$\boxed{1.a}$

$$\mathbb{E}_D[P]_{ij} = \mathbb{E}_D\left[(D \odot X)_{ij}\right] \xlongequal[\text{است}]{D \text{ مستقل } X} X_{ij}\, \mathbb{E}_D\left[D_{ij}\right] \xlongequal{D_{ij} \sim \mathrm{Bern}(p)} p X_{ij}$$

$$P^T P = \begin{bmatrix} D_{11} x_1^{(1)} & \cdots & D_{N1} x_1^{(N)} \\ \vdots & & \vdots \\ D_{1d} x_d^{(1)} & \cdots & D_{Nd} x_d^{(N)} \end{bmatrix}_{d \times N} \begin{bmatrix} D_{11} x_1^{(1)} & \cdots & D_{1d} x_d^{(1)} \\ \vdots & & \vdots \\ D_{N1} x_1^{(N)} & \cdots & D_{Nd} x_d^{(N)} \end{bmatrix}_{N \times d} = \begin{bmatrix} \sum_{k=1}^{N} D_{k1}^2 x_1^{(k)2} & \sum_{k=1}^{N} D_{k1} D_{k2} x_1^{(k)} x_2^{(k)} & \cdots \\ \vdots & & \\ & & \end{bmatrix}_{d \times d}$$

$$\mathbb{E}_D\left[P^T P\right]_{ij} = \begin{cases} \mathbb{E}_D\left[\sum_{k=1}^{N} D_{ki}^2 x_i^{(k)^2}\right] = \sum_{k=1}^{N} \underbrace{\mathbb{E}_D\left[D_{ki}^2\right]}_{p\ \mathrm{I}} x_i^{(k)^2} = p \sum_{k=1}^{N} x_i^{(k)} x_j^{(k)} = p (X^T X)_{ij} & i = j \\ \mathbb{E}_D\left[\sum_{k=1}^{N} D_{ki} D_{kj} x_i^{(k)} x_j^{(k)}\right] = \sum_{k=1}^{N} \underbrace{\mathbb{E}_D\left[D_{ki} D_{kj}\right]}_{p^2\ \mathrm{II}} x_i^{(k)} x_j^{(k)} = p^2 \sum_{k=1}^{N} x_i^{(k)} x_j^{(k)} = p^2 (X^T X)_{ij} & i \neq j \end{cases}$$

$$D_{ki} \sim \mathrm{Bern}(p) \longrightarrow D_{ki}^2 \sim \mathrm{Bern}(p) \longrightarrow \mathbb{E}\left[D_{ki}^2\right] = p \quad \mathrm{I}$$

$$D_{ki} \perp\!\!\!\perp D_{kj} \longrightarrow \mathbb{E}_D\left[D_{ki} D_{kj}\right] = \mathbb{E}_D\left[D_{ki}\right] \mathbb{E}_D\left[D_{kj}\right] = p^2 \quad \mathrm{II}$$

$\boxed{1.b}$

$$\mathcal{L}(\hat{w}) = \mathbb{E}_D\left[\|y - P\hat{w}\|_2^2\right] = \mathbb{E}_D\Big[\|y\|^2 + \underbrace{\|P\hat{w}\|^2}_{\hat{w}^T P^T P \hat{w}} - 2 y^T P \hat{w}\Big]$$

$$= \|y\|^2 + \hat{w}^T \mathbb{E}_D\left[P^T P\right] \hat{w} - 2 y^T \mathbb{E}_D[P]\, \hat{w}$$

$$\overset{*}{=} \underbrace{\|y\|^2 - 2 y^T p X \hat{w} + p^2 \hat{w}^T X^T X \hat{w}}_{\|y - pX\hat{w}\|^2} + p(1-p) \underbrace{\hat{w}^T \hat{\Gamma}^2 \hat{w}}_{\|\hat{\Gamma}\hat{w}\|^2}$$

$*$

طبق بخش $\boxed{1.a}$ می دانیم که $\mathbb{E}_D[P] = pX$

همچنین برای $\mathbb{E}_D[P^T P]$ نیز می توان فرم بسته زیر را داشت:

$$\mathbb{E}_D\left[P^T P\right] = p^2 X^T X + p \hat{\Gamma}^2 \underbrace{- p^2 \hat{\Gamma}^2}$$

برای اینکه عناصر قطر اصلی $p^2 X^T X$ را حذف کنیم

$$\hat{\Gamma} = \mathrm{diag}(X^T X)^{\frac{1}{2}}$$

$\boxed{1.c}$

تغییر متغیر های زیر را به ترتیب انجام می‌دهیم:

$$\xrightarrow{p\hat{w}=w} \mathcal{L}(w)=\|y-Xw\|^2+\frac{1-p}{p}\|\hat{\Gamma}w\|^2$$

$$\xrightarrow{\sqrt{\frac{1-p}{p}}\hat{\Gamma}=\Gamma} \mathcal{L}(w)=\|y-Xw\|^2+\|\Gamma w\|^2$$

$\boxed{1.d}$

ابتدا سعی می‌کنیم بخش regularization را ایجاد کنیم

$$\Gamma w=\sqrt{\lambda}\,\tilde{w} \longrightarrow w=\sqrt{\lambda}\,\Gamma^{-1}\tilde{w}$$

$$\tilde{w}=\frac{1}{\sqrt{\lambda}}\Gamma w$$

$$\longrightarrow \mathcal{L}(\tilde{w})=\|y-\underbrace{X\sqrt{\lambda}\,\Gamma^{-1}}_{\tilde{X}}\tilde{w}\|^2+\|\sqrt{\lambda}\,\tilde{w}\|^2 \xrightarrow{\tilde{X}=\sqrt{\lambda}\,X\Gamma^{-1}} \mathcal{L}(\tilde{w})=\|y-\tilde{X}\tilde{w}\|^2+\lambda\|\tilde{w}\|^2$$

