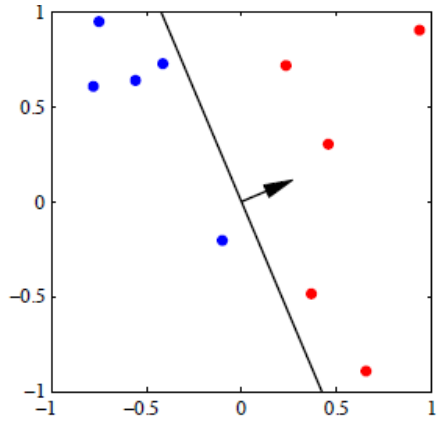
Deep Learning

Winter 2026

Gradient Descent

Recall Learning Tasks



1. Classification (Linear Discriminants)

- We have a data matrix X of shape (n, d)
- Each of the n rows is a d-dimensional vector
- Assume there's a true function f() that outputs a **discrete** label for each X_i
- Our goal is to find h() that approximates f()

$$\mathbf{X} = \{\mathbf{x_1}, \ldots, \mathbf{x_n}\}$$

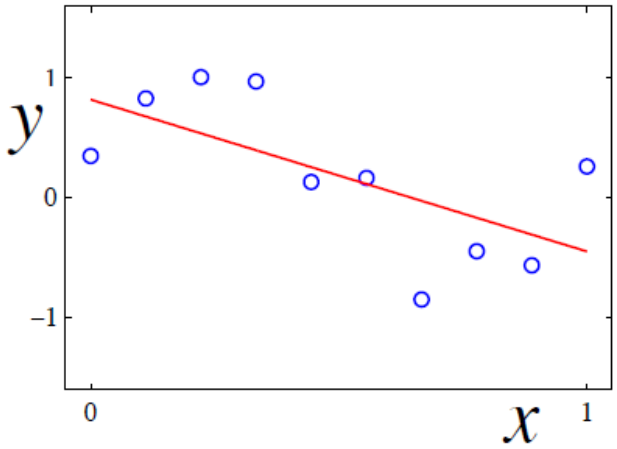
$$\mathbf{x_i} = \langle x_{i,1}, \ldots, x_{i,d} \rangle$$

$$\mathbf{y} = \{y_1, \ldots, y_n\}$$

$$y_i = f(\mathbf{x_i})$$

$$\mathcal{D} = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_n}, y_n)\}$$

Recall Learning Tasks (*Cont.*)



2. Regression (Ordinary Least Squares)

- We have a data matrix X of shape (n, d)
- Each of the n rows is a d-dimensional vector
- Assume there's a true function f() that outputs a **real-valued** label for each X_i
- Our goal is to find h() that approximates f()

$$\mathbf{X} = \{\mathbf{x_1}, \ldots, \mathbf{x_n}\}$$

$$\mathbf{x_i} = \langle x_{i,1}, \ldots, x_{i,d} \rangle$$

$$\mathbf{y} = \{y_1, \ldots, y_n\}$$

$$y_i = f(\mathbf{x_i})$$

$$\mathcal{D} = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_n}, y_n)\}$$

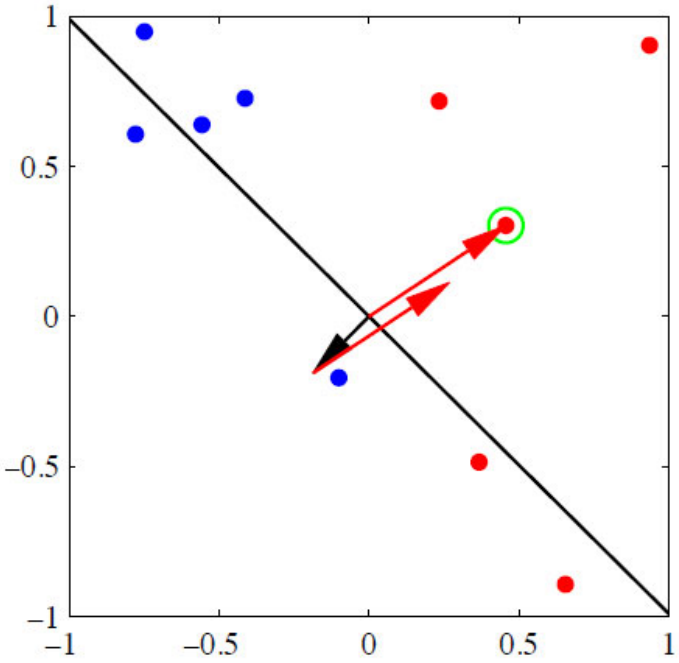
Recall Supervised Machine Learning in One Slide

1. Pick data **D,** model (function) $\mathbf{M}(\mathbf{w})$ and objective function $\mathbf{J}(\mathbf{D}, \mathbf{w})$
2. Initialize model weights (parameters) $\mathbf{w}$ somehow
3. Measure model performance with the objective function $\mathbf{J}(\mathbf{D}, \mathbf{w})$

HOW?

