Week 1

Monday, October 16, 2017 10:29 AM

Setting Up

Monday, October 16, 2017 10:30 AM

Train / Dev / Test sets

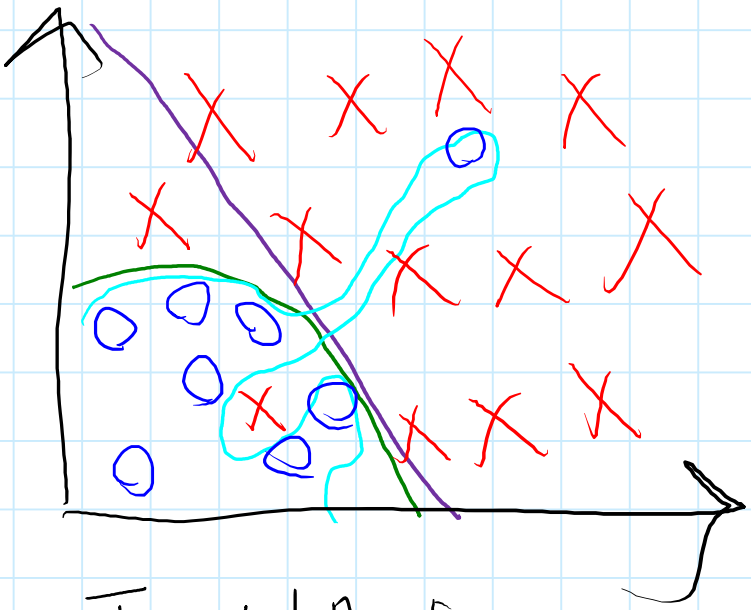
split data into 3 groups

Train - train NN on bulk of data

Dev - cross validate multiple training styles to find best

Test - Reported results

Bias / Variance



high bias, underfit

high variance, overfit

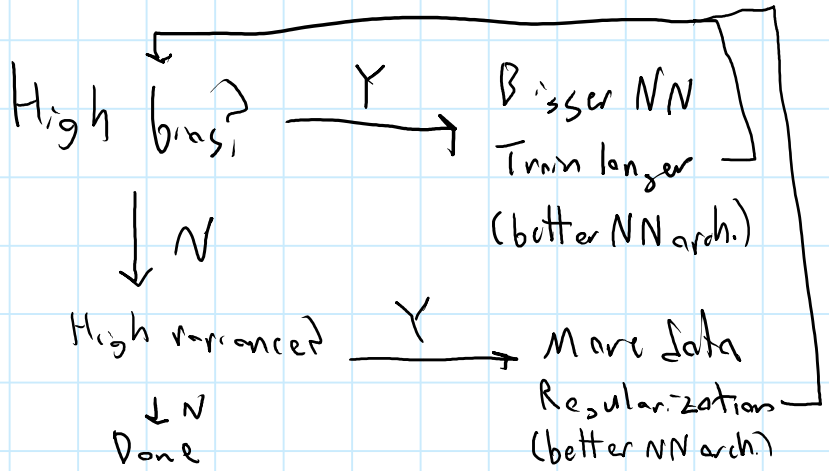
best fit

Train Err	Dev Err	
1%	11%	high variance
15%	16%	high bias

1%	11%	high variance
15%	16%	high bias
15%	30%	both!
0.5%	1%	

(if Bayes' error ~ 0%)

Basic recipe for ML



Regularization

Monday, October 16, 2017 10:56 AM

Logistic Regression

$$\min_{w,b} J(w,b)$$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w\|_2^2$$

(regularization parameter)

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

L_1 regularization $\frac{\lambda}{2m}\sum_{i=1}^{n_x}|w| = \frac{\lambda}{2m}\|w\|_1$ "sparse" (lots of 0's)

Neural Network

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

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

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

$$w^{[l]} := w^{[l]} - \alpha\left(dw^{[l]} + \frac{\lambda}{m} w^{[l]}\right)$$

Why does regularization help with overfitting?

