[3-1. Hyperparameter Tuning]

< Tuning Process >

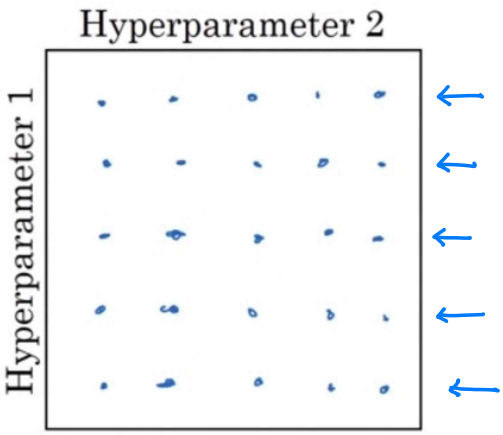
* 하이퍼 파라미터 튜닝 중요 순위

1순위) 학습률 (α)

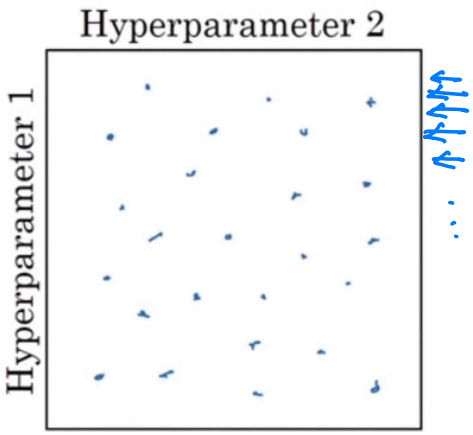
2순위) 모멘텀(momentum)의 β ~0.9
mini-batch size
hidden units

3순위) # layers
learning rate decay
Adam 알고리즘의 $\beta_1, \beta_2, \varepsilon$
0.9 0.999 10^{-8}

방법 1)



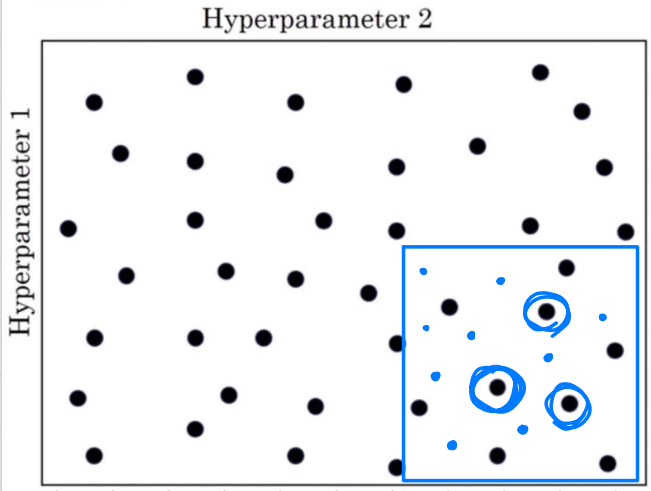
방법 2)



- 방법 1의 경우, 하이퍼파라미터의 개수가 비교적 많지 않을 때 잘 작동하는 편
- 방법 2의 무작위 접근 방식이 더 좋다. 어떤 하이퍼파라미터가 문제 해결에 더 중요한지 알 수 없기 때문.

ex) hyperparameter 1: α, hyperparameter 2: ε 이라면,
즉, hyperparameter의 하나는 수치 조정에 중요한 값이고 하나는 아니라면,
방법 1에서는 중요한 α를 9번 밖에 시험을 못하지만 방법 2는 많이 해서 더 값을 잘 찾음

방법3)



- 방법3은 전체 하이퍼파라미터 공간에서 탐색하여 좋은 점을 찾은 후, 그 근방에서 더 정밀하게 탐색

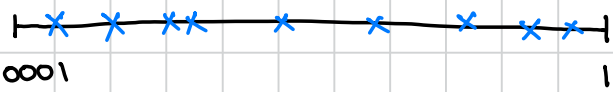
<Using an Appropriate Scale to pick Hyperparameters>

ex1) # hidden units : 50 , ··· , 100

layers : 2 ~ 4

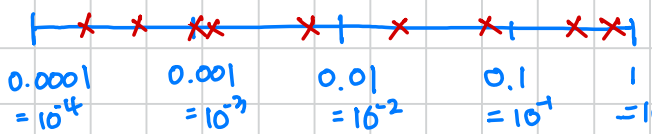
∴ #hidden units , # layers는 균등척도를 이용하여 무작위로 뽑는 것이 합리적!

