و values : v مینامیم

اگه بخوایم این 3 جز رو با روش اولیه مقایسه کنیم در واقع بردار query شبیه به خروجی قبلی دیکدر یا s(t-1) هست و values شبیه به ورودی اینکد شده یا hi هست و در واقع توی مدل قبلی key ها با value ها یکسان هستند.

بعدش محاسبات زیر رو انجام میدیم.

هر بردار query در بردار های key ضرب داخلی میشه و امتیازی رو محاسبه میکنه

$$e_{q,k_i} = q \cdot k_i$$

و این امتیازها به تابع سافت مکس داده میشن تا وزن هایی رو تولید کنن

$$\alpha_{q,k_i} = \mathrm{softmax}(e_{q,k_i})$$

و سپس اتنشن کلی از طریق جمع وزن دار بردارهای value بدست میاد و در واقع گفتیم که هر بردار value هم با یک بردار key جفت بود.

$$\mathrm{attention}(q, K, V) = \sum_i \alpha_{q,k_i} v_{k_i}$$

در واقع مثلا اگه بخوایم از این روش در تسک ترجمه ماشینی استفاده کنیم اینطوریه که مثلا هر کلمه در جمله ورودی باید بردارهای query and values and keys خودشو داشته باشه. و این بردارها از طریق بازنمایی که برای اون کلمه از طریق اینکدر بدست اومده ضربدر 3 ماتریس وزنی بدست میان. یعنی برای مثلا بدست اوردن بردار key ... ما بازنمایی اون کلمه رو در یک بردار وزنی مثل key prim ضرب میکنیم. و این 3 ماتریس وزن در فرایند training ، بهینه میشن. در واقع با این روش، مشخص میشه که مثلا هر کلمه چقدر به کلممات دیگه ربط داره و مثلا برای تولید امبدینگ جدید اون کلمه خاص، با چه نسبتی باید از کلمات دیگر موجود در جمله استفاده کرد. و مثلا تسک دیگه ای که این اتنشن در اون کاربرد داره تسک corefrence resolution هست که در واقع ما در جمله یک ضمیری داریم که باید بفهمیم مرجع ضمیر کی هست و قبلا در امبدینگ این ضمیر، با استفاده از مکانیسم اتنشن تا حدی مشخص شده که به کی اشاره میکنه.

بعد از این مقاله، ساختار های جدیدی در رابطه با اتنشن معرفی شد. مثل global attention یا local attention یا hard attention و یا soft attention.

که مثلا یکی از اون ساختار ها رو در رابطه با تسک image captioning با همدیگه بررسی میکنیم.

فرض کنیم میخوایم برای یک تصویر، یک کپشن تولید کنیم، برای این کار
باید به قسمتهای مختلف تصویر توجه کنیم و متناسب با اون قسمتها یک کپشن تولید کنیم.
زمانی که از attentionاستفاده میکردیم، تصویر رو به عنوان ورودی به شبکه lstmمیدادیم.
اما در soft attention،ابتدا چندین فیچر رو در تصویر در نظر میگیریم و هر بار به اون بخش از تصویر، وزن بیشتری میدیم. به این ترتیب که قسمتهایی که باید بیشتر بهشون توجه کرد، روشنتر و و باقی قسمتها تیره تر میشن. چون مقدار اونها به صفر نزدیک میشه.

حال بررسی میکنیم چطوری فیچرهای وزندار رو برای ورودی lstmبسازیم.
فرض کنیم که ۴x ۴ , x2 , x ۴, ۳هرکدوم یک زیربخش از تصویر ماست. برای محاسبه اینکه به کدوم بخش باید توجه بیشتری کرد، داریم

