Deep Learning Study

2020/07/07

Byungchan Lee

딥러닝 2단계 : 심층 신경망 성능 향상시키기

하이퍼파라미터 튜닝

- 튜닝 프로세스
  업데이트 : 2020.01.23 | 3
- 적절한 척도 선택하기
  업데이트 : 2020.01.23 | 3
- 하이퍼파라미터 튜닝 실전
  업데이트 : 2020.01.28 | 4

Batch Normalization

- 배치 정규화
  업데이트 : 2020.02.10 | 4
- 배치 정규화 적용시키기
  업데이트 : 2020.06.16 | 5
- 배치 정규화가 잘 작동하는 이유는 무엇일까요?
  업데이트 : 2020.02.10 | 3
- 테스트시의 배치 정규화
  업데이트 : 2020.02.11 | 2

다중 클래스 분류

- Softmax Regression
  업데이트 : 2020.03.17 | 4
- Softmax 분류기 훈련시키기
  업데이트 : 2020.02.11 | 3

프로그래밍 프레임워크 소개

- 지역 최적값 문제
  업데이트 : 2020.02.11 | 3
- Tensorflow
  업데이트 : 2020.02.11 | 6

다음 코스 안내

- 다음 코스: 머신러닝 프로젝트 구조화하기
  업데이트 : 2020.02.11 | 5

Hyperparameter Tuning

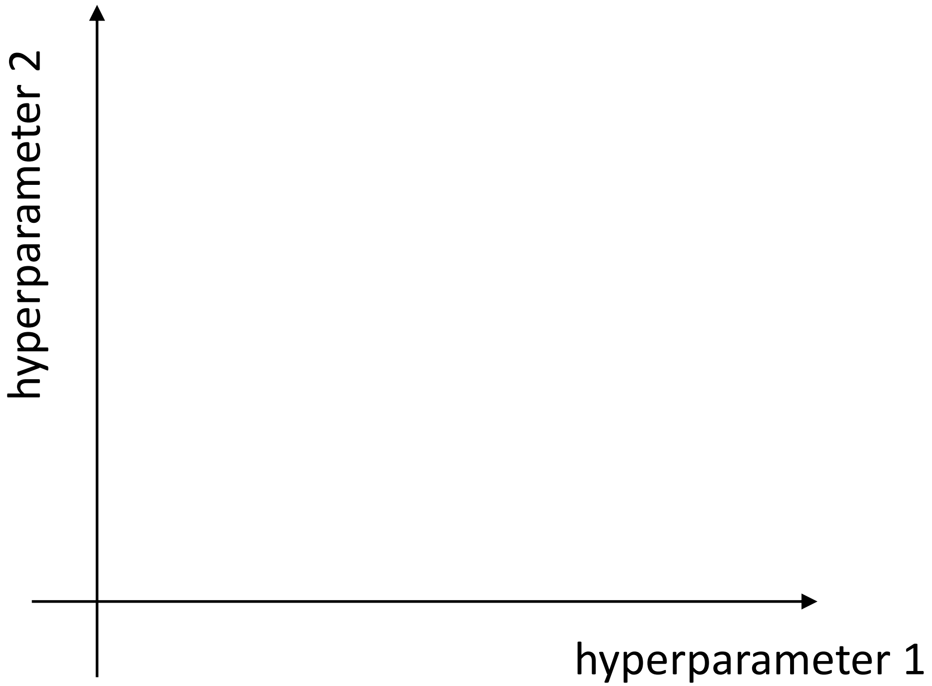
- α

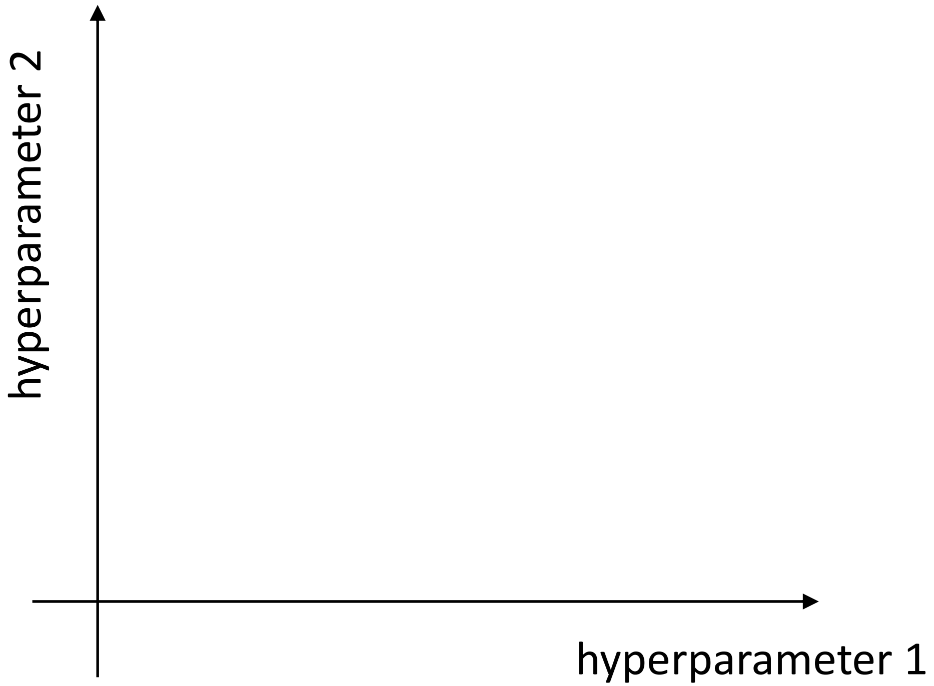
- β
- # hidden units
- Learning rate decay