ex2) α = 0.0001 , ··· , 1



→ 90%의 값이 0.1 ~ 1 사이의 값

로그척도 ⟹



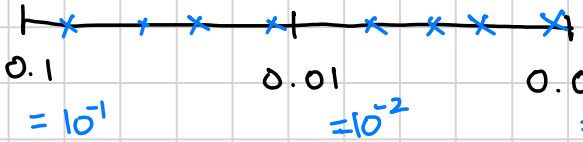
→ r = -4 * np.random.rand()

$\alpha = 10^{r}$

∴ 학습률(α)는 로그척도를 이용하여 무작위로 뽑는 것이 합리적!

ex3) β = 0.9 , ··· , 0.999

↳ 1-β = 0.1 , ··· , 0.001



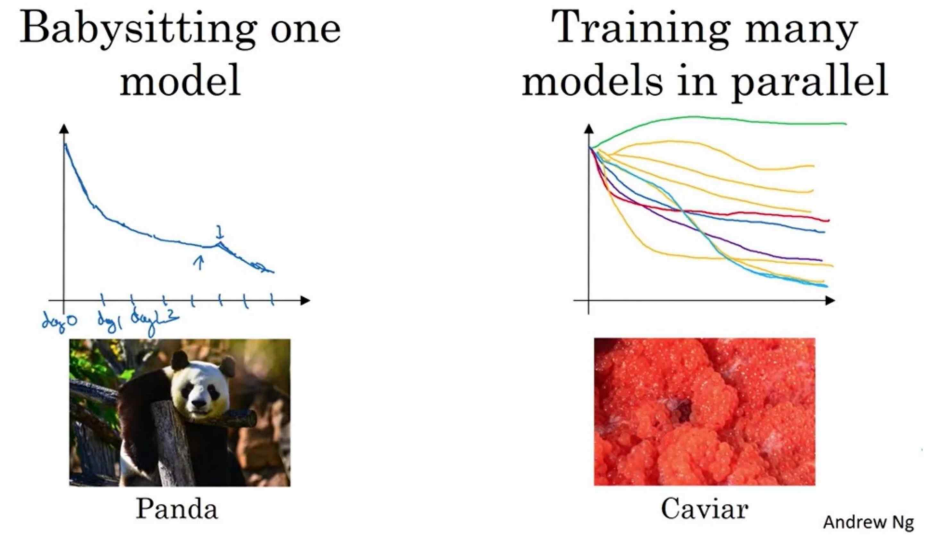
⇒ r ∈ [-3, -1]

1-β = 10^{r}

β = 1-10^{r}

∴ β는 1-β로 변환한 후, 로그척도를 이용하여 무작위로 뽑는 것이 합리적!

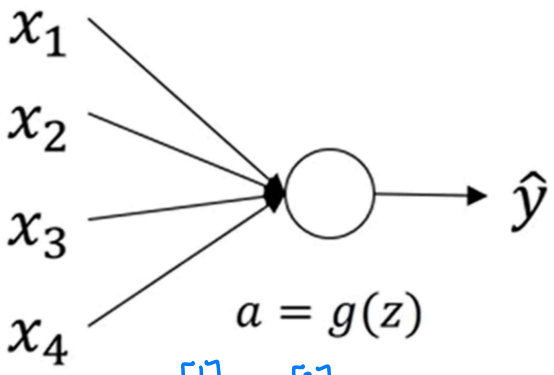
<Hyperparameter Tuning in Practice : Pandas vs Caviar>



- Babysitting one model (= 판다접근) : 하나의 모델의 성능을 학습 속도를 바꿔가면서 확인. CPU/GPU가 부족할 때
- Training many models in parallel : 여러 모델을 한번에 학습시킬 때 사용. CPU/GPU가 여유로울 때

