

Gradient Descent

- how are we going to find the solution for

$$\arg \min_{b,w} \sum_{i=1}^n \ell(b + w^T x_i, y_i)$$

- e.g., Logistic Regression do not have closed form solution for

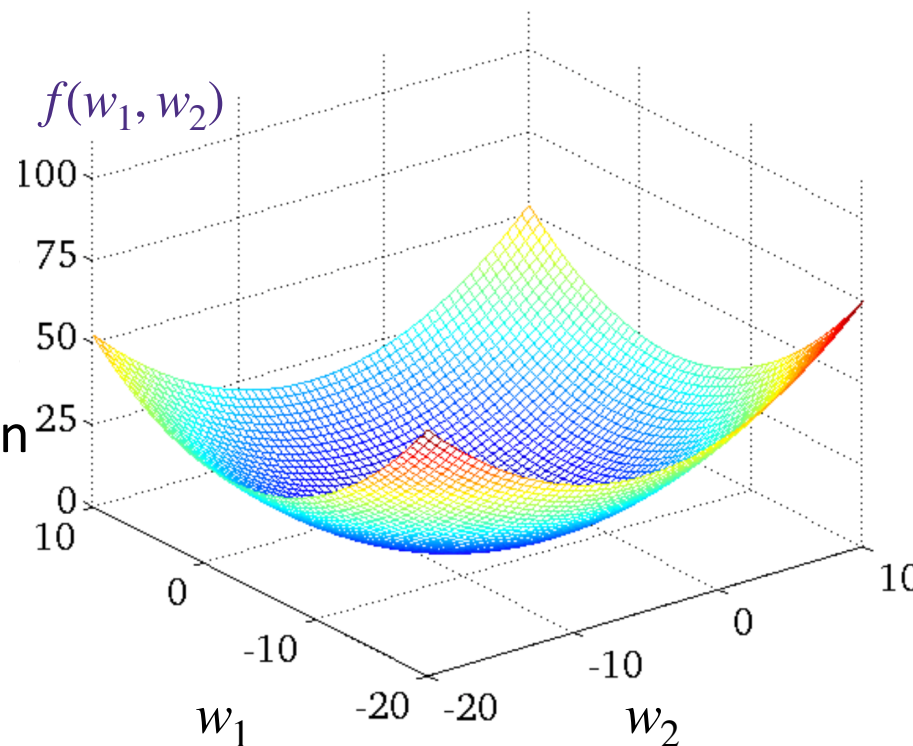
$$\nabla_{b,w} \mathcal{L}(b, w) = 0$$

Running example: linear regression

- **Given data:** $\{(x_i, y_i)\}_{i=1}^n$ $x_i \in \mathbb{R}^d$ $y_i \in \mathbb{R}$
- **Learning model parameters:**

$$\hat{w}_{\text{LS}} = \arg \min_{w \in \mathbb{R}^d} \underbrace{\|\mathbf{y} - \mathbf{X}w\|_2^2}_{f(w)}$$

- Although we know the optimal solution in a closed form, we will use this as a running example to understand GD



1-dimensional gradient descent

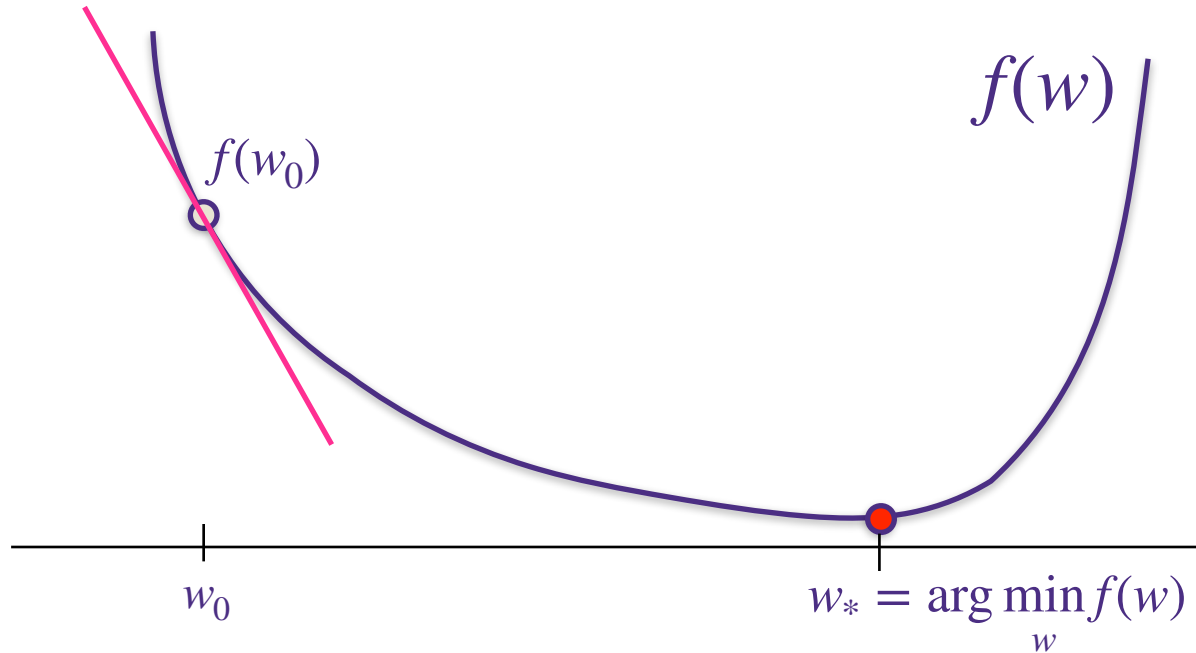
Let w_0 be an initial guess. How can we improve this solution?

Taylor series approximation:

For w very close to w_0 we have

$$f(w_0) + (w - w_0) \left. \frac{df(w)}{dw} \right|_{w=w_0}$$

is very close to $f(w)$



1-dimensional gradient descent

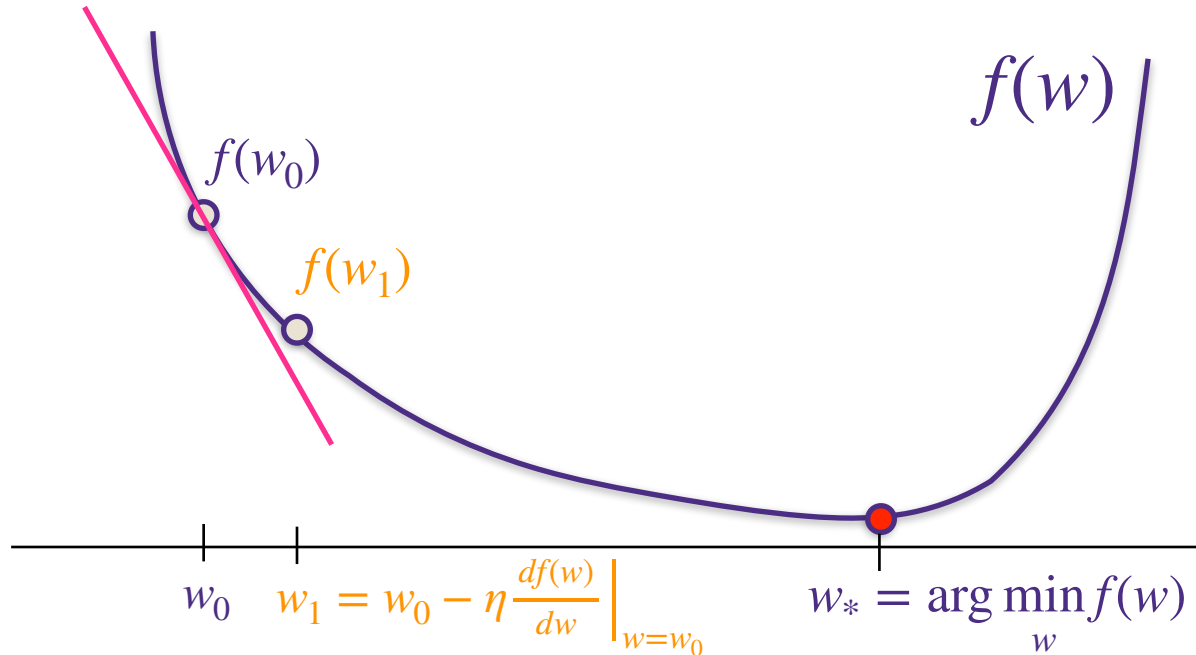
Let w_0 be an initial guess. How can we improve this solution?

Taylor series approximation:

For w very close to w_0 we have

$$f(w_0) + (w - w_0) \frac{df(w)}{dw} \Big|_{w=w_0}$$

is very close to $f(w)$



Thus, for very small $\eta > 0$,

if $w_1 = w_0 - \eta \frac{df(w)}{dw} \Big|_{w=w_0}$ then

$$f(w_0) - \eta \left(\frac{df(w)}{dw} \Big|_{w=w_0} \right)^2$$

is very close to $f(w_1) < f(w_0)$

1-dimensional gradient descent

Let w_0 be an initial guess. How can we improve this solution?

Taylor series approximation:

For w very close to w_0 we have

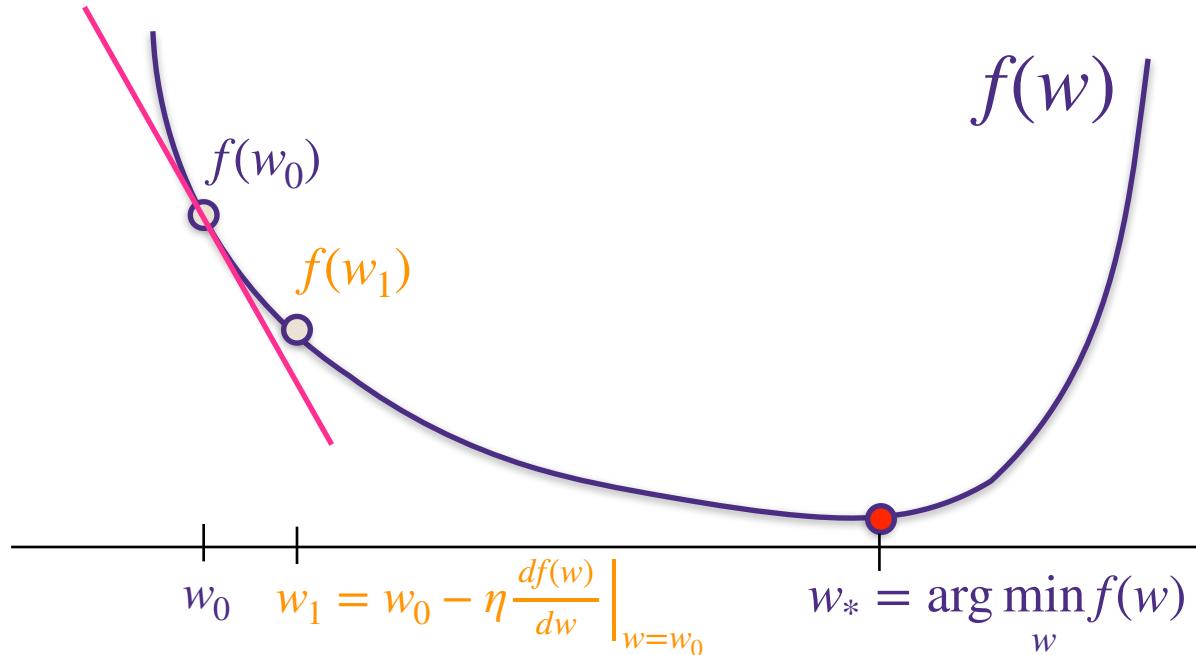
$$f(w_0) + (w - w_0) \frac{df(w)}{dw} \Big|_{w=w_0}$$

is very close to $f(w)$

Thus, for very small $\eta > 0$,

if $w_1 = w_0 - \eta \frac{df(w)}{dw} \Big|_{w=w_0}$ then

$f(w_0) - \eta \left(\frac{df(w)}{dw} \Big|_{w=w_0} \right)^2$
is very close to $f(w_1) < f(w_0)$



Gradient descent

For $k=0,1,2,3,\dots$

$$w_{k+1} = w_k - \eta \frac{df(w)}{dw} \Big|_{w=w_k}$$

1-dimensional gradient descent

Let w_0 be an initial guess. How can we improve this solution?

Taylor series approximation:

For w very close to w_0 we have

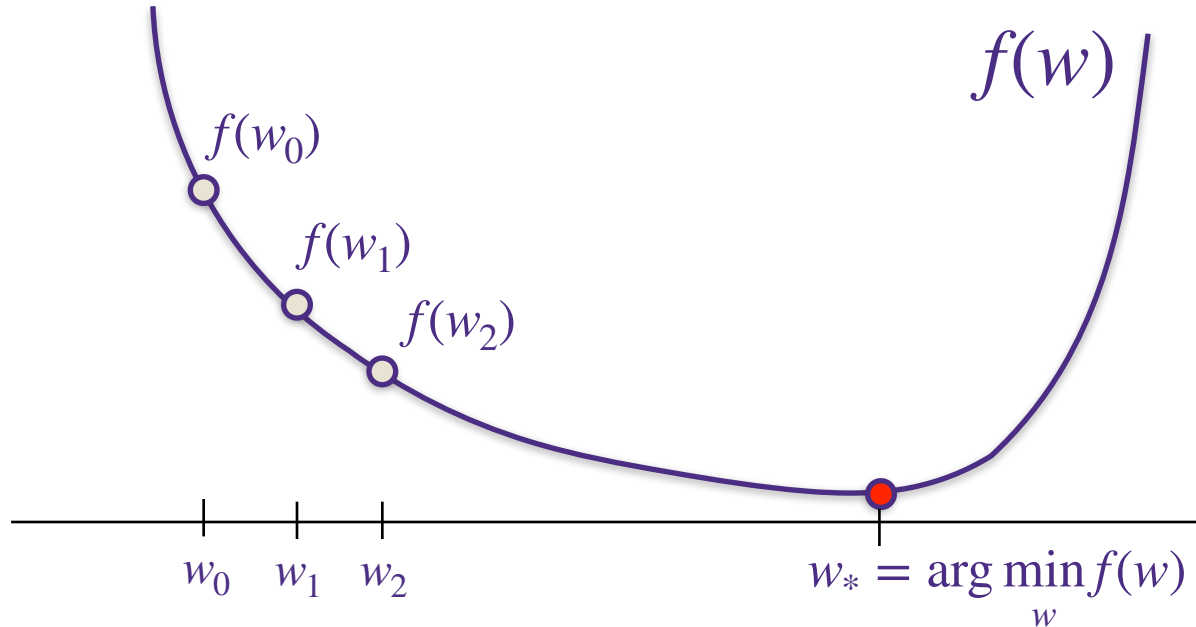
$$f(w_0) + (w - w_0) \frac{df(w)}{dw} \Big|_{w=w_0}$$

is very close to $f(w)$

Thus, for very small $\eta > 0$,

if $w_1 = w_0 - \eta \frac{df(w)}{dw} \Big|_{w=w_0}$ then

$f(w_0) - \eta \left(\frac{df(w)}{dw} \Big|_{w=w_0} \right)^2$
is very close to $f(w_1) < f(w_0)$



Gradient descent

For $k=0,1,2,3,\dots$

$$w_{k+1} = w_k - \eta \frac{df(w)}{dw} \Big|_{w=w_k}$$

1-dimensional gradient descent

Let w_0 be an initial guess. How can we improve this solution?

Taylor series approximation:

For w very close to w_0 we have

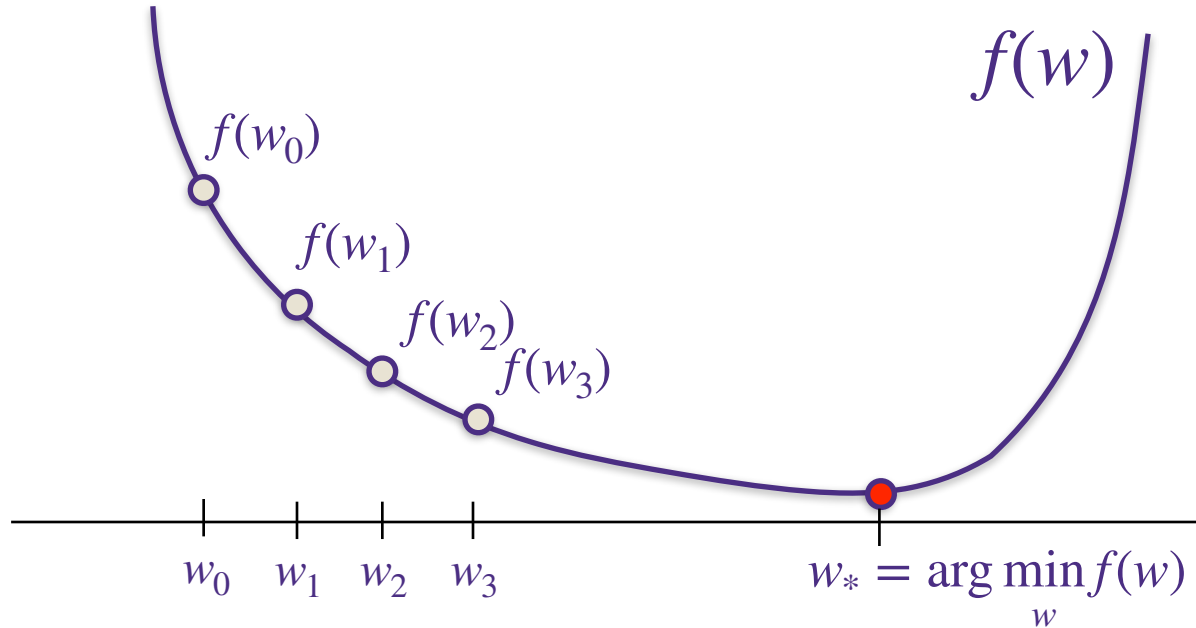
$$f(w_0) + (w - w_0) \frac{df(w)}{dw} \Big|_{w=w_0}$$

is very close to $f(w)$

Thus, for very small $\eta > 0$,

if $w_1 = w_0 - \eta \frac{df(w)}{dw} \Big|_{w=w_0}$ then

$f(w_0) - \eta \left(\frac{df(w)}{dw} \Big|_{w=w_0} \right)^2$
is very close to $f(w_1) < f(w_0)$



Gradient descent

For $k=0,1,2,3,\dots$

$$w_{k+1} = w_k - \eta \frac{df(w)}{dw} \Big|_{w=w_k}$$

1-dimensional gradient descent

Let w_0 be an initial guess. How can we improve this solution?

