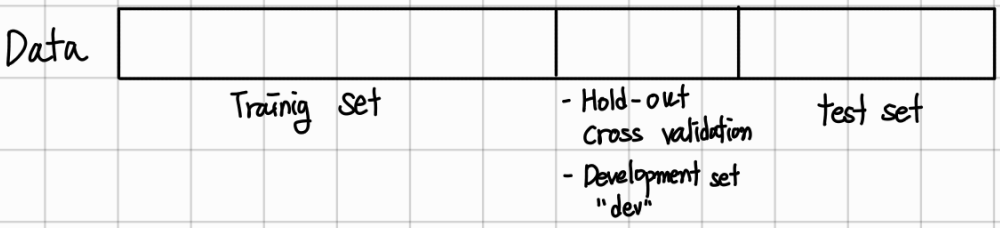
Chapter 2. Improving Deep Neural Network

1. Setting up your ML application

① Train / dev / Test sets

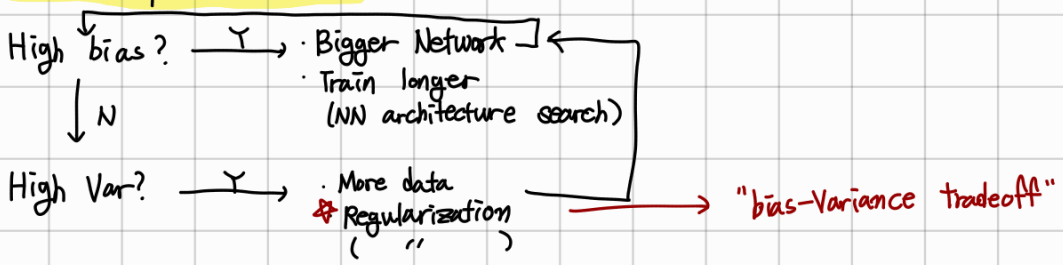


- Mismatched train/test distribution
  - Training set : Cat pictures from webpages
  - Dev/test sets : Cat pictures from users using your app
  - ⇒ Make sure dev and test come from same distribution
- Not having a test set might be okay. (Only dev set)

② Bias / Variance

- Train set error : 1% → high var. | 15% → high bias | 15% / 30% → high bias, high var. | 0.5% / 1% → low bias, low var.
- Dev set error : 11%
- Human : ≈ 0% → Optimal (Bayes) error : ≈ 0%

③ Basic recipe for ML



2. Regularizing your neural network

⓪ Frobenius norm

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

  행렬의 행 i : 현재 레이어의 뉴런수 $n^{[l]}$

  〃 열 j : 이전 레이어의 뉴런수 $n^{[l-1]}$

① Regularization

- Logistic regression

  $\min_{w,b} J(w,b)$, $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

  $J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w\|_2^2 + \overbrace{\frac{\lambda}{2m}b^2}^{\text{omit}}$, λ = regularization parameter

  L_2 regularization $\|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$

  L_1 regularization $\frac{\lambda}{2m}\sum_{i=1}^{n_x}|w| = \frac{\lambda}{2m}\|w\|_1$, w will be sparse

- Neural Network

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

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

  $\rightarrow w^{[l]} := w^{[l]} - \alpha\, dw^{[l]}$, $\frac{dJ}{dw^{[l]}} = dw^{[l]}$

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

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

② Why regularization reduces overfitting

- $\lambda \uparrow$: $w^{[l]} \downarrow$

  $z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$ → z가 상대적으로 작아짐.

- $J(\cdots) = \sum L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_l \|W^{[l]}\|_F^2$

구분	Lasso	Ridge
수식	$\|w\|_1$ 	$\frac{1}{2}\|w\|^2$ 
미분	$\begin{cases} 1 & w<0 \\ -1 & w>0 \end{cases}$ 	w 
특성	• 가중치 값을 정확하게 0으로 만듦 • 중요한 특징을 '선택'하는 효과 • 모델에 Sparsity를 가함.	• 큰 가중치의 값을 작게 만듦 • 모델 전반적인 복잡도를 감소시키는 효과 • 가중치의 값이 0이 되게 하지는 못함

