经常让 vanishing gradients 发生的 **Activation func:** 1. **ReLU** saturates for negative values, giving zero gradient. **2.** Leaky ReLU: max(0.01x, x). Does not saturate. Is not di↵erentiable at x = 0. Does not approximates identity near the origin **3. Tanh**, like the **sigmoid** unit, saturates when x very positive or negative. Is differentiable everywhere as it has no discontinuities. Approximates identity near the origin.and no additional learning occurs. **4. Sigmoid**. At extremes, the unit saturates and thus has zero gradient. This results in no learning with gradient descent **5.**Gaussian Error Linear Unit (GELU): it is popular in recent transformer structures like BERT. Saturates when x is very negative. Is differentiable everywhere as it has no discontinuities. Does not approximate identity near the origin. ^^^

$f(z) = \begin{cases} 1, & z \geq 0 \\ -1, & z < 0 \end{cases}$ 这个 activation func has 0 gradients for all inputs (except at 0 where it's non-differentiable)意味着没 learning

$f(x) = x^3$ This activation func is nonlinear and differentiable everywhere, 这满足 good activation func 条件. 但很可能造成 exploding gradients input value 大, gradient 大. 而且, small inputs 会导致 vanishing gradients, e.g. inputs that are close to zero.

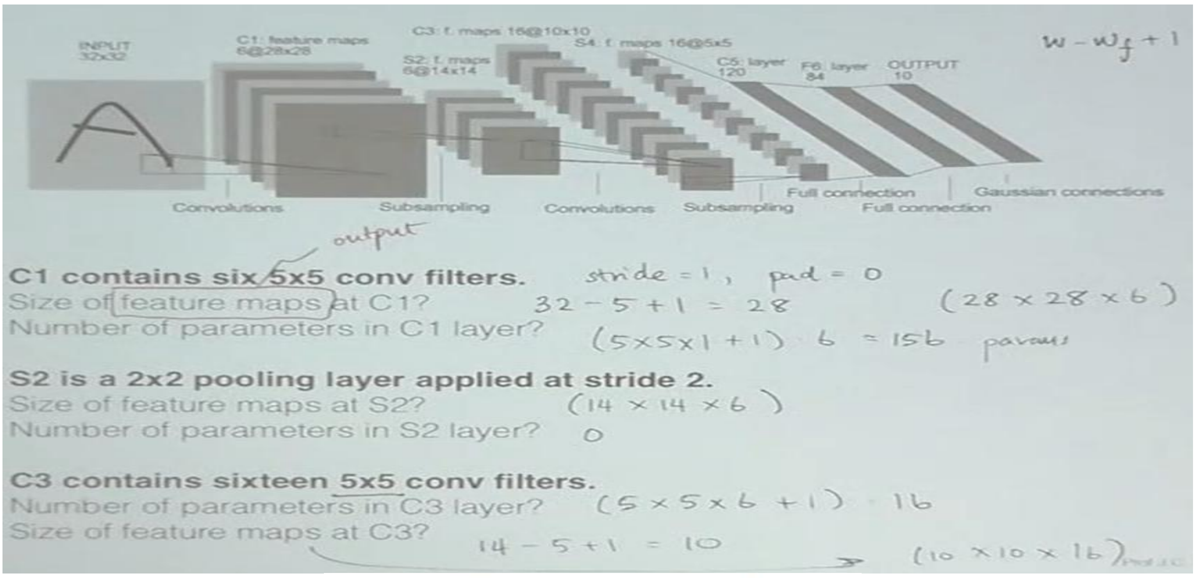
CNN

有 5 个 3*3 filter,stride=1, padding = 1,再 relu()再 2 × 2 max pooling with stride 2. Input image size 256×256×3. First conv-pool layer 的 trainable parameters (including bias):(3*3(filterSize)*3(RGB)+1(biasTerm))*5(filterNum) =140^^^**CNN** 32 *32 * 3 第一层是 CNN, output=26 * 26 *16 filters number in the first convolutional = depth of the output volume(16),

$$\text{Output size} = \frac{\text{Input size} - \text{Filter size} + 2 \times \text{Padding}}{\text{Stride}} + 1 \qquad 26 = \frac{32 - \text{Filter size} + 0}{1} + 1$$

each filter has dimensions of 7 * 7 * 3. **Trainable parameters** 数量是 32(第二层 filter 数)*3*3*16(filter). ^^^input 32*32*3, fully connected architecture 有 500 output neuron; convolutional layer 有 4 filter(4*4), 则 convolutional layer 的 output: 29(32-4+1)*29*4. FC 的 param:(32*32*3+1)*500, convolutional param(4*4*3+1)*4(filterNum)

Convolutional layers 比 fully connected layers 有更少的 param 因为 Convolutional layers have sparse connectivity and share parameters. stack layer 越多 receptive field 越大



First layer: $\mathbf{h}_1 = f(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{W}_s\mathbf{x}$

^^^**forward propagation** Second (Output) layer: $\mathbf{z} = \mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2$.有第二个 convolutional layer(有 32 filters, 3 *3) depth of each filter in the second convolutional layer must match the depth of the input volume,所以是 16 ^^^

where $\mathbf{b}_1 \in \mathbb{R}^m$ and $\mathbf{b}_2 \in \mathbb{R}^l$. $\mathbf{W}_s$ is the linear mapping for the "shortcut" connection, f is the non-linear activation function like ReLU. $\mathbf{W}_1 \in \mathbb{R}^{m\times n}$, $\mathbf{W}_s \in \mathbb{R}^{m\times n}$, $\mathbf{W}_2 \in \mathbb{R}^{l\times m}$. Total number of trainable parameters:= mn + m + mn + ml + l = 2mn + ml + m + l..^^

Backpropagation

1.Binary 分类,1wX 光没病 1k 有病数据集(很不平衡).用 **Data Augmentation**(能解决 image classification 的 class imbalance 问)解决(对 train data/label 进行 Translation,**random Cropping**(flipping/mirroring 改 input pixel 但 label 不变能增加 train data point) , **label smooth**(add noise to network,把 label 概率 1 分给其他 label 一些

$\mathcal{L}^{(i)} = \sum_{c=1}^{C} -y_c \log p_c$ $\mathcal{L}^{(i)} = \overset{0\sim 1}{y_c}(1-\alpha)\log p_c + \alpha/(K-1)$), Reflection,Gaussian Blurring). 去建神经网络

$$\begin{aligned} \mathbf{z}_1 &= \mathbf{W}_1\mathbf{x}^{(i)} + \mathbf{b}_1 \\ \mathbf{a}_1 &= ReLU(\mathbf{z}_1) \\ z_2 &= \mathbf{W}_2\mathbf{a}_1 + b_2 \\ \hat{y}^{(i)} &= \sigma(z_2) \end{aligned}$$

$$\mathcal{L}^{(i)} = \alpha \cdot y^{(i)} \cdot \log(\hat{y}^{(i)}) + \beta \cdot (1-y^{(i)}) \cdot \log(1-\hat{y}^{(i)})$$

$$J = -\frac{1}{m}\sum_{i=1}^{m}\mathcal{L}^{(i)}$$

where $\hat{y}^{(i)} \in \mathbb{R}$, $y^{(i)} \in \mathbb{R}$, $\mathbf{x}^{(i)} \in \mathbb{R}^{D_x \times 1}$, $\mathbf{W}_1 \in \mathbb{R}^{D_{a_1} \times D_x}$, $\mathbf{W}_2 \in \mathbb{R}^{1 \times D_{a_1}}$, $\mathbf{b}_1 \in \mathbb{R}^{D_{a_1}}, b_2 \in \mathbb{R}$. Hyperparameter α和β的用处: Weigh 每个 class 给 loss func 贡献多少 能帮 GC 因为 network 会 take larger steps 当从 underrepresented class instances 学习. A取 1,β取 10,因为 sample 比是 10,这样每个 class 的 samples 才会 contribute equally. $\nabla_{\hat{y}^{(i)}}\mathcal{L}^{(i)} = \alpha\frac{y^{(i)}}{\hat{y}^{(i)}} - \beta\frac{1-y^{(i)}}{1-\hat{y}^{(i)}} = \delta_{\hat{y}^{(i)}}$ 求 $\nabla_{b_2}\mathcal{L}^{(i)}$ and denote it as δ_{b_2}. Hint: $\frac{d\sigma(z)}{dz} = \sigma(z)(1-\sigma(z))$, $\nabla_{z_2}\hat{y}^{(i)} = \sigma(z_2)[1-\sigma(z_2)]$. backpropagate to z2: $\nabla_{z_2}\mathcal{L}^{(i)} = \sigma(z_2)[1-\sigma(z_2)]\nabla_{\hat{y}^{(i)}}\mathcal{L}^{(i)}$. 因为是+号(pass gradient)所以

$$\begin{aligned} \nabla_{b_2}\mathcal{L}^{(i)} &= \sigma(z_2)[1-\sigma(z_2)]\nabla_{\hat{y}^{(i)}}\mathcal{L}^{(i)} \\ \delta_{b_2} &= \sigma(z_2)[1-\sigma(z_2)]\delta_{\hat{y}^{(i)}} \end{aligned}$$.

