# WEEK 3. Hyperparameter Tuning, Batch Normalization and Programming Frameworks

## 1. Hyperparameter tuning

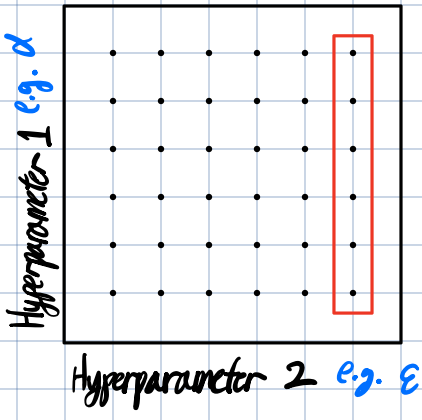
- Hyperparameters (Andrew Ng's tips)

① $\alpha$ ⇒ most important
② $\beta$ ~0.9
③ mini-batch size
④ # of hidden units
⑤ # of layers & learning rate decay
⑥ $\beta_1$, $\beta_2$, $\varepsilon$ for Adam optimization ⇒ commonly 0.9, 0.999, $10^{-8}$

⇒ How to explore?

- Try random values : Don't use a grid

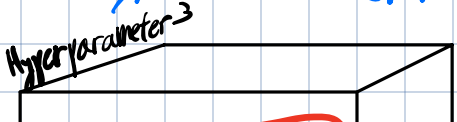
  - if # of hyperparameters ↓
    ⇒ search points in grid

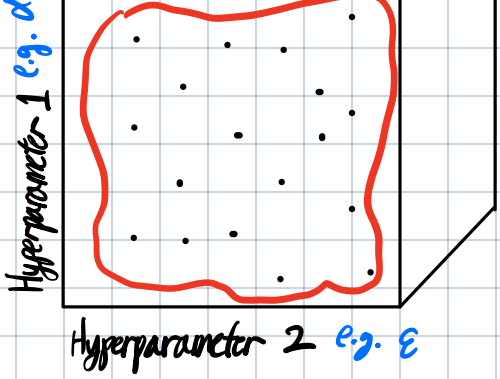



⇒ 만약, 6개의 서로 다른 $\alpha$에 대해 훈련했을 때 $\varepsilon$ 값이 달라도 결과가 같다면,
6개의 $\alpha$에 대해서만 학습시킨 셈과 다를 바 없음

  - if # of hyperparameters ↑ (i.e. in DL)
    ⇒ choose points randomly

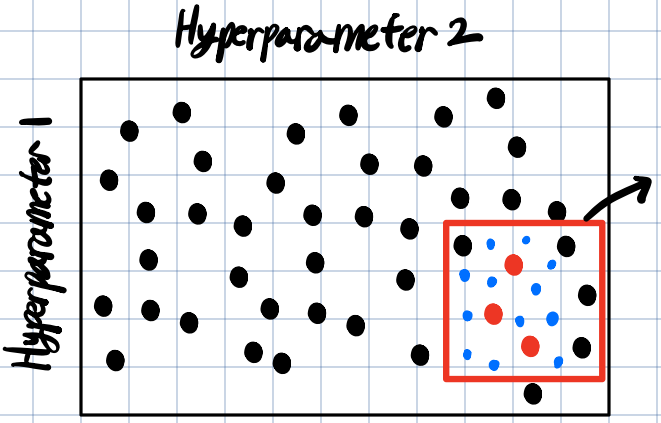
↗ 상대적으로 덜 중요한 hyperparameter






⇒ 25개의 서로 다른 점에 대해 학습 가능

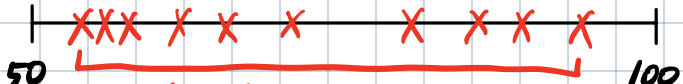
- Course to find




만약 빨간점 3개에서 model이 work well
⇒ 더 작은 영역을 정해 확대해서 더 조밀하게 점들을 탐색

- **Using an appropriate scale to pick hyperparameters.**
  - Picking hyperparameters at random

    Let's say we explore best $n^{[l]}$ ($n^{[l]} = 50, \dots, 100$)



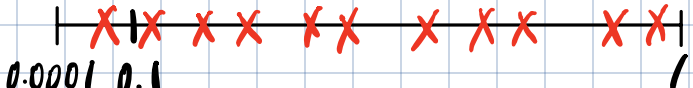

uniformly random over their range of contemplating

⇒ resonable case for random search

# layers ($L = 2, \dots, 4$) ⇒ grid search is OK

  - Appropriate scale for hyperparameters

    Let's say if we explore best $\alpha$ ($\alpha = 0.0001, \dots, 1$)




90% of samples are over the range between 0.1 and 1

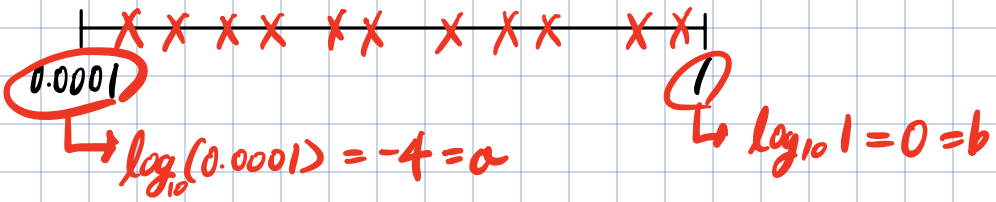
⇒ unreasonable case for random search

⇒ 0.0001 ~ 0.1 ?