③ Dropout regularization

- 신경망의 노드와 링크 제거
- Illustrate with layer $l=3$, keep_prob = 0.8

```
d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keep_prob
a3 = np.multiply(a3, d3)        # a3 *= d3
a3 /= keep_prob
```

50 units → 10 units shut off

$z^{[4]} = w^{[4]} a^{[3]} + b^{[4]}$

- No drop-out

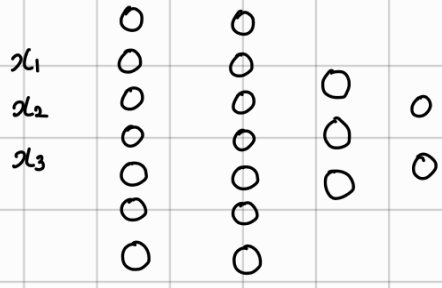
$z^{[1]} = w^{[1]} a^{[0]} + b^{[1]}$

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

$z^{[2]} = w^{[2]} a^{[1]} + b^{[2]}$

$a^{[2]} = \cdots$

④ Understanding Dropout

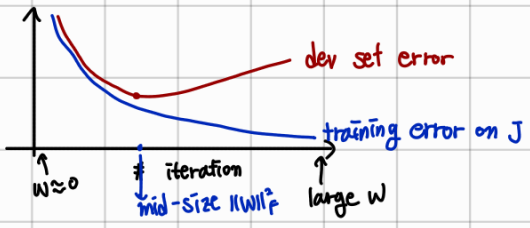
- 한 가지 feature에 의존하면 안 됨.
- Dropout은 L2 정규화와 유사
- 

⇒ 링크 생략.

$W^{[1]} = 7 \times 3$, $W^{[3]} = 3 \times 7$

⑤ Other regularization methods

- Early Stopping



단점 : 비용함수 최적화, 정규화를 한 번에 해결할 수 없음.

3. Setting up your optimization problem

① Normalizing inputs

- Mean $= \mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$, $x := x - \mu$
- Variance $= \sigma^2 = \frac{1}{m}\sum_{i=1}^{m} x^{(i)**2}$, $x /= \sigma$

- Use same μ, σ to normalize test set

② Vanishing / exploding gradients

- $y = W^{[L]}W^{[L-1]}W^{[L-2]} \cdots W^{[3]}W^{[2]}\underbrace{W^{[1]}x}$

$z^{[1]} = W^{[1]}x$

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

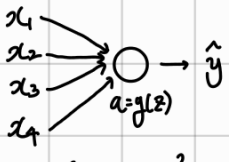
$a^{[2]} = g(z^{[2]}) = g(W^{[2]}a^{[1]})$

- 해결을 위해 가중치 초기화가 중요!
- 기울기 소실 : 역전파 과정에서 입력층으로 갈수록 기울기가 점차 작아지는 현상

  → 입력층에 가까운 층들에서 가중치들이 업데이트 되지 않음.
- 기울기가 입력층으로 갈수록 점차 커지다가 가중치들이 비정상적으로 큰 값이 되며 발산하는 현상

③ Weight initialization for deep networks

- Single neuron example

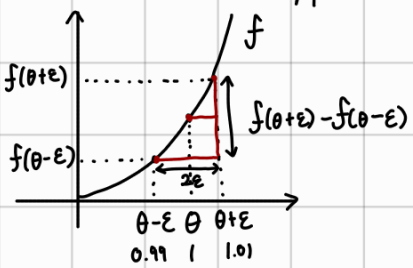


$z = w_1x_1 + w_2x_2 + \cdots + w_nx_n$

large n → smaller w_i

- $Var(w_i) = \frac{2}{n}$, $w^{[l]} = np.random.randn(slope) * np.sqrt\left(\frac{2}{n^{[l-1]}}\right)$ → ReLU
- tanh ⇒ $\sqrt{\frac{1}{n^{[l-1]}}}$: Xavier initialization $\sqrt{\frac{2}{n^{[l-1]}+n^{[l]}}}$

④ Numerical approximation of gradients



- $\frac{f(\theta+\varepsilon)-f(\theta-\varepsilon)}{2\varepsilon} \approx g(\theta)$
- $\frac{(1.01)^3-(0.99)^3}{2(0.01)} = 3.0001$

$g(\theta)=3\theta^2=3$

- $f'(\theta)=\lim_{\varepsilon\to 0}\frac{f(\theta+\varepsilon)-f(\theta-\varepsilon)}{2\varepsilon}$

approx error : 0.0001

⑤ Gradient Checking

- $J(W^{[1]}, b^{[1]}, \cdots, W^{[L]}, b^{[L]}) = J(\theta)$

  ⇒ Is $d\theta$ the gradient of $J(\theta)$?

