Week7
Hyperparameter tuning

Hyperparameter tuning

α, β, β_1, β_2, ϵ — # layers — # hidden units — learning rate decay — mini-batch size

Red → Green → Blue (random)

Try random values. don't use grid search as fine method.

Great more samples around that.

Using appropriate scale :-

$n^{(l)}$: 50 ..... 100 (hidden units)

L : 2 ..... 4 (# layers)

α : 0.0001 ..... 1. (linear scale)

(log-scale)

↳ use $r = -4 * np.random.rand()$

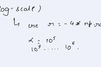
$\alpha = 10^r$
$10^{-1} \ldots 10^{-4}$

Weighted average:-

β : 0.9 ..... 0.999

$\frac{1}{1-\beta}$: 10 1000

$1-\beta$: 0.1 ..... 0.0001



$r \in [-3, -1]$ $\quad 1-\beta = 10^r$

random variable $\quad \rightarrow \beta = 1 - 10^r$

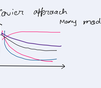
In practice:-

Pandas approach

one model, tune over days.

Caviar approach

Many models



Batch Normalization :-

Normalization is used to speed up learning process.

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

$X = X - \mu$

$\sigma^2 = \frac{1}{m} \sum X^{(i)^2}$

$X = X / \sigma^2$ (Now normalized)

→ normalize $z^{[l]}$ and then move forward.

done.

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

$z^{(i)}_{norm} = \frac{z - \mu}{\sqrt{\sigma^2 + \epsilon}}$ $\quad$ mean 0, variance = 1.

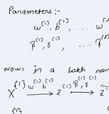
$\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$ $\quad$ learnable parameters of model $\quad$ (to avoid $\mu = 0$, $\sigma^2 = 1$)

If $\gamma = \sqrt{\sigma^2 + \epsilon}$ & $\beta = \mu$,

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

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

Adding batch Norm to a NN:-



The flow:-

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

Parameters:-

$w^{[1]}, \beta^{[1]}, \ldots w^{[L]}, b^{[L]}$

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

now, in a batch norm.

$X^{\{1\}} \xrightarrow[\beta^{[1]}]{w^{[1]}, b^{[1]}} Z^{[1]} \xrightarrow[\gamma^{[1]}, \beta^{[1]}]{BN} \tilde{Z}^{[1]} \longrightarrow a^{[1]} \xrightarrow[\beta^{[2]}]{w^{[2]}, b^{[2]}} Z^{[2]} \ldots$

$X^{\{t\}} \ldots \ldots X^{\{t\}} \ldots \ldots \tilde{Z}^{[1]}, \tilde{Z}^{[2]}$

Params → $w^{[l]}$, $b^{[l]}$, $\beta^{[l]}$, $\gamma^{[l]}$ $\quad$ $\gamma^{[l]}, \beta^{[l]}$ → $\tilde{Z}^{[l]}, z$

↳ $b^{[l]}$ is ineffective bcz we are zeroing mean = 0.

∴ $z^{[l]} = w \cdot a$

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

$[n^{[l]}, 1]$

Algorithm:-

for t = 1 to num_batches :-

compute f/p on $X^{\{t\}}$

In each hidden layer, compute $z^{[l]}$ with $\tilde{z}^{[l]}$

Use B/P to compute $dw^{[l]}$, $d\beta^{[l]}$, $d\gamma^{[l]}$

update parameters $w^{[l]} = w^{[l]} - \alpha \cdot dw^{[l]}$

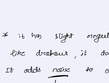
$\beta^{[l]} = \beta^{[l]} - \alpha \cdot d\beta^{[l]}$

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

↳ works with momentum, RMSprop, Adam.

Why Batch norm work ? → speeds up learning!

* Shifting input distribution



Co-variance shift

Now will perform freely, thinks to learn better.

Stabilizes the input values.

↳ later layers are not affected by change in earlier layers.

* It has slight regularization effect.

like dropout, it does not depends too much on one node.

It adds noise to each hidden layer's activations.

↳ because of mean and variance.

↳ dropout noise & random 0's y

* batch norm + dropout = OK.

↳ use large m if feasible. eg.. 512 instead of 64.

Batch Norm at test-time:- (ZMP)

Problem:- in training time we take a batch and calculate μ & σ^2

but in test time, we take one example at a time.

Solution:- Estimate estimate of μ & σ^2 exponentially weighted avg. across mini batch.

$Y^{(1)} \ldots X^{(1)} \ldots \ldots X^{(t)}$

$\mu^{[1][1]}$ $\quad \mu^{[1][2]}$ $\quad \mu^{[1][t]}$ → μ

$\sigma^{2[1][1]}$ $\quad \sigma^{2[1][2]}$ $\quad \sigma^{2[1][t]}$ → σ^2

} → exponentially weighted avg.

at test time ↓ ↓ not clear

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

take from train?

Multi-Class classification.

$X \rightarrow$ 0, 0, 0, 0, 0 → $P(cat/X)$, $P(dog/X)$, $P(\text{none}/X)$

$z^{(l)} = w \cdot a^{(l-1)} + b$

Activation function:- $\quad t = e^{(z^{[l]})}$ $\quad a^{[l]} = \frac{e^{z^{[l]}}}{\sum_{j=1}^{4} t_i}$ $\quad (3,1)$

$a^{[l]} = \frac{e^{z^{[l]}}}{\sum_{j=1}^{4} t_i}$ $\quad (4,1)$

Eg.. $z^{[l]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$ $\quad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \end{bmatrix}$ $\quad \sum_{j=1}^{4} t_i = 156.2$

$a^{[l]} = \frac{t}{\sum_{j=1}^{4} t_i} = \frac{t}{156.2} = \frac{e^5}{156.2}$ $\quad$ Softmax the most made at layer L.

$\hat{y}$ (3,1) vector.

Loss Function:-

$y^{(i)} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ cat, dog, none $\quad \hat{y}^{(i)} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \end{bmatrix}$ $\quad C = 3$

$J(\hat{y}, y) = -\sum_{j=1}^{4} y_j \cdot \log \hat{y}_j$ $\quad J(w^{[1]}, b^{[1]}, \ldots) = \frac{1}{m} \sum_{i=1}^{m} J(\hat{y}^{(i)}, y^{(i)})$

In this case, $-y_2 \log \hat{y}_2 = -\log \hat{y}_2$ {make $\hat{y}_2$ big to make loss small}

$Y = [y^{(1)}, y^{(2)}, \ldots y^{(m)}]$ $\quad \hat{Y} = [\hat{y}^{(1)}, \ldots, \hat{y}^{(m)}]$

$\begin{bmatrix} 0 & 1 & 0 & 0 & \ldots \\ 1 & 0 & 0 & 0 & \ldots \\ 0 & 0 & 1 & 0 & \ldots \end{bmatrix}$ $(3 \times m)$ $\quad$ $\begin{bmatrix} 0.1 & 0.7 & 0.9 & \ldots \\ 0.8 & 0.2 & 0.1 & \ldots \\ 0.1 & 0.1 & 0.0 & \ldots \end{bmatrix}$ $(3 \times m)$

If $c = 2$, it resembles logistic regression.

For B/P, the $dz^{[L]} = \hat{y} - y$ remember.