$\boxed{1.e}$

$$\tilde{X}=\sqrt{\lambda}\,X\Gamma^{-1}=\sqrt{\lambda}\begin{bmatrix} x_1^{(1)} & \cdots & x_d^{(1)} \\ \vdots & & \\ x_1^{(N)} & \cdots & x_d^{(N)} \end{bmatrix}\begin{bmatrix} \Gamma_{1,1}^{-1} & & 0 \\ & \ddots & \\ 0 & & \Gamma_{d,d}^{-1} \end{bmatrix}=\sqrt{\lambda}\begin{bmatrix} \Gamma_{1,1}^{-1}x_1^{(1)} & \cdots & \Gamma_{d,d}^{-1}x_d^{(1)} \\ \vdots & & \vdots \\ \Gamma_{1,1}^{-1}x_1^{(N)} & \cdots & \Gamma_{d,d}^{-1}x_d^{(N)} \end{bmatrix}$$

$\Gamma_{j,j}$ متناسب با نورم ستون j ام X است. بنابراین مشاهده می‌شود که ماتریس به دست آمده، هر ستونش نرمالایز شده است یعنی نورم ستون های $\tilde{X}$ یکسان شده است. همچنین توسط λ می‌توان مقدار آن‌را کنترل کرد.

این مسئله در حقیقت همان batch normalization است زیرا آنجا نیز فیچرها به صورت جدا در هر batch نرمالایز می‌شوند.

می‌توان گفت $\sqrt{\lambda}$ نیز به نوعی شبیه γ در batch norm عمل می‌کند

1.f

$$\frac{\partial J}{\partial w_i} = \left(y_d - \sum_{k=1}^{n} (w_k + \delta_k) x_k\right)(-x_i)$$

$$\rightarrow \mathbb{E}_\delta \left[\frac{\partial J}{\partial w_i}\right] = \mathbb{E}_\delta \left[-y_d x_i + x_i \sum_{k=1}^{n} w_k x_k + x_i \sum_{k=1}^{n} \delta_k x_k\right]$$

$$= -y_d x_i + x_i \sum_{k=1}^{n} w_k x_k + x_i \sum_{k=1}^{n} \underbrace{\mathbb{E}_{\delta_k}[\delta_k]}_{0} x_k = x_i \sum_{k=1}^{n} w_k x_k - y_d x_i$$

* مشاهده می‌شود که چون δ_k از توزیع نرمال که متقارن است پیروی می‌کند $\mathbb{E}\left[\frac{\partial J}{\partial w_i}\right]$ با $\mathbb{E}\left[\frac{\partial J_n}{\partial w_i}\right]$ که J_n تابع هزینه بدون dropout است، یکی می‌شود

1.g

تابع هزینه بدون dropout را به صورت زیر تعریف می‌کنیم:

$$J_n = \frac{1}{2}\left(y_d - \sum_{k=1}^{n} w_k x_k\right)^2$$

حال امید ریاضی تابع با dropout را حساب می‌کنیم:

$$\mathbb{E}[J] = \mathbb{E}\left[\frac{1}{2}\Big(\underbrace{y_d - \sum_{k=1}^{n} w_k x_k}_{H} - \sum_{k=1}^{n} \delta_k x_k\Big)^2\right]$$

$$= \frac{1}{2}\mathbb{E}\left[H^2 + \Big(\underbrace{\sum_{k=1}^{n} x_k \delta_k}_{\text{این عبارت را Z می‌نامیم}}\Big)^2 - 2H\sum_{k=1}^{n} \delta_k x_k\right] = \underbrace{\frac{1}{2}\mathbb{E}[H^2]}_{J_n} + \frac{1}{2}\mathbb{E}[Z^2] - 2H\sum_{k=1}^{n} \underbrace{\mathbb{E}[\delta_k]}_{0} x_k$$

می‌دانیم که $\mathbb{E}[Z]$ برابر 0 است بنابراین $\mathbb{E}[Z^2]$ در حقیقت همان واریانس Z است و داریم:

$$\mathrm{var}(Z) = \mathrm{var}\left(\sum_{k=1}^{n} x_k \delta_k\right) \xlongequal[\text{اند}]{\delta_k \text{ ها مستقل}} \sum_{k=1}^{n} \mathrm{var}(x_k \delta_k) = \sum_{k=1}^{n} x_k^2 \, \mathrm{var}(\delta_k) = \sum_{k=1}^{n} x_k^2 \alpha w_k^2$$

$$\mathbb{E}[J] = J_n + \frac{\alpha}{2}\sum_{k=1}^{n} x_k^2 w_k^2$$

همان طور که مشاهده می‌شود به فرمی شبیه L2 regularization رسیدیم.

البته در حالت کلی می‌توان نشان داد اگر از تابع هزینه least squares استفاده شود، صرف نظر از توزیعی که در dropout استفاده شود، به تعبیری از L2 regularization می‌رسیم!

