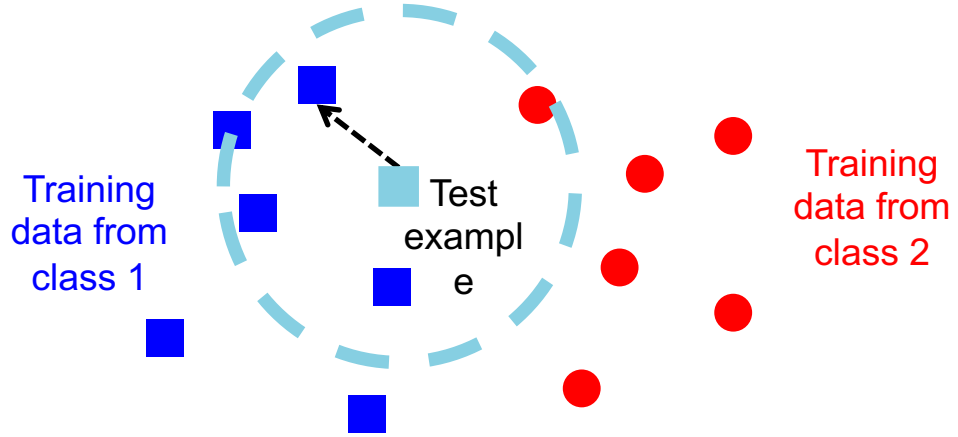
Image Classification: Linear Classifier and Optimization

