Artificial Intelligence and Machine Learning

Logistic Regression

Lecture 2: Outline

- Linear Regression (Review)
- Logistic Regression (Classification)
- Optimization

Recap

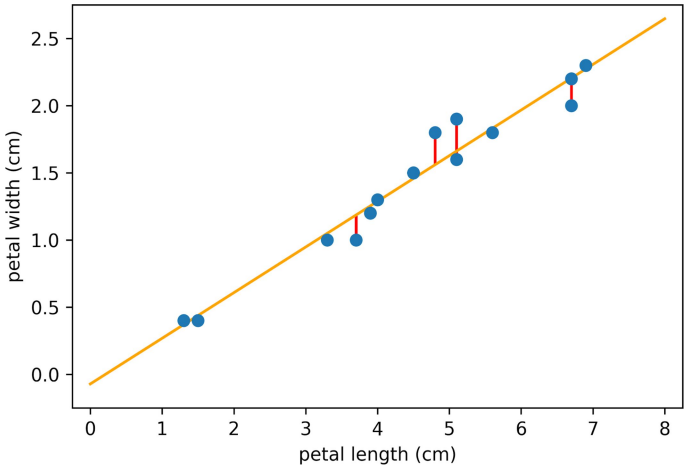
$$\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}, \mathbf{x}_i \in \mathbb{R}^n, y_i \in \mathbb{R}$$

Design your model

- Input scalar linear model (line fitting)
- Fitting polynomials (synthetically designing features from a one-dimensional input)

Design your loss function

- We used mean squared error loss throughout

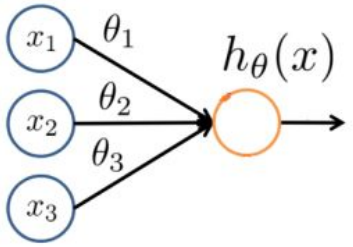


Finding optimal parameter fitting

- Closed form solution to the linear least squares?
- Why is it linear least squares?
- Solution is closed form

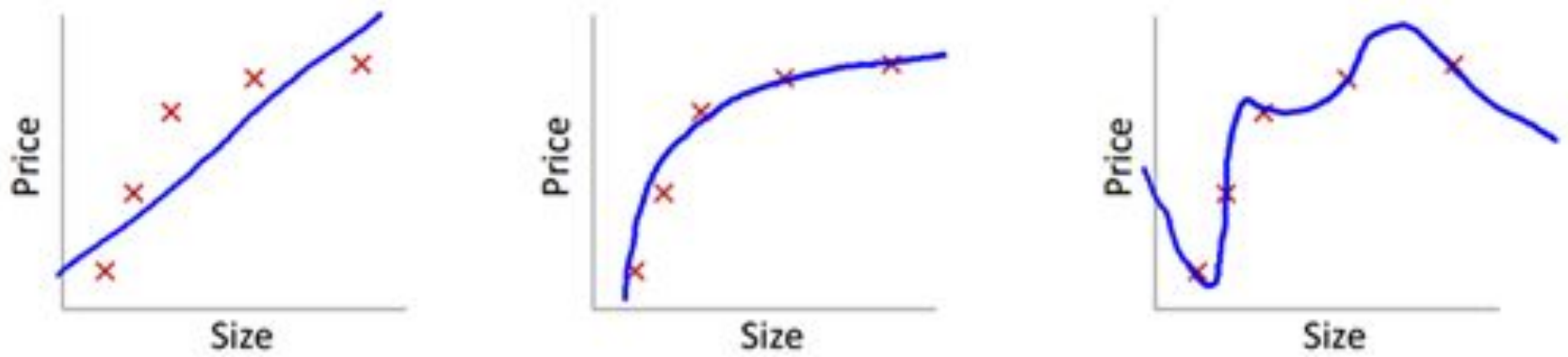
Logistic regression or Classification

Regression VS classification

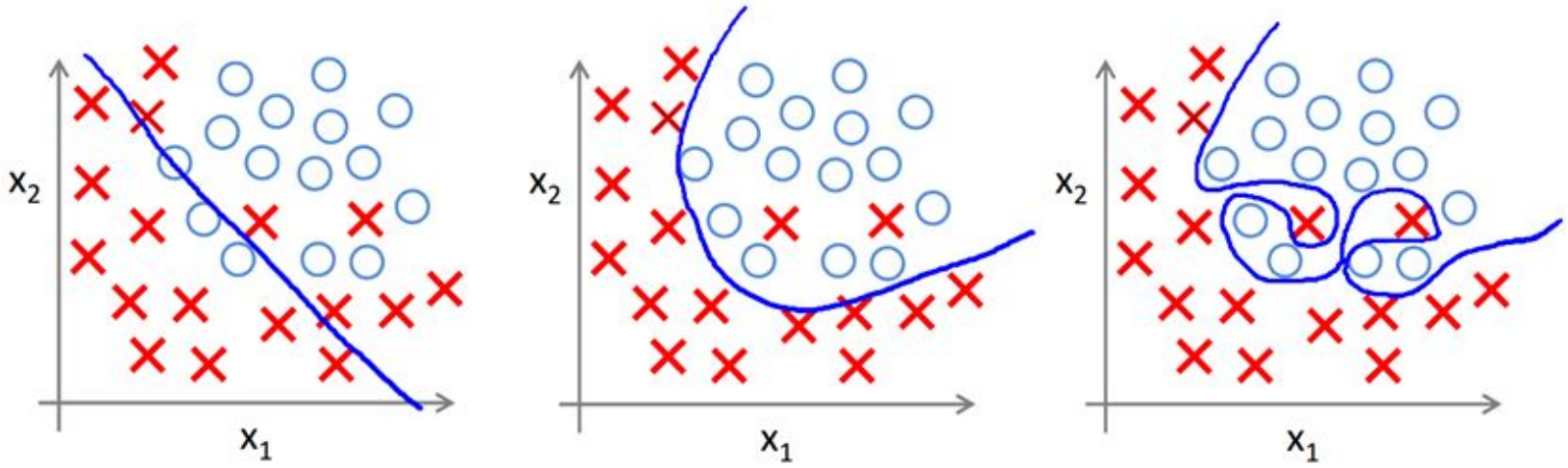


$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

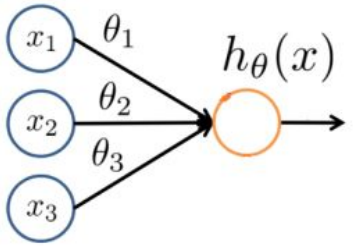
- Regression (linear and polynomial): for prediction



- Classification:



