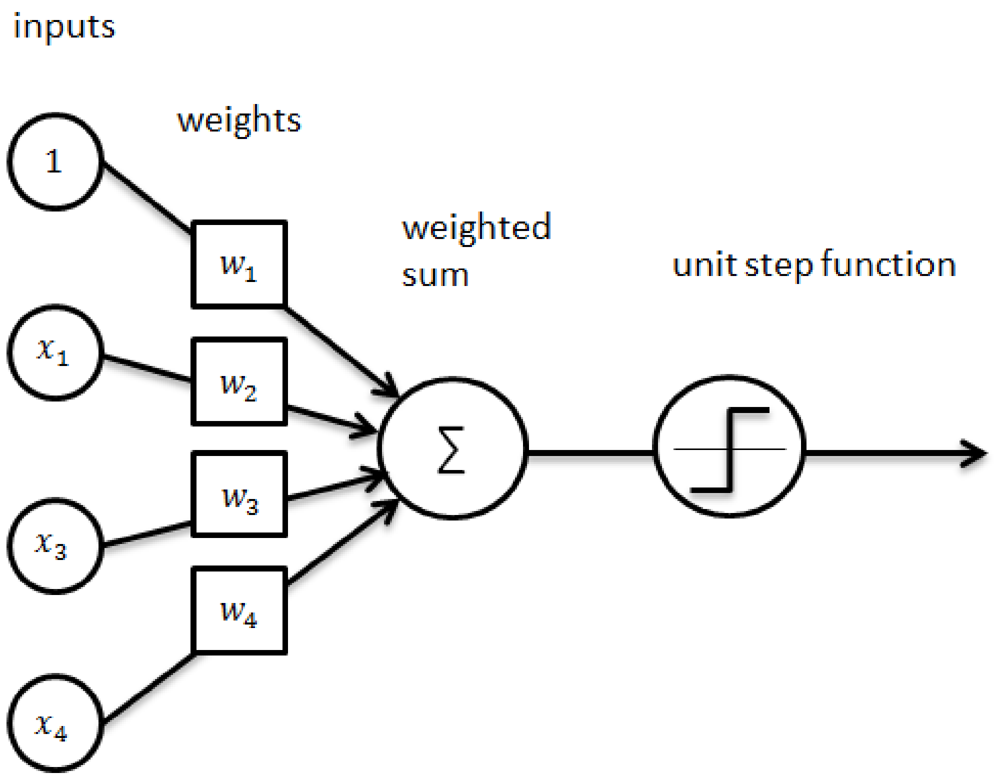
Introduction to Computer Vision

Kaveh Fathian
Assistant Professor
Computer Science Department
Colorado School of Mines

Lecture 19

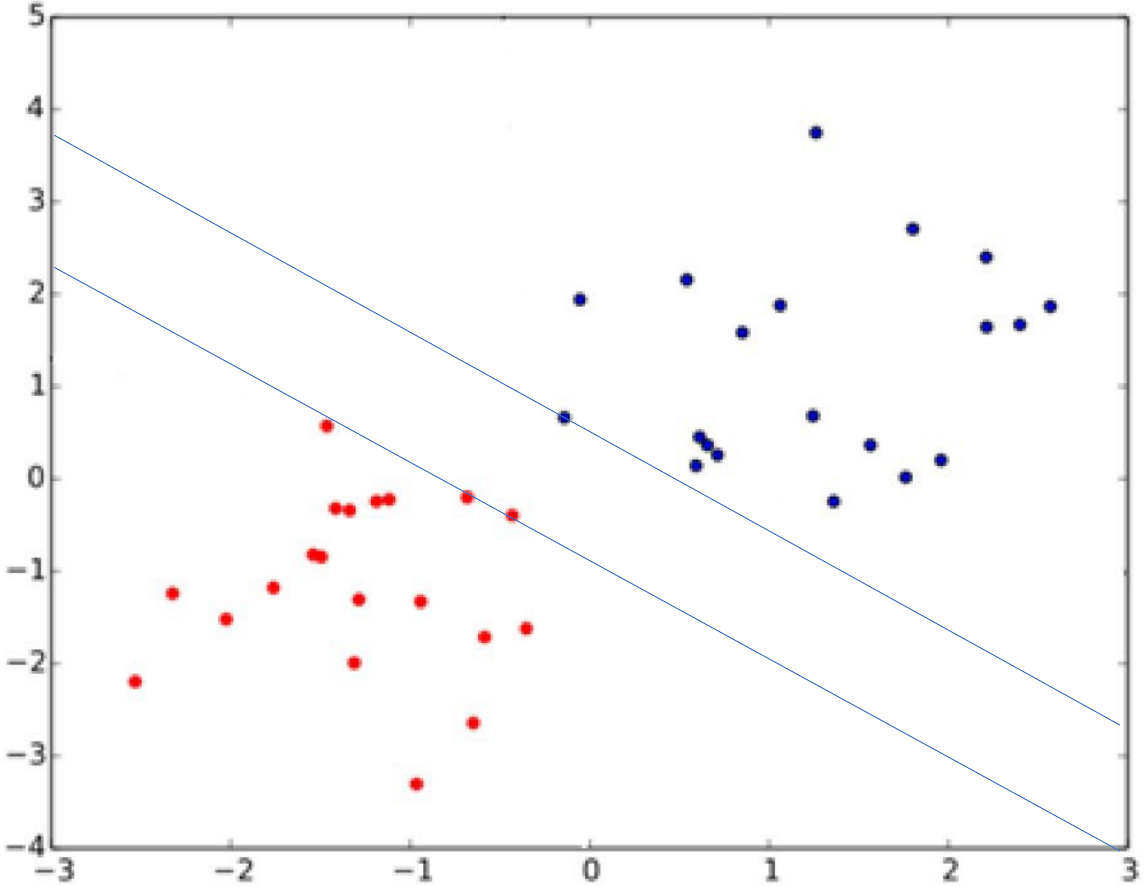
Training Perceptrons



SVMs vs. Perceptrons

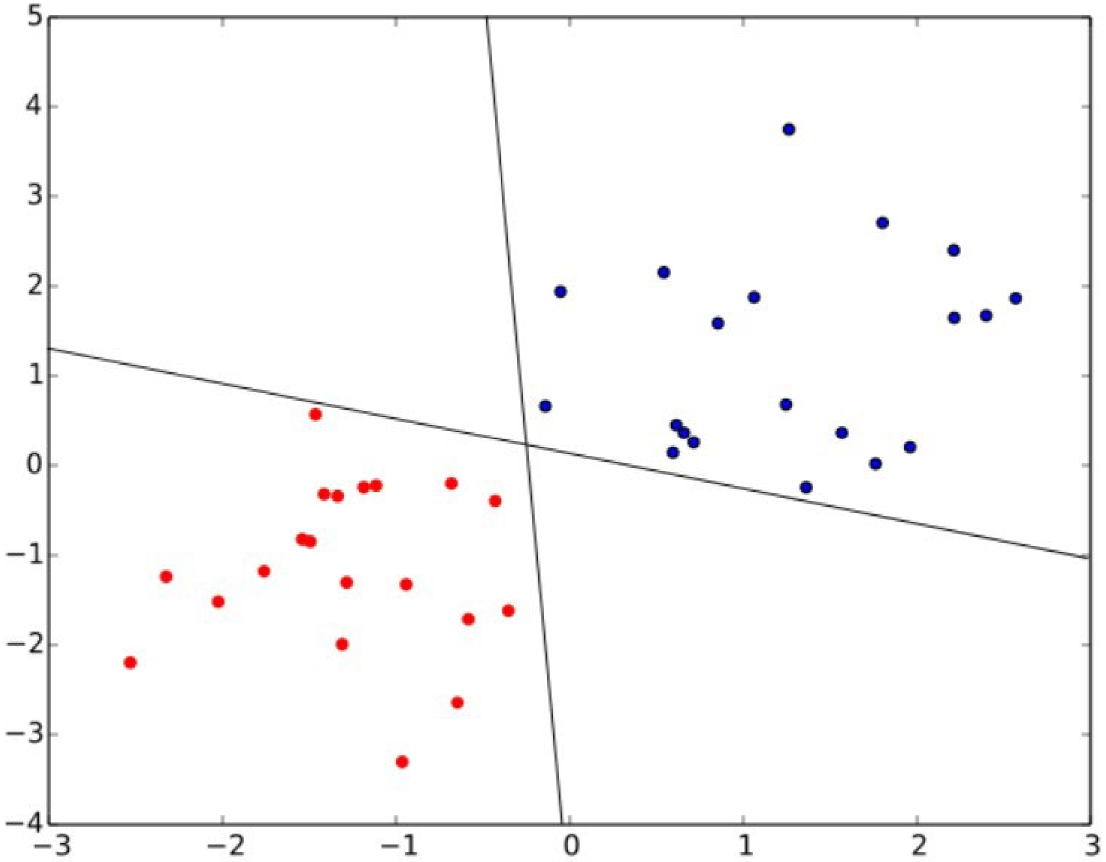
- What is the relation between **SVMs** & **perceptrons**?

- **SVMs** attempt to learn the support vectors which **maximize the margin between classes**



SVMs vs. Perceptrons

- What is the relation between **SVMs** & **perceptrons**?
- **SVMs** attempt to learn the support vectors which **maximize the margin between classes**
- A **perceptron does not**



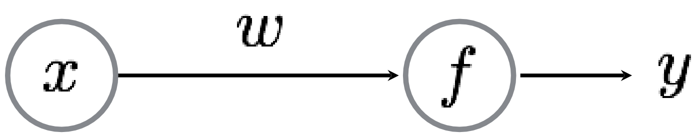
World’s Smallest Perceptron!

$$x \xrightarrow{w} f \longrightarrow y$$

$$y = wx$$

What does this look like?

World's Smallest Perceptron!



$$y = wx$$

(a.k.a. line equation, linear regression)

Learning a Perceptron

Given a set of samples and a Perceptron

$$\{x_i, y_i\}$$

$$y = f_{\text{PER}}(x; w)$$

Estimate the parameters of the Perceptron

$$w$$

Given training data:

x	y
10	10.1
2	1.9
3.5	3.4
1	1.1

What do you think the weight parameter is?

$$y = wx$$

Given training data:

x	y
10	10.1
2	1.9
3.5	3.4
1	1.1

What do you think the weight parameter is?

not so obvious as the network gets more complicate!

$$y = wx$$

An Incremental Learning Strategy
(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

An Incremental Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

Modify weight w such that $\hat{y}$ gets **'closer'** to y

An Incremental Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

Modify weight w such that $\hat{y}$ gets ‘**closer**’ to y

perceptron parameter

perceptron output

true label

An Incremental Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

Modify weight w such that $\hat{y}$ gets ‘**closer**’ to y

perceptron parameter

perceptron output

what does this mean?

true label

Before diving into gradient descent, we need to understand …

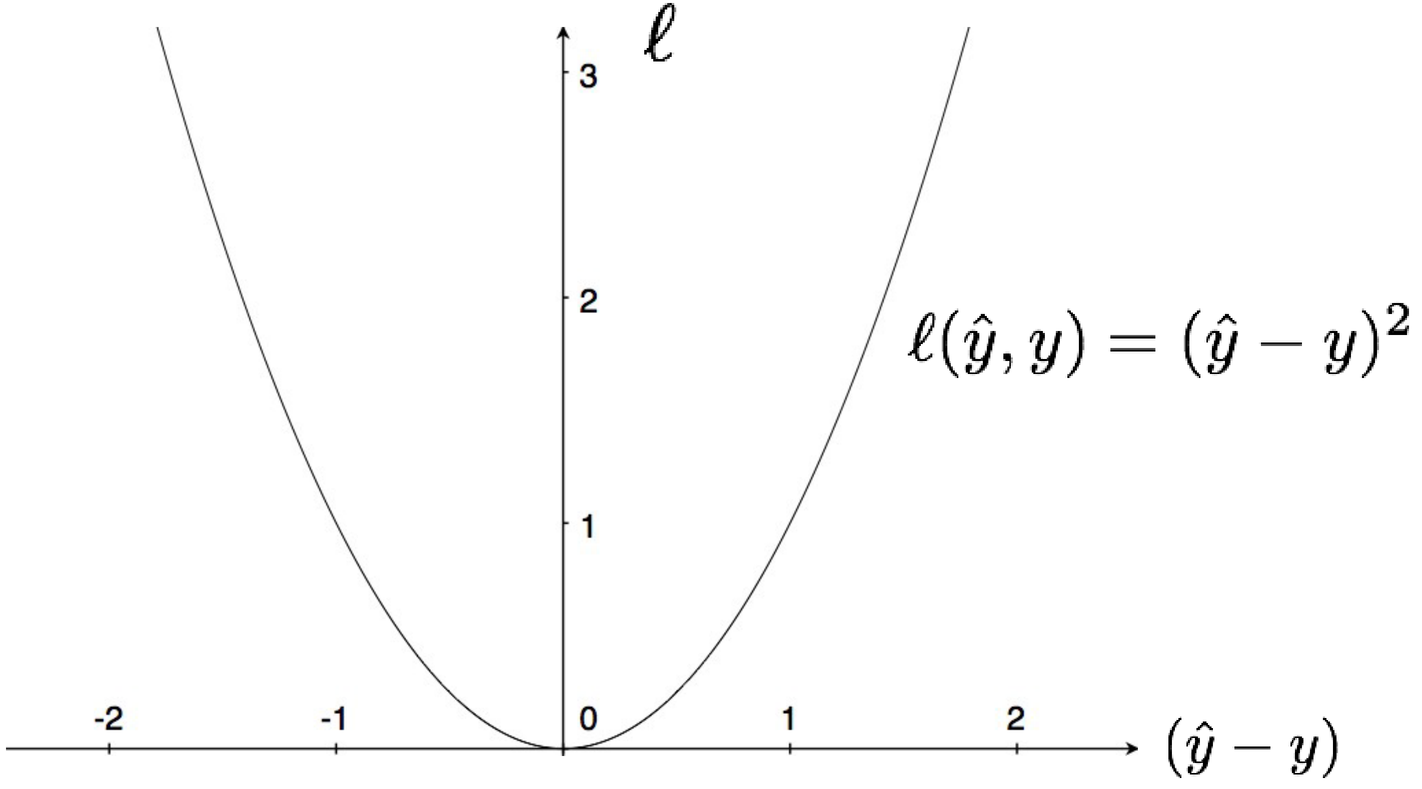
Loss Function
defines what is means to be
close to the true solution