1.h

additive gaussian dropout

- یک نویز گاوسی به وزن هر نورون اضافه می‌کند و به این صورت تأثیر نورون‌ها را کم یا زیاد می‌کند. دقت شود که در dropout عادی یک متغیر تصادفی برنولی در وزن‌ها ضرب می‌شود و dropout گاوسی یک جور تعمیم است که از توزیع پیوسته گاوسی استفاده می‌کند.
- به واسطه اضافه کردن نویز به activation ها، قدرت generalization بیشتری به مدل می‌دهد و مدل را robust تر می‌کند

spatial dropout

- در شبکه‌های CNN کاربرد دارد. به جای اعمال روی نورون‌ها، روی تمام feature map اعمال می‌شود. یعنی در حالت عادی که از توزیع برنولی استفاده می‌شود با احتمال 1-p ممکن است هر کدام از feature map ها صفر شود
- مدل را مجبور می‌کند که از تمام feature map ها استفاده کند و به یک پترن خاص متکی نباشد ← مدل robust تر

2

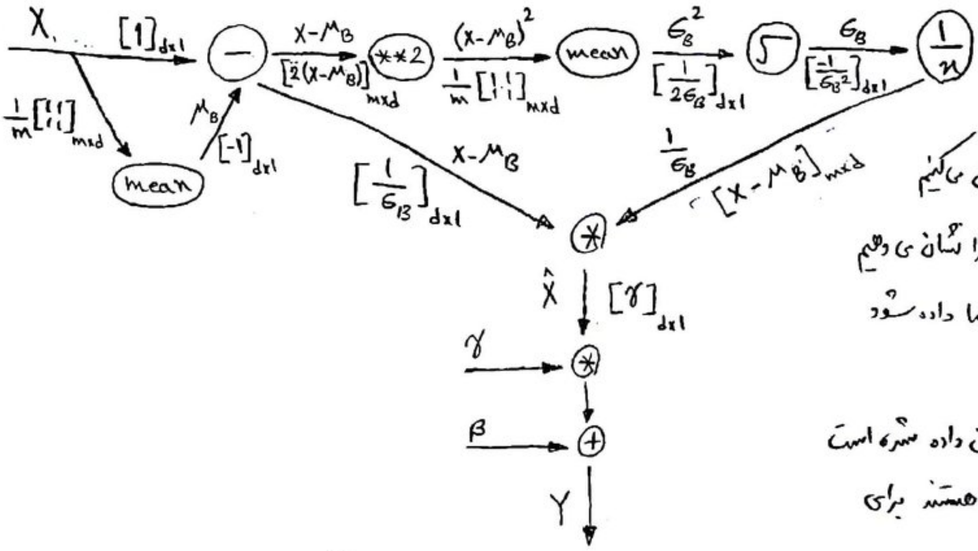
$$\frac{\partial Z}{\partial W} = \begin{bmatrix} \frac{\partial z_1}{\partial w_1} & \cdots & \overset{X_{1+k}}{\frac{\partial z_1}{\partial w_k}} \\ \vdots & & \vdots \\ \frac{\partial z_\ell}{\partial w_1} & \cdots & \frac{\partial z_\ell}{\partial w_k} \end{bmatrix}_{\ell \times k}$$

$$\frac{\partial L}{\partial W} = \left[\frac{\partial Z}{\partial W}^T \right]_{k\times \ell} @ \left[\frac{\partial L}{\partial Z} \right]_{\ell \times 1} \longrightarrow \frac{\partial L}{\partial w_j} = \left[\frac{\partial Z}{\partial w_j}^T \right]_{1\times \ell} @ \left[\frac{\partial L}{\partial Z} \right]_{\ell\times 1} = \begin{bmatrix} \frac{\partial z_1}{\partial w_j} \\ \vdots \\ \frac{\partial z_\ell}{\partial w_j} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial L}{\partial z_1} \\ \vdots \\ \frac{\partial L}{\partial z_\ell} \end{bmatrix}$$

$$\longrightarrow \frac{\partial L}{\partial w_j} = \sum_{i=1}^{\ell} \underbrace{\frac{\partial z_i}{\partial w_j}}_{X_{i+j}} \times \underbrace{\frac{\partial L}{\partial z_i}}_{\substack{w'_i \\ \text{می نامیم}}} \longrightarrow \frac{\partial L}{\partial w_j} = \sum_{i=1}^{\ell} w'_i \, X_{j+i}$$

فرض شده مقدار w'_i را برای همه مقادیر i می دانیم
و مشاهده می شود که دقیقاً به فرم اعمال کانولوشن مانند
صورت سوال رسیدیم

3.a



⊛ مسئله را در کلی ترین حالت و به صورت ماتریسی حل می کنیم

⊛ تمام ضرب ها element wise است و با * آنرا نشان می دهیم

⊛ $\frac{\partial L}{\partial Y}$ گرادیان upstream است که باید به ما داده شود

⊛ مقادیر forward, backward روی گراف نشان داده شده است
مقادیری که ابعاد شان ذکر شده و درون براکت هستند برای
backward هستند

$$\frac{\partial L}{\partial \gamma} = \text{sum}\left(\left[\frac{\partial L}{\partial Y}\right]_{m\times d} * \overbrace{\left[\frac{\partial Y}{\partial \gamma}\right]}^{[\hat{X}]_{m\times d}} , \text{axis}=0 \right) = \sum_{i=1}^{m} \frac{\partial L}{\partial y_i} * \hat{x}_i$$

$$\frac{\partial L}{\partial \beta} = \text{sum}\left(\left[\frac{\partial L}{\partial Y}\right]_{m\times d} , \text{axis}=0 \right) = \sum_{i=1}^{m} \frac{\partial L}{\partial y_i}$$