Taylor series approximation:

For w very close to w_0 we have

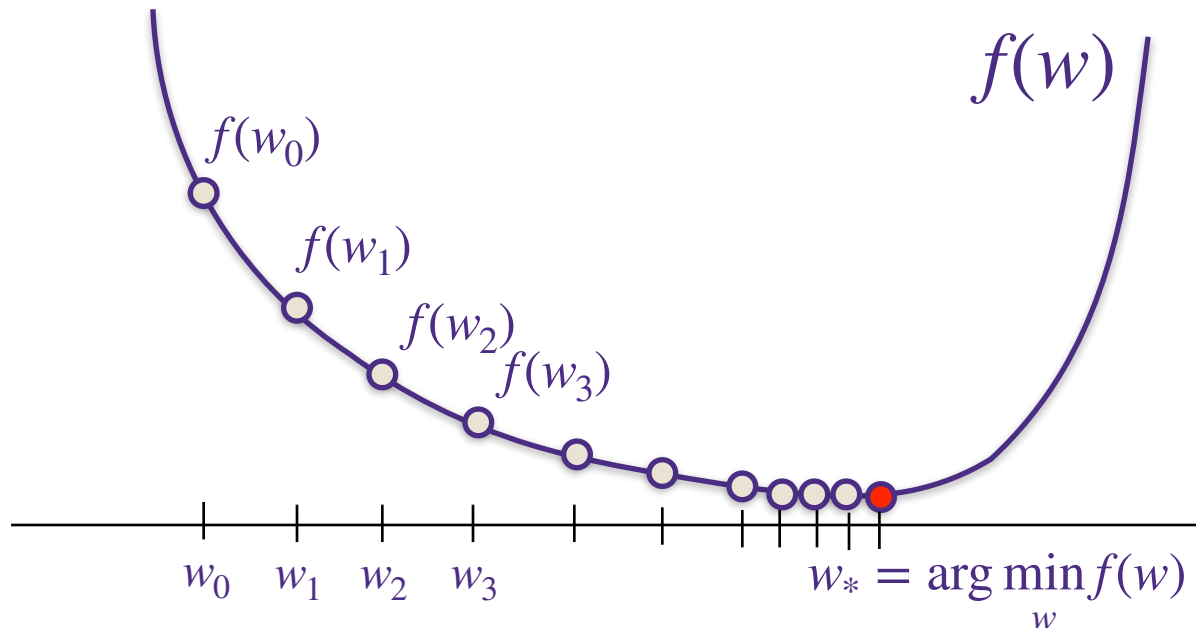
$$f(w_0) + (w - w_0) \frac{df(w)}{dw} \Big|_{w=w_0}$$

is very close to $f(w)$

Thus, for very small $\eta > 0$,

if $w_1 = w_0 - \eta \frac{df(w)}{dw} \Big|_{w=w_0}$ then

$f(w_0) - \eta \left(\frac{df(w)}{dw} \Big|_{w=w_0} \right)^2$
is very close to $f(w_1) < f(w_0)$



Gradient descent

For $k=0,1,2,3,\dots$

$$w_{k+1} = w_k - \eta \frac{df(w)}{dw} \Big|_{w=w_k}$$

Note that as $k \rightarrow \infty$ we have $\frac{df(w)}{dw} \Big|_{w=w_k} \rightarrow 0$

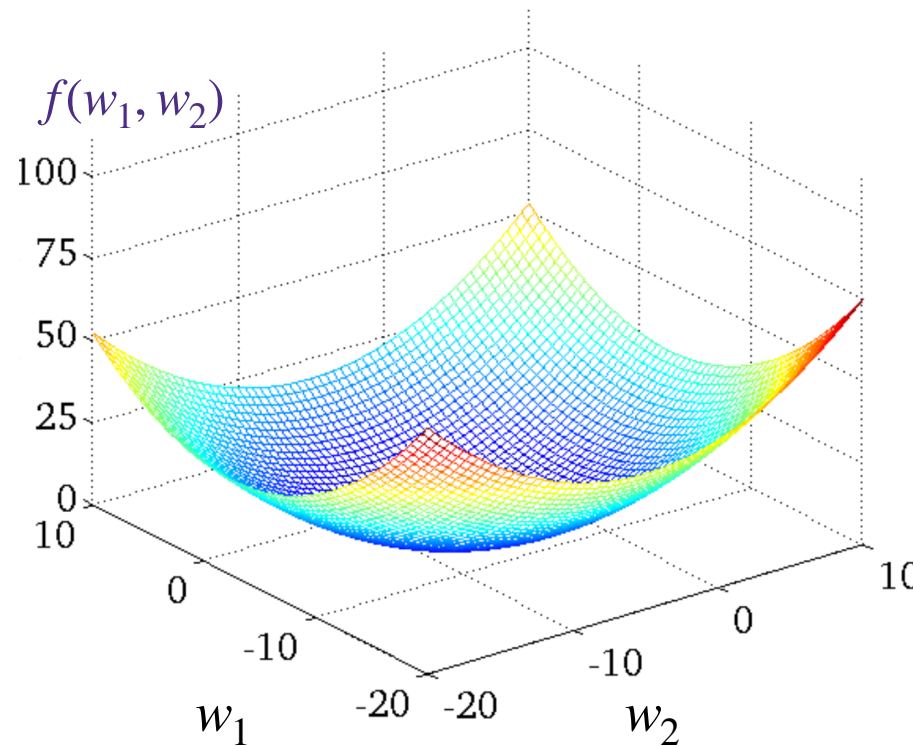
Running example: linear regression

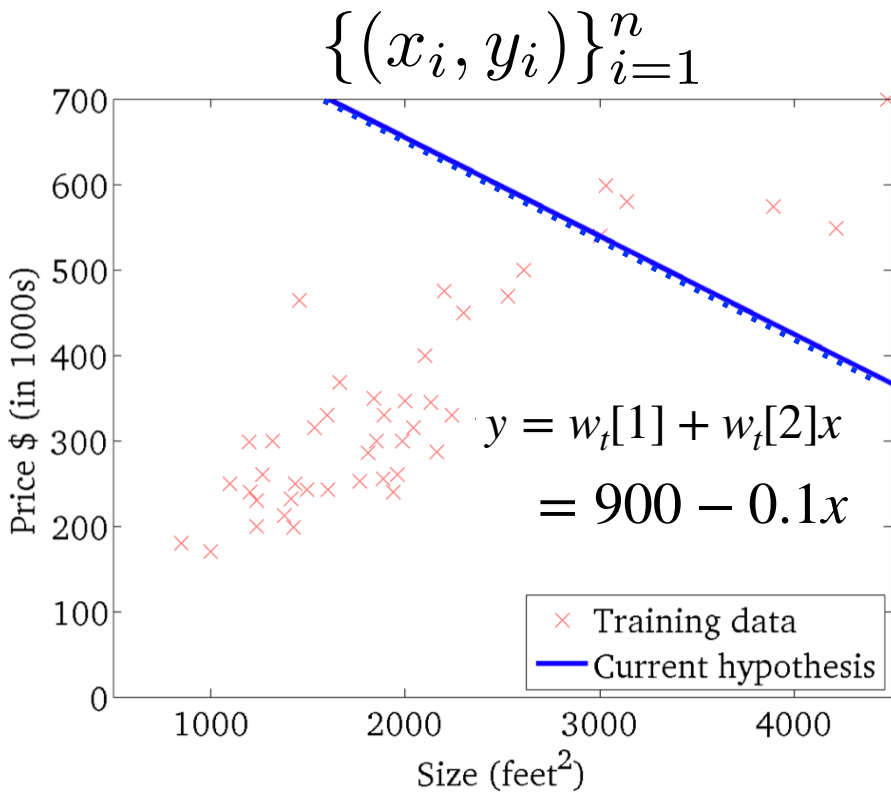
- **Given data:** $\{(x_i, y_i)\}_{i=1}^n$ $x_i \in \mathbb{R}^d$ $y_i \in \mathbb{R}$
- **Learning model parameters:**

$$\hat{w}_{\text{LS}} = \arg \min_{w \in \mathbb{R}^d} \underbrace{\|y - Xw\|_2^2}_{f(w)}$$

- **Gradient descent:**

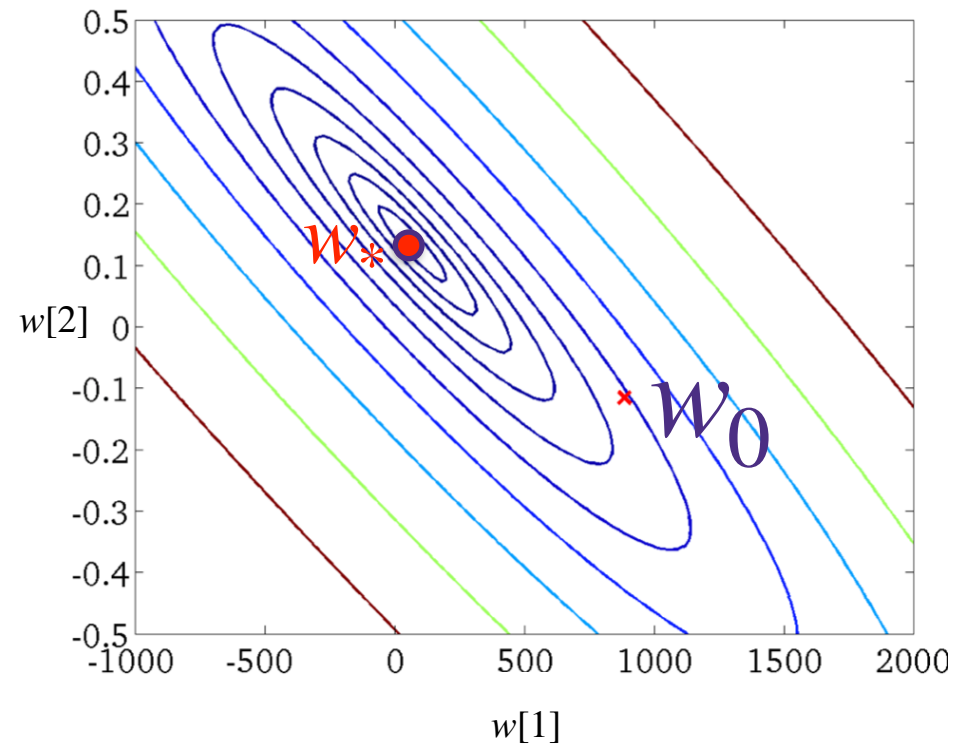
- Initialize: $w_0 = 0$
- **For** $t=0,1,2,\dots$
 - $w_{t+1} \leftarrow w_t - \eta \cdot \nabla_w f(w_t)$





Evolution of the predictor

- $w_0 = (900, -0.1)$
- For $t=0,1,2,\dots$
 - $w_{t+1} \leftarrow w_t - \eta \cdot \nabla_w f(w_t)$



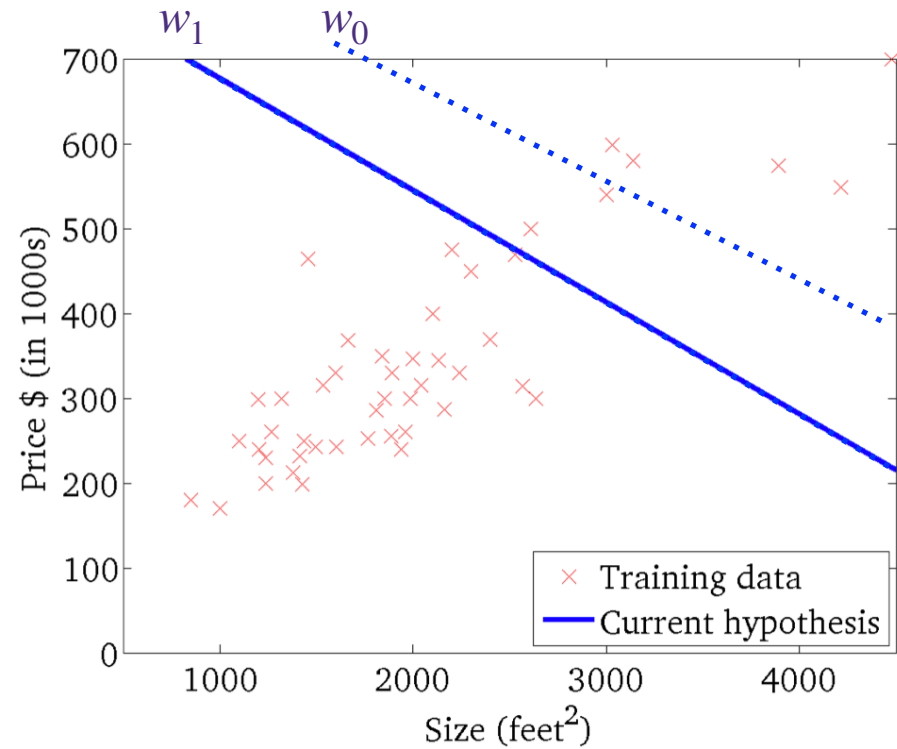
GD dynamics in the Parameter space

- Which direction will the GD move?