⇒ Change linear scale to log scale for random search.

① hyperparameter α는 $\alpha = 10^r$ ($r \in [a, b]$)로 나타낼 수 있다.

② 최솟값과 최댓값에 log 취해 define a and b




$\log_{10}(0.0001) = -4 = a$

$\log_{10} 1 = 0 = b$

③ 이제 r을 uniformly random하게 a~b 사이에서 뽑을 수 있음

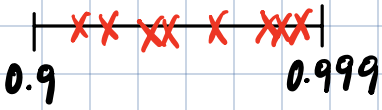
⇒ hyperparameter ($\alpha = 10^r$)를 random search할 수 있다.

$r \in [-4, 0]$ → $r = -4 * np.random.rand()$

$\alpha = 10^r$

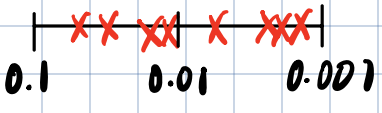
— Hyperparameters for exponentially weighted averages.

$\beta = 0.9 \dots 0.999$ → linear scale :




10 days 1000 days

$1-\beta = 0.1 \cdots 0.001$ → linear scale :




$10^{-1}$ $10^{-3}$

e.g. $\beta = 0.9000$ → $0.9005$ (0.0995)

$\frac{1}{1-\beta} = 10$ $\frac{1}{1-\beta} \approx 10$

$\beta = 0.999$ → $0.9995$

$\frac{1}{1-\beta} = 1000$ → $\frac{1}{1-\beta} = 2000$

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

$1-\beta = 10^r$

$\beta = 1 - 10^r$

⇒ In linear scale, huge impact when $\beta \approx 1$ (sample이 더 많을 때 봄)

• Hyperparameters tuning in practice : Pandas vs Caviar

— Re-test hyperparameters occasionally

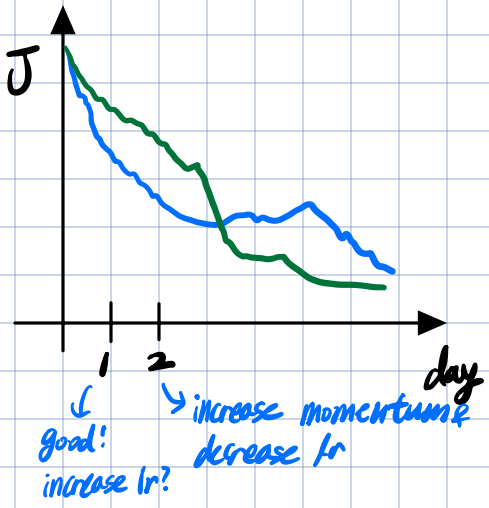
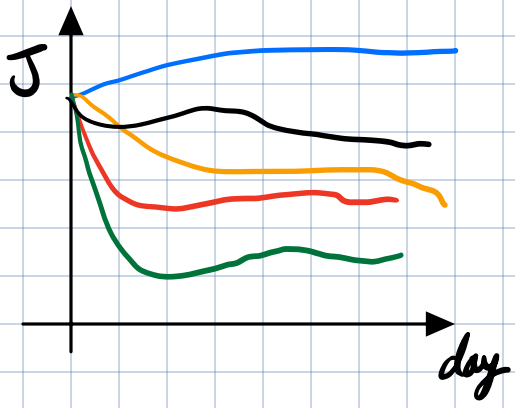
① hyperparameter에 대한 직관으로 다른 영역에서 쓸 수도 아닐 수도 있음.

e.g. NLP, Vision, Speech, Ads, logistics, ...

② 좋은 hyperparameter는 찾아도 아래 이유 때문에 자주 재점검 필요

- 알고리즘의 변화
- 데이터 갱신
- 데이터 센터 서버 업그레이드

— How to search hyperparameters?

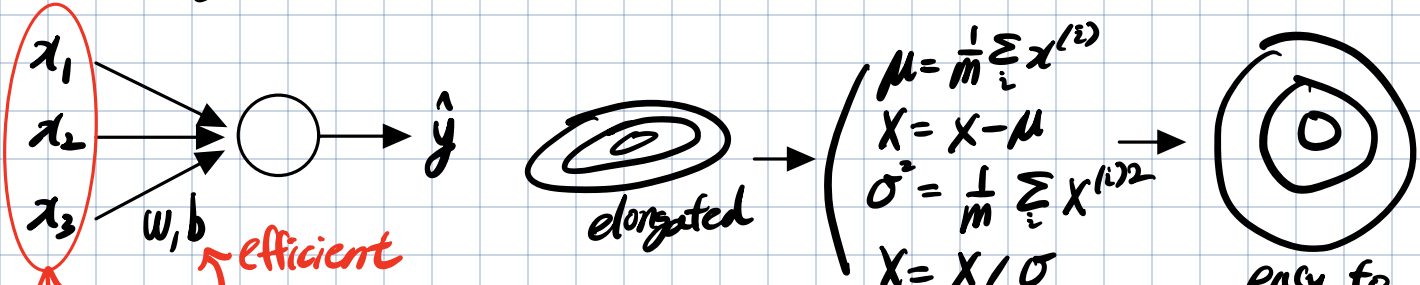
| Pandas | Caviar |
|---|---|
| Babysitting one model. | Training many models in parallel |
|   |   |
| 며칠, 몇주에 걸쳐 매일 모델 하나를 학습 (성능을 지켜보며 hyperparameter 조정) | 여러모델에 동시에 여러 hyperparameter 스스로 학습하게 한 뒤 성능 확인 |

## 2. Batch Normalization

- Normalizing activations in a network.