4. Modify parameters $\mathbf{w}$ somehow, hoping to improve $\mathbf{J}(\mathbf{D}, \mathbf{w})$
5. Repeat 3 and 4 until you stop improving or run out of time

Improving Parameters



The weight update algorithm

$\mathbf{w} =$ *some random setting*

Do

$k = (k+1) \bmod (m)$

if $h(\mathbf{x}_k) ! = y_k$

$\mathbf{w} = \mathbf{w} + \mathbf{x}y$

Until $\forall k,\ h(\mathbf{x}_k) = y_k$

$$\begin{aligned}
\mathrm{RSS}(\mathbf{w}) &= (\mathbf{y} - \mathbf{X}\mathbf{w})^\intercal (\mathbf{y} - \mathbf{X}\mathbf{w}) \\
&= (\mathbf{y} - \mathbf{X}\mathbf{w})^2 \\
\frac{\partial \mathrm{RSS}}{\partial \mathbf{w}} &= -2\mathbf{X}^\intercal (\mathbf{y} - \mathbf{X}\mathbf{w}) \\
0 &= \mathbf{X}^\intercal (\mathbf{y} - \mathbf{X}\mathbf{w}) \\
0 &= \mathbf{X}^\intercal \mathbf{y} - \mathbf{X}^\intercal \mathbf{X}\mathbf{w} \\
\mathbf{X}^\intercal \mathbf{X}\mathbf{w} &= \mathbf{X}^\intercal \mathbf{y} \\
(\mathbf{X}^\intercal \mathbf{X})^{-1} \mathbf{X}^\intercal \mathbf{X}\mathbf{w} &= (\mathbf{X}^\intercal \mathbf{X})^{-1} \mathbf{X}^\intercal \mathbf{y} \\
\mathbf{w} &= (\mathbf{X}^\intercal \mathbf{X})^{-1} \mathbf{X}^\intercal \mathbf{y}
\end{aligned}$$

Tradeoffs in Optimization

Perceptron algorithm

- Can't find a good but imperfect solution
- Extremely fast per iteration
- Supports streaming by design

Closed-form linear regression

- Solution is optimal, regardless of final loss value
- Requires slow matrix inversion
- Can't run in streaming settings

More General Approach to Optimization

HOW?

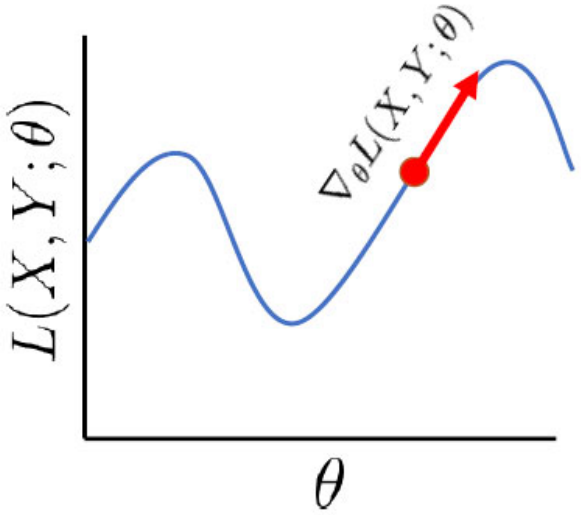
1. Measure how the the loss changes when we change the parameters θ slightly

2. Pick the next set of parameters to be close to the current set, but in the direction that most changes the loss function for the better

3. Repeat

What Does the Gradient Tell Us?

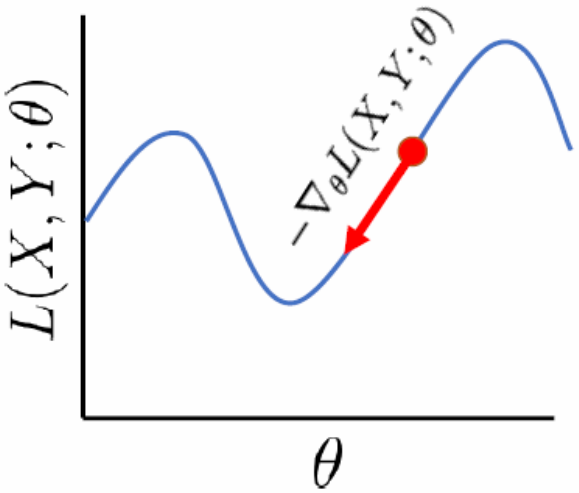
- If the loss function and hypothesis function encoded by the model are differentiable* (i.e., the gradient exists)
- We can evaluate the gradient for some fixed value of our model parameters θ and get the *direction* in which the loss *increases* fastest