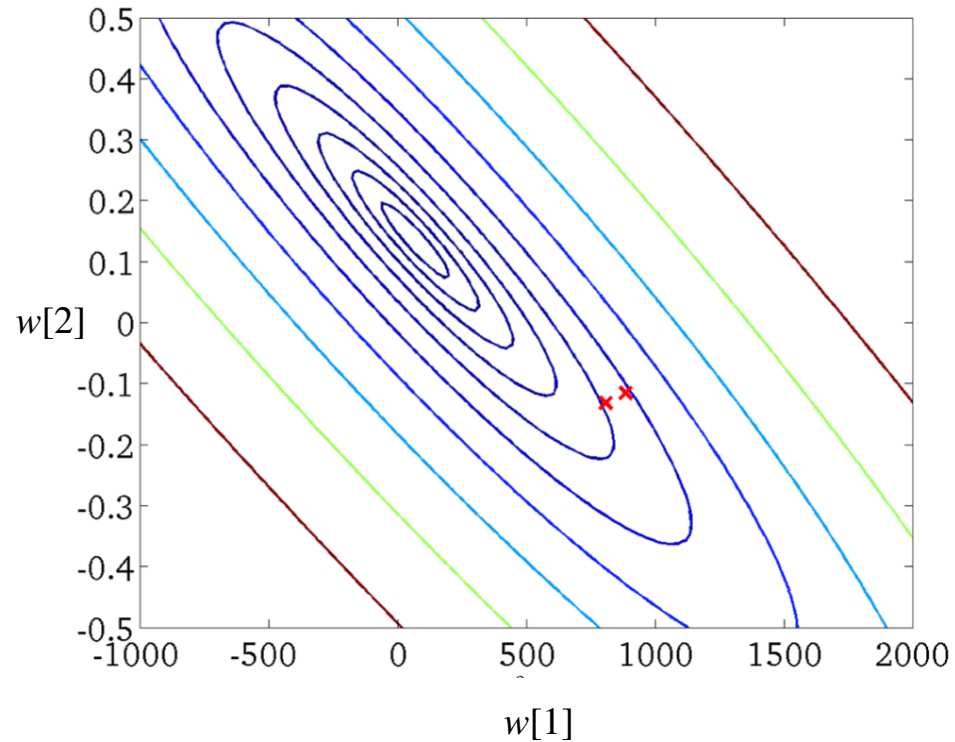
- $w_0 = (900, -0.1)$

- **For $t=0,1,2,\dots$**

- $w_{t+1} \leftarrow w_t - \eta \cdot \nabla_w f(w_t)$



Evolution of the predictor

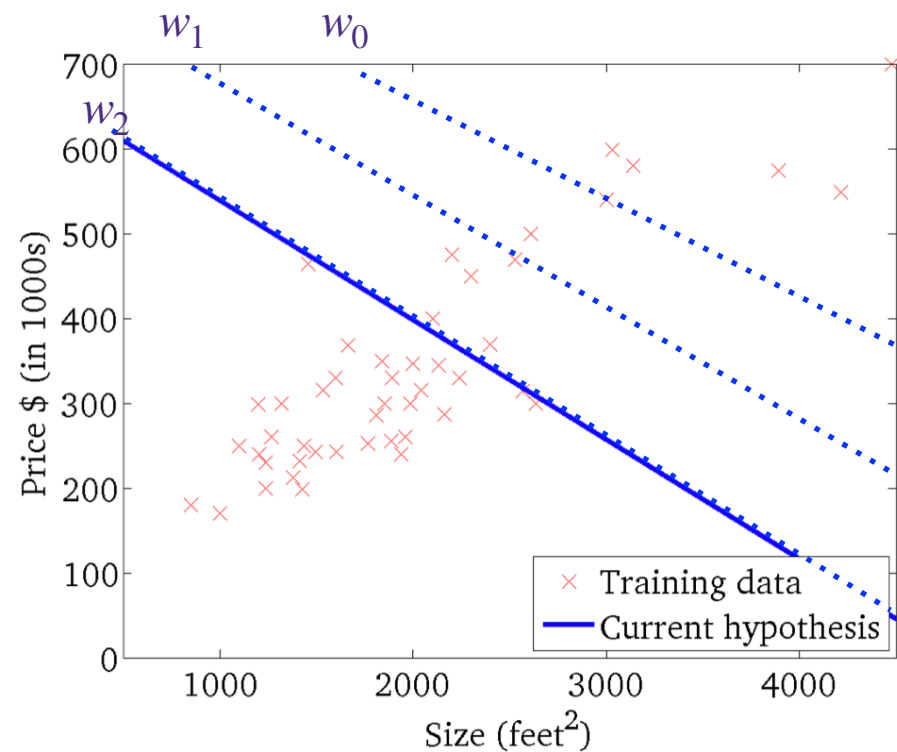


GD dynamics in the Parameter space

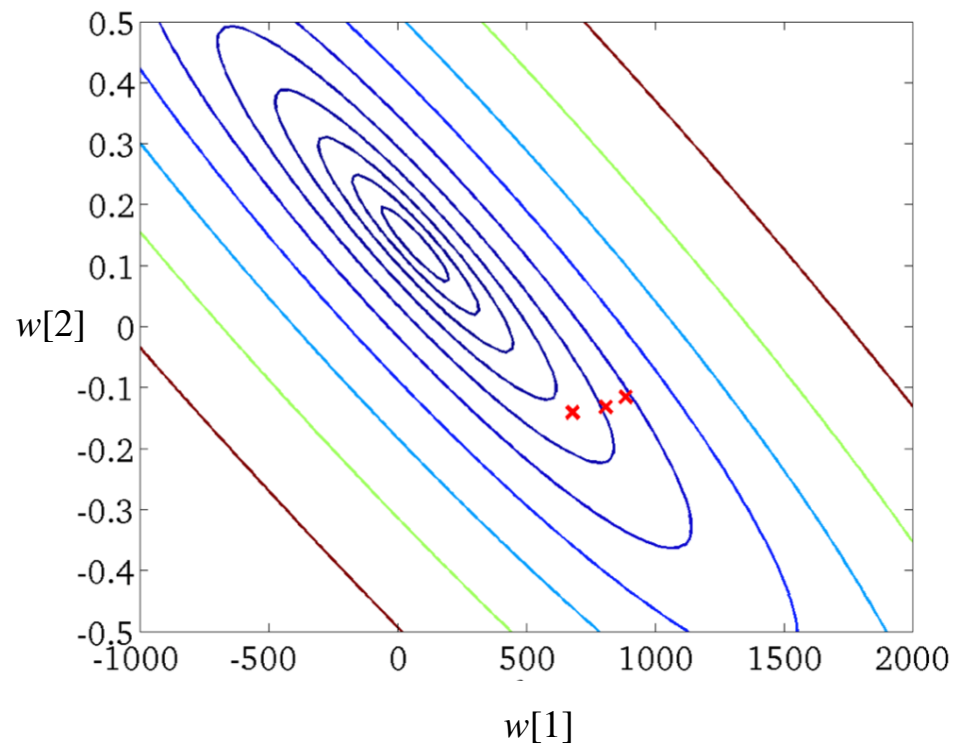
- $w_0 = (900, -0.1)$

- **For $t=0,1,2,\dots$**

- $w_{t+1} \leftarrow w_t - \eta \cdot \nabla_w f(w_t)$



Evolution of the predictor

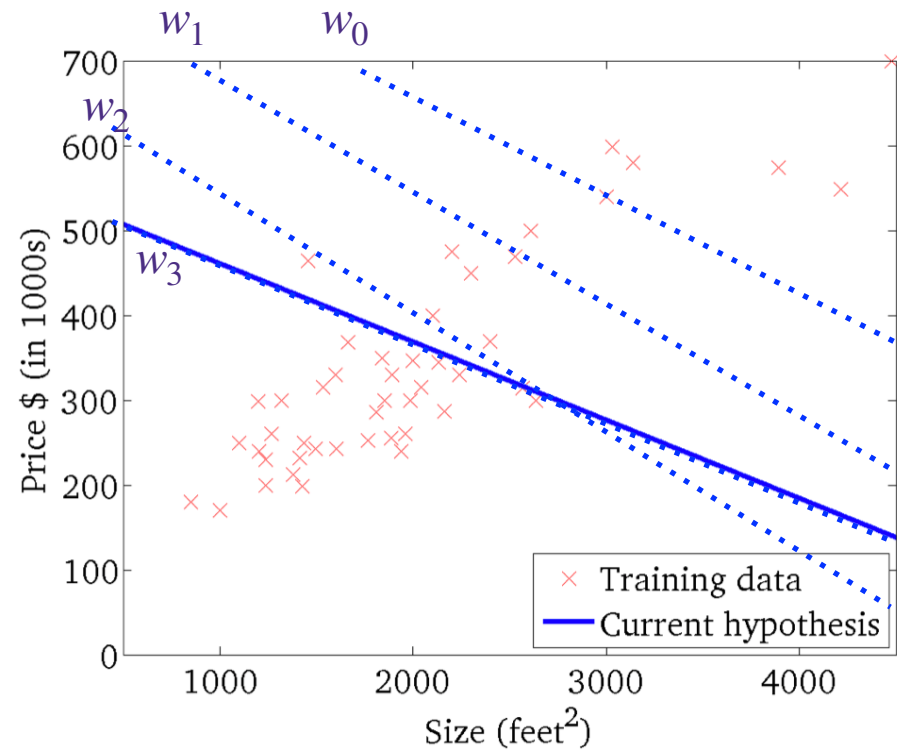


GD dynamics in the Parameter space

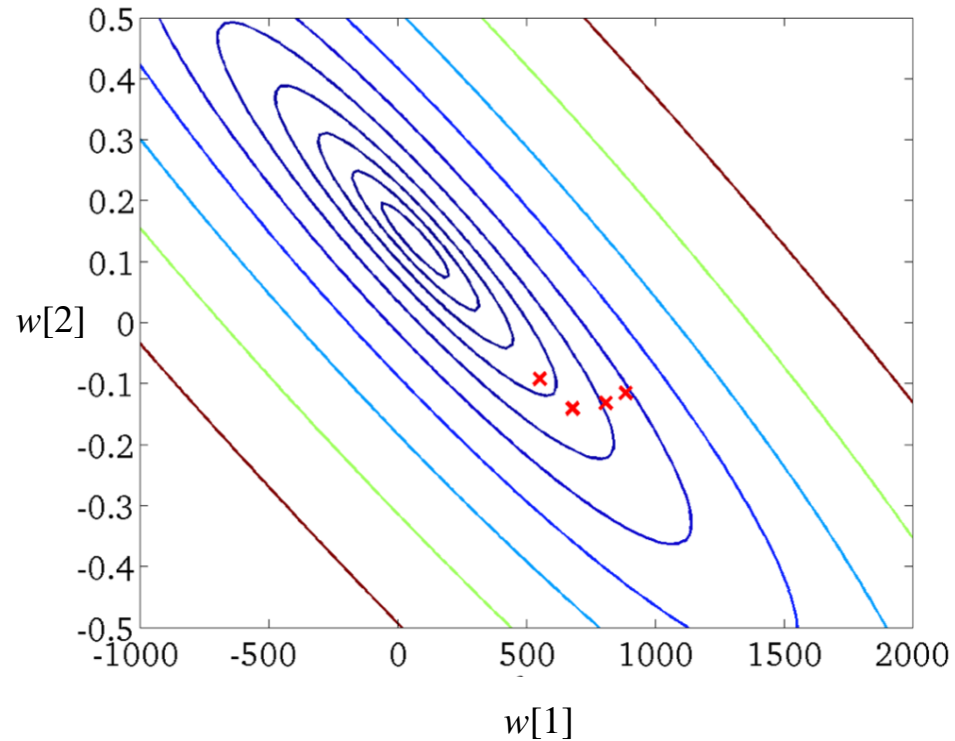
- $w_0 = (900, -0.1)$

- **For $t=0,1,2,\dots$**

- $w_{t+1} \leftarrow w_t - \eta \cdot \nabla_w f(w_t)$



Evolution of the predictor

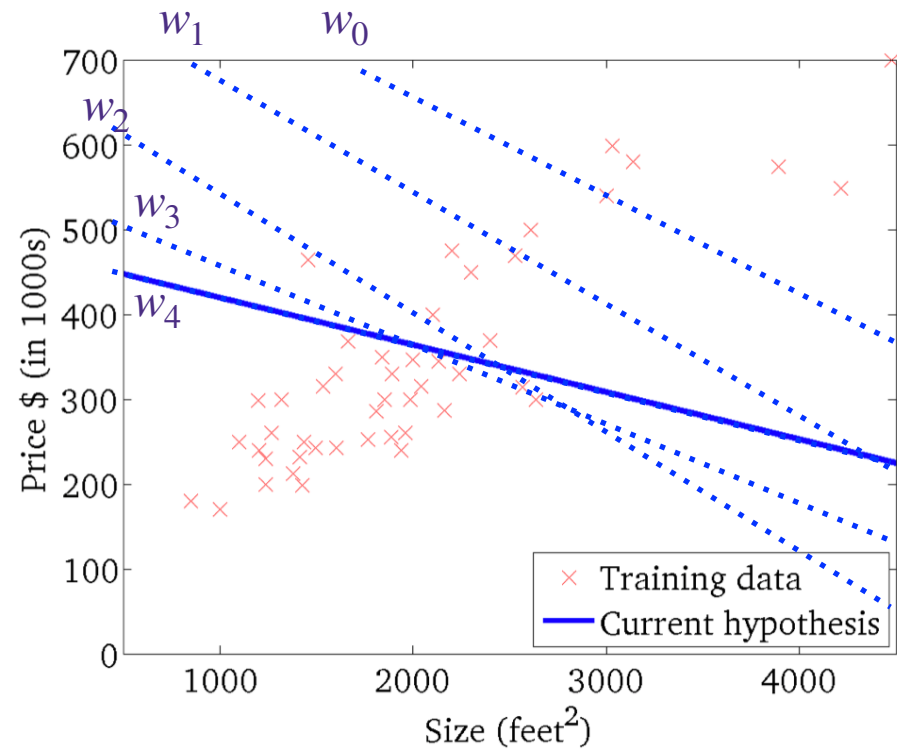


GD dynamics in the Parameter space

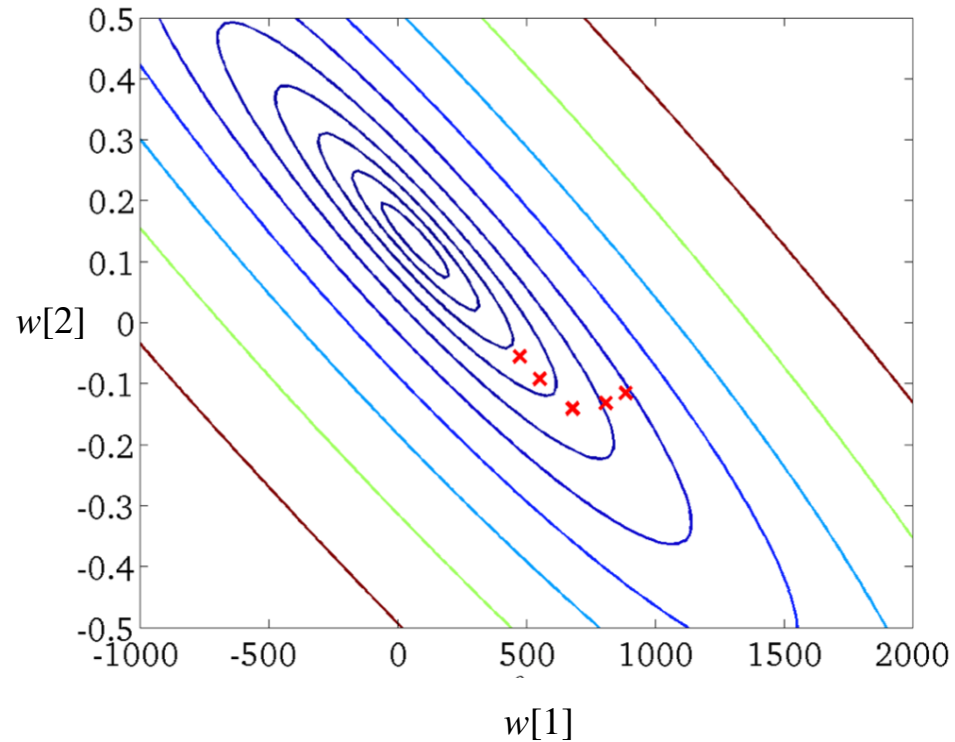
- $w_0 = (900, -0.1)$

- For $t=0,1,2,\dots$

- $w_{t+1} \leftarrow w_t - \eta \cdot \nabla_w f(w_t)$



Evolution of the predictor

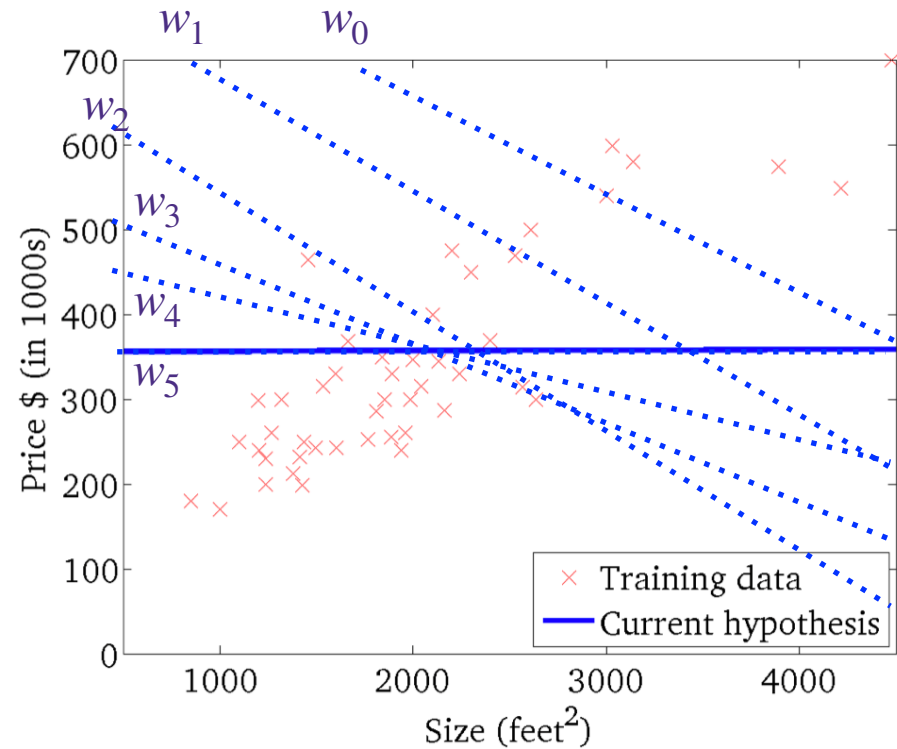


GD dynamics in the Parameter space

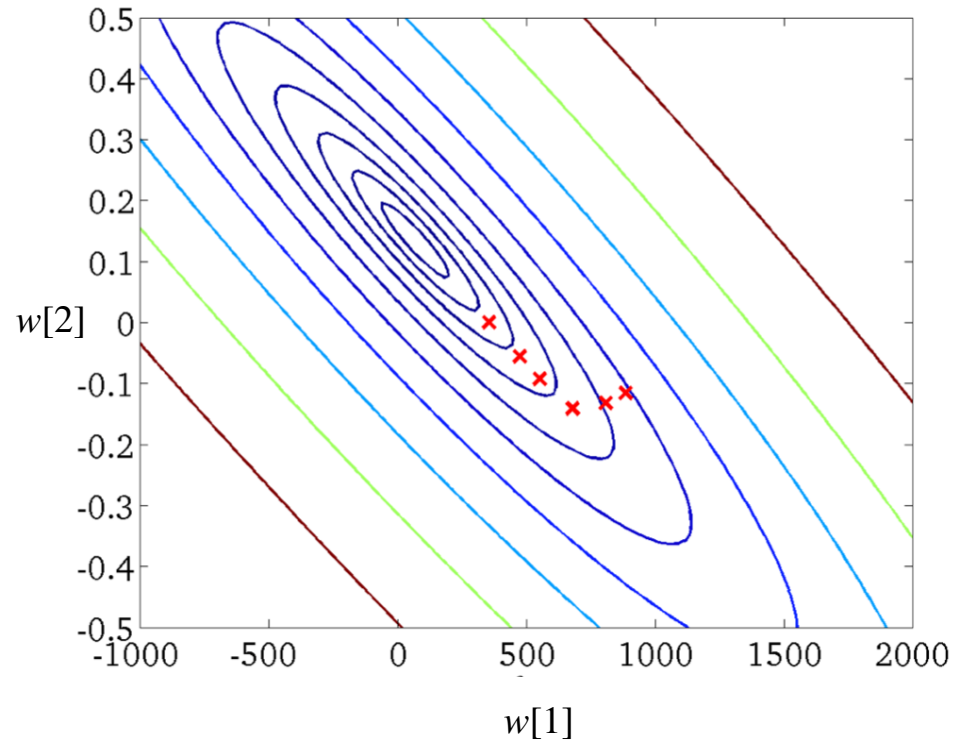
- $w_0 = (900, -0.1)$

- **For $t=0,1,2,\dots$**

- $w_{t+1} \leftarrow w_t - \eta \cdot \nabla_w f(w_t)$



Evolution of the predictor

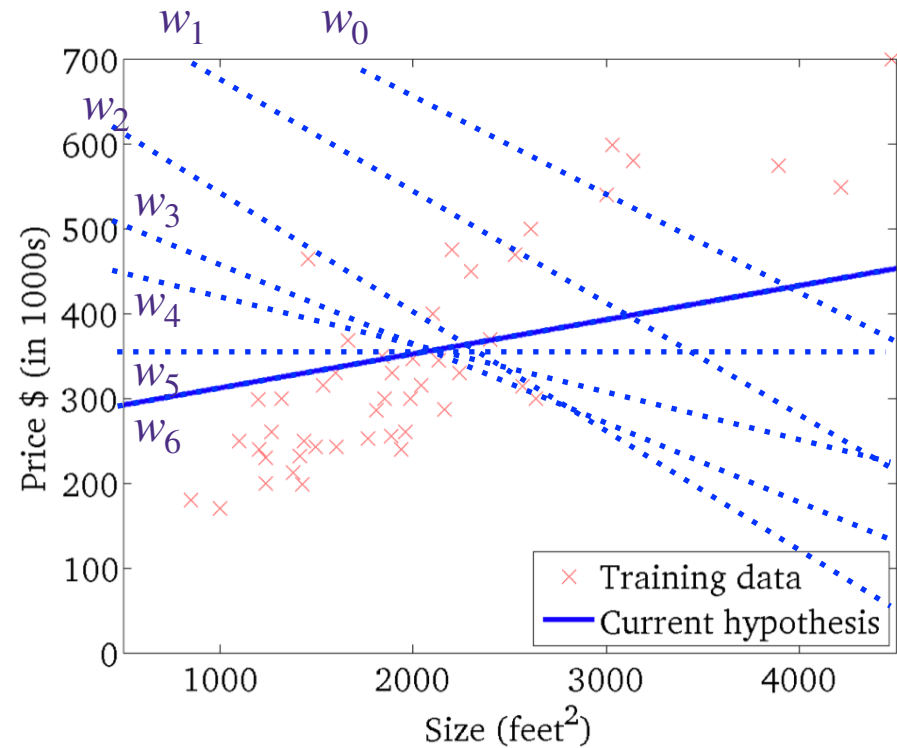


GD dynamics in the Parameter space

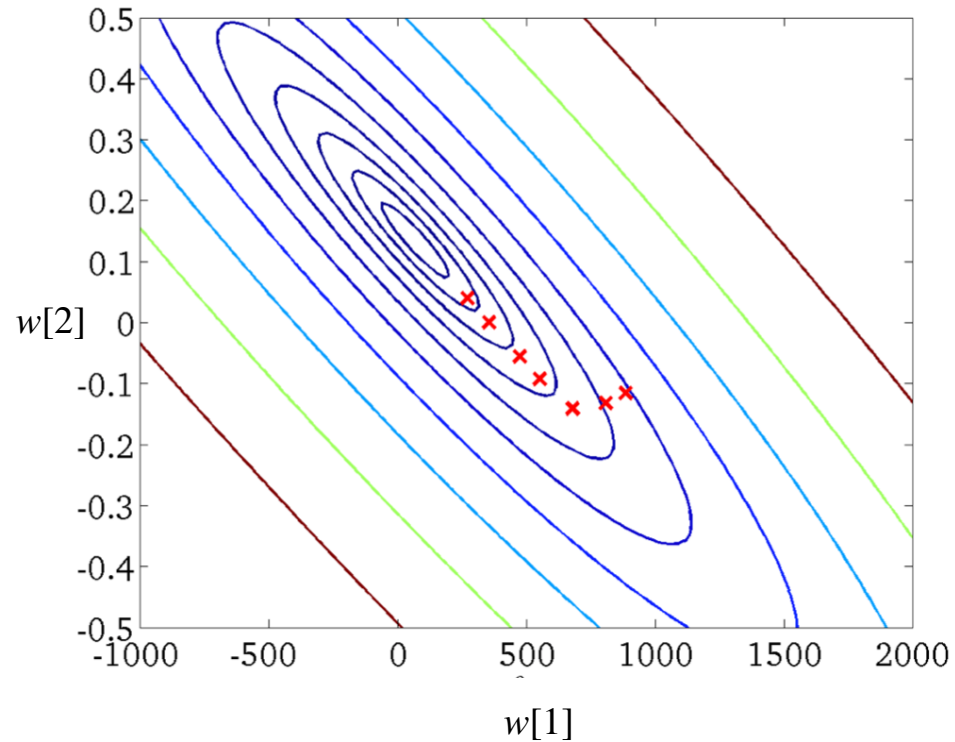
- $w_0 = (900, -0.1)$

- For $t=0,1,2,\dots$

- $w_{t+1} \leftarrow w_t - \eta \cdot \nabla_w f(w_t)$



Evolution of the predictor

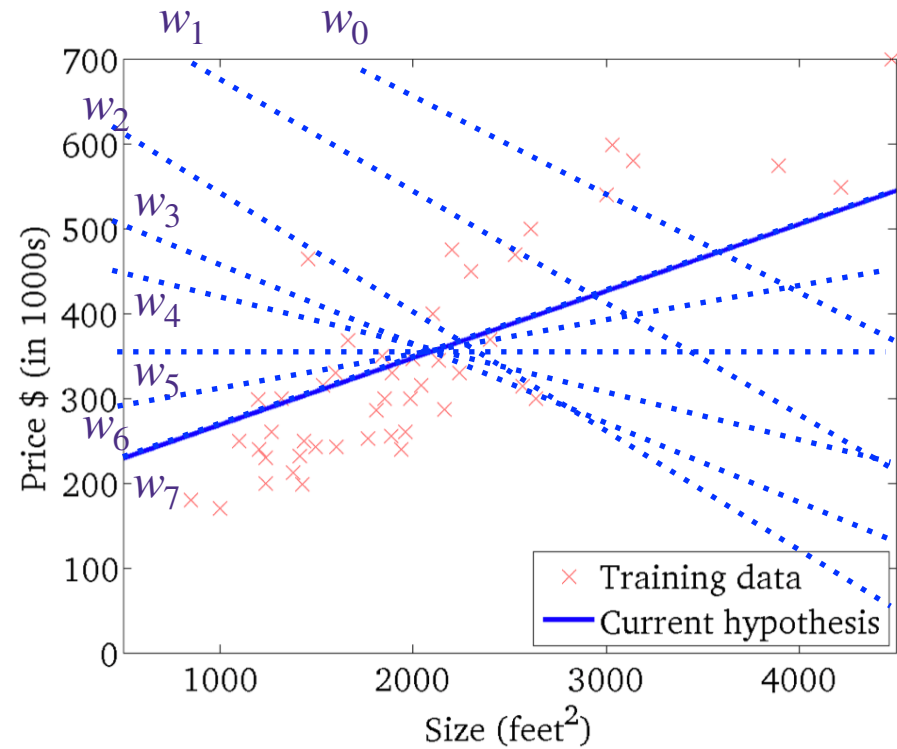


GD dynamics in the Parameter space

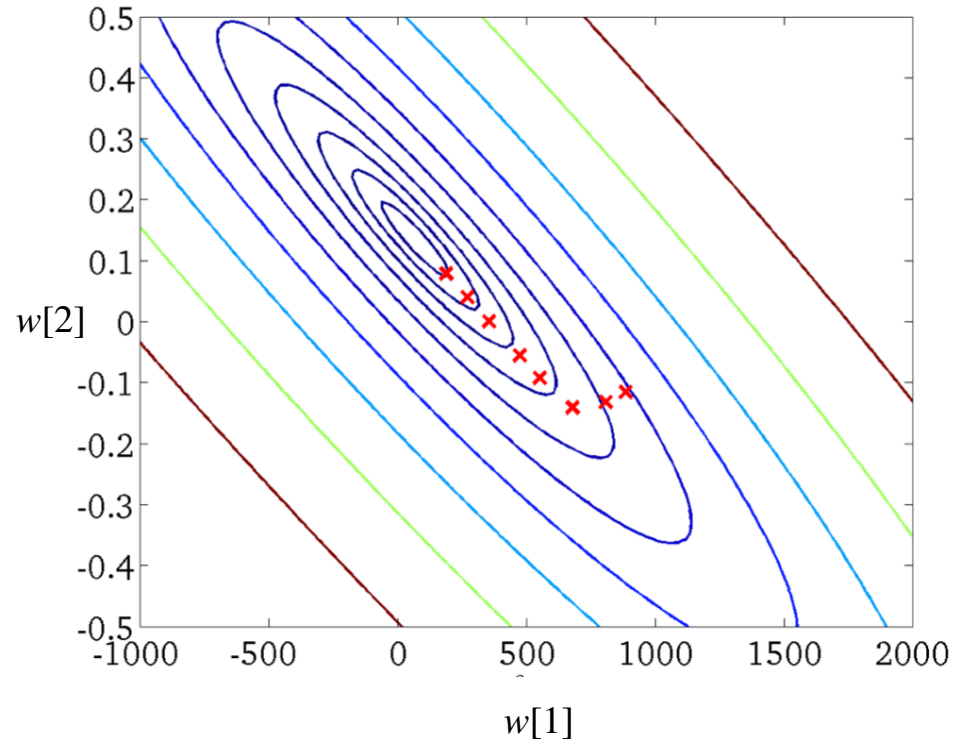
- $w_0 = (900, -0.1)$

- For $t=0,1,2,\dots$

- $w_{t+1} \leftarrow w_t - \eta \cdot \nabla_w f(w_t)$



Evolution of the predictor

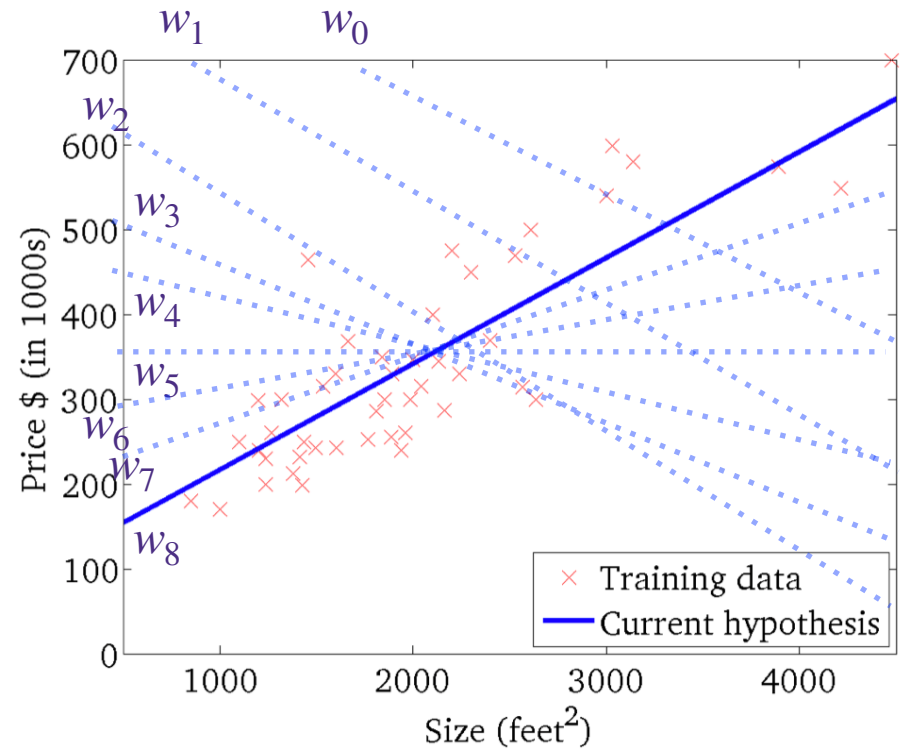


GD dynamics in the Parameter space

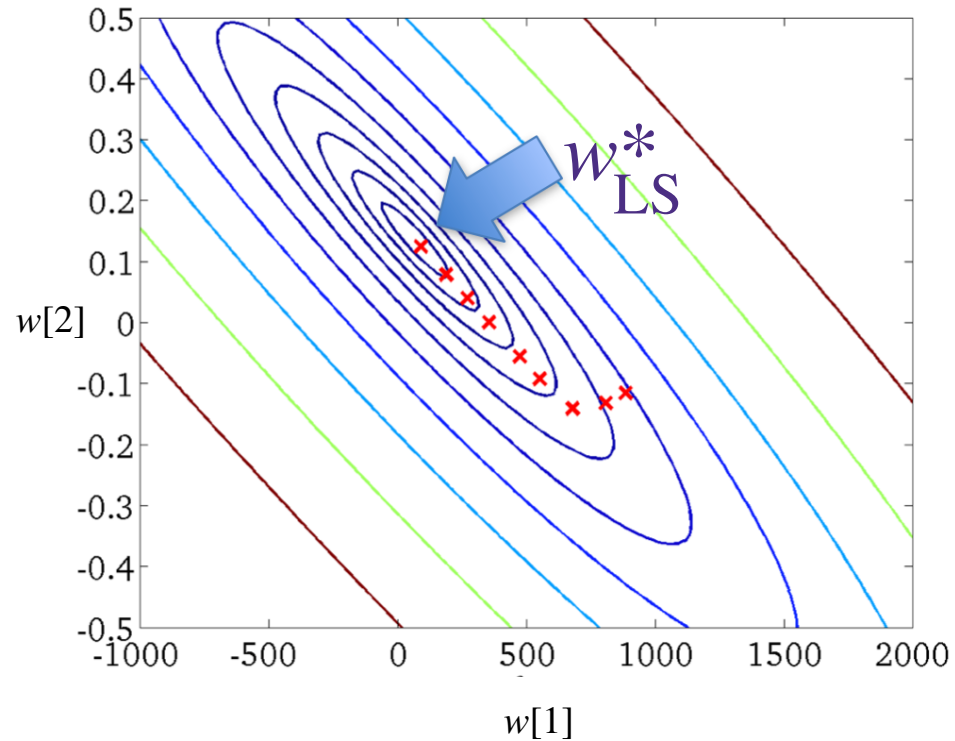
- $w_0 = (900, -0.1)$

- **For $t=0,1,2,\dots$**

- $w_{t+1} \leftarrow w_t - \eta \cdot \nabla_w f(w_t)$



Evolution of the predictor



GD dynamics in the Parameter space

Gradient descent for linear regression

- In this example of linear regression, we can derive exactly the gradient descent trajectory
- Initialize: $w_0 = 0$
- **For** $t=0,1,2,\dots$
 - $w_{t+1} \leftarrow w_t - \eta \cdot \nabla_w f(w_t)$

$$\nabla f(w_t) = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}w_t)$$

For linear regression, we have

$$\hat{w}_{\text{LS}} = \arg \min_{w \in \mathbb{R}^d} \underbrace{\|\mathbf{y} - \mathbf{X}w\|_2^2}_{f(w)}$$

Gradient descent for linear regression

- In this example of linear regression, we can derive exactly the gradient descent trajectory
- Initialize: $w_0 = 0$
- **For** $t=0,1,2,\dots$
 - $w_{t+1} \leftarrow w_t - \eta \cdot \nabla_w f(w_t)$

For linear regression, we have

$$\hat{w}_{\text{LS}} = \arg \min_{w \in \mathbb{R}^d} \underbrace{\|y - Xw\|_2^2}_{f(w)}$$

$$\nabla f(w_t) = -2X^T(y - Xw_t)$$

$$w_{t+1} = w_t + \eta 2X^T(y - Xw_t) = (\mathbf{I} - 2\eta X^T X)w_t + 2\eta X^T y$$

Gradient descent for linear regression

- In this example of linear regression, we can derive exactly the gradient descent trajectory
- Initialize: $w_0 = 0$
- **For** $t=0,1,2,\dots$
 - $w_{t+1} \leftarrow w_t - \eta \cdot \nabla_w f(w_t)$

For linear regression, we have

$$\hat{w}_{\text{LS}} = \arg \min_{w \in \mathbb{R}^d} \underbrace{\|y - Xw\|_2^2}_{f(w)}$$

