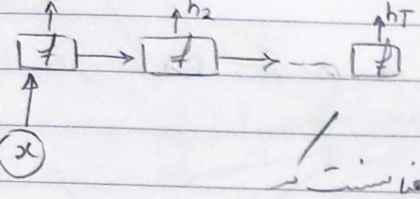
یادگیری ماشین برای بیوانفورماتیک - تمرین پنجم

مهدی کافی 99210753

① الف) one-to-many: این شبکه‌ها تنها یک ورودی می‌گیرند و سپس یک دنباله، و دنباله‌ای از خروجی را تولید می‌کنند:



هر کدام از f ها می‌توانند یک یا چند لایه بازگشتی مخفی باشند. کلمه «یک» در ورودی به این معنا نیست که

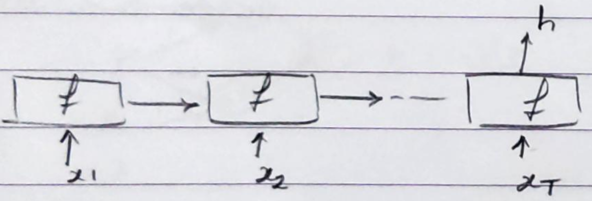
فقط یک ویژگی ورودی داریم. بلکه به این معنا است که ورودی از یک time step است یا اصلاً مستقل از زمان است.

کاربرد: عموماً به عنوان تولیدکننده دنباله به کار می‌روند. مانند تولید یک موسیقی با گرفتن نوت آغازین یا ژانر موسیقی یا تولید

نمایشنامه با ورودی کلمه شروع کننده نمایشنامه.

many-to-one: این شبکه‌ها دنباله‌ای از ورودی با طول دلخواه می‌گیرند و پس از پردازش تمام دنباله ورودی، یک خروجی تولید می‌کنند:

در اینجا نیز، f می‌تواند شامل یک یا چند لایه بازگشتی مخفی باشد.

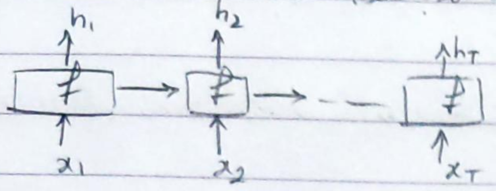


کاربرد: معمولاً برای دسته‌بندی داده‌های sequential

استفاده می‌شوند. مانند Sentiment Analysis که کل نظر یک مشتری را خوانده و دسته‌بندی می‌کند که نظر مثبت، منفی یا خنثی

است. یا می‌توان برای مشخص کردن موضوع اخبار از آن استفاده کند.

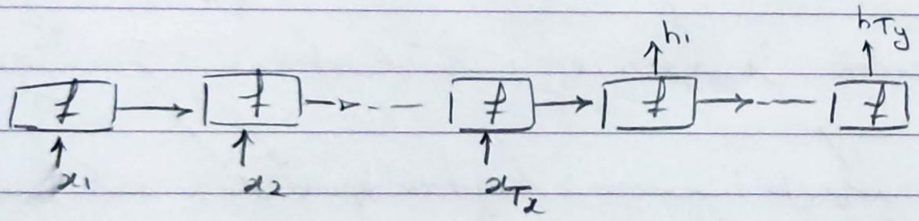
many-to-many (synced): در این مدل هر یک از ورودی‌ها می‌توانند خروجی داشته باشند.



همانطور که مشخص است هر خروجی براساس ورودی خودش و تمامی خروجی‌های قبلی محاسبه می‌شود.