⇒ Make the model robust to hyperparameters

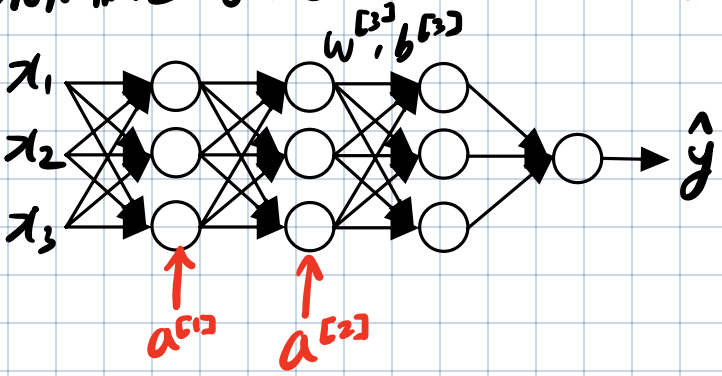
— Normalizing inputs to speed up learning



Normalize ↑ learning

optimize

⇒ DNN에서는 입력층뿐 아니라 hidden layer의 값도 정규화




⇒ Can we normalize $a^{[2]}$ for faster learning of $w^{[3]}$ & $b^{[3]}$?

A: Normalize $z^{[2]}$ (commonly used)

- Implementing Batch Norm

Given some intermediate values in NN

$$\mu = \frac{1}{m}\sum_i z^{(i)}$$

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

$$z_{norm}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$ → make sure not to divide by 0

$$\tilde{z}^{(i)} = \gamma z_{norm}^{(i)} + \beta$$ → learnable parameters ⇒ gradient descent, ...

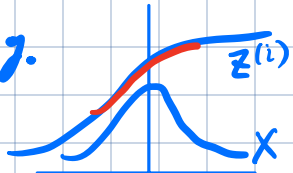
Use $\tilde{z}^{[l](i)}$ instead of $z^{[l](i)}$

Why $\gamma$ & $\beta$?

⇒ 입력층과 달리 hidden unit의 $\mu$ & $\sigma^2$이 항상 (0, 1)이 되는 건 bad

(hidden unit의 분포가 다양해야 효과적인 학습 가능)

⇒ $\gamma$, $\beta$를 이용해 hidden unit이 다양한 값을 가지게 조정

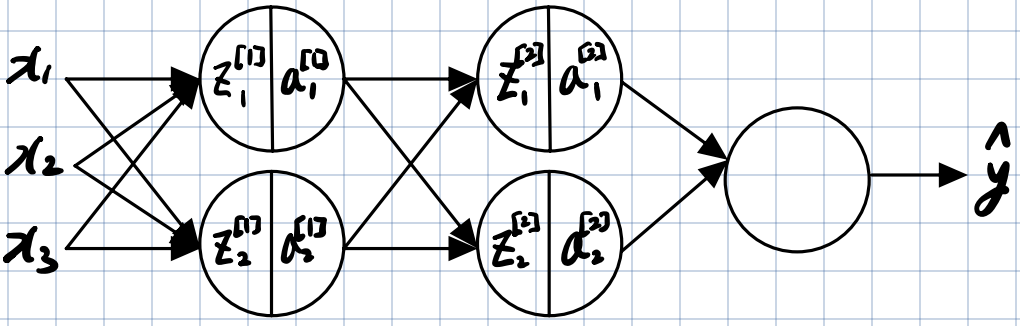
e.g. 


활성화 함수가 sigmoid일 때 값이 0에 몰리면 비선형성 약화

⇒ ∴ $\gamma$와 $\beta$로 분포 조정

If $\gamma = \sqrt{\sigma^2+\varepsilon}$, $\beta = \mu \Rightarrow \tilde{z}^{(i)} = z^{(i)}$

- Fitting batch norm into a neural network
  - Adding Batch Norm to a network




$$X \xrightarrow{w^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[\text{Batch Norm (BN)}]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \rightarrow a^{[1]} = g^{[1]}(\tilde{z}^{[1]}) \xrightarrow{w^{[2]}, b^{[2]}} z^{[2]} \xrightarrow[BN]{\beta^{[2]}, \gamma^{[2]}} \tilde{z}^{[2]} \rightarrow a^{[2]} \dots$$

Learnable parameters:

$$\{ w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}, \dots, w^{[L]}, b^{[L]}, \beta^{[1]}, \gamma^{[1]}, \beta^{[2]}, \gamma^{[2]}, \dots, \beta^{[L]}, \gamma^{[L]} \}$$

different from momentum / Adam

$\Rightarrow$ Gradient descent: $\beta^{[l]} = \beta^{[l]} - \alpha d\beta^{[l]}$

↳ RMS prop, Momentum 등도 사용가능!