$$\nabla f(w_t) = -2X^T(y - Xw_t)$$

$$w_{t+1} = w_t + \eta 2X^T(y - Xw_t) = (\mathbf{I} - 2\eta X^T X)w_t + 2\eta X^T y$$

Let the least-squares solution be $w^* = (X^T X)^{-1} X^T y$

$$\begin{aligned} w_{t+1} - w^* &= (\mathbf{I} - 2\eta X^T X)w_t + 2\eta X^T y - w^* \\ &= (\mathbf{I} - 2\eta X^T X)(w_t - w^*) + 2\eta X^T y - 2\eta X^T X w^* \\ &= (\mathbf{I} - 2\eta X^T X)(w_t - w^*) \end{aligned}$$

Gradient descent for linear regression

- Initialize: $w_0 = 0$
- **For** $t=0,1,2,\dots$
 - $w_{t+1} \leftarrow w_t - \eta \cdot \nabla_w f(w_t)$

For linear regression, we have

$$\hat{w}_{\text{LS}} = \arg \min_{w \in \mathbb{R}^d} \underbrace{\|\mathbf{y} - \mathbf{X}w\|_2^2}_{f(w)}$$

$$\nabla f(w_t) = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}w_t)$$

$$w_{t+1} = w_t + \eta 2\mathbf{X}^T(\mathbf{y} - \mathbf{X}w_t) = (\mathbf{I} - 2\eta\mathbf{X}^T\mathbf{X})w_t + 2\eta\mathbf{X}^T\mathbf{y}$$

Let the least-squares solution be $w^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$

$$\begin{aligned} w_{t+1} - w^* &= (\mathbf{I} - 2\eta\mathbf{X}^T\mathbf{X})w_t + 2\eta\mathbf{X}^T\mathbf{y} - w^* \\ &= (\mathbf{I} - 2\eta\mathbf{X}^T\mathbf{X})(w_t - w^*) + 2\eta\mathbf{X}^T\mathbf{y} - 2\eta\mathbf{X}^T\mathbf{X}w^* \\ &= (\mathbf{I} - 2\eta\mathbf{X}^T\mathbf{X})(w_t - w^*) \end{aligned}$$

How do you choose step size?

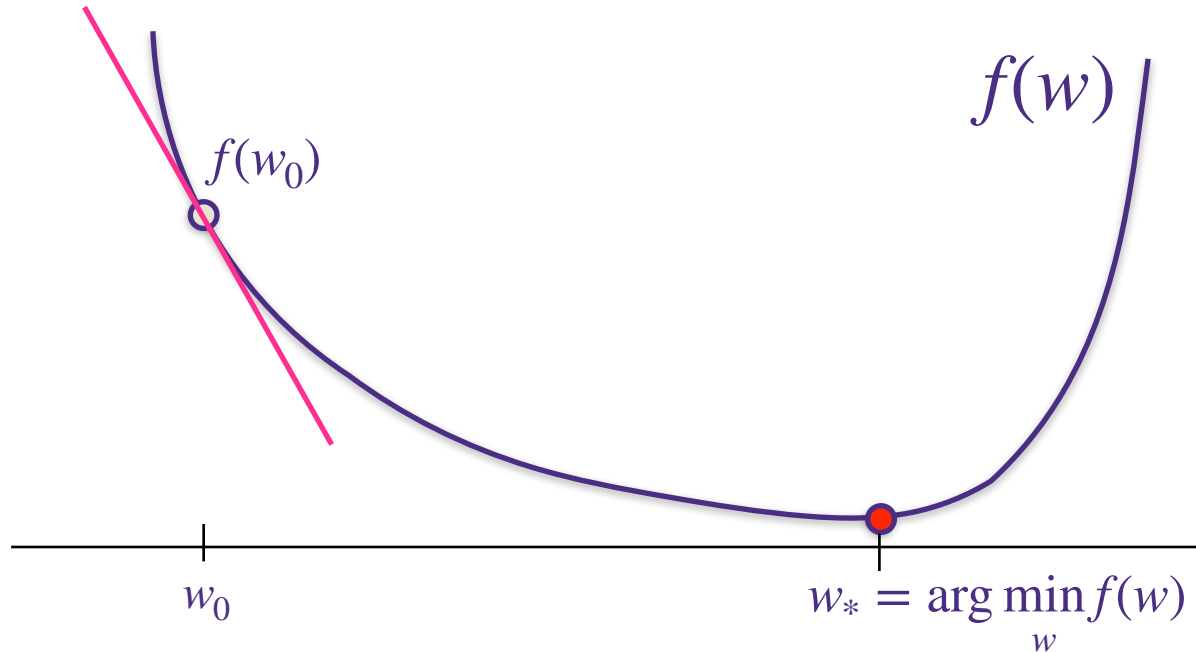
Let w_0 be an initial guess. How can we improve this solution?

Taylor series approximation:

For w very close to w_0 we have

$$f(w_0) + (w - w_0) \frac{df(w)}{dw} \Big|_{w=w_0}$$

is very close to $f(w)$



If η too big, does not converge!

If η too small, converges very, very slowly.

In practice: choose the largest value of η that converges (guess and check)

Stochastic Gradient Descent

Machine Learning Problems

- **Given data:**

$$\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d \quad y_i \in \mathbb{R}$$

- **Learning a model's parameters:** $\sum_{i=1}^n \ell_i(w)$

Gradient Descent:

$$w_{t+1} = w_t - \eta \nabla_w \left(\frac{1}{n} \sum_{i=1}^n \ell_i(w) \right) \Big|_{w=w_t}$$

Machine Learning Problems

- Given data:

$$\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d \quad y_i \in \mathbb{R}$$

- Learning a model's parameters: $\sum_{i=1}^n \ell_i(w)$

Gradient Descent:

$$w_{t+1} = w_t - \eta \nabla_w \left(\frac{1}{n} \sum_{i=1}^n \ell_i(w) \right) \Big|_{w=w_t}$$

Stochastic Gradient Descent:

$$w_{t+1} = w_t - \eta \nabla_w \ell_{I_t}(w) \Big|_{w=w_t} \quad I_t \text{ drawn uniform at random from } \{1, \dots, n\}$$

$$\mathbb{E}[\nabla \ell_{I_t}(w)] =$$

Machine Learning Problems

- Learning a model's parameters:

$$\sum_{i=1}^n \ell_i(w)$$

Stochastic Gradient Descent:

$$w_{t+1} = w_t - \eta \nabla_w \ell_{I_t}(w) \Big|_{w=w_t}$$

I_t drawn uniform at
random from $\{1, \dots, n\}$

$$\mathbb{E}[\nabla \ell_{I_t}(w)] = \nabla \ell(w)$$

Mini-batch SGD

- Instead of one iterate, average B stochastic gradient together
- Advantages:
 - Smaller variance: the variance of the stochastic gradient is smaller by a factor of $1/\sqrt{B}$
 - Parallelization: each gradient in the mini-batch can be computed in parallel

- If you have regularizer, $\frac{1}{n} \sum_{i=1}^n \ell_i(w) + r(w)$, then update with the stochastic gradient of the loss and gradient of the regularizer