$\nabla_{\mathbf{W}_2}\mathcal{L}^{(i)} = \mathbf{a}_1^T\delta_{b_2} = \delta_{\mathbf{W}_2}$ ✕号 switch gradient.对于 b1:backpro 到 a1 $\nabla_{\mathbf{a}_1}\mathcal{L}^{(i)} = \mathbf{W}_2^T\nabla_{b_2}\mathcal{L}^{(i)} = \mathbf{W}_2^T\delta_{b_2}$ back 到 z1 $\nabla_{\mathbf{z}_1}\mathcal{L}^{(i)} = \mathbb{I}(\mathbf{z}_1 > 0) \odot \nabla_{\mathbf{a}_1}\mathcal{L}^{(i)}$.

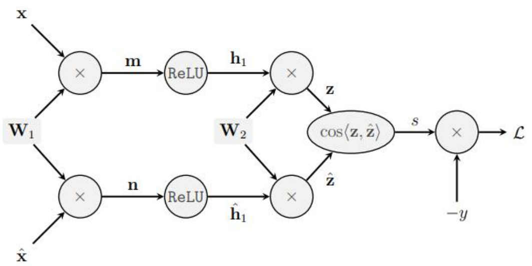
因为是加号所以直接 pass $\nabla_{\mathbf{b}_1}\mathcal{L}^{(i)} = \nabla_{\mathbf{z}_1}\mathcal{L}^{(i)} = \delta_{\mathbf{b}_1}$. $\nabla_{\mathbf{W}_1}\mathcal{L}^{(i)} = \nabla_{\mathbf{b}_1}\mathcal{L}^{(i)}\mathbf{x}^{(i)T}$, $\delta_{\mathbf{W}_1} = \delta_{\mathbf{b}_1}\mathbf{x}^{(i)T}$ 因为是✕号(switch).Add L2 regu(len=1)to W2 $\hat{J} = J + \|\mathbf{W}_2\|_2^2$. 用 vanilla GC(learning rate ϵ), 则 update role: $\mathbf{W}_2^{(k+1)} = \mathbf{W}_2^{(k)} - \epsilon[\nabla_{\mathbf{W}_2}J^{(k)} + 2\mathbf{W}_2^{(k)}]$

2. If the i^{th} pair of input $(\mathbf{x}^{(i)}, \hat{\mathbf{x}}^{(i)})$ is composed of signature images both of which are genuine, then the label for the i^{th} example is +1 ($y^{(i)} = +1$).

If the i^{th} pair of input $(\mathbf{x}^{(i)}, \hat{\mathbf{x}}^{(i)})$ is composed of signature images both of which are forged, then the label for the i^{th} example is -1 ($y^{(i)} = -1$).

If the i^{th} pair of input $(\mathbf{x}^{(i)}, \hat{\mathbf{x}}^{(i)})$ is composed of signature images one of which is genuine and the other is forged, then the label for the i^{th} example is -1 ($y^{(i)} = -1$).

$$\begin{aligned} \mathbf{h}_1 &= \text{ReLU}(\mathbf{W}_1\mathbf{x}) \\ \hat{\mathbf{h}}_1 &= \text{ReLU}(\mathbf{W}_1\hat{\mathbf{x}}) \\ \mathbf{z} &= \mathbf{W}_2\mathbf{h}_1 \\ \hat{\mathbf{z}} &= \mathbf{W}_2\hat{\mathbf{h}}_1 \\ s &= \cos\langle\mathbf{z}, \hat{\mathbf{z}}\rangle = \frac{\mathbf{z}^T\hat{\mathbf{z}}}{\|\mathbf{z}\|_2\|\hat{\mathbf{z}}\|_2} \\ \mathcal{L} &= -y \cdot s \end{aligned}$$



$$\frac{d}{dz}\left(\frac{f(z)}{g(z)}\right) = \frac{f'(z)g(z) - g'(z)f(z)}{g(z)^2}$$

for $f'(z) = \frac{df(z)}{dz}$ and $g'(z) = \frac{dg(z)}{dz}$.

$$\nabla_{\mathbf{z}}\mathcal{L} = -y\frac{\partial s}{\partial \mathbf{z}} \qquad \nabla_{\hat{\mathbf{z}}}\mathcal{L} = -y\frac{\partial s}{\partial \hat{\mathbf{z}}}.$$

$$\frac{\partial s}{\partial \mathbf{z}} = \frac{(\|\mathbf{z}\|_2\|\hat{\mathbf{z}}\|_2)\hat{\mathbf{z}} - (\mathbf{z}^T\hat{\mathbf{z}})(\frac{\|\hat{\mathbf{z}}\|_2}{\|\mathbf{z}\|_2})\mathbf{z}}{\|\mathbf{z}\|_2^2\|\hat{\mathbf{z}}\|_2^2} \qquad \delta_{\mathbf{z}} = -y \cdot \frac{(\|\mathbf{z}\|_2\|\hat{\mathbf{z}}\|_2)\hat{\mathbf{z}} - (\mathbf{z}^T\hat{\mathbf{z}})(\frac{\|\hat{\mathbf{z}}\|_2}{\|\mathbf{z}\|_2})\mathbf{z}}{\|\mathbf{z}\|_2^2\|\hat{\mathbf{z}}\|_2^2}$$

$$\frac{\partial s}{\partial \hat{\mathbf{z}}} = \frac{(\|\mathbf{z}\|_2\|\hat{\mathbf{z}}\|_2)\mathbf{z} - (\mathbf{z}^T\hat{\mathbf{z}})(\frac{\|\mathbf{z}\|_2}{\|\hat{\mathbf{z}}\|_2})\hat{\mathbf{z}}}{\|\mathbf{z}\|_2^2\|\hat{\mathbf{z}}\|_2^2}. \qquad \delta_{\hat{\mathbf{z}}} = -y \cdot \frac{(\|\mathbf{z}\|_2\|\hat{\mathbf{z}}\|_2)\mathbf{z} - (\mathbf{z}^T\hat{\mathbf{z}})(\frac{\|\mathbf{z}\|_2}{\|\hat{\mathbf{z}}\|_2})\hat{\mathbf{z}}}{\|\mathbf{z}\|_2^2\|\hat{\mathbf{z}}\|_2^2}$$

$$\nabla_{\mathbf{W}_2}\mathcal{L} = \frac{\partial \mathbf{z}}{\partial \mathbf{W}_2}\frac{\partial \mathcal{L}}{\partial \mathbf{z}} + \frac{\partial \hat{\mathbf{z}}}{\partial \mathbf{W}_2}\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{z}}}. \qquad \delta_{\mathbf{W}_2} = \delta_z\mathbf{h}_1^T + \delta_{\hat{z}}\hat{\mathbf{h}}_1^T$$

$$\nabla_{\mathbf{h}_1}\mathcal{L} = \frac{\partial \mathbf{z}}{\partial \mathbf{h}_1}\frac{\partial \mathcal{L}}{\partial \mathbf{z}} \qquad \nabla_{\hat{\mathbf{h}}_1}\mathcal{L} = \frac{\partial \hat{\mathbf{z}}}{\partial \hat{\mathbf{h}}_1}\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{z}}}. \qquad \frac{\partial \mathbf{z}}{\partial \mathbf{h}_1} = \frac{\partial \hat{\mathbf{z}}}{\partial \hat{\mathbf{h}}_1} = \mathbf{W}_2^T,$$

$$\delta_{\mathbf{h}_1} = \mathbf{W}_2^T\delta_{\mathbf{z}} \qquad \delta_{\hat{\mathbf{h}}_1} = \mathbf{W}_2^T\delta_{\hat{\mathbf{z}}}.$$

$$\nabla_{\mathbf{m}}\mathcal{L} = \frac{\partial \mathbf{h}_1}{\partial \mathbf{m}}\frac{\partial \mathcal{L}}{\partial \mathbf{h}_1} \qquad \nabla_{\mathbf{n}}\mathcal{L} = \frac{\partial \hat{\mathbf{h}}_1}{\partial \mathbf{n}}\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{h}}_1}.$$

$$\delta_{\mathbf{m}} = \mathbb{1}(\mathbf{m} \geq 0) \odot \delta_{\mathbf{h}_1} \qquad \delta_{\mathbf{n}} = \mathbb{1}(\mathbf{n} \geq 0) \odot \delta_{\hat{\mathbf{h}}_1}$$

$$\nabla_{\mathbf{W}_1}\mathcal{L} = \frac{\partial \mathbf{m}}{\partial \mathbf{W}_1}\frac{\partial \mathcal{L}}{\partial \mathbf{m}} + \frac{\partial \mathbf{n}}{\partial \mathbf{W}_1}\frac{\partial \mathcal{L}}{\partial \mathbf{n}}.$$

$$\delta_{\mathbf{W}_1} = \delta_{\mathbf{m}}\mathbf{x}^T + \delta_{\mathbf{n}}\hat{\mathbf{x}}^T$$