$\boxed{3.a}$ ادامه

برای محاسبه $\frac{\partial L}{\partial X}$ باید تمامی مسیرهایی که به X می‌رسد را حساب کنیم :

$$\frac{\partial L}{\partial X} = \left[\frac{\partial L}{\partial Y}\right]_{m\times d} * \left[\gamma\right]_{d\times 1} * \left[\frac{1}{\sigma_B}\right]_{d\times 1}$$

$$+$$

$$\text{sum}\left(\left[\frac{\partial L}{\partial Y}\right]_{m\times d} * \left[\gamma\right]_{d\times 1} * \left[\frac{1}{\sigma_B}\right]_{d\times 1} * \left[-1\right]_{d\times 1},\ \text{axis}=0\right) * \frac{1}{m}\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}_{m\times d}$$

$$+$$

$$\text{Sum}\left(\left[\frac{\partial L}{\partial Y}\right]_{m\times d} * \left[\gamma\right]_{d\times 1} * \left[X-\mu_B\right]_{m\times d},\ \text{axis}=0\right) * \left[\frac{-1}{\sigma_B^2}\right]_{d\times 1} * \left[\frac{1}{2\sigma_B}\right]_{d\times 1} * \frac{1}{m}\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}_{m\times d} * \left[2(X-\mu_B)\right]_{m\times d}$$

$$+$$

$$\text{Sum}\left(\left[\frac{\partial L}{\partial Y}\right]_{m\times d} * \left[\gamma\right]_{d\times 1} * \left[X-\mu_B\right]_{m\times d},\ \text{axis}=0\right) * \left[\frac{-1}{\sigma_B^2}\right]_{d\times 1} * \left[\frac{1}{2\sigma_B}\right]_{d\times 1} * \frac{1}{m}\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}_{m\times d}$$

$$* \ \text{sum}\left(\left[2(X-\mu_B)\right]_{m\times d} * \left[-1\right]_{d\times 1},\ \text{axis}=0\right) * \frac{1}{m}\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}_{m\times d}$$

برای کوتاه‌تر شدن جواب به جای $\frac{\partial L}{\partial Y} * \gamma$ از $\frac{\partial L}{\partial \hat{X}}$ استفاده می‌کنیم .
همچنین برای 2 عبارت آخر σ_B ها را زیر $X-\mu_B$ می‌آوریم تا به $\hat{X}$ تبدیل شوند .

$$\frac{\partial L}{\partial X} = \left[\frac{\partial L}{\partial \hat{X}}\right]_{m\times d} * \left[\frac{1}{\sigma_B}\right]_{d\times 1}$$

$$-\left[\frac{1}{m\sigma_B} * \sum_{i=1}^{m}\frac{\partial L}{\partial \hat{x}_i}\right]_{d\times 1} * \begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}_{m\times d}$$

$$-\left[\sum_{i=1}^{m}\frac{\partial L}{\partial \hat{x}_i} * \hat{x}_i\right]_{d\times 1} * \left[\frac{1}{m\sigma_B}\right]_{d\times 1} * \left[\hat{X}\right]_{m\times d}$$

$$-\left[\sum_{i=1}^{m}\frac{\partial L}{\partial \hat{x}_i} * \hat{x}_i\right]_{d\times 1} * \left[\frac{1}{m\sigma_B}\right]_{d\times 1} * \underbrace{\left[\sum_{j=1}^{m}\hat{x}_j\right]_{d\times 1}} * \frac{1}{m}\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}_{m\times d}$$

چون $\hat{X}$ ماتریس standardize شده است
پس جمع ردیف های آن 0 می‌شود ، عبارت آخر بی‌تأثیر است

بنابراین برای $\frac{\partial L}{\partial x_i}$ کافی است به سطرهای ماتریس $\frac{\partial L}{\partial X}$ نگاه کنیم و داریم :

$$\frac{\partial L}{\partial x_i} = \left(\frac{\partial L}{\partial \hat{x}_i} * \frac{1}{\sigma_B}\right) - \left(\frac{1}{m\sigma_B} * \sum_{j=1}^{m}\frac{\partial L}{\partial \hat{x}_j}\right) - \left(\frac{1}{m\sigma_B} * \hat{x}_i \sum_{j=1}^{m}\frac{\partial L}{\partial \hat{x}_j} * \hat{x}_j\right)$$

3.b

در حالت کلی در شبکه های عصبی به واسطه تغییر وزن های لایه های قبل (در back prop) توزیع ورودی لایه‌های بعدی تغییر می‌کند و این مسئله روی گرادیان جزئی‌ای که برای آن لایه محاسبه می‌شود تأثیر گذاشته و باعث می‌شود گرادیان، سیگنال sub optimal ای برای آپدیت وزن های آن لایه باشد. batch norm با محدود کردن توزیع pre activation های هر لایه، این مسئله را ازبین می‌برد و باعث می‌شود گرادیان های بهتری برای آپدیت وزن ها داشته باشیم. بنابراین مدل سریع تر و stable تر آموزش می‌بیند. (internal covariate shift)
به بیان دیگر coupling بین لایه‌ها کمتر می‌شود.

جدای از پایدار تر شدن مدل نسبت به تغییرات جزئی در وزن ها و به طور کلی ورودی هر لایه، batch norm باعث می‌شود هر لایه بتواند pre activation هایش را مستقل از لایه‌های دیگر، در رنج مناسب activation func بیاورد. این مسئله به خصوص در activation func هایی مانند sigmoid مهم است که اگر ورودی شان در رنج مناسبی نباشد saturate می‌شوند و گرادیان های بسیار کمی (نزدیک صفر) خواهند داشت.