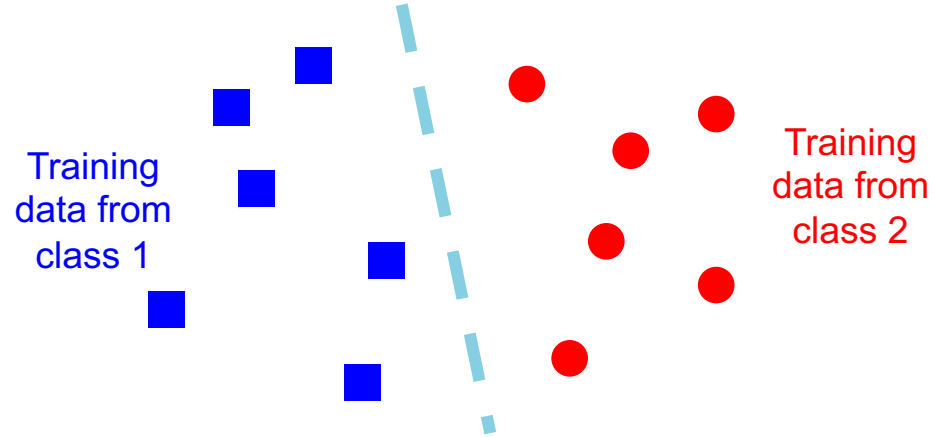
Xiaolong Wang

Last Class

- K-nearest neighbor classifier



- Linear classifier with linear regression



Last Class

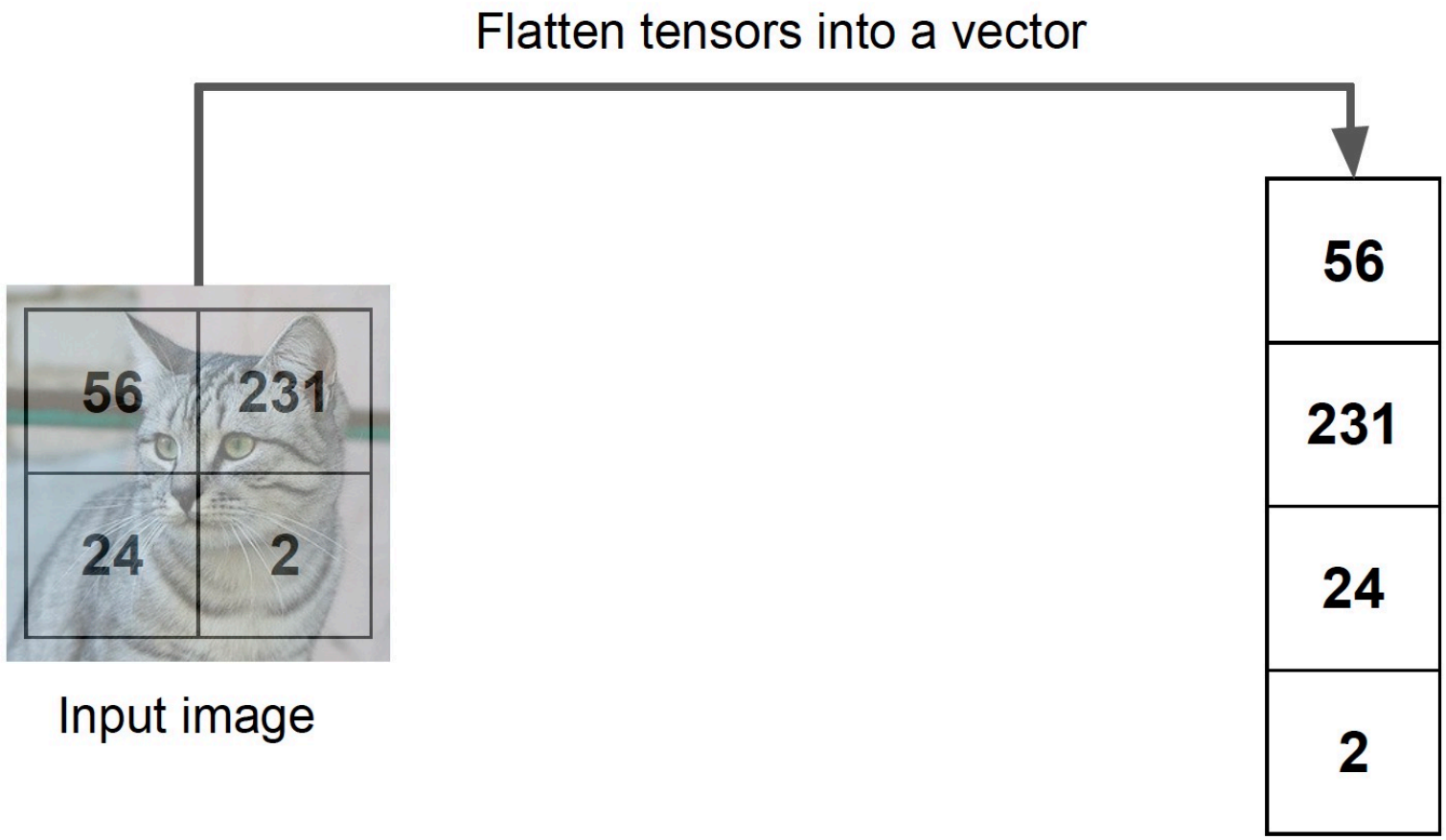
- Nearest Neighbors: Do not need training, but need to go over the whole dataset in testing
- Linear Classifier: Linear regression might not be the best fit for classification
- We have not talked about how to optimize with the loss function

Today

- More on linear classifier
- Optimization
- Regularization
- SoftMax

Logistic Regression

Linear Regression : 4-pixel example



Recall Supervised Learning

$$y = f(x)$$

output label — classifier — input image

- **Training** (or **learning**): given a *training set* of labeled examples $\{(x_1, y_1), \ldots, (x_N, y_N)\}$, train a predictor f

- **Testing** (or **inference**): apply predictor f to a new *test example* x and output the predicted value $y = f(x)$

The problem of linear regression

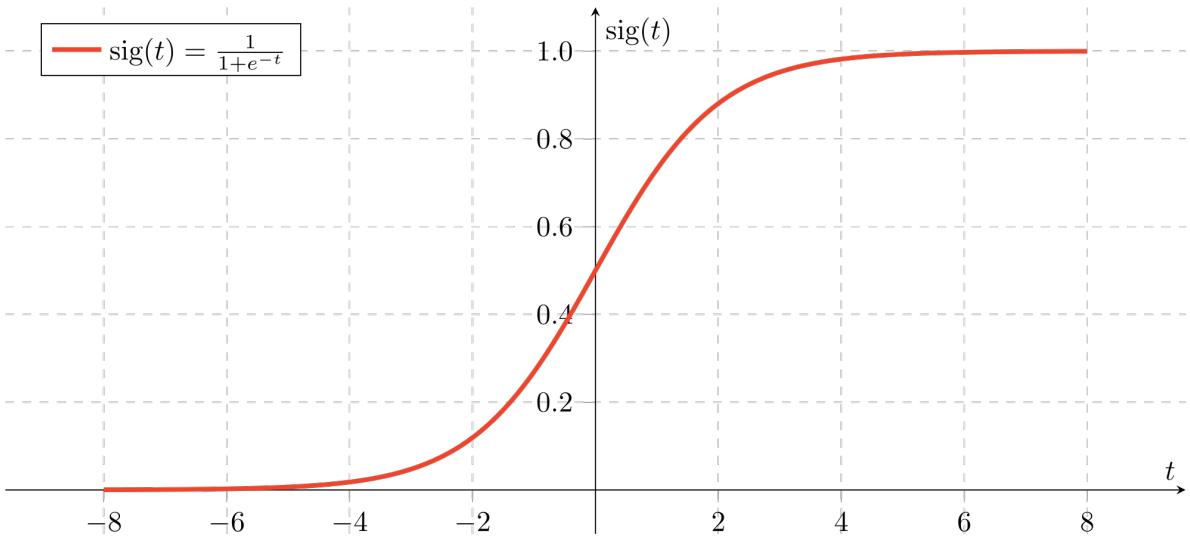
$$(f(x_i) - y_i)^2 = (Wx_i - y_i)^2$$

- Removing bias for convenience
- $y \in \{-1,1\}$. However, there is no regulation to constrain the output range.
- Can we restrict the output to a certain range?