(b) (9 points) In the similarity network architecture, $\mathbf{z}$ and $\hat{\mathbf{z}}$ represents the embedding vectors for input signature images $\mathbf{x}$ and $\hat{\mathbf{x}}$ respectively. Suppose we are given a training sample, $\{(\mathbf{x}^{(g)}, \hat{\mathbf{x}}^{(g)}), +1\}$. 计算 train sample 的

loss 如果 $\mathbf{z}^{(g)} = \hat{\mathbf{z}}^{(g)}$. $s^{(g)} = \cos\langle \mathbf{z}^{(g)}, \hat{\mathbf{z}}^{(g)} \rangle = 1.$ $\mathcal{L}^{(g)} = -1 \cdot s^{(g)} = -1.$ 计算 loss 如果 $\mathbf{z}^{(g)}$ and $\hat{\mathbf{z}}^{(g)}$ are orthogonal to each other $s^{(g)} = \cos\langle \mathbf{z}^{(g)}, \hat{\mathbf{z}}^{(g)} \rangle = 0.$ $\mathcal{L}^{(g)} = -1 \cdot s^{(g)} = 0.$ 计算 loss 如果 $\mathbf{z}^{(g)} = -\hat{\mathbf{z}}^{(g)}$. $s^{(g)} = \cos\langle \mathbf{z}^{(g)}, \hat{\mathbf{z}}^{(g)} \rangle = -1.$ $\mathcal{L}^{(g)} = -1 \cdot s^{(g)} = 1.$ (c)解释是否 loss function 让 embedding vectors 到 right direction: From part (b), 通过最小化 loss func, embedding vectors for the input pair of images that are genuine are getting forced to be similar to each other (loss for b(i) is the least) which is what we want to achieve for the signature forgery detection application. **3.**

$$
\begin{aligned}
\mathcal{L} &= \frac{1}{2}||\mathbf{f} - \mathbf{y}||^2 \\
\mathbf{f} &= \max(\alpha \cdot \mathbf{g}, \mathbf{g}), 0 < \alpha < 1 \\
\mathbf{g} &= \mathbf{h} + \mathbf{b} \\
\mathbf{h} &= \mathbf{W}_k \mathbf{x}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial f_i} &= \frac{\partial}{\partial f_i}\left[\frac{1}{2}(\mathbf{f}-\mathbf{y})^T(\mathbf{f}-\mathbf{y})\right] \\
&= \frac{\partial}{\partial f_i}\left[\frac{1}{2}(\mathbf{f}^T\mathbf{f} - 2\mathbf{y}^T\mathbf{f} + \mathbf{y}^T\mathbf{y})\right] \\
&= \frac{1}{2}(2f_i - 2y_i) \\
&= f_i - y_i.
\end{aligned}
\qquad
\frac{\partial \mathcal{L}}{\partial \mathbf{g}} = \frac{\partial \mathbf{f}}{\partial \mathbf{g}} \frac{\partial \mathcal{L}}{\partial \mathbf{f}}.
$$

$$
\frac{\partial f_j}{\partial g_i} = \frac{\partial}{\partial g_i}[\max(\alpha g_j, g_j)] = \begin{cases} 0 & \text{if } i \neq j \\ \alpha & \text{if } i = j \text{ and } \alpha g_j > g_j \\ 1 & \text{if } i = j \text{ and } \alpha g_j \leq g_j \end{cases}
\qquad
\frac{\partial g_j}{\partial h_i} = \frac{\partial}{\partial h_i}(h_j + b_j) = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{else} \end{cases}
$$

Therefore $\frac{\partial \mathbf{g}}{\partial \mathbf{h}} = I$, and analogously $\frac{\partial \mathbf{g}}{\partial \mathbf{b}} = I$. Therefore $\frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}} = \frac{\partial \mathcal{L}}{\partial \mathbf{g}}$.

Finally we can use the provided formula to derive that $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_k} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}} \mathbf{x}^T$. $\mathbf{W}_1\mathbf{x} = \mathbf{W}_1(\mathbf{x} + \lambda \mathbf{z})$. 在 train set 把 x 换成 x+λz, 再 train new dataset. 哪些 param 会变？ Fix a given input x, and consider its obfuscated variant x + λz. Because the NN activations 在 linear transform 之后会一样，所以所有哪些 gradient 都一样. The only 被影响的 gradients 是直接用 input 计算的. If the input is x, then the gradient ∂L ∂W1 = ∂L ∂h x T ; if the input is x+λz, then the gradient is $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}} \mathbf{x}^T$. This gradient 会不同, so the rest of the network will train well. It's worth mentioning that this is only valid on the first step of training, 因为每次改变 w 都要重新计算 z **4.**

$\mathbf{z} = \mathbf{W}_3\text{relu}(\mathbf{W}_1\mathbf{x}) + \mathbf{W}_2\mathbf{x}.$ $\mathcal{L} = \frac{1}{2}||\mathbf{z} - \mathbf{y}||_2^2 = \frac{1}{2}(\mathbf{z} - \mathbf{y})^T(\mathbf{z} - \mathbf{y}),$ with $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{y}, \mathbf{z} \in \mathbb{R}^m$, $\mathbf{W}_1 \in \mathbb{R}^{p \times n}$, $\mathbf{W}_2 \in \mathbb{R}^{m \times n}$, and $\mathbf{W}_3 \in \mathbb{R}^{m \times p}$,

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W_1}} = \frac{\partial \mathbf{b}}{\partial \mathbf{W_1}} \frac{\partial \mathbf{c}}{\partial \mathbf{b}} \frac{\partial \mathbf{d}}{\partial \mathbf{c}} \frac{\partial \mathbf{z}}{\partial \mathbf{d}} \frac{\partial \mathcal{L}}{\partial \mathbf{z}}$$

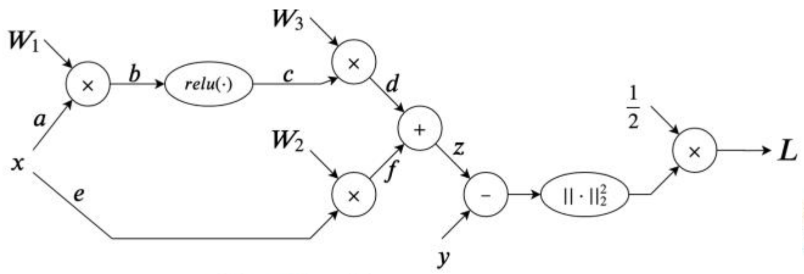
$$\frac{\partial \mathbf{z}}{\partial \mathbf{d}} = \mathbf{I}$$

$$\frac{\partial \mathbf{d}}{\partial \mathbf{c}} = \mathbf{W}_3^T$$

$$\frac{\partial \mathbf{c}}{\partial \mathbf{b}} = diag(\mathbb{1}(\mathbf{b}))$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{b}} &= \frac{\partial \mathbf{c}}{\partial \mathbf{b}} \frac{\partial \mathbf{d}}{\partial \mathbf{c}} \frac{\partial \mathbf{z}}{\partial \mathbf{d}} \frac{\partial \mathcal{L}}{\partial \mathbf{z}} \\ &= \mathbb{1}(\mathbf{b}) \odot (\mathbf{W}_3^T \frac{\partial \mathcal{L}}{\partial \mathbf{z}})\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} &= \frac{\partial \mathbf{b}}{\partial \mathbf{W}_1} \frac{\partial \mathcal{L}}{\partial \mathbf{b}} \\ &= \frac{\partial \mathcal{L}}{\partial \mathbf{b}} \mathbf{x}^T \\ &= \left(\mathbb{1}(\mathbf{b}) \odot (\mathbf{W}_3^T \frac{\partial \mathcal{L}}{\partial \mathbf{z}}) \right) \mathbf{x}^T \\ &= \left(\mathbb{1}(\mathbf{W_1 x}) \odot (\mathbf{W}_3^T \frac{\partial \mathcal{L}}{\partial \mathbf{z}}) \right) \mathbf{x}^T\end{aligned}$$



$$\begin{aligned}\mathcal{L} &= \frac{1}{2} ||\mathbf{z} - \mathbf{y}||_2^2 \\ &= \frac{1}{2} (\mathbf{z} - \mathbf{y})^T (\mathbf{z} - \mathbf{y}) \\ &= \frac{1}{2} (\mathbf{z}^T \mathbf{z} - 2\mathbf{y}^T \mathbf{z} + \mathbf{y}^T \mathbf{y})\end{aligned}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}} = \mathbf{z} - \mathbf{y}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{x}} &= \frac{\partial \mathcal{L}}{\partial \mathbf{a}} + \frac{\partial \mathcal{L}}{\partial \mathbf{e}} \\ &= \frac{\partial \mathbf{b}}{\partial \mathbf{x}} \frac{\partial \mathcal{L}}{\partial \mathbf{b}} + \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \frac{\partial \mathcal{L}}{\partial \mathbf{f}} \\ &= \frac{\partial \mathbf{b}}{\partial \mathbf{x}} \frac{\partial \mathcal{L}}{\partial \mathbf{b}} + \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \frac{\partial \mathbf{z}}{\partial \mathbf{f}} \frac{\partial \mathcal{L}}{\partial \mathbf{z}} \\ &= \mathbf{W}_1^T \left(\mathbb{1}(\mathbf{b}) \odot (\mathbf{W}_3^T \frac{\partial \mathcal{L}}{\partial \mathbf{z}}) \right) + \mathbf{W}_2^T \frac{\partial \mathcal{L}}{\partial \mathbf{z}} \\ &= \mathbf{W}_1^T \left(\mathbb{1}(\mathbf{W_1 x}) \odot (\mathbf{W}_3^T \frac{\partial \mathcal{L}}{\partial \mathbf{z}}) \right) + \mathbf{W}_2^T \frac{\partial \mathcal{L}}{\partial \mathbf{z}}\end{aligned}$$

with

$$\frac{\partial \mathbf{z}}{\partial \mathbf{f}} = \mathbf{I}$$

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \mathbf{W}_2^T$$

5.

