

Coursera Notes

Improving Deep neural Networks: Hyperparameter Tuning, Regularization and Optimization Methods

Setting up an ML application:

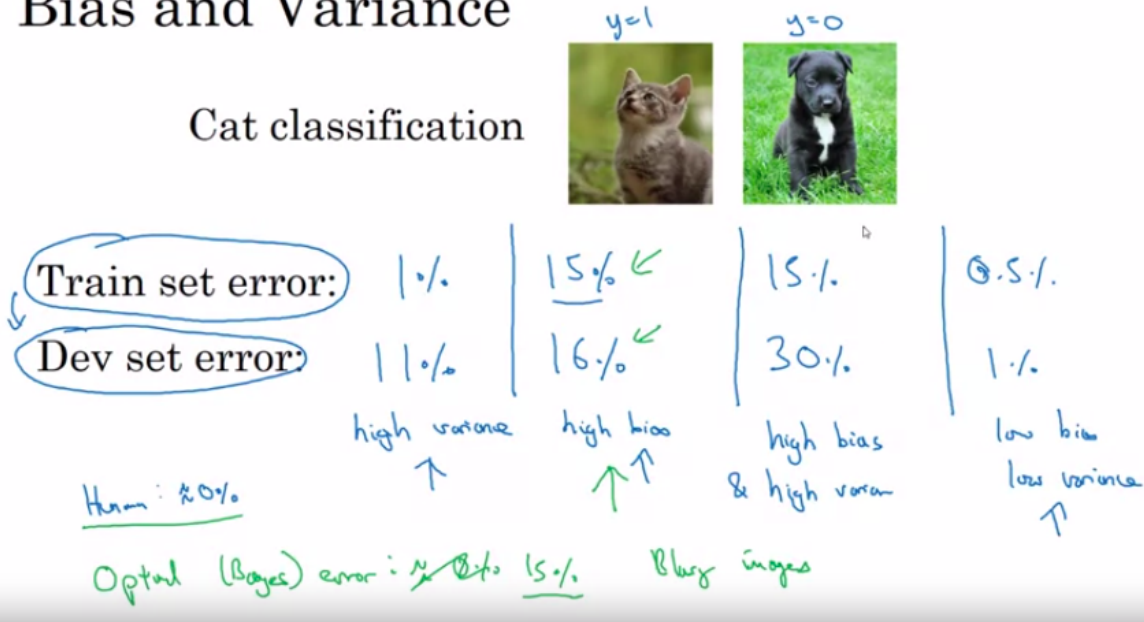
W1L1: train/dev/test sets

- Smaller datasets: 60/20/20 or 70/15/15
- Big datasets: 98/1/1
- Make sure that your dev and test sets come from the same data distributions or source of inputs

W1L2: bias/variance

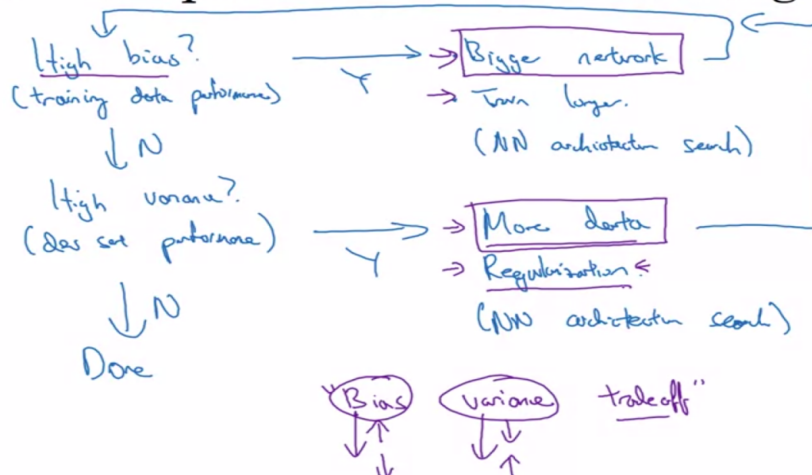
Bias and Variance

Cat classification



W1L3:

Basic recipe for machine learning



Andrew Ng

The vector norm $\|\mathbf{x}\|_p$ for $p = 1, 2, \dots$ is defined as

$$\|\mathbf{x}\|_p \equiv \left(\sum_i |x_i|^p \right)^{1/p}.$$

W1L4: Regularization:

- To prevent high variance problem & overfitting: use regularization
- L2 - generally used - also called as weight decay
- L1 - weights will be sparse

Neural network

$$J(\mathbf{w}^{[1]}, b^{[1]}, \dots, \mathbf{w}^{[L]}, b^{[L]}) = \underbrace{\frac{1}{m} \sum_{i=1}^m \ell(\hat{y}^{(i)}, y^{(i)})}_{\text{Loss}} + \underbrace{\frac{\lambda}{2m} \sum_{l=1}^L \|\mathbf{w}^{[l]}\|_F^2}_{\text{Regularization}}$$

$$\|\mathbf{w}^{[l]}\|_F^2 = \sum_{i=1}^{n^{[l-1]}} \sum_{j=1}^{n^{[l]}} (w_{ij}^{[l]})^2$$

"Frobenius norm"

$$\mathbf{w}^{[l]} = \begin{pmatrix} w_{11}^{[l]} & w_{12}^{[l]} & \dots & w_{1n^{[l]}}^{[l]} \\ w_{21}^{[l]} & w_{22}^{[l]} & \dots & w_{2n^{[l]}}^{[l]} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n^{[l-1]}1}^{[l]} & w_{n^{[l-1]}2}^{[l]} & \dots & w_{n^{[l-1]}n^{[l]}}^{[l]} \end{pmatrix}$$

$$\|\cdot\|_2^2 \quad \|\cdot\|_F^2$$

$$d\mathbf{w}^{[l]} = (\text{from backprop}) + \frac{\lambda}{m} \mathbf{w}^{[l]}$$

$$\rightarrow \mathbf{w}^{[l]} := \mathbf{w}^{[l]} - \alpha d\mathbf{w}^{[l]}$$

"Weight decay"

$$\mathbf{w}^{[l]} := \mathbf{w}^{[l]} - \alpha \left[(\text{from backprop}) + \frac{\lambda}{m} \mathbf{w}^{[l]} \right]$$

$$\frac{\partial J}{\partial \mathbf{w}^{[l]}} = d\mathbf{w}^{[l]}$$

W1L5: Why regularization reduces overfitting?

Regularization in machine learning is the process of regularizing the parameters that constrain, regularizes, or shrinks the coefficient estimates towards zero. In other words, this technique discourages learning a more complex or flexible model, avoiding the risk of **Overfitting**.