$\Rightarrow$ tf.nn.batch-normalization

- Working with mini-batches

$\Rightarrow$ In practice, BN is often applied to mini-batches

$X^{\{1\}}$ (mini batch #1) $\xrightarrow{w^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[\text{Batch Norm (BN)}]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \rightarrow a^{[1]} = g^{[1]}(\tilde{z}^{[1]}) \xrightarrow{w^{[2]}, b^{[2]}} z^{[2]} \rightarrow \dots$

$X^{\{2\}} \xrightarrow{w^{[1]}, b^{[1]}} z^{[1]} \xrightarrow{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]}$

⋮ ↳ 2번째 mini batch에 대해 BN

Parameters: $w^{[l]}$, ~~$b^{[l]}$~~, $\beta^{[l]}$, $\gamma^{[l]}$

$(n^{[1]}, 1)$ $(n^{[2]}, 1)$ $(n^{[l]}, 1)$

for scaling

BN에서 $z$의 $\mu$ 계산 할 때에 더하기 때문에 bias 값은 없어짐 ($E(x+b) = E(x)+b$)

e.g.

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

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

$z^{[l]}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$

$\mu = \frac{1}{m}\sum_i x^{(i)}$

$\beta$가 대신해서 평균값을 역할

$\tilde{z}^{[l]} = \gamma^{[l]} z^{[l]}_{norm} + \beta^{[l]}$

- Implementing gradient descent → RMSprop, Adam, momentum, etc...

for $t = 1 \ldots$ num_minibatches

Compute forward prop on $X^{\{t\}}$

In each hidden layer, use BN to replace $z^{[l]}$ with $\tilde{z}^{[l]}$

Use backprop → $dW^{[l]}$, $d\beta^{[l]}$, $d\gamma^{[l]}$

Update parameters:

$W^{[l]} := W^{[l]} - \alpha dW^{[l]}$

$\beta^{[l]} := \beta^{[l]} - \alpha d\beta^{[l]}$

$\gamma^{[l]} := \gamma^{[l]} - \alpha d\gamma^{[l]}$

## • Why does Batch Norm work?

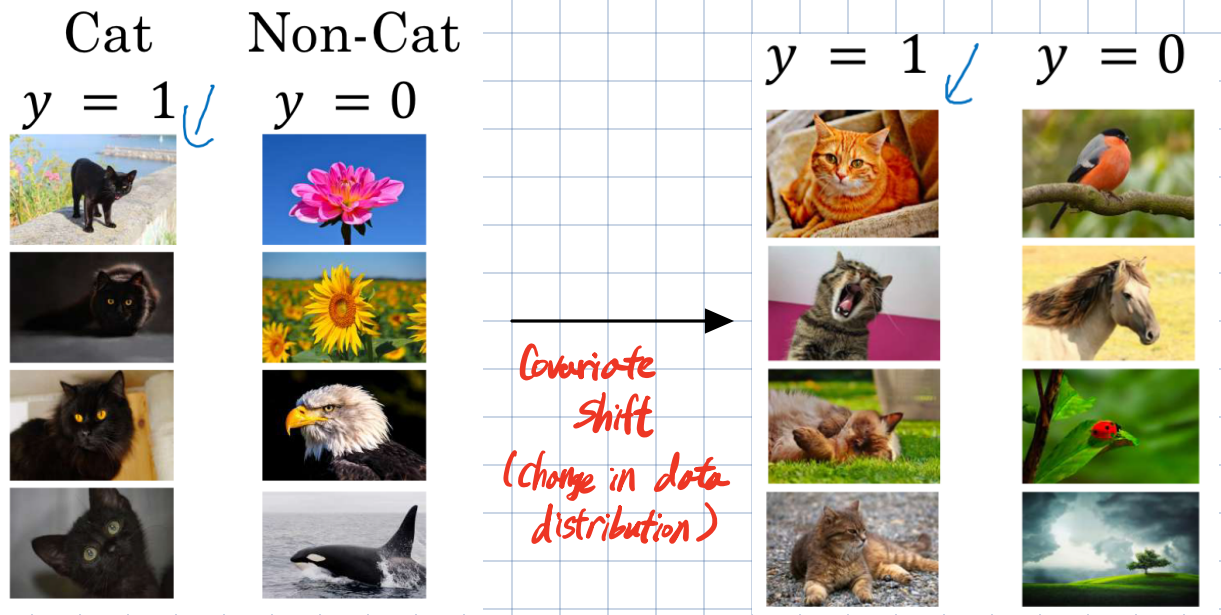
- Learning on shifting input distribution

