# CENG 506 Deep Learning

Lecture 2 – Linear Classification

(Logistic Regression, Multi-class Classification,

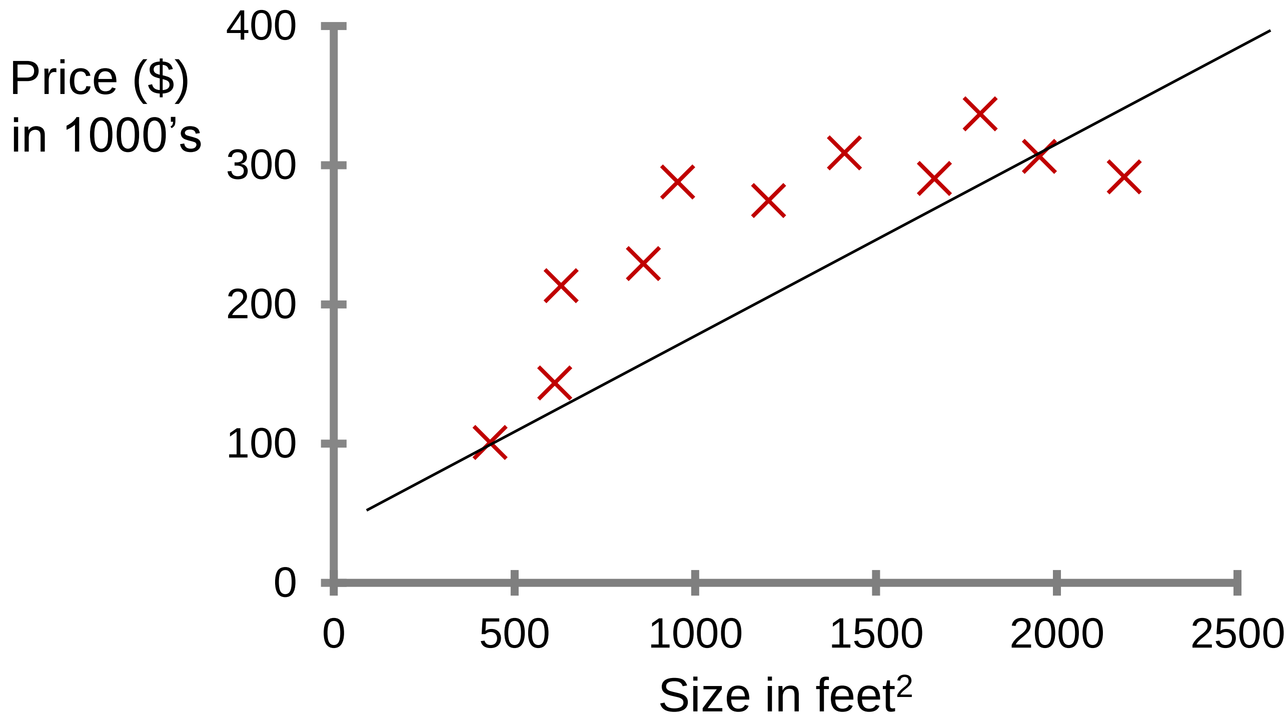
Softmax Loss, Regularization, Learning Rate)

# Recap

Last week we talked about linear regression problem.

Example:

House prices according to house size

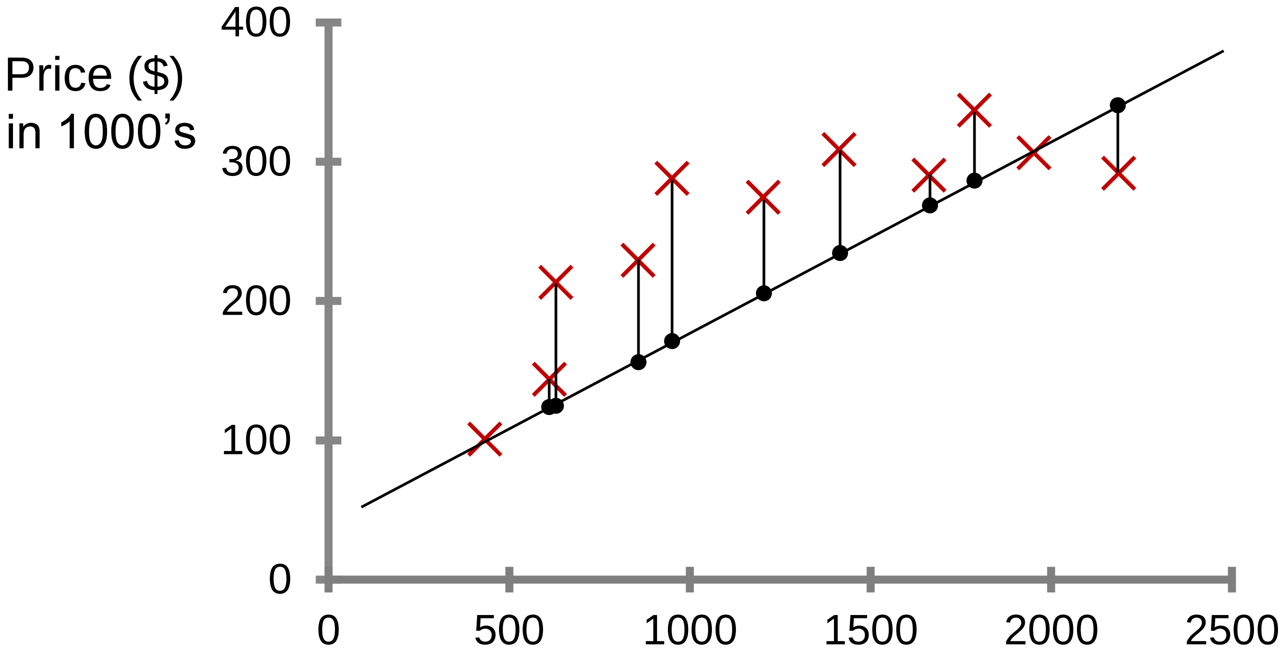



Size of house → $h$ → Estimated price

$$h_w(x) = w_0 + w_1 x$$

# Recap

We defined a cost function to minimize.