W1L6/7: Dropout regularization

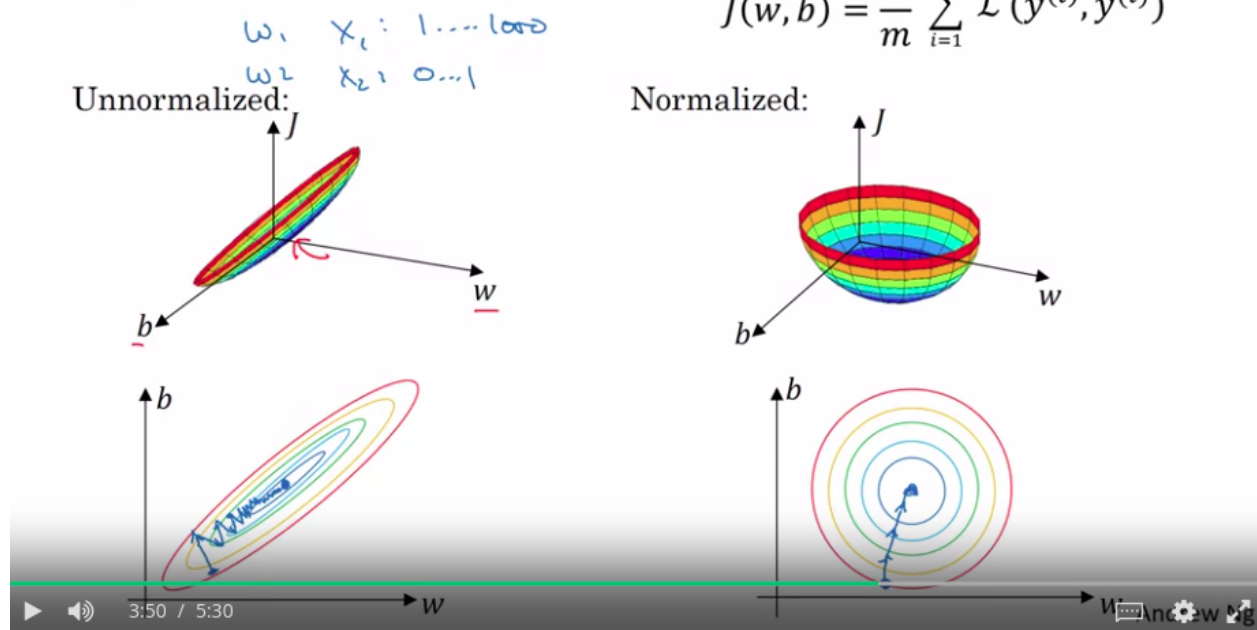
- It helps prevent overfitting as the weights have to shrink when some features are dropped.
- In computer vision applications: dropout is used most often
- Downside: cost function J is not well defined, harder to double-check the performance of the gradient descent model.
- To avoid this: Ng runs the code with dropouts off, keep prob = 1, and then turns the dropout on to avoid introducing bugs.

W1L8: few more regularization techniques:

- Data Augmentation:
- Early stopping:

W1L9: Normalizing the inputs

Why normalize inputs?



W1L10: vanishing/exploding gradients:

for a deep neural network with L layers (L is large integer):

- If $W < \text{Identity matrix}$: vanishing gradients
- If $W > \text{identity matrix}$: exploding gradients as W raised to $L-1$ gives the hypothesis

W1L11: Weight Initialization: To avoid vanishing and exploding gradients

- Relu : $W = \text{np.random.rand(shape)} * \text{np.sqrt}(1/(n^{(L-1)}))$ where n: number of hidden units
- Tanh: Xavier Initialization

W2L1/2: Mini Batch Gradient Descent

- Mini batch is faster than batch GD.
- Dividing the whole dataset into small batches and update weights for every batch.
- Choosing the size of mini-batch:

Choosing your mini-batch size

If small toy set : Use batch gradient descent.
($m \leq 2000$)

Typical mini-batch sizes:

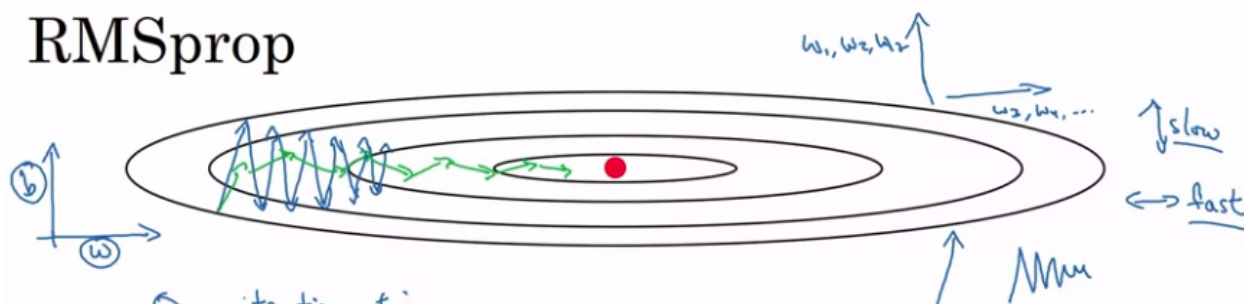
64, 128, 256, 512
2⁶ 2⁷ 2⁸ 2⁹

$$\frac{1024}{2^{10}}$$

Make sure min-batches $B \leq$ in CPU/GPU memory.
 $X^{\{t\}}, Y^{\{t\}}$.