*or subdifferentiable

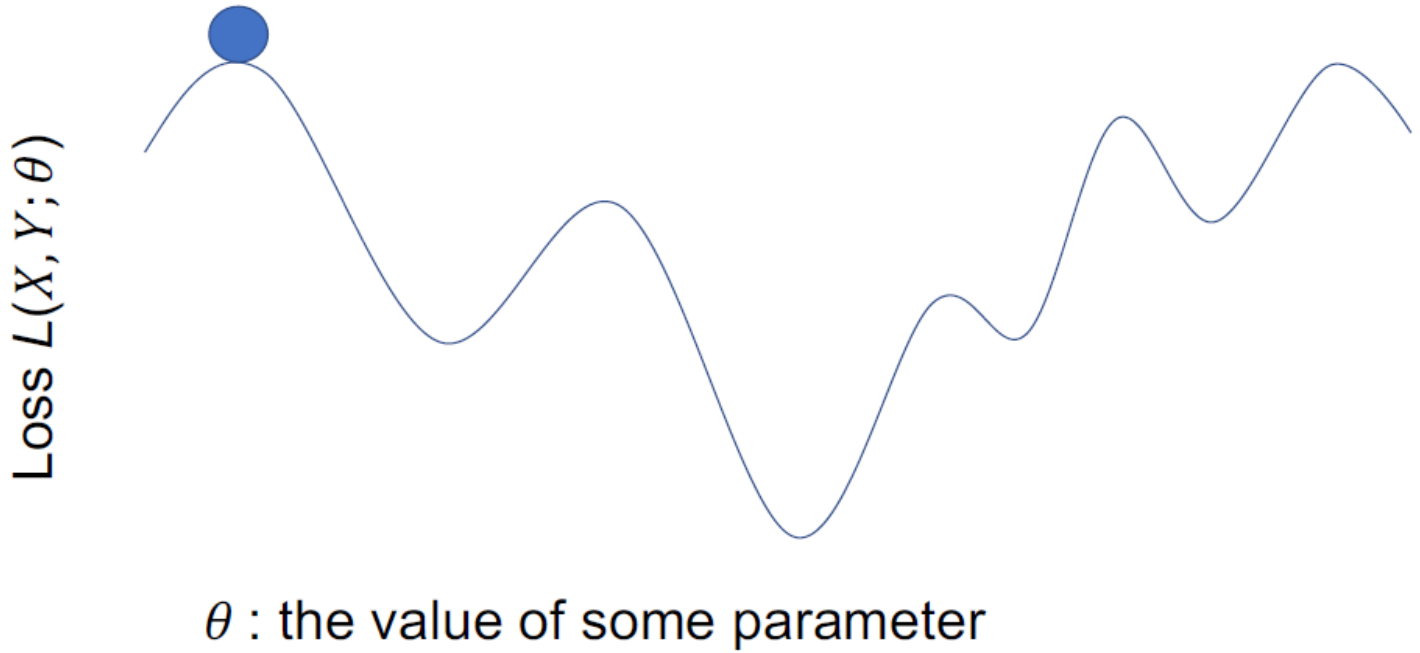
What Does the Gradient Tell Us?

- We want to *decrease* our loss, so let's go the other way instead



Gradient Descent: Promises and Caveats

- Much faster than guessing new parameters randomly
- Finds the global optimum only if the objective function is convex



Gradient Descent Pseudocode

Initialize $\theta^{(0)}$

Repeat until stopping condition met:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla L(X, Y; \theta^{(t)})$$

Return $\theta^{(t_{max})}$

$\theta^{(t)}$ are the parameters of the model at time step t

X, Y are the input data vectors and the output values.

$\nabla L(X, Y; \theta^{(t)})$ is the gradient of the loss function with respect to model parameters $\theta^{(t)}$

η controls the step size

$\theta^{(t_{max})}$ is the set of parameters that did best on the loss function.