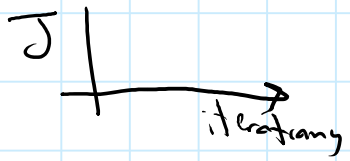
λ penalizes weights (w) for being too large, reduces impact of hidden units

$$\lambda \uparrow \rightarrow W^{[l]} \downarrow \rightarrow Z^{[l]} \downarrow$$, makes layer more linear

Good behavior method

$\lambda \uparrow \rightarrow W \downarrow$? $\Sigma \downarrow$, more layer more linear

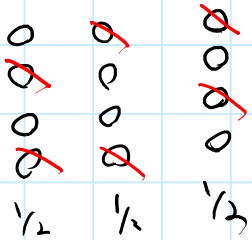
Good debugging method:



Dropout Regularization

Each node has chance of being eliminated

ex. coin toss to elim node in each layer



Implementing: illustrate w/ layer 3 (inverted dropout)

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

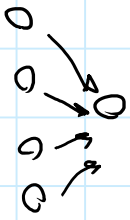
At test time, dropout won't be implemented

Why does dropout work?

Intuition: can't rely on any one feature, spread out weights

i.e.

i.e.



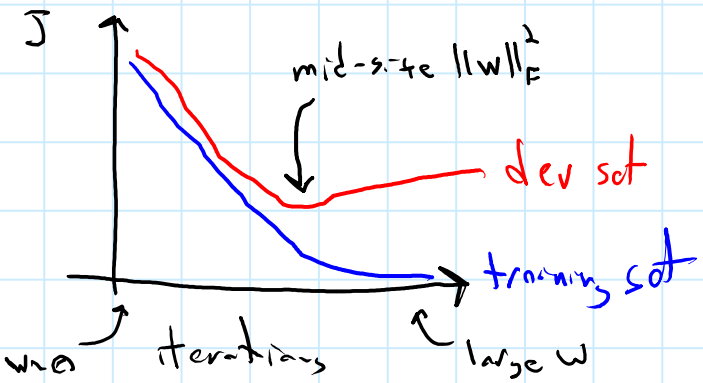
if any input can disappear, none favored

Can vary keep_prob by layer (lower w/ more nodes, higher w/ less)

Other techniques

Augmentation
Flip images horizontally, random zooms, crops, rotations, distortions

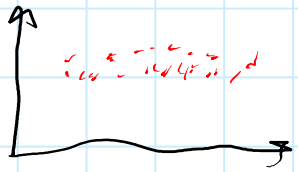
Early stopping, (alt to L_2)



Setting Up Optimization

Wednesday, October 18, 2017 10:28 AM

Normalizing Training Sets



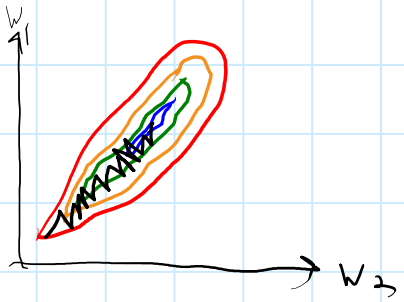
Subtract mean: $\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$, $X := x - \mu$

Normalize variance: $\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} **2$ (element-wise square)

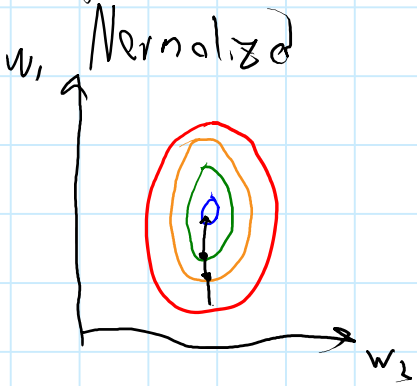
Make sure to use same σ, μ for test set

Why? Faster gradient descent

Unnormalized



Normalized



Vanishing / Exploding Gradients

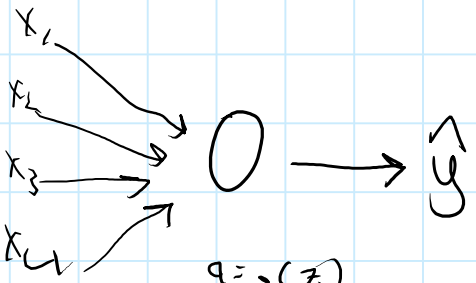
Given $w^{[L]}$, $g(z) = z$, $b^{[l]} = 0$, $w^{[l]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$