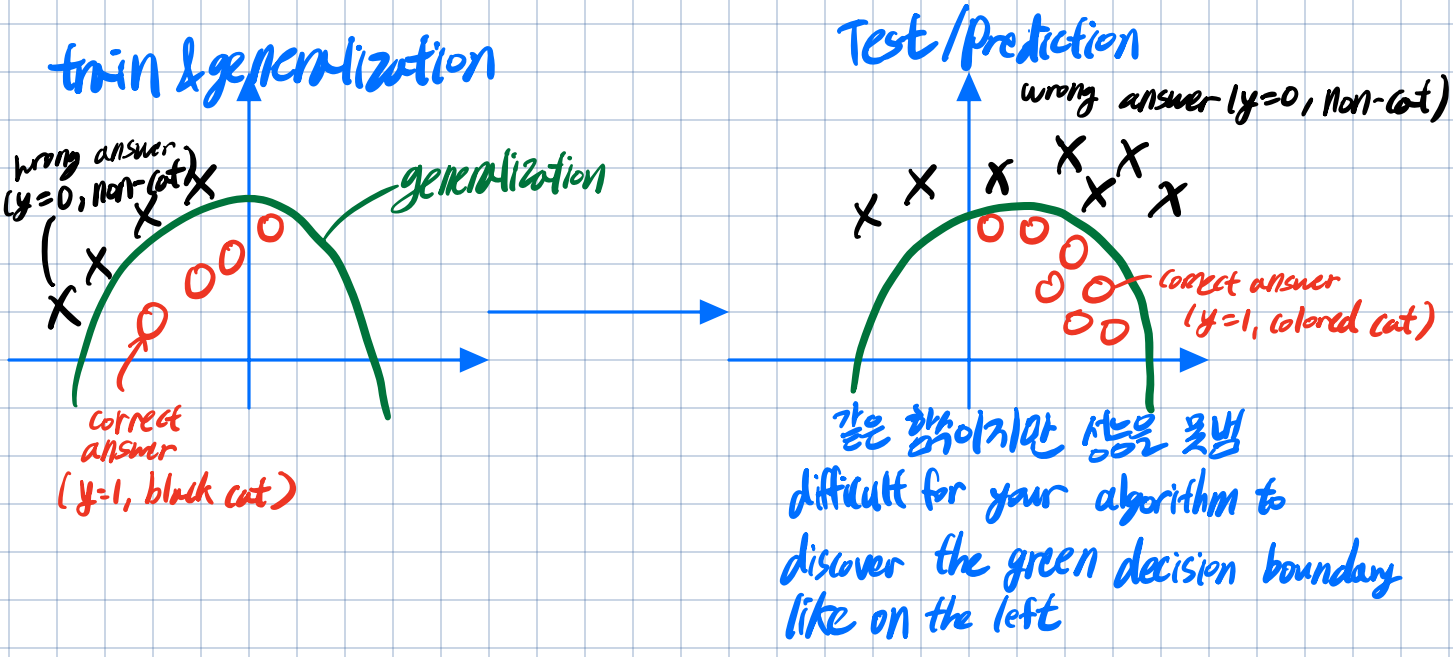
NN에서 앞쪽 층의 가중치 변화에 영향을 받는 정도

Shallow layer > deeper layer

e.g. logistic regression (or deeper model)을 같은 고양이 사진으로 훈련 $\rightarrow$

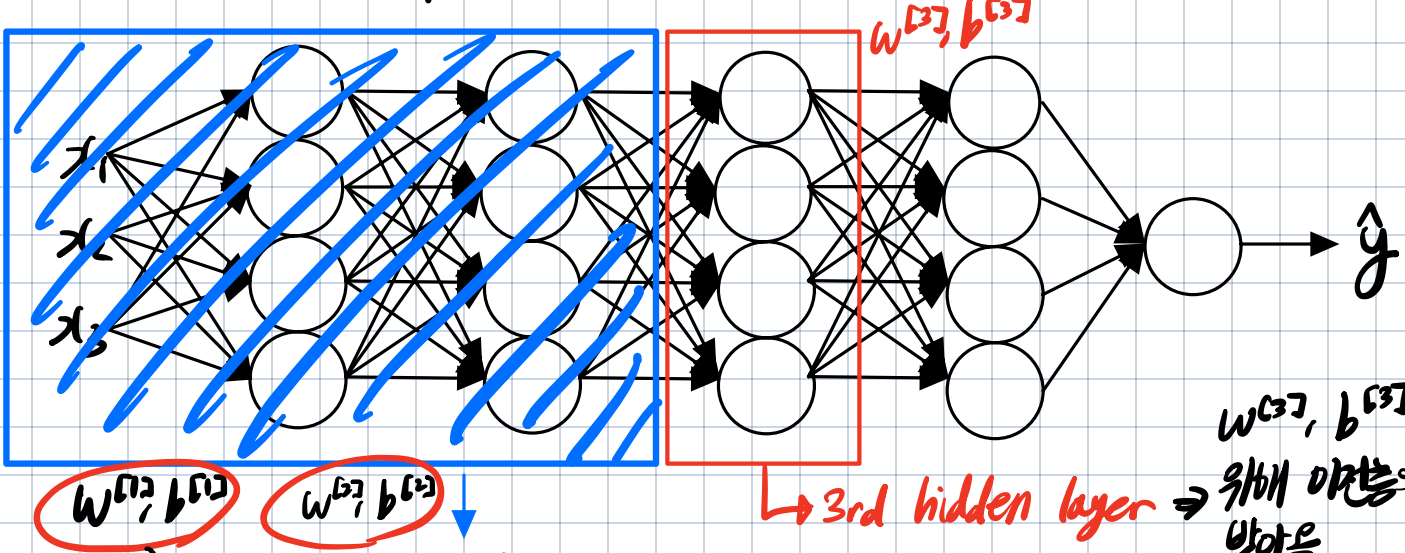
위색고양이 데이터에 적용 ⇒ low performance

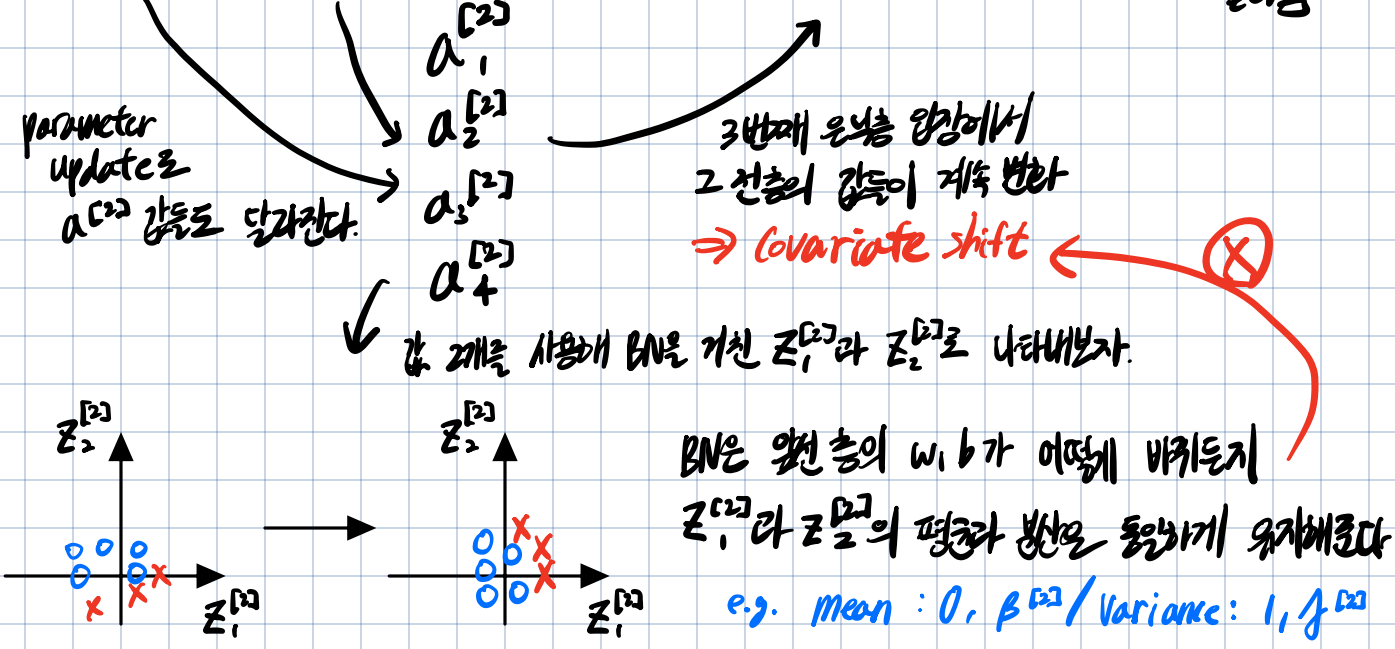



⇒ X → Y 학습 시 X의 분포가 바뀌면 (covariate shift) 알고리즘 재학습 필요

— Why this is a problem with neural networks?






⇒ BN은 은닉층 값들의 분포가 변화하는 양을 줄여준다.

① 즉, 앞전 층에서 parameter가 바뀌었을 때 3rd hidden layer에 간 값의 분포를 제한해서 covariate shift를 안정화

② 앞층 parameter - 뒷층 parameter 간 상관관계↓ ⇒ 뒷층 학습에 용이

## – Batch Norm as regularization

① Each mini-batch is scaled by the mean/variance computed on just that mini-batch

⇒ X에 비해 $X^{\{t\}}$ ($n = 64, 128, \cdots$)가 더 작은 표본이라 noise가 존재

∴ mini-batch size ↑ ⇒ noise↓ ⇒ regularization effect↓

② 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

⇒ noise가 있는 $\mu$와 $\sigma^2$이용하기 때문에 $z \to \tilde{z}^{[l]}$ 과정에도 noise +

(dropout은 확률에 따라 $a^{[l]} \times 0$ or $a^{[l]} \times 1$ ⇒ 곱셈 작용
BN은 곱셈 작용($\sigma^2$) & 덧셈 작용($\mu$))

③ This has a light regularization effect

⇒ 은닉에 덧붙여서 여러 은닉층의 하나의 은닉에 대해 의존하지 않도록 함
(but, noise가 작아서 일반화 효과가 크지는 X, 강한 효과를 원하면 dropout과 함께 사용)

## • Batch Norm at Test time

| BN | Test |
|---|---|
| One mini-batch | a sample one at a time |

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

Sum over $z$'s in one mini-batch ($n$=64, 128...)

$$z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$
$$\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$$

There is no mini-batch in test set

Come up with some separate estimates of $\mu$ & $\sigma^2$ from training set.