The Sigmoid Function (2-class)

- Squash the linear response of the classifier to the interval $[0,1]$ to represent the prediction probability:

$$\sigma(Wx) = \frac{1}{1 + \exp(-Wx)}$$



The Sigmoid Function (2-class)

- Thus we let $P(y = 1|x) = \sigma(Wx) = \frac{1}{1+\exp(-Wx)}$

- For the other category:

$$P(y = -1|x) = 1 - P(y = 1|x) = 1 - \sigma(Wx)$$

$$= 1 - \frac{1}{1 + \exp(-Wx)} = \frac{\exp(-Wx)}{1 + \exp(-Wx)}$$

$$= \frac{1}{\exp(Wx) + 1} = \sigma(-Wx)$$

The sigmoid function is *symmetric:* $1 - \sigma(Wx) = \sigma(-Wx)$

Logistic regression: Training Objective

- Given: $\{(x_i, y_i), i = 1, \dots, n\}$, $y_i \in \{-1,1\}$

$$\hat{L}(W) = -\frac{1}{N}\sum_{i=1}^{N} \log P(y_i|x_i)$$

$$= -\frac{1}{N}\sum_{i:y_i=1} \log \sigma(Wx_i) - \frac{1}{N}\sum_{i:y_i=-1} \log[1 - \sigma(Wx_i)]$$

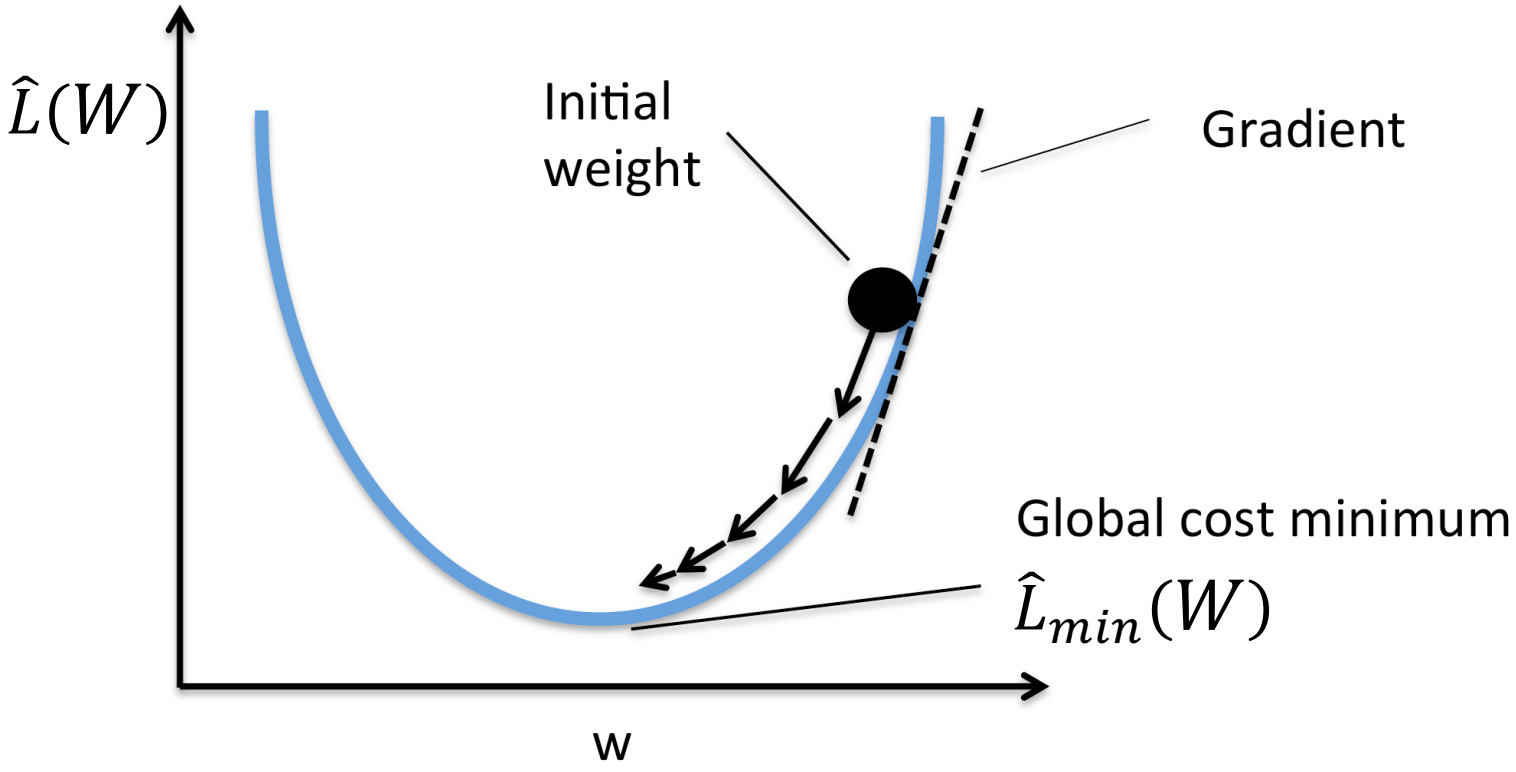
$$= -\frac{1}{N}\sum_{i:y_i=1} \log \sigma(Wx_i) - \frac{1}{N}\sum_{i:y_i=-1} \log[\, \sigma(-Wx_i)]$$