$y$: real prices (red crosses)

Our hypothesis:
$h(x) = w_0 + w_1 x$

First sample cost:
$(y^{(1)} - h(x^{(1)}))^2$

Total cost:
$\Sigma\ (y^{(i)} - h(x^{(i)}))^2$

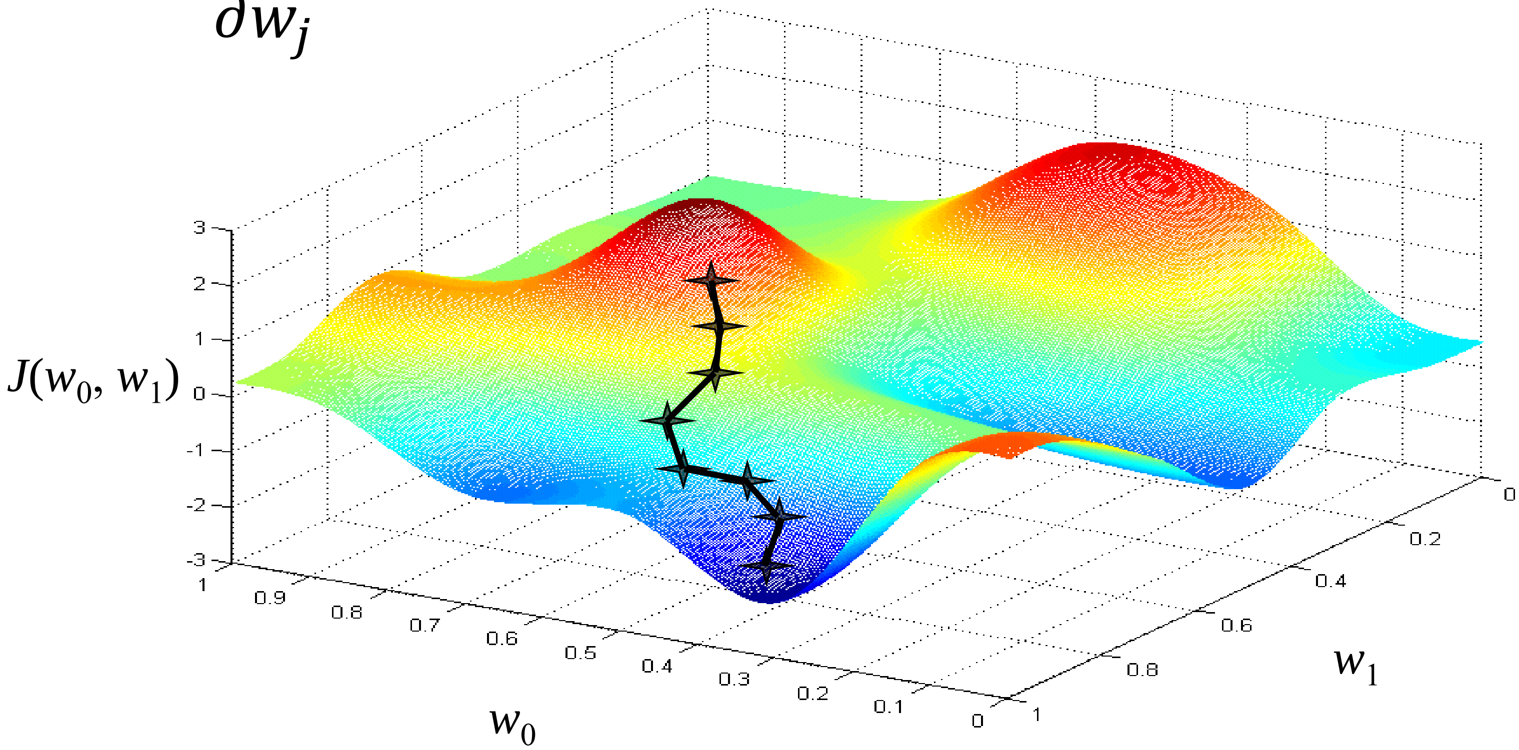
# Recap

We changed $w_0, w_1$ with gradient descent to reach min cost.