- Algorithm For each i :

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

$$\approx d\theta[i] = \frac{\partial J}{\partial \theta_i} \quad \Big| \quad d\theta_{approx} \overset{?}{\approx} d\theta$$

Check $\frac{\|d\theta_{approx}-d\theta\|_2}{\|d\theta_{approx}\|_2+\|d\theta\|_2}$ $\approx$ 10^{-7} – great!

10^{-5}

10^{-3} – worry.

⑥ Gradient Checking implementation notes.

- Don't use in training – only to debug
- If algorithm fails grad check, look at components to try to identify bug.
- Remember regularization
- Doesn't work with dropout.
- Run at random initialization ; perhaps again after some training

4. Optimization Algorithms

① Mini-batch gradient descent

- 미니배치 : 훈련세트를 다시 나누어 만든 훈련 세트

ex) $X = [\underbrace{x^{(1)}\ x^{(2)}\ x^{3} \cdots x^{(1000)}}_{\text{mini-batch}} \mid \underbrace{x^{(1001)}\ x^{(1002)} \cdots x^{(2000)}}_{\text{mini-batch}} \mid \cdots \mid \cdots x^{(m)}]$

mini-batch t : $X^{\{t\}}, Y^{\{t\}}$ / $X^{\{1\}}$의 dimension : $(n_x, 1000)$ / $Y^{\{1\}}$ 〃 : $(1, 1000)$

- Mini-batch gradient descent

for $t = 1, \cdots, 5000$

Forward prop on $X^{\{t\}}$

$$Z^{[1]} = W^{[1]}X^{\{t\}} + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$\vdots$$
$$A^{[L]} = g^{[L]}(Z^{[L]})$$

} Vectorized implementation

Compute Cost $J = \frac{1}{1000}\sum_{i=1}^{l} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2\cdot 1000}\sum_{l}\|W^{[l]}\|_F^2$, l : mini-batch 샘플 수

Backprop to compute gradients cost $J^{\{t\}}$ (using $(X^{\{t\}}, Y^{\{t\}})$)

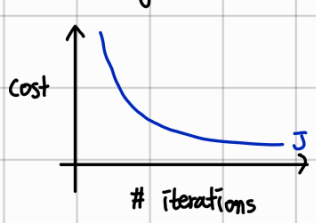
$W^{[l]} := W^{[l]} - \alpha dW^{[l]}$, $b^{[l]} := b^{[l]} - \alpha db^{[l]}$

repeat

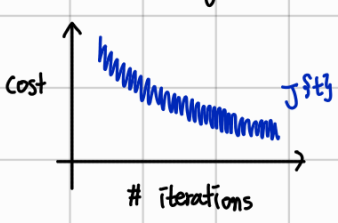
- epoch : 훈련세트로 1번 통과

② Understanding mini-batch gradient descent

- Batch gradient descent



- Mini-batch gradient descent



- Choose mini-batch size

If mini-batch size=m : Batch Gradient descent

If mini-batch size=1 : Stochastic gradient descent

Batch gradient descent (mini-batch size=m) → Too long per iteration

Stochastic gradient descent → Lose speed up from vectorization.

⇒ In-between(size not too big/small) → fastest learning

- recommendation

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

Typical mini-batch sizes : 2의 거듭제곱 (64, 128, 256, 512)

Make sure mini-batch fit in CPU / GPU memory.

