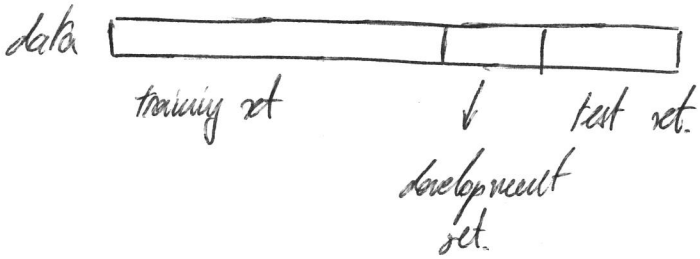
Hyperparameter training, regularization and optimization

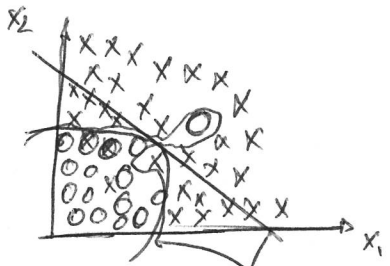
* Train, dev, test →



development set = those which model performs best.

98% / 1% / 1%.

* Bias and variance →



balanced

overfitted → high variance

underfitted.

↓

high bias.

Training set	1%	15%	15%	0.5%
Dev set	11%	16%	30%	1%
		(2)		
	↓	↓	↓	↓
	Overfitting high variance	high bias underfitting	both.	low bias and variance.

* This is based on optimal error ≥ 0%, but if optimal error is 15% (2) is a very good model.

A. High variance →

1. % train error is large compare to optimal.

2. Improve →

A. Bigger Network.

B. Train longer

C. NN arch ref.

B. High variance ⟶ large % Error dev compared to training.

1. Improvements ⟶

A. More data.

B. Regularization.

C. NN arch ret.

* Regularization ⟶

- combat overfitting

$$J(w,b) = \frac{1}{M}\sum_{i=1}^{M} L(\hat{y}(i), y(i)) + \frac{\lambda}{2M}\|w\|_2^2$$

l_2 regularization ⟶ $\|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$

l_1 regularization ⟶ $\frac{1}{M}\sum_{i=1}^{n_x} |w| = \frac{1}{M}\|w\|_1$

$\lambda \equiv$ Regularization parameter.

- Neural Network ⟶

$$J(w[1], b[1], \ldots, w[L], b[L]) = \frac{1}{M}\sum_{i=1}^{M} L(\hat{y}(i), y(i)) + \frac{\lambda}{2M}\sum_{l=1}^{L} \|w[l]\|_2^2$$

$$\|w[l]\|_F^2 = \|w[l]\|_2^2 = \sum_{i=1}^{n[l-1]} \sum_{j=1}^{n[l]} (w_{ij}[l])^2 \quad ; \quad w : (n[l], n[l-1])$$

* gradient descent ⟶ $-\alpha\, dw$

$$w[l] = w[l] - \frac{\alpha\lambda}{M} w[l] - \alpha[\text{back propagation}]$$

$\frac{\alpha\lambda}{M} w[l]$ ⟶ weight decode.

* Vanishing / Exploding gradients →

- In very deep networks → slopes can get really high or low.
- Activation values → increase / decrease exponentially.

* Weight initialization for deep Networks →

- Need to balance the $z = w_1 x_1 + \dots + w_n x_n$ so the values are not very high, low. → control weight with variance.

$$Var(w_i) = \frac{1}{n} \rightarrow w[l] = \text{Random matrix} * np.sqrt\left(\frac{1}{n[l-1]}\right)$$

- ReLU → $\sqrt{\frac{2}{n[l-1]}}$
- Tanh → $\sqrt{\frac{1}{n[l-1]}}$
- Other probably → $\sqrt{\frac{2}{n[l] * n[l-1]}}$

* Gradient checking →

$w[1], b[1], \dots, w[L], b[L] \rightarrow \theta \Rightarrow J(w[1], b[1], \dots, w[L], b[L]) = J(\theta)$