Repeat until convergence {

$$w_j := w_j - \alpha \frac{\partial J(w_0, w_1)}{\partial w_j}$$

}



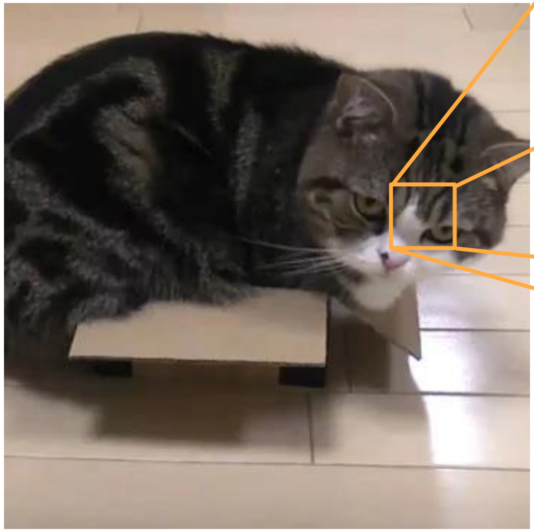
# Binary Classification

128x128x3 image

$$y \in \{0,1\}$$

0: Non-cat
1: Cat

# Binary Classification



| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 226 | 241 | 192 | 99 | 204 | 44 | 151 | 221 | 52 | 37 | 125 | 118 | 44 | 26 | 84 | 8 | 211 | 10 | 237 | 248 |
| 250 | 47 | 203 | 53 | 98 | 195 | 146 | 159 | 226 | 115 | 226 | 227 | 251 | 51 | 197 | 104 | 136 | 158 | 76 | 16 |
| 231 | 201 | 10 | 177 | 158 | 130 | 198 | 27 | 114 | 239 | 241 | 203 | 163 | 76 | 217 | 191 | 203 | 33 | 244 | 2 |
| 232 | 48 | 0 | 66 | 222 | 73 | 185 | 52 | 242 | 242 | 156 | 9 | 41 | 251 | 64 | 97 | 250 | 202 | 180 | 1 |
| 169 | 212 | 50 | 123 | 20 | 214 | 224 | 254 | 84 | 178 | 57 | 115 | 29 | 2 | 206 | 205 | 222 | 24 | 115 | 168 |
| 84 | 88 | 238 | 233 | 100 | 183 | 59 | 118 | 216 | 118 | 73 | 127 | 50 | 217 | 174 | 163 | 166 | 127 | 105 | 102 |
| 165 | 238 | 7 | 249 | 188 | 15 | 242 | 151 | 76 | 253 | 173 | 242 | 6 | 8 | 255 | 0 | 102 | 165 | 22 | 38 |
| 103 | 219 | 236 | 59 | 177 | 93 | 208 | 8 | 246 | 71 | 140 | 14 | 74 | 237 | 118 | 64 | 18 | 195 | 192 | 232 |
| 168 | 186 | 144 | 170 | 235 | 134 | 16 | 206 | 35 | 161 | 218 | 3 | 157 | 46 | 2 | 104 | 86 | 134 | 123 | 111 |
| 195 | 74 | 201 | 48 | 118 | 71 | 211 | 200 | 248 | 241 | 16 | 96 | 173 | 113 | 215 | 173 | 49 | 205 | 127 | 48 |
| 24 | 218 | 35 | 50 | 109 | 179 | 234 | 18 | 27 | 143 | 173 | 165 | 110 | 93 | 147 | 101 | 122 | 239 | 189 | 82 |
| 246 | 132 | 154 | 255 | 90 | 96 | 121 | 50 | 246 | 250 | 194 | 219 | 134 | 48 | 213 | 186 | 59 | 202 | 241 | 175 |
| 193 | 202 | 66 | 69 | 32 | 153 | 64 | 89 | 156 | 174 | 163 | 125 | 179 | 197 | 238 | 57 | 177 | 33 | 106 | 200 |
| 30 | 198 | 86 | 161 | 55 | 157 | 27 | 140 | 198 | 6 | 120 | 215 | 12 | 207 | 89 | 221 | 185 | 12 | 195 | 153 |
| 131 | 31 | 7 | 64 | 159 | 82 | 154 | 101 | 217 | 35 | 19 | 40 | 208 | 144 | 136 | 151 | 21 | 37 | 201 | 112 |
| 159 | 77 | 244 | 124 | 110 | 74 | 218 | 73 | 70 | 145 | 30 | 160 | 82 | 122 | 136 | 159 | 9 | 155 | 96 | 239 |
| 248 | 250 | 13 | 16 | 68 | 236 | 3 | 125 | 146 | 70 | 91 | 114 | 32 | 188 | 78 | 171 | 42 | 4 | 235 | 13 |
| 130 | 209 | 54 | 100 | 73 | 197 | 94 | 135 | 253 | 110 | 43 | 98 | 226 | 154 | 205 | 16 | 131 | 237 | 237 | 198 |
| 172 | 63 | 16 | 18 | 228 | 56 | 163 | 222 | 143 | 149 | 128 | 196 | 21 | 127 | 249 | 79 | 39 | 156 | 84 | 177 |
| 209 | 141 | 242 | 6 | 204 | 36 | 174 | 202 | 64 | 223 | 18 | 16 | 210 | 213 | 241 | 252 | 57 | 170 | 161 | 107 |

128x128x3 = 49152. Data vector becomes $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{49152} \end{bmatrix}$ $y = 1$

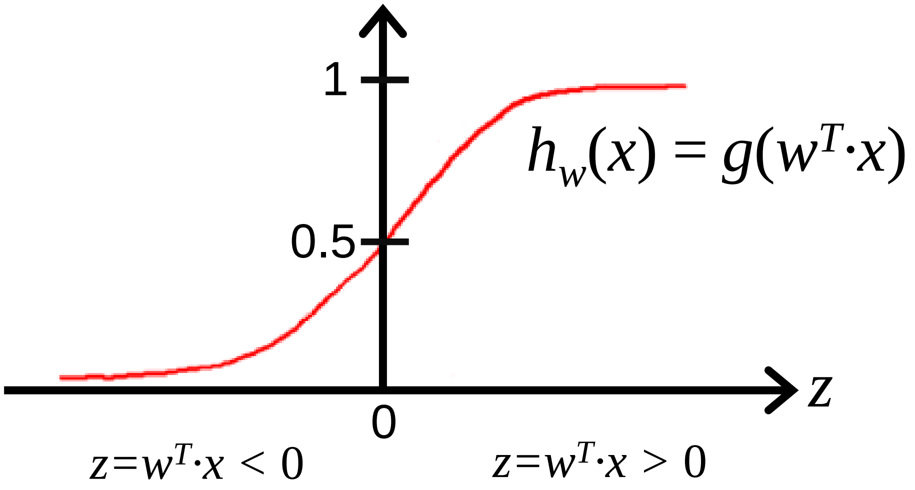
# Logistic Regression

$$h_w(x) = g(w^T \cdot x)$$

$$g(z) = \frac{1}{1+e^{-z}}$$

$$h_w(x) = \frac{1}{1+e^{-w^T \cdot x}}$$

$g(z)$ is called
*Logistic function*
or
*Sigmoid fuction*




Parameter vector $w$ has the same size with $x$.
Parameters are learned (fitted) with Log. Reg. algorithm.

# Decision Boundary

One strategy for decision is:

Predict $y=1$ if $h_\theta(x) > 0.5$

$y=0$ if $h_\theta(x) <= 0.5$

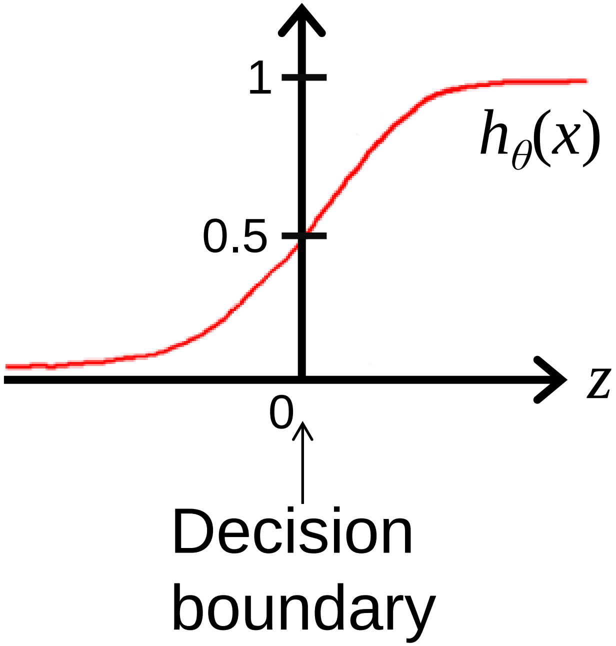
This means

Predict $y=1$ if $\theta^T \cdot x > 0$

$y=0$ if $\theta^T \cdot x <= 0$

$$h_\theta(x) = g(\theta^T \cdot x)$$

$$g(z) = \frac{1}{1+e^{-z}}$$



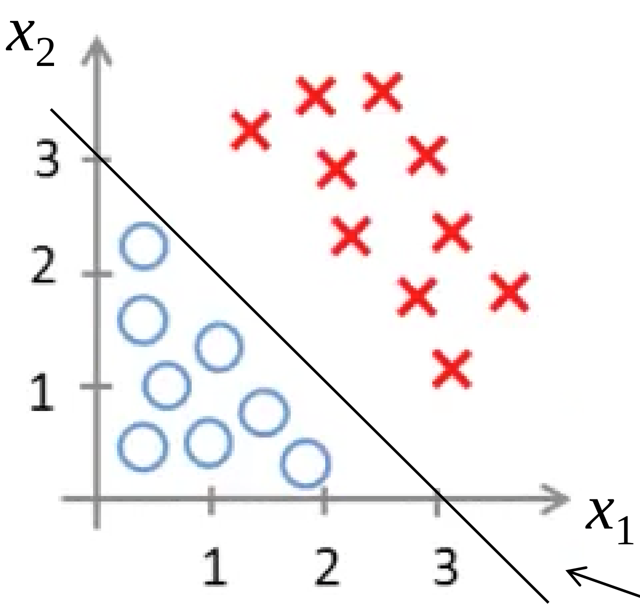
# Logistic Regression

For an image (e.g. 49152 dimensions) we can not visualize, but it is a linear boundary.

An example in 2D looks like this:

$$h_w(x) = g(w_0 + w_1 \cdot x_1 + w_2 \cdot x_2)$$

$$h_w(x) = g(-3 + 1 \cdot x_1 + 1 \cdot x_2)$$




$x_1 + x_2 = 3$ : Decision boundary

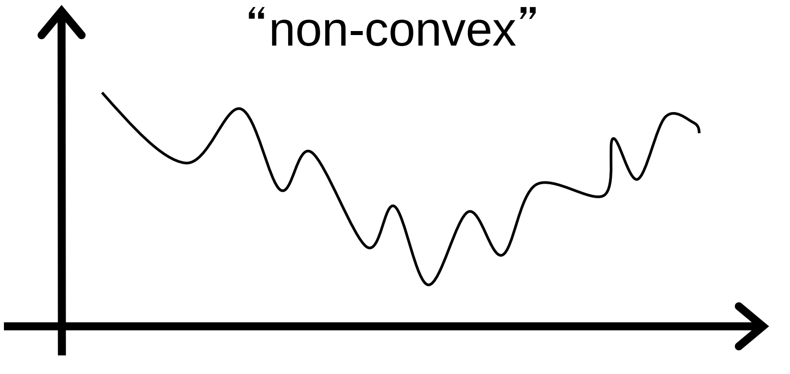
# Logistic Regression

How to estimate the best $w$ for the decision boundary?

We could minimize a simple cost function as follows:

$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_w(x^{(i)}) - y^{(i)}\right)^2$$

It turns out that, because of the non-linearity of the sigmoid function that we use in $h_w(x)$,
this cost function
becomes non-convex
(i.e. contains local minima).

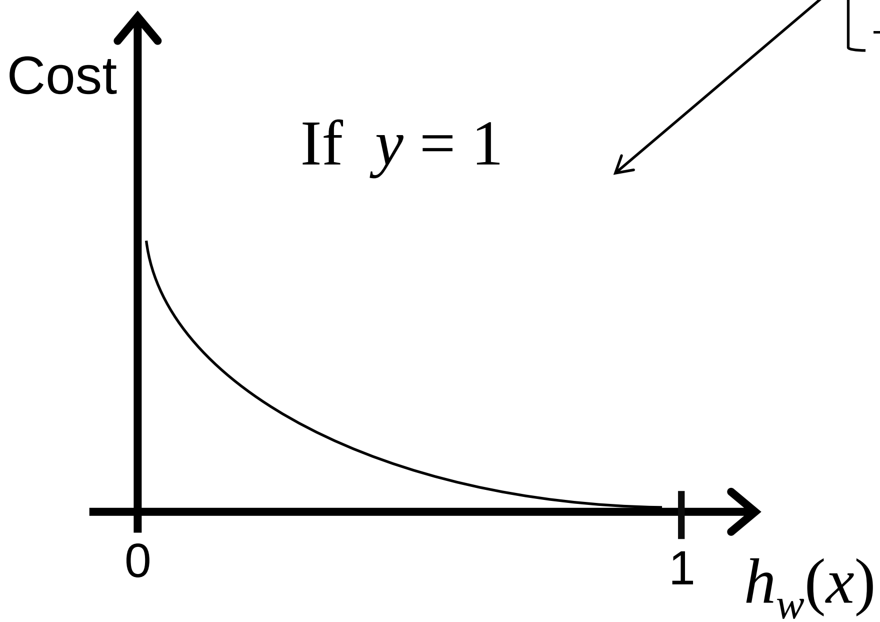


# Logistic Regression

We need another function which is convex.
Let’s use **cross-entropy loss**:

$$\text{Cost}\,(\,h_w(x),\,y\,) = \begin{cases} -\log(h_w(x)) & \text{if } y=1 \\ -\log(1-h_w(x)) & \text{if } y=0 \end{cases}$$