کاربرد: یکی از کاربردهای گسترده این مدل‌ها، پیش‌بینی‌های مرتبط در طول زمان است مانند پیش‌بینی میزان فروش هر روز یک فروشگاه یا ارزش‌های سهام و کاربردهای دیگر آن مانند تشخیص گفتار real-time است.

many-to-many (unsynced): بعضی مواقع می‌خواهیم که دنباله خروجی را پس از بررسی تمام دنباله ورودی، تولید کنیم. در این موقعیت‌ها از مدل زیر استفاده می‌کنیم:

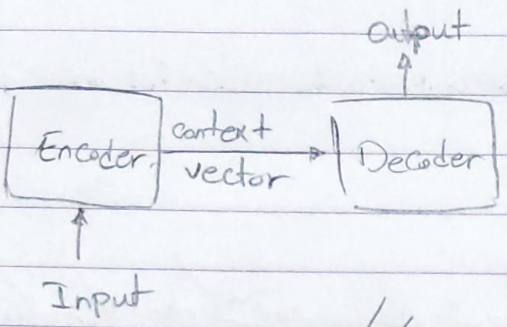


طول دنباله ورودی و خروجی می‌تواند متفاوت باشد.

کاربرد: در ترجمه ماشینی استفاده می‌شود. به طور مثال در ترجمه فارسی به انگلیسی، تمام رشته فارسی خوانده شده و سپس رشته انگلیسی تولید می‌شود.

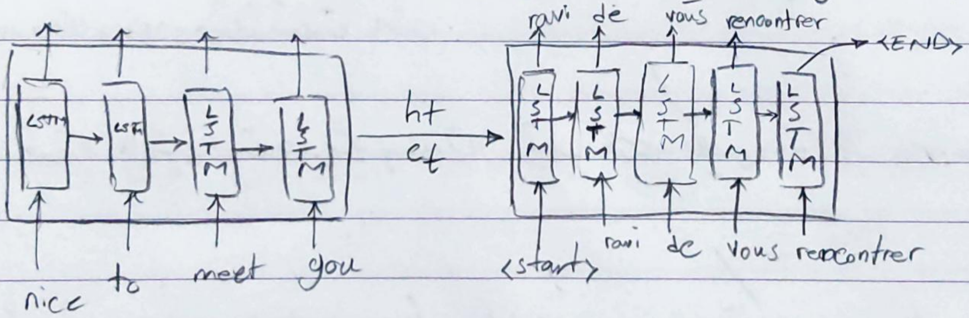
ب) مسئله Seq2Seq، نوعی از مسائل Sequence Modelling هستند. هر دوی input و output آن‌ها sequence هستند. بنابراین در دسته many-to-many قرار می‌گیرند.

معماری encoder-decoder بصورت کلی یک معماری متشکل از دو بخش encoder و decoder است که بطور مثال اگر بخواهیم نمونه‌ای از ترجمه ماشینی را مثال بزنیم بصورت زیر خواهد بود:



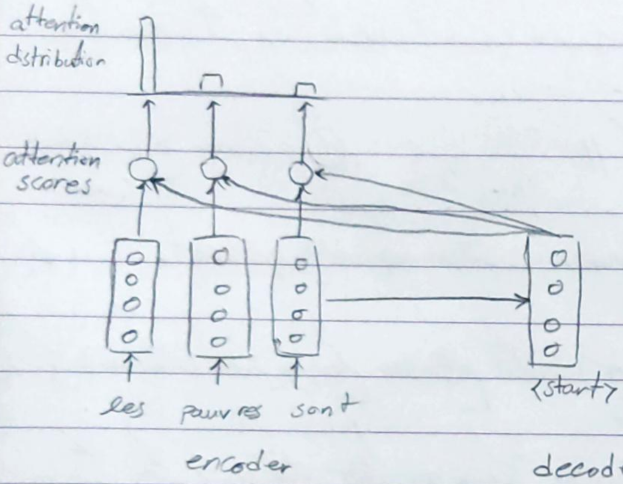
در این مسأله به طور خاص، بخش encoder، تمامی لغات رشته ورودی را پردازش می‌کند و سعی می‌کند که تمامی اطلاعات را در context vector نگهداری کند. سپس این بردار که باید مفهوم کلی جمله را داشته باشد به بخش decoder داده می‌شود که جمله ترجمه شده را لغت به لغت تولید کند.