- # layers
- mini-batch size

- β_1, β_2, ε

Hyperparameter Tuning

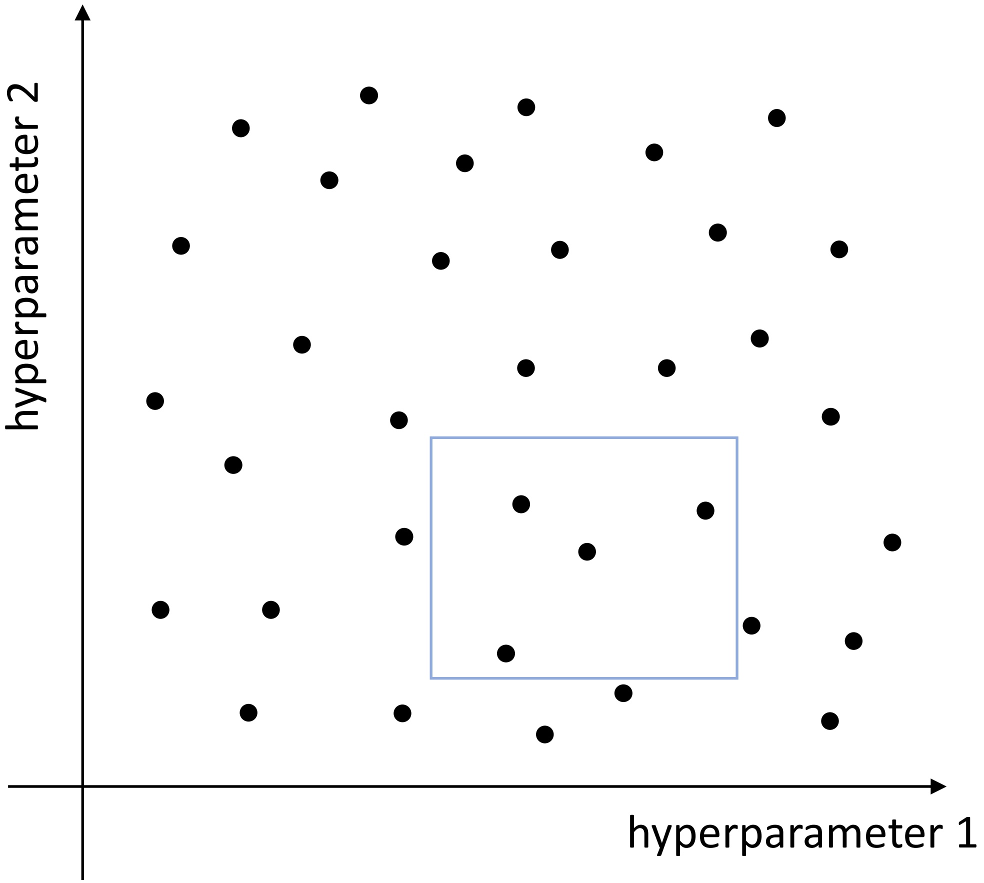




- Try Random Values

Hyperparameter Tuning

- Coarse to fine



Hyperparameter Tuning

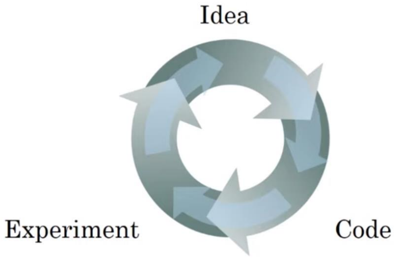
- Using an appropriate scale

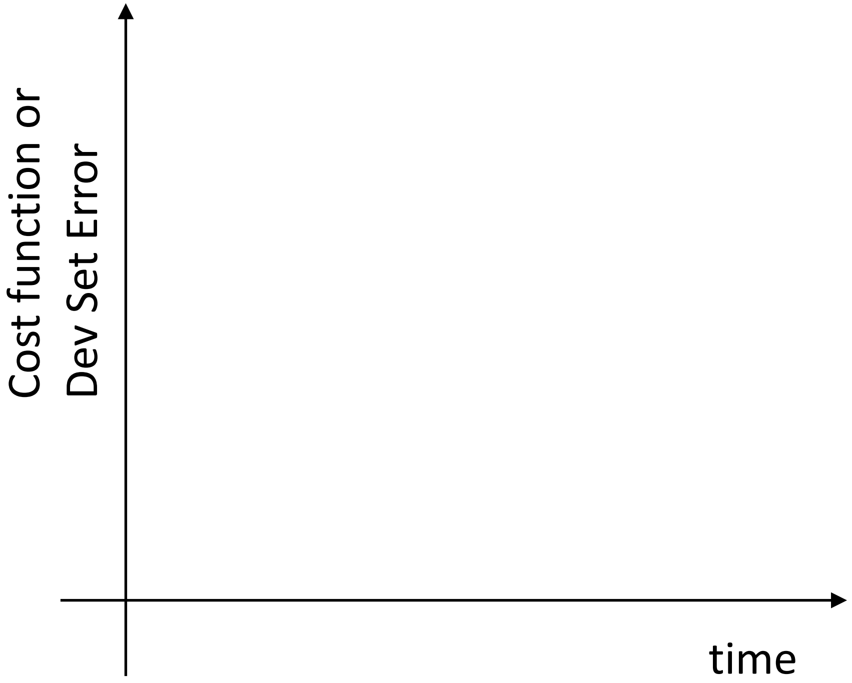
- # of layers   2, 3, 4

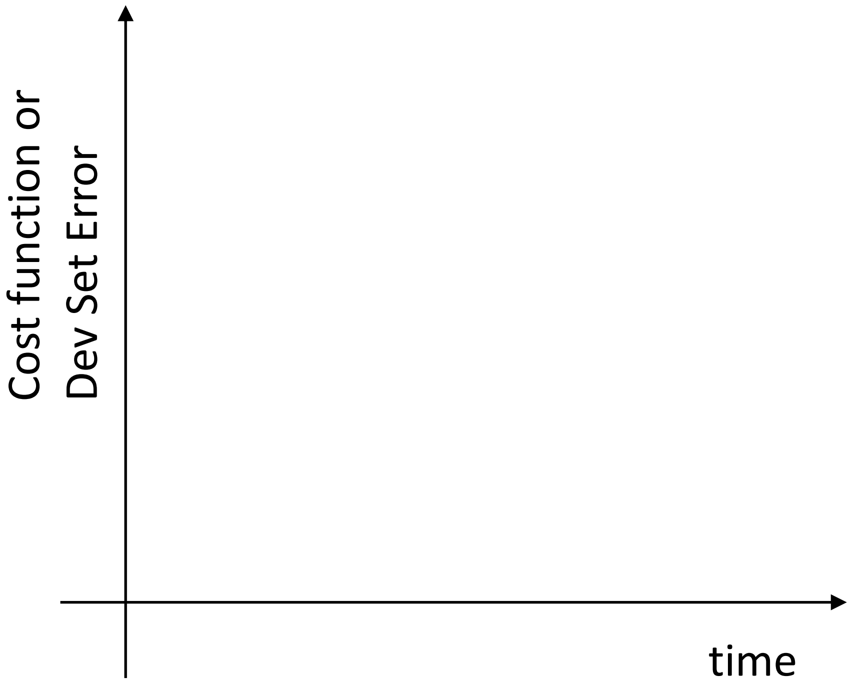
- α 10^r 0.0001 ———————— 1

- β $1 - 10^r$ 0.9 ———————— 0.999

Hyperparameter Tuning







- Panda approach
- Caviar approach

Batch Normalization

- Sergey Loffe, Christian Szegedy (2015) https://arxiv.org/abs/1502.03167

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe, Christian Szegedy

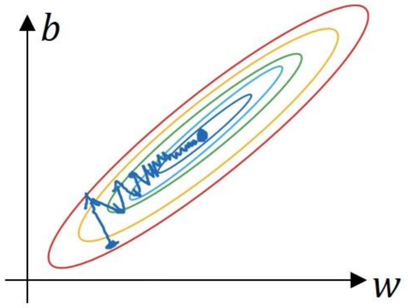
Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as internal covariate shift, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization for each training mini-batch. Batch Normalization allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularizer, in some cases eliminating the need for Dropout. Applied to a state-of-the-art image classification model, Batch Normalization achieves the same accuracy with 14 times fewer training steps, and beats the original model by a significant margin. Using an ensemble of batch-normalized networks, we improve upon the best published result on ImageNet classification: reaching 4.9% top-5 validation error (and 4.8% test error), exceeding the accuracy of human raters.

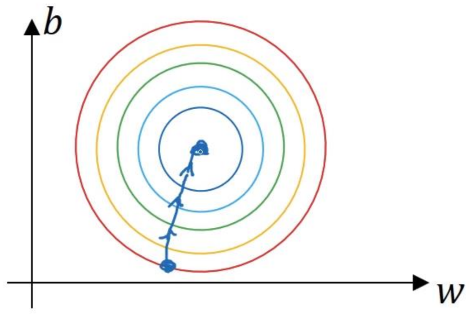
In Tensorflow

- tf.nn.batch-normalization(..)