YOU get to chose the loss function!
(some are better than others depending on what you want to do)

Loss Function

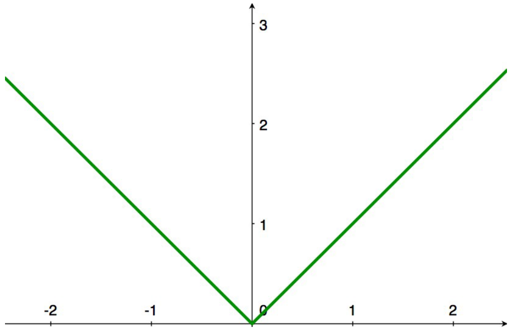
Squared Error (L2)

(a popular loss function)



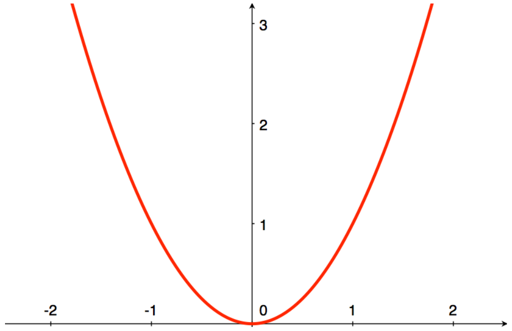
L1 Loss

$$\ell(\hat{y}, y) = |\hat{y} - y|$$



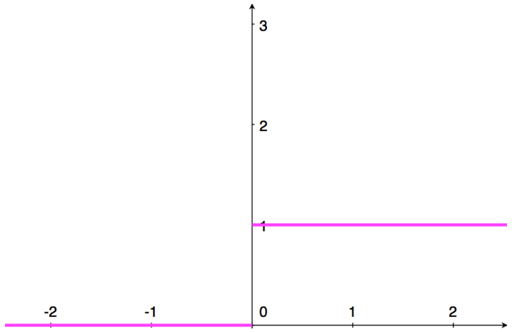
L2 Loss

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



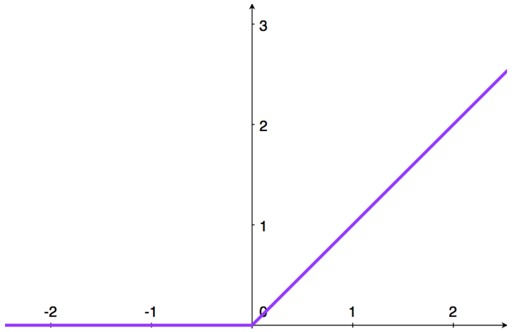
Zero-One Loss

$$\ell(\hat{y}, y) = \mathbf{1}[\hat{y} = y]$$



Hinge Loss

$$\ell(\hat{y}, y) = \max(0, 1 - y \cdot \hat{y})$$



back to the…

World’s Smallest Perceptron!

$$x \xrightarrow{\;w\;} f \longrightarrow y$$

$$y = wx$$

(a.k.a. line equation, linear regression)

function of **ONE** parameter!

Learning a Perceptron

Given a set of samples and a Perceptron

$$\{x_i, y_i\}$$

$$y = f_{\text{PER}}(x; w)$$

what is this activation function?

Estimate the parameter of the Perceptron

$$w$$

Learning a Perceptron

Given a set of samples and a Perceptron

$$\{x_i, y_i\}$$

$$y = f_{\mathrm{PER}}(x; w)$$

what is this
activation function?

linear function! $f(x) = wx$

Estimate the parameter of the Perceptron

$$w$$

Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

Modify weight w such that $\hat{y}$ gets ‘**closer**’ to y

perceptron parameter

perceptron output

true label

Let's demystify this process first…

Code to train your perceptron:

Let’s demystify this process first…

Code to train your perceptron:

$$\text{for } n = 1 \dots N$$
$$w = w + (y_n - \hat{y})x_i;$$

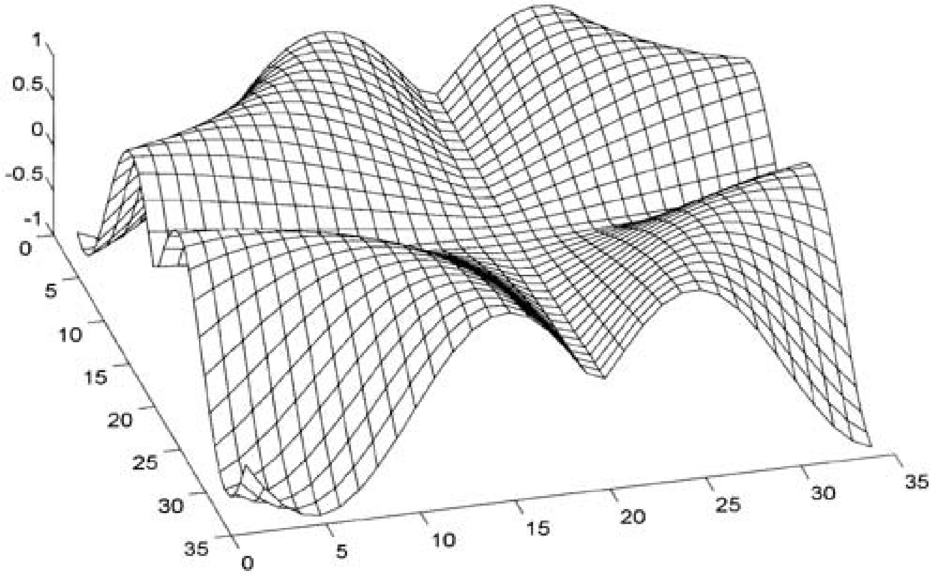
just one line of code!

Now where does this come from?

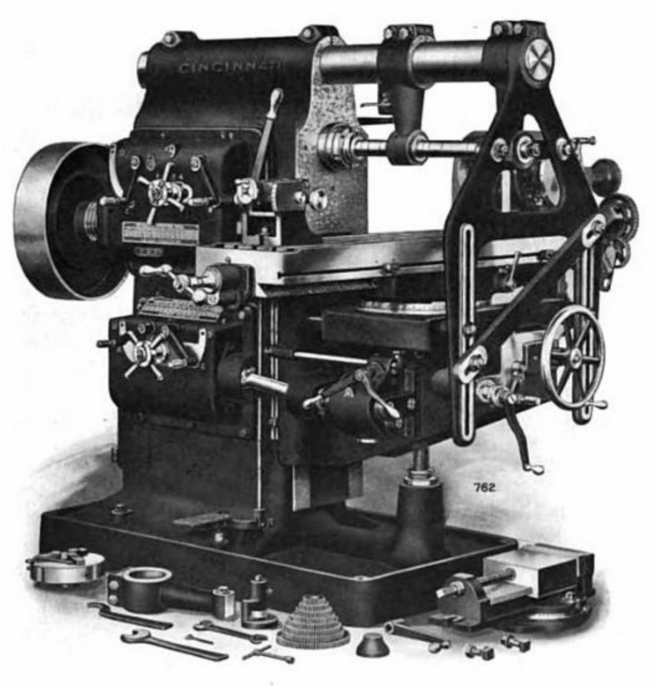
Gradient Descent

(partial) derivatives tell us how much one variable affects another

Two ways to think about them:

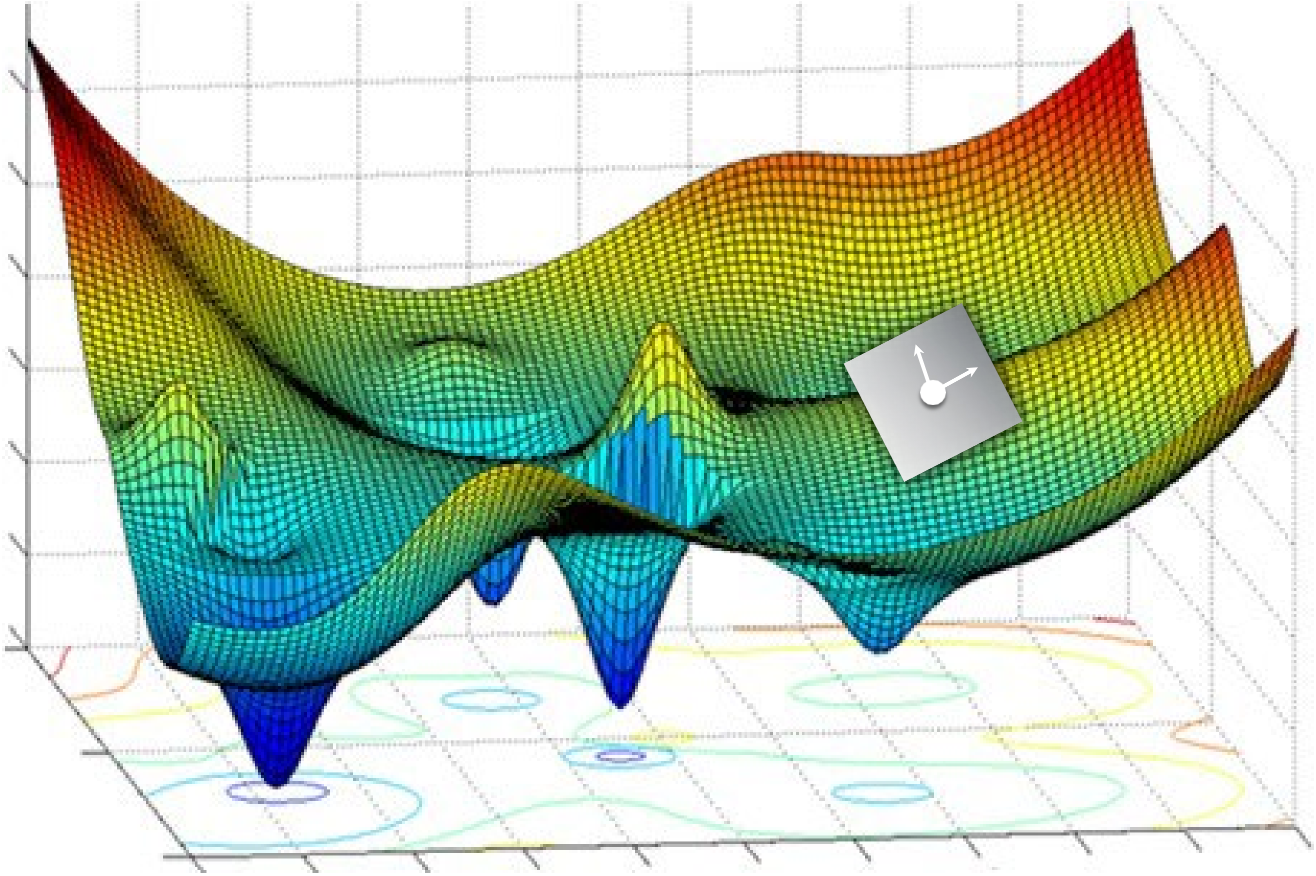


Slope of a function



Knobs on a machine

1. Slope of a function:



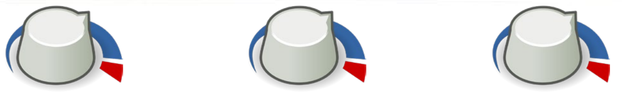
$$\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} = \left[\frac{\partial f(\boldsymbol{x})}{\partial x}, \frac{\partial f(\boldsymbol{x})}{\partial y}\right]$$

describes the slope around a point

2. Knobs on a machine:

input
$\boldsymbol{x}$

output
$f(\boldsymbol{x}; \boldsymbol{w})$

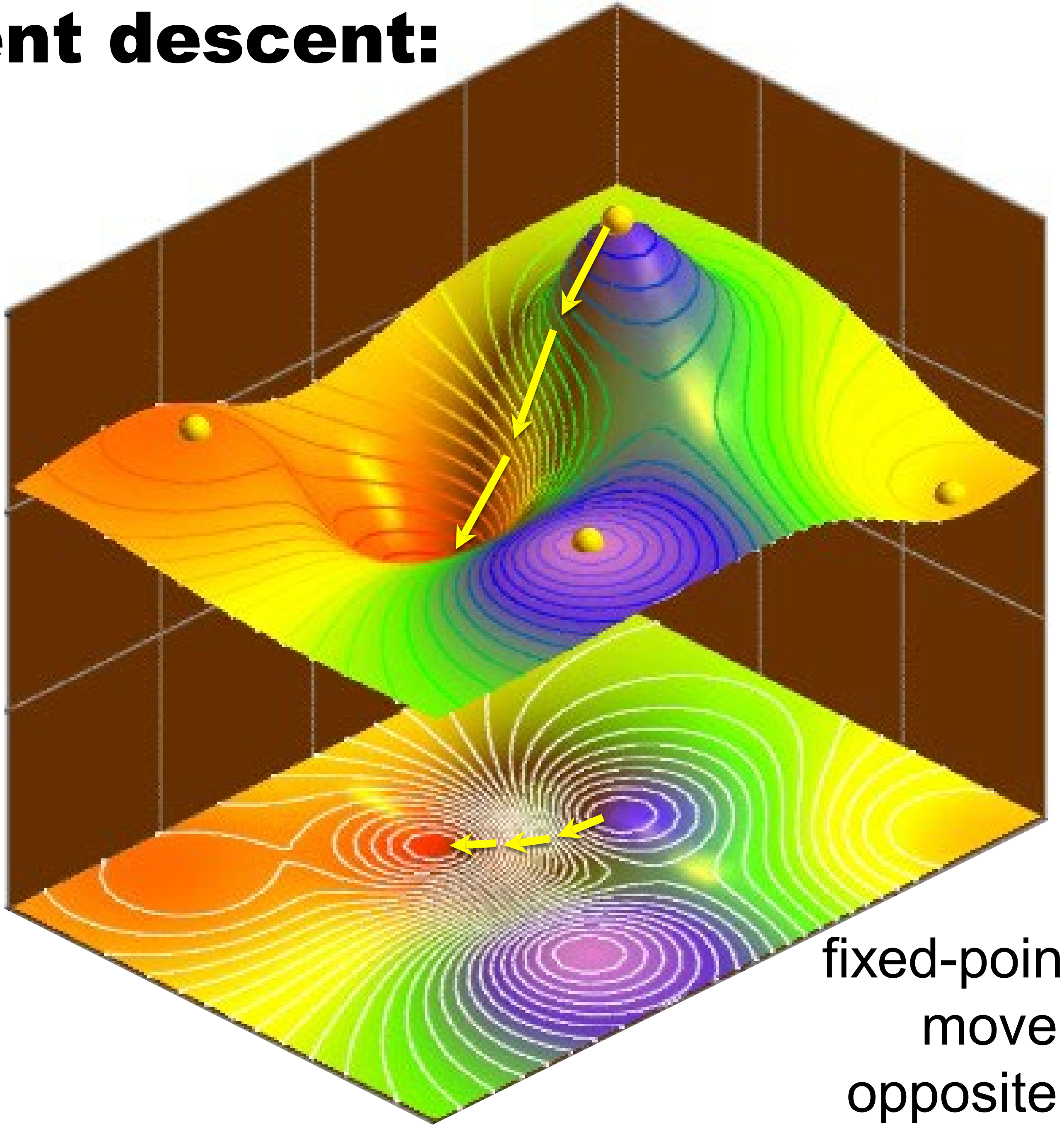


describes how each ‘knob’ affects the output

$$\frac{\partial f(x)}{\partial w_1} \qquad \frac{\partial f(x)}{\partial w_2} \qquad \frac{\partial f(x)}{\partial w_3}$$