$$= -\frac{1}{N}\sum_{i} \log \sigma(y_i W x_i)$$

Optimization

Gradient descent

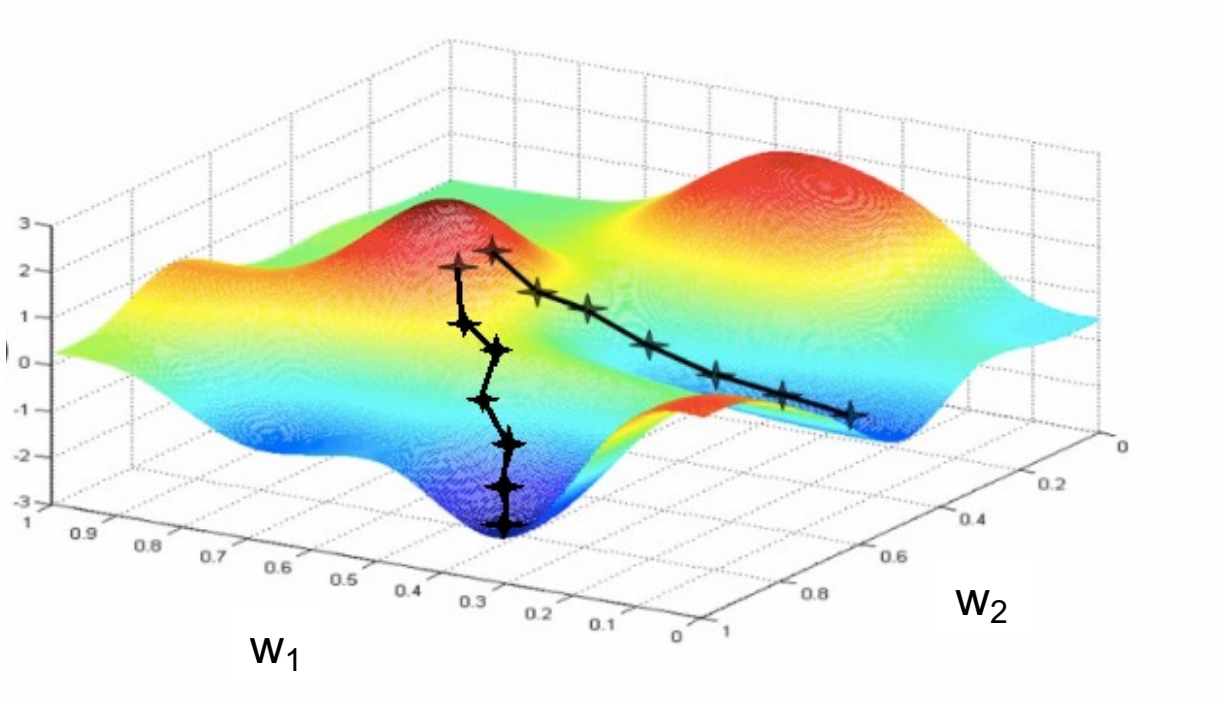
- Start with some initial estimate of W.
- At each step, compute the gradient $\nabla\hat{L}(W)$.
- Move in the opposite direction of the gradient



2D Example

Take a small step in the *opposite* direction, using learning rate α:

$$W \leftarrow W - \alpha \nabla \hat{L}(W)$$



Source: Svetlana Lazebnik

Gradient descent for logistic regression

$$\hat{L}(W) = -\frac{1}{N}\sum_{i=1}^{N} \log \sigma(y_i W x_i)$$

$$\nabla \hat{L}(W) = -\frac{1}{N}\sum_{i=1}^{N} \nabla_w \log \sigma(y_i W x_i)$$