$$\begin{aligned}\mathcal{L} &= \frac{1}{2} ||h_3(h_2(h_1(\mathbf{x}))) - \mathbf{y}||^2 \\ h_1(\mathbf{x}) &= \text{PReLU}(\mathbf{W}_1 \mathbf{x}) \\ h_2(\mathbf{x}) &= \text{ELU}(\mathbf{W}_2 \mathbf{x}) \\ h_3(\mathbf{x}) &= \mathbf{W}_3 \mathbf{x}\end{aligned}$$

- $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{n \times n}$ and $\mathbf{W}_3 \in \mathbb{R}^{m \times n}$
- For the PReLU unit, $f(x) = \max(\alpha x, x)$.
- For the ELU unit, $f(x) = \max(\alpha(e^x - 1), x)$.

$$\begin{aligned}e &= \mathbf{W}_1 x \\ d &= ReLU(e) \\ c &= \mathbf{W}_2 d \\ b &= ELU(c) \\ a &= \mathbf{W}_3 b\end{aligned}$$

Therefore, $y = ||a||^2$.

$\frac{\partial \mathcal{L}}{\partial a} = a - y$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial W_2} &= \frac{\partial c}{\partial W_2} \cdot \frac{\partial b}{\partial c} \cdot \frac{\partial a}{\partial b} \cdot \frac{\partial \mathcal{L}}{\partial a} \\ &= \frac{\partial b}{\partial c} \cdot W_3^T \cdot (a - y) \cdot d^T\end{aligned}$$

$$\frac{\partial \mathcal{L}}{\partial W_3} = \frac{a}{\partial W_3} \cdot \frac{\partial \mathcal{L}}{\partial a} = (a - y) b^T$$

$\frac{\partial a}{\partial b} = W_3^T$

$\frac{\partial b}{\partial c} = \text{diag}(\mathbb{1}\{c_1 < 0\} \cdot \alpha e^{c_1} + \mathbb{1}\{c_1 > 0\}, ..., \mathbb{1}\{c_n < 0\} \cdot \alpha e^{c_n} + \mathbb{1}\{c_n > 0\})$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial W_1} &= \frac{\partial e}{\partial W_1} \cdot \frac{\partial d}{\partial e} \cdot \frac{\partial c}{\partial d} \cdot \frac{\partial \mathcal{L}}{\partial c} \\ &= \frac{\partial d}{\partial e} \cdot W_2^T \cdot \frac{\partial b}{\partial c} \cdot W_3^T \cdot (a - y) \cdot x^T\end{aligned}$$

$\frac{\partial c}{\partial d} = W_2^T$

$\frac{\partial d}{\partial e} = \text{diag}(\mathbb{1}\{e_1 < 0\} \cdot \alpha + \mathbb{1}\{e_1 > 0\}, ..., \mathbb{1}\{e_n < 0\} \cdot \alpha + \mathbb{1}\{e_n > 0\})$

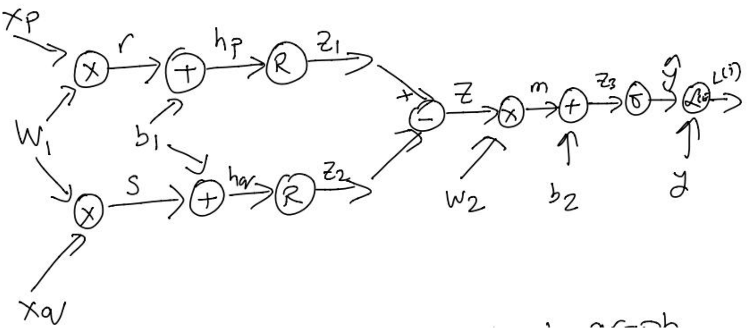
$$h_p = W_1 x_p^{(i)} + b_1$$
$$z_1 = ReLU(h_p)$$
$$h_q = W_1 x_q^{(i)} + b_1$$
$$z_2 = ReLU(h_q)$$
$$z = z_1 - z_2$$
$$z_3 = W_2 z + b_2$$
$$\hat{y}^{(i)} = \sigma(z_3)$$
$$L^{(i)} = L_{CE}(y^{(i)}, \hat{y}^{(i)})$$
$$L = -\frac{1}{m}\sum_{i=1}^{m} L^{(i)}$$

6. Lce=Cross Entropy Loss 先写下 dimension

$x_p^{(i)} \in \mathbb{R}^{D_x}$, $x_q^{(i)} \in \mathbb{R}^{D_x}$, $\hat{y}^{(i)} \in \mathbb{R}$

$W_1 \in \mathbb{R}^{D_z \times D_x}$, $b_1 \in \mathbb{R}^{D_z}$

$W_2 \in \mathbb{R}^{1 \times D_z}$, $b_2 \in \mathbb{R}$



$$L^{(i)} = L_{CE}(y^{(i)}, \hat{y}^{(i)})$$
$$= y^{(i)} \log(\hat{y}^{(i)}) + (1-y^{(i)}) \log(1-\hat{y}^{(i)})$$
$$\frac{dL^{(i)}}{d\hat{y}^{(i)}} = \frac{y^{(i)}}{\hat{y}^{(i)}} - \left\{\frac{1-y^{(i)}}{1-\hat{y}^{(i)}}\right\} = \delta_{\hat{y}^{(i)}}$$

$$\hat{y}^{(i)} = \sigma(z_3)$$

$$\sigma(z_3) = \frac{1}{1+e^{-z_3}} \quad \frac{d\hat{y}^{(i)}}{dz_3} = \sigma(z_3)[1-\sigma(z_3)]$$

$$\frac{dL^{(i)}}{dz_3} = \frac{d\hat{y}^{(i)}}{dz_3} \cdot \frac{dL^{(i)}}{d\hat{y}^{(i)}} = \underline{\sigma(z_3)[1-\sigma(z_3)]\,\delta_{\hat{y}^{(i)}}}$$

$$\frac{dL^{(i)}}{db_2} = \frac{dL}{dz_3} = \delta_{z_3} \qquad \frac{dL^{(i)}}{dW_2} = \frac{dm}{dW_2}\frac{dL^{(i)}}{dm}$$

$$\delta_{b_2} = \delta_{z_3} \qquad \frac{dm}{dW_2} \in \mathbb{R}^{1 \times D_z}$$

因为 w2 是 row vector,所以结果是 z^T $\frac{dm}{dW_2} = z^T$

$$\frac{dL^{(i)}}{dW_2} = \underline{\delta_{z_3}}\, z^T \qquad \frac{dL^{(i)}}{dz} = \frac{dm}{dz}\frac{dL^{(i)}}{dm} \qquad \frac{dm}{dz} = w_2^T$$

$$\therefore \delta_z = \delta_{z_3} w_2^T \qquad \delta_{z_1} = \delta_z$$

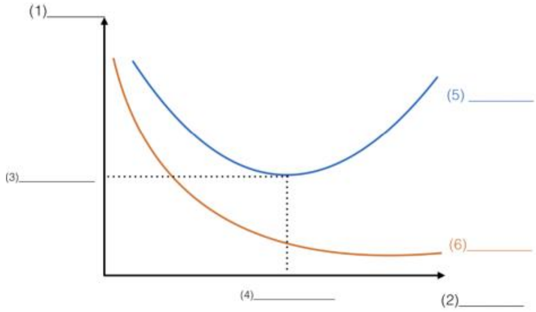
$$\frac{dL^{(i)}}{dh_q} = \frac{dz_2}{dh_q}\frac{dL^{(i)}}{dz_2} \qquad \frac{dL^{(i)}}{dr} = \delta_{h_p}$$

$$\delta_{z_2} = -\delta_z \qquad = \mathbb{I}(h_q > 0) \odot \delta_{z_2} \qquad \delta_{b_1} = \delta_{h_p} + \delta_{h_q} \qquad \frac{dL^{(i)}}{ds} = \delta_{h_q}$$

$$r = W_1 x_p$$

$$\frac{dL^{(i)}}{dW_1}\bigg|_{Path\,1} = \frac{dr}{dW_1}\frac{dL^{(i)}}{dr} = \frac{dL^{(i)}}{dr} x_p^T \qquad \frac{dL^{(i)}}{dW_1}\bigg|_{Path\,2} = \frac{ds}{dW_1}\frac{dL^{(i)}}{ds} = \frac{dL^{(i)}}{dh_q} x_q^T$$