Regression VS classification



$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

Income prediction -> regression

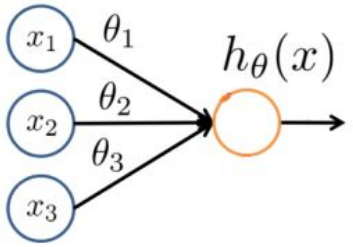
Male or Female -> classification

House price -> regression

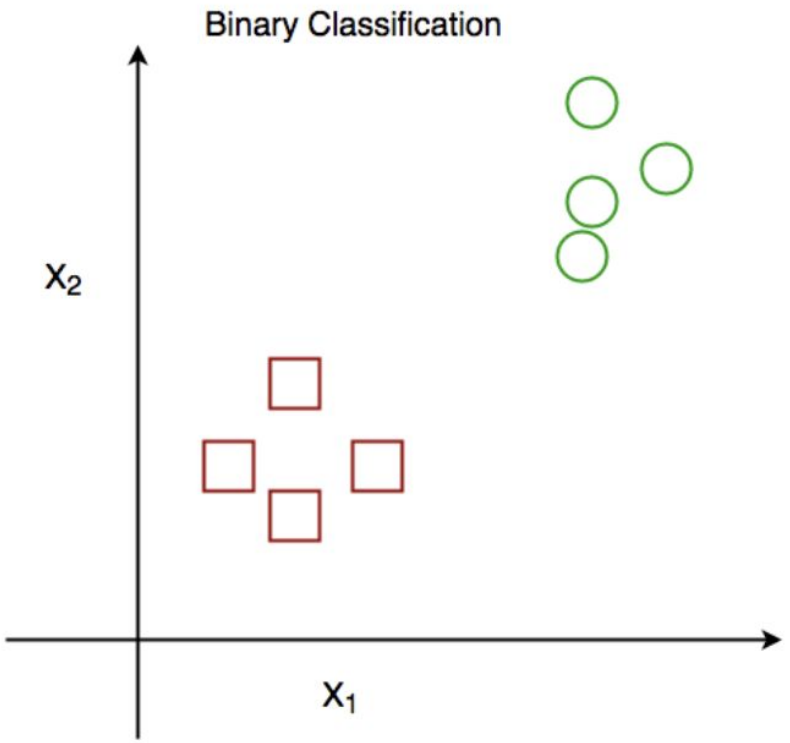
Spam detection -> classification

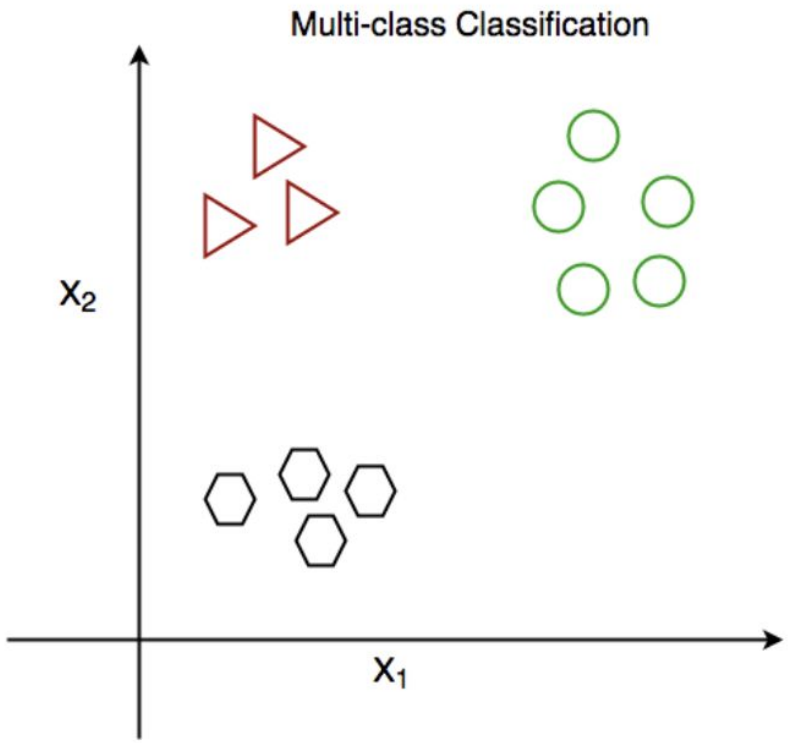
Image recognition -> classification

Classification



$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

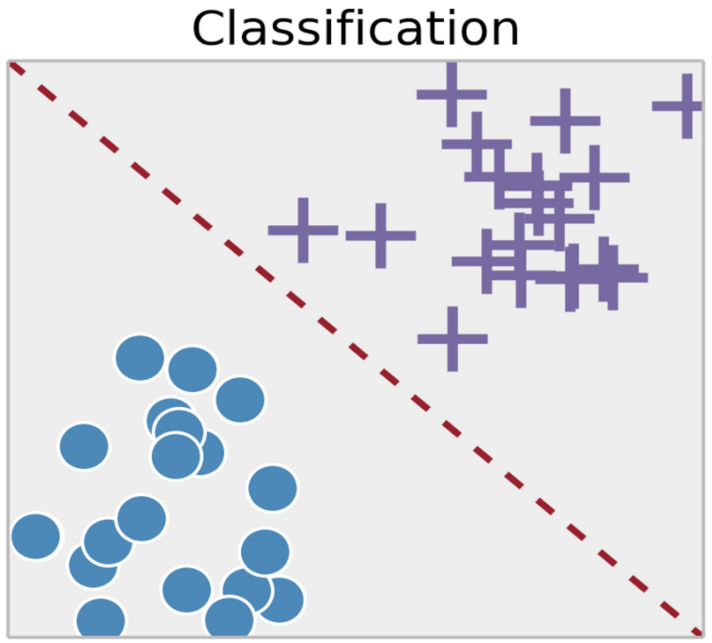




Logistic Regression

- Regular vs Fraudulent transaction
- Spam vs Non-spam emails
- Benign vs Malignant tumors
- Rising vs Falling stocks

$$\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}, \mathbf{x}_i \in \mathbb{R}^n, y_i \in \{0, 1\}$$



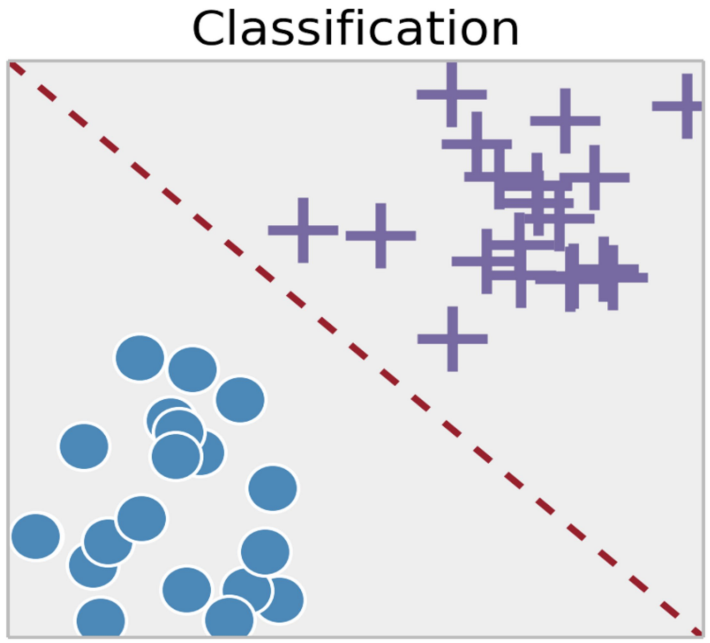
Logistic Regression vs linear regression

Linear Regression	Logistic Regression
For Regression	For Classification
We predict the target value for any input value	We predict the probability that the input value belongs to the specific target
Target: Real Values, continuous values	Target: Discrete values
Graph: Straight Line	Graph: S-curve

Logistic Regression

Despite the name, logistic regression is a

classification algorithm



Logistic Regression is a linear model with a "special function" that helps us use this linear model for classification