$$s_i = \tanh(W_c C + W_x X_i) = \tanh(W_c h_{t-1} + W_x x_i)$$

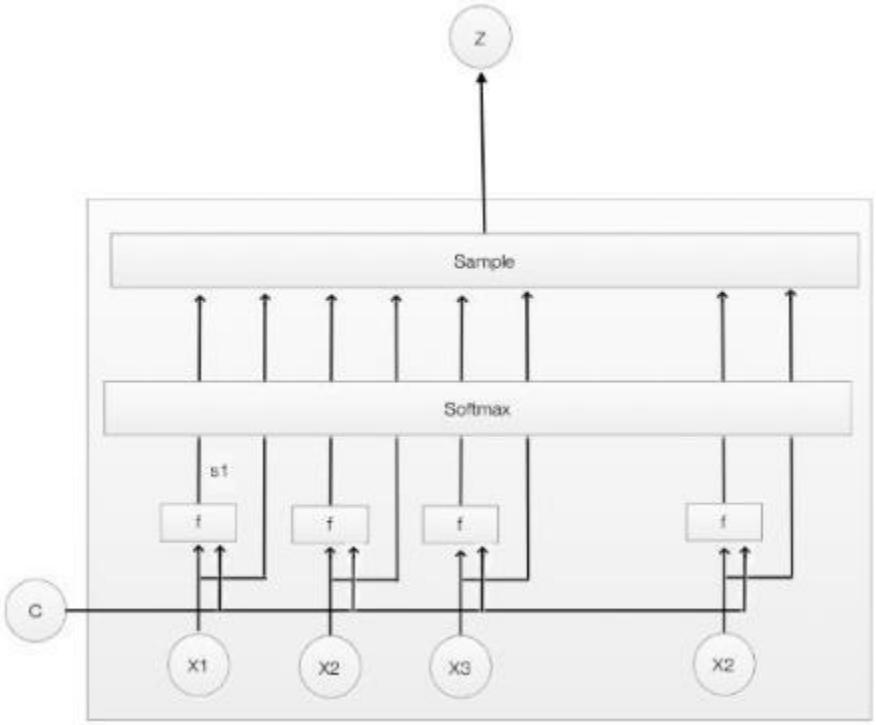
که siنشان دهندهی این هستش که به xiچقدر باید توجه کرد.
Siرو برای نرمال کردن به softmaxمیدیم تا وزن آلفا رو محاسبه کنه.

$$\alpha_i = softmax(s_1, s_2, \ldots, s_i, \ldots)$$

با softmax مجموع آلفا ها برابر یک میشه و ما میتونیم میانگین وزنها رو برای ۴Xتا ۳X محاسبه کنیم.

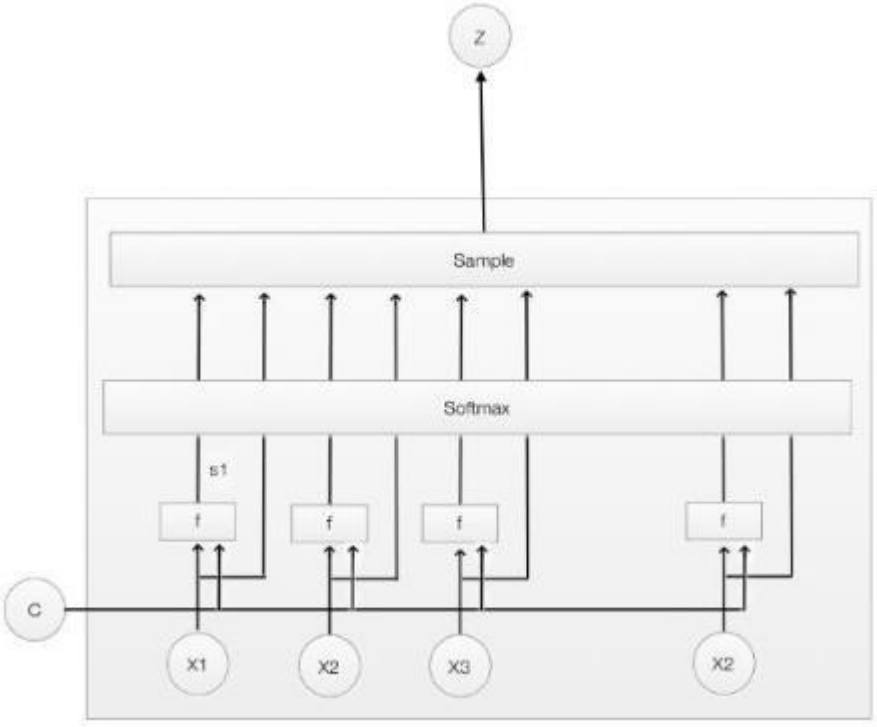
$$Z = \sum_i \alpha_i x_i$$

در نهایت، ما از zبرای جایگزین کردن xبرای ورودی lstmاستفاده میکنیم.



در ،softmax attentionما وزن آلفا iرو برای هر xiحساب میکنیم، و ازش استفاده میکنیم برای محاسبه میانگین وزنی برای xiبه عنوان ورودی lstm .جمع آلفا iها برابره یک میشه.که به

عنوان احتمال اینکه xiناحیه ای هست که ما باید بهش توجه کنیم، تفسیرش میکنیم. اما به جای میانگین وزنی، hard attentionاز آلفا iبه عنوان نرخ نمونه برداری، برای برداشتن یک xiبرای ورودی lstmاستفاده میکنه.



در hard attentionاز یک نمونه گیری تصادفی استفاده میشه و در نهایت میانگین خروجی مربوط به نمونه ها رو میگیریم.
اینکه چه تعداد نمونه گرفته شده باشه و نمونه گیری چقدر خوب انجام شده باشه، دقت تغییر میکنه. اما در soft attentionبر روی کلیه نمونه ها انجام میشه.

با تشکر