$$\delta_{W_1} = \delta_{h_p} x_p^T + \delta_{h_q} x_q^T$$



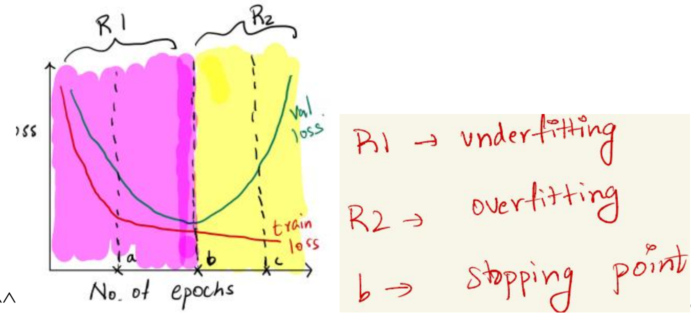
Regularization: **特征：** L1 regularization 学到的 weight 比 L2 的更 sparse. + L1 经常导致 weights 变 0 + L1 imposes a penalty = sum of model weights 的绝对值. This encourages the optimization of certain feature weights to 0, promoting simpler and sparser patterns in the model**. By pushing less important feature weights to zero, L1 aids in preventing overfitting**. Additionally, examining the trained weights allows for **feature selection**, simplifying the model and conserving computational resources.+ **Dropout** acts as regularization(防止overfitting due to a layer's "over-reliance" on a few of its inputs. 因为这些 input 不总是 present during training (i.e. they are dropped at random), the layer learns to 用所有 input, improving generalization.实现:Dropout applies a random mask to activations during training to simulate training a large ensemble of models. Each ensemble comprises units that passed through a mask, and at test time, predictions are averaged across these subnetworks. This approach approximates bagging, where multiple models are trained on different subsets of data. However, in dropout, all models share a significant portion of weights, distinguishing it slightly from traditional bagging.) **Implement**: inverted drop.train 的时候产生 mask*values after affine transform/p(prob of keep), test 的时候 nothing(用 hidden layer 所有的 unit) + 失败的 regularization attempts (such as having too large a weight on a parameter norm penalty) 会导致 model underfitting +用太多 regu 会 underfit train set 会导致 worse performance + Helpful for generalization/ test set performance 而不是 training set + optimizer 用 Adagrad 会有 equal or smaller learning rate in successive iterations.(因为 gradient history stays the same or increases in each dimension for every iteration of Adagrad.) + (不对:)Adding a regularization penalty 总是减少 training loss. ^^^why **Ensemble** 一些 NN 会增加 generalization Perform(If NN make independent errors->they 可能 to make errors on different trials. We only get trial wrong when several NN make 一样 errors. So, model 之间 correlation 更低,ensemble model accuracy 更好)^^ **Transfer Learning** 可以 freeze most parameters of the original network^^(不对)**Multitask learning** is not applicable 如果一个 task 的数据少^^**Early stopping** 是一种 regu 方法，在每个 iteration evaluate train 和 validation error loss 来返回有最低 validation error 的 model(1)Loss(2) Number of iterations (3) best validation error (4) Optimal stopping point (5) Validation loss) (6) Training loss^^^During training, training error 减少 validation error 先减再加. After training completes, the testing error 比 train 高很多, 因为 **overfitting**,可能的原因: The model is overtrained. The training data is not enough. The training data is enough but highly correlated. The model is too complex for the dataset. The data is too noisy.

NN 有高 train 准确但 validation 准确低,则用 dropout 或 dataset augmentation 用在 input 层, 增加 layer 数没用

Training, Validation, and Testing. Training set 用于 adjust parameters to fit the input. Validation set 用于选 optimal hyperparameters. Testing set 用于作为 proxy to see how well the model (the architecture and hyperparameters) would generalize to new data. The test set 只能用 1 次, after the cross-validation procedure, where each fold will be used k times for training/validation.

增加 trials 数 can allow more complicated model to be learned. 也能 against overfitting and make the model more robust.

Validation accuracy 高因为： The group was effectively training on the validation set, since the subsampled trials were highly correlated, and hence the parameters were adjusted to fit the highly correlated trials in the training set, which were effectively in the validation set as well. They should have first split the training and validation set, and then subsampled. 多次在 train set 测试会导致 overfit to the testing set,不会产生 new data ^^^



overfitting: training error keep go down,但 validation error go up,在 b 停止因为 lowest validation

L−∞ regularization

$\|x\|_\infty = \max_i |x_i|$

$LSE(x) = \ln\left(\sum_{i=1}^{n} e^{x_i}\right)$ Show that the following inequality holds for n ≥ 1 $\|x\|_\infty \le LSE(x) \le \|x\|_\infty + \ln(n)$ (4)

suppose $P = \max_i \{|x_1|, |x_2|, \dots, |x_n|\}$ lower bound: $e^P \le \sum_{i=1}^{n} e^{|x_i|}$ Upper bound $\sum_{i=1}^{n} e^{|x_i|} \le n e^P$ Combine: $e^P \le \sum_{i=1}^{n} e^{|x_i|} \le n e^P$ taking natural algorithm:

$\ln(e^P) \le \ln\left(\sum_{i=1}^{n} e^{|x_i|}\right) \le \ln(ne^P)$ Since $P = \|x\|_\infty$, so we have the required inequality

$\Rightarrow P\ln e \le \ln\left(\sum_{i=1}^{n} e^{|x_i|}\right) \le \ln(n) + P\ln(e)$

$\|x\|_\infty \le LSE(x) \le \|x\|_\infty + \ln(n)$

Is the lower bound in (4) strict for n > 1?

Clearly for $n > 1$,

$\ln(e^P) < \ln\left(\sum_{i=1}^{n} e^{|x_i|}\right)$

$e^P < \sum_{i=1}^{n} e^{|x_i|}$ Taking natural algorithm $P\ln(e) < \ln\left(\sum_{i=1}^{n} e^{|x_i|}\right)$ $\|x\|_\infty < LSE(x)$ Under what condition on x, will the upper bound in (4) be satisfied with equality?

for all $i, j \in n$

Suppose $|x_i| = |x_j|$ $\sum_{i=1}^{n} e^{|x_i|} = \sum_{i=1}^{n} e^P = ne^P$ Hence, the upper bound will be satisfied with equality $LSE(x) = \|x\|_\infty + \ln(n)$

(d)Use the result from (4) to show that the following inequality holds, $\|x\|_\infty \le \frac{1}{t} LSE(tx) \le \|x\|_\infty + \frac{\ln(n)}{t}$ for some scaling constant t >

0

Let $t > 0$ be some scaling constant. Substituting x with tx in (2), we get

$\|tx\|_\infty \le LSE(tx) \le \|tx\|_\infty + \ln(n)$

$t\|x\|_\infty \le LSE(tx) \le t\|x\|_\infty + \ln(n)$

Dividing by t, we get the required inequality,

Now,

$\|tx\|_\infty = t\|x\|_\infty$ $\|x\|_\infty \le \frac{1}{t} LSE(tx) \le \|x\|_\infty + \frac{\ln(n)}{t}$

^^^on Wi(也叫 Chebyshev norm) $\mathcal{L}_{reg}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) + R(\mathbf{W})$

$R(\mathbf{W}) = \frac{\lambda}{n} \sum_{i=1}^{n} ||\mathbf{W}_i||_\infty$. $||\mathbf{A}||_\infty = \max_{1 \le k \le p} \sum_{j=1}^{m} |A_{kj}|$ For $\mathbf{A} \in \mathbb{R}^{p \times m}$, the $L - \infty$ norm returns the maximum row sum of $\mathbf{A}$. Mathematically,

Assume gradient of |w| with respect to w is sign(w). 算 R(w)对 w 求导

Based on the dimensions of input, hidden layer and output dimensions, the individual weight matrix dimensions will be known to us. Without loss of generality, let the weight matrix of the i^{th} hidden layer $\mathbf{W}_i \in \mathbb{R}^{m \times p}$ comprising of row vectors $\mathbf{w}_r^i$ and elements represented as w_{rc}^i then $R(\mathbf{W}) = \frac{\lambda}{n} \sum_{i=1}^{n} \left(\max_{1 \le r \le m} \sum_{c=1}^{p} |w_{rc}^i| \right)$ j=index of row

$$R(\mathbf{W}) = \frac{\lambda}{n}\sum_{i=1}^{n}\sum_{c=1}^{p}|w_{jc}^i| \quad \nabla_{\mathbf{W}_i}R(\mathbf{W}) = \frac{\lambda}{n}\begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \text{sign}(w_{j1}^i) & \text{sign}(w_{j2}^i) & \cdots & \text{sign}(w_{jp}^i) \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

with maximum row-sum

$$= \frac{\lambda}{n}\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \text{sign}(\mathbf{w}_j^i) \\ \mathbf{0} \end{bmatrix}$$