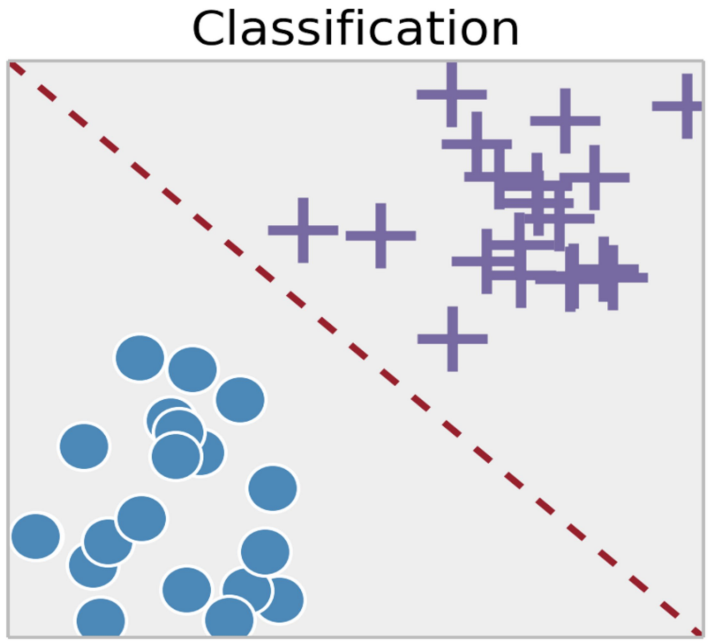
$$\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}, \mathbf{x}_i \in \mathbb{R}^n, y_i \in \{0, 1\}$$

Linear Model in Disguise

$$\hat{y} = \mathbf{w}^T \mathbf{x}$$

$$\mathbf{w} = [w_0, w_1, \cdots, w_m]^T$$

$$\mathbf{x} = [1, x^1, \cdots, x^m]^T$$



$$\{(\mathbf{x}_i, y_i)\}_{i=1}^N, \mathbf{x}_i \in \mathbb{R}^n, y_i \in \{0, 1\}$$

Challenge:

$$\hat{y} = \mathbf{w}^T \mathbf{x}$$

- The above equation predicts continuous outputs.
- Unbounded output: There is no natural constraint on the output, and prediction could be any real number
- Intuitively, it also doesn't make sense for $\hat{y}$ to take values larger than 1 or smaller than 0 when we know that y $\in$ {0, 1}.

Solution

- In logistic regression, we define the problem as follows:
  - Instead of just predicting the class, give the probability of the instance being that class

$$\hat{y} = p(y \mid \boldsymbol{x})$$

- Thus we need a function that transforms the output into a probability distribution.

$$\hat{y} = \sigma\left(\mathbf{w}^T \mathbf{x}\right)$$

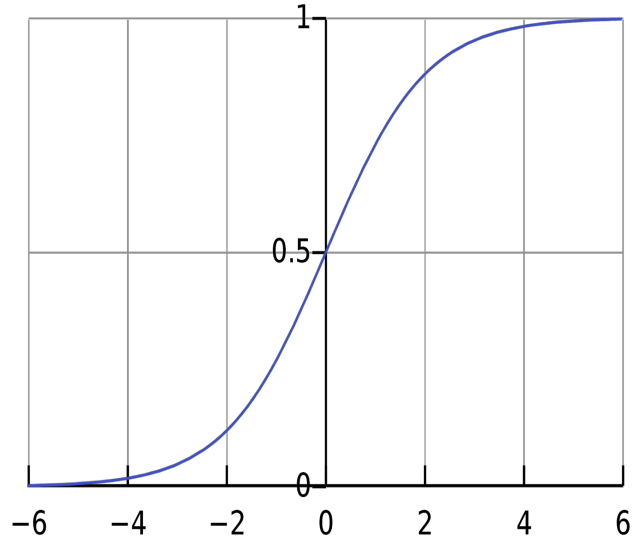
Sigmoid Function

Widely used in classification

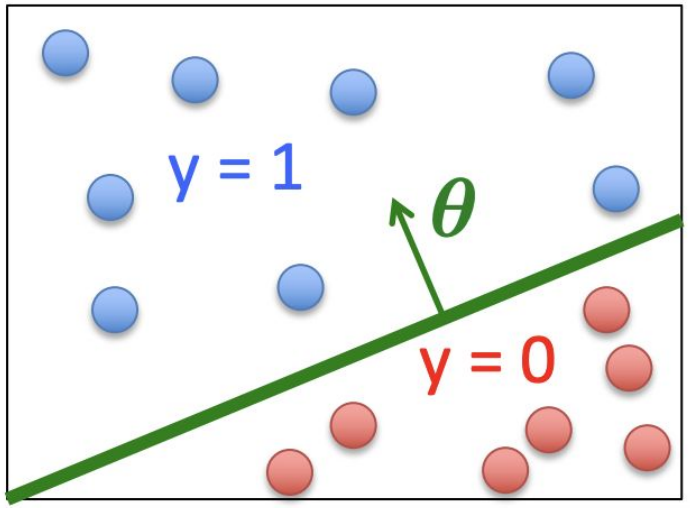
$$\sigma(z) = \frac{1}{1 + exp(-z)}$$

$$\lim_{z \to \infty} \sigma(z) = 1$$

$$\lim_{z \to -\infty} \sigma(z) = 0$$

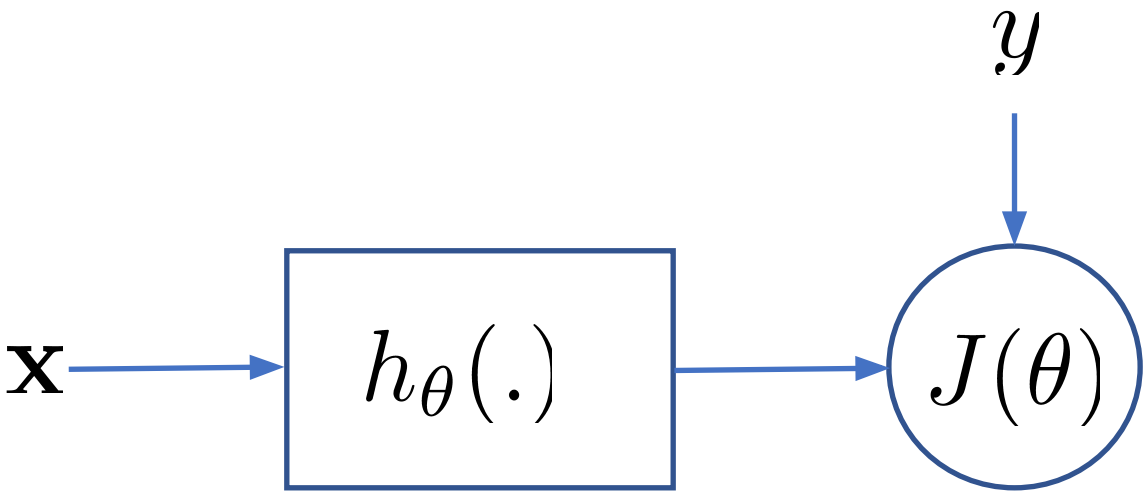


- Assume a threshold:
  - Predict y=1 if $\sigma(z)$ >= 0.5
  - Predict y=0 if $\sigma(z)$ < 0.5



Cost Function

- We want to minimize the discrepancy between our model hypothesis and the observed label.