اگر بخواهیم نگاه دقیق‌تری به داخل هر بخش داشته باشیم که درون هر یک از encoder و decoder یک ساختار LSTM بطور معمول قرار دارد زیرا مشکل long-term dependency را خیلی خوب، حل می‌کند.



نکات قابل توجه این هستند که خروجی‌های encoder، دور ریخته می‌شوند و صرفاً hidden state و cell state پایانی همان context vector هستند که به بخش decoder داده می‌شوند. در بخش decoder نیز با داشتن توکن <start> و context vector شروع به تولید جمله خروجی می‌کند و هر لغت را به ورودی LSTM بعدی می‌دهد تا در نهایت به توکن <END> برسیم.

ج: در پایان کار بخش encoder، تمام جمله ورودی به یک جمله تبدیل شده است که برای decoder سخت است که از روی این جمله، ترتیب لغات در جمله خروجی را متوجه شود. برای رفع این مشکل از مکانیزم attention استفاده می‌کنیم. attention به decoder اجازه می‌دهد که بخش‌های خاصی از جمله ورودی توجه بیشتری برای تولید هر لغت در خروجی. و این کار باعث می‌شود که جملات ورودی و خروجی با یکدیگر align شوند.



به طور مثال در شکل بالا، بیشترین امتیاز attention برای کلمه "les" است و این باعث می‌شود که جمله خروجی با "the" آغاز شود.

(۲)

الف) برای انجام Back P.T. Time، نیاز است که در ابتدا شبکه RNN را unfold کنیم و تمامی لایه‌های آن را زیر یکدیگر بکشیم. حال هر لایه زمانی یک خروجی دارد و مقادیر وزن و بایاس بین لایه‌ها مشترک است. حال نیاز است، خروجی هر لایه را با خروجی مورد نظر مقایسه کرده، مقدار loss را محاسبه کنیم و مقدار loss کلی برابر با جمع تمامی loss های تمامی لایه‌ها است. حال طبق است مانند B.P. عادی، شروع به گرادیان گرفتن کنیم و با استفاده از chain rule شروع به حرکت از سمت خروجی و لایه‌های بالایی به ورودی و لایه‌های اولیه کنیم و مقادیر وزن‌ها و بایاس‌ها را آپدیت کنیم. تفاوت این روش با Back Propagation عادی در این است که در اینجا هر لایه زمانی loss اش باید محاسبه شود بنابراین مرحله unfolding داریم که در B.P. عادی وجود ندارد.

s.a.m

ب) vanishing gradient و exploding gradient دو مشکلی هستند که باعث می‌شوند یادگیری انجام نشود. به اینصورت که در اولی مقدار گرادیان خیلی سریع صفر می‌شود و دیگر آموزشی نخواهیم داشت و در دومی مقدار گرادیان بسیار بزرگ و NaN می‌شود و حالت ناپایداری ایجاد می‌شود.

در شبکه‌های بازگشتی می‌توان گرادیان را بصورت رو به رو نوشت $\frac{\partial L}{\partial W} = \sum_{i=0}^{T} \frac{\partial L_i}{\partial W} \propto \sum_{i=0}^{T} \left(\prod_{i>k}^{i} \frac{\partial h_i}{\partial h_{i-1}} \right) \frac{\partial h_k}{\partial W}$. در اینجا ممکن است دو حالت رخ بدهد: ① $\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 < 1$ که باعث vanishing gradient می‌شود و ② $\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\| > 1$ که باعث exploding gradient می‌شود. در حالت اول مقدار گرادیان بصورت نمایی به سمت صفر حرکت می‌کند و باعث می‌شود که long period dependency ها را نتوانیم آموزش ببینیم و در حالت دوم مقدار گرادیان به سمت بی‌نهایت و NaN می‌رود و ناپایداری می‌شود. دلیل دیگری که این اتفاق در شبکه‌های بازگشتی شدیدتر است این است که دنباله ورودی می‌تواند بسیار بزرگ باشد و این به معنای تعداد زیاد مشتق‌گیری‌ها است که باعث ایجاد مشکلات بالا می‌شود که با روش‌هایی مثل truncated BPTT و gradient clipping می‌توان آن‌ها را رفع کرد.

