manually decay after observation

Week-3

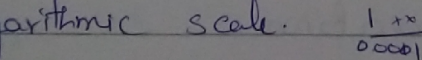
Hyperparam Tuning Process

Hyper parametes ~~rand~~

- □ α — ★ 3rd priority
- ○ β ~0.9 — □ 1st priority
- $\beta_1, \beta_2, \varepsilon$ → 0.9; 0.99; 10^{-8} — ○ 2nd pr
- ★ # layers
- ○ # hidden units
- ★ learning rate decay
- ○ mini batch size

Previously a grid was made (5×5). And all 25 params were used to test for best model.
But in case 2 hyper paraam is more important than the other, then in such a case, you get to try only 5 values of that param
Now, we take 25 points only, but select randomly.
In case if we have more than 2 parmeters, we use a cube
In such a case, even if we dont know, which hyper parameter is important, we can still get the best result
The area where optimal solution is obtained can be then more densely sampled

But in order to choose, we need an appropriate scale to choose from.
For example $\alpha = 0.0001, \ldots, 1$
it may happen that 90% of samples are btwn 0.1 to 1. and only 10% are from 0.0001 to 0.1
In such a case, we might want to use a logarithmic scale.



python implementation

$r = -4 \times$ np.random.rand() $\leftarrow r \in [-4, 0]$
$\alpha = 10^r$ $\leftarrow 10^{-4} \ldots 10^0$

For exponentially weighted averages
$\beta = 0.9 \ldots 0.999$
↓ 10 samples ↓ 1000 samples avg.

$1-\beta = 0.1 \ldots 0.001$
$r \in [-3, -1]$
$1-\beta = 10^r$
$\beta = 1-10^r$

⇒ Retest hyperparameters occassionally

Two approaches

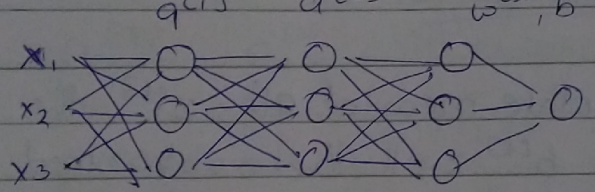
1) Babysit a mode [watch carefully the decrease in cost, make some changes, make more efficient]

2) Training many models in parallel

should choose based on your computat^n al research

Batch Normalization

Makes hyper parameter search problem easy, by increasing the scale



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

In this case we will normalize $z^{[2]}$

Given some intermediate values in NN $z^{(1)}, \dots, z^{(m)}$

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

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

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

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

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

if $\gamma = \sqrt{\sigma^2 + \varepsilon}$ (if $\sigma^2 \neq 0$)

$\beta = \mu$

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

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

$\mu = 1/m \sum x^{(i)}$

$X = X - \mu$

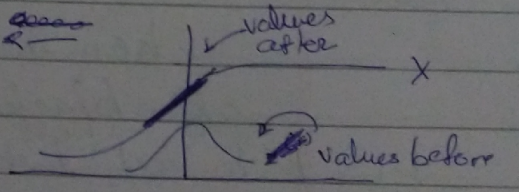
$\sigma^2 = 1/m \sum X^{(i)2}$

$X = X/\sigma^2$

Normalizat^n on i/p in Logi regr.

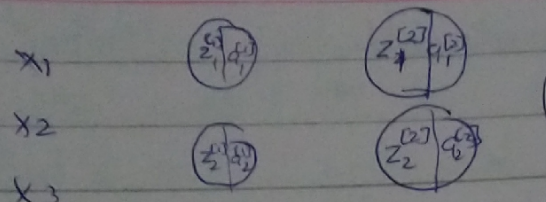
$X \leftarrow$

$Z^{(i)}$



(normalizat^n)

So to adjust mean and variances deep in the NN we use γ & β



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

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

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

To update β,

$d\beta^{[l]}$ is used; $\beta^{[l]} = \beta^{[l]} - \alpha d\beta^{[l]}$

→ Actually applied to mini-batches

When we actuall calculate mean of all $z^{[l]}$, the constants are subtracted. Here, $b^{[l]}$ will be subtracted from each entry and thus, they can be set to zero.

In the normalized $\tilde{z}$, beta is replaced by $\beta^{[l]}$, and is responsible for shift and bias.