^^^**加上 L2** 的 updated loss func: $\hat{J} = J + \|\mathbf{W}_2\|_2^2$. 用了 vanilla gradient descent with learning rate ϵ后 update

$$\mathbf{W}_2^{(k+1)} = \mathbf{W}_2^{(k)} - \epsilon[\nabla_{\mathbf{W}_2}J^{(k)} + 2\mathbf{W}_2^{(k)}]$$

role

Shrink size of model(加 **L1**--impose a sparsity constraint on the model. some parameters 有 optimal value 0) $\Omega(\theta) = \|\boldsymbol{W}\|_1 = \sum_{i,j}|W_{i,j}|$ .Loss func: $\tilde{J}(\boldsymbol{W}) = \alpha\|\boldsymbol{W}\|_1 + J(\boldsymbol{W})$ encourage a subset of the **weights to become zero**. During training, **dropout** make the model seem smaller 因为一些 activations are zeroed out. 但 多个 training epochs 之后, each unit is likely included in at least one subnetwork, making all units integral to the model. 所以测试时候 a model with dropout still 仍有一样数量的 parameters as a vanilla network. Additionally, weights corresponding to dropped units are not zeroed out but are skipped during backpropagation. Scaling activations during test time does not reduce model size since the number of nonzero weights remains the same. **L2 regularization** shrinks the magnitude of the weights, but the overall size of the model stays the same. Specifically, L2 does not cause parameters to

Mathematically show that ℓ_2 regularization shrinks the weight in gradient descent.
Hint: start with $\tilde{\mathcal{L}}(\theta; \mathbf{X}, \mathbf{y}) = \mathcal{L}(\theta; \mathbf{X}, \mathbf{y}) + \frac{\alpha}{2}||\theta||_2^2$ and derive the gradient descent step for θ.

be sparse.

(c) $\tilde{\mathcal{L}}(\theta; X, y) = \mathcal{L}(\theta; X, y) + \frac{\alpha}{2}\|\theta\|_2^2$

θ is a vector

$\nabla_\theta \tilde{\mathcal{L}}(\theta; X, y) = \frac{\partial}{\partial\theta}\tilde{\mathcal{L}}(\theta; X, y)$

$\frac{\alpha}{2}\|\theta\|_2^2 = \frac{\alpha}{2}\theta^T\theta$

$\frac{\partial}{\partial\theta}\frac{\alpha}{2}\theta^T\theta = \alpha\theta$

$= \frac{\partial}{\partial\theta}\left[\mathcal{L}(\theta; X, y) + \frac{\alpha}{2}\|\theta\|_2^2\right]$

$= \nabla_\theta \mathcal{L}(\theta; X, y) + \alpha\theta$

$\theta \leftarrow \theta - \epsilon\nabla_\theta\tilde{\mathcal{L}}(\theta; X, y)$

$\leftarrow \theta - \epsilon[\nabla_\theta \mathcal{L}(\theta; X, y) + \alpha\theta]$

$\leftarrow (1-\epsilon\alpha)\theta - \epsilon\nabla_\theta \mathcal{L}(\theta; X, y)$

weight decay

^^^ **Why can L2 regularization "weight decay"** 有 L2 的 loss function

$$\Omega(\theta) = \frac{1}{2}\|\boldsymbol{W}\|_2^2 = \frac{1}{2}\sum_{i,j}W_{i,j}^2$$

$$\tilde{J}(\theta) = \frac{\alpha}{2}\boldsymbol{W}^\top\boldsymbol{W} + J(\theta) \quad \boldsymbol{W} \leftarrow \boldsymbol{W} - \epsilon\left(\alpha\boldsymbol{W} + \nabla_W J(\theta)\right)$$

weights 有 decay tendency $\nabla_W\tilde{J}(\theta) = \alpha\boldsymbol{W} + \nabla_W J(\theta)$ $\leftarrow (1-\epsilon\alpha)\boldsymbol{W} - \epsilon\nabla_W J(\theta)$ By modifying the gradient update with a weight decay term (1-..), for nonzero .., the weight matrix shrinks by a constant factor on each training iteration, before applying the gradient update. Across training, this will lead to weight matrices with smaller, more diffuse entries than vanilla gradient update^^^

Batch vs Layer Normalization： batch normal 比 random weight initial 让网络更 robust,因为 Random w i 有 exploding/vanishing gradients +exacerbate internal covariate shift 问题. 对于一个合适的 batch size, random w i 不影响 Batch Normalization as strongly 因为 Batch Normal 的 output 的 mean 和 variance 被限制和被模型学习了///
Layer 和 Batch Normal 不同点: batch 的 normalization occurs on a per neuron per batch that is the mean and variance calculated is for all instances in a batch per channel. 在 layer N 里 mean 和 variance 被计算 for a single instance across all channels that is across a layer.///**用 layer 不用 batch 的情况:** As layer N normalizes the activations across an entire layer, 它比 Batch N 表现更好当 batch size 被减少. 它表现 also 更好当 modelling sequential data. Layer N is extensively used in RNNs.///**Batch 特征**： Batch N introduces noise to a hidden layer's activation,因为 mean 和 standard deviatio nare estimated with a mini-batch of data. + 加速 train by requiring fewer iterations to converge to a given loss value + 有 learnable parameter + is a linear transformation+用 running mean and variance from training statistics ^^**Batch N 把每个 artificial neuron nomalize,所以 mean=0 variance=1,这有用因为**当 gradient update for a given weight, matrix assume 其他不变. 这不对! So, gradient steps in earlier layers may dramatically change the statistics of the neurons for the next layer. Batch N normalizes these statistics, so that they aren't dramatically different after each gradient step. Batch N prevents the outputs of each layer from being 太大/太小.^^ **Batch N 怎么帮助 NN more robust to initialization**: (BN) normalizes the outputs at each layer, preventing activations from decaying to zero or exploding to infinity when cascading through multiple layers. This robustness to vanishing or exploding gradients during backpropagation is crucial for effective training. Vanishing gradients lead to information loss, making it difficult to propagate gradients upstream, while exploding gradients cause training instability or overflow errors due to excessively large steps in the loss

$$\text{mean} \rightarrow \mu_i = \frac{1}{m}\sum_{j=1}^{m} x_i^{(j)}$$

$$\text{variance} \rightarrow \sigma_i^2 = \frac{1}{m}\sum_{j=1}^{m} (x_i^{(j)} - \mu_i)^2$$

landscape.**实现**： **In Training**,会有 m(batchSize)*n(Num Unit),步骤

$$\hat{x}_i^{(j)} = \frac{x_i^{(j)} - \mu_i}{\sqrt{\sigma_i^2 + \varepsilon}} \qquad y_i^{(j)} = \underbrace{\gamma_i \hat{x}_i^{(j)} + \beta_i}_{\text{trainable}}$$

两个 trainable 参数(scaling 和 shifting).主要是算 minibatch 的 mean 和 variance,再 BatchNorm normalizes the layer's input based on the mean and variance 得到 average mean 和 variance. 用 momentum 更新 running average mean 和 vaiance,

$$\bar{\mu} \leftarrow \bar{\mu} \cdot \text{momentum} + (1-\text{mome})\,\mu$$

$$\bar{\sigma}^2 \leftarrow \bar{\sigma}^2 \cdot \text{mome} + (1-\text{mome})\,\bar{\sigma}^2$$

.用两个参数(learnable during training)scaling 和 shifting 决定 optimal distribution.也会 track avg mean 和 variance. **In Testing**,用 avgs 去 normalize test data,再把γ和β apply 到 nomalized data.

optimization: Nesterov Momentum, update rule

$$v_t = \alpha v_{t-1} - \epsilon \nabla_\theta L(\theta_{t-1} + \alpha v_{t-1})$$
$$\theta_t = \theta_{t-1} + v_t \qquad (1)$$

$$v_{new} = \alpha v_{old} - \epsilon \nabla_{\tilde{\theta}_{old}} L(\tilde{\theta}_{old})$$
$$\tilde{\theta}_{new} = \tilde{\theta}_{old} + v_{new} + \alpha(v_{new} - v_{old})$$

(2)one advantage of using the update update rule in (2) over the update rule in (1)

From ①, we have:

$$v_{new} = \alpha v_{old} - \epsilon \nabla_\theta L(\theta_{old} + \alpha v_{old})$$

$$\nabla_{\tilde{\theta}} L(\tilde{\theta}_{old}) = \frac{\partial L(\tilde{\theta}_{old})}{\partial \tilde{\theta}} = \frac{\partial \theta}{\partial \tilde{\theta}} \cdot \frac{\partial L(\tilde{\theta}_{old})}{\partial \theta}$$

$$= \cancel{\frac{\partial \theta}{\partial \tilde{\theta}}}^{\,I} \cdot \frac{\partial L(\theta_{old} + \alpha v_{old})}{\partial \theta} = \nabla_\theta L(\theta_{old} + \alpha v_{old})$$

⇒ Let us assume that new parameter $\tilde{\theta}$

$$\tilde{\theta} = \theta + \alpha v$$

This is : $\tilde{\theta}_{old} = \theta_{old} + \alpha v_{old}$ | $\theta_{old} = \tilde{\theta}_{old} - \alpha v_{old}$

$\tilde{\theta}_{new} = \theta_{new} + \alpha v_{new}$ | $\theta_{new} = \tilde{\theta}_{new} - \alpha v_{new}$

$$\left[\because \frac{\partial \theta}{\partial \tilde{\theta}} = \frac{\partial}{\partial \tilde{\theta}}(\tilde{\theta} + \alpha v) = I + 0 = I \right]$$

From ①, we have: $\theta_{new} = \theta_{old} + v_{new}$

So, we have, $\nabla_{\tilde{\theta}} L(\tilde{\theta}_{old}) = \nabla_{\theta} L(\theta_{old} + \alpha v_{old})$ $\quad \tilde{\theta}_{old} - \alpha v_{new} = \theta_{old} - \alpha v_{old} + v_{new}$

$v_{new} = \alpha v_{old} - \epsilon \nabla_{\tilde{\theta}} L(\tilde{\theta}_{old}) \longrightarrow$ This is ② equation $\quad \tilde{\theta}_{old} = \theta_{old} + v_{new} + \alpha(v_{new} - v_{old}) \longrightarrow$ This is ② equation

Hence, we have shown that both ① & ② are equivalent as we can get ② from ① by a change in variable.