W2L4: RMSprop

RMSprop



On iteration t :

Compute dw, db on cost mini-batch
 \nwarrow element-wise

$$\underline{S_{dw}} = \beta_2 S_{dw} + (1-\beta_2) \underline{dw^2} \leftarrow \text{small}$$

$$\rightarrow \underline{S_{ab}} = \beta_2 \underline{S_{ab}} + (1-\beta_2) \underline{ab^2} \leftarrow \text{large}$$

$$w := w - \frac{\alpha \frac{dw}{dw}}{\sqrt{S_{dw} + \epsilon}} \quad b := b - \frac{\alpha \frac{db}{db}}{\sqrt{S_{db} + \epsilon}}$$

$$\epsilon = 10^{-8}$$

W2L5: Adam

Combination of exponential weighted avg and RMSprop

Adam optimization algorithm

$$V_{dw}=0, S_{dw}=0, V_{db}=0, S_{db}=0$$

On iteration t :

Compute dw, db using current mini-batch

$$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dw, \quad V_{db} = \beta_1 V_{db} + (1-\beta_1) db \quad \leftarrow \text{"momentum"} \beta_1$$

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dw^2, \quad S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2 \quad \leftarrow \text{"RMSprop"} \beta_2$$

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

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

$$W := W - \alpha \frac{V_{dw}^{\text{corrected}}}{\sqrt{S_{dw}^{\text{corrected}} + \epsilon}}, \quad b := b - \alpha \frac{V_{db}^{\text{corrected}}}{\sqrt{S_{db}^{\text{corrected}} + \epsilon}}$$

Andrew Ng

W2L6: learning rate decay

method1:

Learning rate decay

1 epoch = 1 pass through data.

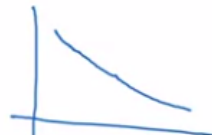
$$\alpha = \frac{1}{1 + \text{decay-rate} * \text{epoch-num}} \alpha_0$$

Epoch	α
1	0.1
2	0.67
3	0.5
4	0.4



$$\alpha_0 = 0.2$$

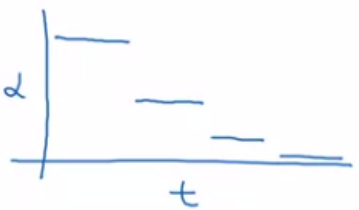
$$\text{decay-rate} = 1$$



Other learning rate decay methods

formula { $\alpha = 0.95^{\text{epoch-num}} \cdot \alpha_0$ - exponentially decay.

$\alpha = \frac{k}{\sqrt{\text{epoch-num}}} \cdot \alpha_0$ or $\frac{k}{\sqrt{t}} \cdot \alpha_0$

 discrete staircase

Manual decay.