If $y = 1$



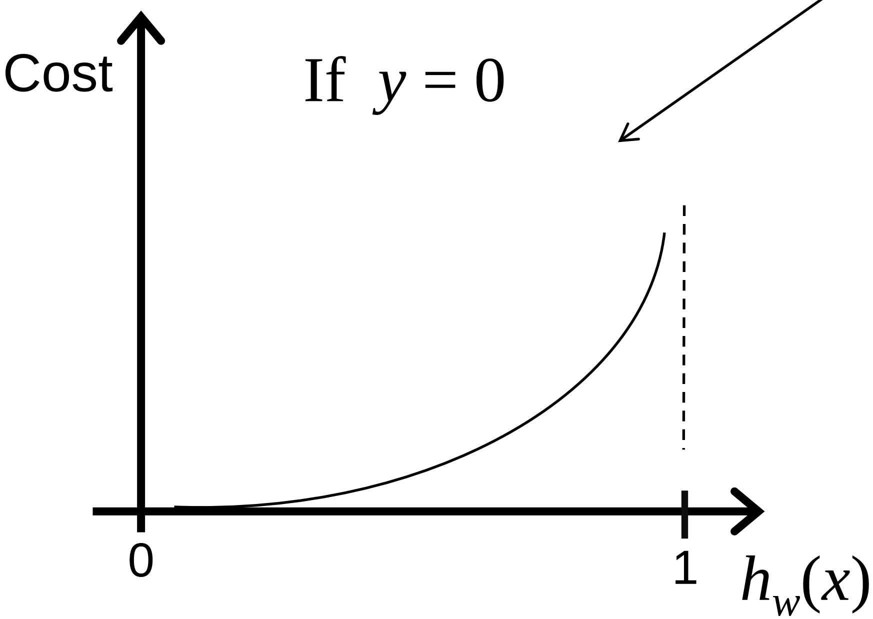

Cost=0 @ $h_w(x)$=1 (since $y$=1)
But, cost $\to\infty$ as $h_w(x) \to 0$
which means: if we predit 0 when $y$=1, we penalize by a very large cost.

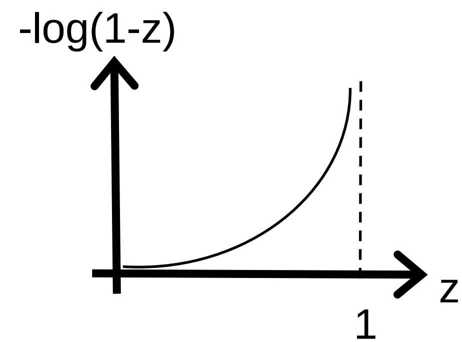
# Logistic Regression

**Cross-entropy loss**:

$$\text{Cost}\,(\,h_w(x), y\,) = \begin{cases} -\log(h_w(x)) & \text{if } y=1 \\ -\log(1-h_w(x)) & \text{if } y=0 \end{cases}$$

If $y = 0$




Cost=0 @ $h_w(x)$=0 (since $y$=0)

But, Cost $\rightarrow \infty$ as $h_w(x) \rightarrow 1$

# Logistic Regression

$$\text{Cost}(\,h_w(x), y\,) = \begin{cases} -\log(h_w(x)) & \text{if } y=1 \\ -\log(1-h_w(x)) & \text{if } y=0 \end{cases}$$

Knowing that $y$ is always equal to $0$ or $1$, we can define the cost function with a single line:

$$\text{Cost}(\,h_w(x), y\,) = -\,y\cdot\log(h_w(x)) - (1-y)\cdot\log(1-h_w(x))$$

if $y=1$, cost becomes $-\log(h_w(x))$

if $y=0$, cost becomes $-\log(1-h_w(x))$

# Derivative of the Cost Function

$$J(w) = \frac{1}{m}\sum_{i=1}^{m}(-y^{(i)}\log(h_w(x^{(i)})) - (1-y)\log(1-h_w(x^{(i)}))$$

To estimate parameters, $\min_{w} J(w)$

Derivative computed using calculus is as follows:

$$\frac{\partial J(w)}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m}(h_w\left(x^{(i)}\right) - y^{(i)}) \cdot x_j^{(i)}$$

# Gradient Descent

Repeat $\{$

$$h_w(x) = g(w^T x) = \frac{1}{1 + e^{-w^T x}}$$

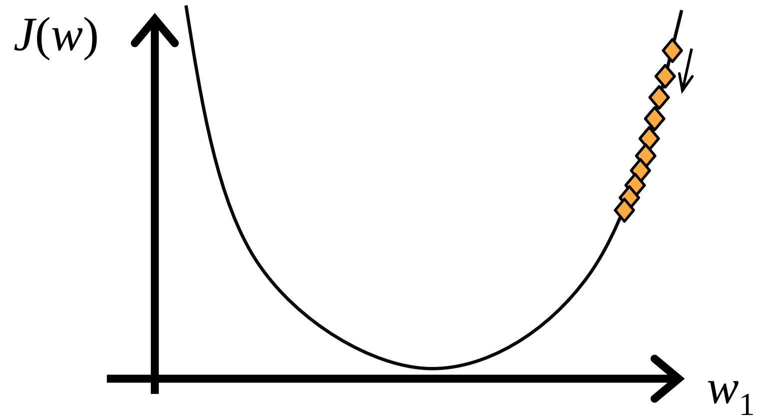
$$w_j := w_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \underbrace{\left(h_w\left(x^{(i)}\right) - y^{(i)}\right) \cdot x_j^{(i)}}_{\text{gradient}}$$

$\}$

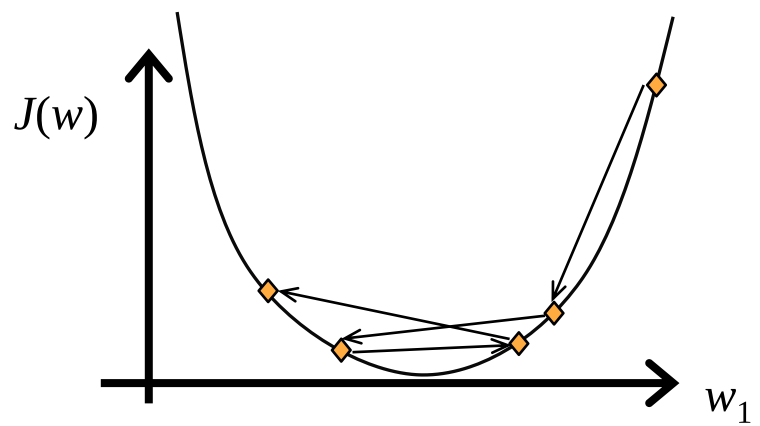
# Gradient Descent – Learning Rate

$$w_j := w_j - \alpha \frac{\partial J(w)}{w_j}$$

If $\alpha$ is too small, gradient descent can be slow.