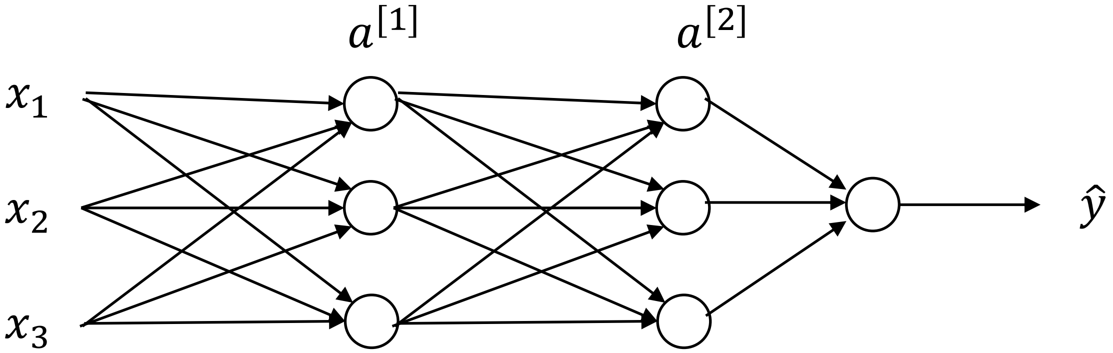
Batch Normalization

- (C2W1L09) Normalizing inputs





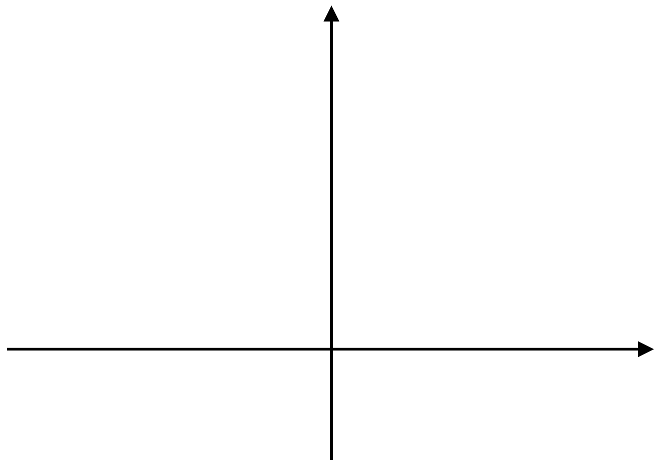
Andrew Ng



- Normalize $z^{[i]}$, not $a^{[i]}$

Batch Normalization

- $\mu = \frac{1}{m}\sum z^{(i)}$
- $\sigma^2 = \frac{1}{m}\sum (z^{(i)} - \mu)^2$
- $z_{norm}^{[i]} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$
- $\tilde{\boldsymbol{z}}^{[\boldsymbol{i}]} = \boldsymbol{\gamma} \boldsymbol{z}_{\boldsymbol{norm}}^{[\boldsymbol{i}]} + \boldsymbol{\beta}$



- γ, β : learnable parameters

for each layer,

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

Batch Normalization

- $X^{\{1\}} \rightarrow \left(w^{[1]}, b^{[1]}\right) \rightarrow z^{[1]} \rightarrow BN\left(\gamma^{[1]}, \beta^{[1]}\right) \rightarrow \tilde{z}^{[1]} \rightarrow a^{[1]} \rightarrow z^{[2]} \rightarrow \ \ldots$
- $X^{\{2\}} \rightarrow \left(w^{[1]}, b^{[1]}\right) \rightarrow z^{[1]} \rightarrow BN\left(\gamma^{[1]}, \beta^{[1]}\right) \rightarrow \tilde{z}^{[1]} \rightarrow a^{[1]} \rightarrow z^{[2]} \rightarrow \ \ldots$

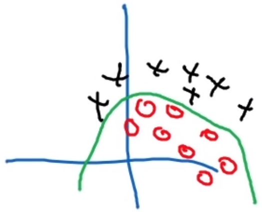
** Normalize $\tilde{z}$ using just the data in that mini batch

At test time, you might need to process the example one at a time

- μ , σ^2 : estimate using exponentially weight moving average(EMWA) across mini-batch
- compute $z_{norm}^{[i]}$ using $\mu , \sigma^2, \gamma^{[i]}, \beta^{[i]}$

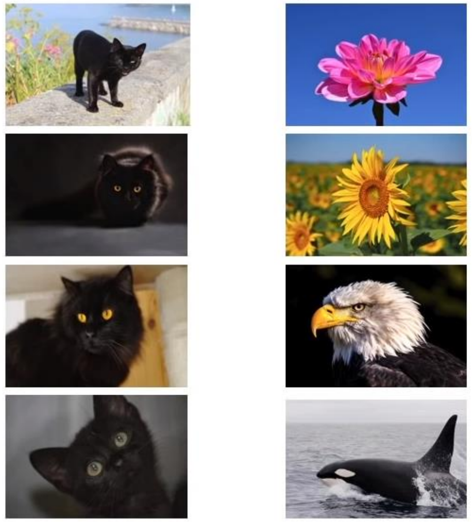
https://en.wikipedia.org/wiki/Moving_average