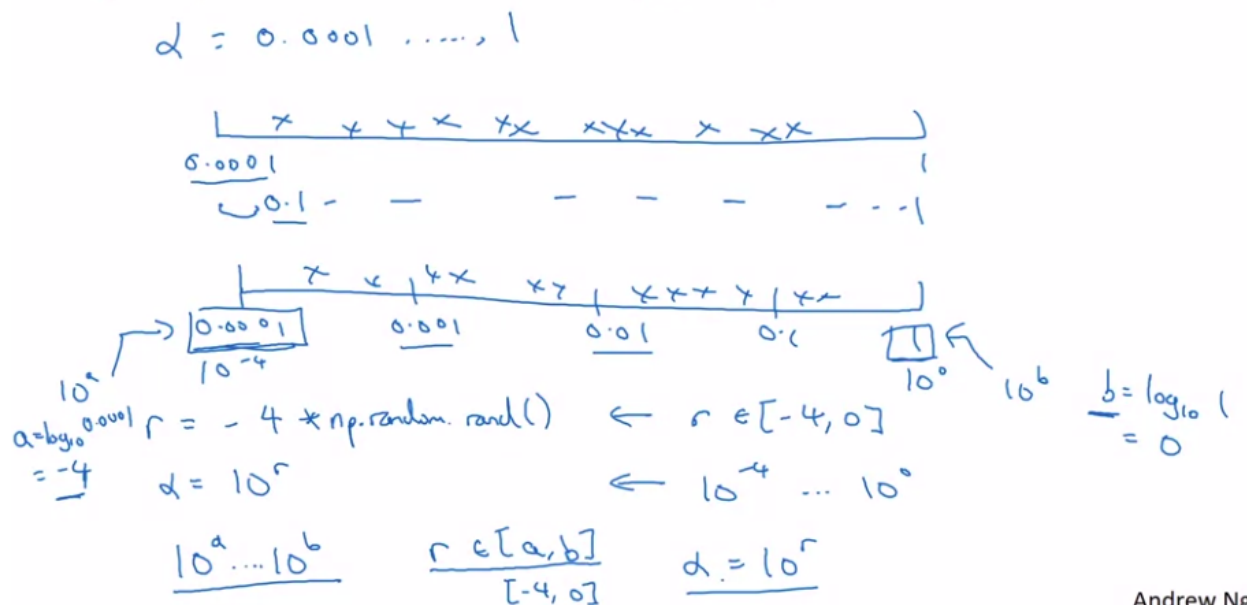
W3L1: tuning process

- Order of importance of tuning the hyperparameters:
learning rate
momentum
hidden units
mini-batch size
#layers
learning rate decay
adam optimizer parameters: beta1, beta2, epsilon
- Do not use a grid, try random samples

W3L2: picking the right scale for tuning: use logarithmic scale

- Learning rate:

Appropriate scale for hyperparameters



Andrew Ng

- Beta for exponentially weighted averages

Hyperparameters for exponentially weighted averages

$$\beta = 0.9 \quad \dots \quad 0.999$$

$$\downarrow \quad \quad \quad \downarrow$$

$$10 \quad \quad \quad 1000$$

$$1 - \beta = 0.1 \quad \dots \quad 0.001$$

$$\beta: 0.999 \rightarrow 0.9995 \quad \sim 10$$

$$\beta: 0.999 \rightarrow 0.9995 \quad \sim 1000 \quad \sim 2000$$

$$\frac{1}{1 - \beta}$$

$$r \in [-3, -1]$$

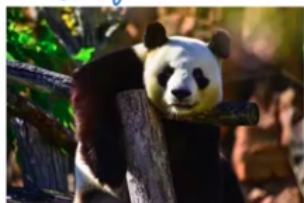
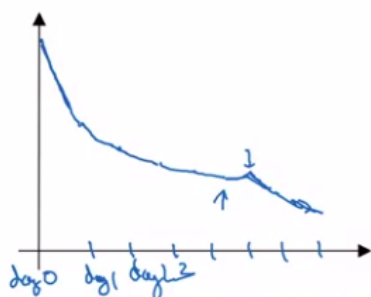
$$1 - \beta = 10^r$$

$$\beta = 1 - 10^r$$

An

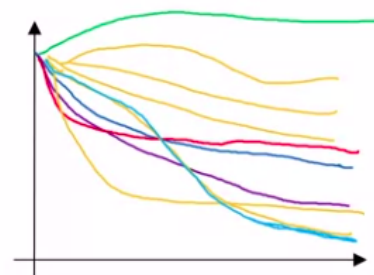
W3L3: tuning in practice: Application dependent