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

$$z^{[l]}_{norm}$$

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

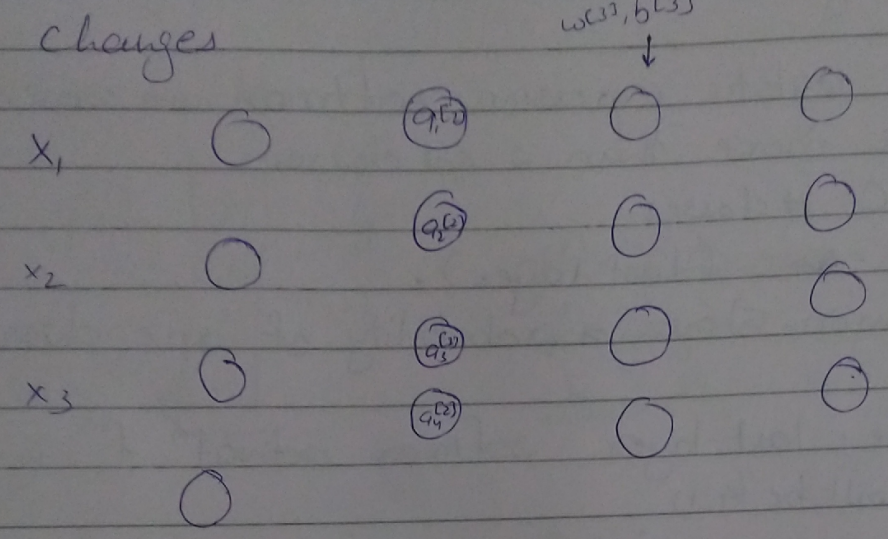
dimension of $z^{[l]}$, $\beta^{[l]}$ and $\gamma^{[l]}$ is $(n^{[l]}, 1)$

When in a case, if a model is trained on all black cats, and coloured cat images given to test, it may not understand.

Such a case is called covariate shift.

Consider a The n/w it wants to learn about the new parametes there will be a lot of change in the weights

What batch norm does, is make weights robust to changes



Consider the 3rd layer. Due to changes in the rest layers, $a_1^{[2]}, \dots a_4^{[2]}$ will change. batch norm makes sure that these changes do not affect the learning process of $w^{[3]}, b^{[3]}$. It makes sure that these values are more stable.

Thus, the early layers dont get to shift much, due to the constraints. This makes learning of later layers easier

→ Batch Norm has a slight regularizat^n effect.

Batch Norm at test time.

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

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

$$z_{norm}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

$$\tilde{z}^{(i)} = \gamma z_{norm}^{(i)} + \beta$$

To estimate μ, σ^2, exponentially weighted avg. (across mini-batches).

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

$\mu^{\{1\}[1]}$ $\mu^{\{2\}[1]}$ $\mu^{\{3\}[1]}$ $\rightsquigarrow \mu$

$\sigma^{2\{1\}[1]}$, $\sigma^{2\{2\}[1]}$ $\rightsquigarrow \sigma^2$

for each layer, μ, σ^2 is taken of each mini batch, and avged.

The finally avged. values are taken during test time and are found to be robust.

Softmax Regression

Instead of logistic regression, softmax is used, when there are more than 2 classes.

Consider C = # classes.

then, $n^{[L]} = C$ (last layer).

Each neuron shows a probability of one class.

So, in the last layer softmax activation fn is used.

$Z^{[L]}$ will be (4,1)

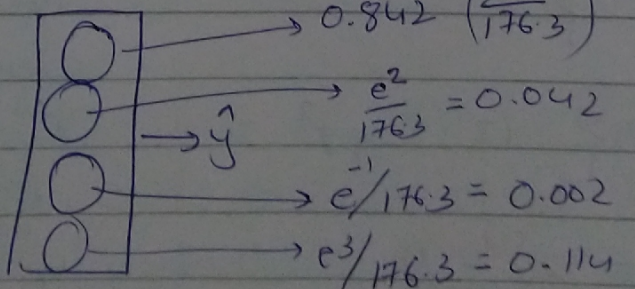
$$t = e^{(Z^{[L]})}$$

$$a^{[L]} = \frac{e^{Z^{[L]}}}{\sum_{i=1}^{4} t_i} \qquad a_i^{[L]} = \frac{t_i}{\sum_{i=1}^{4} t_i}$$

(4,1)

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

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



Without hidden layer it can work only for linear decision boundary.

Training Softmax Classifier

Loss function

$y^{(i)} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ cat, $y_2 = 1$

$y_1 = y_3 = y_4 = 0$

$a^{[L]} / \hat{y}^{(i)}$

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

(make small)

$$= -y_2 \log \hat{y}_2 = -\log \hat{y}_2$$

Make $\hat{y}_2$ big

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

$$Y = [y^{(1)}\ y^{(2)} \dots y^{(m)}]$$

$$= \begin{bmatrix} 0 & 0 & 1 & \dots \\ 1 & 0 & 0 & \\ 0 & 1 & 0 & \\ 0 & 0 & 0 & \end{bmatrix}_{(4,m)}$$

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

Backpropagation: $dz^{[L]}_{(4,1)} = \hat{y} - y$

$\frac{\partial J}{\partial z^{[L]}}$

Deep Learning Frameworks

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

Tensorflow