Batch Normalization



Cat | Non-Cat

$y = 1$ | $y = 0$

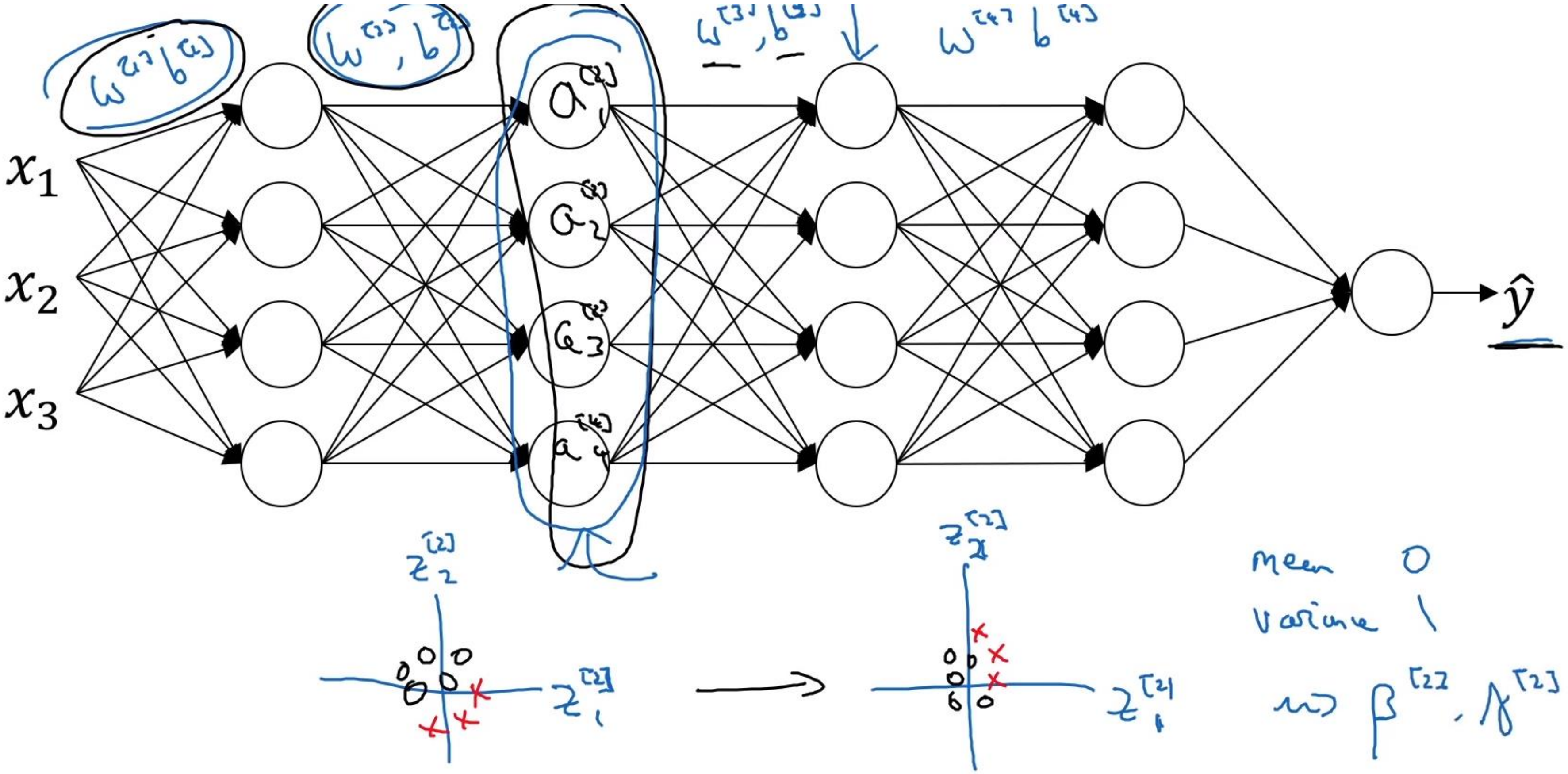


Covariant shift

X → Y

$y = 1$ | $y = 0$

Batch Normalization



Batch Normalization

Batch Normalization as regularization

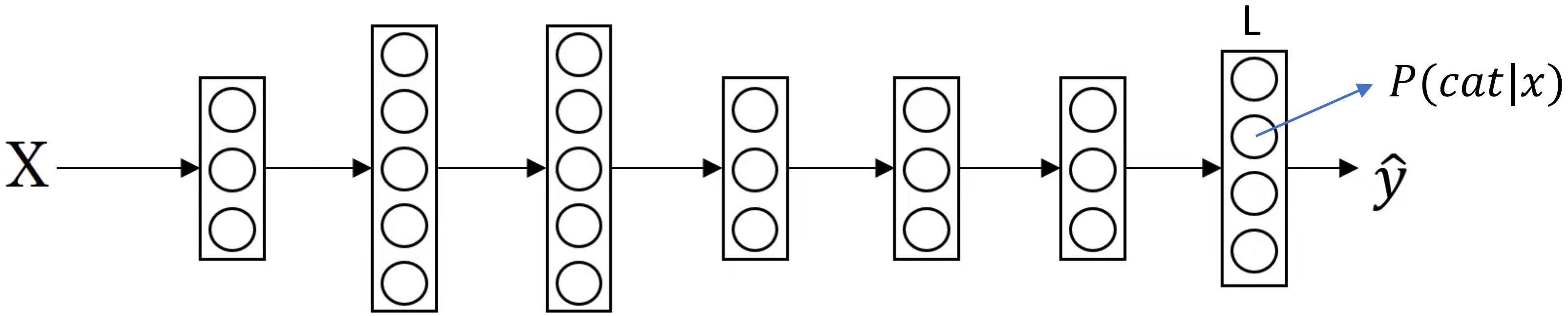
- Each mini-batch is scaled by the mean/variance computed on just that mini-batch
- This adds some noise to the values $z^{[l]}$ within that minibatch. So similar to dropout, it adds some noise to each hidden layer's activations
- This has a slight regularization effect