$J(\theta_1, \theta_2, \dots, \theta_L) \Rightarrow$

- For each layer →

$$d\theta_{approx}[i] = \frac{J(\theta_1, \theta_2, \dots, \theta_i + \varepsilon, \dots) - J(\theta_1, \theta_2, \dots, \theta_i - \varepsilon, \dots)}{2\varepsilon}$$

$$\approx d\theta[i] = \frac{\partial J}{\partial \theta_i}$$

- Check if $d\theta_{approx}[i] \stackrel{?}{\approx} d\theta[i]$? → Euclidean distance

$$\frac{\| d\theta_{approx} - d\theta \|_2}{\| d\theta_{approx} \|_2 + \| d\theta \|_2} \approx \varepsilon = 10^{-7}$$

1. Do not use in training → overhead reading.

2. If it fails → check components on i → dO(i), dOapprox(i)

3. Keep in mind regularization term →

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^{L} \| W^{[l]} \|_F^2$$

4. Doesn't work with dropout.

5. Run at random initialization → to check that it not only works for $w, b \geq 0$.

* Optimization Algorithms →

- Mini-batch gradient descent →

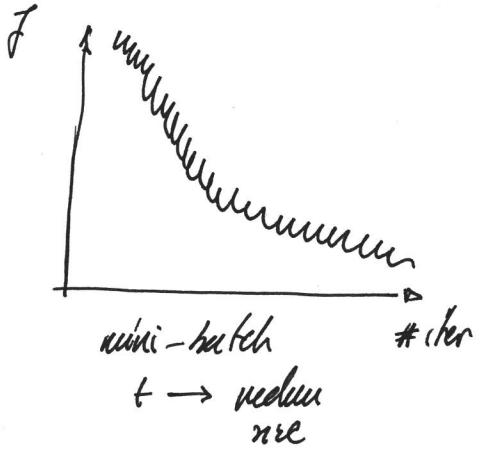
$X = [\underbrace{X^{(1)}\ X^{(2)}}_{X^{\{1\}}} \ldots \underbrace{X^{(m)}}_{X^{\{m/t\}}}]$

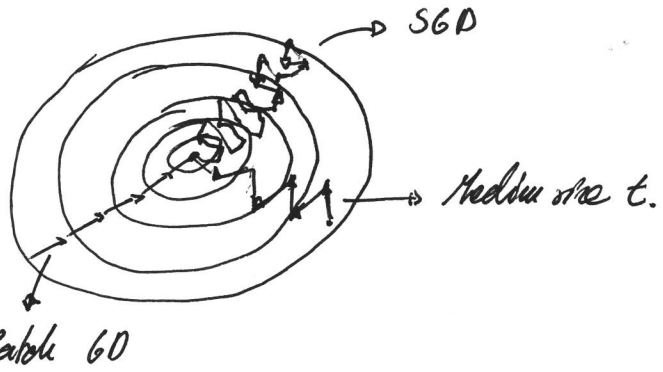
t-size

* 1 epoch ≡ Pass through training set m.

t = 1 → Stochastic GD ≡ Loses the advantage of vectorization.

t = m → Batch GD ≡ Gradient descent not updated until m.





* Mini-batches size → $2^{\wedge}x$

* the λ will control weight for each layer, making them fall within small values of z; $a = g(z) = \tanh(z) \longrightarrow$ linear region.

* λ = hyperparameter to be set; takes all <u>medium quadratic</u> values. A large value will control the gradient for weight update $\longrightarrow$ w/

$$J(W[l], b[l]) = \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}(\hat{y}(i), y(i)) + \frac{\lambda}{2M} \sum_{l=1}^{L} \|W\|_F^2$$

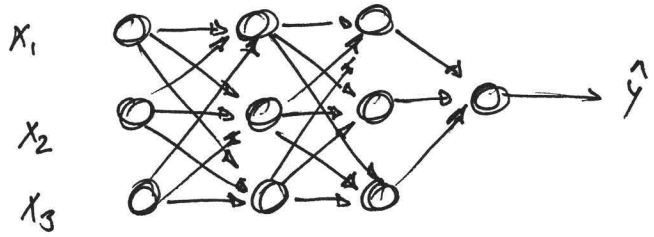
$$\|W_F\|^2 = \sum_{i=1}^{n[l-1]} \sum_{j=1}^{n[l]} (W_{ij}[l])^2$$

* large weights give too much importance to certain feature, we are looking for general features, not to stand out outliers.