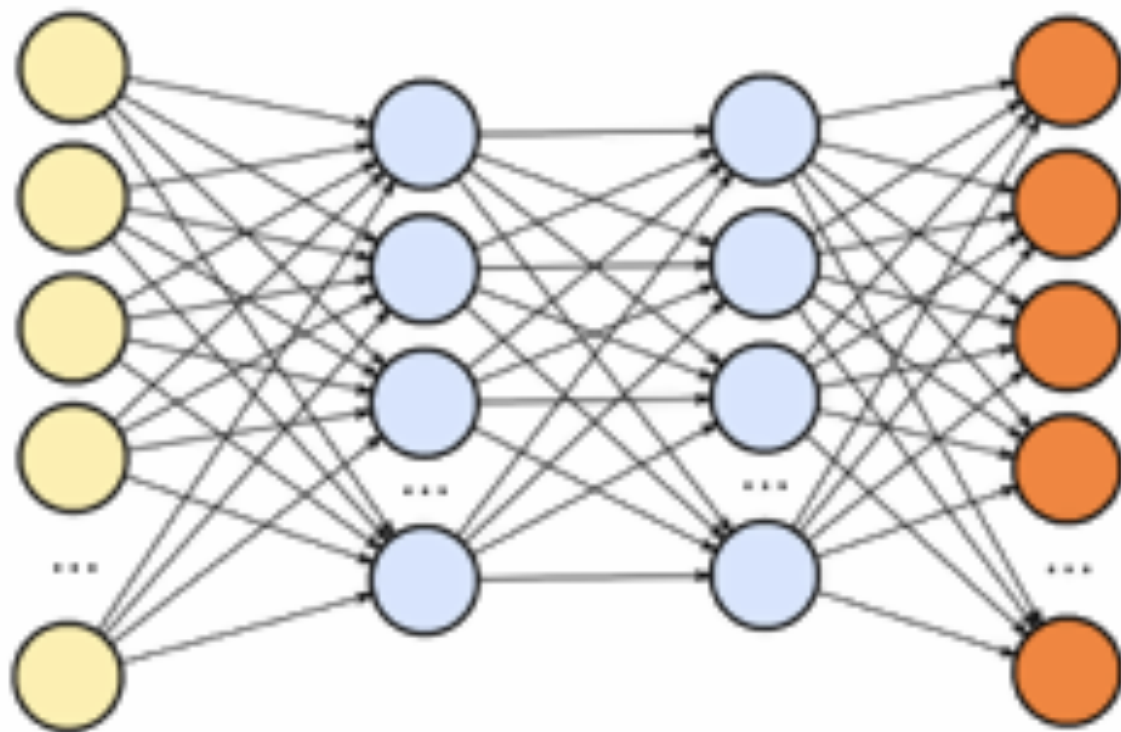
Fully-Connected Neural Networks



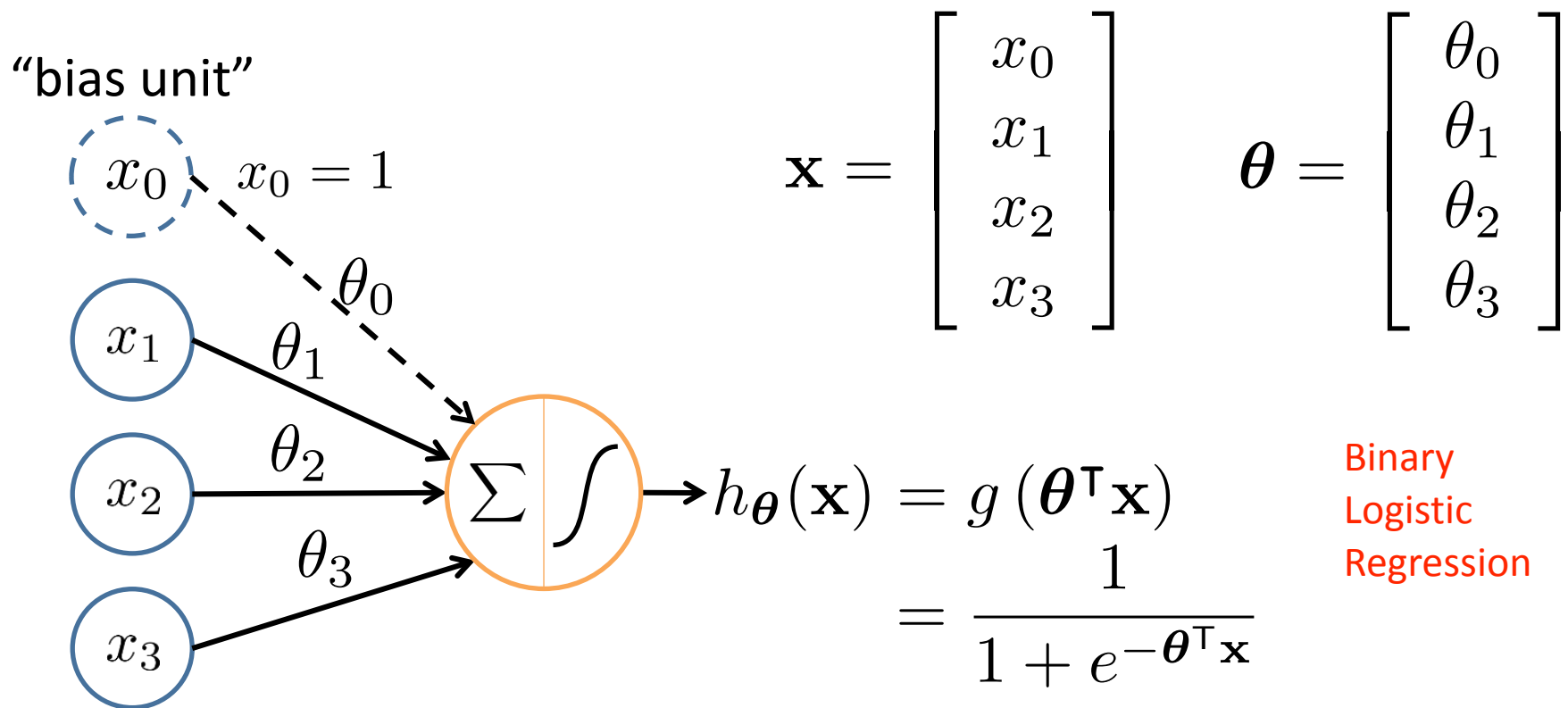
Neural Networks

- Origins: Algorithms that try to mimic the brain.
- Widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence from 10s: state-of-the-art techniques for many applications:
 - Computer Vision
 - Natural language processing: e.g., GPT
 - Speech recognition
 - Decision-making / control problems (AlphaGo, Dota, robots)
- Limited theory:
 - Non-convexity
 - Model are complex but generalization error is small

Neural Networks

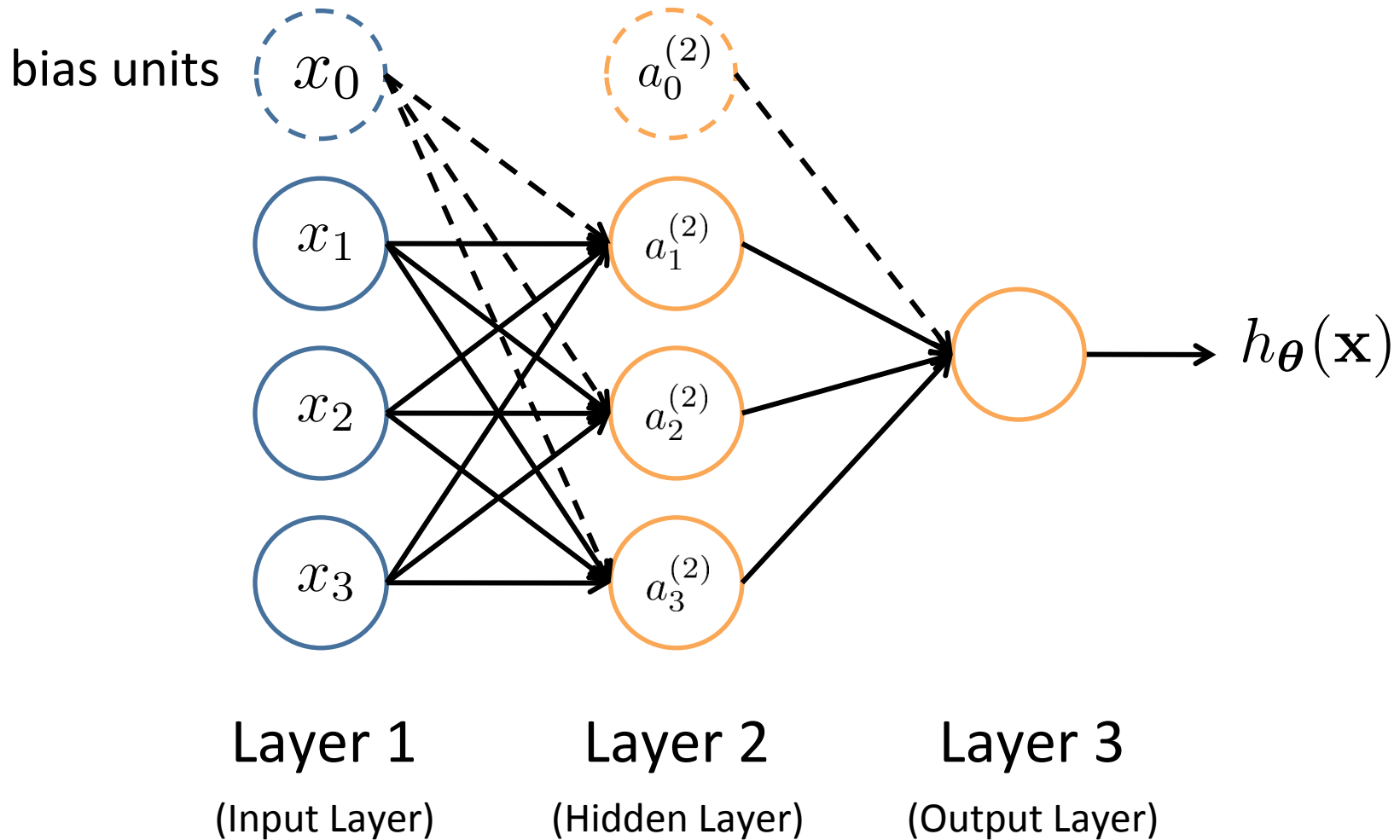


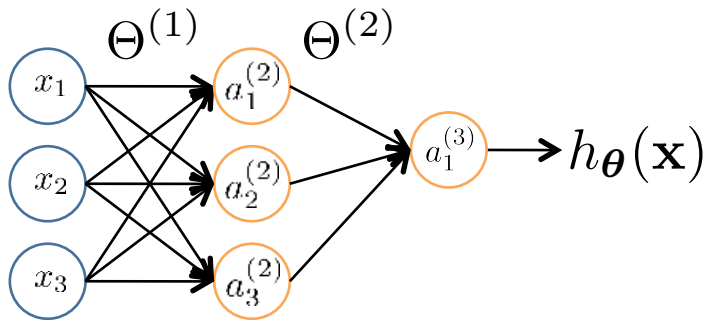
Single Node



Sigmoid (logistic) activation function: $g(z) = \frac{1}{1 + e^{-z}}$

Neural Network





$a_i^{(j)}$ = “activation” of unit i in layer j
 $\Theta^{(j)}$ = weight matrix stores parameters
 from layer j to layer $j + 1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

$$h_{\Theta}(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

If network has s_j units in layer j and s_{j+1} units in layer $j+1$,
 then $\Theta^{(j)}$ has dimension $s_{j+1} \times (s_j + 1)$.

$$\Theta^{(1)} \in \mathbb{R}^{3 \times 4} \quad \Theta^{(2)} \in \mathbb{R}^{1 \times 4}$$

Multi-layer Neural Network - Binary Classification

$$a^{(1)} = x$$

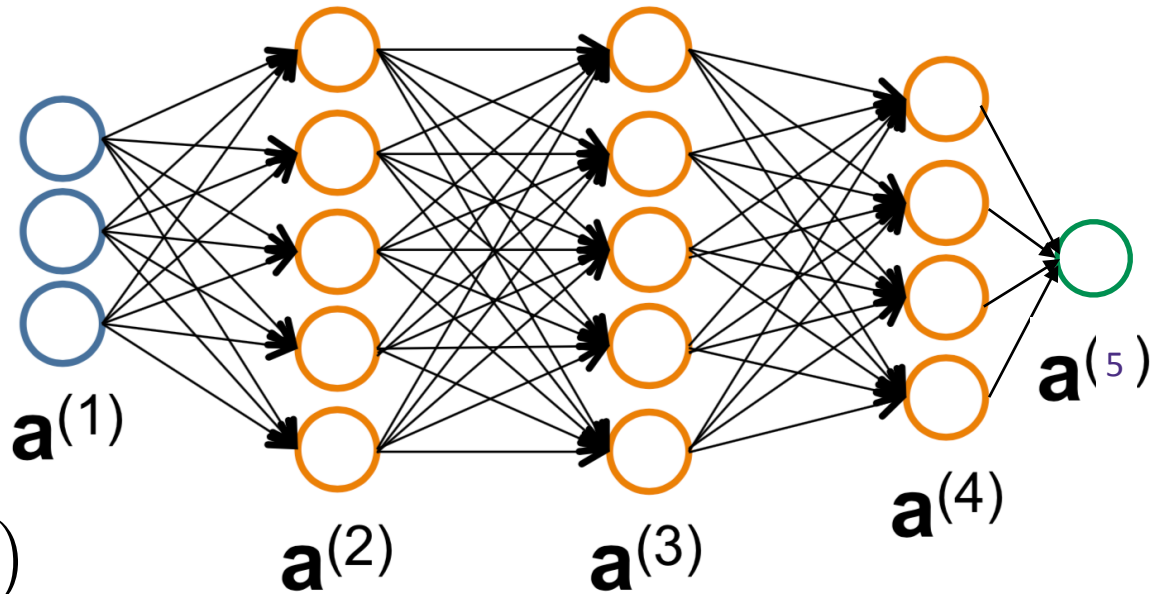
$$a^{(2)} = g(\Theta^{(1)} a^{(1)})$$

⋮

$$a^{(l+1)} = g(\Theta^{(l)} a^{(l)})$$

⋮

$$\hat{y} = g(\Theta^{(L)} a^{(L)})$$



$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

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

Binary
Logistic
Regression

Multi-layer Neural Network - Binary Classification

$$a^{(1)} = x$$

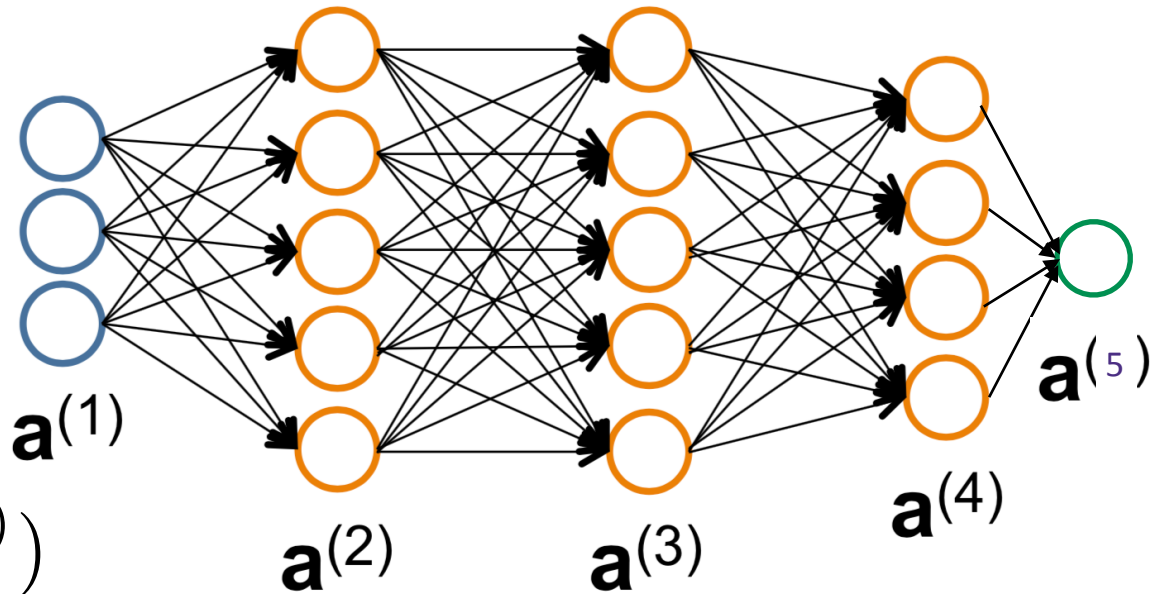
$$a^{(2)} = \sigma(\Theta^{(1)} a^{(1)})$$

$\vdots$

$$a^{(l+1)} = \sigma(\Theta^{(l)} a^{(l)})$$

$\vdots$

$$\hat{y} = g(\Theta^{(L)} a^{(L)})$$



$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

$$\sigma(z) = \max\{0, z\} \quad g(z) = \frac{1}{1 + e^{-z}} \quad \text{Binary Logistic Regression}$$

Multiple Output Units: One-vs-Rest



Pedestrian



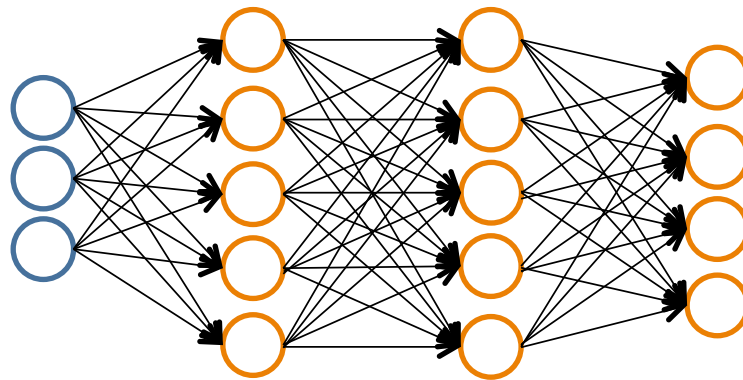
Car



Motorcycle



Truck



$$h_{\Theta}(\mathbf{x}) \in \mathbb{R}^K$$

Multi-class
Logistic
Regression

We want:

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

when pedestrian

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

when car

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

when motorcycle

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

when truck

Multi-layer Neural Network - Regression

$$a^{(1)} = x$$

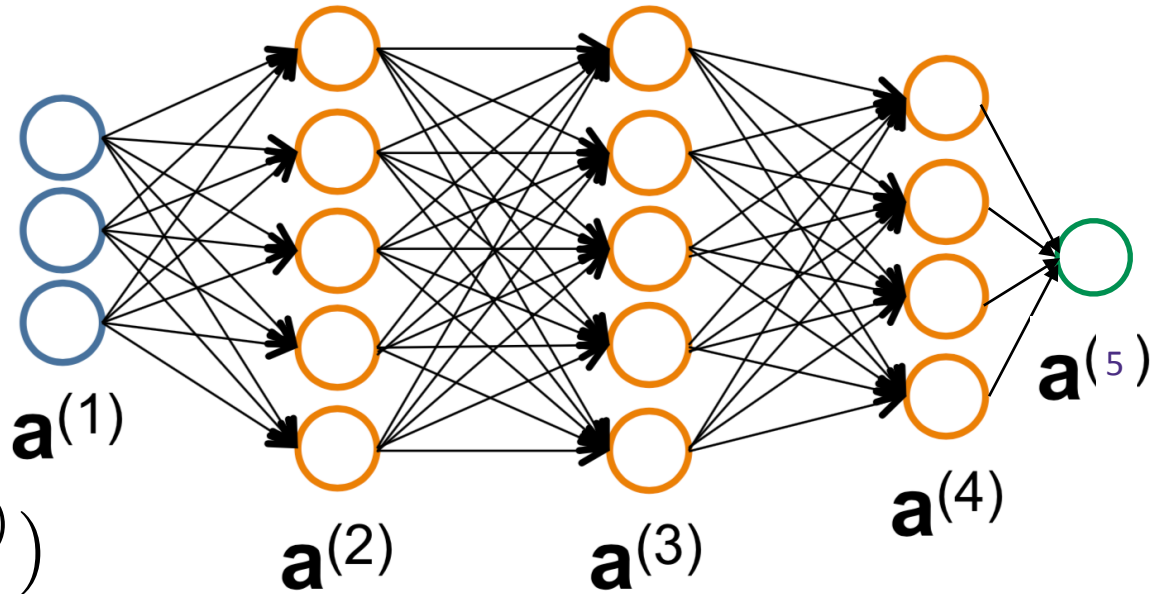
$$a^{(2)} = \sigma(\Theta^{(1)} a^{(1)})$$

$\vdots$

$$a^{(l+1)} = \sigma(\Theta^{(l)} a^{(l)})$$

$\vdots$

$$\hat{y} = \Theta^{(L)} a^{(L)}$$



$$L(y, \hat{y}) = (y - \hat{y})^2$$

$$\sigma(z) = \max\{0, z\}$$

Regression

Neural Network Optimization



Machine Learning Problems

- **Given data:**

$$\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d \quad y_i \in \mathbb{R}$$

- **Learning a model's parameters:** $\sum_{i=1}^n \ell_i(w)$

Logistic Loss: $\ell_i(w) = \log(1 + \exp(-y_i x_i^T w))$

Squared error Loss: $\ell_i(w) = (y_i - x_i^T w)^2$

Machine Learning Problems

- Given data:

$$\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d \quad y_i \in \mathbb{R}$$

- Learning a model's parameters: $\sum_{i=1}^n \ell_i(w)$

Logistic Loss: $\ell_i(w) = \log(1 + \exp(-y_i x_i^T w))$

Squared error Loss: $\ell_i(w) = (y_i - x_i^T w)^2$

Gradient Descent:

$$w_{t+1} = w_t - \eta \nabla_w \left(\frac{1}{n} \sum_{i=1}^n \ell_i(w) \right) \Big|_{w=w_t}$$