- Bigger mini-batch size reduce the regularization effect.
- Don't use batch normalization as a regularization. (It is almost an unintended side effect)

Multi-class classification

Recognizing cats, dogs, and baby chicks

- C = # classes
- In this case, C=4



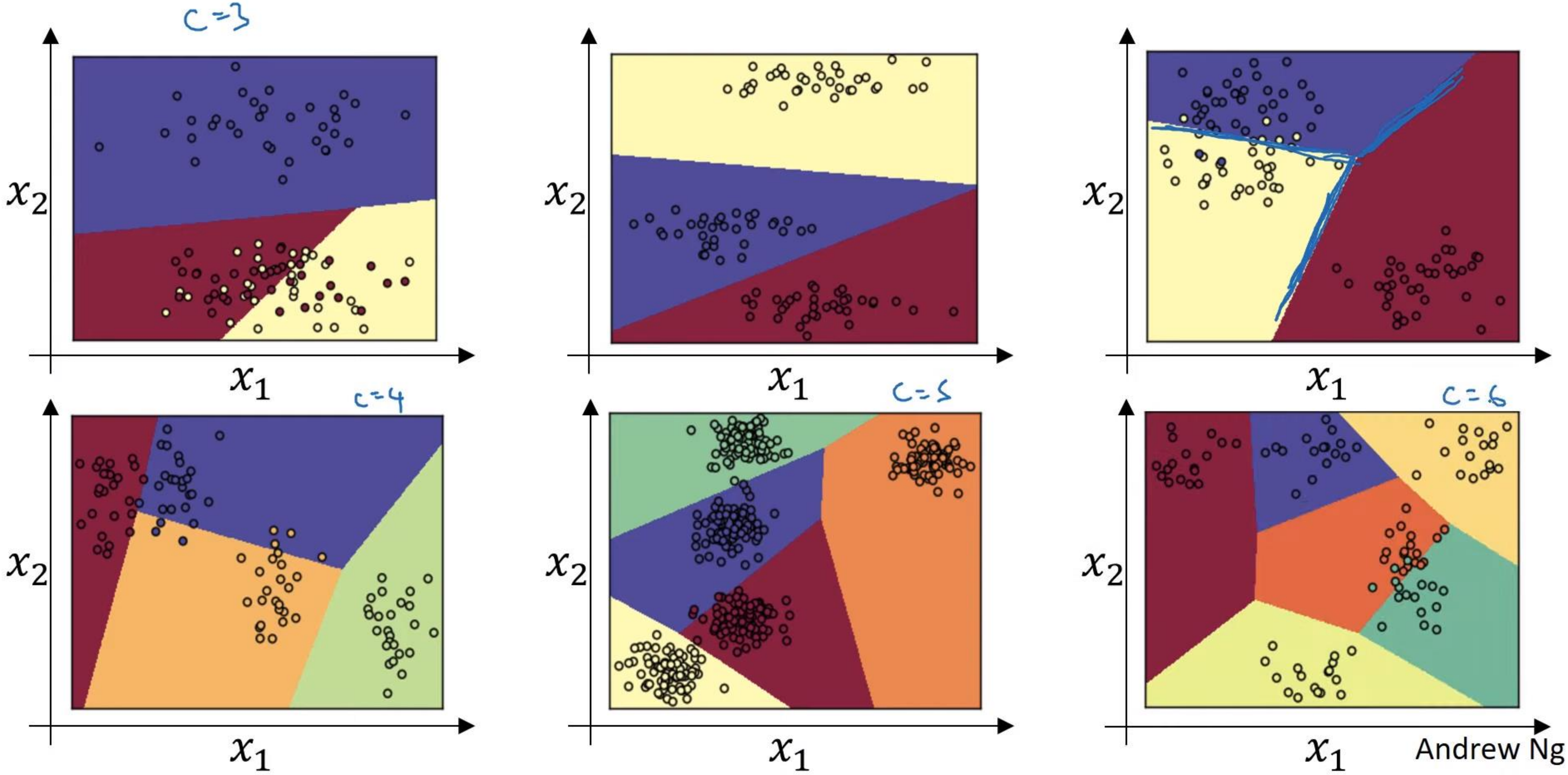
- $t = \exp(z^{[L]})$
- $a_i^{[L]} = t_i / \sum t_i$ $\leftarrow a^{[L]} = g^{[L]}(z^{[L]})$