What type of loss to use?

MSE?

$$J = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

- Not the best when it comes to classification
- Leads to suboptimal results
- Not ideal for probability output

Binary Cross Entropy Loss

$$J(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \text{compare}(y_i, \sigma(\mathbf{w}^T\mathbf{x}_i))$$

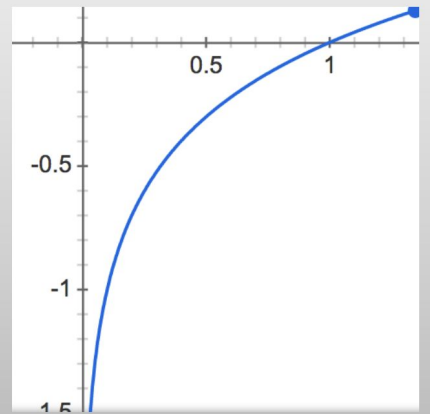
$$J(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\sigma(\mathbf{w}^T\mathbf{x}_i)) + (1-y_i)\log(1-\sigma(\mathbf{w}^T\mathbf{x}_i))$$

Intuition of Cost Function

- Cost of a single instance:

$$\text{cost}\left(\sigma(\mathbf{w}^T\mathbf{x}), y\right) = \begin{cases} -\log\left(\sigma(\mathbf{w}^T\mathbf{x})\right) & \text{if } y = 1 \\ -\log\left(1 - \sigma(\mathbf{w}^T\mathbf{x})\right) & \text{if } y = 0 \end{cases}$$

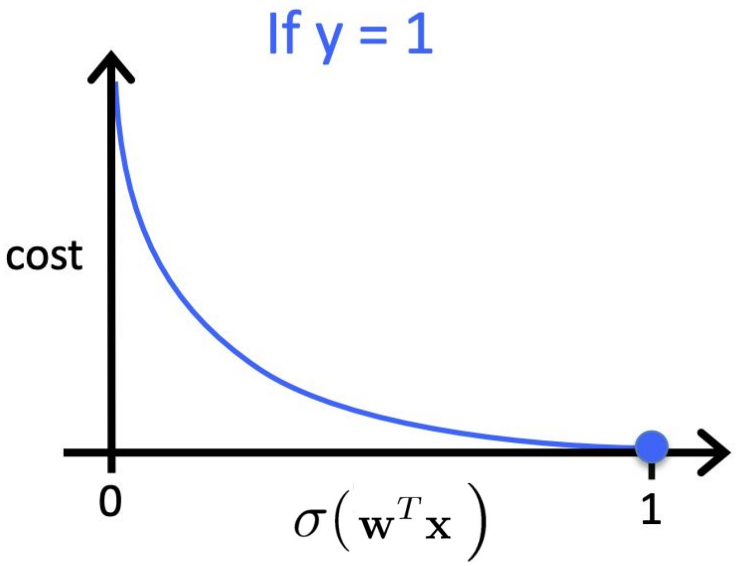
Aside: Recall the plot of log(z)



Intuition of Cost Function

- Cost of a single instance:

$$\text{cost}\left(\sigma(\mathbf{w}^T\mathbf{x}), y\right) = \begin{cases} -\log(\sigma(\mathbf{w}^T\mathbf{x})) & \text{if } y = 1 \\ -\log(1 - \sigma(\mathbf{w}^T\mathbf{x})) & \text{if } y = 0 \end{cases}$$



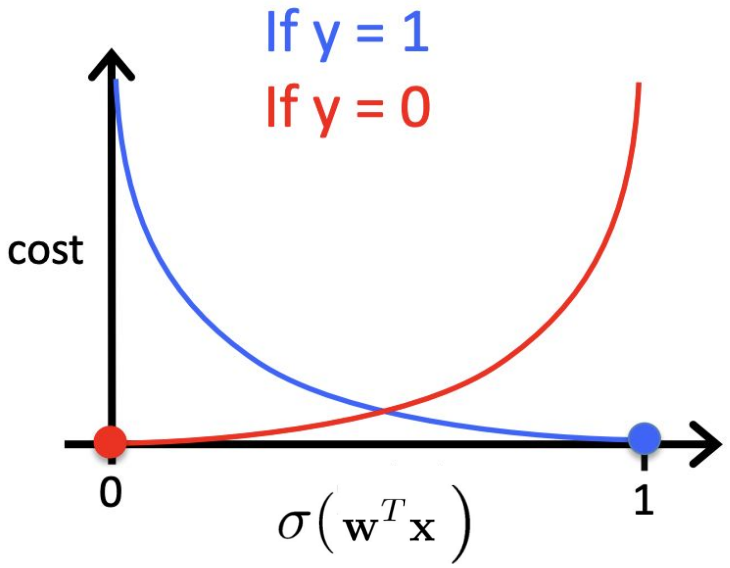
If y = 1

- Cost = 0 if prediction is correct
- As $\sigma(\mathbf{w}^T\mathbf{x}) \rightarrow 0$, cost $\rightarrow \infty$
- Captures intuition that larger mistakes should get larger penalties
  - e.g., predict $\sigma(\mathbf{w}^T\mathbf{x}) = 0$, but y = 1

Intuition of Cost Function

- Cost of a single instance:

$$\text{cost}\left(\sigma(\mathbf{w}^T\mathbf{x}), y\right) = \begin{cases} -\log\left(\sigma(\mathbf{w}^T\mathbf{x})\right) & \text{if } y = 1 \\ -\log\left(1 - \sigma(\mathbf{w}^T\mathbf{x})\right) & \text{if } y = 0 \end{cases}$$



If y = 0

- Cost = 0 if prediction is correct
- As $(1 - \sigma(\mathbf{w}^T\mathbf{x})) \to 0, \text{cost} \to \infty$
- Captures intuition that larger mistakes should get larger penalties

How to find optimal Parameters?

$$J(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\sigma(\mathbf{w}^T\mathbf{x}_i)) + (1-y_i)\log(1-\sigma(\mathbf{w}^T\mathbf{x}_i))$$

Just like before, simply take

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = 0$$

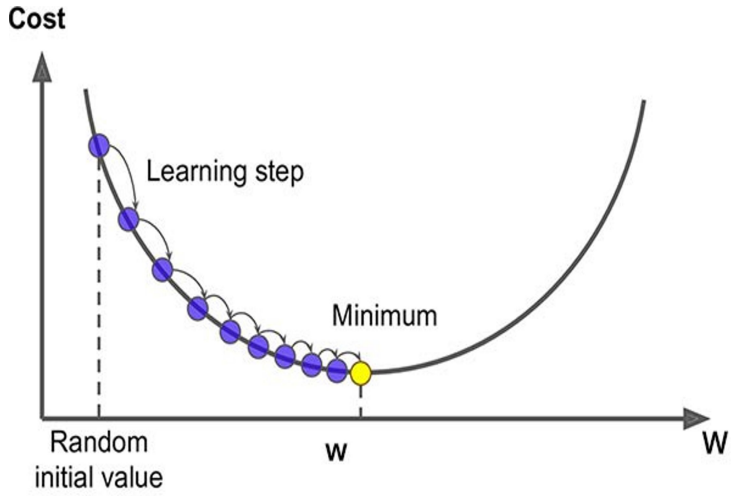
However, this does not have a nice closed solution.