Babysitting one model



Panda

Training many models in parallel

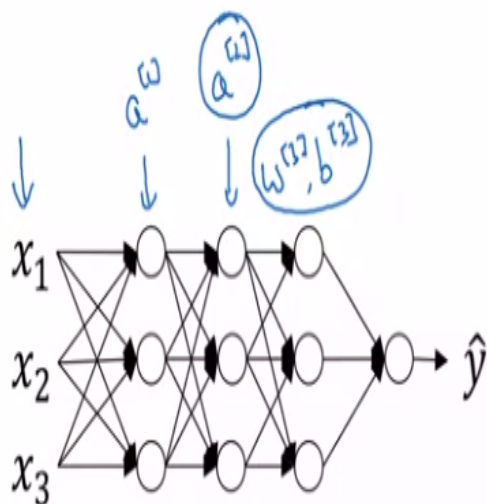


Caviar

Andrew

W3L4: Batch Normalization

- Normalizing the values of the parameters before applying the activation function.



Can we normalize $\frac{a^{[2]}}{w^{[2]}, b^{[2]}}$ so as to train faster

Normalize $\frac{z^{[2]}}{\uparrow}$

Andrew Ng

- Implementing Batch Norm:

Implementing Batch Norm

Given some intermediate values in NN

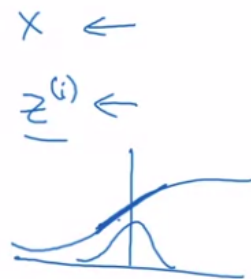
$z^{(1)}, \dots, z^{(n)}$

$$\begin{aligned} \mu &= \frac{1}{m} \sum_i z^{(i)} \\ \sigma^2 &= \frac{1}{m} \sum_i (z^{(i)} - \mu)^2 \\ z_{\text{norm}}^{(i)} &= \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}} \\ \hat{z}^{(i)} &= \gamma z_{\text{norm}}^{(i)} + \beta \end{aligned}$$

Use $\hat{z}^{(i)}$ instead of $z^{(i)}$.

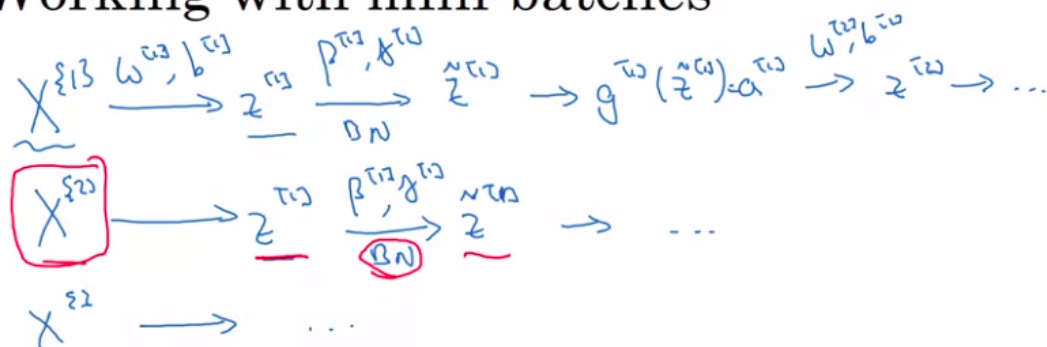
If $\gamma = \sqrt{\sigma^2 + \epsilon}$
 $\beta = \mu$
 then $\hat{z}^{(i)} = z^{(i)}$

learnable parameters of model.



Andrew Ng

Working with mini-batches



Parameters: $W^{t3}, \cancel{b^{t3}}, \beta^{t3}, \gamma^{t3}$

Dimensions: $(n^{t3}, 1), (n^{t3}, 1), (n^{t3}, 1)$

Input: $z^{t3} (n^{t3}, 1)$

Forward pass equations:

$$z^{t4} = W^{t4} a^{t3} + \cancel{b^{t4}}$$

$$z^{t4} = W^{t4} a^{t3}$$

$$z_{\text{norm}}^{t4} = \gamma^{t4} z_{\text{norm}}^{t3} + \beta^{t4}$$

Andrew Ng

Implementing gradient descent

for $t = 1 \dots \text{num Mini Batches}$ Compute forward pass on X^{t3} .In each hidden layer, use BN to replace z^{t3} with \tilde{z}^{t3} .Use backprop & compute $\frac{dW^{t3}}{dt}, \frac{d\beta^{t3}}{dt}, \frac{d\gamma^{t3}}{dt}$