③ Exponentially weighted averages

- $v_t = \beta \cdot v_{t-1} + (1-\beta)\theta_t$ ← 지수 가중 이동 평균

$\beta = 0.9$, v_t approximately average $\approx \frac{1}{1-\beta}$

ex) $v_{100} = 0.1\theta_{100} + 0.9 v_{99}$

$= 0.1\theta_{100} + 0.1 \times 0.9 \cdot \theta_{99} + 0.1(0.9)^2\theta_{98} + 0.1(0.9)^3\theta_{97} + 0.1(0.9)^3\theta_{96}$

$0.9^{10} \approx 0.35 \approx \frac{1}{e}$, $\frac{(1-\varepsilon)}{0.9} = \frac{1}{e}$

- Implementing exponentially weighted averages

$v_0 = 0$

$v_1 = \beta v_0 + (1-\beta)\theta_1$

$v_2 = \beta v_1 + (1-\beta)\theta_2$

$v_3 = \beta v_2 + (1-\beta)\theta_3$

$v_0 := 0$

$v_\theta := \beta v + (1-\beta)\theta_1$

$v_\theta := \beta v + (1-\beta)\theta_2$

$v_\theta = 0$

Repeat {

Get next θ_t

$v_\theta := \beta v_\theta + (1-\beta)\theta_t$ }

④ Bias correction in exponentially weighted average

- 처음 데이터에는 이전의 데이터가 존재하지 않으므로 처음 데이터로 0에 대한 가중 평균을 구하게 되면 값이 매우 작아짐.

⇒ 이를 수정하는 방법이 Bias correction

- v_t 대신 $\frac{v_t}{1-\beta^t}$ 사용, 추정치 : $\frac{\beta v_{t-1} + (1-\beta)\theta_t}{1-\beta^t}$

⑤ Gradient descent with momentum

- Momentum :

On iteration t :

Compute dW, db on correct mini-batch.

$v_{dW} = \beta v_{dw} + (1-\beta)dW$ "$v_\theta = \beta v_\theta + (1-\beta)\theta_t$"

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

$W = W - \alpha v_{dw}$, $b := b - \alpha v_{db}$

⑥ RMS prop (Root Mean Square prop)

On iteration t:

Compute dw, db on correct mini-batch

$S_{dw} = \beta S_{dw} + (1-\beta) dw^2$ ← element-wise ← small

$S_{db} = \beta S_{db} + (1-\beta) db^2$ ← large

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

⑦ Adam (Adoption moment estimation) optimization algorithm

- $V_{dw} = 0$, $S_{dw} = 0$, $V_{ab} = 0$, $S_{ab} = 0$

On iteration t:

Compute dw, db using correct mini-batch

$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dw$, $V_{ab} = \beta_1 V_{db} + (1-\beta_1) db$ ← "momentum" β_1

$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dw^2$, $S_{ab} = \beta_2 S_{db} + (1-\beta_2) db^2$ ← "RMS prop" β_2

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

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

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

- Hyperparameters choice

α : needs to be tune

β_1 : 0.9 $\quad (dw) \rightarrow$ 1차 모멘트

β_2 : 0.999 $\quad (dw^2) \rightarrow$ 2차 모멘트

ε : 10^{-8}

⑧ Learning Rate Decay

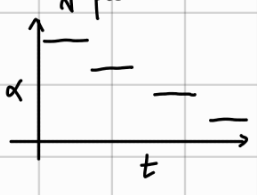
- $\alpha = \frac{1}{1 + \text{decayRate} \times \text{epochNumber}} \alpha_0$
- 1 epoch = 1 pass through data

Epoch	α
1	0.1
2	0.067
3	0.05
4	0.04

- Other learning rate decay methods

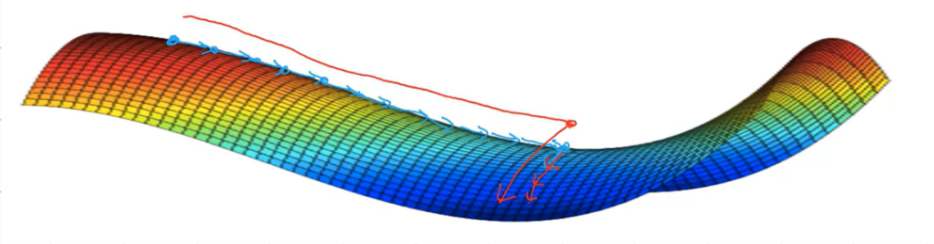
$\alpha = 0.95^{\text{epoch-num}} \cdot \alpha_0$ - exponentially decay