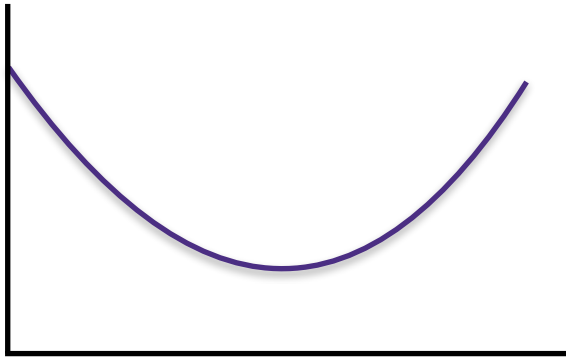
Gradient Descent

Initialize: $w_0 = 0$

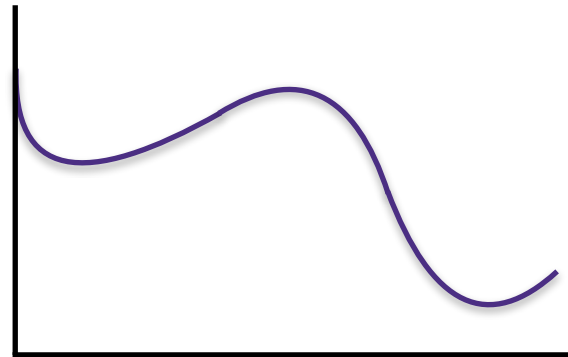
for $t = 1, 2, \dots$

$$w_{t+1} = w_t - \eta \nabla f(w_t)$$

Convex Function



Non-convex Function



Sub-Gradient Descent

Initialize: $w_0 = 0$

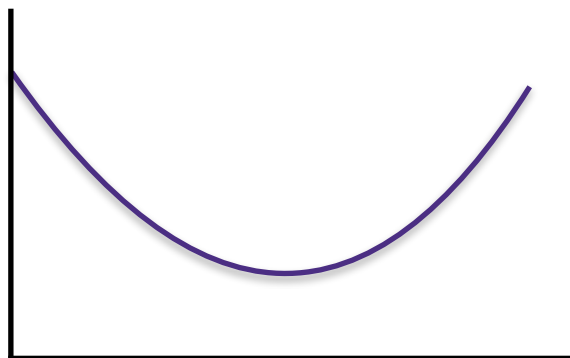
for $t = 1, 2, \dots$

Find any g_t such that $f(y) \geq f(w_t) + g_t^\top (y - w_t)$

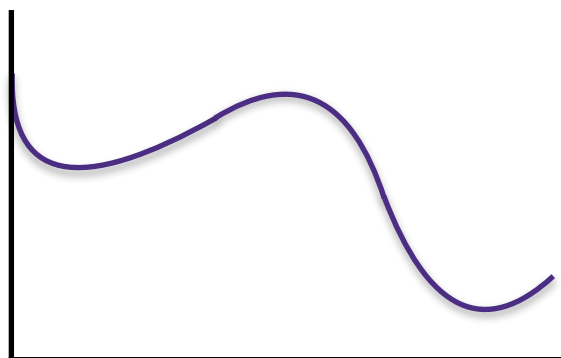
$$w_{t+1} = w_t - \eta g_t$$

g is a subgradient at x if $f(y) \geq f(x) + g^\top (y - x)$

Convex Function



Non-convex Function



Machine Learning Problems

- Given data:

$$\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d \quad y_i \in \mathbb{R}$$

- Learning a model's parameters: $\sum_{i=1}^n \ell_i(w)$

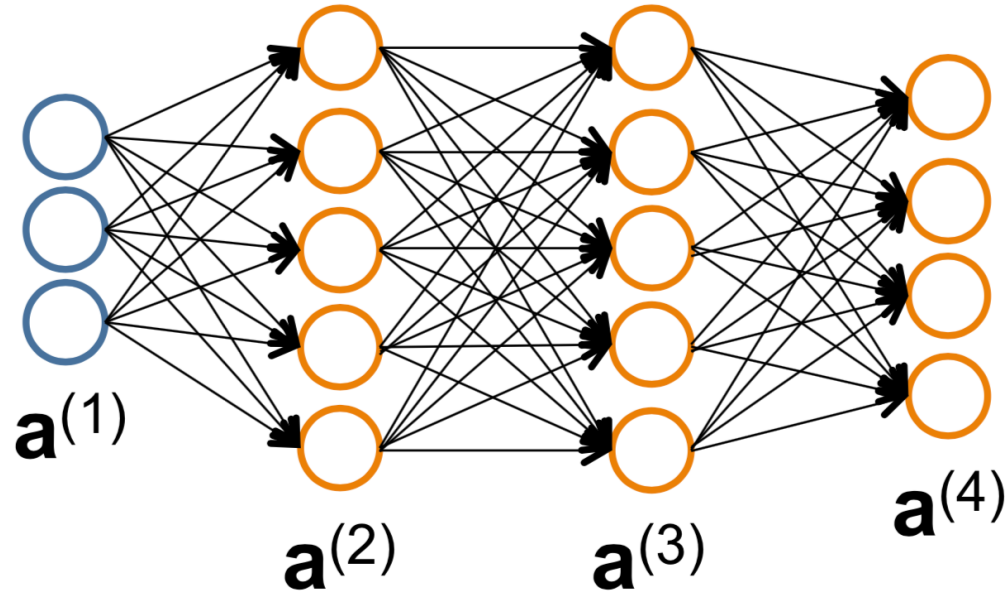
Gradient Descent:

$$w_{t+1} = w_t - \eta \nabla_w \left(\frac{1}{n} \sum_{i=1}^n \ell_i(w) \right) \Big|_{w=w_t}$$

Stochastic Gradient Descent:

$$w_{t+1} = w_t - \eta \nabla_w \ell_{I_t}(w) \Big|_{w=w_t} \quad I_t \text{ drawn uniform at random from } \{1, \dots, n\}$$

Gradient Computation on a Graph



Naive computation: node by node

A brief history

- **Back propagation:** the workhorse for training neural networks. An algorithm that for a network with V nodes and E edges calculates that gradient in **linear time** $O(V+E)$.
- The name was introduced by Rumelhart, Hinton, Williams '86. Same idea was rediscovered multiple times. Also mentioned in Werbos' thesis '74 in the context of neural networks.
- **Control theory:** Kelly '60, Bryson '61 [**dynamic programming**]
- **Theoretical computer science:** Baur-Strassen lemma '83 [**algebraic circuits**]

$$\underline{a^{(1)} = x}$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g(z^{(2)})$$

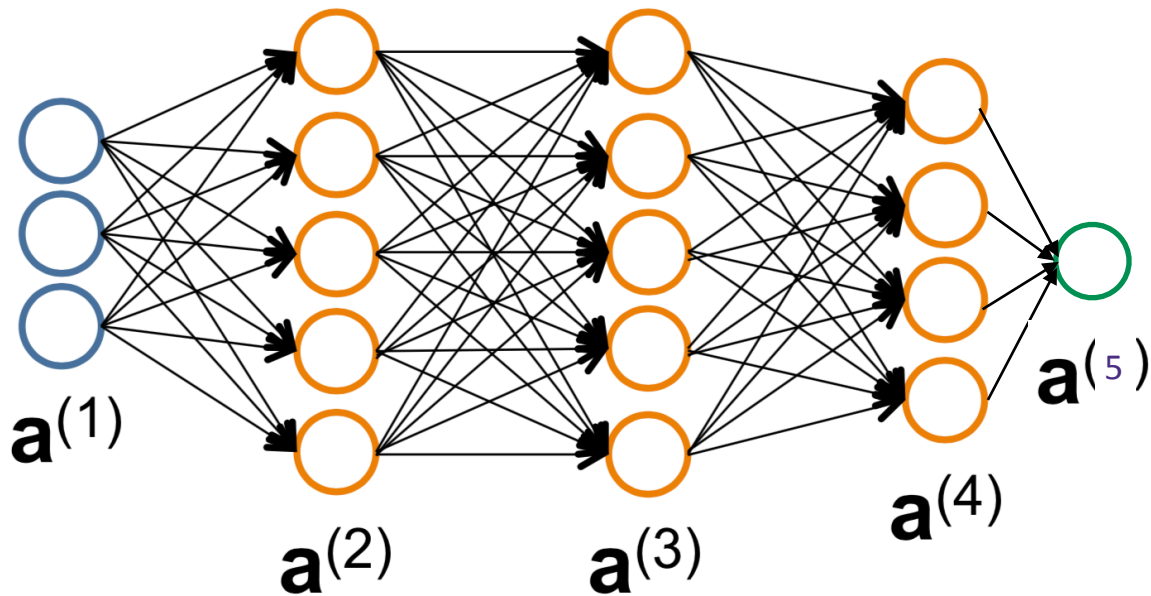
$$\vdots$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g(z^{(l+1)})$$

$$\vdots$$

$$\hat{y} = g(\Theta^{(L)} a^{(L)})$$



$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

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

$$\text{Gradient Descent: } \Theta^{(l)} \leftarrow \Theta^{(l)} - \eta \nabla_{\Theta^{(l)}} L(y, \hat{y}) \quad \forall l$$

Forward Propagation

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g(z^{(2)})$$

$$\vdots$$

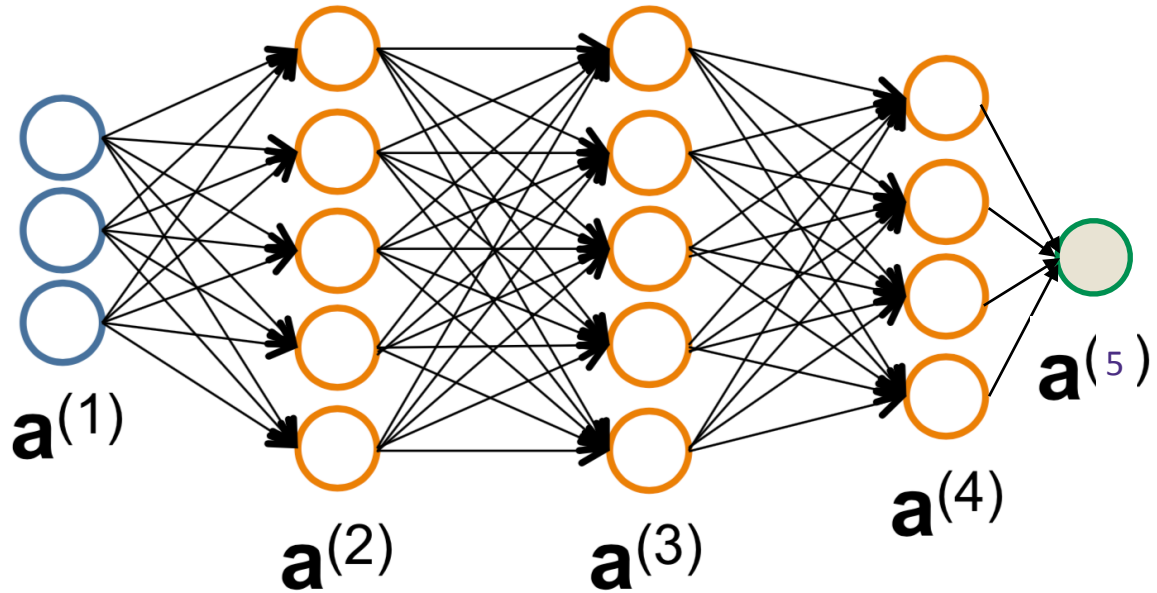
$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g(z^{(l+1)})$$

$$\vdots$$

$$\hat{y} = a^{(L+1)}$$



$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

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

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g(z^{(2)})$$

$$\vdots$$

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g(z^{(l+1)})$$

$$\vdots$$

$$\hat{y} = a^{(L+1)}$$

Train by Stochastic Gradient Descent:

$$\Theta_{i,j}^{(l)} \leftarrow \Theta_{i,j}^{(l)} - \eta \frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}}$$

$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \quad \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g(z^{(2)})$$

$$\vdots$$
$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g(z^{(l+1)})$$

$$\vdots$$
$$\hat{y} = a^{(L+1)}$$

$$\frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

Train by Stochastic Gradient Descent:

$$\Theta_{i,j}^{(l)} \leftarrow \Theta_{i,j}^{(l)} - \eta \frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}}$$

$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \quad \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g(z^{(2)})$$

$$\vdots$$
$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g(z^{(l+1)})$$

$$\vdots$$
$$\hat{y} = a^{(L+1)}$$

$$\frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\delta_i^{(l)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l)}} = \sum_k \frac{\partial L(y, \hat{y})}{\partial z_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}}$$

$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

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

$$\delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g(z^{(2)})$$

$$\vdots$$
$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g(z^{(l+1)})$$

$$\vdots$$
$$\hat{y} = a^{(L+1)}$$

$$\frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\begin{aligned} \delta_i^{(l)} &= \frac{\partial L(y, \hat{y})}{\partial z_i^{(l)}} = \sum_k \frac{\partial L(y, \hat{y})}{\partial z_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}} \\ &= \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)} g'(z_i^{(l)}) \\ &= a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)} \end{aligned}$$

$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \quad \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g(z^{(2)})$$

$$\vdots$$
$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g(z^{(l+1)})$$

$$\vdots$$
$$\hat{y} = a^{(L+1)}$$

$$\frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\delta_i^{(l)} = a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)}$$

$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \quad \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g(z^{(2)})$$

$$\vdots$$
$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g(z^{(l+1)})$$

$$\vdots$$
$$\hat{y} = a^{(L+1)}$$

$$\frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\delta_i^{(l)} = a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)}$$

$$\begin{aligned} \delta_i^{(L+1)} &= \frac{\partial L(y, \hat{y})}{\partial z_i^{(L+1)}} = \frac{\partial}{\partial z_i^{(L+1)}} [y \log(g(z^{(L+1)})) + (1 - y) \log(1 - g(z^{(L+1)}))] \\ &= \frac{y}{g(z^{(L+1)})} g'(z^{(L+1)}) - \frac{1 - y}{1 - g(z^{(L+1)})} g'(z^{(L+1)}) \\ &= y - g(z^{(L+1)}) = y - a^{(L+1)} \end{aligned}$$

$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \quad \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g(z^{(2)})$$

$$\vdots$$
$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g(z^{(l+1)})$$

$$\vdots$$
$$\hat{y} = a^{(L+1)}$$

$$\frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\delta_i^{(l)} = a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)}$$

$$\delta^{(L+1)} = y - a^{(L+1)}$$

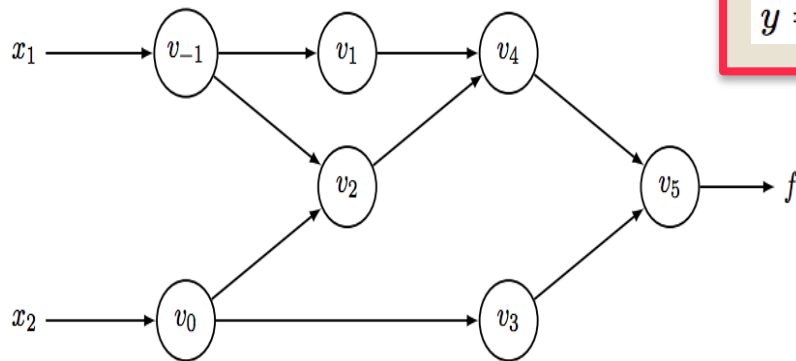
Recursive Algorithm!

$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \quad \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}}$$

Auto-differentiation

Backprop for this simple network architecture is a special case of *reverse-mode auto-differentiation*:



$$y = f(x_1, x_2) = \ln(x_1) + x_1x_2 - \sin(x_2)$$

Forward Primal Trace

$v_{-1} = x_1$	$= 2$
$v_0 = x_2$	$= 5$
$v_1 = \ln v_{-1}$	$= \ln 2$
$v_2 = v_{-1} \times v_0$	$= 2 \times 5$
$v_3 = \sin v_0$	$= \sin 5$
$v_4 = v_1 + v_2$	$= 0.693 + 10$
$v_5 = v_4 - v_3$	$= 10.693 + 0.959$
$y = v_5$	$= 11.652$

Reverse Adjoint (Derivative) Trace

$\bar{x}_1 = \bar{v}_{-1}$	$= 5.5$
$\bar{x}_2 = \bar{v}_0$	$= 1.716$
$\bar{v}_{-1} = \bar{v}_{-1} + \bar{v}_1 \frac{\partial v_1}{\partial v_{-1}} = \bar{v}_{-1} + \bar{v}_1 / v_{-1}$	$= 5.5$
$\bar{v}_0 = \bar{v}_0 + \bar{v}_2 \frac{\partial v_2}{\partial v_0} = \bar{v}_0 + \bar{v}_2 \times v_{-1}$	$= 1.716$
$\bar{v}_{-1} = \bar{v}_2 \frac{\partial v_2}{\partial v_{-1}} = \bar{v}_2 \times v_0$	$= 5$
$\bar{v}_0 = \bar{v}_3 \frac{\partial v_3}{\partial v_0} = \bar{v}_3 \times \cos v_0$	$= -0.284$
$\bar{v}_2 = \bar{v}_4 \frac{\partial v_4}{\partial v_2} = \bar{v}_4 \times 1$	$= 1$
$\bar{v}_1 = \bar{v}_4 \frac{\partial v_4}{\partial v_1} = \bar{v}_4 \times 1$	$= 1$
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3} = \bar{v}_5 \times (-1)$	$= -1$
$\bar{v}_4 = \bar{v}_5 \frac{\partial v_5}{\partial v_4} = \bar{v}_5 \times 1$	$= 1$
$\bar{v}_5 = \bar{y}$	$= 1$

Auto-differentiation

- Given a function, computes its partial derivatives
- Compute all of the partial derivatives of a function with (nearly) same computation runtime [Griewank '89, Baur and Strassen '83]
- Backbone of (applied) machine learning: Pytorch, Tensorflow, ...

Example of Computation Graph

$$f(w_1, w_2) = \left(\sin\left(\frac{2\pi w_1}{w_2}\right) + \frac{3w_1}{w_2} - \exp(2w_2) \right) \cdot \left(\frac{3w_1}{w_2} - \exp(2w_2) \right)$$

Input: $z_0 = (w_1, w_2)$

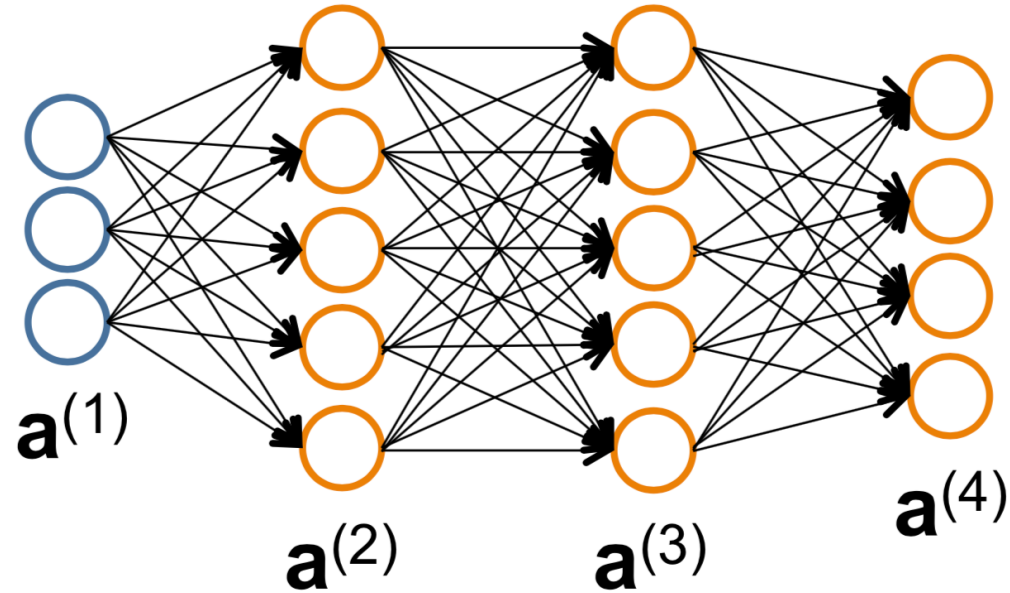
Computation Model

- Given access to a set of differentiable real functions $h \in \mathcal{H}$
 - Use functions in $\mathcal{H}$ to create intermediate variables.
 - Evaluation trace:
 - All intermediate variables will be scalars; each corresponds to a node.
 - Input $z_0 = w \in \mathbb{R}^d$. $[z_0]_1 = w_1, [z_0]_2 = w_2, \dots, [z_0]_d = w_d$
 - Step 1: $z_1 = h_1$ (a subset of variables in w)
 -
 - Step t: $z_t = h_t$ (a subset of variables in $z_1, \dots, z_{t-1}, w$)
 - ...
 - Step T: $z_T = h_T$ (a subset of variables in $z_1, \dots, z_{T-1}, w$)
 - **Return:** z_T
- $(h_1, \dots, h_T \in \mathcal{H})$

Computation Model

- Every $h \in \mathcal{H}$ is one of the following:
 - Type 1: An affine transformation of the inputs
 - Type 2: A product of variables, to some power
 - Type 3: A fixed set of one dimensional differentiable functions: $\sin(\cdot)$, $\cos(\cdot)$, $\exp(\cdot)$, $\log(\cdot)$, $\dots$
 - We assume we can easily compute the derivatives for each of this functions.
 - Type 3 can be approximated by Type 1 and Type 2, using polynomials.