Derivative rule:

$$\left[\log\left(f(x)\right)\right]' = \frac{f'(x)}{f(x)}$$

Gradient descent for logistic regression

$$\hat{L}(W) = -\frac{1}{N}\sum_{i=1}^{N} \log \sigma(y_i W x_i)$$

$$\nabla \hat{L}(W) = -\frac{1}{N}\sum_{i=1}^{N} \nabla_w \log \sigma(y_i W x_i)$$

$$= -\frac{1}{N}\sum_{i=1}^{N} \frac{\nabla_W \sigma(y_i W x_i)}{\sigma(y_i W x_i)}$$

Derivative rule:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) = \sigma(x)\sigma(-x)$$

Gradient descent for logistic regression

$$\hat{L}(W) = -\frac{1}{N}\sum_{i=1}^{N} \log \sigma(y_i W x_i)$$

$$\nabla \hat{L}(W) = -\frac{1}{N}\sum_{i=1}^{N} \nabla_w \log \sigma(y_i W x_i)$$

$$= -\frac{1}{N}\sum_{i=1}^{N} \frac{\nabla_W \sigma(y_i W x_i)}{\sigma(y_i W x_i)}$$

$$= -\frac{1}{N}\sum_{i=1}^{N} \frac{\sigma(y_i W x_i)\sigma(-y_i W x_i) y_i x_i}{\sigma(y_i W x_i)}$$

Gradient descent for logistic regression

$$\hat{L}(W) = -\frac{1}{N}\sum_{i=1}^{N} \log \sigma(y_i W x_i)$$

$$\nabla \hat{L}(W) = -\frac{1}{N}\sum_{i=1}^{N} \nabla_w \log \sigma(y_i W x_i)$$

$$= -\frac{1}{N}\sum_{i=1}^{N} \frac{\nabla_W \sigma(y_i W x_i)}{\sigma(y_i W x_i)}$$

$$= -\frac{1}{N}\sum_{i=1}^{N} \frac{\sigma(y_i W x_i)\sigma(-y_i W x_i) y_i x_i}{\sigma(y_i W x_i)}$$

$$= -\frac{1}{N}\sum_{i=1}^{N} \sigma(-y_i W x_i) y_i x_i$$

5 mins break

Gradient descent for logistic regression

Update rule:

$$W \leftarrow W - \alpha \nabla \hat{L}(W)$$

$$\nabla \hat{L}(W) = -\frac{1}{N} \sum_{i=1}^{N} \sigma(-y_i W x_i) y_i x_i$$

Combine both:

$$W \leftarrow W + \alpha \frac{1}{N} \sum_{i=1}^{N} \sigma(-y_i W x_i) y_i x_i$$

We update the parameters iteratively, compute the gradient over all examples each gradient step

Gradient descent for logistic regression

$$W \leftarrow W - \alpha \nabla \hat{L}(W)$$

- We can set $\alpha = 0.1$ or other smaller number if the parameters diverge.

- However, it might be too slow to perform one update by calculating the gradients over all the training examples.

- Can we approximate the gradients more efficiently?

Stochastic gradient descent (SGD)

- We approximate the gradient of the whole dataset $\nabla \hat{L}(W)$ by using only ONE example (x_i, y_i) as $\nabla L(W, x_i, y_i)$

- Instead of

$$W \leftarrow W + \alpha \frac{1}{N} \sum_{i=1}^{N} \sigma(-y_i W x_i) y_i x_i$$

- Use

$$W \leftarrow W + \alpha \, \sigma(-y_i W x_i) y_i x_i$$

- Since gradient on each example is unstable, it is “stochastic”

Stochastic gradient descent (SGD)

- Instead of using only one example, or the whole dataset, we can try something in between.

- Sample a batch of examples (e.g., $B = 128$ examples) to compute the gradients for update

$$W \leftarrow W + \alpha \frac{1}{B} \sum_{i=1}^{B} \sigma(-y_i W x_i) y_i x_i$$