پ) می‌دانیم که معماری LSTM دارای سه بخش به نام‌های forget gate، input gate و update gate است که باعث می‌شود cell state را علاوه بر hidden state به لایه‌های بعدی ببرد و باعث می‌شود long term dependency ها بهتر توجه شود. حال می‌دانیم که دو عامل هستند که اندازه گرادیان را تحت تاثیر قرار می‌دهند یکی وزن‌ها و دیگری توابع فعال‌سازی (یا بطور دقیق‌تر مقدار مشتق آن‌ها). حال اگر هرکدام از این‌ها از ۱ کمتر شوند vanishing و اگر بزرگتر از ۱ شوند exploding اتفاق می‌افتد. حال می‌دانیم در LSTM، تابع فعال‌سازی تابع همانی با مشتق ۱ است. بنابراین گرادیان‌ها که back propagate شده

به صفر و بی‌نهایت نمی‌شود بلکه ثابت می‌ماند. وزن مؤثر بازگشتی همان تابع فعال سازی forget gate است. بنابراین

اگر این gate روشن باشد یا activation نزدیک ۱ باشد، گرادیان دچار vanishing نمی‌شود و به این دلیل که مقدار

فعال سازی forget gate از ۱ بیشتر نمی‌شود دچار exploding نمی‌شود. بنابراین معماری LSTM در حل مسئله

vanishing و exploding گرادیان به ما کمک می‌کند.

۳) مشکل exposure bias این است که همانطور که در صورت سوال آمده است در مرحله یادگیری teacher forcing

داریم و داده های درست را به عنوان ورودی به مدل یاد گیری زمانی میدهیم و این اتفاق باعث نابرابری بین train-test می شود که

exposure bias نامیده می شود. این اتفاق می تواند منجر به overfitting شود و بنابراین باید مدل را generalize

کنیم. روش هایی از قبیل scheduled sampling برای مقابله با exposure bias توسعه داده شده اند. همچنین داده های

آموزش و تست توزیع نزدیک به هم نداشته باشند، عملکرد شبکه بسیار ضعیف خواهد شد.

به نظر می رسد که اگر نخواهیم داده های صحیح را به مراحل بعدی بدهیم نتوانیم اصلاً مدل را آموزش دهیم و یا این آموزش

زمان بسیار زیادی می برد. زیرا که داده های صحیح بخشی از ورودی هر مرحله هستند.

۴

الف) می‌دانیم که discriminator به دنبال بیشینه‌کردن $V(G,D)$ است به ازای هر generator داده‌شده داشته باشیم:

$$V(G,D) = E_{x\sim P_{data}}[\log D(x)] + E_{x\sim P_g}[\log(1-D(x))]$$

$$= \int_x P_{data}(x)\log(D(x))\,dx + \int_z P_z(z)\log(1-D(g(z)))\,dz$$

$$= \int_x P_{data}(x)\log(D(x)) + P_g(x)\log(1-D(x))\,dx$$

برای هر $(a,b)\in R^2\setminus\{0,0\}$، مقدار بیشینه تابع $y = a\log y + b\log(1-y)$ در بازه $[0,1]$، برابر با مقدار $\frac{a}{a+b}$ خواهد بود. بنابراین مقدار بهینه برای discriminator برابر با $D^*_G(x) = \frac{P_{data}(x)}{P_{data}(x)+P_g(x)}$ خواهد بود.