Softmax examples

$x_1, x_2 \rightarrow \boxed{\circ\,\circ\,\circ} \rightarrow \hat{y}$

$z^{[l]} = w^{[l]} x + b^{[l]}$

$a^{[l]} = \hat{y} = g(z^{[l]})$



Andrew Ng

Multi-class classification

- Softmax vs Hardmax

$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$ → $a^{[L]} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$ hardmax : $a^{[L]} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

- Softmax regression generalizes logistic regression to C classes

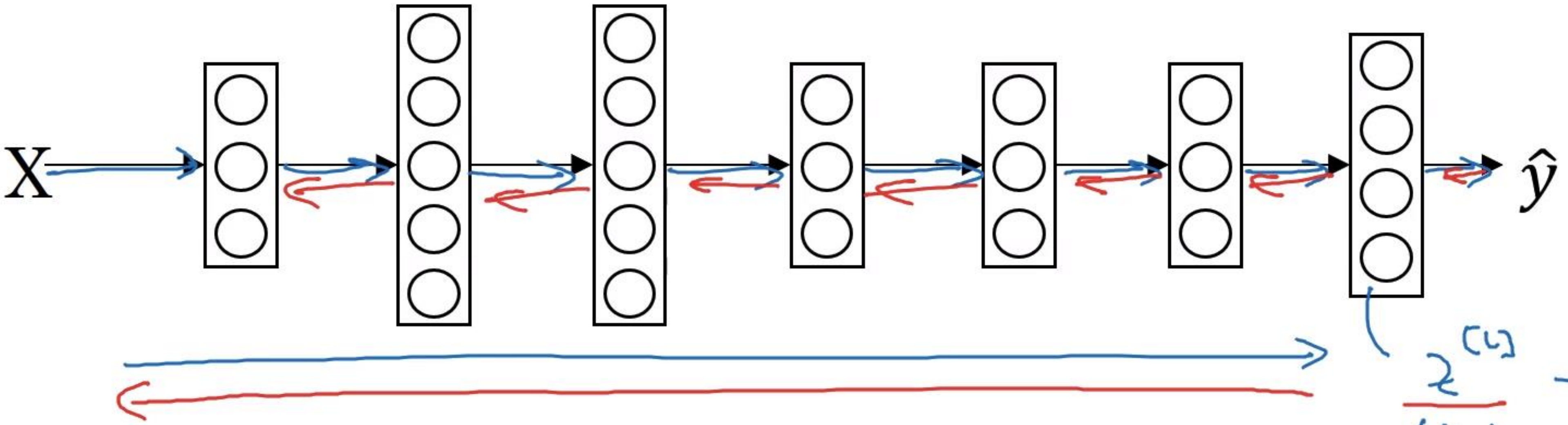
Loss function & Cost function

- $\mathcal{L}(\hat{y}, y) = -\sum_{i=1}^{4} y_i \log \hat{y}_i$

ex) $y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $\hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$ → $\mathcal{L}(\hat{y}, y) = -\log \hat{y}_2$

- $\mathrm{J} = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$

Multi-class classification



$\underset{(4,1)}{z^{[L]}} \longrightarrow a^{[L]} = \hat{y} \rightarrow \mathcal{L}(\hat{y}, y)$

Backprop: $\underset{(4,1)}{dz^{[L]} = \hat{y} - y}$

$\frac{\partial J}{\partial z^{[L]}}$