Neural Network Example



Reverse Mode of Automatic Differentiation

Goal: Compute partial derivatives of $f(w)$, i.e., df/dw .

- Step 1: compute $f(w)$ and store in memory all intermediate variables $z_1, \dots, z_T$

- Step 2: Initialize: $\frac{dz_T}{dz_T} = 1$.

- Step 3: For $t = T, T-1, \dots, 0$

- $\frac{dz_T}{dz_t} = \sum_{\text{c is a child of t}} \frac{dz_T}{dz_c} \cdot \frac{\partial z_c}{\partial z_t}$

(Child: a node z_t directly points to)

- Step 4: Return $\frac{dz_T}{dz_0} = \frac{df}{dw}$

Time Complexity

Theorem (Baur and Strassen '83, Griewak '89): Assume every h is specified as in our computational model. For $h(\cdot)$ of type 3, assume we can compute the derivative $h'(z)$ in time as the same order of computing $h(z)$. Let T denote the time to compute $f(w)$. Then the reverse mode computes df/dw in time $O(T)$.

Time Complexity

Clarke Differential



Subdifferential and Subgradient

Definition: Given $f : \mathbb{R}^d \rightarrow \mathbb{R}$, for every x , the subdifferential set is defined as

$\partial_s f(x) \triangleq \{s \in \mathbb{R}^d : \forall x' \in \mathbb{R}^d, f(x') \geq f(x) + s^\top (x' - x)\}$. The elements in the subdifferential set are subgradients.

Subdifferential and Subgradient

Definition: Given $f : \mathbb{R}^d \rightarrow \mathbb{R}$, for every x , the subdifferential set is defined as

$\partial_s f(x) \triangleq \{s \in \mathbb{R}^d : \forall x' \in \mathbb{R}^d, f(x') \geq f(x) + s^\top (x' - x)\}$. The elements in the subdifferential set are subgradients.

Subdifferential is not enough

Definition: Given $f : \mathbb{R}^d \rightarrow \mathbb{R}$, for every x , the subdifferential set is defined as

$\partial_s f(x) \triangleq \{s \in \mathbb{R}^d : \forall x' \in \mathbb{R}^d, f(x') \geq f(x) + s^\top (x' - x)\}$. The elements in the subdifferential set are subgradients.

Clarke Differential

Definition: Given $f : \mathbb{R}^d \rightarrow \mathbb{R}$, for every x , the Clark differential is defined as

$$\partial f(x) \triangleq \text{conv} \left(\{s \in \mathbb{R}^d : \exists \{x_i\}_{i=1}^{\infty} \rightarrow x, \{ \nabla f(x_i) \}_{i=1}^{\infty} \rightarrow s\} \right).$$

The elements in the subdifferential set are subgradients.

When does Clarke differential exists

Definition (Locally Lipschitz): $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is locally Lipschitz if $\forall x \in \mathbb{R}^d$, there exists a neighborhood S of x , such that f is Lipschitz in S .

Optimization Methods for Deep Learning



Gradient descent for non-convex optimization

Decsent Lemma: Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be twice differentiable, and $\|\nabla^2 f\|_2 \leq \beta$. Then setting the learning rate $\eta = 1/\beta$, and applying gradient descent, $x_{t+1} = x_t - \eta \nabla f(x_t)$, we have:

$$f(x_t) - f(x_{t+1}) \geq \frac{1}{2\beta} \|\nabla f(x_t)\|_2^2.$$

Converging to stationary points

Theorem: In $T = O(\frac{\beta}{\epsilon^2})$ iterations, we have $\|\nabla f(x)\|_2 \leq \epsilon$.

Gradient Descent for Quadratic Functions

Problem: $\min_x \frac{1}{2} x^\top A x$ with $A \in \mathbb{R}^{d \times d}$ being positive-definite.

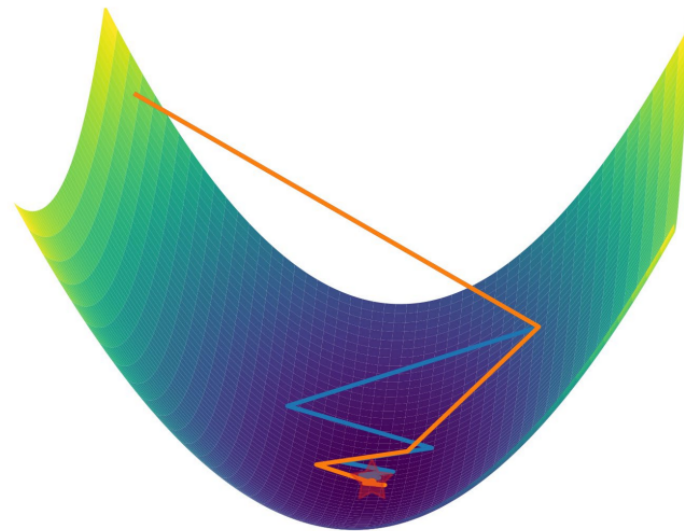
Theorem: Let $\lambda_{\max}$ and $\lambda_{\min}$ be the largest and the smallest eigenvalues of A . If we set $\eta \leq \frac{1}{\lambda_{\max}}$, we have

$$\|x_t\|_2 \leq (1 - \eta \lambda_{\min})^t \|x_0\|_2$$

Momentum: Heavy-Ball Method (Polyak '64)

Problem: $\min_x f(x)$

Method: $v_{t+1} = -\nabla f(x_t) + \beta v_t$
 $x_{t+1} = x_t + \eta v_{t+1}$

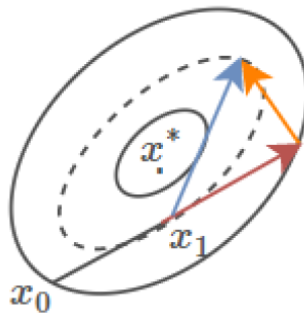


Momentum: Nesterov Acceleration (Nesterov '89)

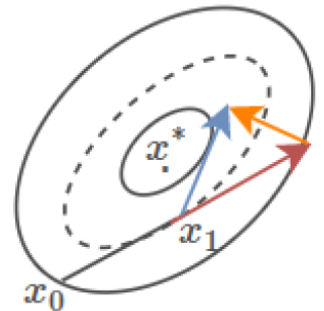
Problem: $\min_x f(x)$

Method: $v_{t+1} = -\nabla f(x_t + \beta v_t) + \beta v_t$
 $x_{t+1} = x_t + \eta v_{t+1}$

Polyak's Momentum

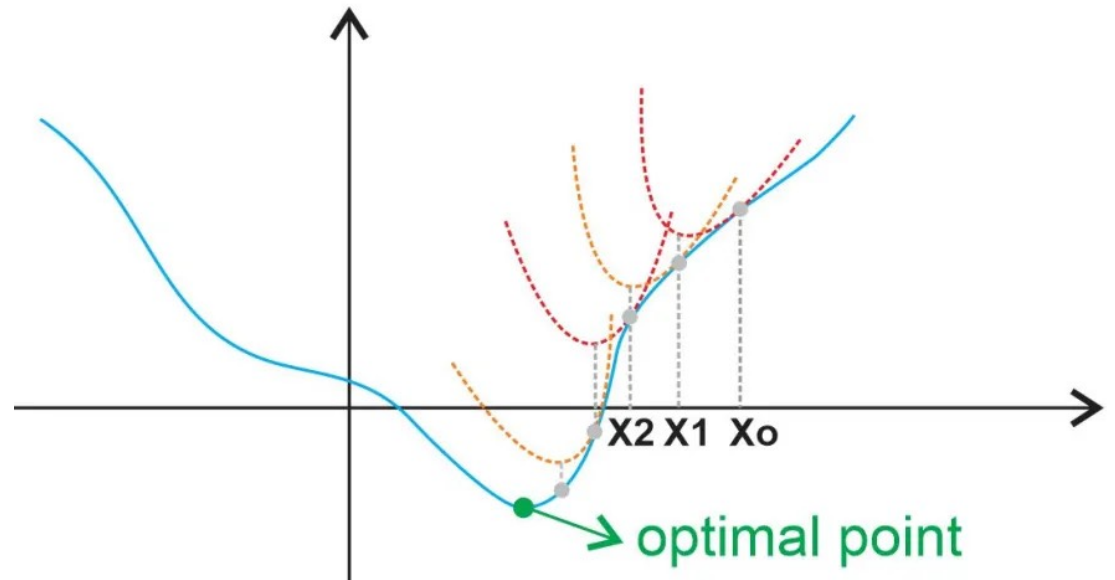


Nesterov Momentum



Newton's Method

Newton's Method: $x_{t+1} = x_t - \eta (\nabla^2 f(x_t))^{-1} \nabla f(x_t)$



AdaGrad (Duchi et al. '11)

Newton Method: $x_{t+1} = x_t - \eta (\nabla^2 f(x_t))^{-1} \nabla f(x_t)$

AdaGrad: separate learning rate for every parameter

$$x_{t+1} = x_t - \eta (G_{t+1} + \epsilon I)^{-1} \nabla f(x_t), (G_t)_{ii} = \sqrt{\sum_{j=1}^{t-1} (\nabla f(x_t)_i)^2}$$

RMSProp (Hinton et al. '12)

AdaGrad: separate learning rate for every parameter

$$x_{t+1} = x_t - \eta(G_{t+1} + \epsilon I)^{-1} \nabla f(x_t), (G_t)_{ii} = \sqrt{\sum_{j=1}^{t-1} (\nabla f(x_t)_i)^2}$$

RMSProp: exponential weighting of gradient norms

$$x_{t+1} = x_t - \eta(G_{t+1} + \epsilon I)^{-1/2} \nabla f(x_t), \\ (G_{t+1})_{ii} = \beta(G_t)_{ii} + (1 - \beta)(\nabla f(x_t)_i)^2$$

AdaDelta (Zeiler '12)

RMSProp:

$$x_{t+1} = x_t - \eta(G_{t+1} + \epsilon I)^{-1/2} \nabla f(x_t),$$
$$(G_{t+1})_{ii} = \beta(G_t)_{ii} + (1 - \beta)(\nabla f(x_t)_i)^2$$

AdaDelta:

$$x_{t+1} = x_t - \eta \Delta x_t,$$
$$\Delta x_t = \sqrt{u_t + \epsilon} \cdot (G_{t+1} + \epsilon I)^{-1/2} \nabla f(x_t)$$
$$(G_{t+1})_{ii} = \rho(G_t)_{ii} + (1 - \rho)(\nabla f(x_t)_i)^2,$$
$$u_{t+1} = \rho u_t + (1 - \rho) \|\Delta x_t\|_2^2$$

Adam (Kingma & Ba '14)

Momentum:

$$v_{t+1} = -\nabla f(x_t) + \beta v_t, \quad x_{t+1} = x_t + \eta v_{t+1}$$

RMSProp: exponential weighting of gradient norms

$$x_{t+1} = x_t - \eta (G_{t+1} + \epsilon I)^{-1} \nabla f(x_t),$$
$$(G_t)_{ii} = \beta (G_t)_{ii} + (1 - \beta) (\nabla f(x_t)_i)^2$$

Adam

$$v_{t+1} = \beta_1 v_t + (1 - \beta_1) \nabla f(x_t)$$
$$(G_{t+1})_{ii} = \beta_2 (G_t)_{ii} + (1 - \beta_2) (\nabla f(x_t)_i)^2$$
$$x_{t+1} = x_t - \eta (G_{t+1} + \epsilon I)^{-1/2} v_{t+1}$$

Default choice nowadays.

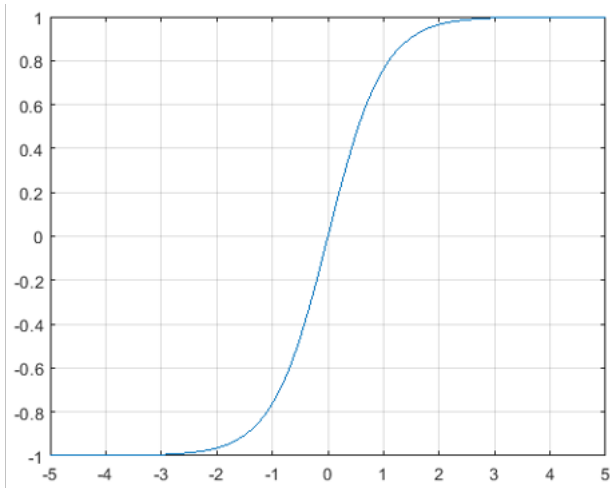
Important Techniques in Neural Network Training



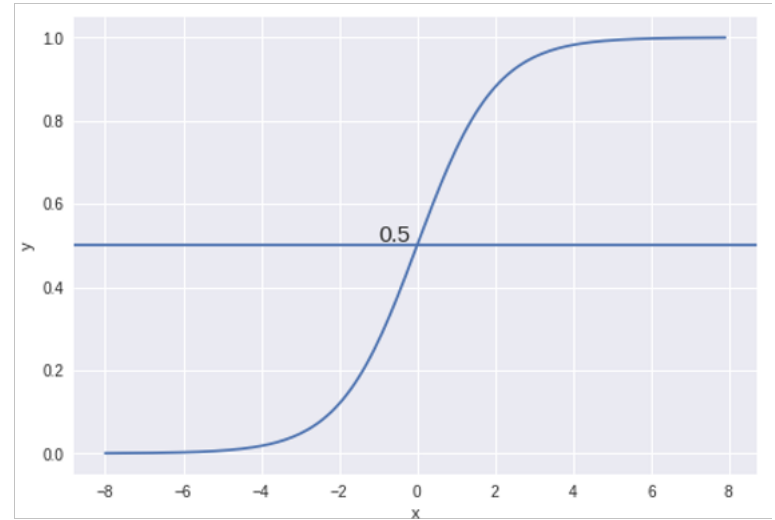
Gradient Explosion / Vanishing

- Deeper networks are harder to train:
 - Intuition: gradients are products over layers
 - Hard to control the learning rate

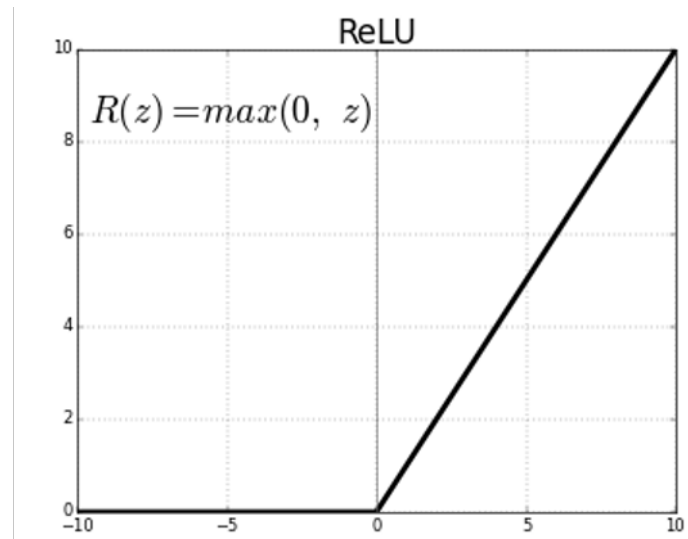
Activation Functions



tanh



sigmoid

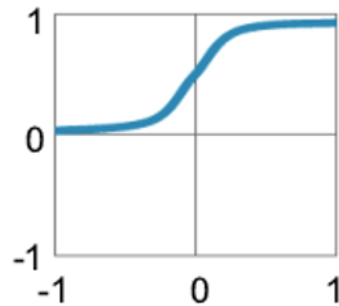


Rectified Linear Unit

Activation Function

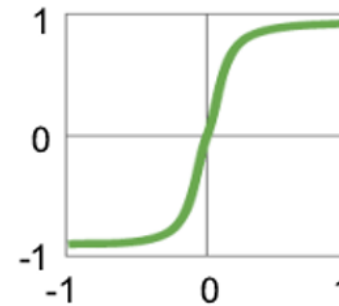
Traditional Non-Linear Activation Functions

Sigmoid



$$y = 1 / (1 + e^{-x})$$

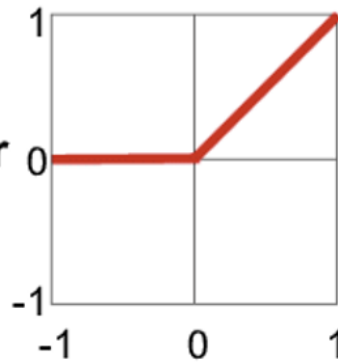
Hyperbolic Tangent



$$y = (e^x - e^{-x}) / (e^x + e^{-x})$$

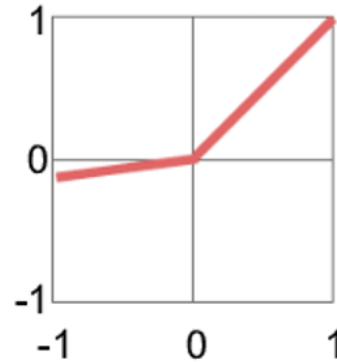
Modern Non-Linear Activation Functions

Rectified Linear Unit (ReLU)



$$y = \max(0, x)$$

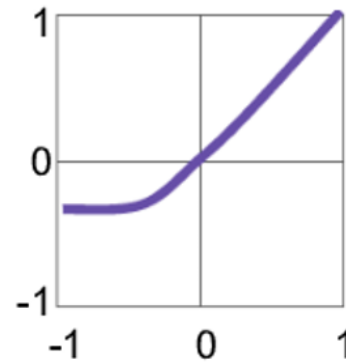
Leaky ReLU



$$y = \max(\alpha x, x)$$

α = small const. (e.g. 0.1)