ب)
$$C(G) = \max_D V(G,D) = E_{x\sim P_{data}}[\log D_G^*(x)] + E_{z\sim P_z}[\log(1-D_G^*(G(z)))]$$

$$= E_{x\sim P_{data}}[\log D_G^*(x)] + E_{x\sim P_g}[\log(1-D_G^*(x))]$$

$$= E_{x\sim P_{data}}\left[\log \frac{P_{data}(x)}{P_{data}(x)+P_g(x)}\right] + E_{x\sim P_g}\left[\log \frac{P_g(x)}{P_{data}(x)+P_g(x)}\right]$$

می‌خواهیم اثبات کنیم که کمینه $C(G)$ تضمینی است که اگر و تنها اگر $P_g = P_{data}$ ، و کمینه آن برابر با $-\log 4$ خواهد بود.

$$P_g = P_{data} \rightarrow D_G^*(x) = \frac{1}{2} \Rightarrow C(G) = \log\frac{1}{2} + \log\frac{1}{2} = -\log 4$$

$$E_{x\sim P_{data}}[-\log 2] + E_{x\sim P_g}[-\log 2] = -\log 4$$

$$C(G) = -\log(4) + KL\left(P_{data} \Big\| \frac{P_{data}+P_g}{2}\right) + KL\left(P_g \Big\| \frac{P_{data}+P_g}{2}\right) \quad (KL = \text{kullback-leibler})$$

$$\Rightarrow C(G) = -\log 4 + 2 \cdot JSD(P_{data} \| P_g)$$

* از آنجایی که فاصله Jensen-Shannon بین دو توزیع همواره نامنفی است و هنگامی که یکسان باشند صفر است. نشان دادیم که کمینه global برای $C(g)$ برابر با $-\log 4$ است. و در هنگامی اتفاق میافتد که فاصله JS بین توزیع دادگان آموزش و توزیع شبکه مولد کمینه و برابر صفر شود.

ج) همانطور که در بالا گفته شد، کمینه مقدار فاصله JS هنگامی است که $P_{data} \approx P_g$ $(P_{data}=P_g)$ بشود. در اینصورت به نقطه بهینه برای تابع هدف میرسیم.

⑤ معمولاً می‌خواهیم که GAN ها مجموعه گسترده‌ای از خروجی را تولید کنند به طور مثال می‌خواهیم که برای هر ورودی تصادفی یک چهره جدید تولید کند. اما اگر generator یاد بگیرد که خروجی پذیرفتنی برای discriminator تولید کند، ممکن است که همواره همان خروجی را تولید کند. در واقع generator سعی می‌کند پذیرفتنی‌ترین خروجی برای discriminator را پیدا کند. اگر generator شروع کند به تولید کردن خروجی یکسان (یا مجموعه کوچکی از خروجی‌ها)، بهترین کار برای discriminator این است که همواره آن خروجی را رد کند. ولی اگر discriminator در نسل بعدی در کمینه محلی گیر کند و بهترین راه را پیدا نکند، برای generator در iteration بعدی بسیار ساده خواهد بود که پذیرفتنی‌ترین خروجی را برای discriminator فعلی پیدا کند. در هر چرخش، generator تولیدکننده، discriminator خاص آن چرخش را over-optimize می‌کند. و discriminator هیچ‌گاه یاد نمی‌گیرد از تله خارج شود. بنابراین generator همواره بین تعداد کمی خروجی می‌چرخد. که به این مشکل هلوتیکا گفته می‌شود.

⑥

۱) وقتی که تابع هدف شبکه مولد به صورت $E_{x\sim p_z}[\log(1-D(x))]$ باشد در شروع یادگیری شبکه، باشیم generator تقریباً خروجی‌های تصادفی و خیلی ضعیفی می‌دهد که به راحتی آن‌ها را discriminator شناسایی می‌دهد. حال اتفاقی که می‌افتد گرادیان برای generator بسیار کم و تقریباً صفر است که باعث می‌شود یادگیری بسیار کند شود که به آن saturation می‌گویند ولی با استفاده از $E_{x\sim p_g}[\log D(x)]$، هنگامی که discriminator در ابتدا به راحتی شناسایی کند مقدار گرادیان شبکه مولد زیاد است و سریعاً به سمت بهتر شدن حرکت می‌کند.