[7-2. Batch Normalization]

<Normalizing Activation in a Network>

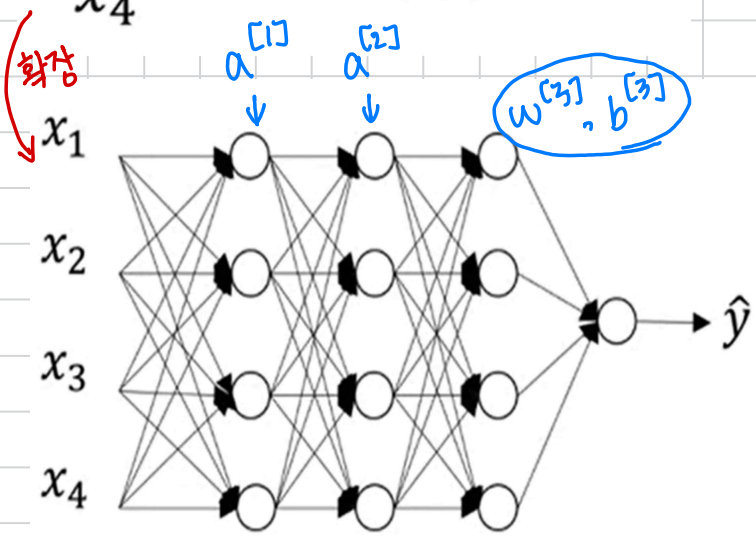


$$\mu = \frac{1}{m}\sum_i x^{(i)}$$

$$X = X - \mu$$ ← element-wise

$$\sigma^2 = \frac{1}{m}\sum_i x^{(i)2}$$

$$X = X / \sigma$$



Can we normalize $a^{[2]}$ so as to train $w^{[3]}, b^{[3]}$ faster.

기술적으로는 $a^{[2]}$가 아니라 $z^{[2]}$를 normalize함

*Implementing Batch Norm

- Given some intermediate values in NN

$z^{(1)}, \cdots, z^{(m)} \rightarrow z^{[l](i)}$

*Batch Normalization

- 배치 정규화의 장점: 하이퍼파라미터의 탐색을 쉽게 만들어주며, 신경망과 하이퍼파라미터의 상관관계를 줄여줌
- 보통 활성화 함수 이전에 사용

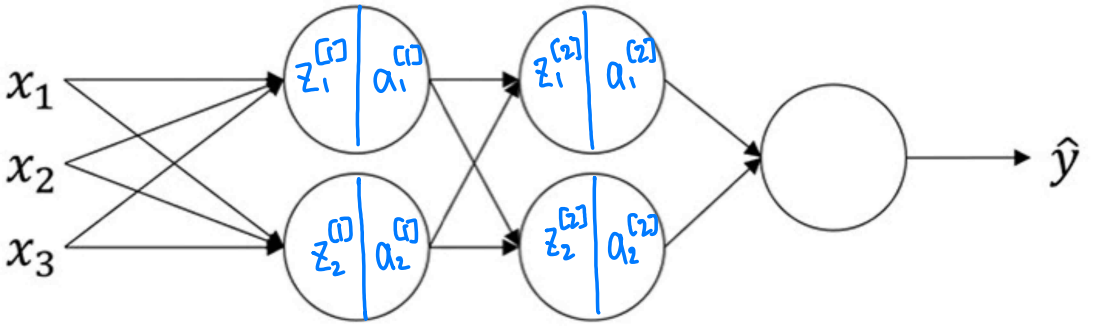
- $\mu = \frac{1}{m}\sum_i z^{(i)}$
- $\sigma^2 = \frac{1}{m}\sum_i (z^{(i)} - \mu)^2$
- $z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$
- $\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$ ← learnable parameter in model

If) $\gamma = \sqrt{\sigma^2 + \varepsilon}$, $\beta = \mu$ 라면 $\tilde{z}^{(i)} = z^{(i)}$ 이므로 적절한 γ과 β를 변화하면 위의 나머지 식은 항등함수!