Exponential LU



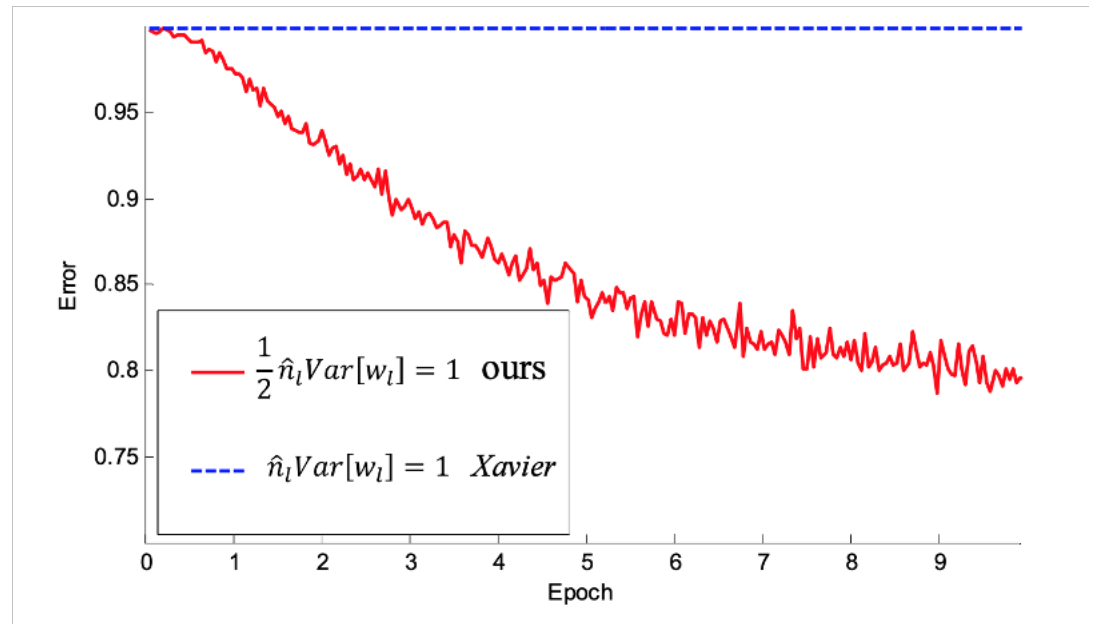
$$y = \begin{cases} x, & x \geq 0 \\ \alpha(e^x - 1), & x < 0 \end{cases}$$

Initialization

- Zero-initialization
- Large initialization
- Small initialization
- Design principles:
 - Zero activation mean
 - Activation variance remains same across layers

Kaiming Initialization (He et al. '15)

- $W_{ij}^{(h)} \sim \mathcal{N}\left(0, \frac{2}{d_h}\right)$.
- $b^{(h)} = 0$
- Designed for ReLU activation
- 30-layer neural network



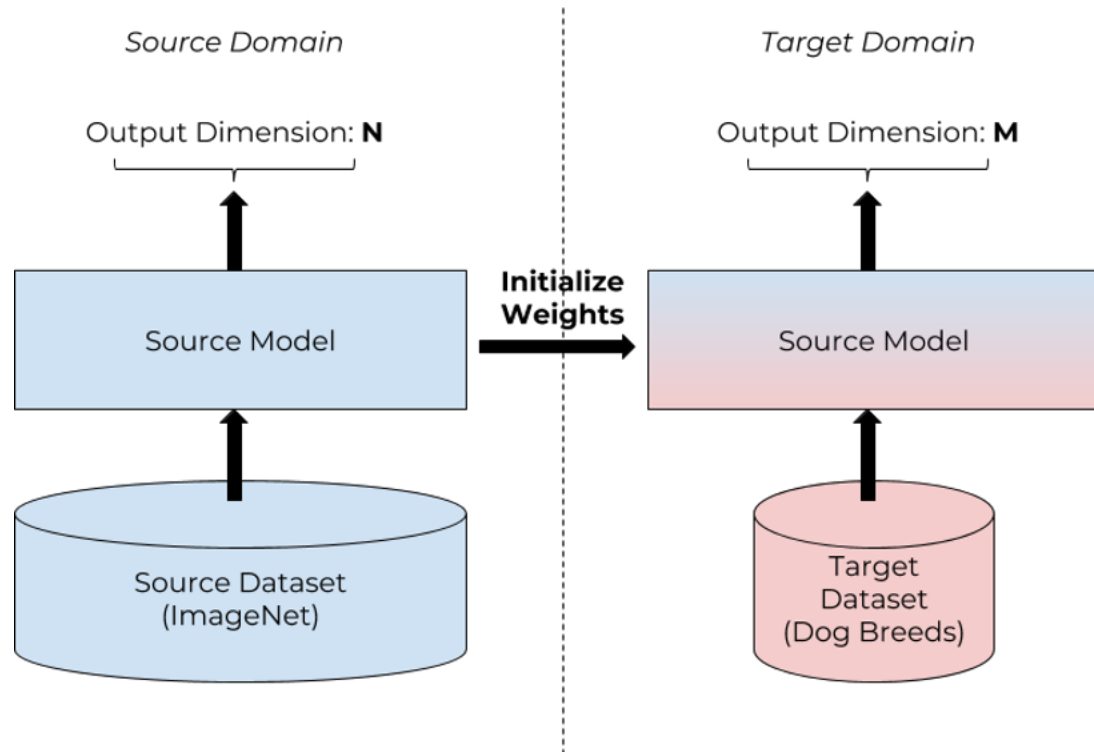
Kaiming Initialization (He et al. '15)

Kaiming Initialization (He et al. '15)

Kaiming Initialization (He et al. '15)

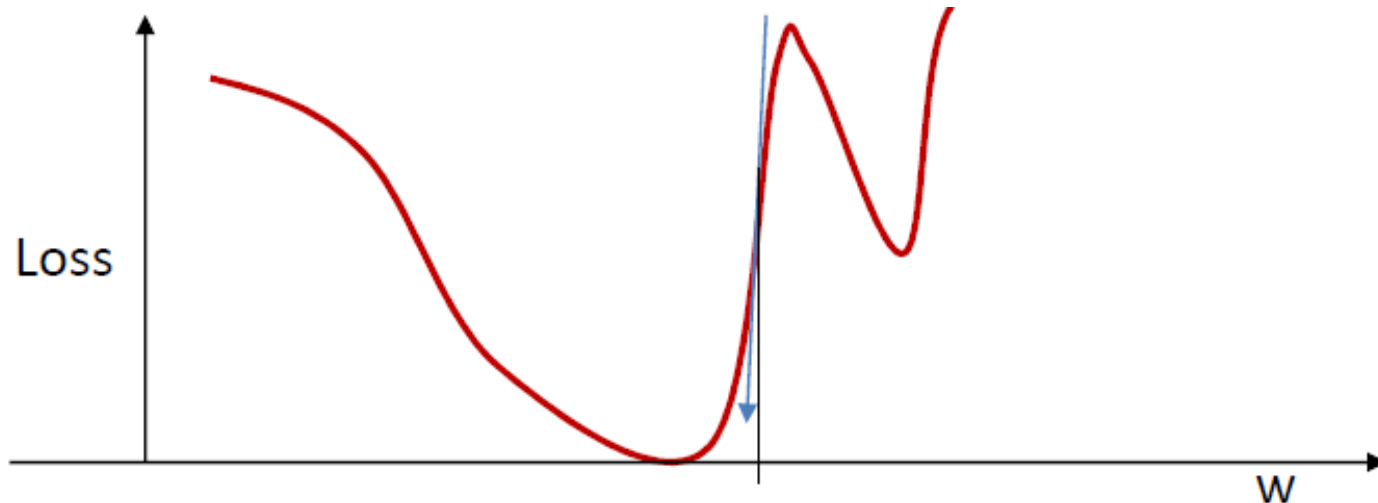
Initialization by Pre-training

- Use a pre-trained network as initialization
- And then fine-tuning



Gradient Clipping

- The loss can occasionally lead to a steep descent
- This result in immediate instability
- If gradient norm bigger than a threshold, set the gradient to the threshold.

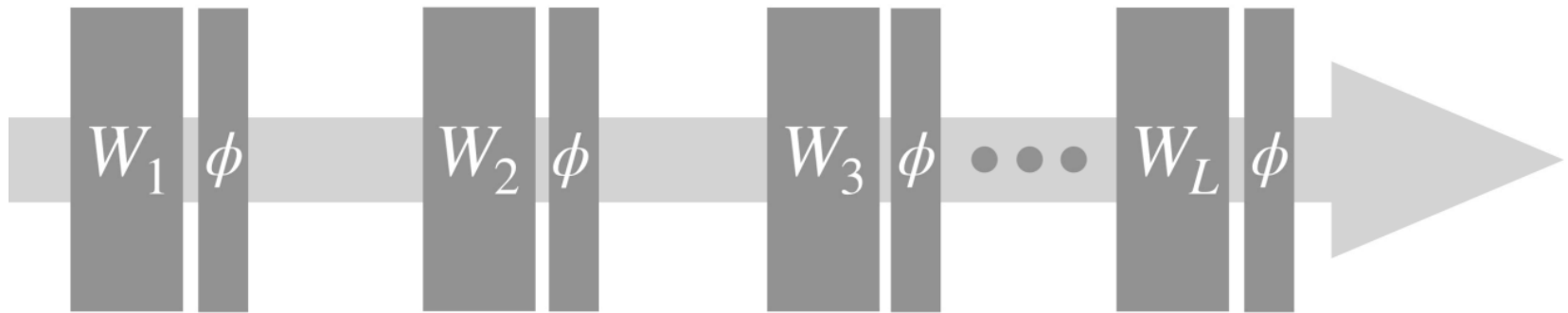


Batch Normalization (Ioffe & Szegedy, '14)

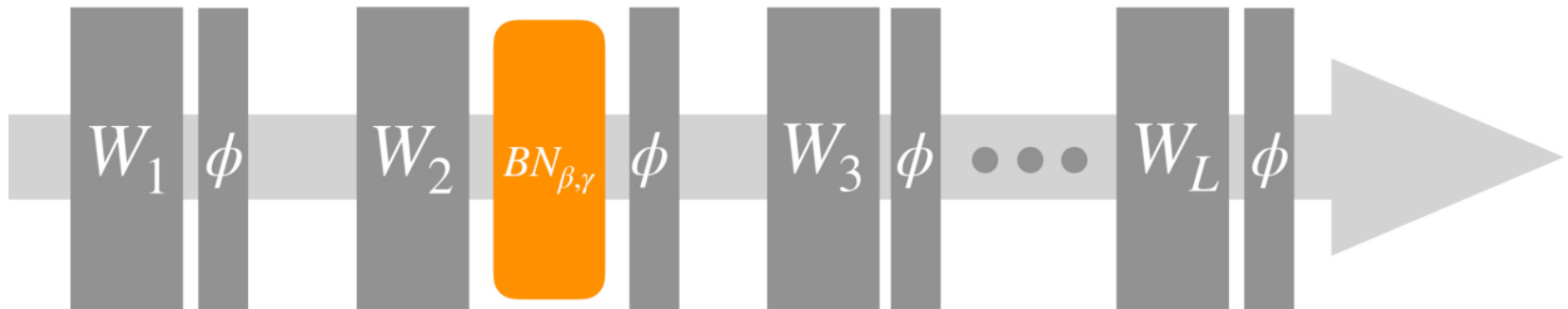
- **Normalizing/whitening** (mean = 0, variance = 1) the inputs is generally useful in machine learning.
 - Could normalization be useful at the level of hidden layers?
 - **Internal covariate shift**: the calculations of the neural networks change the distribution in hidden layers even if the inputs are normalized
- **Batch normalization** is an attempt to do that:
 - Each unit's **pre-activation** is normalized (mean subtraction, std division)
 - During training, mean and std is computed for each minibatch (can be backproped!)

Batch Normalization (Ioffe & Szegedy, '14)

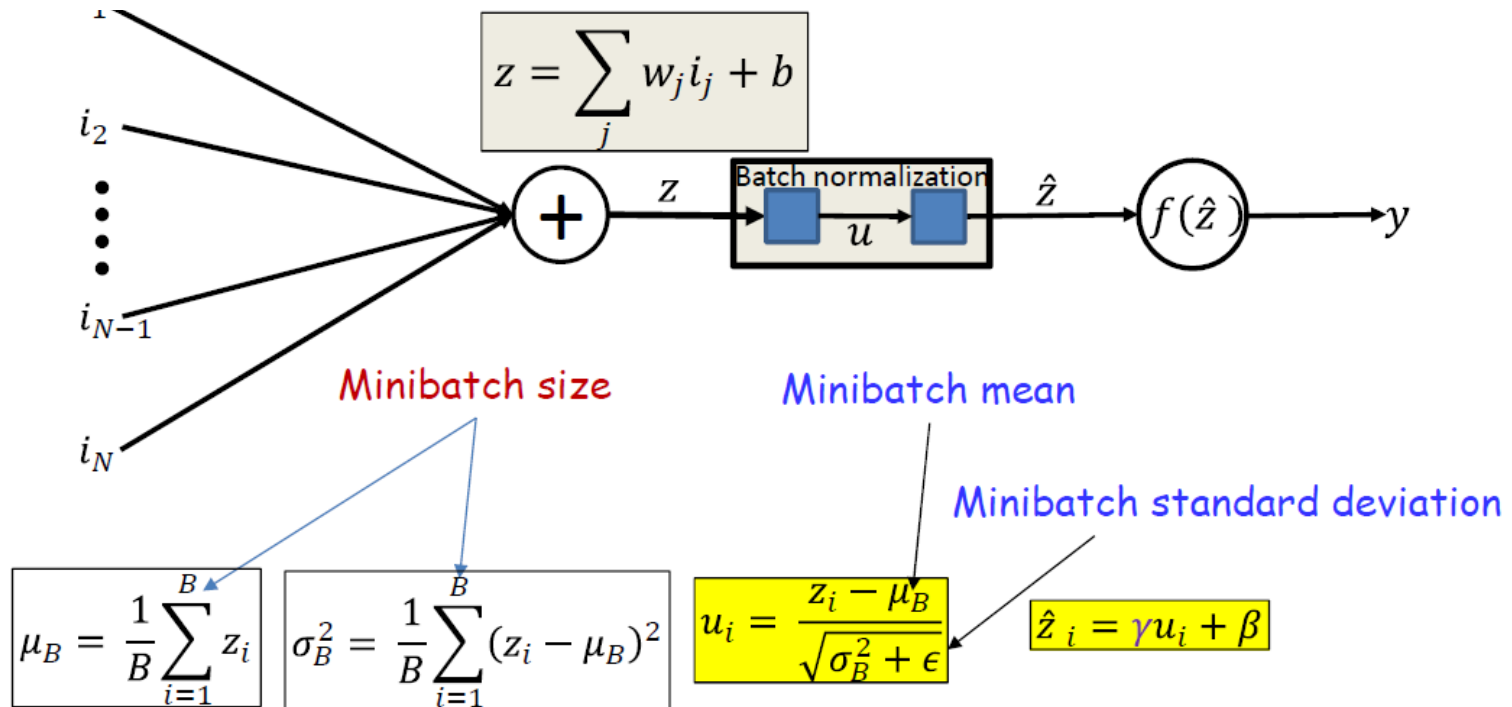
Standard Network



Adding a BatchNorm layer (between weights and activation function)



Batch Normalization (Ioffe & Szegedy, '14)

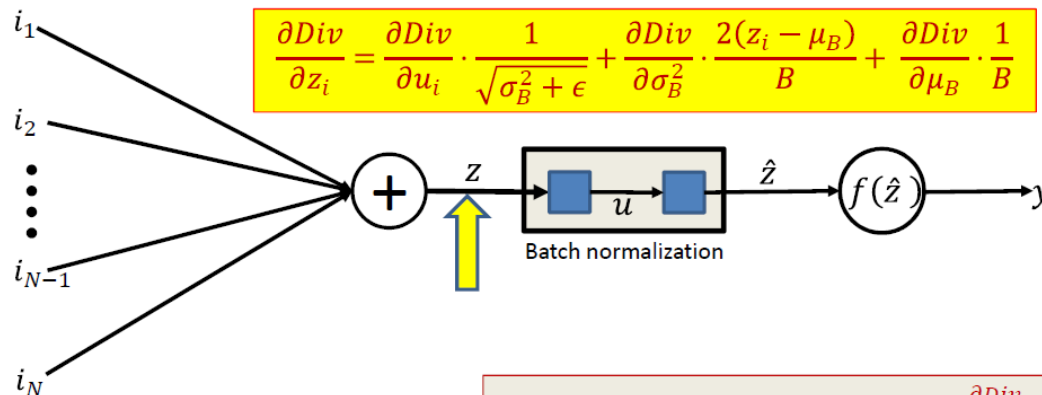


Batch Normalization (Ioffe & Szegedy, '14)

- BatchNorm at training time
 - Standard backprop performed for each single training data
 - Now backprop is performed over entire batch.

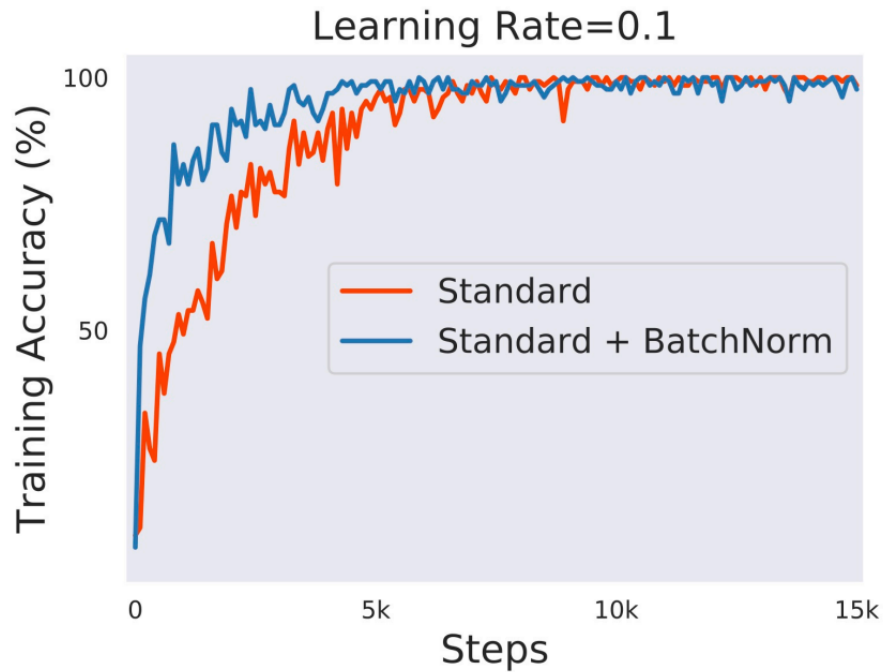
$$\frac{\partial Div}{\partial \sigma_B^2} = \frac{-1}{2} (\sigma_B^2 + \epsilon)^{-3/2} \sum_{i=1}^B \frac{\partial Div}{\partial u_i}$$

$$\frac{\partial Div}{\partial \mu_B} = \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} \sum_{i=1}^B \frac{\partial Div}{\partial u_i}$$



The rest of backprop continues from $\frac{\partial Div}{\partial z_i}$

Batch Normalization (Ioffe & Szegedy, '14)



What is BatchNorm actually doing?

- May not due to covariate shift (Santurkar et al. '18):
 - Inject non-zero mean, non-standard covariance Gaussian noise after BN layer: removes the whitening effect
 - Still performs well.
- Only training β, γ with random convolution kernels gives non-trivial performance (Frankle et al. '20)
- BN can use exponentially increasing learning rate! (Li & Arora '19)

More normalizations

- Layer normalization (Ba, Kiros, Hinton, '16)
 - Batch-independent
 - Suitable for RNN, MLP
- Weight normalization (Salimans, Kingma, '16)
 - Suitable for meta-learning (higher order gradients are needed)
- Instance normalization (Ulyanov, Vedaldi, Lempitsky, '16)
 - Batch-independent, suitable for generation tasks
- Group normalization (Wu & He, '18)
 - Batch-independent, improve BatchNorm for small batch size