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



⑨ The problem of local optima

- 안장점 (Saddle point) : 비용함수에서 기울기가 0인 지점
- problem of plateaus

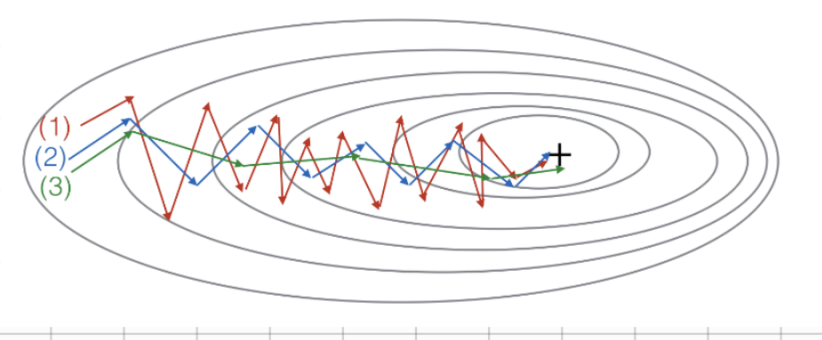


- Unlikely to get stuck in a bad local optima
- Plateaus can make learning slow

  ⇒ Momentum, RMSprop, Adam이 도움이 될 수 있음.

✱ 퀴즈

- $a^{[2]\{4\}(3)}$: 네 번째 미니배치, 세번째 예일 때, 두번째 레이어
- $v_2 = \beta v_{t-1} + (1-\beta)\theta_t$ ⇒ 편향 보정 : $\frac{v_t}{1-\beta^t}$
- 곡선의 noise가 심한경우 β 값이 더 작음.



(1) : 경사 하강

(2) : 모멘텀이 있는 경사하강 (small β)

(3) : 모멘텀이 있는 경사 하강 (large β)

- NN에서 J가 작은 값에 도달하도록 할 수 있는 기법

  1) 모멘텀이 있는 경사 하강 사용

  2) 입력 데이터 정규화

  3) 가중치에 대한 더 나은 무작위 초기화 시도.

5. Hyperparameter Tuning

① Tuning process

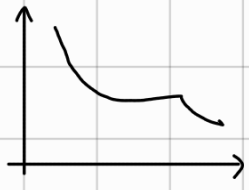
- Hyperparameters
  1) α
  2) β (default: 0.9)
  3) $\beta_1, \beta_2, \varepsilon$ → Adam 사용하는 경우 0.9, 0.999, 10^{-8}
  4) # layers
  5) # hidden units
  6) learning rate decay
  7) mini-batch size
- grid를 사용하지 말고, 랜덤하게 배치하여 하이퍼파라미터 선택

② Using an appropriate scale to pick hyperparameters.

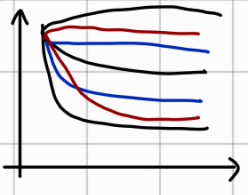
- $n^{[l]} = 50, \cdots, 100$
- # layers L : 2-4
- $\alpha = 0.0001, \cdots, 1$ ⇒ $r = -4 * np.random.randn()$, $r \in [-4, 0]$, $\alpha = 10^r$
- $\beta = 0.9, \cdots, 0.999$

③ Hyperparameters tuning in practice : Pandas vs Caviar

- Babysitting one model vs Training many models in parallel



⇒ panda



→ Caviar

6. Batch Normalization

① Normalizing activations in a network

- $\mu = \frac{1}{m}\sum_i x^{(i)}$, $\quad X = X - \mu$

  $\sigma^2 = \frac{1}{m}\sum_i X^{(i)2}$, $\quad X = X/\sigma$

- Batch Norm

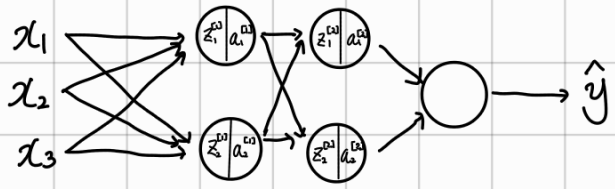
  $\mu = \frac{1}{m}\sum_i z^{(i)}$, $\quad \sigma^2 = \frac{1}{m}\sum_i (z^{(i)}-\mu)^2$, $\quad z^{(i)}_{norm} = \frac{z^{(i)}-\mu}{\sqrt{\sigma^2+\varepsilon}}$

  $\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$ (γ, β : learnable parameters of Model)

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

② Fitting Batch Norm into a neural network.