즉, γ와 β를 다르게 설정하면서 hidden unit이 다른 평균과 분산을 갖도록 해야 함

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

<Fitting Batch Norm In to a Neural Network>



$X \xrightarrow{w^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[\text{batch norm (=BN)}]{\beta^{[1]}, r^{[1]}} \tilde{z}^{[1]} \xrightarrow[\text{activation function}]{} a^{[1]} = g^{[1]}(\tilde{z}^{[1]})$: 선형결합 값을 계산하고, 이것을 BN 시킨 후, 활성화함수를 거쳐 a를 구함

$\xrightarrow{w^{[2]}, b^{[2]}} z^{[2]} \xrightarrow[BN]{\beta^{[2]}, r^{[2]}} \tilde{z}^{[2]} \longrightarrow a^{[2]} \rightarrow \cdots$

*Working with mini-batches

$X^{\{1\}} \xrightarrow{w^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[\text{batch norm (=BN)}]{\beta^{[1]}, r^{[1]}} \tilde{z}^{[1]} \xrightarrow[\text{activation function}]{} a^{[1]} = g^{[1]}(\tilde{z}^{[1]}) \rightarrow \cdots$

$X^{\{2\}} \xrightarrow{w^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[\text{batch norm (=BN)}]{\beta^{[1]}, r^{[1]}} \tilde{z}^{[1]} \xrightarrow[\text{activation function}]{} a^{[1]} = g^{[1]}(\tilde{z}^{[1]}) \rightarrow \cdots$

$X^{\{3\}} \rightarrow \cdots$

⊕ Parameters : $w^{[l]}$, ~~$b^{[l]}$~~, $\beta^{[l]}$, $r^{[l]}$ 중에서 BN을 적용하면 값의 평균을 빼주기 때문에 상수항 $b^{[l]}$은 사라진다

$\therefore z^{[l]} = w^{[l]} a^{[l-1]}$, $\tilde{z}^{[l]} = r^{[l]} \cdot z_{norm} + \beta^{[l]}$

*Implementing gradient descent

for t=1 to num MiniT-Batches

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

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

Use back prop to compute $dw^{[l]}$, ~~$db^{[l]}$~~, $dr^{[l]}$, $d\beta^{[l]}$

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

Work with momentum / RMSprop / Adam

<Why does Batch Norm work?>

- 입력특성 X를 정규화하여 값들을 비슷한 범위의 값들(평균 0, 분산 1)로 만들어서 학습을 수
- 이전 층의 가중치 영향을 덜 받게 한다. 은닉층의 값의 분포의 변화를 줄여주어서, 입력값의 분포를 제한 함. 즉, 입력값이 바뀌어서 발생하는 문제인 'covariance shift' 문제를 안정화 시킴.
  ∴ 앞층과 뒷층의 매개변수의 상관관계를 줄여서 학습독도를 향상시킴 (층이 독립적)
  ↳ X→Y로 가는 mapping을 train한 경우, X의 분포가 변했을 때 알고리즘을 변경시키지 않고 ground true function에 적용되도록 하는 것
- 파라미터 정규화 (regularization)
  - Each mini-batch is scaled by the mean/variance computed on just that mini-batch. (∵ 평균과 분산이 추정치)
  - 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. (∵ dropout은 은닉유닛의 활성에 대해 0또는 1을 곱하기 때문에 곱셈 noise 존재)
  - This has a slight regularization effect.

  ↳ batch normalization도 곱셈잡음(x1or0)과 덧셈잡음(-씨)가 존재하여 regularization 효과 o
  ↳ 은닉층에 noise를 추가하는 것은 하나의 hidden unit에 너무 의존하지 않도록 하는 것

  but) regularization의 효과가 있다는 것이지, 효과가 크지는 않으며 mini batch의 size가 크면 효과가 더 적어진다.