بنابراین batch norm به مشکل vanishing gradients هم کمک می‌کند

طبق موارد ذکرشده batch norm فرایند weight initialization را ساده‌تر کرده و اجازه استفاده از learning rate های بزرگتری را می‌دهد.
اجازه استفاده از activation func های بیشتری را داده و می‌توان شبکه های عمیق‌تری داشت زیرا وابستگی لایه‌ها کمتر شده است و جریان گرادیان بهتری داریم

3.c

استفاده از $\hat{X}$ قدرت مدل را به شدت کاهش می‌دهد زیرا pre activation ها همگی حول صفر و با واریانس 1 خواهند بود. به این صورت تنها از بخشی از activation func استفاده می‌شود. مثلاً در sigmoid تنها از قسمت شبه خطی آن استفاده می‌شود و یا در relu دیگر آن expressive بودن نورون ها در بخش مثبت را نداریم.

هدف اصلی آن است که $Y = \gamma \hat{X} + \beta$ هر میانگین و واریانسی بتواند داشته باشد اما میانگین و واریانس آن از لایه های قبل decouple شده باشد

3.d

یکی از چالش هایی که در استفاده از batch norm داریم آن است که moving average به دست آمده در زمان آموزش نزدیک به میانگین هر batch در زمان تست است. به واسطه مشکل ذکرشده میانگین به دست آمده برای هر batch در زمان آموزش، به سمت یکی از کلاس های سگ یا گربه بایاس می‌شود و بنابراین توزیع داده های متفاوتی را در زمان آموزش و تست (برای هر batch) خواهیم داشت. بنابراین pre activation ها به صورت مطلوب نرمالایز نمی‌شوند و بین آموزش و تست discrepancy خواهیم داشت

4.a

ضرر ایجاد مشکل نمی‌کند. وقتی پس از لایه conv از batch norm استفاده شود، مقادیر آن لایه standardize می‌شوند و بایاس که به همه پیکسل‌های هر feature map اضافه شده است، از بین می‌رود. ضمناً خود لایه batch norm دارای پارامتر β است که حکم بایاس را دارد. بنابراین معمولاً وقتی از batch norm استفاده می‌شود، بایاس لایه قبل را False می‌کنند.

4.b

فرض می‌کنیم منظور از وزن‌ها، وزن‌های conv layer باشد

در این صورت عملکرد شبکه در زمان تست دچار مشکل می‌شود. چه وزن‌ها در α ضرب شوند چه ورودی، ورودی $\overbrace{BN}^{XW}$ α برابر می‌شود. حال چون μ_B و σ_B از moving average زمان آموزش به دست آمده پس داریم:

$$BN_{\gamma,\beta}(\alpha X \cdot W) = \gamma * \frac{\alpha XW - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

که با حالت مورد انتظار برای تست متفاوت است.

⊛ شایان ذکر است که این مورد در زمان آموزش تأثیری ندارد زیرا standardization اثر آنرا خنثی می‌کند

⊛ اگر منظور از وزن‌ها μ_B، σ_B، γ و β نیز باشد، در حالتی که وزن‌ها در α ضرب شوند خروجی BN نیز تقریباً α برابر می‌شود

4.c

برای اعمال یک کرنل داریم:

$$H \times W \times C \longrightarrow \boxed{\text{conv } k \times k, s} \longrightarrow H' \times W' \times 1$$

$$H' = \left\lceil \frac{H-k+1}{s} \right\rceil$$

$$W' = \left\lceil \frac{W-k+1}{s} \right\rceil$$

$$\text{تعداد ضرب} = H' \times W' \times k \times k \times C$$

$$\text{تعداد جمع} = H' \times W' \times (k \times k \times C - 1)$$

$$\longrightarrow \text{مرتبه محاسباتی} = (H-k)(W-k)\frac{k^2 C}{s^2}$$

$$H \times W \times C \longrightarrow \boxed{\text{avg pool } k \times k, s} \longrightarrow H' \times W' \times C$$

$$H' = \left\lceil \frac{H-k+1}{s} \right\rceil$$

$$W' = \left\lceil \frac{W-k+1}{s} \right\rceil$$

$$\text{تعداد جمع و تقسیم} = H' \times W' \times C \times (k \times k - 1 + 1) \longrightarrow \text{مرتبه محاسباتی} = (H-k)(W-k)\frac{k^2 C}{s^2}$$

4.c

ادامه

در حالت کلی اگر در فاز train باشیم و batch size = m داریم :

$H \times W \times C \longrightarrow \boxed{\text{batch norm}} \longrightarrow H \times W \times C$

$$\text{calc } \mu_B : \; C\,(\overbrace{m-1}^{\text{جمع}} + \overbrace{1}^{\text{تقسیم}})\,HW$$

$$\text{calc } \sigma_B : \; C\,(\underbrace{1}_{\text{تفریق}} + \underbrace{1}_{\text{توان } 2} + \underbrace{m-1}_{\text{جمع}} + \underbrace{1}_{\text{تقسیم}} + \underbrace{1}_{\sqrt{}})\,HW$$

$$\text{calc } \hat{X} : \; C\,(\underbrace{1}_{\text{تفریق}} + \underbrace{1}_{\text{تقسیم}})\,HW$$

$$\text{calc } Y : \; C\,(\underbrace{1}_{\text{ضرب}} + \underbrace{1}_{\text{جمع}})\,HW$$