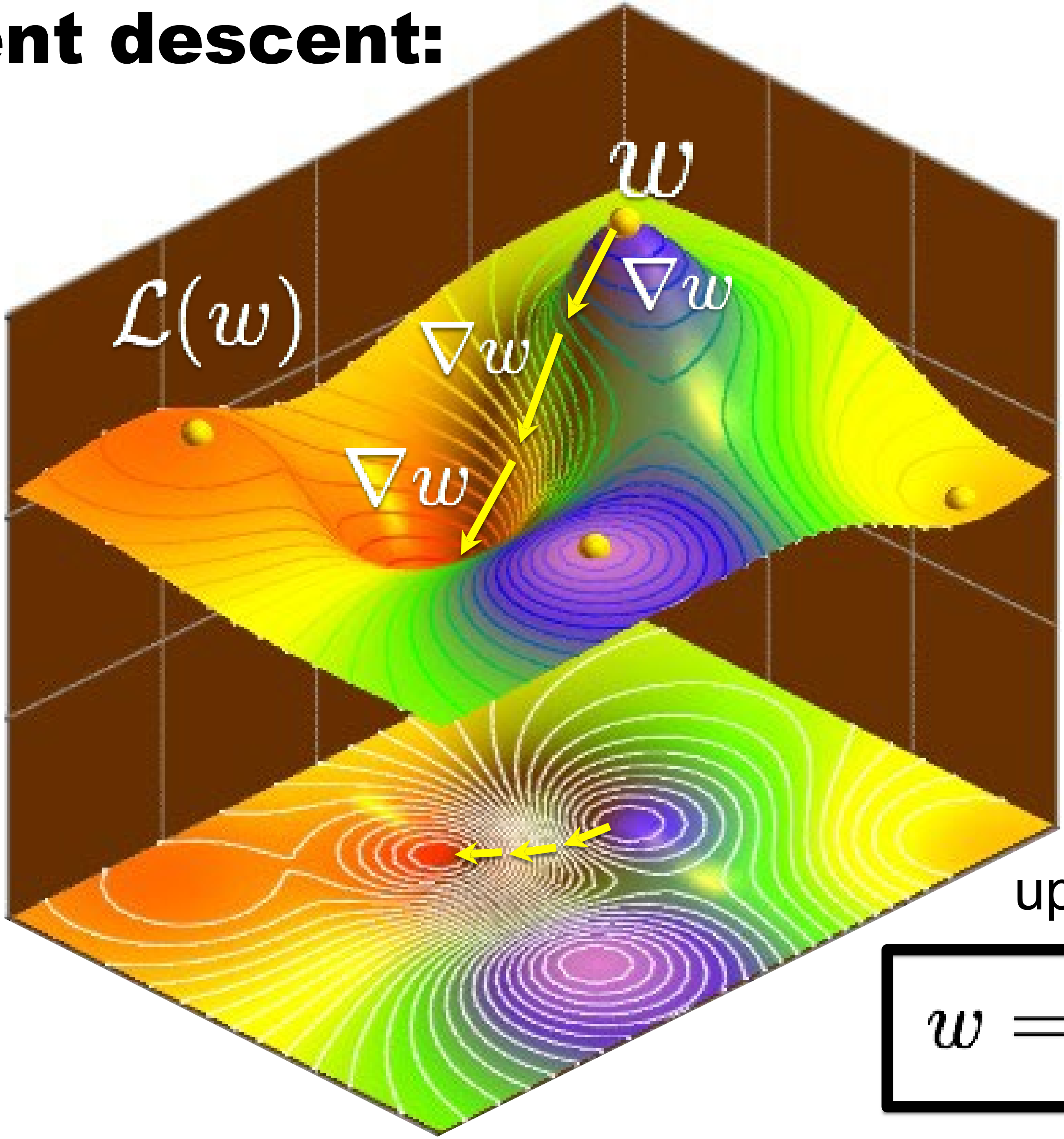
small change in parameter Δw_1 → output will change by $\dfrac{\partial f(x)}{\partial w_1} \Delta w_1$

Gradient descent:



Given a
fixed-point on a function,
move in the direction
opposite of the gradient

Gradient descent:



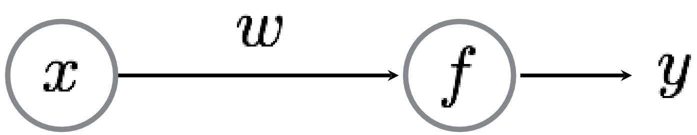
update rule:

$$\boxed{w = w - \nabla w}$$

Backpropagation

back to the…

World’s Smallest Perceptron!



$$y = wx$$

(a.k.a. line equation, linear regression)

function of **ONE** parameter!

Training the world's smallest perceptron

$$\text{for } n = 1 \ldots N$$

$$w = w + \underline{(y_n - \hat{y})x_i};$$

This is just gradient descent, that means…

this should be the gradient of the loss function

Now where does this come from?

$$\frac{d\mathcal{L}}{dw}$$

…is the rate at which this will change…

$$\mathcal{L} = \frac{1}{2}(y - \hat{y})^2$$

the loss function

… per unit change of **this**

$$y = wx$$

the weight parameter

Let's compute the derivative…

Compute the derivative

$$\frac{d\mathcal{L}}{dw} = \frac{d}{dw}\left\{\frac{1}{2}(y-\hat{y})^2\right\}$$

$$= -(y-\hat{y})\frac{dwx}{dw}$$

$$= -(y-\hat{y})x = \nabla w$$ just shorthand

That means the weight update for **gradient descent** is:

$$w = w - \nabla w$$ move in direction of negative gradient

$$= w + (y-\hat{y})x$$

Gradient Descent (world's smallest perceptron)

For each sample $\{x_i, y_i\}$

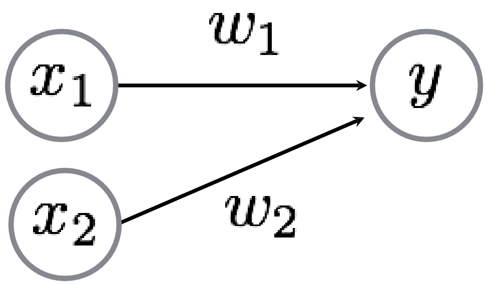
1. Predict
   a. Forward pass $\hat{y} = wx_i$
   b. Compute Loss $\mathcal{L}_i = \frac{1}{2}(y_i - \hat{y})^2$
2. Update
   a. Back Propagation $\frac{d\mathcal{L}_i}{dw} = -(y_i - \hat{y})x_i = \nabla w$
   b. Gradient update $w = w - \nabla w$

Training the world's smallest perceptron

$$\text{for } n = 1 \ldots N$$

$$w = w + (y_n - \hat{y})x_i;$$

World's (second) smallest **perceptron**!



function of **two** parameters!

Gradient Descent

For each sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass
   b. Compute Loss
2. Update
   a. Back Propagation
   b. Gradient update

we just need to compute partial derivatives for this network

Derivative computation

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_1} &= \frac{\partial}{\partial w_1}\left\{\frac{1}{2}(y-\hat{y})^2\right\} \\
&= -(y-\hat{y})\frac{\partial \hat{y}}{\partial w_1} \\
&= -(y-\hat{y})\frac{\partial \sum_i w_i x_i}{\partial w_1} \\
&= -(y-\hat{y})\frac{\partial w_1 x_1}{\partial w_1} \\
&= -(y-\hat{y})x_1 = \nabla w_1
\end{aligned}$$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_2} &= \frac{\partial}{\partial w_2}\left\{\frac{1}{2}(y-\hat{y})^2\right\} \\
&= -(y-\hat{y})\frac{\partial \hat{y}}{\partial w_2} \\
&= -(y-\hat{y})\frac{\partial \sum_i w_i x_i}{\partial w_1} \\
&= -(y-\hat{y})\frac{\partial w_2 x_2}{\partial w_2} \\
&= -(y-\hat{y})x_2 = \nabla w_2
\end{aligned}$$

Why do we have partial derivatives now?

Derivative computation

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial}{\partial w_1}\left\{\frac{1}{2}(y-\hat{y})^2\right\}$$
$$= -(y-\hat{y})\frac{\partial \hat{y}}{\partial w_1}$$
$$= -(y-\hat{y})\frac{\partial \sum_i w_i x_i}{\partial w_1}$$
$$= -(y-\hat{y})\frac{\partial w_1 x_1}{\partial w_1}$$
$$\boxed{= -(y-\hat{y})x_1 = \nabla w_1}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial}{\partial w_2}\left\{\frac{1}{2}(y-\hat{y})^2\right\}$$
$$= -(y-\hat{y})\frac{\partial \hat{y}}{\partial w_2}$$
$$= -(y-\hat{y})\frac{\partial \sum_i w_i x_i}{\partial w_1}$$
$$= -(y-\hat{y})\frac{\partial w_2 x_2}{\partial w_2}$$
$$\boxed{= -(y-\hat{y})x_2 = \nabla w_2}$$

Gradient Update

$$w_1 = w_1 - \eta \nabla w_1$$
$$\boxed{= w_1 + \eta(y-\hat{y})x_1}$$

$$w_2 = w_2 - \eta \nabla w_2$$
$$\boxed{= w_2 + \eta(y-\hat{y})x_2}$$

Gradient Descent

For each sample $\{x_i, y_i\}$

1. Predict

   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$

   b. Compute Loss $\mathcal{L}_i = \frac{1}{2}(y_i - \hat{y})$

   (side computation to track loss. not needed for backprop)

2. Update

   two lines now

   a. Back Propagation

   $$\nabla w_{1i} = -(y_i - \hat{y})x_{1i}$$
   $$\nabla w_{2i} = -(y_i - \hat{y})x_{2i}$$

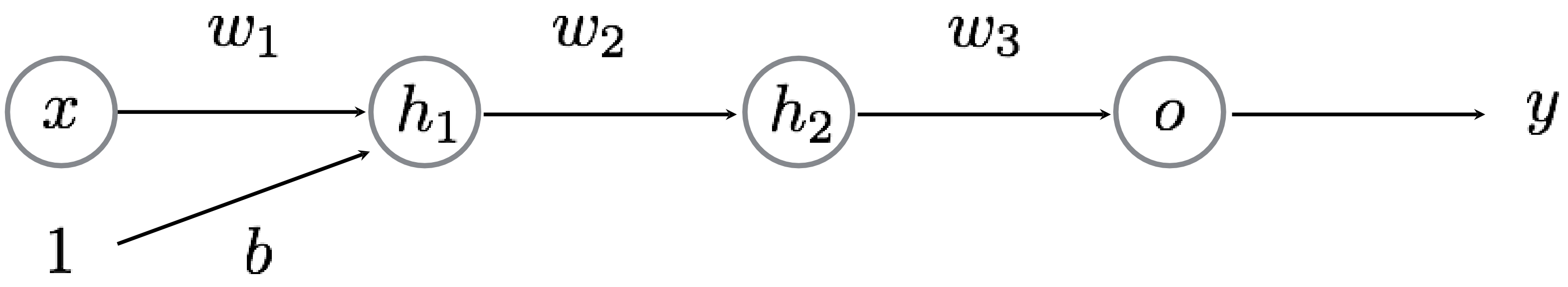
   b. Gradient update

   $$w_{1i} = w_{1i} + \eta(y - \hat{y})x_{1i}$$
   $$w_{2i} = w_{2i} + \eta(y - \hat{y})x_{2i}$$