This representation helps us in implementation of nestorov momentum as this doesn't require us to calculate gradient at a different value of $\theta + \alpha v$.

Also, both θ and $\tilde{\theta}$ start from the same value of parameter initialization as $\theta = \tilde{\theta}$ initially as $v = 0$ at start

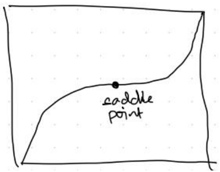
L1, L2 saddle point

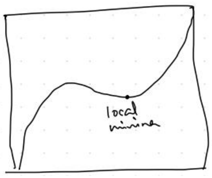
$L_1(x)$ has a saddle point. The curve decreases on one side of the saddle and increases on other side of saddle point. The gradient at a saddle point is equal to zero.

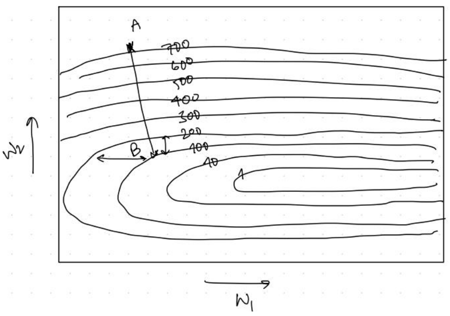
$L_2(x)$ has a poor local minima. This is a local minima as it is a minimum value of the function in it's surrounding until a certain limit in all the direction. The gradient is zero at a local minima.

We need momentum to get out of saddle point or local minima because the gradient becomes zero at these points.

It is also more likely to escape the saddle point than the local minima because after crossing the local minima, the momentum decreases due to gradient being in opposite direction to the momentum accumulated down the slope.

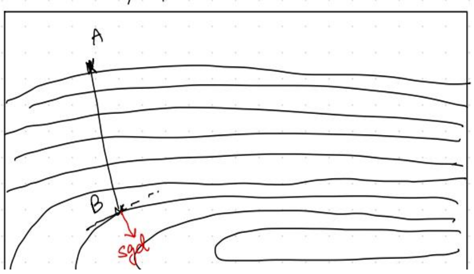




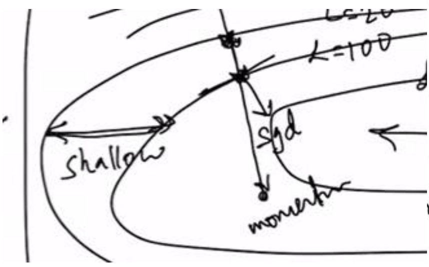


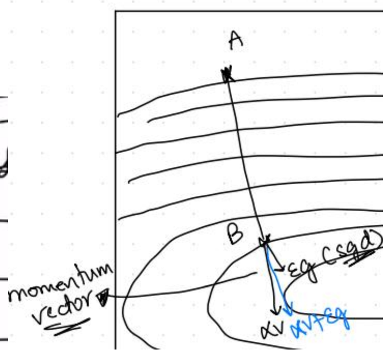
(1) The gradient along w_2 direction is higher than the gradient along w_1 direction. This is because the contour lines along w_2 directions are very close to each other which indicates a steep curve in w_2 direction.
The contour lines along w_1 direction are further apart and hence has slower descent. (or low gradient)

Vanilla gradient descent is in the direction perpendicular to a contour line

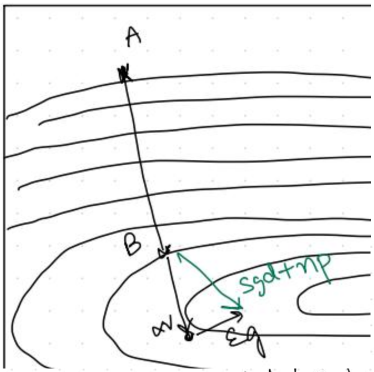


(3) sgd + momentum





sgd+ nesterov momentum.



In this, first we take a step along the momentum and then calculate the gradient at that point in graph.

Adagrad.

The update equation:

$$a \leftarrow a + g \odot g \quad [\text{accumulated gradient}]$$

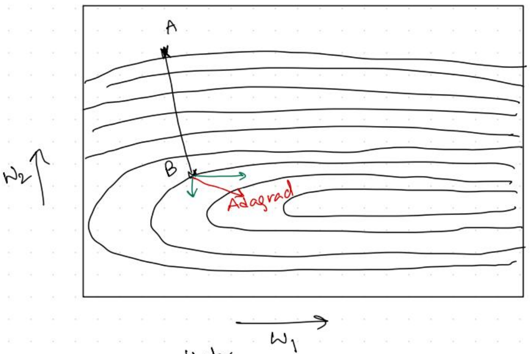
$$\theta \leftarrow \theta + \frac{\epsilon}{\sqrt{a+v}} \odot g$$

If gradient is accumulated more in a direction, then the step $\frac{\epsilon}{\sqrt{a+v}}$ is less in that direction.

If gradient accumulated is less in a direction, then the step $\frac{\epsilon}{\sqrt{a+v}}$ is more in that direction

From plot, we can see that when it traversed from A to B, the gradients accumulated along w_2 direction is more and gradient along w_1 direction is lesser than it.

Hence, the step along w_1 direction is more and step along w_2 direction is less.



Also, in this particular case we can see that Adagrad is converging faster to minima without much zigzagging.

with Goodnotes

In the Gradient descent + momentum scheme, find a general expression of v_t in terms of gradients $g_1, g_2, ..., g_t$ and ϵ(learning rate), considering an initial value of momentum $v_0 = 0$.

sgd + momentum update:

$$V_t = \alpha V_{t-1} - \varepsilon g_t$$

$$\theta_t = \theta_{t-1} + V_t$$

$\rightarrow V_0 = 0$ [Given]

$$V_1 = \alpha V_0 - \varepsilon g_1 = -\epsilon g_1$$

$$V_2 = \alpha V_1 - \varepsilon g_2 = \alpha(-\varepsilon g_1) - \varepsilon g_2 = -\varepsilon(g_2 + \alpha g_1)$$

$$V_3 = \alpha V_2 - \varepsilon g_3 = \alpha(-\varepsilon(g_2 + \alpha g_1)) - \varepsilon g_3$$

$$= -\varepsilon(g_3 + \alpha g_2 + \alpha^2 g_1)$$

From this, we can observe that

$$\Rightarrow V_t = -\varepsilon(g_t + \alpha g_{t-1} + \alpha^2 g_{t-2} + \dots + \alpha^{t-1} g_1)$$

$$\mathbb{E}(V_t) = \mathbb{E}\left[-\varepsilon(g_t + \alpha g_{t-1} + \alpha^2 g_{t-2} + \dots + \alpha^{t-1} g_1)\right]$$

$$= -\varepsilon\left[\mathbb{E}(g_t) + \alpha \mathbb{E}(g_{t-1}) + \dots + \alpha^{t-1}\mathbb{E}(g_1)\right]$$

$\theta_0 = \theta_0$
$\theta_1 = \theta_0 + v_1$
$\theta_2 = \theta_1 + v_2 = \theta_0 + v_1 + v_2$
$\theta_3 = \theta_2 + v_3 = \theta_0 + v_1 + v_2 + v_3$
$\vdots$
So, for general $\theta_t = \theta_0 + v_1 + v_2 + \dots + v_t$

Consider that the gradients $g_1, g_2, ..., g_t$ in part (d) are i.i.d. random variables with mean μ and variance σ. Find the expected value of weights θ_t at t = 3.

$$\mathbb{E}(\theta_3) = \mathbb{E}[\theta_0 + v_1 + v_2 + v_3]$$
$$= \mathbb{E}(\theta_0) + \mathbb{E}(v_1) + \mathbb{E}(v_2) + \mathbb{E}(v_3)$$