$\mu, \sigma^2$: estimate using exponentially weighted average across the mini-batches

layer $l$:

$X^{\{1\}}, X^{\{2\}}, X^{\{3\}}$ → $\theta_1, \theta_2, \theta_3$

track $\mu^{\{t\}[l]}$ & $\sigma^{2\{t\}[l]}$ during training

$\mu^{\{1\}[l]}$ $\mu^{\{2\}[l]}$ $\mu^{\{3\}[l]}$ ⇒ $\mu$

$\sigma^{2\{1\}[l]}$ $\sigma^{2\{2\}[l]}$ $\sigma^{2\{3\}[l]}$ ⇒ $\sigma^2$

$$z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$
$$\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$$

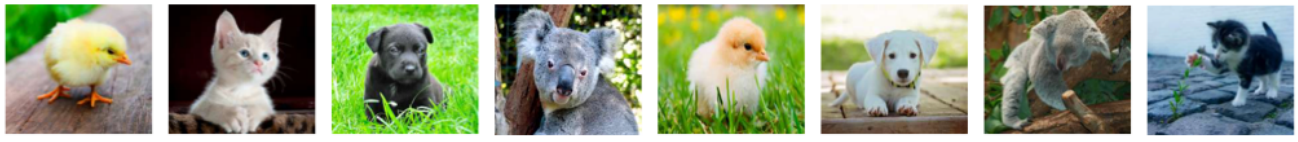
exponentially weighted averages

## 3. Multi-class classification

### • Softmax regression

— BN은 한 번에 하나의 mini-batch의 $\mu$ & $\sigma^2$ 계산 ⇒ test에서는 mini-batch X
⇒ 예측을 잘하도록 다른 접근 필요

— Recognizing cats, dogs, and baby chicks

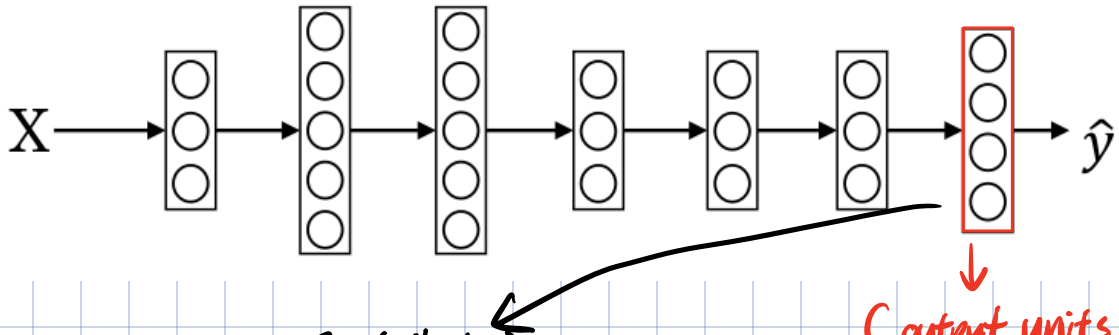


3 1 2 0 3 2 0 1

→ none of the classes

C = # of classes = 4 (label: 0 ~ C-1)

## - Softmax layer




C output units

$n^{[L]} = C = 4$

$$y = (4,1) = \begin{bmatrix} P(\text{other}|X) \\ P(\text{cat}|X) \\ P(\text{dog}|X) \\ P(\text{bc}|X) \end{bmatrix} \rightarrow \text{Sum to 1}$$

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

e.g. $z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$

Activation function for softmax:

$t = e^{z^{[L]}}$ (element-wise)

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

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

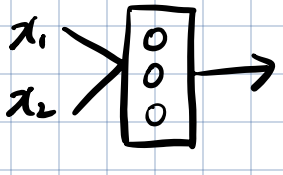
전체 1이 되도록 normalize

$$a^{[L]} = \frac{t}{176.3} = \begin{bmatrix} \frac{e^5}{176.3} = 0.842 \\ \frac{e^2}{176.3} = 0.042 \\ \frac{e^{-1}}{176.3} = 0.002 \\ \frac{e^3}{176.3} = 0.114 \end{bmatrix}$$

$\Rightarrow a^{[L]} = g^{[L]}(z^{[L]})$ $\quad (4,1) \quad (4,1)$

Relu, sigmoid가 입/출력이 모두 실수인 반면, 입/출력값이 모두 vector

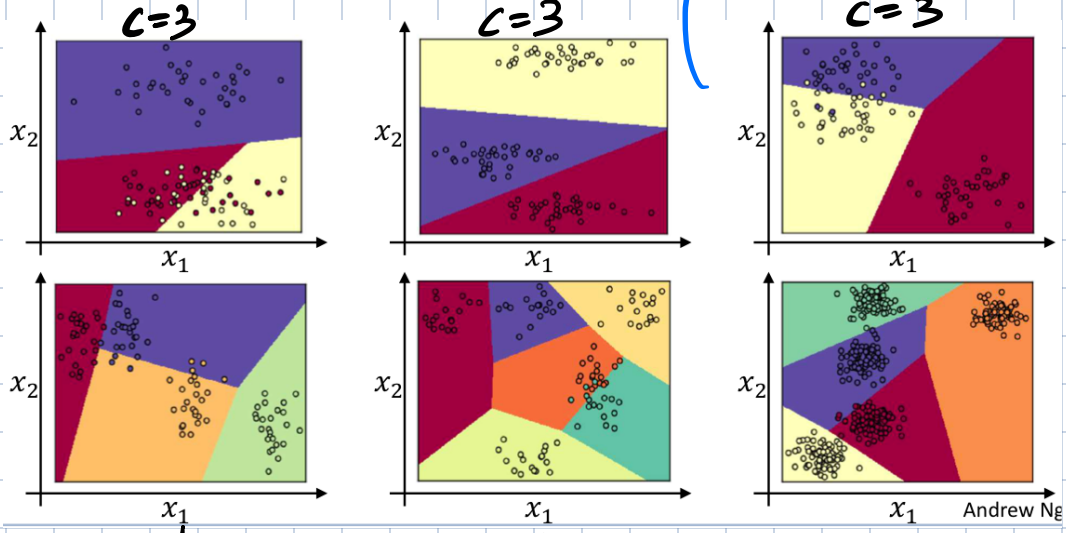
## - Softmax examples (w.o. hidden layer)