در صورت سوال چون گفته شده برای یک تصویر ورودی پس مرتبه محاسباتی CHW می‌شود

4.d

از 1x1 conv می‌توان برای کنترل تعداد چنل ها استفاده کرد. اکثراً کاربرد آن، آن است که قبل از لایه‌های conv دیگر با سایز کرنل بالاتر، از 1x1conv استفاده می‌کنیم تا عمق را کاهش دهیم و هزینه محاسبات را کاهش دهیم. به عنوان مثال inception architecture گوگل قبل از لایه‌های کانولوشن 3x3 و 5x5 از آن استفاده شد. (به طور مستقیم هم می‌توان عمق را توسط لایه‌های conv با اندازه کرنل بالاتر تعیین کرد اما بسیار کندتر خواهد بود)

4.e

لایه‌های pooling مزایای زیر را دارند :

- با کاهش ابعاد feature map ها هزینه محاسبات را پایین می‌آورند
- با حذف پیچرهای کم اهمیت‌تر مدل را تا حدی در برابر تغییر ورودی robust می‌کنند و تا حدی translation invarience ایجاد می‌کنند
- generalization مدل را افزایش می‌دهند و تا حدی overfitting را بهبود می‌دهند
- به واسطه کاهش ابعاد، receptive field را افزایش می‌دهند

نکته مهم آن است که این موارد را توسط روش‌های دیگری نیز می‌توان به دست آورد اما pooling پارامتری برای learn شدن ندارد و محاسبات را سریع‌تر و efficient تر می‌کند.

4.f

اعداد مختلفی را برای سایز ورودی امتحان کردیم به طوریکه به سایز کرنل تمام مراحل بخش پذیر باشد. عدد 21 و مضارب آن این ویژگی را دارند:

$$21\times21 \xrightarrow{\text{conv } 3\times3 \quad s=2} 10\times10 \xrightarrow{\text{pool } 2\times2} 5\times5 \xrightarrow{\text{conv } 3\times3 \quad s=2} 2 \xrightarrow{\text{pool } 2\times2} 1$$

پس هر پیکسل خروجی تحت تأثیر 21x21 پیکسل ورودی است

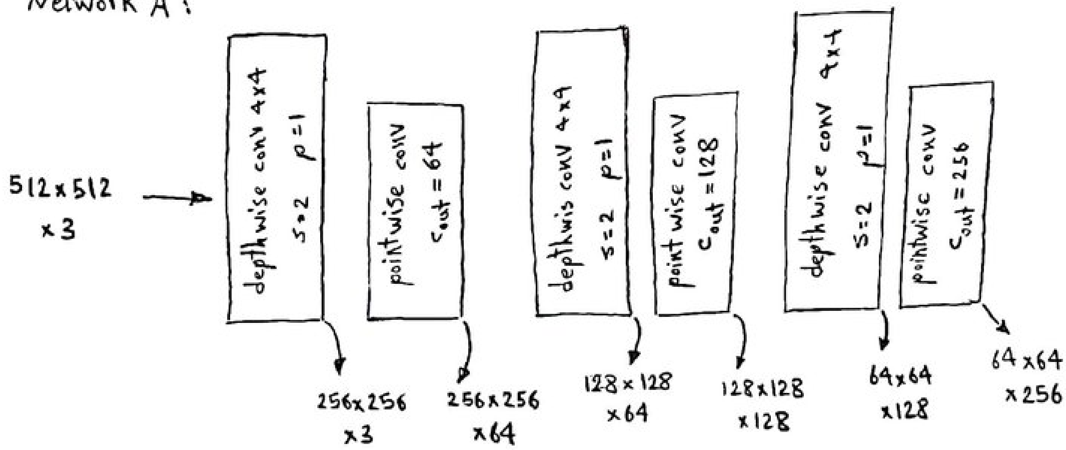
$$\text{receptive field} = 21^2 = 441$$

5.a

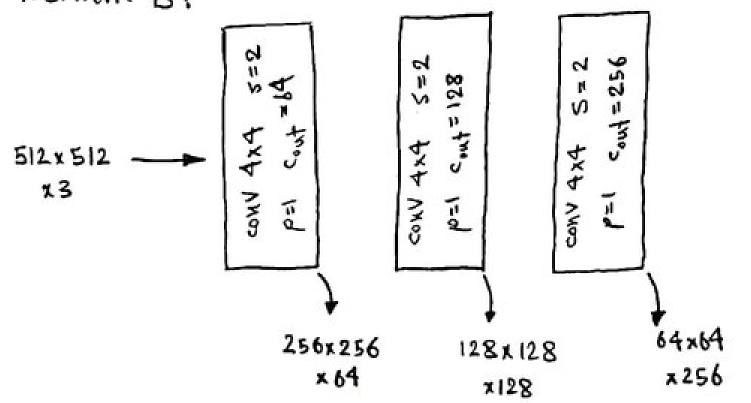
فرمول به دست آوردن
H,W خروجی

$$O = \left\lceil \frac{i+2p-k+1}{s} \right\rceil$$

Network A :



Network B :



5.b

Network A parameters :

$(4\times4\times3)+(3\times64) = 240$

$+$

$(4\times4\times64)+(64\times128) = 9{,}216$

$+$

$(4\times4\times128)+(128\times256) = 34{,}816$

$= 44{,}272$

Network B parameters :