Update parameters:

$$\left. \begin{aligned} W^{t3} &:= W^{t3} - \alpha \frac{dW^{t3}}{dt} \\ \beta^{t3} &:= \beta^{t3} - \alpha \frac{d\beta^{t3}}{dt} \\ \gamma^{t3} &:= \dots \end{aligned} \right\} \leftarrow$$

Works w/ momentum, RMSprop, Adam.

Batch Norm at test time

→ $\mu = \frac{1}{m} \sum_i z^{(i)}$

→ $\sigma^2 = \frac{1}{m} \sum_i (z^{(i)} - \mu)^2$

→ $z_{\text{norm}}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$

→ $\tilde{z}^{(i)} = \gamma z_{\text{norm}}^{(i)} + \beta$

μ, σ^2 : estimate using exponentially weighted average (across mini-batches).

$X^{[1]}, X^{[2]}, X^{[3]}, \dots$

$\mu^{[1]}, \mu^{[2]}, \mu^{[3]}, \dots \rightarrow \mu$

$\sigma^{[1]}, \sigma^{[2]}, \sigma^{[3]}, \dots \rightarrow \sigma^2$

$z_{\text{norm}} = \frac{z - \mu}{\sqrt{\sigma^2 + \epsilon}}$

$\tilde{z} = \gamma z_{\text{norm}} + \beta$

Andrew Ng

W3L6: Multiclass classification:

- Softmax Layer: takes a vector as an input and outputs a vector with the same dimensions. It computes the chance of occurring a particular class in the multiclass classification.
- In the example below: there is 84% chance of the predicted class to be class 0.

Softmax layer

Diagram showing the flow of data through a neural network layer to the Softmax layer.

Input X passes through several layers of nodes, leading to the Softmax layer (Layer L).

Layer L output vector $z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$.

Softmax output vector $\hat{y} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$.

Handwritten calculations for the Softmax layer:

Layer L: $z^{[L]} = w^{[L]} a^{[L-1]} + b^{[L]}$ (4,1)

Activation function: $t = e^{z^{[L]}}$ (4,1)

Output: $a^{[L]} = \frac{e^{z^{[L]}}}{\sum_{j=1}^4 t_j}$, $a_i = \frac{t_i}{\sum_{j=1}^4 t_j}$

Calculation: $a^{[L]} = g^{[L]}(z^{[L]})$ (4,1)

Calculation: $t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \\ 20.1 \end{bmatrix}$, $\sum_{j=1}^4 t_j = 176.3$

Calculation: $a^{[L]} = \frac{t}{176.3}$

Andrew Ng

Loss function

$(4,1)$
 $y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ ← cat $y_2 = 1$
 $y_1 = y_3 = y_4 = 0$

$(4,1)$
 $a = \hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$ ←

$C = 4$

$\mathcal{L}(\hat{y}, y) = - \sum_{j=1}^C y_j \log \hat{y}_j$
s small

$\mathcal{J}(w^{(1)}, b^{(1)}, \dots) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$

$-y_2 \log \hat{y}_2 = -\log \hat{y}_2$ make \hat{y}_2 big.

$Y = [y^{(1)} \ y^{(2)} \ \dots \ y^{(m)}]$ $\hat{Y} = [\hat{y}^{(1)} \ \dots \ \hat{y}^{(m)}]$

$(4, m)$ $(4, m)$

$= \begin{bmatrix} 0 & 0 & 1 & 0 & \dots \\ 1 & 0 & 0 & 0 & \dots \\ 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & \dots \end{bmatrix}$ $= \begin{bmatrix} 0.3 & & & \\ 0.2 & & & \\ 0.1 & & & \\ 0.4 & & & \end{bmatrix}$

Andrew Ng