<Batch Norm at Test Time>

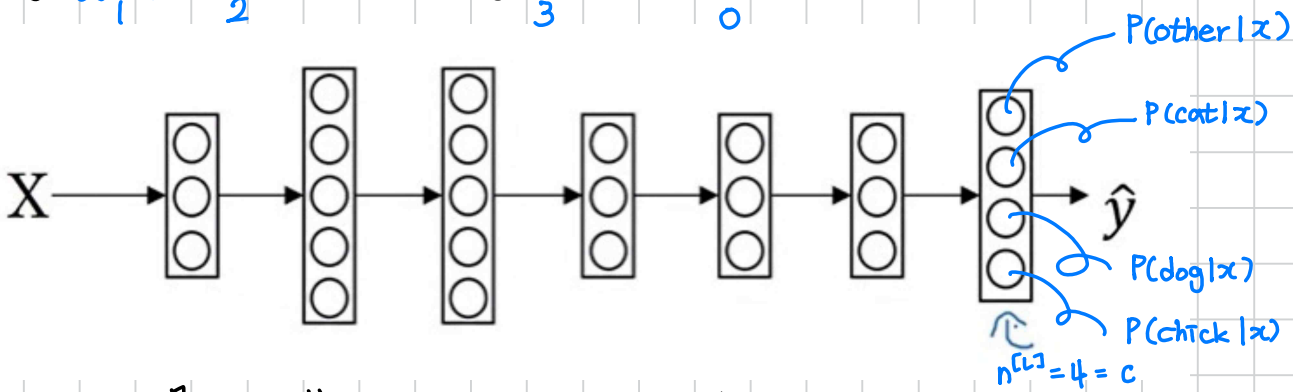
- BN은 한번에 하나의 mini-batch로 처리하지만, test 시에는 배치가 하나이기 때문에 평균과 분산을 계산할 수 없다. (한번에 하나의 예제를 처리해야 할 수도 있음)

  ∴ exponentially weighted average를 이용하여 학습 시와 0을 사용 (across mini-batch)

[3-3 Multi-class Classification]

<Softmax Regression>

* Recognizing cats, dogs, and baby chicks, other
  1 2 3 0



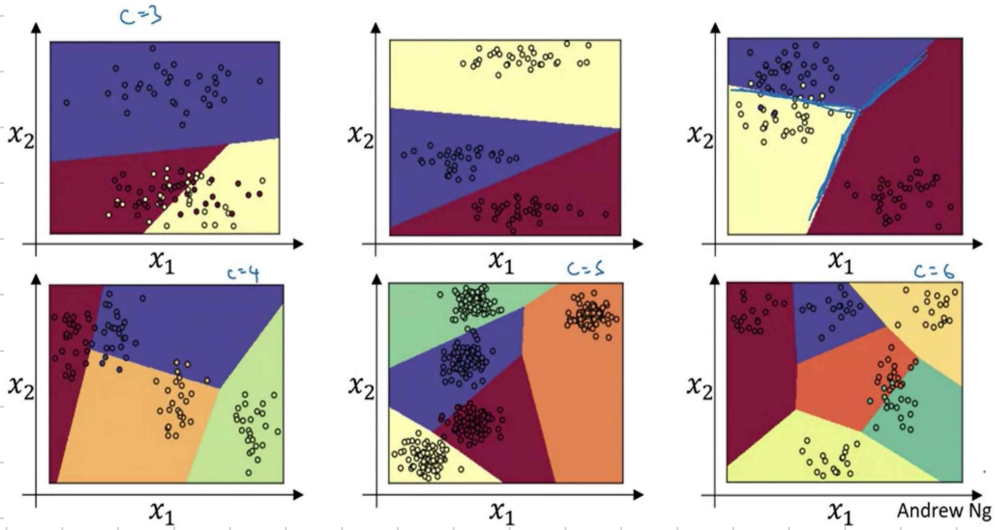
- Softmax : 여러개의 클래스 분류시 사용하며 generalize logistic regression to C classes.

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

$$t = e^{z^{[L]}}$$

$$a_i = \frac{t_i}{\sum_{i=1}^{C} t_i}$$

Softmax examples



<Training a softmax Classifier>

ex)

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

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

*Loss function

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

ex) $y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $a^{[L]} = \hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$ 의 loss function을 구해보면, $L(\hat{y}, y) = -\sum_{i=1}^{4} y_i \log \hat{y}_i = -\log \hat{y}_2$

즉, loss function을 최적화하는 것은 $\hat{y}_2$를 최대화 하는 것!

- Back prop : $dz^{[L]} = \hat{y} - y$