- batch size: A trade off between accurate gradient approximation and efficiency

Gradients for linear regression

$$\hat{L}(W) = \frac{1}{N}\sum_{i=1}^{N}(Wx_i - y_i)^2$$

Compute the gradient:

$$\nabla\hat{L}(W) = \frac{1}{N}\sum_{i=1}^{N}\nabla_W(Wx_i - y_i)^2$$

$$= \frac{1}{N}\sum_{i=1}^{N}2(Wx_i - y_i)\,x_i$$

Gradients for linear regression

Update rule:

$$W \leftarrow W - \alpha \nabla \hat{L}(W)$$

$$\nabla \hat{L}(W) = \frac{1}{N} \sum_{i=1}^{N} 2(W x_i - y_i)\, x_i$$

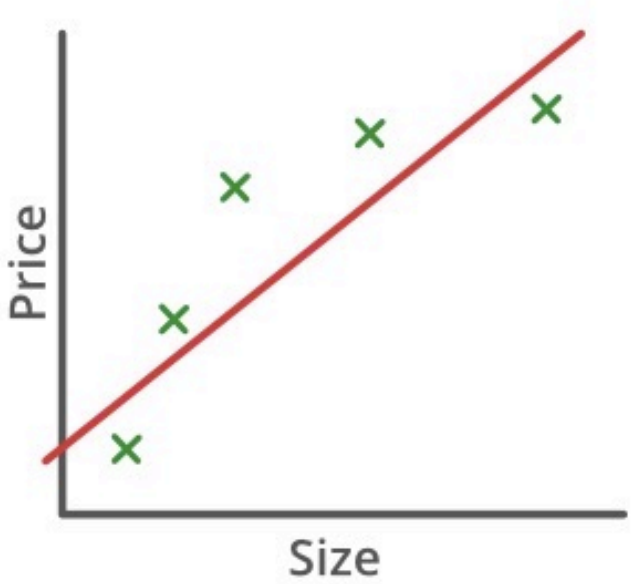
Combine both:

$$W \leftarrow W - \alpha \frac{1}{N} \sum_{i=1}^{N} 2(W x_i - y_i)\, x_i$$

Regularization

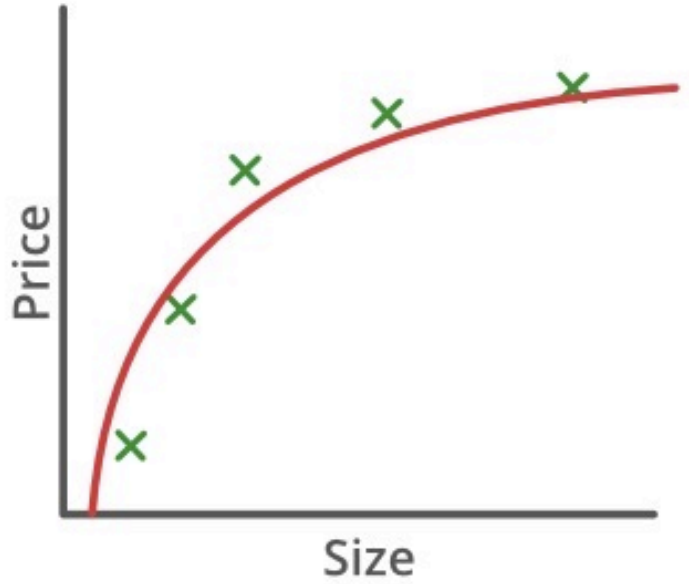
Overfitting

We want to estimate a function to fit the green data points.