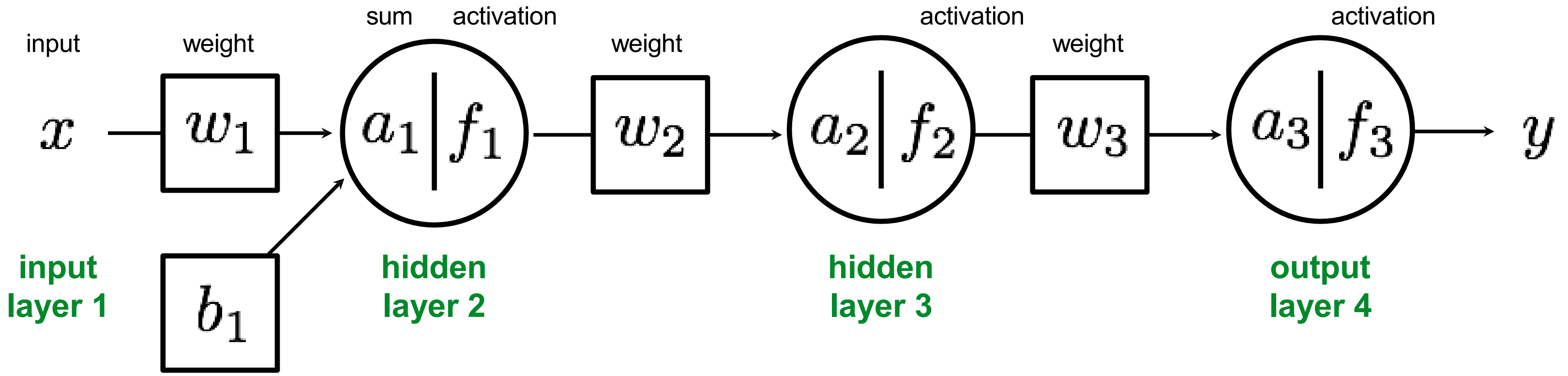
   (adjustable step size)

We haven’t seen a lot of ‘propagation’ yet
because our perceptrons only had <u>one</u> layer…

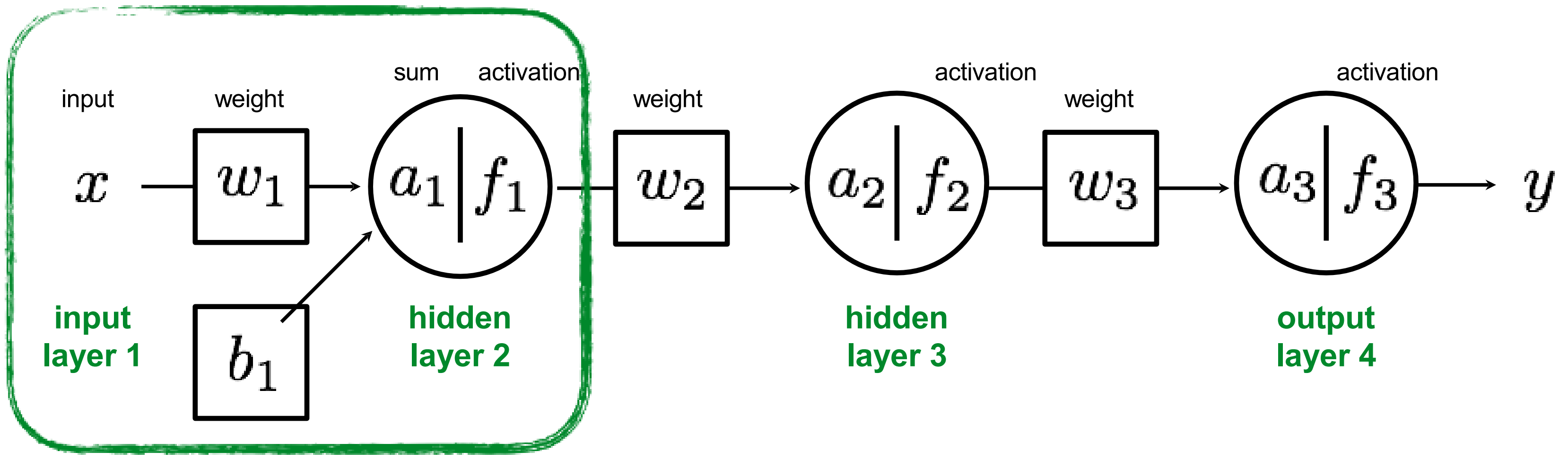
Multi-Layer Perceptron



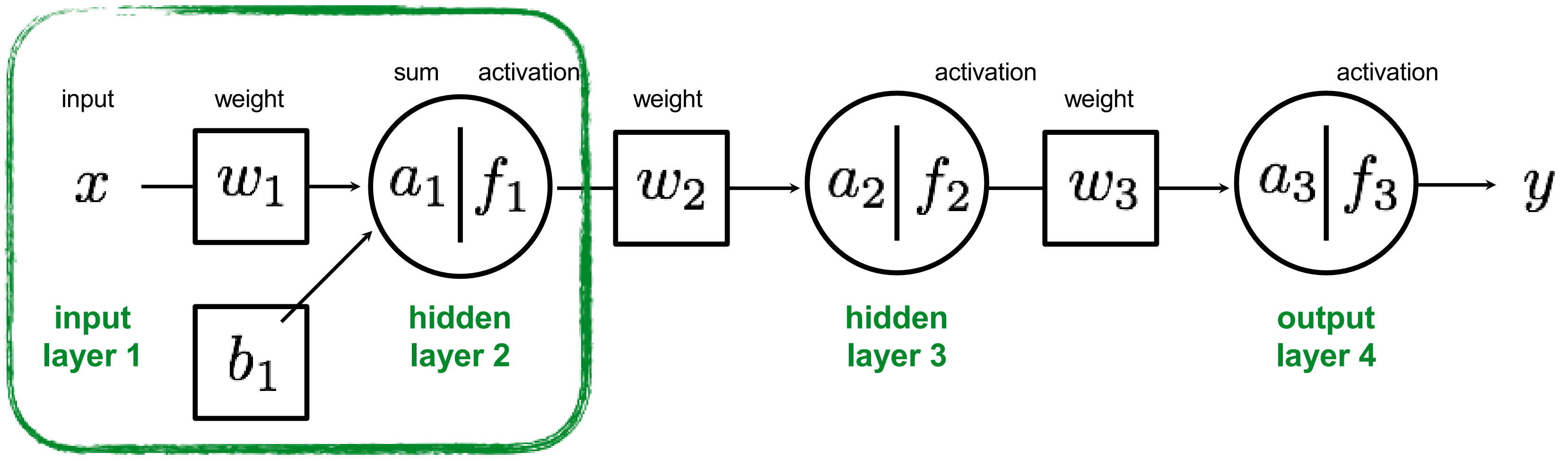
function of **FOUR** parameters and **FOUR** layers!



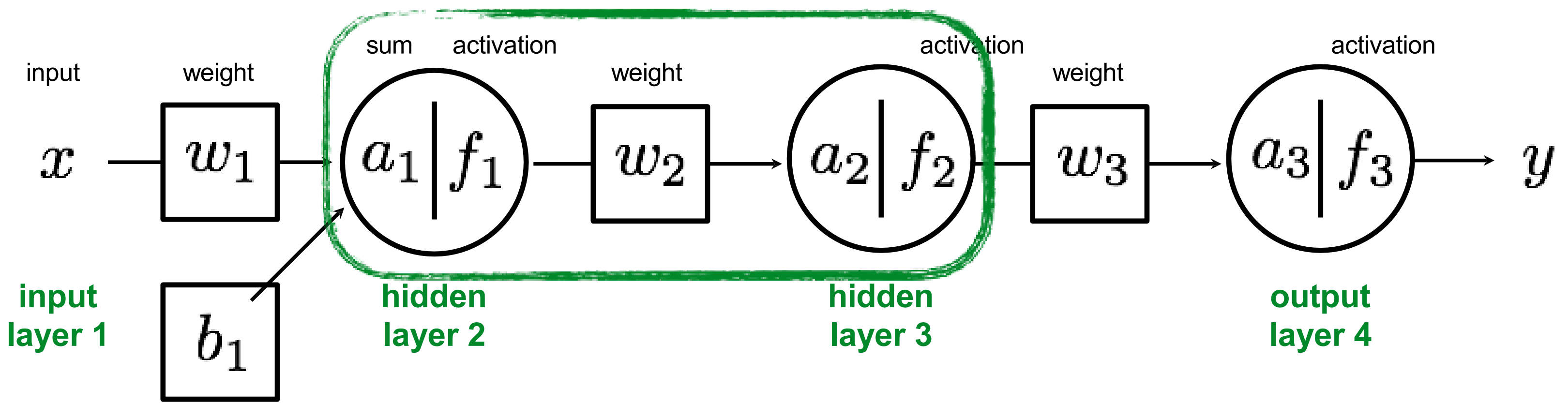
input
weight
sum
activation
weight
activation
weight
activation
x
w_1
a_1 | f_1
w_2
a_2 | f_2
w_3
a_3 | f_3
y
b_1
input layer 1
hidden layer 2
hidden layer 3
output layer 4



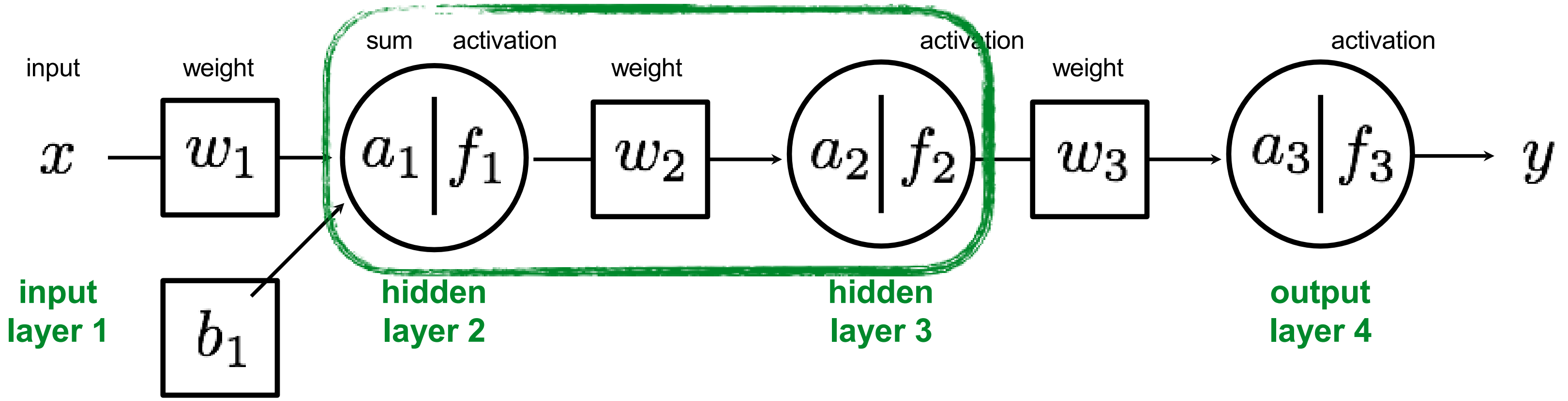
input
weight
sum
activation
weight
activation
weight
activation
x
w_1
b_1
a_1 f_1
w_2
a_2 f_2
w_3
a_3 f_3
y
input layer 1
hidden layer 2
hidden layer 3
output layer 4



$$a_1 = w_1 \cdot x + b_1$$

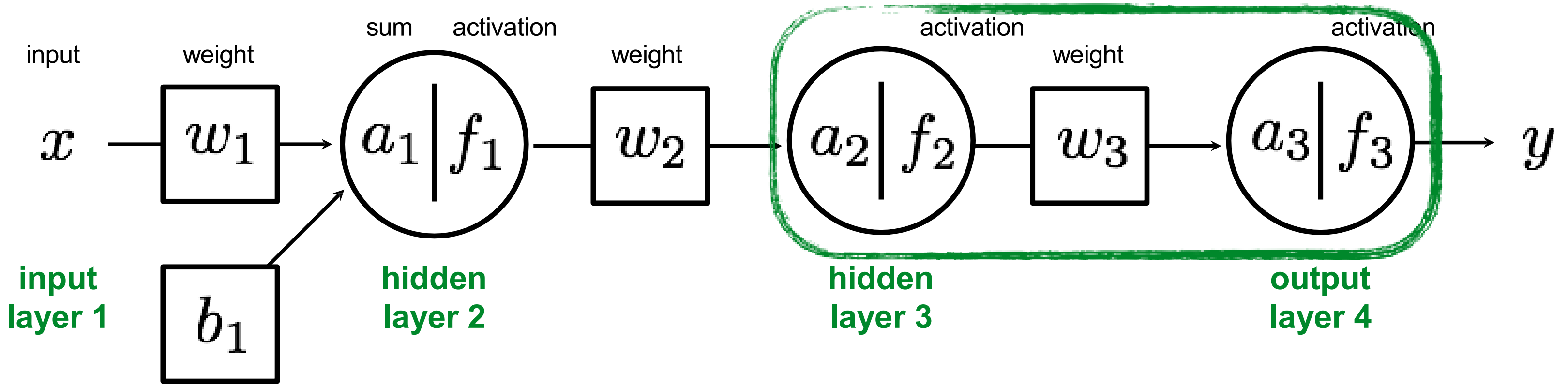


$$a_1 = w_1 \cdot x + b_1$$



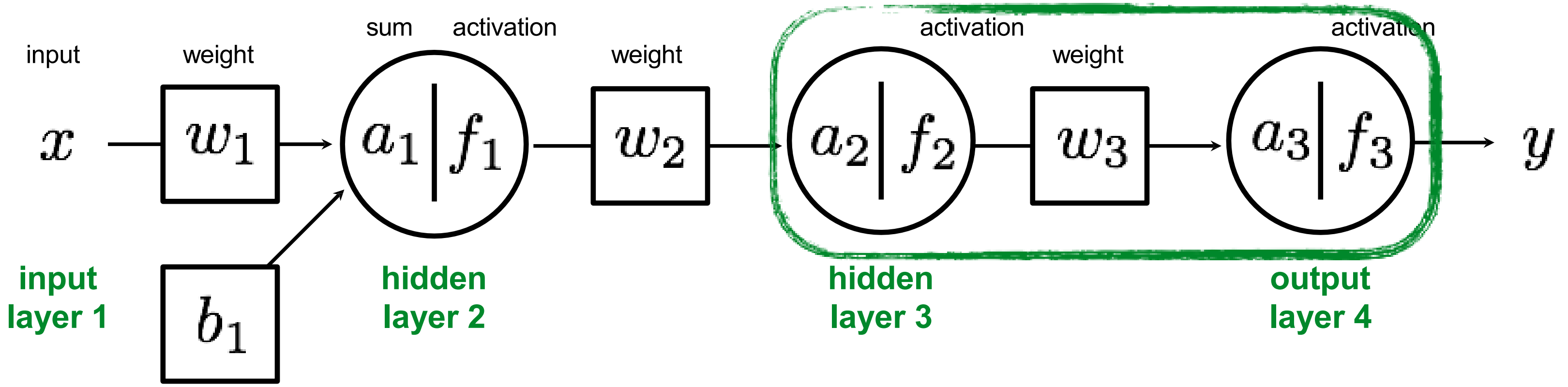
$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$