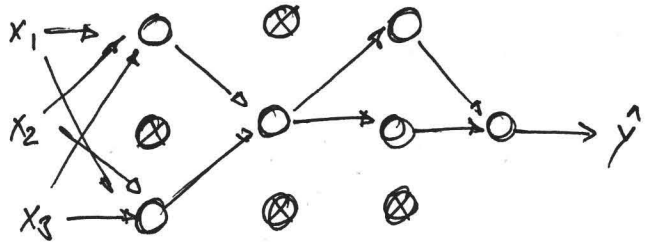
$$W[l] = W[l]\left(1 - \alpha \frac{\lambda}{M}\right) - \alpha [\text{backpropagation}].$$

$\underbrace{\qquad\qquad}_{0.X}$

* <u>Dropout</u> $\longrightarrow$



Each iteration, network changes.



drop out probability.

- During training each neuron has a dropout probability on the network

- Only done in training; not in test.

* Daniel training cost function will change with each iteration.

* Implementation →

- Forward propagation →

each layer [l]

dl = np.random.rand(al.shape[0], al.shape[1]) < keep_prob.

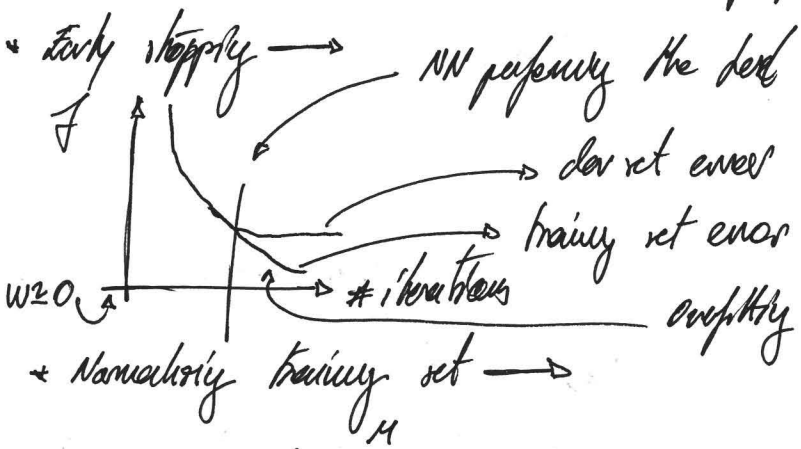
al = np.multiply(al, dl)

al = al / keep_prob → since some neurons have been eliminated from network, give more importance to other values → balance outputs.

- Backward propagation →

dA[l] = dA[l] / keep_prob.

* Early stopping →



- Takes out the fact that the next are orthogonal.

1. Optimize cost function J
2. Not overfitting.

* Normalizing training set →

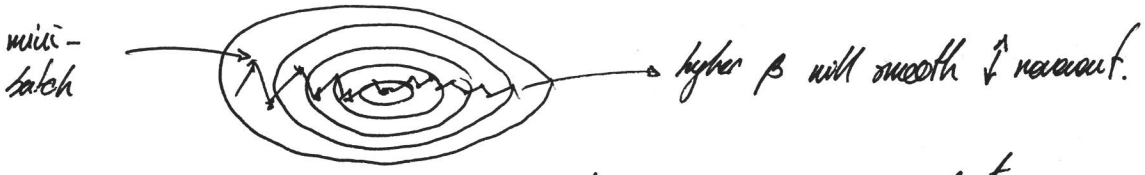
$$\mu = \frac{1}{\mu} \sum_{i=1}^{\mu} x(i) \longrightarrow x = x - \mu$$

$$\sigma^2 = \frac{1}{\mu} \sum_{i=1}^{\mu} x(i) \longrightarrow x = x / \sigma^2$$

- Centers and distributes points based on variance →

makes it faster to reduce cost function

- Gradient descent with momentum ⟶



* Idea ⟶ average out vertical axis, and move faster on axis.