23

Gradient Descent Design Choices

Initialize $\theta^{(0)}$

Repeat until stopping condition met:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla L(X, Y, \theta^{(t)})$$

Return $\theta^{(t_{max})}$

- Initialization of θ
- Convergence criterion (i.e., when to stop)
- How much data to use per batch
- Step size for updating model parameters
- Choosing a loss function

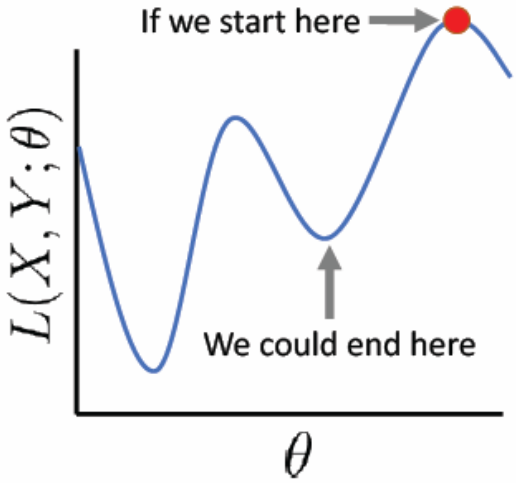
Parameter Initialization

Common initializations:

- $\theta^{(0)} = 0$
- $\theta^{(0)} =$ random values

What happens if our initialization is bad?

- Convergence to a *local* minimum
- No way to determine if you've converged to the global minimum



Convergence Criteria: When to Stop

- Stop when the gradient is close (within ε) to 0
  (i.e., we reached a minimum)

- Stop after some fixed number of iterations

- Stop when the loss on a *validation set* stops decreasing
  (This helps prevent overfitting)

Batch Size: How Much Data?

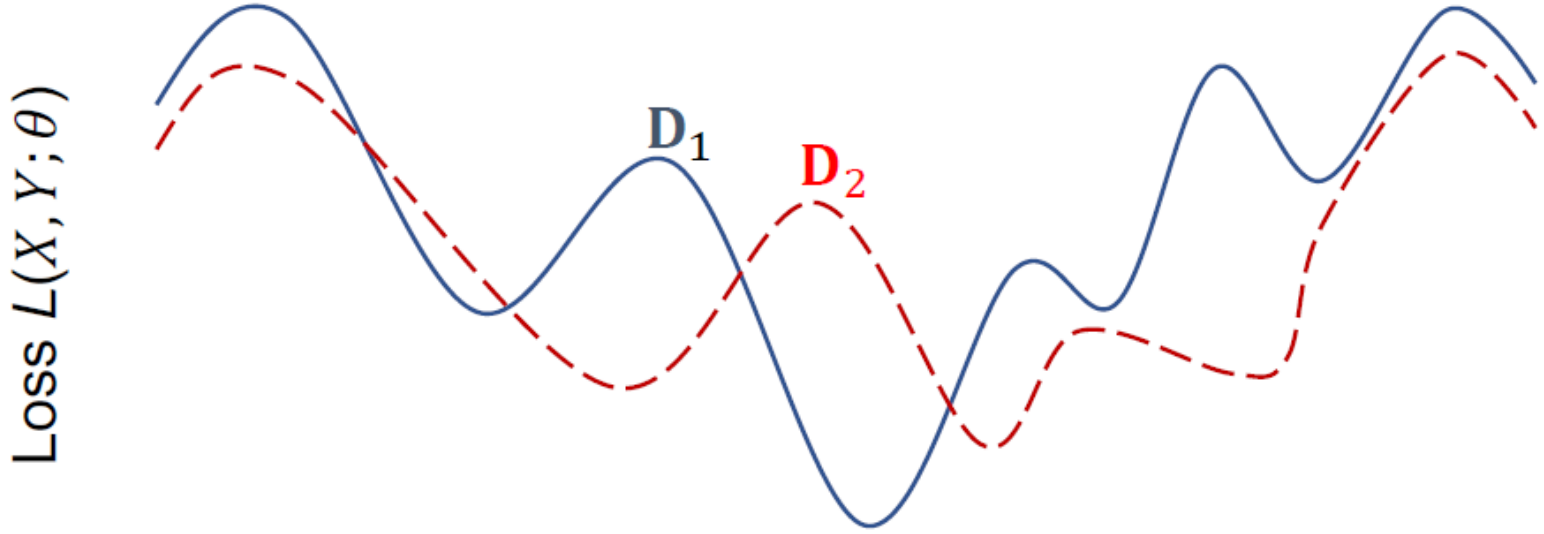
- In **batch gradient descent**, the loss is a function of both the parameters θ and the set of all training data **D.**
- In **stochastic gradient descent**, loss is a function of the parameters and a different single random training sample at each iteration.
- In **mini-batch gradient descent**, random subsets of the data (e.g., 100 examples) are used at each step in the iteration.

How Much Data to Use in Each Step?

- All of it (*batch gradient descent*)
  - The *most accurate* representation of your training loss
  - It can be slow or impossible for large datasets
- Just one data point (*stochastic gradient descent*)
  - A *noisy, inaccurate* representation of your training loss
  - Very fast, but can be inconsistent: random shuffling is important
- More than one data point, but less than all (*mini-batch)*
  - Most common approach: balances *speed* and *accuracy*
  - Usually want batch size to be as large as possible for your machine

Different Data, Different Loss

- If the data D changes….then the landscape of the loss function changes
- You typically won't know how it has changed.