Gradient Descent

$$J(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\sigma(\mathbf{w}^T\mathbf{x}_i)) + (1-y_i)\log(1-\sigma(\mathbf{w}^T\mathbf{x}_i))$$

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \eta\nabla_{\mathbf{w}} J(\mathbf{w}^k)$$

Learning rate



- We have a linear model for prediction
- For classification, we want to output a probability
- We map the prediction to probabilities with a sigmoid function
- We have a loss function (BCE) to compare models

Gradient Descent Algorithm

Gradient Descent

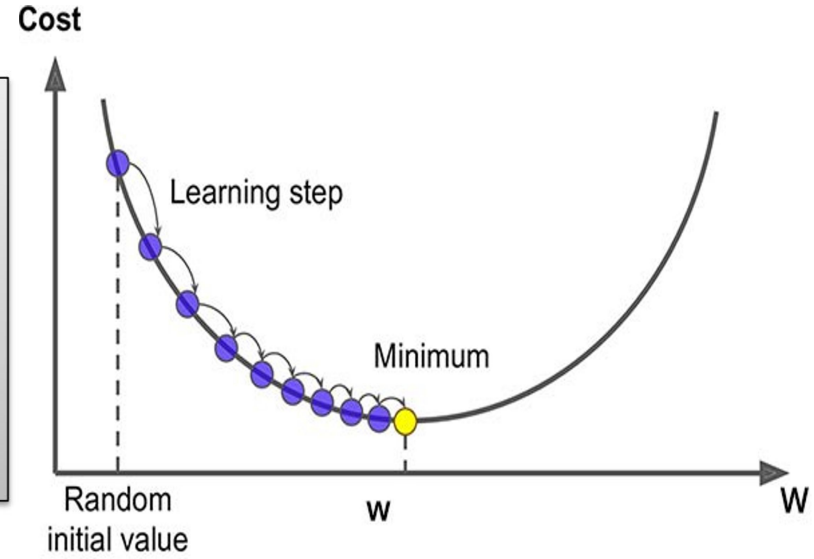
Want $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

Learning rate

simultaneous update
for j = 0 ... d



Direction of maximum increase and decrease for a function

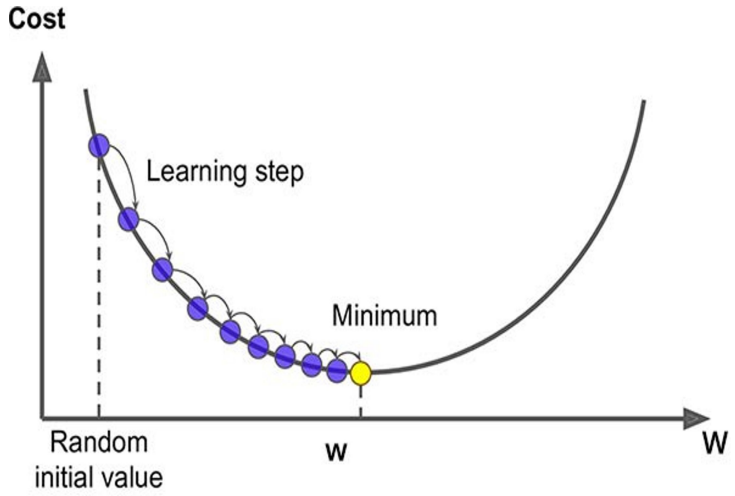
- Gradient direction is the direction of maximum increase for a function
- Negative gradient is the direction of maximum decrease for a function

Gradient Descent

$$J(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\sigma(\mathbf{w}^T\mathbf{x}_i)) + (1-y_i)\log(1-\sigma(\mathbf{w}^T\mathbf{x}_i))$$

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \eta\nabla_{\mathbf{w}} J(\mathbf{w}^k)$$

Learning rate



- We have a linear model for prediction
- For classification, we want to output a probability
- We map the prediction to probabilities with a sigmoid function
- We have a loss function (BCE) to compare models

Gradient Descent

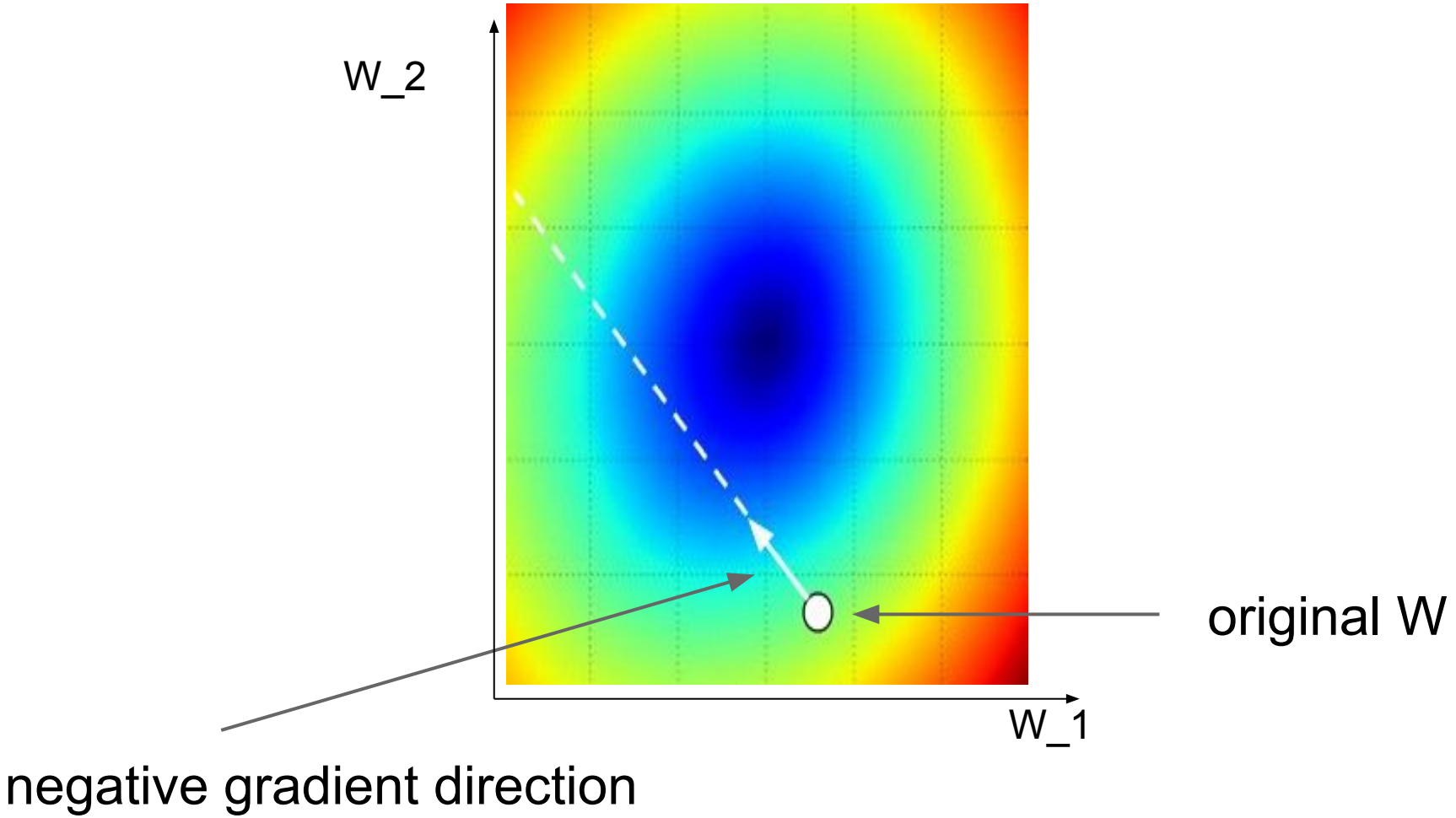
- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