$$a_1 = w_1 \cdot x + b_1$$

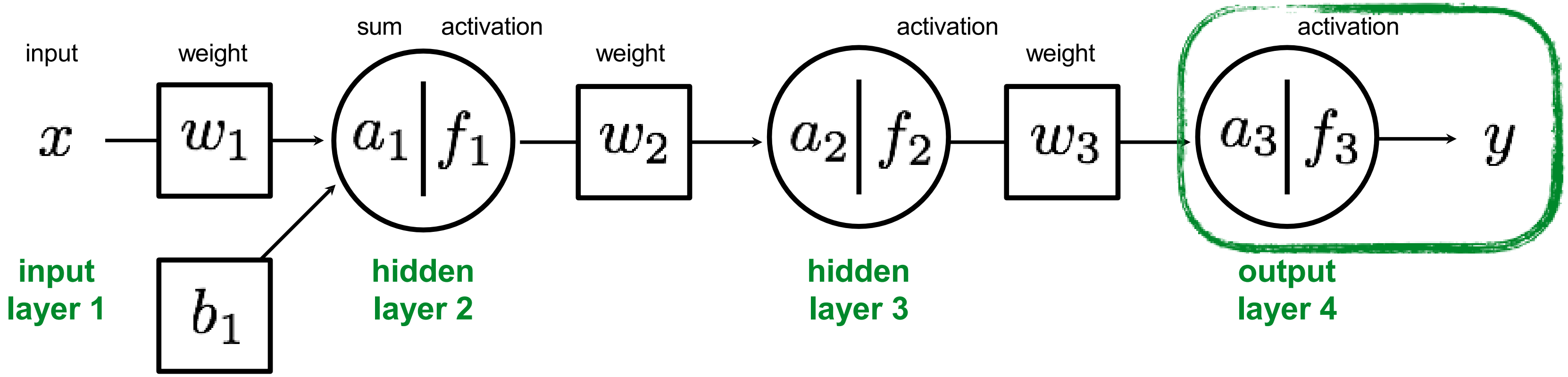
$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

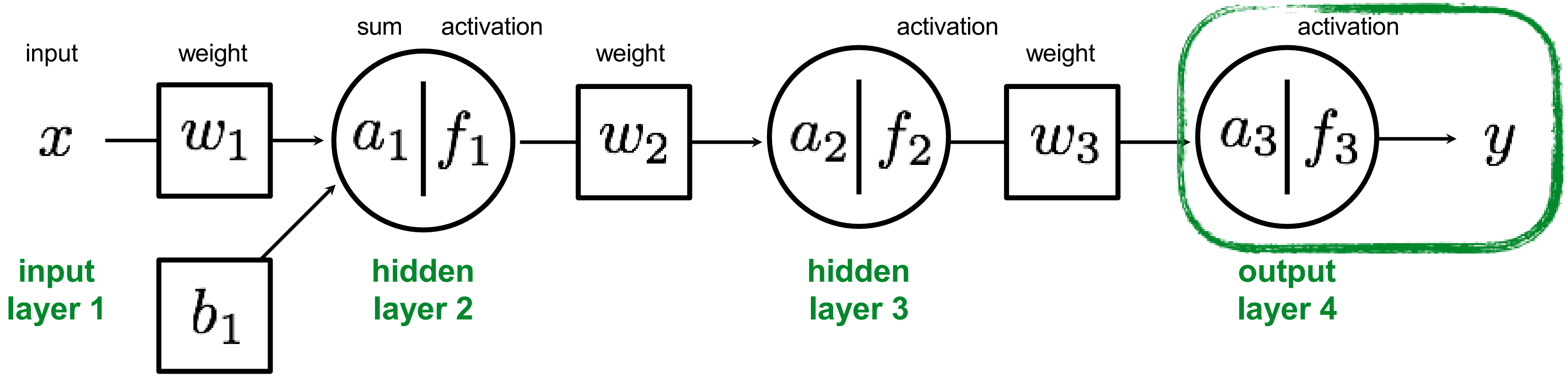
$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

Entire network can be written out as one long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

We need to train the network:

What is known? What is unknown?

Entire network can be written out as one long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

known

We need to train the network:

What is known? What is unknown?

Entire network can be written out as one long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

activation function sometimes has parameters

unknown

We need to train the network:

What is known? What is unknown?

Learning an MLP (Multi-Layer Perceptron)

Given a set of samples and a MLP

$$\{x_i, y_i\}$$

$$y = f_{\text{MLP}}(x; \theta)$$

Estimate the parameters of the MLP

$$\theta = \{f, w, b\}$$

Gradient Descent

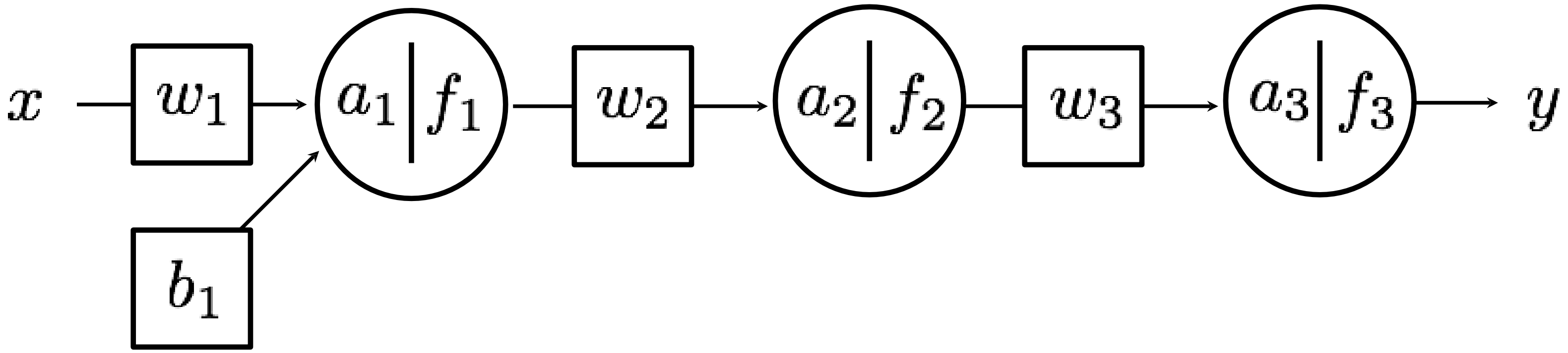
For each **random** sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
   b. Compute Loss
2. Update
   a. Back Propagation $\frac{\partial \mathcal{L}}{\partial \theta}$ vector of parameter partial derivatives
   b. Gradient update $\theta \leftarrow \theta - \eta \nabla \theta$

vector of parameter update equations

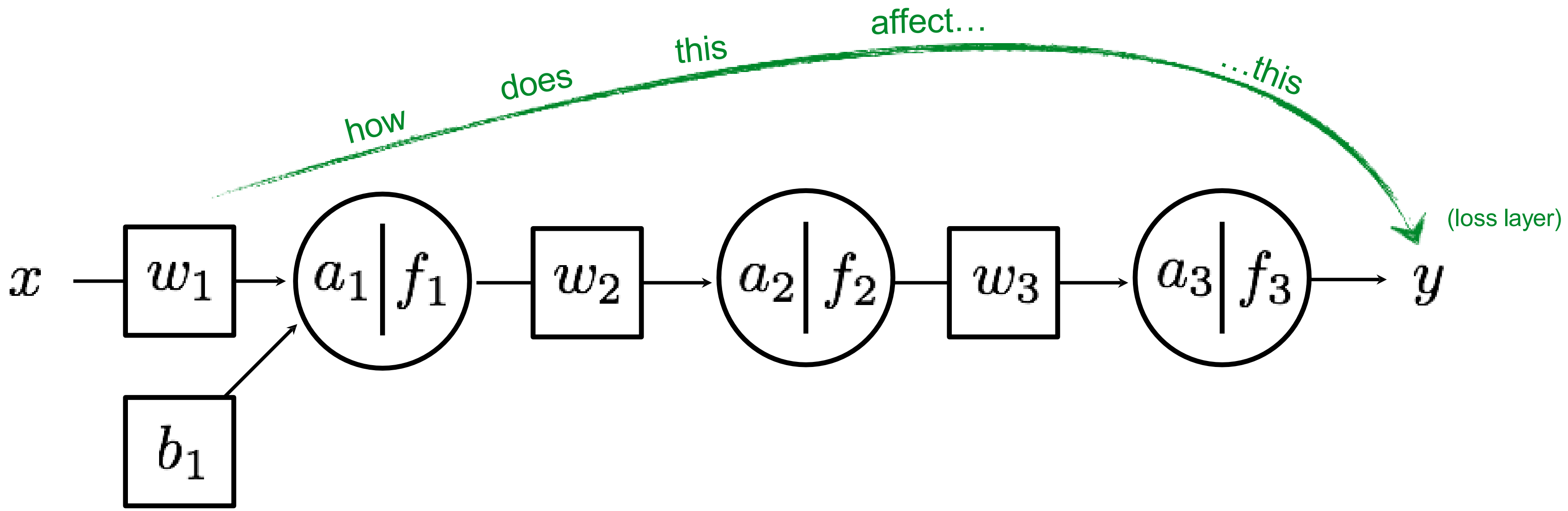
So we need to compute the partial derivatives

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \left[\frac{\partial \mathcal{L}}{\partial w_3} \frac{\partial \mathcal{L}}{\partial w_2} \frac{\partial \mathcal{L}}{\partial w_1} \frac{\partial \mathcal{L}}{\partial b} \right]$$



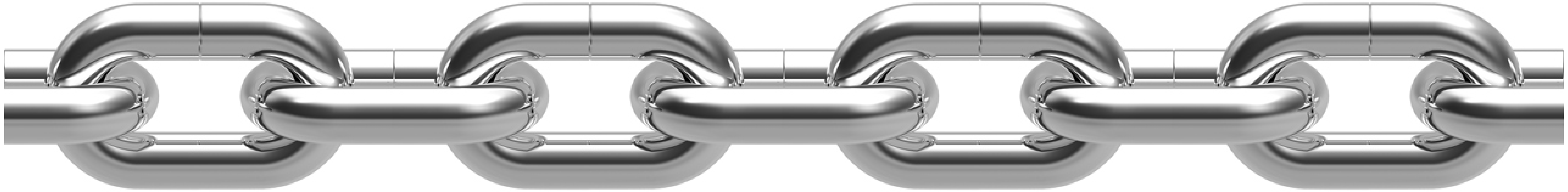
Remember,

Partial derivative $\frac{\partial L}{\partial w_1}$ describes…



So, how do you compute it?

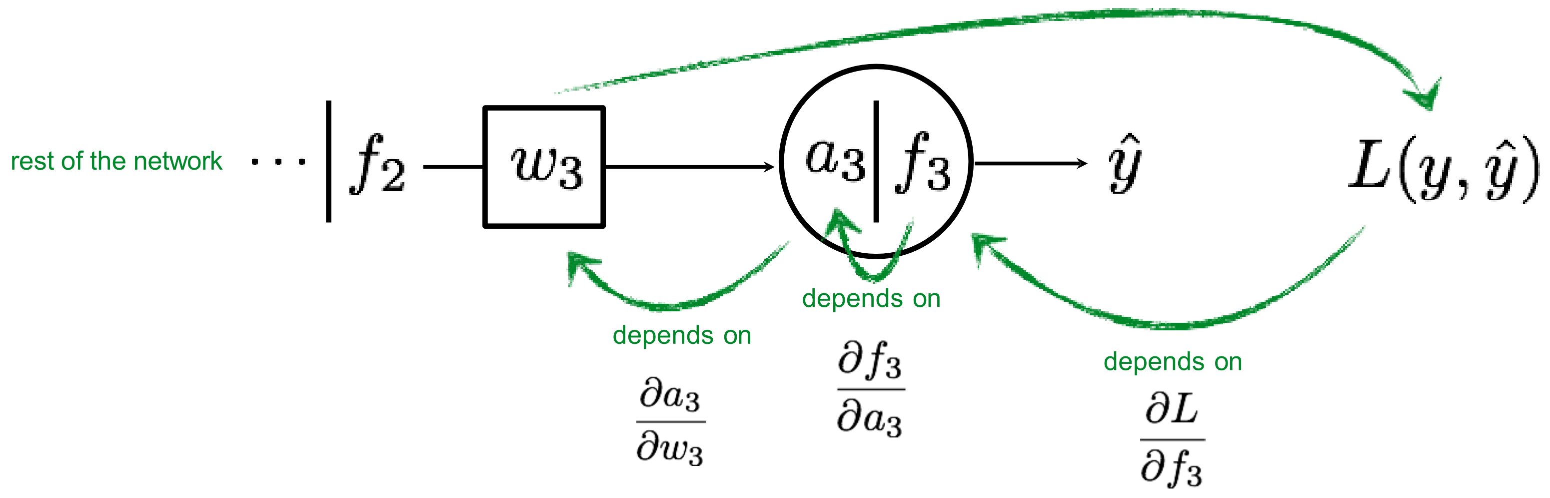
The Chain Rule

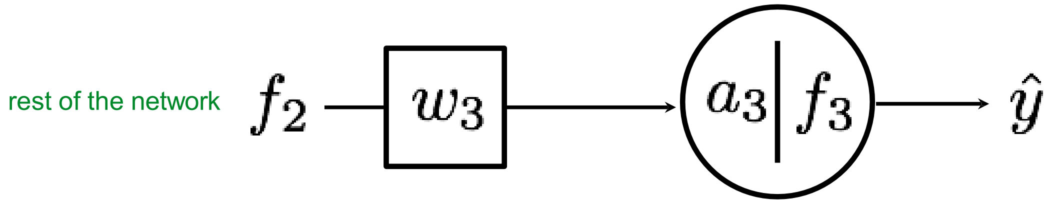


According to the chain rule…

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

Intuitively, the effect of weight on loss function : $\frac{\partial L}{\partial w_3}$

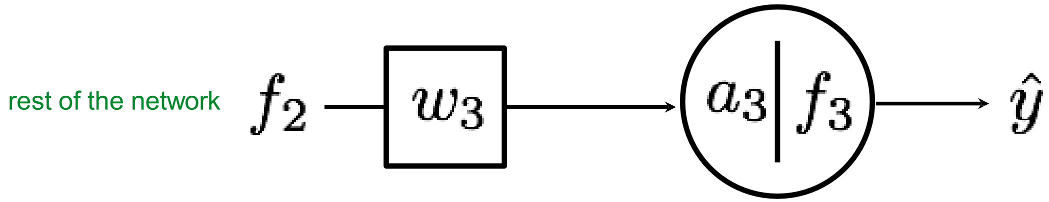




$$L(y, \hat{y}) = \frac{\eta}{2}(y - \hat{y})^2$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3}$$

Chain Rule!