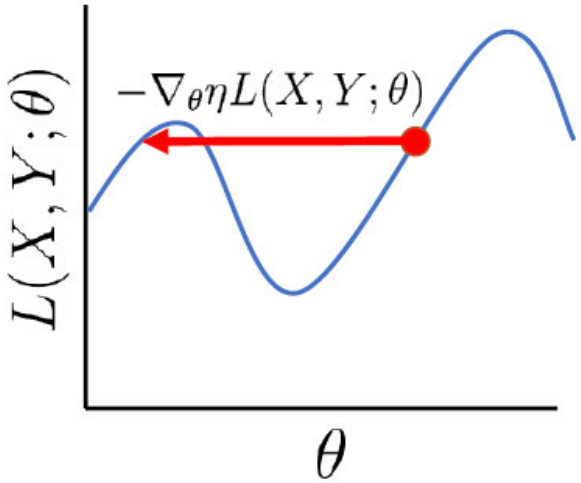


θ : the value of some parameter

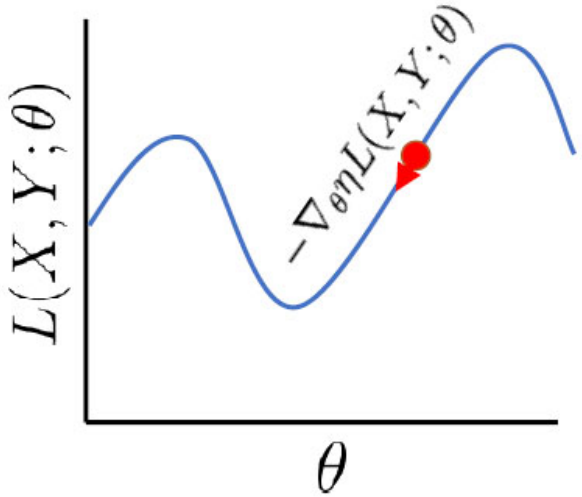
Step Size: How Far Should We Go?

Step Size: how far should we go?

- The gradient we calculated was based on a fixed value of θ
- As we move away from this point, the gradient changes



If the step size is too large, we may overshoot the minimum



If the step size is too small, we need to take more steps (more computation)

35

Add Momentum

Initialize $\theta^{(0)}, V^{(0)}$

Repeat until stopping condition met:

$$V^{(t+1)} = mV^{(t)} - \eta \nabla L(X, Y, \theta^{(t)})$$

$$\theta^{(t+1)} = \theta^{(t)} + V^{(t+1)}$$

Return $\theta^{(t_{max})}$

Selected Cost Functions

Perceptron (0-1 Loss):

$$PERCEPTRON(X,Y) = \begin{cases} 1 & if\ h(x) \cdot y > 0 \\ -1 & otherwise \end{cases}$$

Multi-Class Cross Entropy:

$$BCE(\hat{p}, p) = \sum_{n=1}^{N} - p_n \ln(\hat{p}_n) - (1 - p_n) \ln(1 - \hat{p}_n)$$

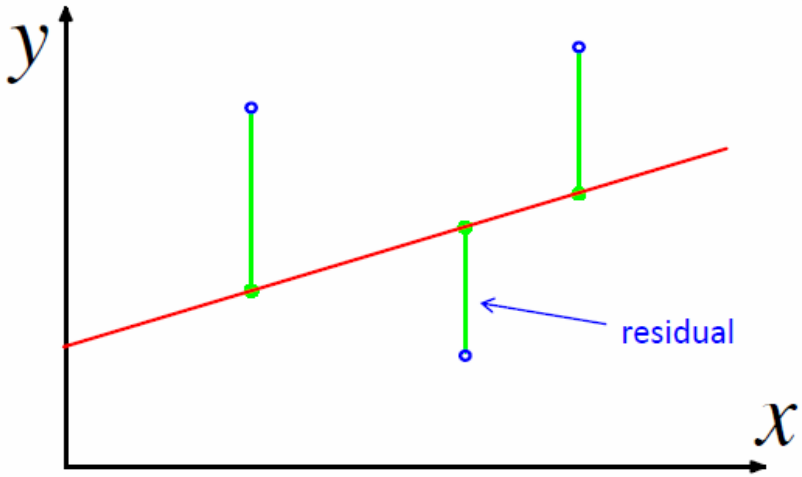
Mean Squared Error:

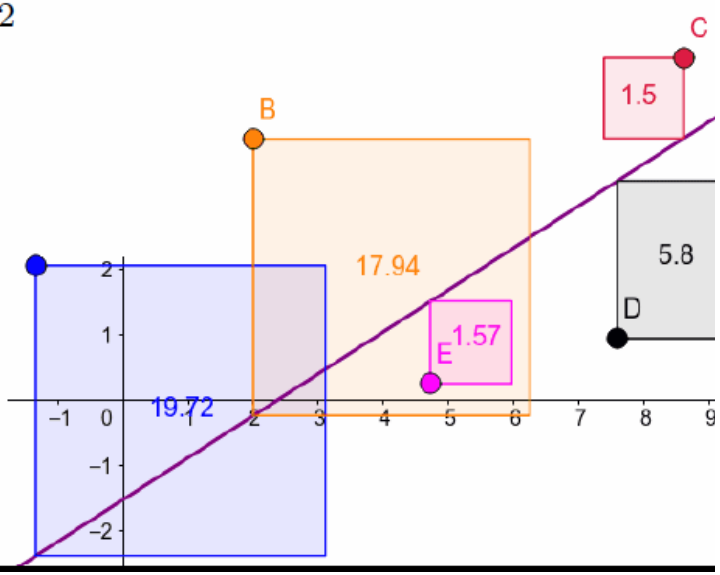
$$MSE(\hat{y}, y) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Objective Function: Mean Squared Error (MSE)

- Minimize the mean squared error (MSE)
  - Also called sum of squared residuals, sum of squared errors

$$\text{MSE} = \sum_{i=1}^{N} (y_i - h(\mathbf{x_i}))^2$$



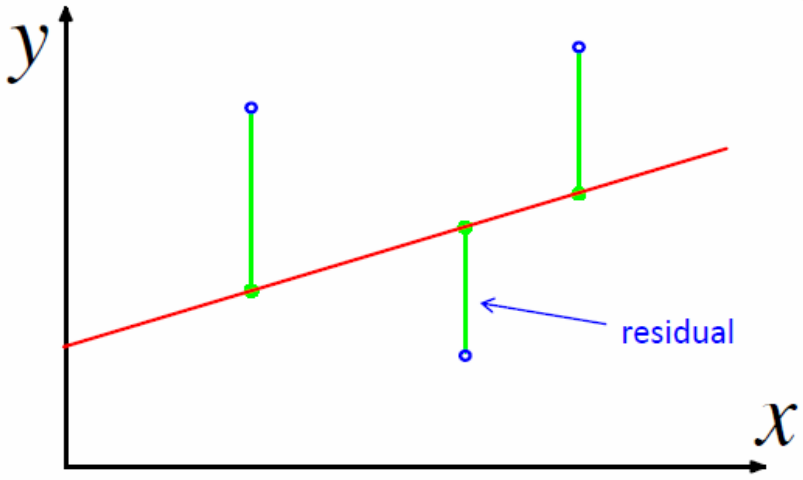


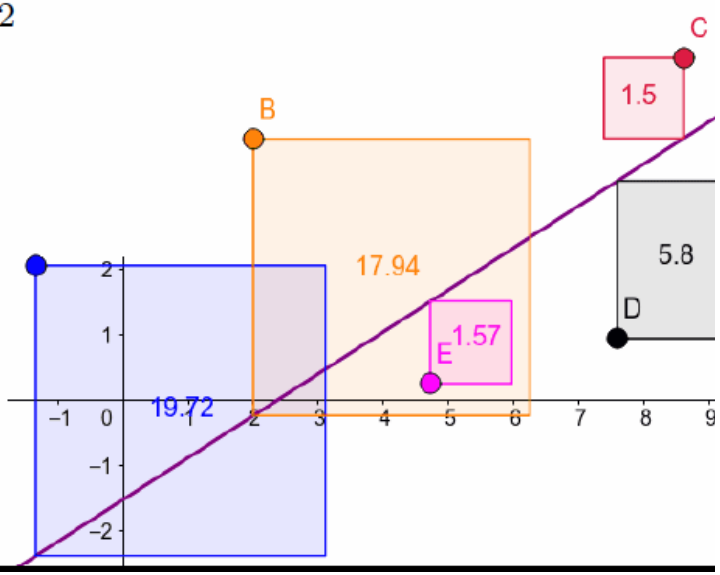
$$\hat{y} = h(x_i) = w_0 + w_1 x_i$$

Objective Function: Mean Squared Error (MSE)