simultaneous update
for j = 0 ... d

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```


W_2
W_1
original W
negative gradient direction

Mini-batch (Stochastic) Gradient Descent

- only use a small portion of the training set to compute the gradient.

```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Common mini-batch sizes are 32/64/128 examples
e.g. Krizhevsky ILSVRC ConvNet used 256 examples

Mini-batch Gradient Descent

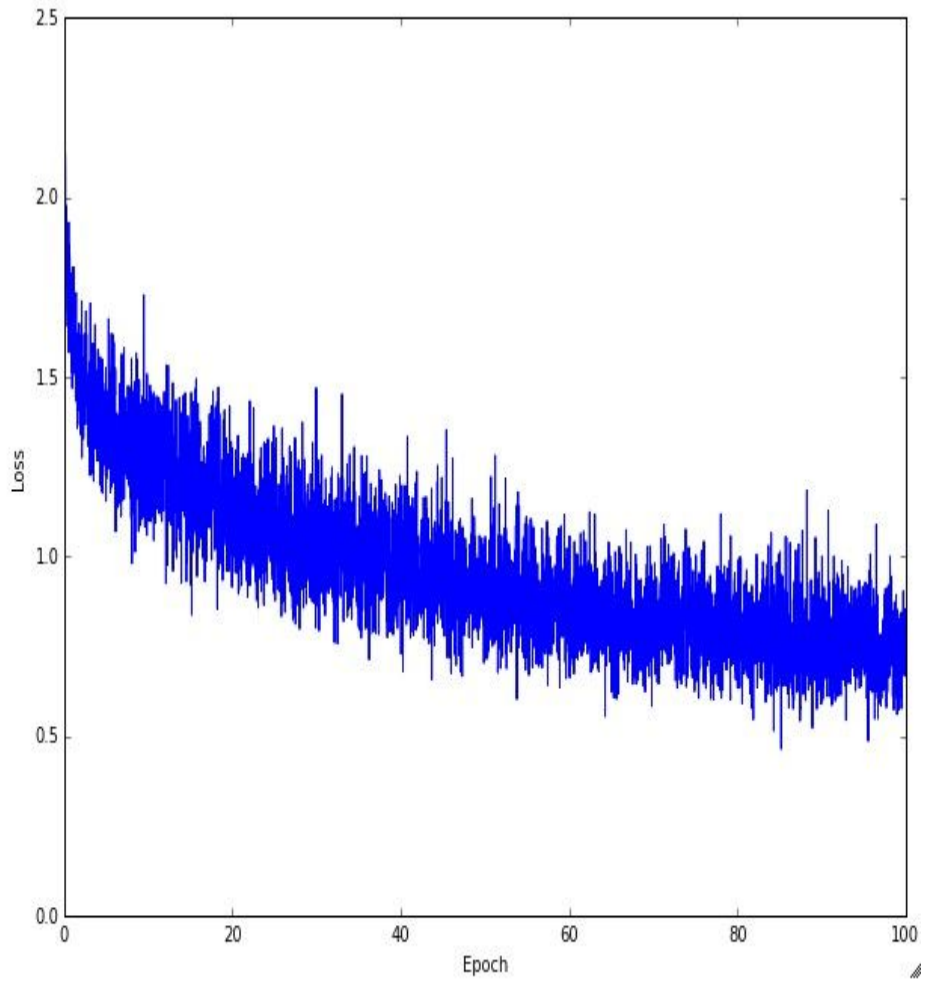
- only use a small portion of the training set to compute the gradient.

```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

we will look at more fancy update formulas (momentum, Adagrad, RMSProp, Adam, …)

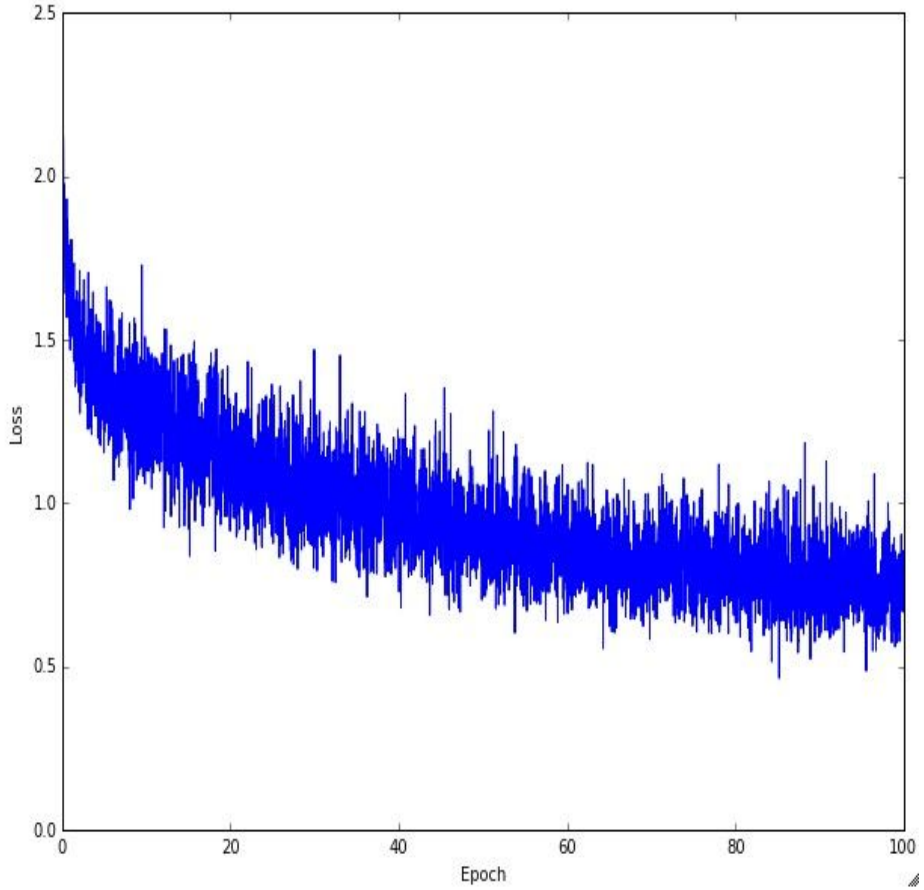
Common mini-batch sizes are 32/64/128 examples
e.g. Krizhevsky ILSVRC ConvNet used 256 examples