$z^{[1]} = w^{[1]} x + b^{[1]}$

$a^{[1]} = \hat{y} = g(z^{[1]})$

hidden unit X ⇒ linear decision boundary




C=4 C=6 C=5

## • Training a softmax classifier

### – Understanding softmax

$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$ $t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$ $C=4$ $g^{[L]}$ ↦ 원소 합이 1이 되도록 정규화

$a^{[L]} = g^{[L]}(z^{[L]}) = \begin{bmatrix} e^5/(e^5+e^2+e^{-1}+e^3) \\ e^2/(\quad\quad\quad) \\ e^{-1}/(\quad\quad\quad) \\ e^3/(\quad\quad\quad) \end{bmatrix} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$ Softmax $\quad \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ hard max

⇒ Softmax regression generalizes logistic regression to C classes (rather than just 2 classes)

(∵ If $C=2$ ⇒ softmax reduces to logistic Regression $a^{[L]} = \begin{bmatrix} 0.842 \\ 0.158 \end{bmatrix}$ → 합하면 여전히 1 ⇒ 하나만 계산해도 ok)

### – Loss function

$y = \begin{bmatrix} y_1 0 \\ y_2 1 \\ y_3 0 \\ y_4 0 \end{bmatrix}$ ⇒ cat

$\hat{y}^{(i)} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix} = a^{[L](i)}$ → our prediction, not working well on this example.

for one training sample: $\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{4} y_j \log \hat{y}_j$

minimize by GD

Since $y_1 = y_3 = y_4 = 0$, $\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{4} y_j \log \hat{y}_j = -\log \hat{y}_2$ ⇒ should make $\hat{y}_2$ big $(0 \le \hat{y}_2 < 1)$

⇒ Loss function은 훈련 세트에서 얼마나 큰 클래스가 되도록 하는 것이 그 클래스의 확률을 최대한 크게 만들

(= 최대우도추정)

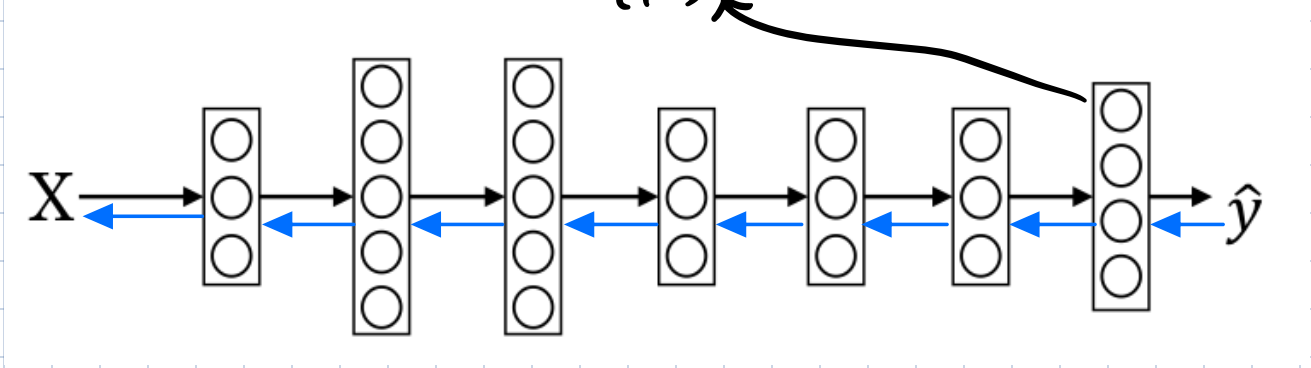
for entire training set: $J(w^{[1]}, b^{[1]}, \dots) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$

$$Y = [y^{(1)}\ y^{(2)} \dots y^{(m)}] = \begin{bmatrix} 0 & 0 & 1 & \dots \\ 1 & 0 & 0 & \\ 0 & 1 & 0 & \end{bmatrix} \quad (4, m)$$

$$\hat{Y} = [\hat{y}^{(1)}\ \hat{y}^{(2)} \dots \hat{y}^{(m)}] = \begin{bmatrix} 0.3 & \dots \\ 0.2 & \\ 0.1 & \\ 0.4 & \end{bmatrix} \quad (4, m)$$

- Gradient descent with softmax

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




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

# 4. Programming Frameworks

## • Deep Learning frameworks

- Caffe/Caffe2
- CNTK
- DL4J
- Keras
- Lasagne
- mxnet
- PaddlePaddle
- TensorFlow
- Theano
- Torch

Choosing deep learning frameworks
- Ease of programming (development and deployment)
- Running speed
- → Truly open (open source with good governance)