$\hat{y} = \left(\prod_{i=1}^{L} w^{[i]}\right) X \rightarrow \hat{y} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}^{L-1} X$ → causes exponential change

Partial remedy:

Weight Initialization for Deep Networks

Single Neuron example



$$z = \sum_{i=1}^{n} w_i x_i$$

larger n → want smaller w_i, variance $w_i \approx \frac{1}{n}$

$$w^{[l]} \approx \sqrt{\frac{c}{n^{[l-1]}}}$$

ReLU: $c = 2$

tanh: $c = 1$ (Xavier init)

Numerical Approx of Gradients

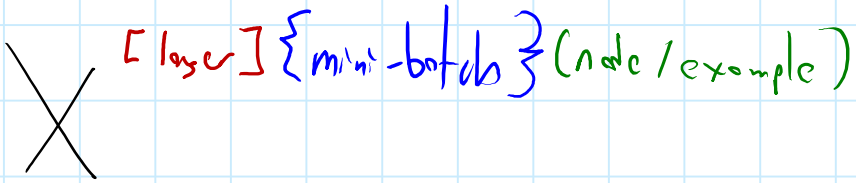
Homework Notes

Saturday, October 21, 2017 11:13 AM

Week 2

Thursday, October 26, 2017 10:29 AM

Notations:



Optimization Algorithms

Thursday, October 26, 2017 10:29 AM

Mini-batch Gradient Descent $\{t\}$

batch: $X = [x^{(1)} | \dots | x^{(m)}]$; $Y = [y^{(1)} | \dots | y^{(m)}]$

mini batch: $X^{\{1\}}$ $X^{\{2\}} \dots$; $Y^{\{1\}}$ $Y^{\{2\}}$

Split into smaller chunks & learn/update sequentially

ex. $t \in \{1, \dots, 5000\}$ $m = 5{,}000{,}000$

for $t = [1, 5000]$

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

$Z^{[1]} = W^{[1]} X^{\{t\}} + b^{[1]}$

$A^{[1]} = g^{[1]}(Z^{[1]})$

from $X^{\{t\}}, Y^{\{t\}}$

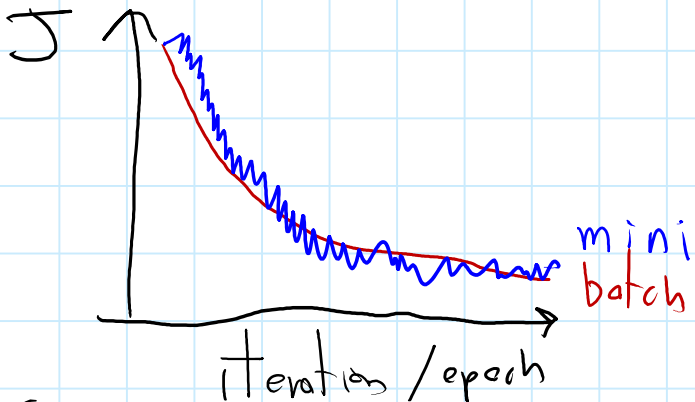
Compute cost $J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^{l} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \cdot 1000} \sum_{l} \| W^{[l]} \|_F^2$

backprop to get gradients

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

All this constitutes 1 "epoch" (pass through training set)