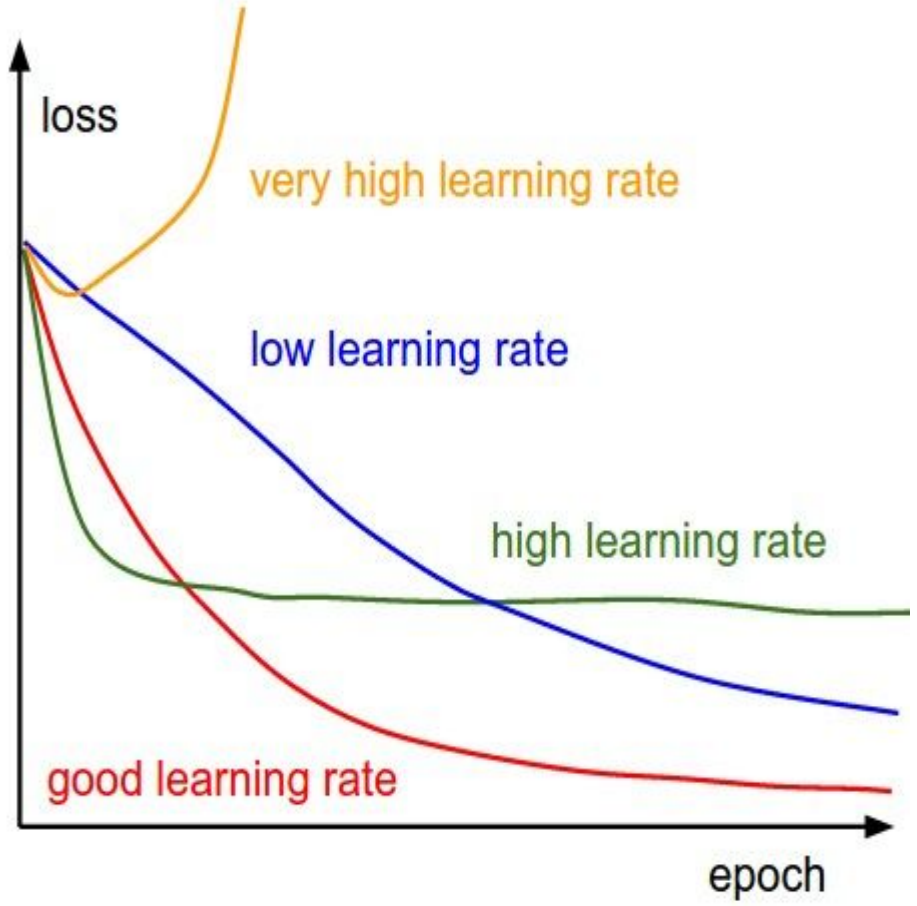


Example of optimization progress while training a neural network.

(Loss over mini-batches goes down over time.)

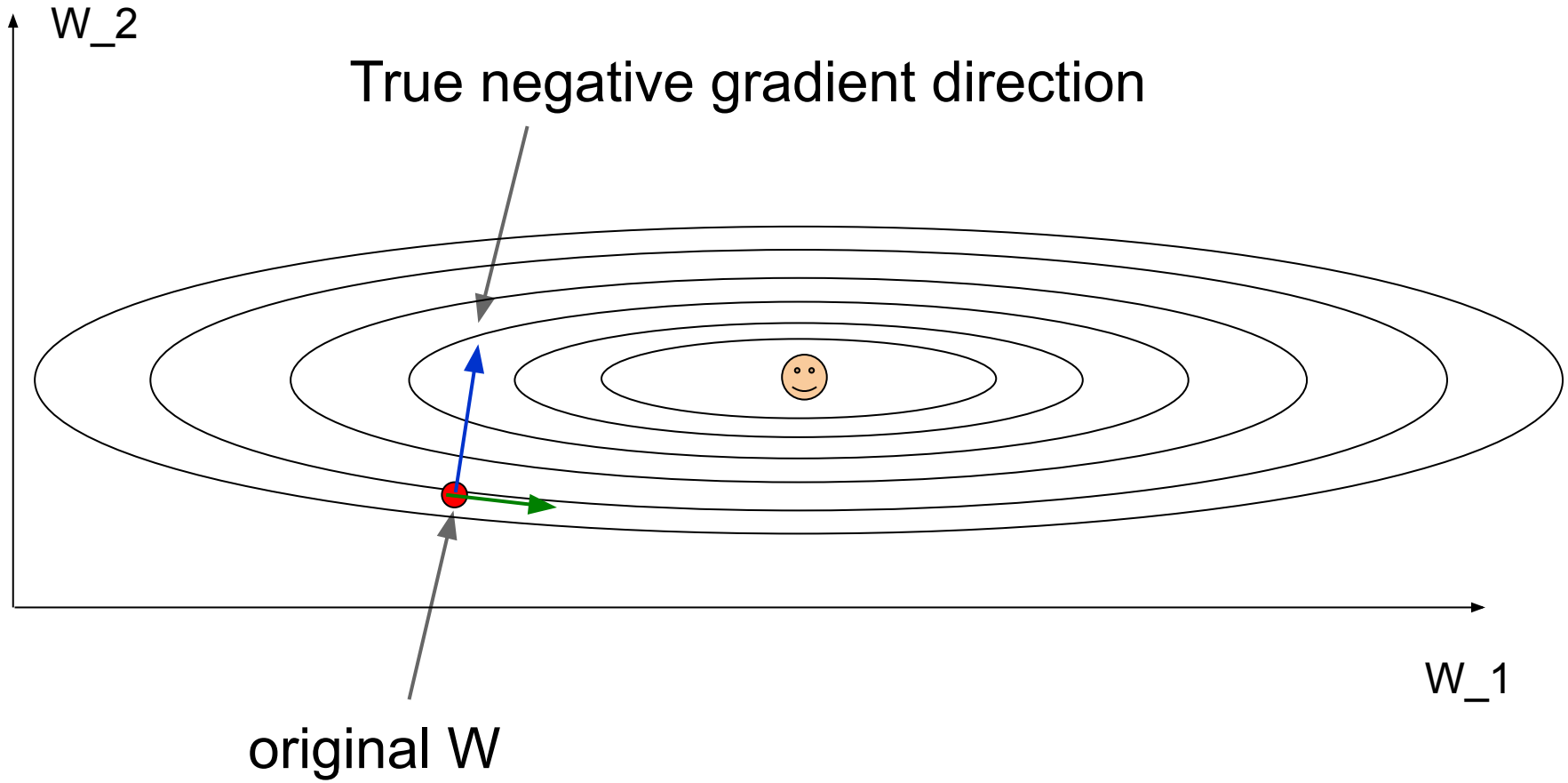
The effects of step size (or “learning rate”)