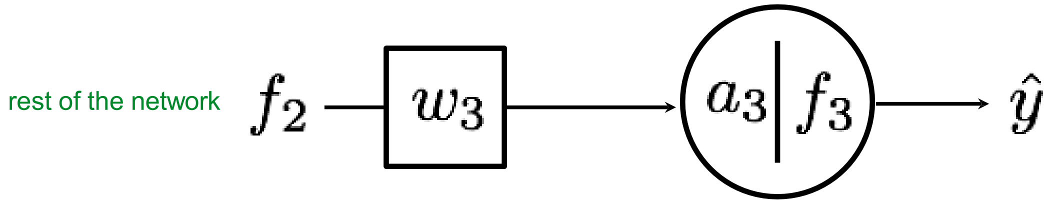


$$L(y, \hat{y}) = \frac{\eta}{2}(y - \hat{y})^2$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

$$= -\eta(y - \hat{y})\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

Just the partial
derivative of L2 loss



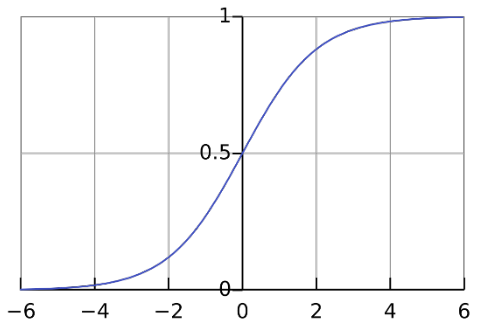
$$L(y,\hat{y}) = \frac{\eta}{2}(y-\hat{y})^2$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

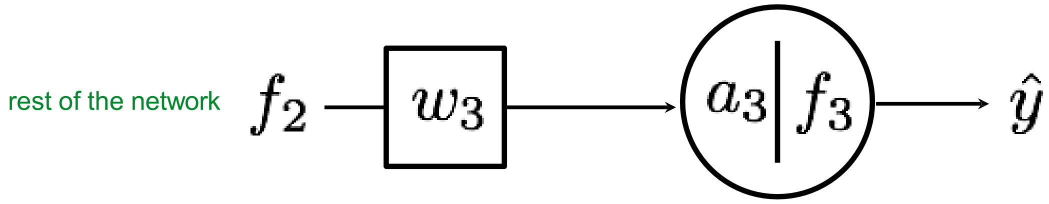
$$= -\eta(y-\hat{y})\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

Let’s use a Sigmoid function

$$\frac{ds(x)}{dx} = s(x)(1-s(x))$$



$$s(x) = \frac{1}{1+e^{-x}}$$



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

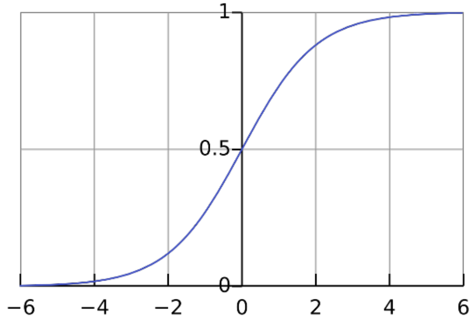
$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

$$= -\eta(y - \hat{y})\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

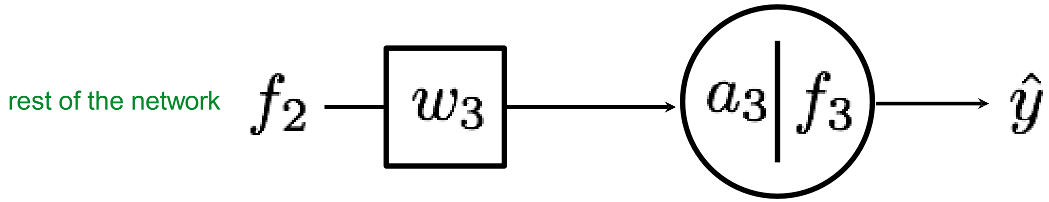
$$= -\eta(y - \hat{y})f_3(1 - f_3)\frac{\partial a_3}{\partial w_3}$$

Let’s use a Sigmoid function

$$\frac{ds(x)}{dx} = s(x)(1 - s(x))$$

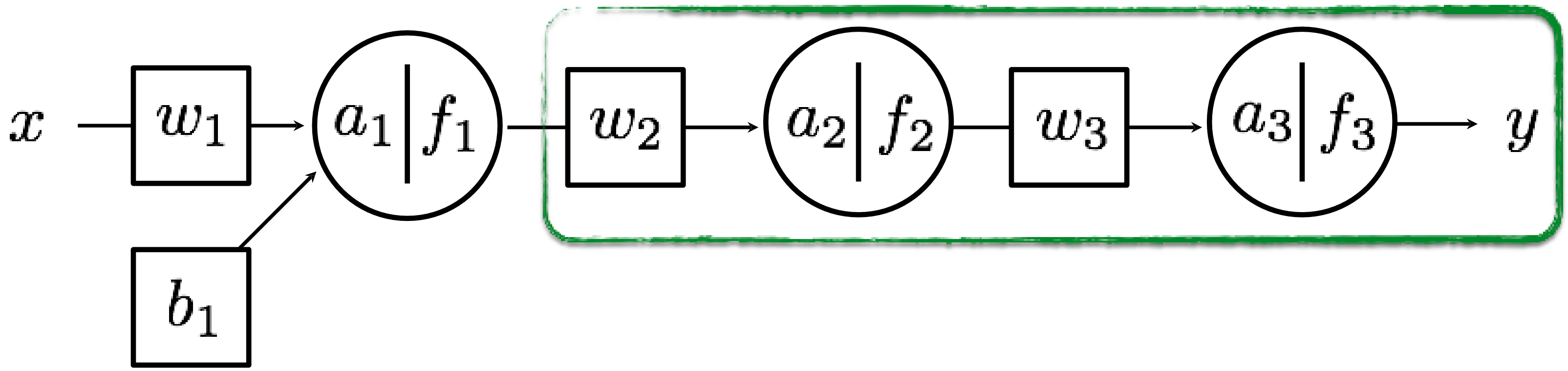


$$s(x) = \frac{1}{1 + e^{-x}}$$

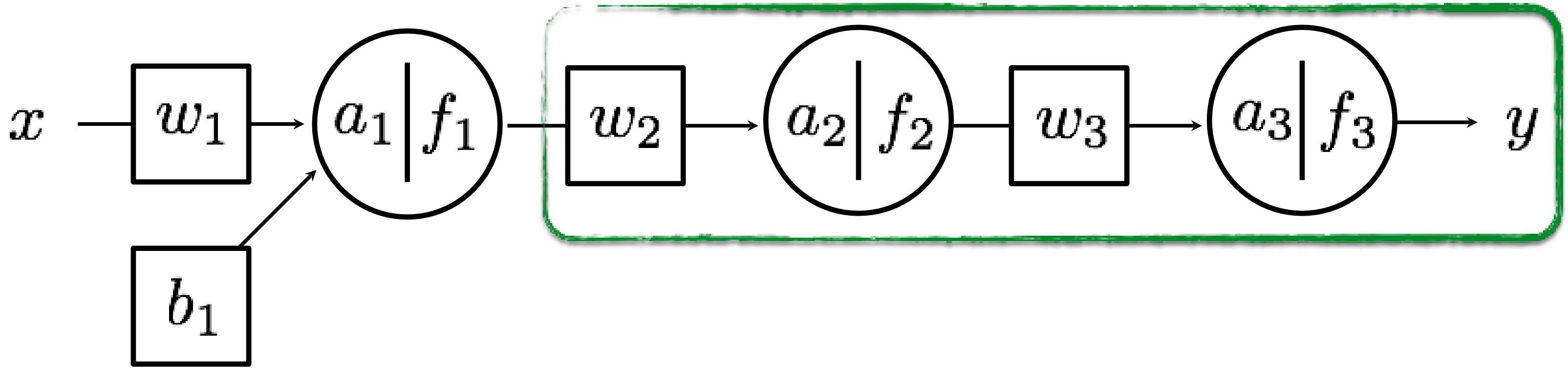


$$L(y,\hat{y}) = \frac{\eta}{2}(y-\hat{y})^2$$

$$\begin{aligned}
\frac{\partial L}{\partial w_3} &= \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3} \\
&= -\eta(y-\hat{y})\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3} \\
&= -\eta(y-\hat{y})f_3(1-f_3)\frac{\partial a_3}{\partial w_3} \\
&= -\eta(y-\hat{y})f_3(1-f_3)f_2
\end{aligned}$$



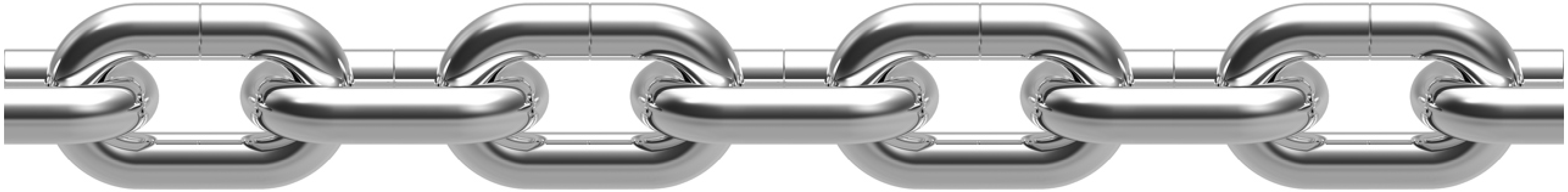
$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial w_2}$$



$$\frac{\partial L}{\partial w_2} = \boxed{\frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$

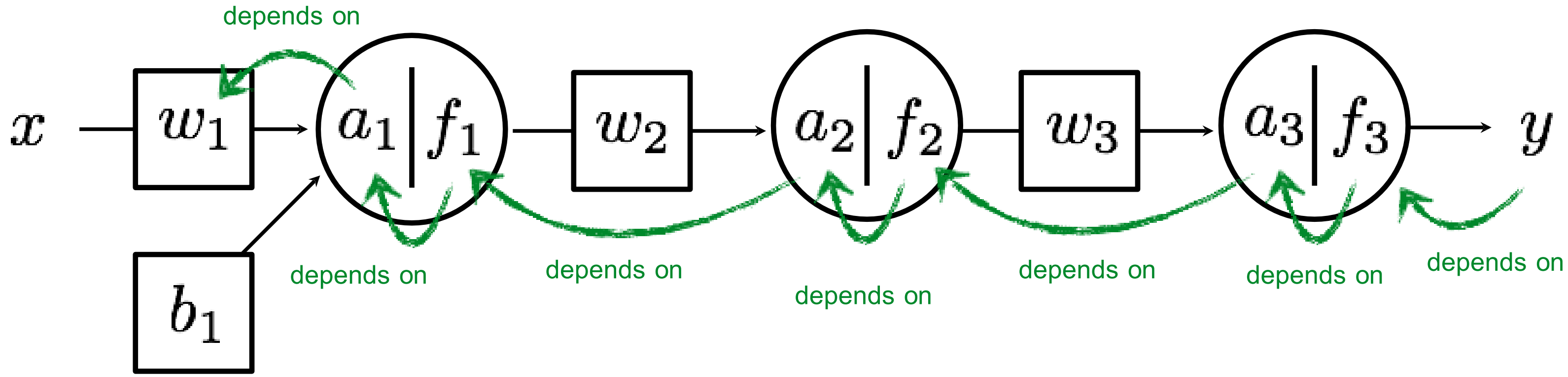
already computed.
re-use (propagate)!