Given, $g_1, g_2 \dots, g_t$ are i.i.d r.vs with mean μ and var σ^2.
From part (d), we saw

$$\mathbb{E}(v_1) = \mathbb{E}(-\varepsilon g_1) = -\varepsilon \mathbb{E}(g_1) = -\varepsilon\mu$$
$$\mathbb{E}(v_2) = \mathbb{E}(-\varepsilon(g_2 + \alpha g_1)) = -\varepsilon[\mu + \alpha\mu]$$
$$= -\varepsilon\mu[1+\alpha]$$
$$\mathbb{E}[v_3] = \mathbb{E}(-\varepsilon(g_3 + \alpha g_2 + \alpha^2 g_1))$$
$$= -\varepsilon[\mu + \alpha\mu + \alpha^2\mu]$$
$$= -\varepsilon\mu[1+\alpha+\alpha^2]$$

Alternately, $\mathbb{E}(v_t) = -\varepsilon[\mu + \alpha\mu + \dots + \alpha^{t-1}\mu]$
$$= -\varepsilon\mu[1+\alpha+\dots+\alpha^{t-1}]$$
$$= -\varepsilon\mu\left[\frac{1-\alpha^t}{1-\alpha}\right]$$

$$\mathbb{E}(\theta_3) = \mathbb{E}(\theta_0) + \sum_{t=1}^{3}\left(-\varepsilon\mu\left[\frac{1-\alpha^t}{1-\alpha}\right]\right)$$
$$= \mathbb{E}(\theta_0) - \frac{\varepsilon\mu}{1-\alpha}\left[\sum_{t=1}^{3}(1-\alpha^t)\right]$$
$$= \mathbb{E}(\theta_0) - \frac{\varepsilon\mu}{1-\alpha}\left[3 - \sum_{t=1}^{3}\alpha^t\right]$$
$$= \mathbb{E}(\theta_0) - \frac{\varepsilon\mu}{1-\alpha}\left[3 - \left(\frac{1-\alpha^3}{1-\alpha}\right)\right]$$

$$\mathbb{E}(\theta_3) = \mathbb{E}(\theta) - \varepsilon\mu[1 + (1+\alpha) + (1+\alpha+\alpha^2)]$$

We know that sum of 'n' terms in geometric series with factor 'r' and first term a is
$$a + ar + \dots + ar^{n-1} = \frac{a(1-r^n)}{1-r}$$

$$\mathbb{E}(\theta_3) = \mathbb{E}(\theta) - \varepsilon\mu[1 + (1+\alpha) + (1+\alpha+\alpha^2)]$$

We know that sum of 'n' terms in geometric series with factor 'r' and first term a is
$$a + ar + \dots + ar^{n-1} = \frac{a(1-r^n)}{1-r}$$

K-nearest 特征 increasing k 会导致 smoother decision boundary+减小 noise or outliers 对 data 的影响+可用 cross-validation 选 k 值+不会增加 overfit 概率+要选能最小化 validaiton error 的 k. ^^分类器对数据 converge 之后 test error 高,train error 靠近 0->overfit,可以通过增加 data size+减少 model complexity 实现+对 train+test 数据集训练然后 test 数据集测试—**没用. ^^^用 k-nearest 分类 1024 *1024 pixels image 不行因为:** 1. Test is computationally expensive. 2. Distance measurements in a high dimensional space do not necessarily reflect similarity between two data points due to the curse of dimensionality).^^^Train 和 test 来自同个 distribution,用 **k-nearest** classifier,减少 test 的分类错误不能通过选择减少 train 错误的 k,而是用 validaiton tech 比如 cross-validation 找到 optimal k,k 小导致高 variance 和 overfitting,反之.通过把数据分 train 和 validation sets 选能最小化 validation error(比最小化 train error 有更好的 generalization,也就是更能 perform 在 unseen data)的 hyperparameters 来优化 **hyperparameters.** ^^^
Classification vs Regression task: Classification 用于当 target 是 categorical, regression 被用当 target 是 continuous. 他们俩都是 supervised machine learning algorithms. **Classification 例子:**预测 yes or no + gender+Type of color. **Regression 问题例子**: 估计产品的 sale 和 prive+预测球队分数^^^
Stochastic gradient descent (SGD) VS full-batch GD (GD): 两个算法都是找 parameter that 最小化 loss func by evaluating parameters against data and then making adjustments **SGD 特征 vs Batch:** SGD 算的 gradient 会 noisier 比 Batch Gradient Descent 算+ SGD avoids trap of poor local minima(因为就 1 个 data point),converge to flat local minimum + SGD take more steps 但 converge faster + Batch GC 要 load entire data(mini-batch 用 some data, SG 用 1 data),所以 computationally more expensive

^^^用 SGDwith 正确算法,但 loss 一直不变,说明 learning rate 太小要增大^^^

Initial small weight: weights 小,activation magnitude 下降 with each successive layer.last layer's activations 接近 0. Z 这导致 backpropagated gradients 小^^^**Initial param same value**: Breaking symmetry is intention for the random initilization. If initialize all weights same value (e.g. 1), each hidden unit will get same signal. E.g. all weights initialized to 1, each unit gets signal equal to sum of inputs (and outputs sigmoid (sum(inputs))). The gradient of all neurons in one layer 会一样. 这导致 train 只会有效更新 neuron in different layers. .^^^FC network 用 tanh, no bias, only weights: **Initial weights to 0** result in 0 gradients + no learning. **Initial weights 大** 导致 vanishing gradients**. Vanishing gradients lead to information loss, making it difficult to propagate gradients upstream, while exploding gradients cause training instability or overflow errors due to excessively large steps in the loss landscape.**

(不对) Initial weights 一样值导致 gradient descent 后 weights 还 equal

Softmax classifier $\mathcal{L} = \sum_{i=1}^{m}\left(\log\sum_{j=1}^{c} e^{a_j(x^{(i)})} - a_{y^{(i)}}(x^{(i)})\right)$ gradient:

$$\frac{\partial \mathcal{L}}{\partial w_k} = \sum_{i=1}^{m}\left(\frac{e^{a_k(x^{(i)})}}{\sum_{j=1}^{c} e^{a_j(x^{(i)})}} x^{(i)} - \mathbb{1}(k = y^{(i)})\cdot x^{(i)}\right)$$

$$\frac{\partial \mathcal{L}}{\partial b_k} = \sum_{i=1}^{m}\left(\frac{e^{a_k(x^{(i)})}}{\sum_{j=1}^{c} e^{a_j^{(x^{(i)})}}} - \mathbb{1}(k = y^{(i)})\right)$$

MTR:

Consider a model $\tilde{\mathcal{L}}(\theta) = \mathcal{L}(\theta; \mathbf{X}, \mathbf{y}) + \alpha\Omega(\theta)$ where $\mathcal{L}(\theta; \mathbf{X}, \mathbf{y})$ is some loss function and $\Omega(\theta)$ is some norm penalty. What are the effects on the model when $\alpha = 0$ and $\alpha \to \infty$?

loss fn. | norm penalty

$\tilde{\mathcal{L}}(\theta) = \mathcal{L}(\theta; \underline{X}, y) + \alpha\Omega(\theta)$

$\alpha = 0$? $\tilde{\mathcal{L}}(\theta) = \mathcal{L}(\theta; \underline{X}, y)$

$\alpha \to \infty$? $\tilde{\mathcal{L}}(\theta) \to \alpha\Omega(\theta)$

no learning of $\mathcal{L}(\theta; \underline{X}, y)$

=0 只有 loss func,model 更可能 overfit trainning data,=∞ no learning.^^^当 minimize the negative log likelihood for a classification problem with c class,等于去做:Maximizing the likelihood of observing the training data. = Minimizing the Cross Entropy loss 不等于 Minimizing the Mean Squared Error.^^^ A 5-fold **cross-validation** fit 5-different model + **(不对)**A 5-fold cross-validation 导致 1 model instance being fitted over and over again 5 times + **(不对)**A 5-fold cross-validation 导致 5-different model instances being fitted over and over again 5 times.1fold 作为 validation, k-1 个 fold 作为 train+test,重复决定哪些 folds 作为 train+test, **前提**是有一个 Separate testing set,training set split into train 和 validation data///**Never do validation on test set.**

Hyperparameter(任何在 GC 没有被 learnt 的东西):Loss Function+ Learning Rate+ Number of Layers+ Batch Size.

Multi-layer Perceptron (MLP) increasing the number of units within each layer can enhance the model's complexity: (i)如果 model **underfitting** on the training data, 加更多 units allows the MLP to capture more complex patterns, 可能降低 training and testing errors. (ii)但这可能也会导致 **overfitting**, as the increased model capacity can make it overly sensitive to the training data. Consequently, while the training loss may continue to decrease, the testing error could increase

Role of the bias correction step in the Adam optimizer: The Adam optimizer utilizes running averages to estimate the gradient and its square, which are initially biased towards 0 due to their initialization. As a result, the optimizer may take larger steps in the early stages of training, leading to unstable training and slower convergence. To address this, a bias correction step is introduced to adjust these estimations, making them more accurate. As training progresses and the estimations become more precise, the bias correction factor gradually approaches 1, diminishing the impact of bias correction.