$(4\times4\times3)\times64 = 3{,}072$

$+$

$(4\times4\times64)\times128 = 131{,}072$

$+$

$(4\times4\times128)\times256 = 524{,}288$

$= 658{,}432$

5.c

Network A mul operations:

$$(4\times4)(256\times256\times3) + (3\times256\times256\times64) = 15,728,640$$

$+$

$$(4\times4)(128\times128\times64) + (64\times128\times128\times128) = 150,994,944$$

$+$

$$(4\times4)(64\times64\times128) + (128\times64\times64\times256) = 142,606,336$$

$$= 309,329,920$$

Network B mul operations:

$$(4\times4\times3)(256\times256\times64) = 201,326,592$$

$+$

$$(4\times4\times64)(128\times128\times128) = 2,147,483,650$$

$+$

$$(4\times4\times128)(64\times64\times256) = 2,147,483,650$$

$$= 4,496,293,888$$

5.d

با اینکه ابعاد Feature map ها در نهایت یکسان است اما شبکه A پارامتر و محاسبات بسیار کمتری دارد
از این نظر می‌توان از این روش در شرایطی که منابع محدودی داریم استفاده کنیم

5.e

هدف از معماری mobileNet آن است که حجم مدل، پیچیدگی محاسبات را پایین بیاورد تا بتوان از لایه های کانولوشنی در دستگاه های با منابع محدود، mobile نیز استفاده کرد.

به این منظور نشان داده می‌شود که هر لایه کانولوشن عادی را می‌توان به 2 لایه depthwise conv و pointwise conv تقسیم کرد به طوری که ابعاد ورودی و خروجی این لایه ها با کانولوشن عادی یکسان است اما تعداد پارامتر ها و مرتبه محاسبات کمتر می‌شود (مانند چیزی که در بخش قبل دیدیم)

در لایه‌ی depthwise conv کانولوشن تنها بر روی یک چنل ورودی (عمق=1) انجام می‌شود و بنابراین به تعداد چنل های ورودی فیلتر kxk داریم

سپس در لایه‌ی pointwise conv به اندازه‌ی تعداد چنل های خروجی مورد نیاز، فیلتر های 1x1 استفاده می‌کنیم.

در حقیقت لایه های depthwise در هر چنل به صورت مجزا دنبال پترن می‌گردند اما لایه های pointwise از ترکیب چنل ها استفاده کرده اما از همسایگی های داخل هر feature map یعنی ویژگی های spatial استفاده نمی‌کنند

به طور کلی در حالتی که ابعاد ورودی و خروجی تغییر نکند و فیلتر kxk داشته باشیم $\left(\begin{array}{c} C_{in}=M \\ stride=1 \\ C_{out}=N \end{array}\right)$ در روش conv عادی و depthwise separable را به صورت زیر می‌شود مقایسه کرد:

	#params	#operations
depthwise separable conv	$NM+Mk^2$	$NMHW+MHWk^2$
conv	NMk^2	$NMHWk^2$

⊛ همان طور که مشاهده می‌شود مرتبه محاسبات و تعداد پارامتر ها بسیار کمتر شده و ضمناً نشان داده شده که دقت کار تا حد خوبی حفظ شده است

5.f

⊛ width multiplier (α)

مقداری بین 0 و 1 است که در عمق (تعداد چنل) ضرب شده و آنرا کاهش می‌دهد یعنی: $M \to \alpha M$ و $N \to \alpha N$

بنابراین تعداد عملیات های محاسباتی و تعداد پارامتر ها کاهش می‌یابد که می‌توان مقدار آنرا توسط جدول بخش قبل حساب کرد.

کاربرد این هایپرپارامتر نازک‌تر کردن لایه ها در مواردی است که منابع کمتری داریم. البته قطعاً دقت نیز با کاهش α کاهش می‌یابد

⊛ resolution multiplier (ρ)

مقداری بین 0 و 1 است که در ابعاد (H,W) ورودی ضرب شده و آنرا کاهش می‌دهد یعنی: $H \to \rho H$ و $W \to \rho W$

این هایپرپارامتر تعداد پارامتر ها را تغییری نمی‌دهد (از جدول معلوم است) اما تعداد عملیات ها را ρ^2 برابر می‌کند.

به صورت ضمنی با تعیین resolution ورودی این هایپرپارامتر ست می‌شود

$\boxed{6.a}$ به طور معمول از این تابع ضرر به عنوان regularization term در کنار توابع دیگری مانند cross entropy loss استفاده می‌شود.

یکی از مشکلاتی که در دسته‌بندی و به خصوص در حوزه تشخیص چهره با آن مواجه هستیم کم بودن فاصله کلاس های مختلف از هم است.

به این صورت که مدل دسته‌بندی را به درستی انجام می دهد اما representation های به دست آمده برای سمپل های کلاس های مختلف به هم نزدیک هستند، اگر دیتای تست با اعمال تغییر کوچکی از سمپل درست دسته‌بندی شده به دست آید، ممکن است مدل آن را به کلاس دیگری نسبت دهد.

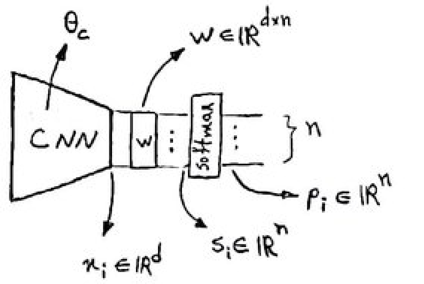
استفاده از center loss به همراه cross entropy loss باعث می‌شود که $\underbrace{\text{inter class dispension}}_{\text{cross entropy loss توسط}}$ ، $\underbrace{\text{intra class compactness}}_{\text{center loss توسط}}$ را همراه هم داشته باشیم و مرز بین representation های سمپل های هر کلاس بیشتر شود.