If $\alpha$ is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



# Multi-class Classification

Now we have a different parameter set $w$ for each class.

$w$'s are stacked up, forms a matrix $W$.

Here, $b$ is actually $w_0$, and $W$ includes $w_1$, $w_2$, etc.

Let's say there are $K$ classes.

image

parameters
(or weights)

bias

$$f(x, W, b) = Wx + b$$

Result: $K$ x 1
i.e. $K$ numbers
indicating class
scores

Image size: [128x128x3]
A vector of length 49152

$K$ x 49152

49152 x 1

$K$ x 1

# Multi-class Classification

Example with a 2x2 image and 3 classes:

$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0.0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{b} = \underbrace{\begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}}_{\mathrm{f}(x,W,b)}$$

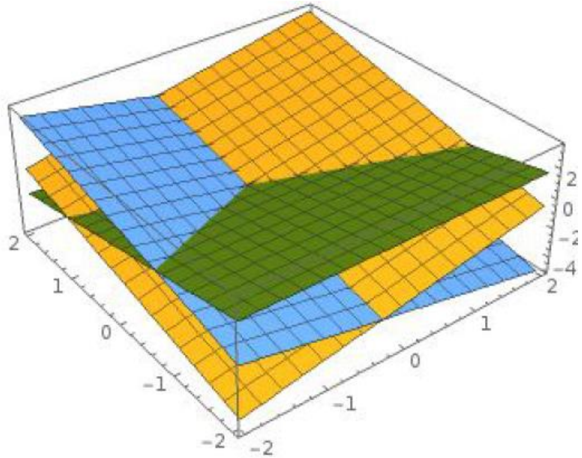
Class 1 score: $s_1$

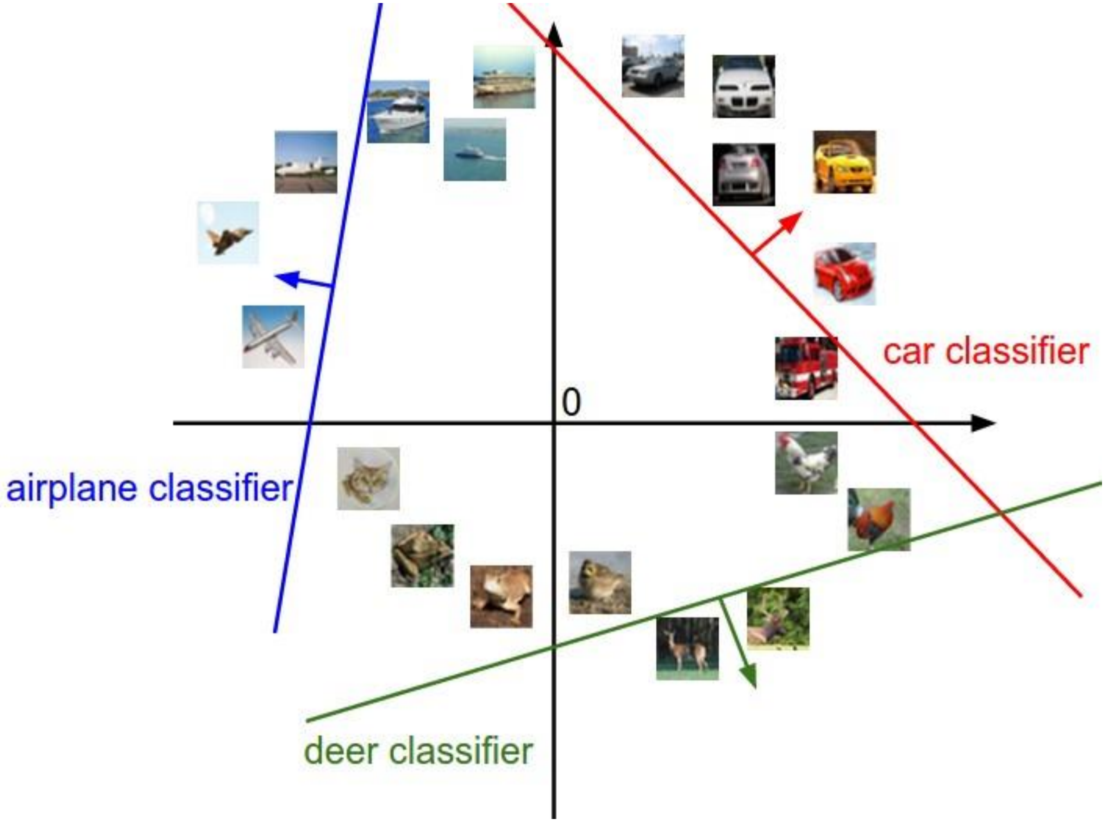
Class 2 score: $s_2$

Class 3 score: $s_3$

# Multi-class Classification

What does a linear classifier do?




Plot created using Wolfram Cloud

# Softmax Function

In binary classification, sigmoid function was enough to convert the scores into two probabilities where they add up to 1.

$$P(y=0|\ x, w\ ) + P(y=1|\ x, w\ ) = 1.$$

In multi-class classification, label $y$ can take on $K$ different values.

Let $s_j = f(x, w_j)$ , i.e. score for the $j^{\text{th}}$ class.

Given a test input $x$, we want our hypothesis to estimate the probabilities for each value of $j$ = 1,..,$K$ (probability of each class).

We need a $K$ dimensional vector whose elements sum up to 1.

$$\begin{bmatrix} P(y=1|x,W) \\ P(y=2|x,W) \\ \vdots \\ P(y=K|x,W) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} e^{s_j}} \begin{bmatrix} e^{s_1} \\ e^{s_2} \\ \vdots \\ e^{s_K} \end{bmatrix}$$

Softmax function

# Multi-class Cross-Entropy Loss

Remember the loss in binary classification:

$$J(w) = \frac{1}{m}\sum_{i=1}^{m}(-y^{(i)}\log(h_w(x^{(i)})) - (1-y)\log(1-h_w(x^{(i)}))$$

With softmax, it can be generalized to *K* classes:

$$J(W) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} -1\{y^{(i)} = k\}\log\frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}}$$

where $1\{\cdot\}$ is the indicator function such that

$1\{a\ true\ statement\} = 1$ and $1\{a\ false\ statement\} = 0$

# Softmax Loss

All together, softmax loss for sample $i$ :

$$L_i = -\log \frac{e^{s_{y^{(i)}}}}{\sum_{j=1}^{K} e^{s_j}}$$

where $y^{(i)}$ is the correct class label.