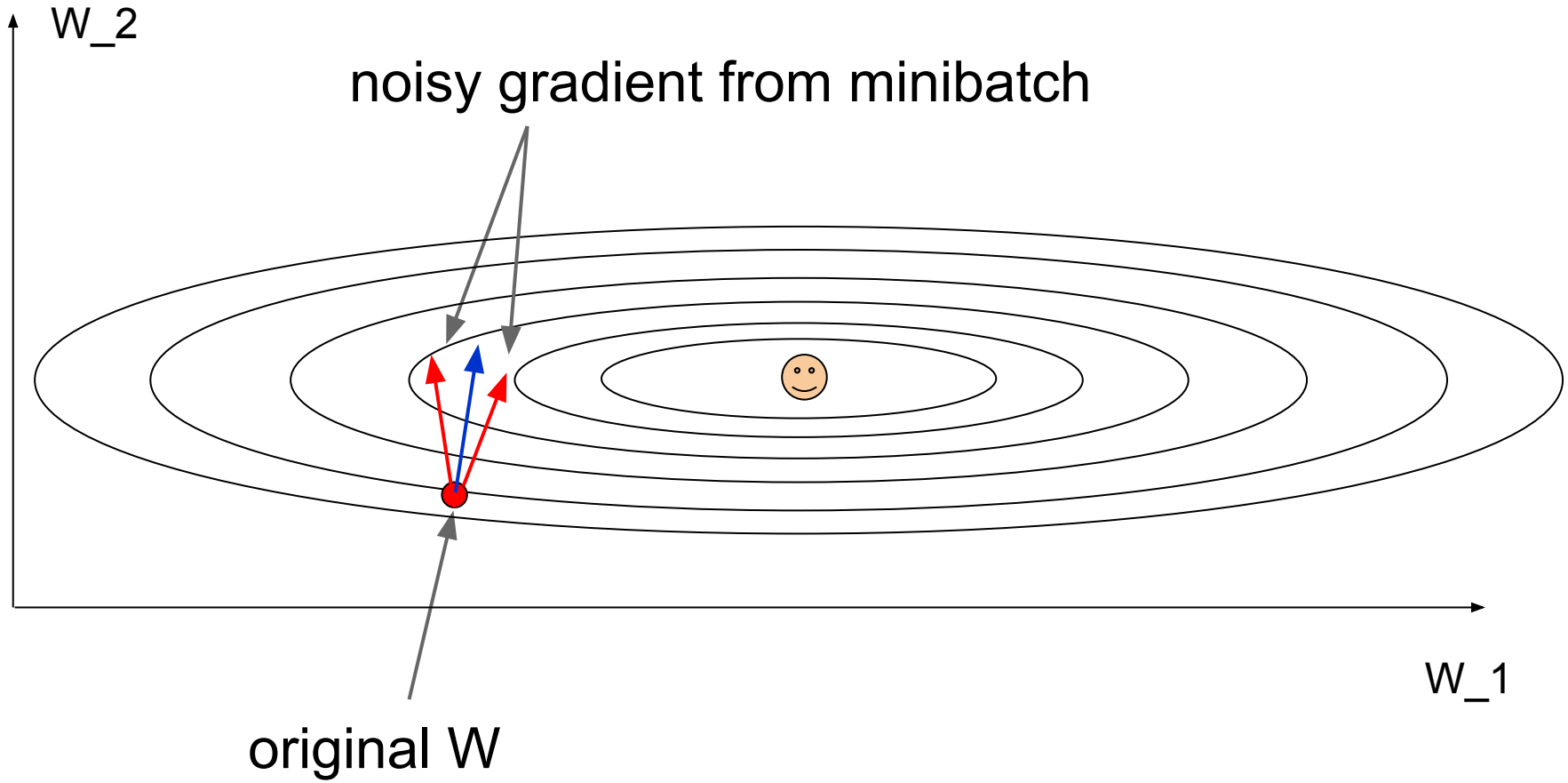
Slides based on cs231n by Fei-Fei Li & Andrej Karpathy & Justin Johnson

Minibatch updates



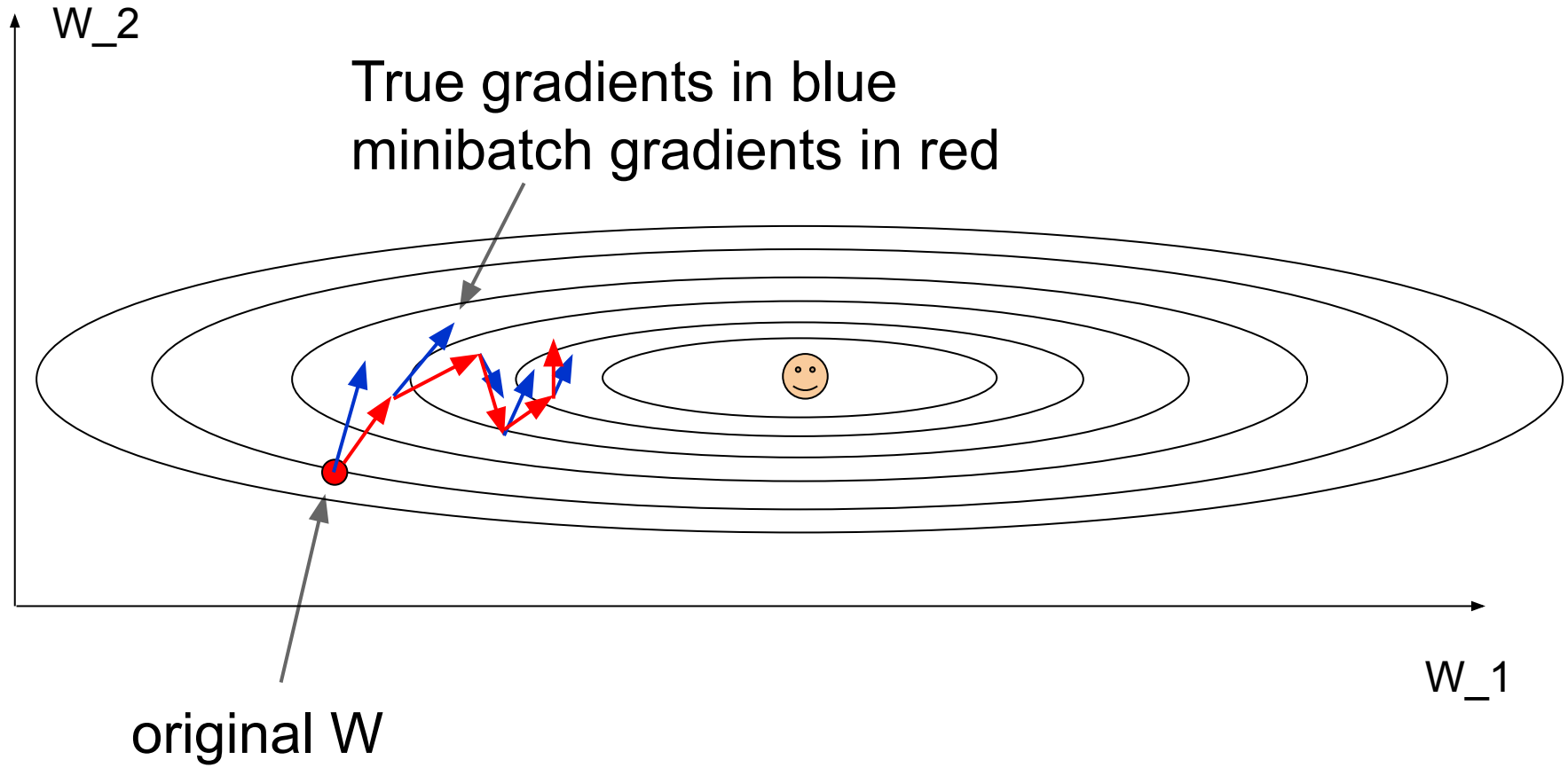
Slide based on CS294-129 by John Canny

Stochastic Gradient



Slide based on CS294-129 by John Canny

Stochastic Gradient Descent



Gradients are noisy but still make good progress on average

Slide based on CS294-129 by John Canny