Understanding Mini-batch $\nabla \searrow$



oscillates since smaller sets of training data trained

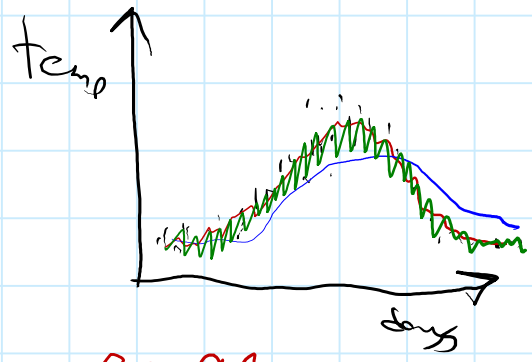
Choosing mini batch size

if size = 1: Stochastic gradient descent
- Lose speed gain from vectorization

Small train set (≤ 2500): Just use batch $\nabla \downarrow$

Typical mini batch size: 2^n (64, 128, 256, 512)
- make sure it fits in CC/GPU memory!

Exponentially Weighted (moving) Averages



$\theta_1 = 40F$

$\theta_{180} = 60F$

$\theta_{356} = 40F$

$V_t = \beta V_{t-1} + (1-\beta)\theta_t$

$V_t \sim$ avg over $\frac{1}{1-\beta}$ days temp

$\beta = 0.9 \sim 10$ days temp

$\beta = 0.98 \sim 50$ days temp (adapts slower)

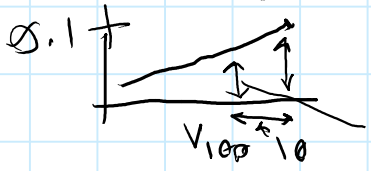
$\beta = 0.5 \sim 2$ days (adapts rapidly, very sensitive)

Understanding

$\beta = 0.9$

$V_{100} = 0.1\theta_{100} + 0.9(0.1\theta_{99} + 0.9(0.1\theta_{98} + 0.9(\dots)$

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



$\frac{1}{3}$ height $[0.9^{10} \sim \frac{1}{e}$,

$v_\theta = 0$

repeat for all θ_t {

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

}

Bias Correction

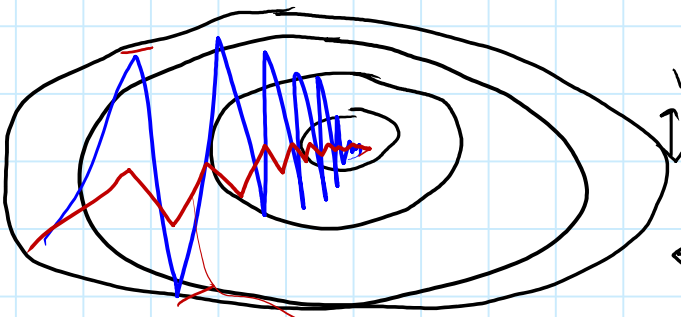
$v_\theta = 0$

$v_1 = 0.98 v_0 + 0.02\theta_1$

First handful of values will be abnormally low

Correction: $\frac{v_t}{1-\beta^t}$

Gradient Descent w/ Momentum



want slower learning rate ⟷ faster learning rate

Use exp weighted avg

avg of oscillations much closer to 0

On iteration t:

$V_{dW} = \beta V_{dW} + (1-\beta) dW$

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

rolling ball downhill analogy

"acceleration" — friction — velocity

$W := W - \alpha V_{dW}$, $b := b - \alpha V_{db}$

(most people don't bias correct)

(most people don't bias correct)

RMS Prop (Root Mean Squared)

On iteration t:

Compute dw, db

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

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

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

← avoids division by 0

Adam Optimization Algorithm (ADAptive Moment estimation)

$V_{dw}, S_{dw}, V_{db}, S_{db} = 0$

On iter t:

Compute dw, db

$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dw$ $V_{db} = \beta_1 V_{db} + (1-\beta_1) db$

$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dw^2$, $S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2$