- Adding Batch Norm to a network



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

Parameters : $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}, \cdots, w^{[L]}, b^{[L]} + \beta^{[1]}, \gamma^{[1]}, \beta^{[2]}, \gamma^{[2]}, \cdots \beta^{[L]}, \gamma^{[L]}$

$\rightarrow d\beta^{[l]}$, $\quad \beta^{[l]} = \beta^{[l]} - \alpha d\beta^{[l]}$

- Working with mini-batches

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

$X^{\{2\}} \rightarrow z^{[1]} \xrightarrow[BN]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \rightarrow \cdots$

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

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

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

$z^{[l]} \Rightarrow (n^{[l]}, 1)$

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

- Implementing gradient descent

  for $t=1, \cdots,$ # Mini-Batches

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

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

  Use backprop to compute $dw^{[l]}$, $db^{[l]}$, $d\beta^{[l]}$, $d\gamma^{[l]}$

  Update parameters $w^{[l]} := w^{[l]} - \alpha dw^{[l]}$

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

  $\gamma^{[l]} := \cdots$

  Work w/ momentum, RMS prop, Adam, ...

③ Why does Batch Norm work?

- Internal Covariance Shift

  네트워크의 각 레이어나 Activation 마다 입력값의 분산이 달라지는 현상

  ⇒ Covariate Shift : 이전 레이어의 파라미터 변화로 인하여 현재 레이어의 입력의 분포가 바뀌는 현상

  ⇒ Internal Covariate Shift : 레이어를 통과할 때 마다 Covariate Shift가 일어나면서 입력의 분포가 약간씩 변하는 현상

- 각 미니 배치가 하나의 미니 배치에서 계산된 평균과 분산으로 조정됨. → 약간의 노이즈 존재
- 미니 배치에서 $z^{[l]}$로 조정 시 노이즈가 생김. dropout과 비슷하게 각 hidden layer's activation에 noise가 생김
- 약간의 정규화 효과가 존재함.

④ Batch Norm at test time

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

  $X^{\{1\}}, X^{\{2\}}, X^{\{3\}}, \cdots$

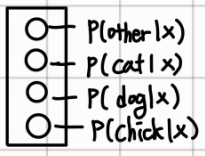
  $\mu^{\{1\}[l]} \quad \mu^{\{2\}[l]} \longrightarrow \mu$

- $z_{norm} = \frac{z-\mu}{\sqrt{\sigma^2+\varepsilon}}$, $\quad \tilde{z} = \gamma z_{norm} + \beta$

7. Multi-class classification

① Softmax regression

Example) 클래스 4개 (0-어느곳에도 속하지 않는 경우, 1-고양이, 2-개, 3-병아리)

마지막 레이어 

$\hat{y} \Rightarrow (4,1)$, $n^{[L]} = 4 = C$

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

Activation function : $t = e^{(z^{[L]})}$, $a^{[L]} = \frac{e^{z^{[L]}}}{\sum_{i=1}^{4} t}$

ex) $z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$, $t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \\ 20.1 \end{bmatrix}$, $\sum_{i=1}^{4} t_i = 176.3$

layer L ⇒ 1) $\frac{e^5}{176.3} = 0.842$, 2) $\frac{e^2}{176.3} = 0.042$, 3) $\frac{e^{-1}}{176.3} = 0.002$, 4) $\frac{e^3}{176.3} = 0.114$

- 신경망 결과값 : $a^{[L]} = g^{[L]}(z^{[L]})$ or $\hat{y}$

② Training a softmax classifier

- Softmax 회귀는 로지스틱 회귀를 C개의 class로 일반화함.
- Loss function $\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{4} y_j \log \hat{y}_j$

  $J(w^{[1]}, b^{[1]}, \cdots) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$
- $z^{[L]} \rightarrow a^{[L]} = \hat{y} \rightarrow \mathcal{L}(\hat{y}, y)$
- Backprop : $dz^{[L]} = \hat{y} - y$

8. Programming frameworks

① Deep Learning frameworks

- Caffe / Caffe 2
- 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)