The Chain Rule



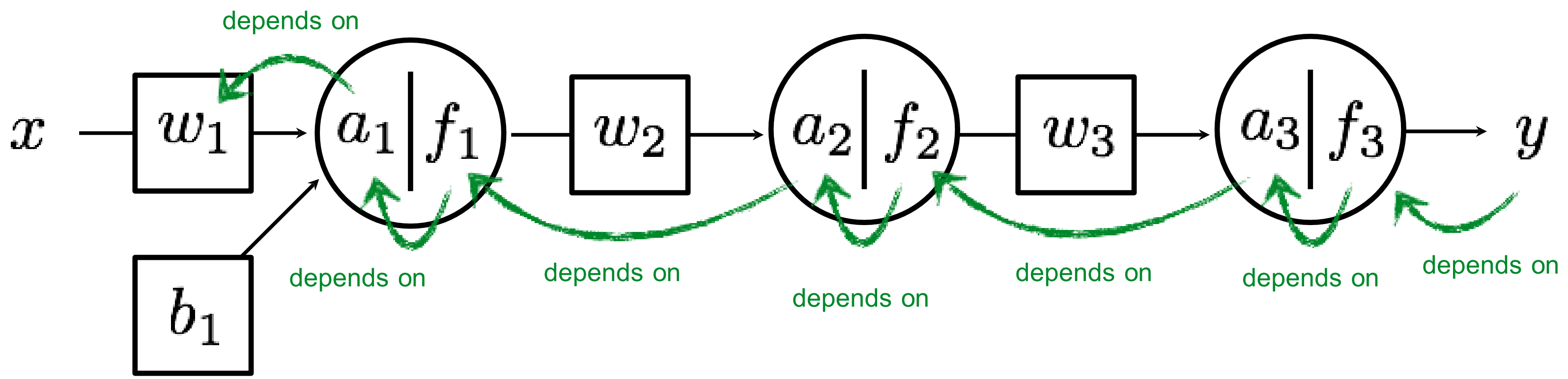
a.k.a. backpropagation

The chain rule says…



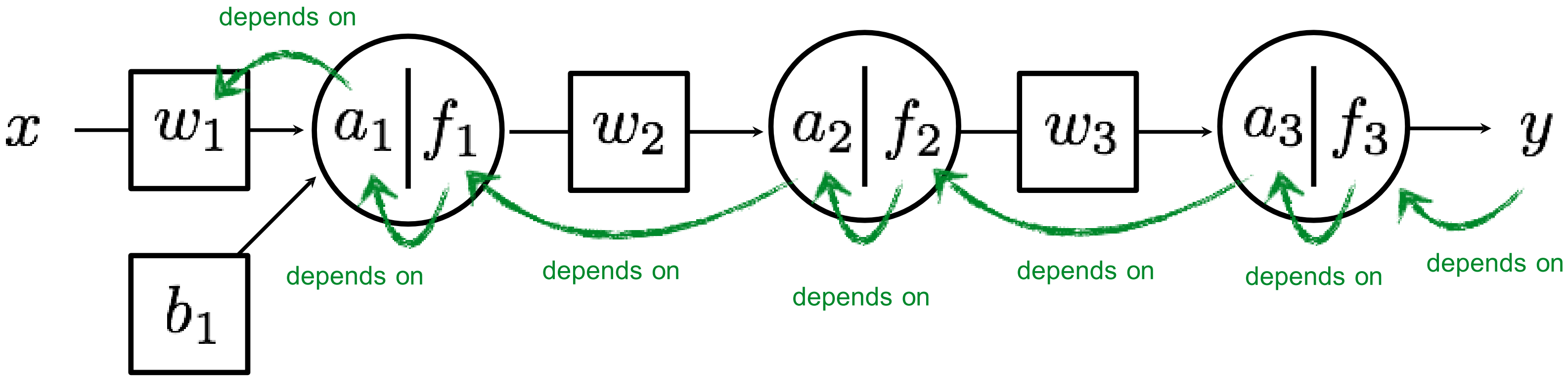
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

The chain rule says…



$$\frac{\partial L}{\partial w_1} = \boxed{\frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

already computed.
re-use (propagate)!

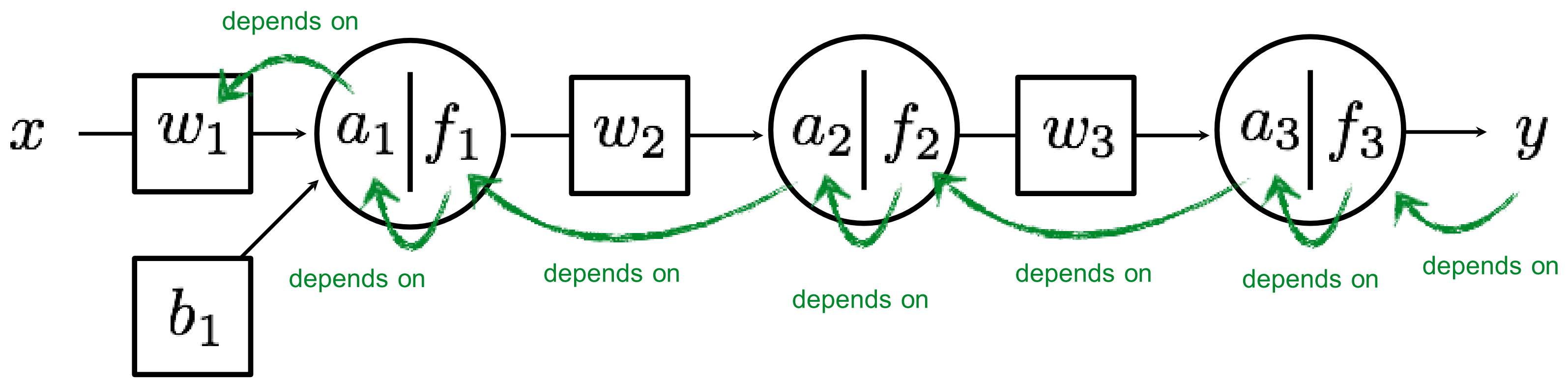


$$\frac{\partial \mathcal{L}}{\partial w_3} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}}\frac{\partial a_3}{\partial w_3}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial b}$$

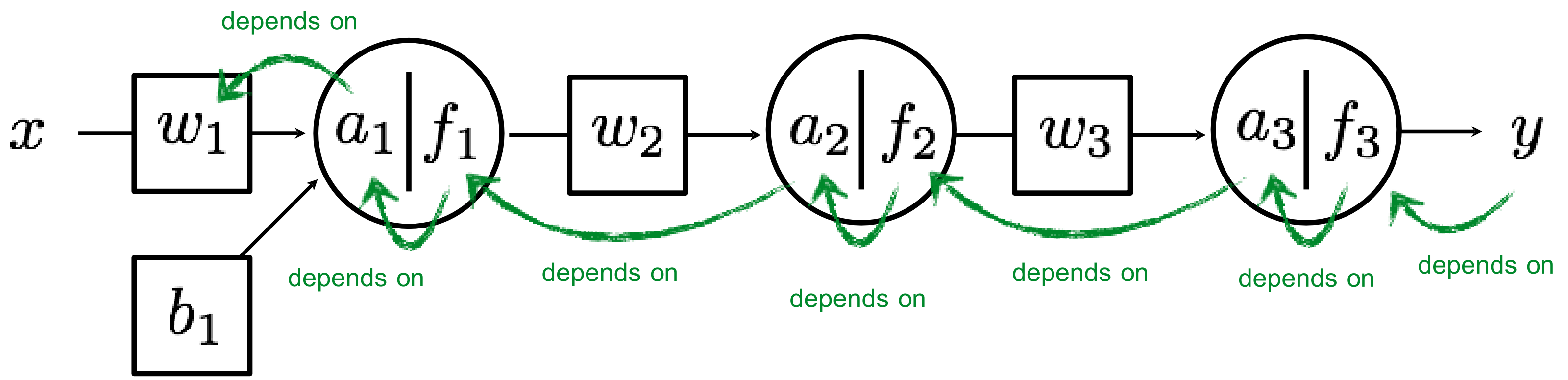


$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}}\frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial b}$$



$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}}\frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}}\frac{\partial a_1}{\partial b}$$

Gradient Descent

For each example sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
   b. Compute Loss $\mathcal{L}_i$
2. Update
   a. Back Propagation

$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$
$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$
$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$
$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial b}$$

   b. Gradient update

$$w_3 = w_3 - \eta \nabla w_3$$
$$w_2 = w_2 - \eta \nabla w_2$$
$$w_1 = w_1 - \eta \nabla w_1$$
$$b = b - \eta \nabla b$$

Gradient Descent

For each example sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
   b. Compute Loss $\mathcal{L}_i$
2. Update
   a. Back Propagation $\frac{\partial \mathcal{L}}{\partial \theta}$

      vector of parameter partial derivatives

   b. Gradient update $\theta \leftarrow \theta + \eta \frac{\partial \mathcal{L}}{\partial \theta}$

      vector of parameter update equations

Stochastic Gradient Descent

What we are truly minimizing:

$$\min_{\theta} \sum_{i=1}^{N} L(y_i, f_{MLP}(x_i))$$

What we are truly minimizing:

$$\min_{\theta} \sum_{i=1}^{N} L(y_i, f_{MLP}(x_i))$$

The gradient is:

$$\sum_{i=1}^{N} \frac{\partial L(y_i, f_{MLP}(x_i))}{\partial \theta}$$

What we are truly minimizing:

$$\min_{\theta} \sum_{i=1}^{N} L(y_i, f_{MLP}(x_i))$$

The gradient is:

$$\sum_{i=1}^{N} \frac{\partial L(y_i, f_{MLP}(x_i))}{\partial \theta}$$

What we use for gradient update is:

$$\frac{\partial L(y_i, f_{MLP}(x_i))}{\partial \theta}$$ for some i

<u>Stochastic</u> Gradient Descent

- For each example sample $\{x_i, y_i\}$

  1. Predict $\hat{y} = f_{\text{MLP}}(x_i; \theta)$

     a. Forward pass $\mathcal{L}_i$

     b. Compute Loss

  2. Update $\frac{\partial \mathcal{L}}{\partial \theta}$

     vector of parameter partial derivatives

     a. Back Propagation

     b. Gradient update $\theta \leftarrow \theta + \eta \frac{\partial \mathcal{L}}{\partial \theta}$

     vector of parameter update equations

How do we select which sample?

How do we select which sample?

- Select randomly!

Do we need to use only one sample?

How do we select which sample?

- Select randomly!

Do we need to use only one sample?

- You can use a *minibatch* of size B < N.

Why not do gradient descent with all samples?

How do we select which sample?

- Select randomly!

Do we need to use only one sample?

- You can use a *minibatch* of size B < N.

Why not do gradient descent with all samples?

- It’s very expensive when N is large (big data).

Do I lose anything by using stochastic GD?

How do we select which

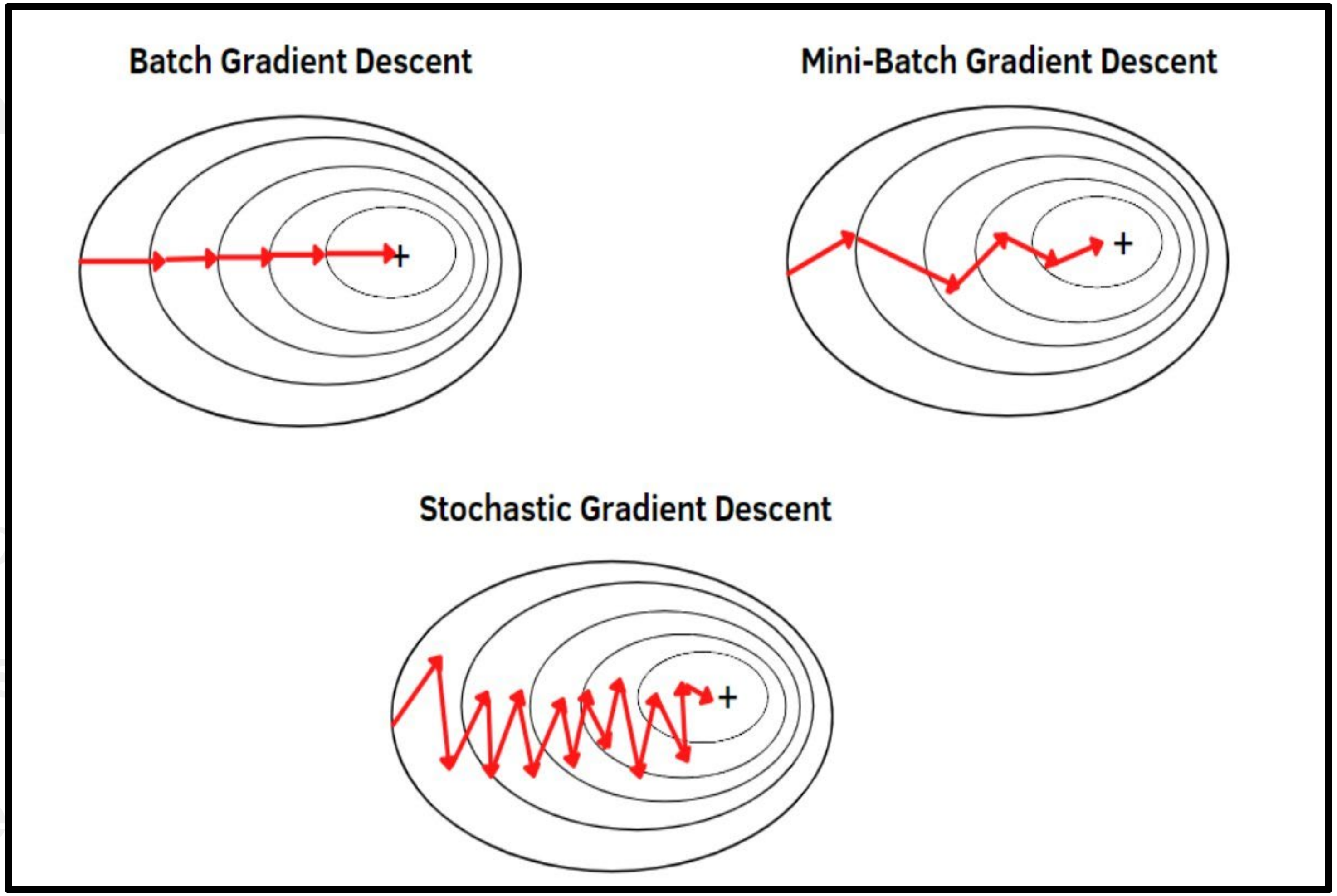
- Select randomly!