# Softmax Classifier Example

Suppose 3 classes. With some $W$, the scores f($x$,$W$,$b$) are:

$$L_i = -\log \frac{e^{s_{y^{(i)}}}}{\sum_{j=1}^{K} e^{s_j}}$$

| | scores | $\xrightarrow{\text{exp}}$ | unnormalized probabilities | $\xrightarrow{\text{normalize}}$ | normalized probabilities | |
|---|---|---|---|---|---|---|
| cat | **3.2** | | **24.5** | | **0.13** | $\rightarrow$ $L_i$ = -log(0.13) = 0.89 |
| car | 5.1 | | 164.0 | | 0.87 | |
| frog | -1.7 | | 0.18 | | 0.00 | |

Observe the probabilistic interpretation in softmax classifier

# Softmax Classifier Example continued

- What is the min/max possible loss?

- Usually at initialization $W$ are small numbers, so all *s* ~= 0. What is the loss?

$$L_i = -\log \frac{e^{s_{y^{(i)}}}}{\sum_{j=1}^{K} e^{s_j}}$$

| | unnormalized log probabilities | exp | unnormalized probabilities | normalize | normalized probabilities | | |
|---|---|---|---|---|---|---|---|
| cat | **3.2** | | **24.5** | | **0.13** | → | $L_i$ = -log(0.13) = 0.89 |
| car | 5.1 | → | 164.0 | → | 0.87 | | |
| frog | -1.7 | | 0.18 | | 0.00 | | |

# Softmax Classifier Example continued

| | | | | |
|---|---|---|---|---|
| cat | **3.2** | 1.3 | 3.7 | 0.03 |
| car | 5.1 | **4.9** | **134.3** | **0.92** |
| frog | -1.7 | 2.0 | 7.4 | 0.05 |
| loss | 0.89 | | | |

$\xrightarrow{\text{exp}}$ $\xrightarrow{\text{norm.}}$

$\rightarrow L_i = -\log(0.92) = 0.04$

# Softmax Classifier Example continued



| | | | | exp $\rightarrow$ | norm. $\rightarrow$ | |
|---|---|---|---|---|---|---|
| cat | **3.2** | 1.3 | 2.2 | 9.03 | 0.43 | |
| car | 5.1 | **4.9** | 2.5 | 12.18 | 0.57 | |
| frog | -1.7 | 2.0 | **-3.1** | **0.05** | **0.002** | $\rightarrow L_i$ = -log(0.002) = 2.7 |
| loss | 0.89 | 0.04 | | | | |

# Regularization

- We want smaller weights, results in smoother models

$$J(W) = \underbrace{\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} 1\{y^{(i)} = k\}\log\frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}}}_{\text{data loss}} + \underbrace{\overset{\text{regularization strength}}{\overset{\uparrow}{\lambda}} R(W)}_{\text{regularization loss}}$$

- L2 regularization $\longrightarrow$ $R(W) = \sum_k \sum_l W_{k,l}^2$
- L1 regularization $\longrightarrow$ $R(W) = \sum_k \sum_l |W_{k,l}|$
- Elastic net (L1 + L2) $\longrightarrow$ $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$
- Dropout (used with NNs, will see later)

# Regularization

- Regularization means, the coefficients ($w$) are chosen so that not only to predict well on the training data, but also the magnitude of coefficients to be as small as possible.
- Intuitively, it means that no input dimension can have a very large influence on the scores all by itself.
- Motivation (L2 regularization example):

$$\left.\begin{aligned} x &= [1,1,1,1] \\ w_A &= [1,0,0,0] \\ w_B &= [0.25,0.25,0.25,0.25] \end{aligned}\right\} \quad w_A^{\top}x = w_B^{\top}x = 1$$

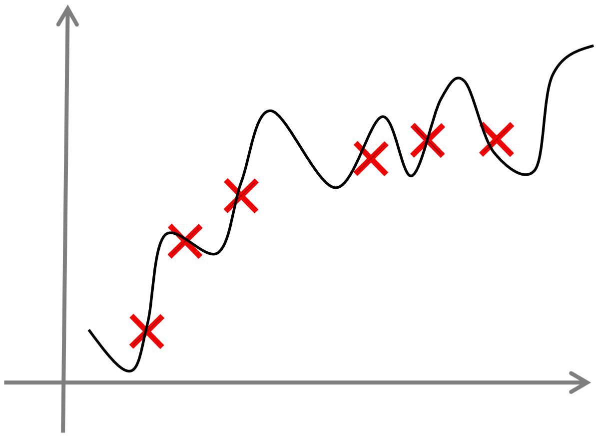
preferable

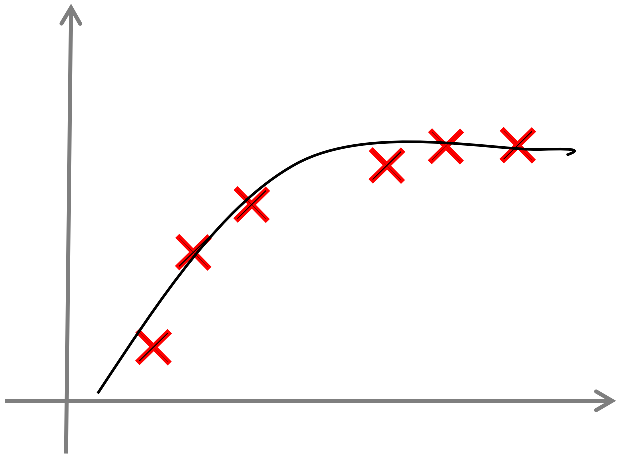
L2 penalty of $w_A$ is 1.0 while the L2 penalty of $w_B$ is 0.25.

- Regularization helps a model to avoid <u>overfitting</u>.