$\beta \geq 0.9$ default.

$$V_{dw} = \beta V_{dw} + (1-\beta)\, dw$$

$$V_{db} = \beta V_{db} + (1-\beta)\, db$$

$$V_{dw}^{corrected} = V_{dw} / (1-\beta^t)$$

$$V_{db}^{corrected} = V_{db} / (1-\beta^t)$$

$\Longrightarrow$

$$w = w - \alpha V_{dw}^{corrected}$$

$$b = b - \alpha V_{db}^{corrected}$$

- RMSprop ⟶

* Same idea ⟶ slowdown b; speed-up w (w small)

$$S_{dw} = \beta S_{dw} + (1-\beta)\, dw^2 \quad ; \; dw^2 \equiv \text{element wise.}$$

$$S_{db} = \beta S_{db} + (1-\beta)\, db^2 \quad \text{(b large)}$$

$$w = w - \alpha \frac{dw}{\sqrt{S_{dw}}}$$

$$b = b - \alpha \frac{db}{\sqrt{S_{db}}}$$

- ADAM optimization algorithm →

  * ADAM ≡ Adaptative Moment Estimation.

  * Momentum + RMSprop

$V_{dw} = 0 \; ; \quad S_{dw} = 0 \; ; \quad V_{db} = 0 \; ; \quad S_{db} = 0$

On every iteration t (mini-batch) →

$$V_{dw} = \beta_1 V_{dw} + (1-\beta_1)\, dw$$

$$V_{db} = \beta_1 V_{db} + (1-\beta_1)\, db$$

$$V_{dw}^{corrected} = V_{dw} / (1-\beta_1^t)$$

$$V_{db}^{corrected} = V_{db} / (1-\beta_1^t)$$

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2)\, dw^2$$

$$S_{db} = \beta_2 S_{db} + (1-\beta_2)\, db^2$$

$$S_{dw}^{corrected} = S_{dw} / (1-\beta_2^t)$$

$$S_{db}^{corrected} = S_{db} / (1-\beta_2^t)$$

$$w = w - \alpha \left(\frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected}} + \varepsilon} \right)$$

$$b = b - \alpha \left(\frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}} + \varepsilon} \right)$$

default parameters → $\beta_1 = 0.9 \; ; \quad \beta_2 = 0.999 \; ; \quad \varepsilon = 10^{-8}$

- learning rate decay →

* Homelown learning decay with time → variation when you reach zone of minimum cost in small.

$|X1(t)|\ X2(t)|\ ..\ |X_{m/t}(t)|$ → epoch $t \equiv$ batch size.

* Implementation →

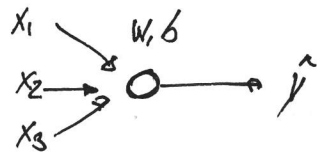
$$\alpha = \frac{1}{1 + \text{decay\_rate} * \text{epoch\#}}\ \alpha_0$$

$$\alpha = 0.95^{\text{epoch\#}}\ \alpha_0$$

$$\alpha = \frac{K}{\sqrt{\text{epoch\#}}}\ \alpha_0 \quad \text{or} \quad \alpha = \frac{K}{\sqrt{t}}\ \alpha_0$$

* Hyperparameter tuning, ~~ha~~ →

- batch normalization →



$\mu = \frac{1}{M} \sum_i x(i)$; over all samples ; $\sigma^2 = \frac{1}{M} \sum_i x(i)^2 \equiv$ elem. wise.

$X = X - \mu$; $X = X / \sigma^2$

* layer 2 →

$$\mu = \frac{1}{M} \sum_i z(i)\ ;\ \sigma^2 = \frac{1}{M} \sum_i (z_i - \mu)^2\ ;\ z_{norm}^{(i)} = \frac{z(i) - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$ ← Avoid x/0

$\hat{z}_{norm}(i) = z_{norm}(i) \cdot \gamma + \beta$ ← learnable param on model.

* γ, β allows you to move the distribution

* After iteration → $d\beta[l]$ → $\beta[l] = \beta[l] - \alpha\, d\beta[l]$

* Normalizing with mini-batches →

$x^{(k)} \longrightarrow z^{[l]} \xrightarrow{BN} \hat{z}^{[l]} \longrightarrow g^{[l]}(\hat{z}^{[l]})$

$w^{[l]}, b^{[l]}$ $\quad \beta^{[l]}, \gamma^{[l]}$

→ Normalize, cancel out.

$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$

$z^{[l]} = w^{[l]} a^{[l-1]}$

$z^{[l]}_{norm} \longrightarrow \hat{z}^{[l]} = \gamma^{[l]} z_{norm}^{[l]} + \beta^{[l]}$; $\underline{\gamma, \beta \in (n^{[l]}, 1)}$

↑ Important

$dw^{[l]}, d\beta^{[l]}, d\gamma^{[l]}$

update params.

