◦ Hyper parameter tuning

◦ Tuning Process

△ Hyperparameters:

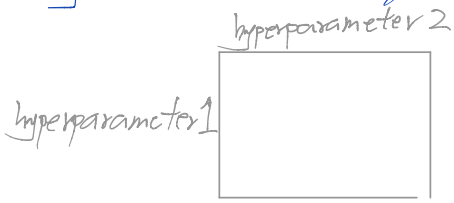
learning rate ← α, β → regularization factor, β_1, β_2, ε, # layers, # hidden units, learning rate decay, mini-batch size

□: Top importance

□: Second importance

□: 3rd importance

△ Try random values: don't use grid



hyperparameter 1 & 2 may not have ~~the~~ same importance

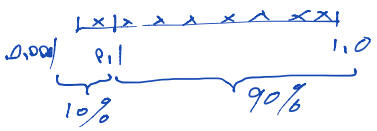
△ Corse to fine

Corse search and then focus on fine search in small region.

◦ Using appropriate scale to pick hyperparameters

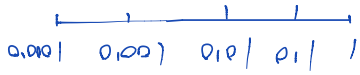
△ Eg. $\alpha = 0.001, \ldots\ldots, 1$

if use uniformly choosen random α:



only 10% resources spent on 0.001 ~ 0.1, not right

Instead, use log scale



$r = -4 * np.random().rand()$

$\alpha = 10^r$

- Hyperparameters for exponentially weighted averages

$\beta = 0.9, \ldots, 0.999$

$V_i = \beta V_{i-1} + (1-\beta)\theta_i$

$1-\beta \rightarrow 0.1, \ldots 0.001$

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

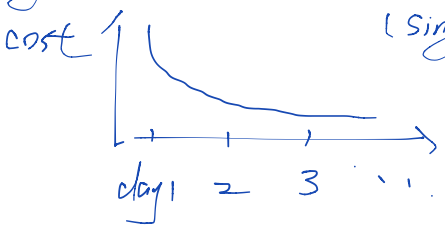
$1-\beta = 10^r$, $\beta = 1-10^r$

$\frac{1}{1-\beta}$ is sensitive to small changes of β when β is close to one

- Hyperparameter tuning in practice: Panda VS. Caviar

Model training:

Babysitting one model ⟶ Panda approach (single model)

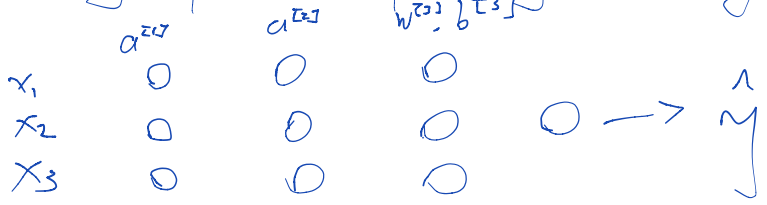


Training many models in parallel ⟶ caviar approach (multi-model)

Batch Normalization

Normalizing activations in a network

Normalizing inputs to speed up learning



Can we normalize values of a so that can train w, b faster?

Implement batch norm

Given some intermediant value in NN: $z^{(1)}, \ldots, z^{(m)}$

$z^{[1](i)}$

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

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

$$z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$ (for numerical stability)

$$\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$$ learnable parameters of model

if $\gamma = \sqrt{\sigma^2 + \varepsilon}$ & $\beta = \mu$

then $\tilde{z}^{(i)} = z$

$\tilde{z}$ makes hidden units have means & stds other than 0 & 1

$z^{[l][i]} \longrightarrow \tilde{z}^{[l][i]}$

Batch Normalization: Not just apply normalization to the input layer, but also deep into the hidden layer in the Neural Network

◦ Fitting Batch Norm Into a NN

◦ Adding batch norm to a network

$x \xrightarrow{w^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[\text{Batch Norm (B.N.)}]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \longrightarrow a^{[1]} = g^{[1]}(\tilde{z}^{[1]})$

$\longrightarrow \cdots\cdots\cdots\cdots$

Parameters: $w^{[1]}, b^{[1]}, \cdots, w^{[L]}, b^{[L]}$

$\beta^{[1]}, \gamma^{[1]}, \cdots, \beta^{[L]}, \gamma^{[L]}$

} GD, SGD, mini-batch, Adam ⋮

Implement batch norm in tensorflow:

tf.nn.batch-normalization

△ Working with mini-batches

(a common usage in practice)

$X^{\{1\}} \xrightarrow{w^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[\text{B.N.}]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \longrightarrow g^{[1]}(\tilde{z}^{[1]}) = a^{[1]}$

$\longrightarrow \cdots\cdots\cdots\cdots$

$X^{\{2\}} \rightarrow Z^{[1]} \xrightarrow[B.N.]{\beta^{[1]}, \gamma^{[1]}} \tilde{Z}^{[1]}$

↳ within mini-batch

$Z^{[1]} = W^{[1]} a^{[1-1]} + b^{[1]}$

$\tilde{Z}^{[1]} = Z_{norm} \cdot \gamma + \beta$

* $b^{[1]}$ does influence Z_{norm} Since Z_{norm} alway has 0 mean. thus we can remove $b^{[1]}$ term in $Z^{[1]} = W^{[1]} a^{[1-1]} + b^{[1]}$