# Regularization

Regularization restricts a model to avoid <u>overfitting</u>.



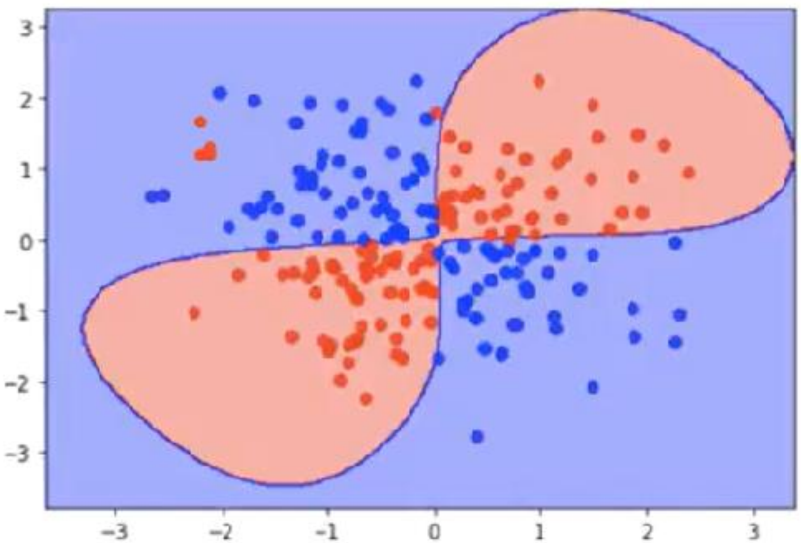


Not regularized well.
Does not work well on test data.
Needs more regularization
($\lambda$ to be increased)

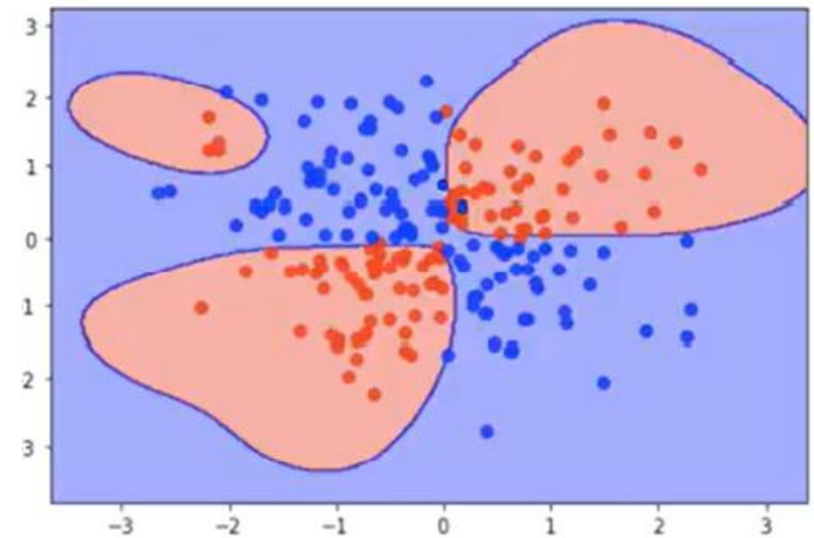
Regularized well.
Works well on test data.

# Regularization

Which one has the best regularization (left or right)?




Good amount of regularization

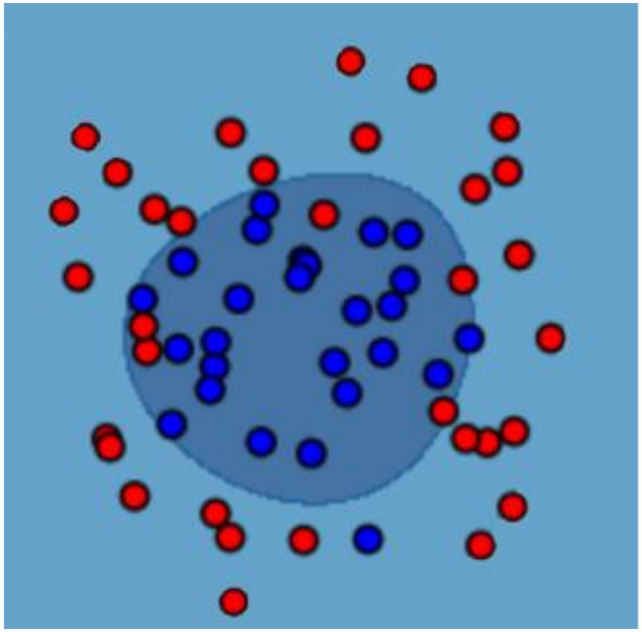



Needs more regularization

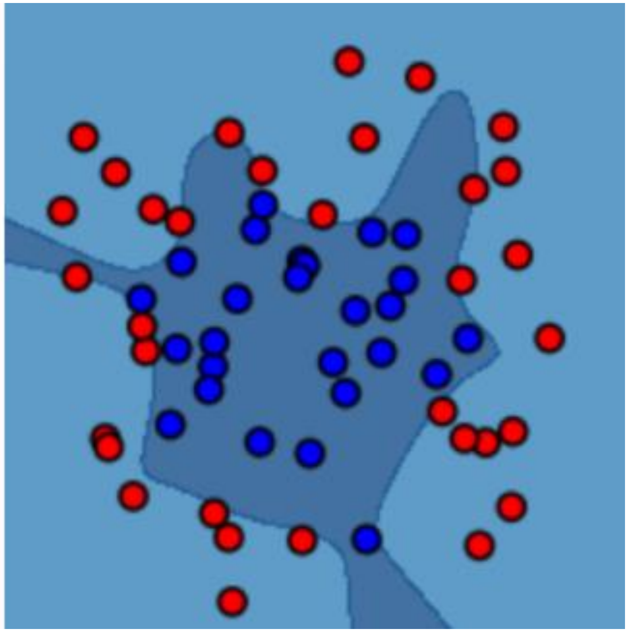
($\lambda$ to be increased)

# Regularization

Which one has the best regularization (left or right)?



Good amount of regularization



Needs more regularization
($\lambda$ to be increased)

# Linear Classification Loss Visualization

http://vision.stanford.edu/teaching/cs231n-demos/linear-classify/