* Batch →

1. Can be implemented using Momentum, RMSprop, Adam.
2. Batch norm reduces the variance of activation function in deeper networks → as you center on distribute data. → more stable.

* Regularization effect →

1. Using mini-batches adds some noise, depending on size.
2. Variation between μ, σ^2.
3. Larger size, less noise.

* Test time →

- the weighted average across mini-batches

$x^{\{1\}}, x^{\{2\}}, \ldots, x^{\{t\}} \longrightarrow \mu^{[l]\{1\}}, \sigma^{[l]\{1\}}, \ldots, \mu^{[l]\{t\}}, \sigma^{[l]\{t\}}$

↳ $z_{norm} = \dfrac{z - \mu}{\sqrt{\sigma^2 + \varepsilon}} \longrightarrow \hat{z} = \gamma z_{norm} + \beta$

- Softmax ⟶

$z = W A_{prev} + B$

$t = e^z$ ⟶ $a_i[l] = \frac{t_i[l]}{\sum_{j=1}^{\#} t_i}$; $A[l] = \frac{t[l]}{\sum_{j=1}^{\#C} t_j[l]}$

* Distributes probabilities between 0,1 for all possible classes in vector.

* Numerical stability ⟶

1. Exponentials can explode values. ⟶ $\log C = -\max(f_{yi})$
2. Numerical unstability.

Shifts values to highest deriv 0.

$$A = \frac{e^{f_{yi} + \log C}}{\sum_{j=1}^{\#C} e^{f_{yi}} + \log C}$$

* The cross-entropy ⟶

$$L(y, \hat{y}) = -\sum_{j=1}^{n_y} y_i \log \hat{y}_i \; ; \; J(W, B) = \frac{1}{M} \sum_{i=1}^{M} L(\hat{y}^{(i)}, y^{(i)})$$

$$\frac{\partial J(y, \hat{y})}{\partial z[l]} = \hat{y} - y$$

- Batch normalization points ⟶

1. Improves gradient flow through network.
2. Allows higher learning rates.
3. Reduces dependency on initializations.
4. Acts as a form of regularization.

Appendix

- Hinge loss function ⟶

  * the for multiclass support vector machine SVM.

  $S \equiv$ score function of (x_i, W)

  $$L = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

  * Expression that sums up all incorrect classes $j \neq y_i$, keeping only larger than one.

  * SVM wants the score of the correct class y_i to be larger than incorrect classes by at least Δ.

  * Squared hinge loss ⟶ $\max^2(0, \ldots)$ L2-SVM

- Cross entropy function ⟶

  $$H(p, q) = -\sum_x p(x) \log q(x)$$

  $p \equiv$ true distribution.

  $q \equiv$ estimated distribution.

```
def cross_entropy(X, Y):
    X ⟶ m, nx
    Y ⟶ m, 1
    m = Y.shape[0]
    p = softmax(X)
    log_likelihood = -np.log(p[range(m), Y])
    loss = np.sum(log_likelihood)/m
```

* leftmax derivative ⟶

$$A_{a,b} = \frac{e^{z_{ab}}}{\sum_{k=1}^{n_K} e^{z_{kb}}} \quad ; \quad h(x) = f(x)/g(x) \longrightarrow h'(x) = \frac{f'(x)g(x) - f(x)g'(x)}{g^2(x)}$$

$$\frac{\partial A_{ab}}{\partial z_{ij}} = \frac{(e^{z_{ij}})' \sum_{k=1}^{n_K} e^{z_{kj}} - e^{z_{ij}} \left(\sum_{k=1}^{n_K} e^{z_{kj}}\right)'}{\left(\sum_{k=1}^{n_K} e^{z_{kj}}\right)^2} = [\text{if } j \neq b \text{ then } 0.] =$$

$$\frac{\partial A_{ab}}{\partial z_{ib}} = \frac{(e^{z_{ab}})' \sum_{k=1}^{n_K} e^{z_{kb}} - e^{z_{ib}} \left(\sum_{k=1}^{n_K} e^{z_{kb}}\right)'}{\left(\sum_{k=1}^{n_K} e^{z_{kb}}\right)^2} =$$