- Minimize the mean squared error (MSE)
  - Also called sum of squared residuals, sum of squared errors

$$\text{MSE} = \sum_{i=1}^{N} (y_i - h(\mathbf{x_i}))^2$$

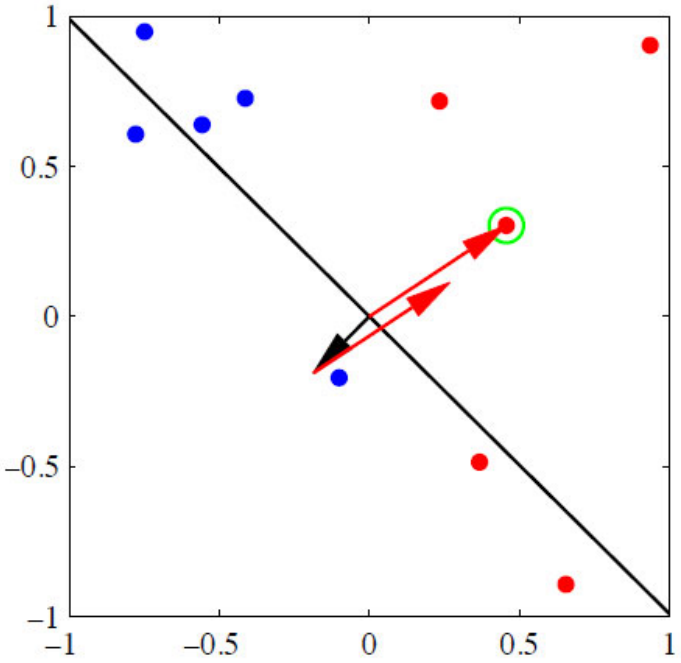




$$\hat{y} = h(x_i) = w_0 + w_1 x_i$$

$$\frac{\partial L}{\partial w_0} = 2(\hat{y}_i - h(x_i)) \qquad \frac{\partial L}{\partial w_1} = 2(\hat{y}_i - h(x_i))x_i$$

Improving Parameters



$$PERCEPTRON(X,Y) = \begin{cases} 1 & if\ \ h(x)\cdot y > 0 \\ -1 & otherwise \end{cases}$$

The weight update algorithm

$\mathbf{w} =$ *some random setting*

Do

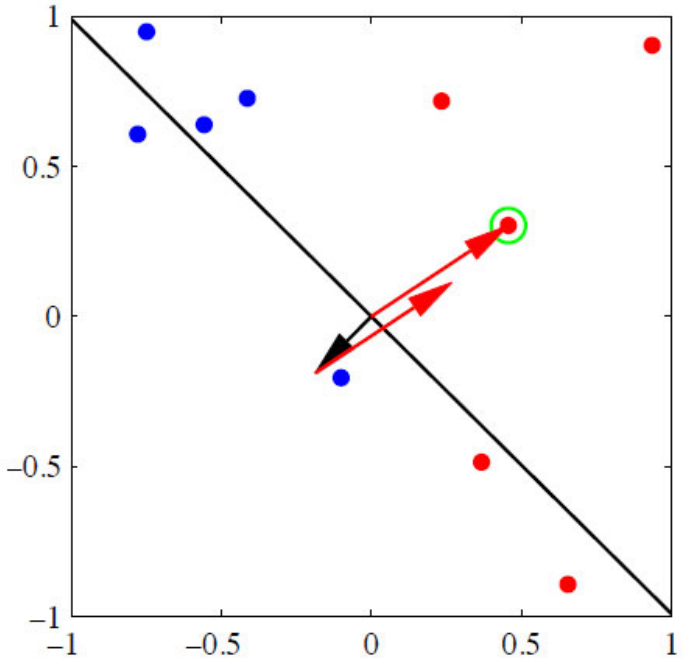
$k = (k+1) \bmod (m)$

if $h(\mathbf{x}_k) != y_k$

$\mathbf{w} = \mathbf{w} + \mathbf{x}y$

Until $\forall k,\ h(\mathbf{x}_k) = y_k$

Improving Parameters



$$PERCEPTRON(X,Y) = \begin{cases} 1 & if\ h(x)\cdot y > 0 \\ -1 & otherwise \end{cases}$$

$$BCE(\hat{p}, p) = \sum_{n=1}^{N} -p_n \ln(\hat{p}_n) - (1-p_n)\ln(1-\hat{p}_n)$$