برای این منظور از فرمول $\mathcal{L}_C = \frac{1}{2}\sum_{i=1}^{m}\|x_i - c_{y_i}\|^2$ برای center loss استفاده می‌شود که فاصله representation سمپل های هر کلاس (یعنی x_i) را از مرکز آن دسته کمتر می کند.

⊛ نکته حائز اهمیت آن است که استفاده هردو لاس ذکر شده به صورت joint است که بهترین بهبود را ایجاد می کند.

- اگر فقط از cross entropy loss استفاده شود که همان مشکل اولیه را داریم
- اگر فقط از center loss استفاده شود مدل ترغیب می‌شود که همه representation ها را 0 به دست آورد

$\boxed{6.b}$ $\mathcal{L} = \underbrace{\mathcal{L}_s}_{\text{cross entropy loss}} + \lambda \underbrace{\mathcal{L}_c}_{\text{center loss}}$



$$\xrightarrow{\text{update } W} \quad W^{t+1} = W^t - \eta^t \frac{\partial L^t}{\partial W^t}$$

$$\frac{\partial L}{\partial W} = \frac{\partial L_s}{\partial W} = X^T @ \frac{\partial L_s}{\partial s} = \left[X^T\right]_{d\times m} @ \left[\quad\right]_{m\times n}$$

$$\begin{cases} p_{ij} - 1 & j = \text{true class} \\ p_{ij} & j \neq \text{true class} \end{cases}$$

$$\xrightarrow{\text{update } \theta_c} \quad \theta_c^{t+1} = \theta_c^t - \eta^t \underbrace{\sum_{i=1}^{m} \frac{\partial L^t}{\partial x_i^t} \times \frac{\partial x_i^t}{\partial \theta_c^t}}_{\left[\frac{\partial L^t}{\partial X^t}\right] \cdot \left[\frac{\partial X^t}{\partial \theta_c^t}\right] \longrightarrow \text{CNN داخل}}$$

$$\frac{\partial L}{\partial X} = \overbrace{\frac{\partial L_s}{\partial X}}^{\frac{\partial L_s}{\partial s} @ W^T} + \lambda \underbrace{\frac{\partial L_c}{\partial X}}_{\begin{bmatrix} x_1 - c_{y_1} \\ \vdots \\ x_m - c_{y_m} \end{bmatrix}_{m\times d}}$$

$$\xrightarrow{\text{update } c_j} \quad c_j^{t+1} = c_j^t - \alpha\, \Delta c_j^t$$

$$\Delta c_j = \frac{\sum_{i=1}^{m} 1_{y_i = j} \times (c_j - x_i)}{1 + \sum_{i=1}^{m} 1_{y_i = j}}$$

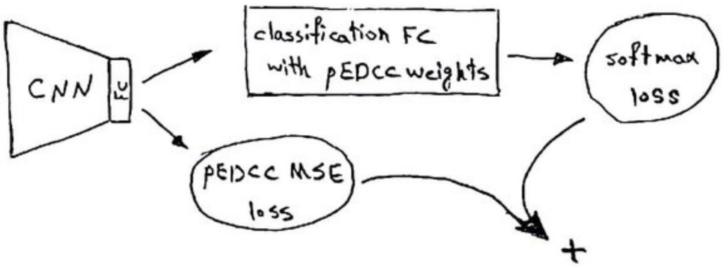
⊛ در حالت کلی باید برای به دست آوردن مراکز دسته ها، میانگین را به ازای تمام سمپل ها بگیریم اما این کار هزینه بسیار زیادی دارد. بنابراین Δc_j^t را به صورت روبه رو به عنوان جایگزین این کار تعریف می کنیم.

همان طور که از فرمول معلوم است، میانگین فاصله تا مرکز برای سمپل های هر دسته (به ازای mini batch) حساب می‌شود و در جهت عکس آن آپدیت می کنیم.

6.c

در این مقاله از PEDCC استفاده شده است تا به صورت evenly distributed مراکز دسته ها پخش شوند.

در روش center loss برای مراکز دسته ها از یک تقریب استفاده کردیم. در اینجا نیز تفاوت آن است که با روش دیگری این مراکز دسته ها را مدیریت می کنیم. اگر hyper sphere واحد را در فضای representation تصور کنیم، در این روش مراکز دسته ها در ابتدا به صورت evenly distributed روی سطح این ابرکره قرار می گیرند و فیکس می شوند. در ادامه به صورت تقریبا مشابه از توابع خطای softmax و MSE به صورت زیر استفاده می شود:



* نکته حائز اهمیت آن است که چون مراکز دسته ها فیکس هستند یک لایه FC جدید قبل softmax اضافه می شود که وزن های آن فریز شده و در فرآیند آموزش آپدیت نمی شوند

بنابراین با جایگذاری و فیکس کردن وزن های PEDCC در لایه قبل softmax، مراکز دسته ها فیکس شده و در تئوری بیشترین فاصله را از هم دارند سپس 2 تابع لاس مانند قبل inter class dispersion و intra class compactness را ایجاد می کنند اما به واسطه مراکز دسته هایی که از هم فاصله مناسبی دارند نتیجه بهتری در تفکیک representation سمپل ها به وجود می آید

* در برخی سوال ها با علی بابابیک همفکری داشتم