الف) در چنین شرایطی اگر discriminator به سمت بهینه برود، مقدار این گرادیان صفر خواهد بود:

$$\lim_{\|D-D^*\|\to 0} \nabla_\theta E_{z\sim P_z}\left[\log(1-D(g_\theta(z)))\right]=0$$

$$\Rightarrow \nabla_{\theta_g} \log(1-D(G(z^i)))=0$$

ب) در اینصورت گرادیانی که به شبکه مولد می‌رسد برابر با صفر است. مشکل vanishing gradent ایجاد می‌شود و دیگر آموزشی نخواهیم داشت.

ج) در اینصورت: $E_{z\sim P(z)}\left[-\nabla_\theta \log D^*(g_\theta(z))\Big|_{\theta=\theta_0}\right]=\nabla_\theta\left[KL(P_{g_\theta}\|P_r)-2JSD(P_{g_\theta}\|P_r)\right]\Big|_{\theta=\theta_0}$

و بنابراین گرادیان جدید از معکوس KL-divergence و JS-divergense تشکیل شده است. که باعث می‌شود شبکه مولد تصاویر واقعی‌تری تولید کند اما ممکن است چسبندگی مد پیش بیاید.

(۷) اگر P_r و P_g در توزیع اصلی و شبکه مولد باشند که در دو manifold، M و P قرار بگیرند به طوری که full dimension نباشند و همینطور کاملاً align نباشند، خواهیم داشت:

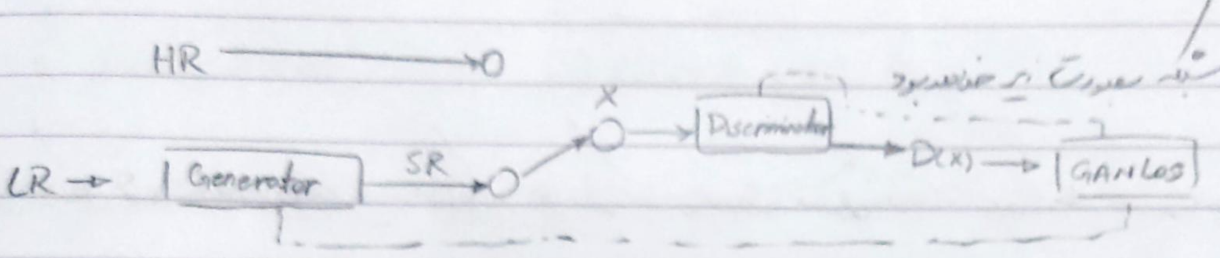
$JSD(P_r\|P_g)=\log 2$
$KL(P_r\|P_g)=+\infty$
$KL(P_g\|P_r)=+\infty$

در چنین حالتی می‌بینیم که مقادیر divergence حتی اگر manifold ها به طور تقریبی به یکدیگر نزدیک باشند هم بیشینه می‌شوند.