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

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

$W := W - \alpha\, V_{dw}^{corr}/\sqrt{S_{dw}^{corr}} + \varepsilon$, $b := b - \alpha\, V_{db}^{corr}/\sqrt{S_{db}^{corr}} + \varepsilon$

Choice of hyperparams:

α : must be tuned

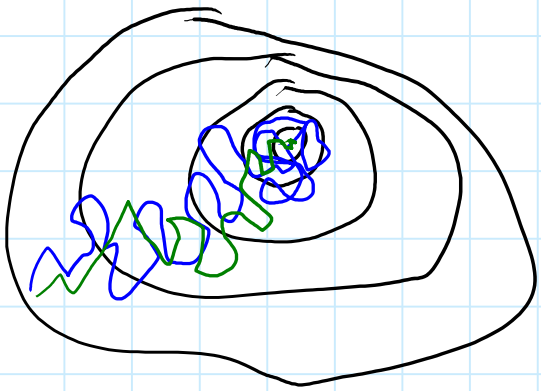
$\beta_1 : 0.9$; $\beta_2 : 0.999$

$\varepsilon : 10^{-8}$

Hyperparameter Tuning

Sunday, October 29, 2017 2:35 PM

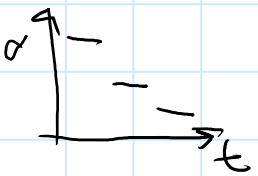
Learning Rate Decay (low on Dr. Ng's priority)



lower α as training goes on

$$\alpha = \frac{\alpha_0}{1 + \text{decay\_rate} * \text{epoch\_num}}$$

$\alpha = 0.95^{(\text{epoch})} \cdot \alpha_0$, $\frac{k \cdot \alpha_0}{\sqrt{\text{epoch}}}$, discrete staircase;



Can also control manually

The Problem of Local Optima

While easy to imagine issue of local optima as "wells", due to high dimensionality they're more like saddle points, real issue is plateaus causing learning to slow, but not stall (ADAM helps here!)

Homework Notes

Sunday, October 29, 2017 2:55 PM

Week 3

Tuesday, October 31, 2017 11:01 AM

Hyperparameter Tuning

Tuesday, October 31, 2017 11:01 AM

rough importance

1, 2, 3

α

β ~0.9

$\beta_1, \beta_2, \varepsilon = [0.9, 0.999, 10^{-8}]$

layers

hidden units

learning rate decay

mini batch size

Try random values, don't use grid of values (p1 p2 )

Use coarse to fine method

if these worked well, zoom in on this area

Appropriate scale to pick hyperparameters

Picking randomly doesn't mean uniformly random!

e.g. $\alpha = 10^{-4} \cdots 1$, sample along log scale

```
r = -4 * np.random.rand() <- r ∈ [-4, 0]
α = 10 ** r
```

$\beta = 0.9 \dots 0.999$; $(1-\beta) = 10^{-1} \dots 10^{-3}$

implement same as α Note $\lim_{\beta \to 1} \frac{R}{1-\beta} = \infty$

Organizing hyperparameter searches

"panda vs caviar" parenting

panda: low resources, "babysit" one model & modify regularly, looking for changes that significantly improve J

caviar: many resources, train many models in parallel

Batch Normalization

Tuesday, October 31, 2017 10:22 PM

Normalizing features improves learning, the same applies to hidden layers!

Typically normalize $z^{[l]}$, not $a^{[l]}$

ex, Given some values $z^{[l]}$

$$\mu = \frac{1}{m}\sum_i^m z^{(i)}$$
$$\sigma^2 = \frac{1}{m}\sum_i^n (z_i - \mu)^2$$
$$z^{(i)}_{norm} = (z^{(i)} - \mu) / \sqrt{\sigma^2 + \varepsilon}$$ (ε: infinitesimal)

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

Don't always want $\mu = 0$ & $\sigma = 1$ for hidden units

learnable parameters

Using Batch Norm in a Deep NN

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

Typically used w/ mini-batches (Same as above but $X^{\{batch\}}$ is input, $\tilde{z}$ is specific to batch)