Do we need to use only

- You can use a *minibatc*

Why not do gradient des

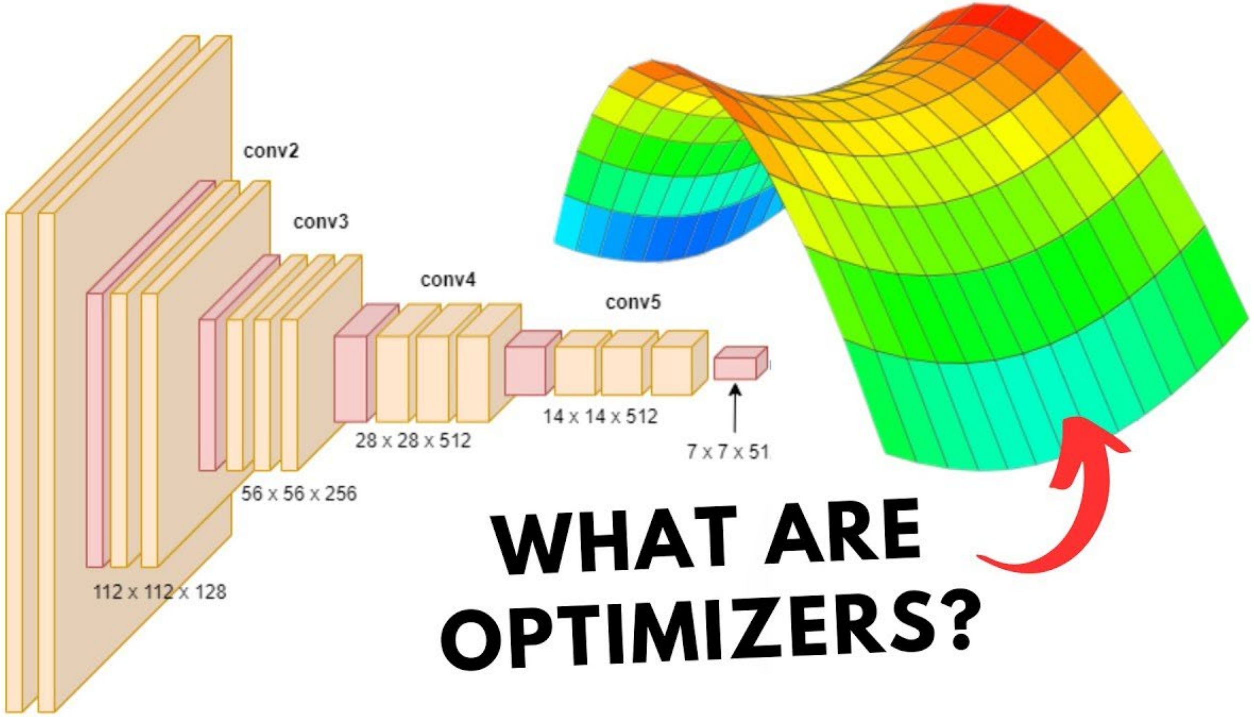
- It's very expensive whe



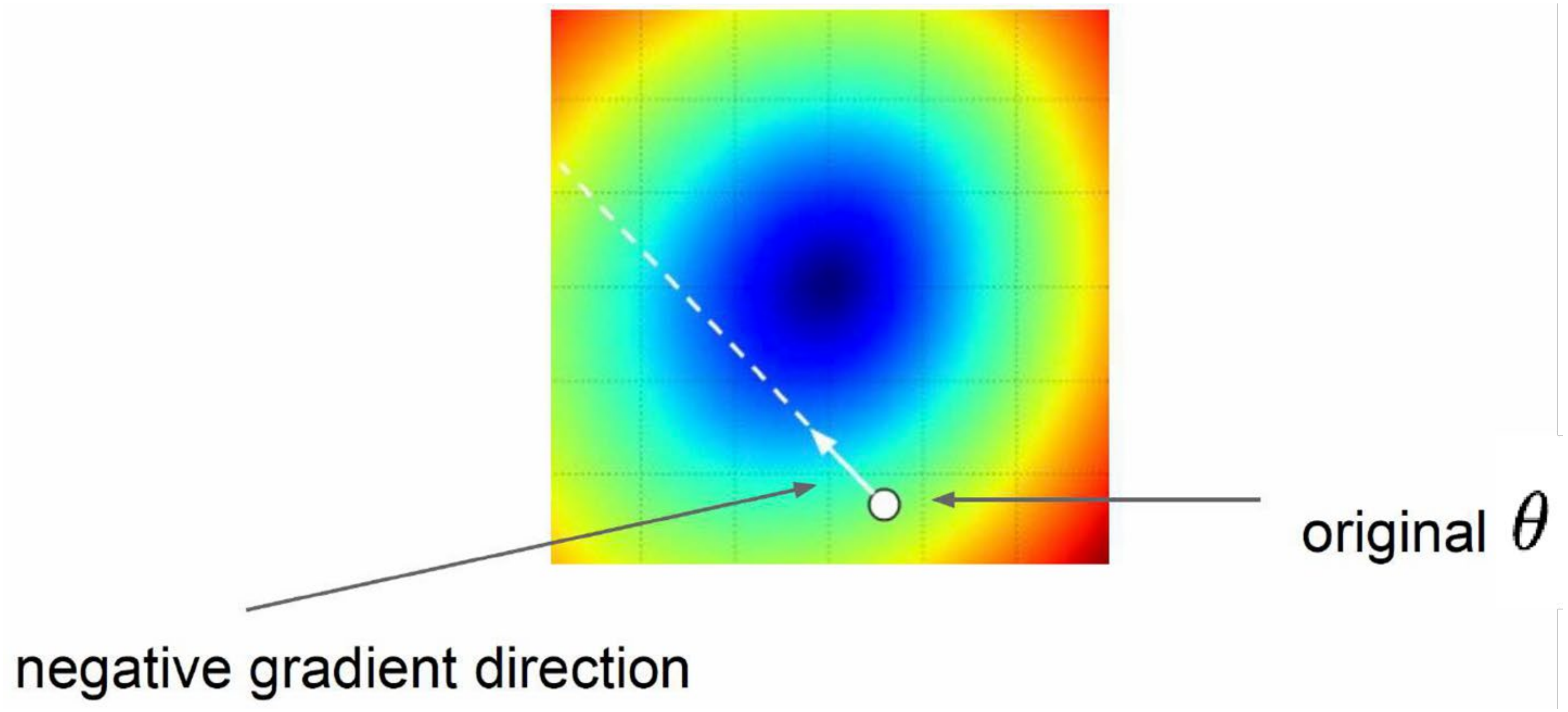
Do I lose anything by using stochastic GD?

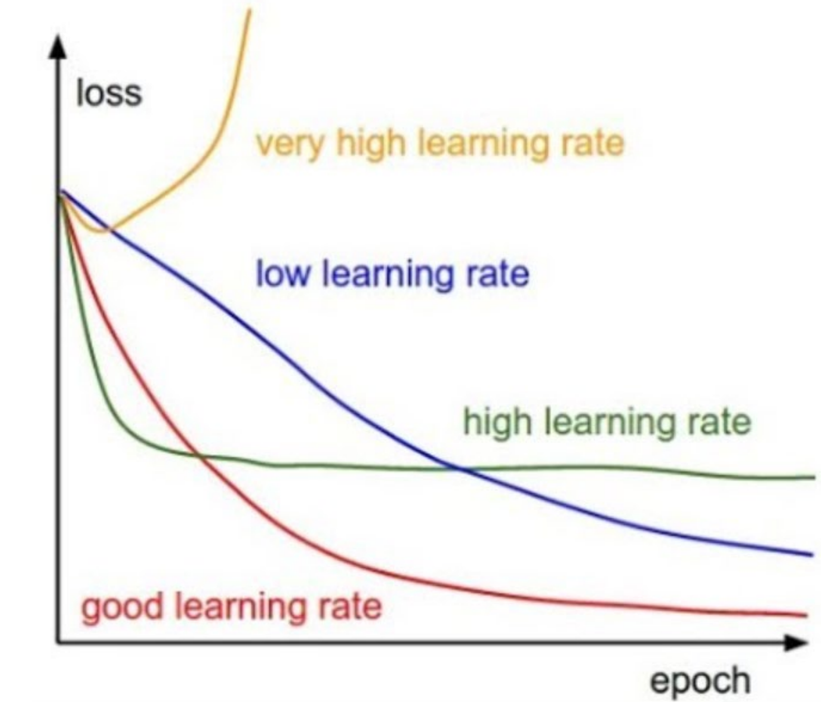
- Same convergence guarantees and complexity!
- Better generalization

Notes on Optimization



Learning rates





$$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}$$

Step size: learning rate

Too big: will miss the minimum

Too small: slow convergence

Learning rate scheduling

- Use different **learning rate** at each iteration
- Most common choice:

$$\eta_t = \frac{\eta_0}{\sqrt{t}}$$

- Need to select initial learning rate η_0, important!!!

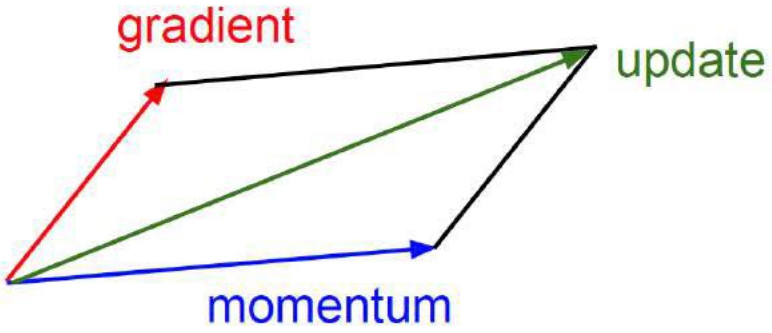
- More modern choice: **Adaptive** learning rates

$$\eta_t = G\left(\left\{\frac{\partial L}{\partial \theta}\right\}_{i=0}^{t}\right)$$

- Many choices for G (**Adam**, **Adagrad**, **Adadelta**)

Momentum Update

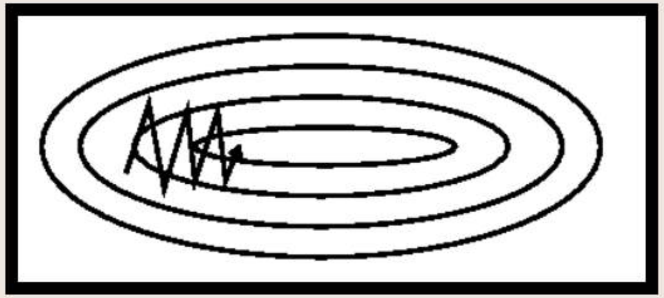
$$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}$$



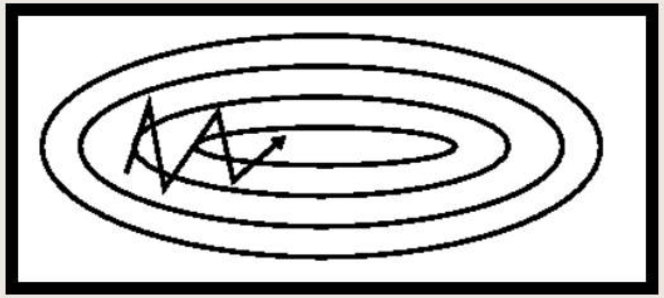
$$\Delta\theta \leftarrow w \frac{\partial L}{\partial \theta} + (1 - w)\Delta\theta$$

Take direction history into account!

```
weights_grad = evaluate_gradient(loss_fun, data, weights)
vel = vel * 0.9 - step_size * weights_grad
weights += vel
```



(Fig. 2a)



(Fig. 2b)

Many other ways to perform optimization…

- Second order methods that use the Hessian (or its approximation): BFGS, **LBFGS**, etc.

- Currently, the lesson from the trenches is that well-tuned SGD+Momentum is very hard to beat for CNNs.

- No consensus on Adam etc.: Seem to give faster performance to worse local minima

Derivatives

- Given f(x), where x is vector of inputs
  - Compute gradient of f at x: $\nabla f(x)$

How do we do differentiation?

Numerical differentiation

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$f(x+h) = f(x) + h\,\frac{df(x)}{dx}$$

Numerical differentiation

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$f(x+h) = f(x) + h\frac{df(x)}{dx}$$

Numerical differentiation is:

– Approximate

– Slow

– Numerically unstable

– Easy to write

Symbolic differentiation

- What Mathematica does: Automatically derive *analytical* expressions for derivative.

Symbolic differentiation

- What Mathematica does: Automatically derive *analytical* expressions for derivative.
- Often results in very redundant (and expensive to evaluate) expressions.

```
D[Log[1 + Exp[w * x + b]], w]
```

Out[11]= $\dfrac{e^{b+w\,x}\,w}{1+e^{b+w\,x}}$

```
In[19]:= D[Log[1 + Exp[w2 * Log[1 + Exp[w1 * x + b1]] + b2]], w1]
```

Out[19]= $\dfrac{e^{b1+b2+w1\,x+w2\,\mathrm{Log}\left[1+e^{b1+w1\,x}\right]}\,w2\,x}{\left(1+e^{b1+w1\,x}\right)\left(1+e^{b2+w2\,\mathrm{Log}\left[1+e^{b1+w1\,x}\right]}\right)}$

- Often intractable.

Automatic differentiation (autodiff)

- An autodiff system will convert the program into a sequence of primitive operations which have specified routines for computing derivatives.
- In this representation, backprop can be done in a completely mechanical way.

Original program:

$$z = wx + b$$

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

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

Sequence of primitive operations:

$$t_1 = wx$$

$$z = t_1 + b$$

$$t_3 = -z$$

$$t_4 = \exp(t_3)$$

$$t_5 = 1 + t_4$$

$$y = 1/t_5$$

$$t_6 = y - t$$

$$t_7 = t_6^2$$

$$\mathcal{L} = t_7/2$$

In summary

- Numerical gradient: easy to implement, bad to use.

- Symbolic gradient: sometimes useful, often intractable.

- Automatic gradient: exact, fast, error-prone.

In practice: Use symbolic gradient for small/trivial programs. Almost always use analytic gradient, but check correctness of implementation with numerical gradient.

- This is called a gradient check.