$$\Rightarrow [i=a] = \frac{e^{z_{ab}}}{\left(\sum_{k=1} e^{z_{kb}}\right)^2} - \frac{e^{z_{ab}} \cdot e^{z_{ab}}}{\left(\sum_{k=1} e^{z_{kb}}\right)^2} = A_{a,b} - A_{ab}^2$$

$$\Rightarrow [i \neq a] = 0 - \frac{e^{z_{ab}} e^{z_{ib}}}{\left(\sum_{k=1}^{n_K} e^{z_{kb}}\right)^2} = -A_{ib} \cdot A_{ab}$$

$$\frac{\partial A_{a,b}}{\partial z_{ij}} = [\delta_{i,a} A_{ab}(1 - A_{ab}) - A_{ib} \cdot A_{ab}(1 - \delta_{ia})] \, \delta_{b,j}$$

cross entropy gradient ⟶

$$\mathcal{L} = \frac{1}{m} \sum_{b=1}^{m} \left(-\sum_{p=1}^{n_y} y_{pb} \cdot \log \hat{y}_{pb} \right)$$

$$g = \log(\hat{y}) \longrightarrow \frac{\partial g_{ke}}{\partial \hat{y}_{ij}} = \frac{1}{\hat{y}_{ke}} \delta_{k,i} \delta_{e,j}$$

$$\frac{\partial \mathcal{L}}{\partial g_{ij}} = \frac{1}{m} \left(-y_{ij} \cdot 1 \right) = -\frac{1}{m} y_{ij}$$

$$\frac{\partial \mathcal{L}}{\partial \hat{y}_{ij}} = \sum_{k=1}^{n_y} \sum_{e=1}^{m} \frac{\partial \mathcal{L}}{\partial g_{ke}} \cdot \frac{\partial g_{ke}}{\partial \hat{y}_{ij}} = \sum_{k=1}^{n_y} \sum_{e=1}^{m} \left(-\frac{1}{m} y_{ke} \right) \cdot \frac{1}{\hat{y}_{ke}} \delta_{k,i} \delta_{e,j} =$$

$$= -\frac{1}{m} \frac{y_{ij}}{\hat{y}_{ij}}$$

- softmax ⟶

$$\frac{\partial \mathcal{L}}{\partial z_{ij}} = \sum_{k=1}^{n_y} \sum_{e=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_{ke}} \cdot \frac{\partial \hat{y}_{ke}}{\partial z_{ij}} = \sum_{k=1}^{n_y} \sum_{e=1}^{m} \left(-\frac{1}{m} \frac{y_{ke}}{\hat{y}_{ke}} \right) \delta_{e,j} \left[\delta_{k,i} \hat{y}_{ke} (1 - \hat{y}_{ke}) - \hat{y}_{ie} \hat{y}_{ke} \right.$$

$$\left. (1 - \delta_{k,i}) \right] \delta_{e,j} = \sum_{k=1}^{n_y} \left(-\frac{1}{m} y_{kj} \right) \left[\delta_{k,i} (1 - \hat{y}_{kj}) - \hat{y}_{ij} (1 - \delta_{k,i}) \right]$$

$$= \left(-\frac{1}{m} y_{ij} \right) (1 - \hat{y}_{ij}) + \sum_{k=1}^{n_y} \left(+\frac{1}{m} y_{kj} \right) \hat{y}_{ij} - \left(\frac{1}{m} y_{ij} \cdot \hat{y}_{ij} \right) =$$

$$= -\frac{1}{m} y_{ij} + \frac{1}{m} \hat{y}_{ij} \underbrace{\sum_{k=1}^{n_y} y_{kj}}_{=1} = \frac{1}{m} \left(\hat{y}_{ij} - y_{ij} \right)$$