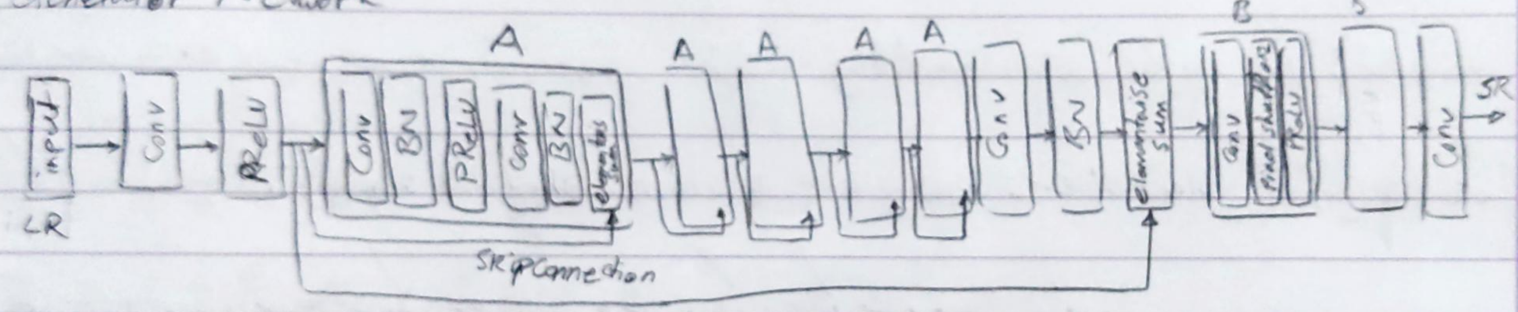
ممکن است خروجی شبکه مولد خیلی خوب باشد ولی باز هم KL divergence بی‌نهایت می‌شود. و در چنین شرایطی

نمی‌توان از gradient descent استفاده کرد، و برای حل این مشکل می‌توانیم از معیارهای مقایسه‌ای ساختاری استفاده کنیم.

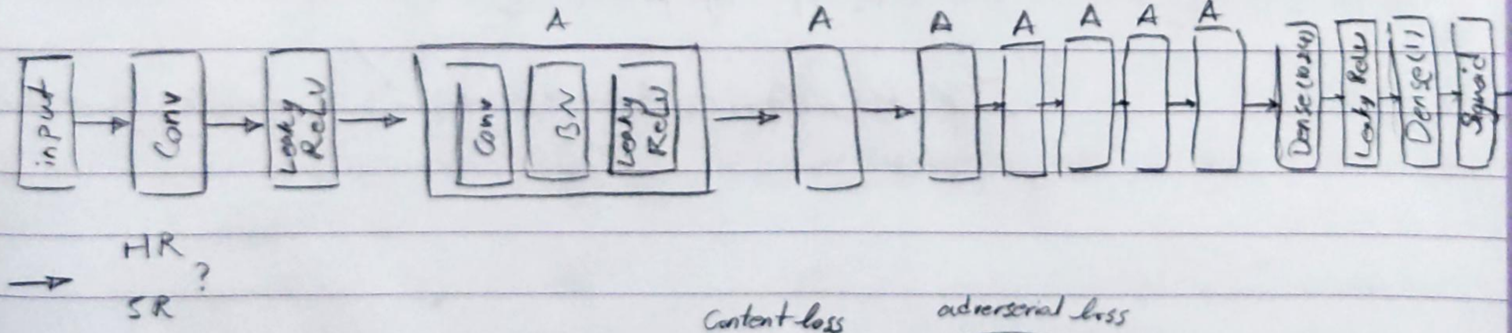
۸) ساختار این شبکه به صورت زیر خواهد بود



Generator Network



Discriminator Network



generator loss function : $l^{SR} = \overbrace{l_X^{SR}}^{\text{Content loss}} + \overbrace{10^{-3}\, l_{Gen}^{SR}}^{\text{adversarial loss}}$

$$\left(l_{Gen}^{SR} = \sum_{n=1}^{N} -\log D_{\theta_D}\left(G_{\theta_G}(I^{LR})\right)\right)$$

$$\left(l_X^{SR} = MSE(HR, SR) \right)$$

discriminator loss function:

$$\min_{\theta_G} \max_{\theta_D} E_{I^{HR} \sim P_{train}(I^{HR})} \left[\log D_{\theta_D}(I^{HR}) \right] + E_{I^{LR} \sim P_G(I^{LR})} \left[\log (1 - D_{\theta_D}(G_{\theta_G}(I^{LR}))) \right]$$