$\longrightarrow Z^{[1]} = W^{[1]} a^{[1-1]}$

$\tilde{Z}^{[1]} = \gamma^{[1]} Z_{norm}^{[1]} + \beta^{[1]}$

△Implement Gradient descent

for $t = 1, \cdots,$ # of mini-batchs =

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

In each hidden layer, use B.N. to replace $Z^{[1]} \rightarrow \tilde{Z}^{[1]}$

Use back prop to compute $dw^{[1]}$, ~~$db^{[1]}$~~, $d\beta^{[1]}$, $d\gamma^{[1]}$

Update params $W^{[1]} := W^{[1]} - \alpha dw^{[1]}$

$\beta^{[1]} := \beta^{[1]} - \alpha d\beta^{[1]}$

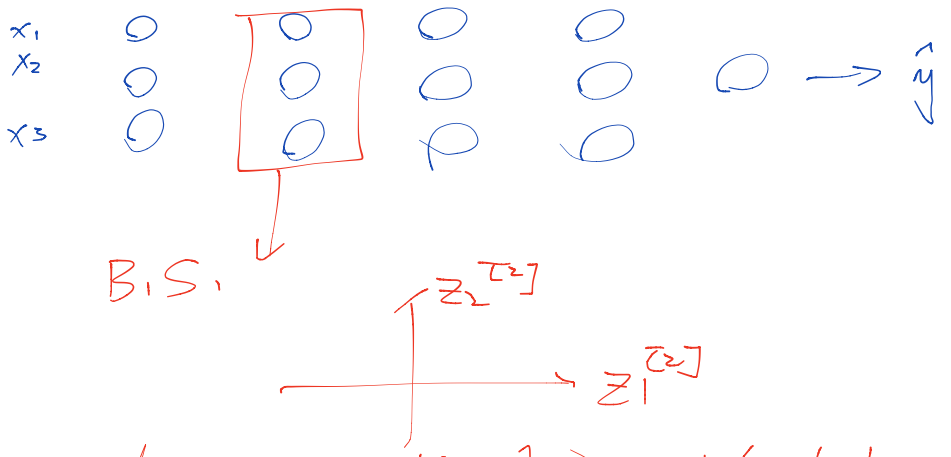
$\gamma^{[1]} := \gamma^{[1]} - \alpha d\gamma^{[1]}$

(also works with GD w/ momentum, adam

○ Why does Batch Norm work ?

△ Learning on shifting input distribution

Covariate shift : if the input distribution changes, then one may retrain the model



B.S. made sure no matter how input/early layers change, $z_1^{[2]}$, $z_2^{[2]}$ will always have the same mean and variance

☆ B.S. reduces influence of input value changing

△ Batch Norm as regularization

- Each mini-batch is scaled by the mean/variance computed on just that mini-batch
- This adds some noise to the values $\tilde{z}^{[l]}$ within that mini-batch, So similar to dropout, it adds some noise to

each hidden layer's activations.

- This has a slight regulation effect

- Don't turn B.N. as a regularization method

- Batch Norm at ~~test time~~
  - ~~At test time~~, there may not be enough examples to take mean/variance to do Batch Norm
  - Have to come up with some seperate estimate of μ, σ
  - μ, σ^2 : estimate using exponentially weighted average across mini-batch

e.g. $X^{\{1\}}$ $X^{\{2\}}$ $X^{\{3\}}$ in training

$\mu^{\{1\}[l]}$ $\mu^{\{2\}[l]}$ $\mu^{\{3\}[l]}$ $\longrightarrow$ μ, σ^2

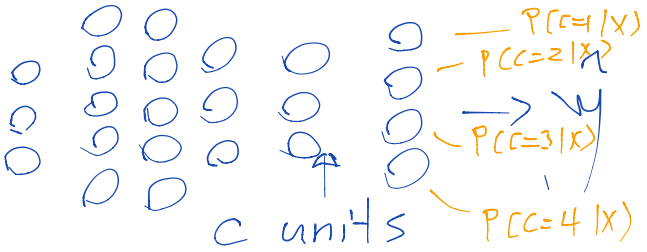
θ_1 θ_2 θ_3

in testing, use μ, σ from above training result to do the B.S.

- Multi-class classification
  - Softmax regression

C = # of classes

e.g. $C=4$ $\quad X \longrightarrow$



$\hat{y}$ is $(4, 1)$ matrix

△ Softmax layer

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

activation function:

$$t^{(i)} = e^{z^{[l](i)}}$$

$$a^{[l]} = \frac{t^{(i)}}{\sum_{i=1}^{4} t^{(i)}}$$

e.g. $z^{[l]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \rightarrow t = \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \\ 20.1 \end{bmatrix}$

↳ $\sum_{i}^{4} t^{(i)} = 176.3$

$a^{[l](1)} = 0.842$ $\quad a^{[l](2)} = 0.042$

$a^{[l](3)} = 0.002$ $\quad a^{[l](4)} = 0.114$

↳ $\hat{y} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$

Softmax + no hidden layers → linear decision boundrary classification

△ Training a Softmax classifier

- Understanding Softmax

"hard max": $\begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

"Softmax": $\begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \rightarrow \begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$ "more gental"

Softmax regression generalizes logistic regression to C classes. if C=2, softmax $\Longleftrightarrow$ logistic regression.

- loss function

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

- Cost

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

- $Y = [y^{(1)}, y^{(2)}, \cdots y^{(m)}]_{C \times m}$

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

- Deep Learning Framworks

- Caffe / caffe2
- CNTK
- DL4J
- Keras
- Lasagne
- mxnet
- PaddlePaddle
- TensorFlow
- Theano
- Torch

- TensorFlow