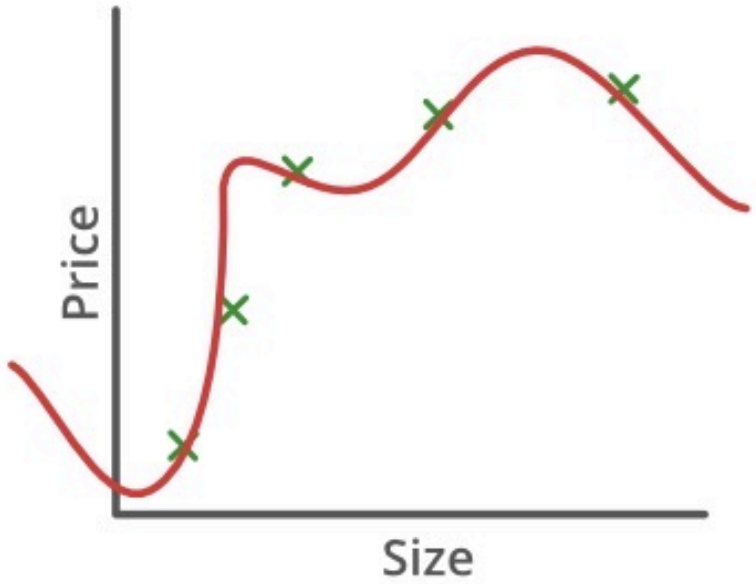
$\theta_0 + \theta_1 x$

Underfit



$\theta_0 + \theta_1 x + \theta_2 x^2$

Ideal fit

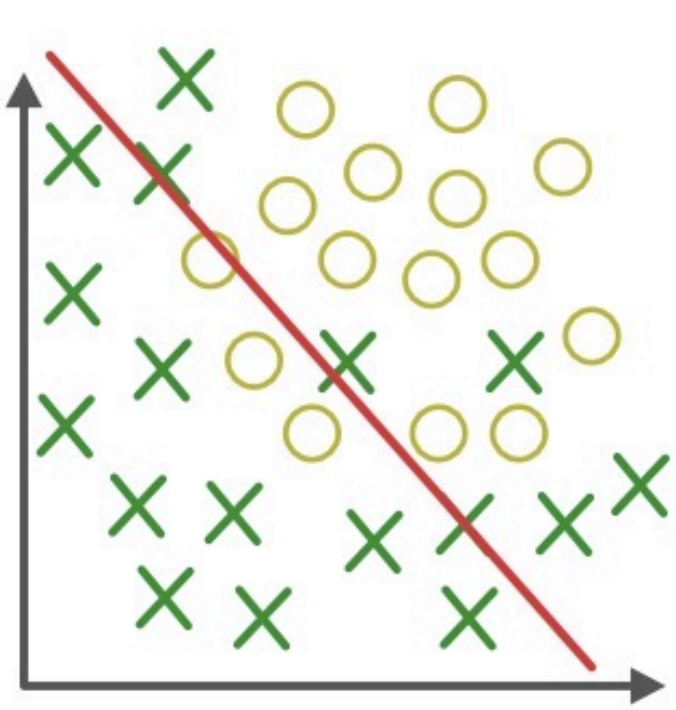


$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_2 x^2 + \theta_2 x^2$

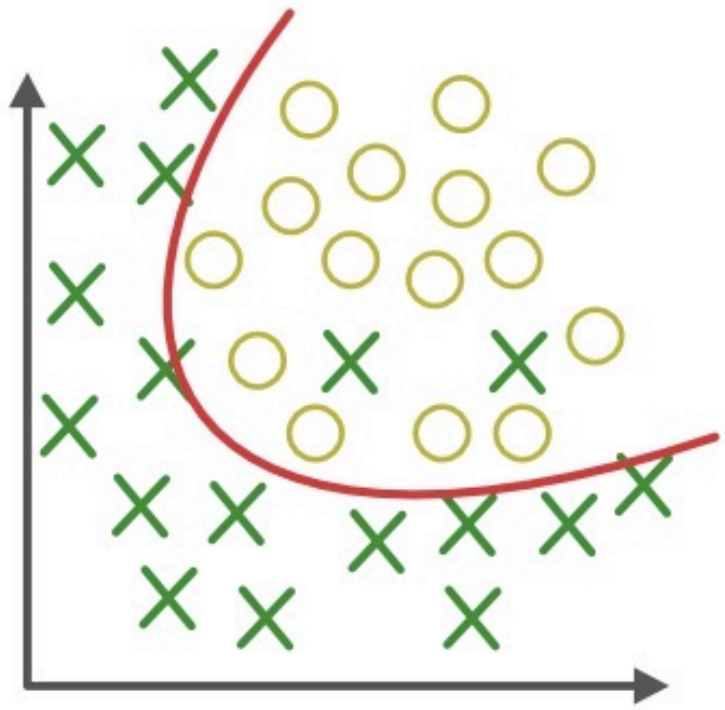
Overfit

Overfitting

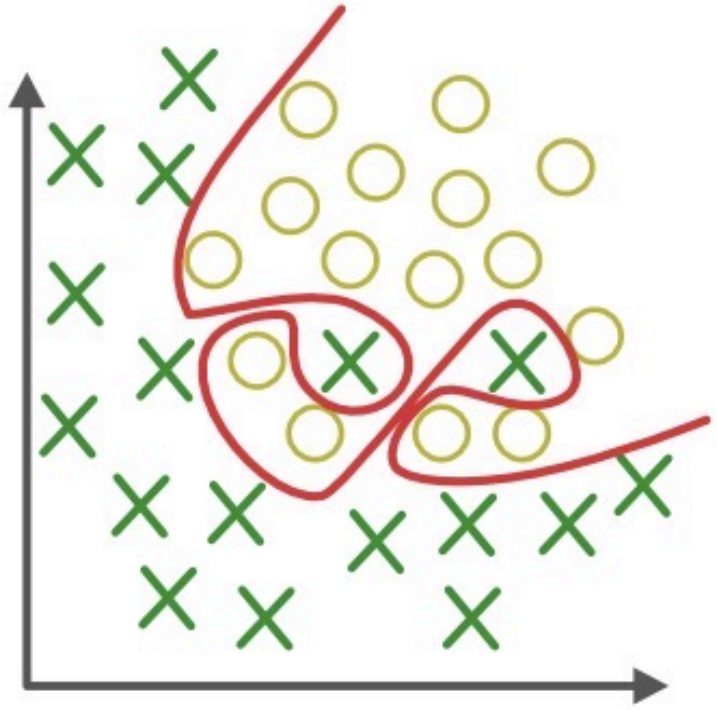
We want to estimate a classifier to separate two types of data.