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

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

$b^{[l]}$ will be cancelled in normalization process due to mean subtraction, gets replaced by $\beta^{[l]}$

Implementing: (w/ $\nabla\downarrow$)

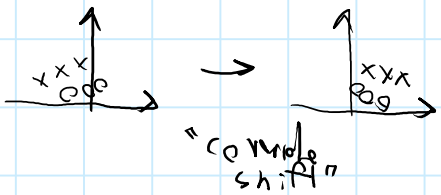
for $t = 1 \ldots$ num_mini_batches

for t=1 ... num_mini_batches
  Compute FP
    Use BN to replace $z^{[l]}$ w/ $\tilde{z}^{[l]}$
  Use BP to find $dw^{[l]}$, ~~$db^{[l]}$~~, $d\beta^{[l]}$, $d\gamma^{[l]}$
  Update params

Why Does BN Work?

Learns on shifting input distribution

e.g. training cat classifier on only black cats may not recognize non-black



BN maintains mean(0) & variance(1)
Weakens coupling between layers

BN as slight Regularization

Since each mini-batch has its own μ/σ computed & normalized, some noise is introduced, forcing downstream neurons not to rely on one in particular (like dropout)
[Not a standin for actual regularization]

BN @ Test Time

BN @ test time

Not using mini batches during test

μ, σ^2 estimated w/ exp. weighted avg. across mini batches

i.e. $X^{\{t\}}$

$\mu^{\{t\}[l]}$, $\sigma^{2\{t\}[l]}$ → combine over t → use

$$Z_{norm} = \frac{Z - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

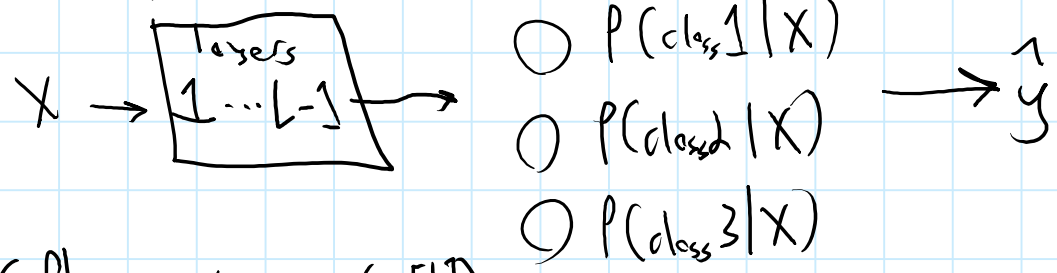
Multi-class Classification

Wednesday, November 1, 2017 10:18 AM

Softmax Regression

C = # classes trying to classify, $n^{[L]} = C$

ex. 4 possibilities



Softmax Layer ($z^{[L]}$)

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

Activation function:

$$t = e^{z^{[L]}}, \quad a^{[L]} = \frac{t}{\sum_{j=1}^{n} t_i}, \quad a_i^{[L]} = \frac{t_i}{\sum t_i}$$

ex. $z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \rightarrow t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix}$, $\sum_{j=1}^{4} t_j \sim 176.3 \rightarrow a^{[L]} = \frac{t}{176.3}$

Training a Softmax Classifier

Loss Function

$y = \begin{bmatrix} 0 \\ 1 \\ \vdots \end{bmatrix}$ $\hat{y} = \begin{bmatrix} 0.3 \\ \vdots \end{bmatrix}$ $P(y|\hat{y})$ $\sum_{j=1}^{C} L(\hat{y}, y)$

$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ $\hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$ $\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{C} y_j \log(\hat{y}_j)$

here, $\mathcal{L}(\hat{y}, y) = -y_2 \log \hat{y}_2$, only way to modify is to increase $\hat{y}_2$

"Maximum likelihood estimation" $J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}, y)$

"Gradient Descent w/ Softmax

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

↳ $\frac{dJ}{dz^{[L]}}$