The weight update algorithm

$\mathbf{w} =$ *some random setting*

Do

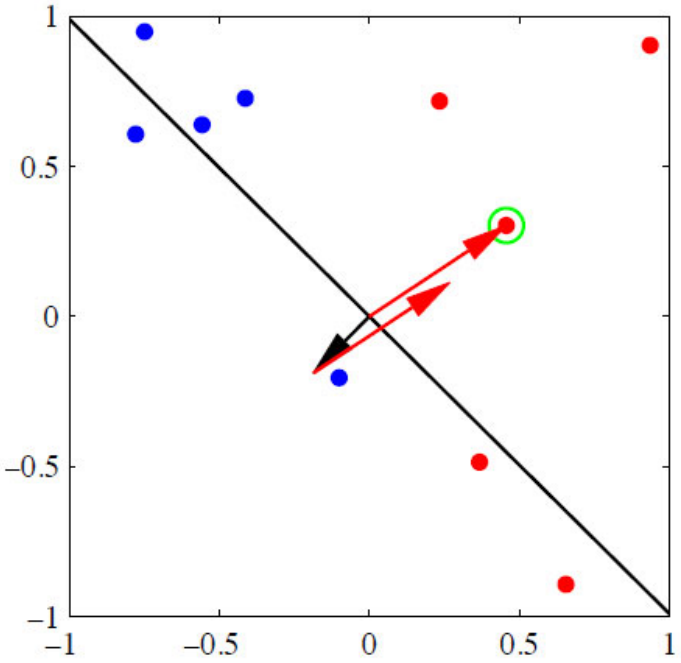
$k = (k+1) \bmod (m)$

if $h(\mathbf{x}_k) != y_k$

$\mathbf{w} = \mathbf{w} + \mathbf{x}y$

Until $\forall k,\ h(\mathbf{x}_k) = y_k$

Improving Parameters



$$PERCEPTRON(X,Y) = \begin{cases} 1 & if\ \ h(x)\cdot y > 0 \\ -1 & otherwise \end{cases}$$

$$BCE(\hat{p}, p) = \sum_{n=1}^{N} - p_n \ln(\hat{p}_n) - (1 - p_n)\ln(1 - \hat{p}_n)$$

$$\hat{p}_n = \frac{e^{\hat{y}_n}}{\sum_{c \in C} e^{\hat{y}_c}} = \frac{e^{\hat{y}_n}}{e^{\hat{y}_1} + e^{\hat{y}_2}}$$

The weight update algorithm

$\mathbf{w} =$ *some random setting*

Do

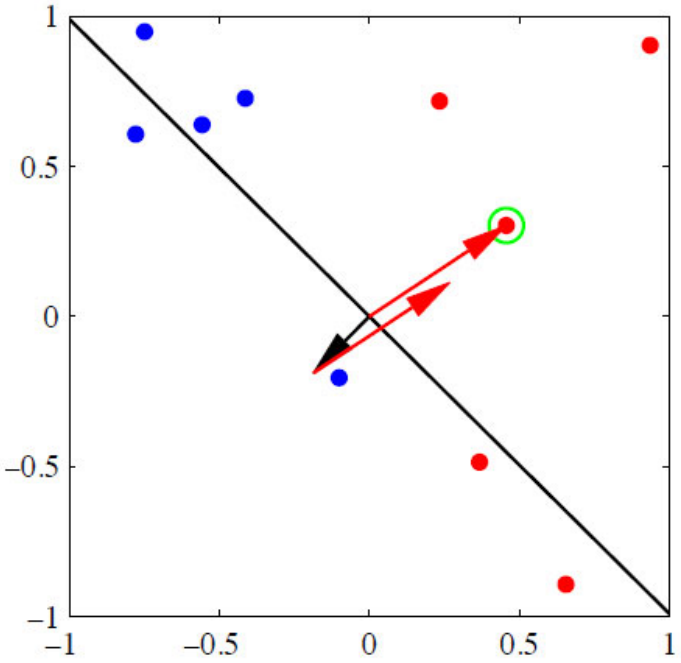
$k = (k+1)\mathrm{mod}(m)$

if $h(\mathbf{x}_k)! = y_k$

$\mathbf{w} = \mathbf{w} + \mathbf{x}y$

Until $\forall k,\ h(\mathbf{x}_k) = y_k$

Improving Parameters



The weight update algorithm

$\mathbf{w} =$ *some random setting*

Do

$k = (k+1) \bmod (m)$

if $h(\mathbf{x}_k) ! = y_k$

$\mathbf{w} = \mathbf{w} + \mathbf{x}y$

Until $\forall k,\ h(\mathbf{x}_k) = y_k$

$$PERCEPTRON(X,Y) = \begin{cases} 1 & if\ h(x) \cdot y > 0 \\ -1 & otherwise \end{cases}$$

$$BCE(\hat{p}, p) = \sum_{n=1}^{N} - p_n \ln(\hat{p}_n) - (1 - p_n)\ln(1 - \hat{p}_n)$$

$$\hat{p}_n = \frac{e^{\hat{y}_n}}{\sum_{c \in C} e^{\hat{y}_c}} = \frac{e^{\hat{y}_n}}{e^{\hat{y}_1} + e^{\hat{y}_2}}$$

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2$$