Softmax gradient ⟶

$$A_{a,b} = \frac{e^{z_{ab}}}{\sum_{k=1}^{n_K} e^{z_{kb}}} \quad ; \quad h(x) = g(x)/f(x) \longrightarrow h'(x) = \frac{g'(x)f(x) - g(x)f'(x)}{f^2(x)}$$

$$\frac{\partial A_{a,b}}{\partial z_{i,j}} = \frac{\left(e^{z_{ab}}\right)' \sum_{k=1}^{n_K} e^{z_{kb}} - e^{z_{ab}} \left(\sum_{k=1}^{n_K} e^{z_{kb}}\right)'}{\left(\sum_{k=1}^{n_K} e^{z_{kb}}\right)^2} = \left| \; 0 \text{ if } j \neq b \; \right| =$$

$$= \frac{\left(e^{z_{ab}}\right)' \sum_{k=1}^{n_K} e^{z_{kb}} - e^{z_{ab}} \left(\sum_{k=1}^{n_K} e^{z_{kb}}\right)'}{\left(\sum_{k=1}^{n_K} e^{z_{kb}}\right)^2} \delta_{j,b} =$$

$$\underline{i=a} \Rightarrow \frac{e^{z_{ab}}}{\sum_{k=1}^{n_K} e^{z_{kb}}} - \frac{e^{z_{ab}} \cdot e^{z_{ab}}}{\left(\sum_{k=1}^{n_K} e^{z_{kb}}\right)^2} = A_{ab} - A_{ab}^2$$

$$i \neq a \Rightarrow 0 - \frac{e^{z_{ab}} e^{z_{ib}}}{\left(\sum_{k=1}^{n_K} e^{z_{kb}}\right)^2} = - A_{ab} A_{ib}$$

$$\frac{\partial A_{ab}}{\partial z_{i,j}} = \left[(A_{ab} - A_{ab}^2) \delta_{i,a} - A_{ab} A_{ib} (1 - \delta_{i,a}) \right] \delta_{jb}$$

Batch Normalization ⟶

samples

$x \longrightarrow (m, n_x)$

$y \longrightarrow (m, n_y)$

$$\mu_\ell = \sum_{p}^{M} x_{p\ell} \cdot \frac{1}{N} \quad ; \quad \sigma^2 = \frac{1}{N} \sum_{p}^{M} (x_{p\ell} - \mu_\ell)^2$$

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \varepsilon}} \longrightarrow \hat{x}_{k\ell} = \frac{x_{k\ell} - \mu_\ell}{\sqrt{\sigma_\ell^2 + \varepsilon}}$$

$$y = \gamma \hat{x} + \beta \longrightarrow \begin{pmatrix} y_{11} & y_{12} & \cdots \end{pmatrix} = \begin{pmatrix} \gamma_1 \hat{x}_{11} + \beta_1 & \gamma_2 \hat{x}_{12} + \beta_2 & \cdots \end{pmatrix}$$

$$y_{k\ell} = \gamma_\ell \hat{x}_{k\ell} + \beta_\ell \longrightarrow \frac{\partial y_{k\ell}}{\partial x_{ij}} = \gamma_\ell (\delta_{ik} \delta_{j\ell}) \longrightarrow \text{Element wise.}$$

$$\frac{\partial L}{\partial x_{ij}} = \sum_{k=1}^{M} \sum_{\ell=1}^{n_y} \frac{\partial L}{\partial y_{k\ell}} \cdot \frac{\partial y_{k\ell}}{\partial x_{ij}} = \sum_{k=1}^{M} \sum_{\ell=1}^{n_y} \frac{\partial L}{\partial y_{k\ell}} \cdot \frac{\partial y_{k\ell}}{\partial \hat{x}_{k\ell}} \cdot \frac{\partial \hat{x}_{k\ell}}{\partial x_{ij}}$$

* Just translation ⟶

$$\hat{x}_{k\ell} = x_{k\ell} - \mu_\ell = x_{k\ell} - \frac{1}{N} \sum_{p}^{M} x_{p\ell}$$

$$\frac{\partial \hat{x}_{k\ell}}{\partial x_{ij}} = \begin{cases} 1 - \frac{1}{N} & \text{if } k = i, \ell = j \\ -1/N & \text{if } k \neq i, \ell = j \\ 0 & \text{otherwise} \end{cases} = \delta_{\ell j} \left(\delta_{ik} - \frac{1}{N} \right)$$