Underfit



Ideal fit



Overfit

One trick to prevent overfitting

- Adding regularization in training objective, L2 regularization:

$$\hat{L}(W) = \frac{\lambda}{2}\left|\left|W\right|\right|^2 + \frac{1}{n}\sum_{i=1}^{n} L(W, x_i, y_i)$$

L2 regularization — Loss from data

$$W \leftarrow W - \alpha\,(\lambda\, W + \nabla_W \frac{1}{n}\sum_{i=1}^{n} L(W, x_i, y_i)\,)$$

To prevent overfitting

$$W \leftarrow W - \alpha \left(\lambda W + \nabla_W \frac{1}{n} \sum_{i=1}^{n} L(W, x_i, y_i)\right)$$

Gradients from
L2 regularization

Also called weight decay

We usually set $\lambda = 0.00005$ in neural networks

Softmax

Softmax

- Goal: Normalize a set of predictions / responses $(z_1, \dots, z_c)$ into a vector of "probabilities":

$$\text{softmax}(z_1, \dots, z_c) = \left(\frac{\exp(z_1)}{\sum_j \exp(z_j)}, \dots, \frac{\exp(z_C)}{\sum_j \exp(z_j)} \right)$$

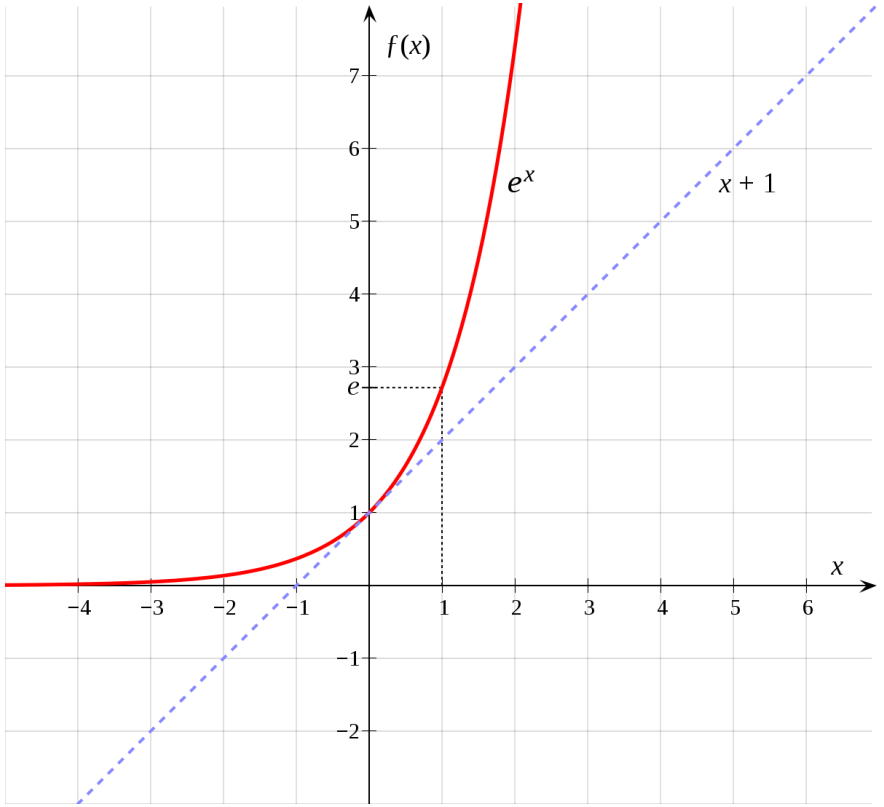
- All the responses are normalized between 0 and 1, and they are summed to 1.

Softmax

- The problem of $\exp(z_i)$ is that if z_i is large then $\exp(z_i)$ will be out of float.

- Take $K = \max_i z_i$

$$\frac{\exp(z_k)}{\sum_j \exp(z_i)} = \frac{\exp(z_k - K)}{\sum_j \exp(z_i - K)}$$



Softmax Loss

- Similar to Sigmoid, we use the negative log of the prediction as the loss:

$$-\log P_W(y_i|x_i) = -\log\left(\frac{\exp\left(W_{y_i}x_i\right)}{\sum_j \exp\left(W_j x_i\right)}\right)$$

- We will optimize this loss function w.r.t each W_{y_i}

In this class

- More on linear classifier
- Optimization
- Regularization
- SoftMax

Next class

- Training Neural Networks with Softmax

- Back-propagation