* Variance ⟶

$$\sigma_l^2 = \sum_p^M (x_{pl} - \mu_l)^2 \cdot \frac{1}{N}$$

$$\frac{\partial \sigma_l^2}{\partial x_{ij}} = \frac{1}{N} \cdot 2 \sum_p^M \frac{\partial (x_{pl} - \mu_l)}{\partial x_{ij}} \cdot (x_{pl} - \mu_l) = \frac{2\delta_{lj}}{N} \sum_p^1 (x_{pl} - \mu_l)(\delta_{p,i} - \frac{1}{N})$$

$$= \frac{2\delta_{lj}}{N} \left[(x_{il} - \mu_l) + \frac{1}{N} \sum_p^M \mu_l - \frac{1}{N} \sum_p^M x_{pl} \right] =$$

$$\sum_p^M \mu_l = \frac{N}{N} \mu_l - \mu_l$$

$$= \frac{2}{N} \delta_{lj} (x_{il} - \mu_l)$$

* Norm ⟶

$$\frac{\partial \hat{x}_{kl}}{\partial x_{ij}} = \frac{\partial (x_{kl} - \mu_l)}{\partial x_{ij}} (\sigma^2 + \varepsilon)^{-1/2} - \frac{1}{2} \frac{\partial \sigma^2}{\partial x_{ij}} \cdot (x_{kl} - \mu_l)(\sigma_l^2 + \varepsilon)^{-3/2}$$

$$= \delta_{lj} (\delta_{ki} - \frac{1}{N}) (\sigma_l^2 + \varepsilon)^{-1/2} - \frac{1}{N} \delta_{lj} (x_{il} - \mu_l)(x_{kl} - \mu_l)(\sigma_l^2 + \varepsilon)^{-3/2}$$

* Cost function ⟶

$$\frac{\partial h}{\partial x_{ij}} = \sum_{k=1}^M \sum_{l=1}^{n_y} \frac{\partial h}{\partial y_{kl}} \cdot \frac{\partial y_{kl}}{\partial \hat{x}_{kl}} \cdot \frac{\partial \hat{x}_{kl}}{\partial x_{ij}} =$$

$$= \sum_{k=1}^M \sum_{l=1}^{n_y} \frac{\partial h}{\partial \hat{x}_{kl}} \cdot \gamma_l \cdot \delta_{lj} (\sigma^2 + \varepsilon)^{-1/2} \left[\delta_{ki} - \frac{1}{N} - \frac{1}{N} (x_{il} - \mu_l)(x_{kl} - \mu_l)(\sigma_l^2 + \varepsilon)^{-1} \right]$$

$$= \frac{\gamma_i}{N}(\sigma_j^2+\varepsilon)^{-1/2}\left[\frac{\partial h}{\partial y_{ij}} - \sum_{k=1}^{M}\frac{\partial h}{\partial y_{kj}} - \sum_{k=1}^{M}\frac{\partial h}{\partial y_{kj}}(x_{ij}-\mu_j)(x_{kj}-\mu_j)(\sigma_j^2+\varepsilon)^{-1/2}\right]$$

$$= \frac{\gamma_i}{N}(\sigma_j^2+\varepsilon)^{-1/2}\left[\frac{\partial h}{\partial y_{ij}} - \sum_{k=1}^{M}\frac{\partial h}{\partial y_{kj}} - (x_{ij}-\mu_j)(\sigma_j^2+\varepsilon)^{-1/2}\sum_{k=1}^{M}\frac{\partial h}{\partial y_{kj}}(x_{kj}-\mu_j)\right]$$

$$\frac{\partial h}{\partial y_{ij}} = \sum_{k=1}^{M}\sum_{e=1}^{n_y}\frac{\partial h}{\partial y_{ke}}\cdot\frac{\partial y_{ke}}{\partial \gamma_{ij}} = \sum_{k=1}^{M}\sum_{e=1}^{n_y}\frac{\partial h}{\partial y_{ke}}\cdot\delta_{ej}\cdot\hat{x}_{kj} =$$

$$= \sum_{k=1}^{M}\frac{\partial h}{\partial y_{kj}}\cdot\hat{x}_{kj}$$

$$\frac{\partial h}{\partial \beta_{ij}} = \sum_{k=1}^{M}\sum_{e=1}^{n_y}\frac{\partial h}{\partial y_{ke}}\frac{\partial y_{ke}}{\partial \beta_{ij}} = \sum_{k=1}^{M}\sum_{e=1}^{n_y}\frac{\partial h}{\partial y_{ke}}\cdot\delta_{ej} = \sum_{k=1}^{M}\frac{\partial h}{\partial y_{kj}}$$