SAIR

Spatial AI & Robotics Lab

CSE 473/573-A

L22: MULTI-LAYERED PERCEPTRON

Chen Wang

Spatial AI & Robotics Lab

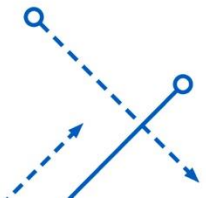
Department of Computer Science and Engineering

University at Buffalo The State University of New York

Many Slides from Kris Kitani

History

- 1950s Age of the Perceptron
- 1957 The Perceptron (Rosenblatt) 1969 Perceptron (Minsky, Papert)
- 1980s Age of the Neural Network 1986 Back propagation (Hinton)
- 1990s Age of the Graphical Model 2000s Age of the Support Vector Machine
- 2010s Age of the Deep Network
- **Deep Learning = known algorithms + computing power + big data**



The milestone

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

There have been many attempts to design self-organizing neural networks. The aim is to find a powerful synaptic modification rule that will allow an arbitrarily connected neural network to develop an internal structure that is appropriate for a particular task domain. The task is specified by giving the desired state vector of the output units for each state vector of the input units. If the input units are directly connected to the output units it is relatively easy to find learning rules that iteratively adjust the relative strengths of the connections so as to progressively reduce the difference between the actual and desired output vectors[2]. Learning becomes more interesting but more difficult when we introduce hidden units whose actual or desired states are not specified by the task. (In perceptrons, there are 'feature analysers' between the input and output that are not true hidden units because their input connections are fixed by hand, so their states are completely determined by the input vector: they do not learn representations.) The learning procedure must decide under what circumstances the hidden units should be active in order to help achieve the desired input–output behaviour. This amounts to deciding what these units should represent. We demonstrate that a general purpose and relatively simple procedure is powerful enough to construct appropriate internal representations.

The simplest form of the learning procedure is for layered networks which have a layer of input units at the bottom; any number of intermediate layers; and a layer of output units at the top. Connections within a layer or from higher to lower layers are forbidden, but connections can skip intermediate layers. An input vector is presented to the network by setting the states of the input units. Then the states of the units in each layer are determined by applying equations (1) and (2) to the connections coming from lower layers. All units within a layer have their states set in parallel, but different layers have their states set sequentially, starting at the bottom and working upwards until the states of the output units are determined.

The total input, x_j, to unit j is a linear function of the outputs, y_i, of the units that are connected to j and of the weights, w_{ji}, on these connections

$$x_j = \sum_i y_i w_{ji} \qquad (1)$$

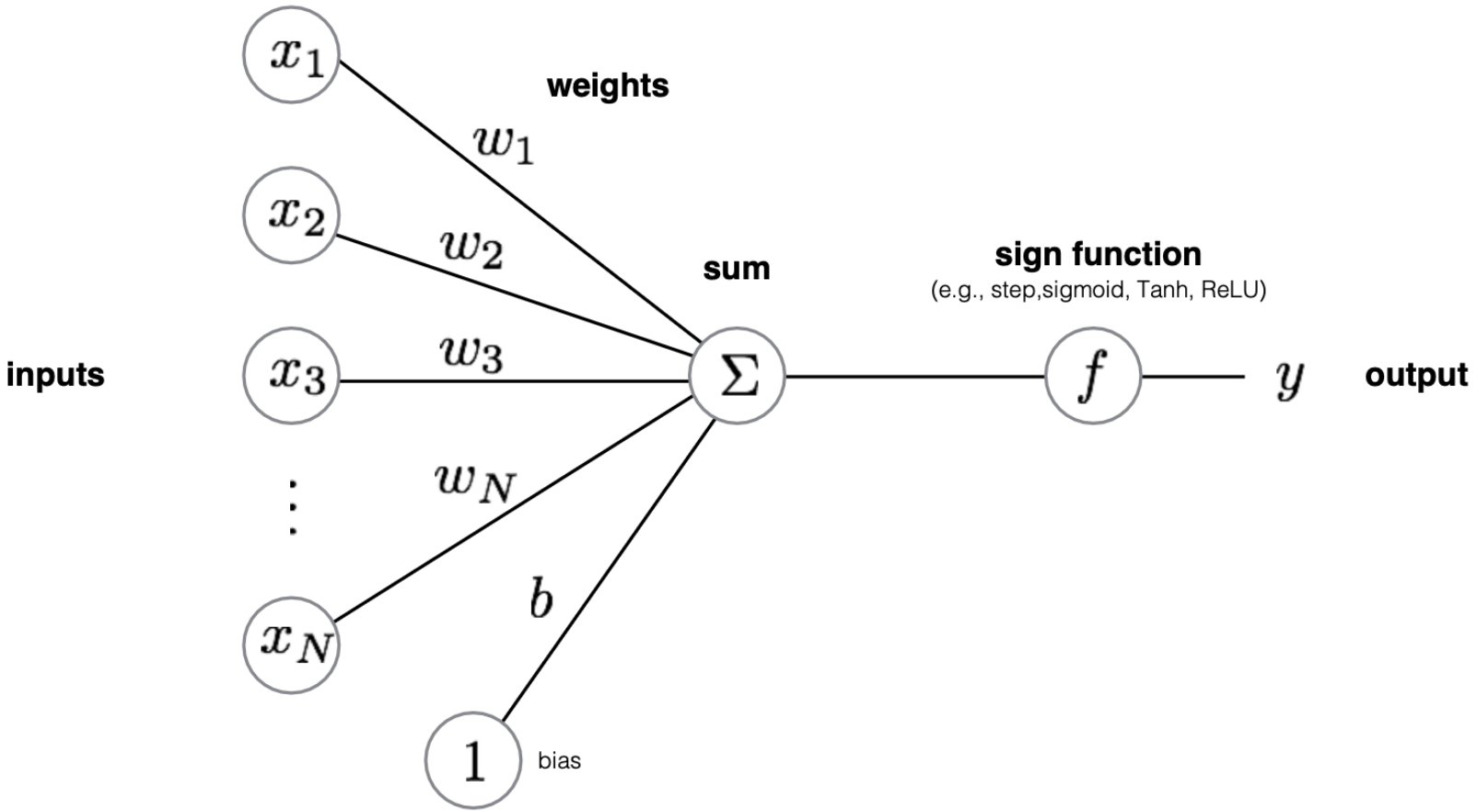
Units can be given biases by introducing an extra input to each unit which always has a value of 1. The weight on this extra input is called the bias and is equivalent to a threshold of the opposite sign. It can be treated just like the other weights.

A unit has a real-valued output, y_j, which is a non-linear function of its total input

$$y_j = \frac{1}{1+e^{-x_j}} \qquad (2)$$

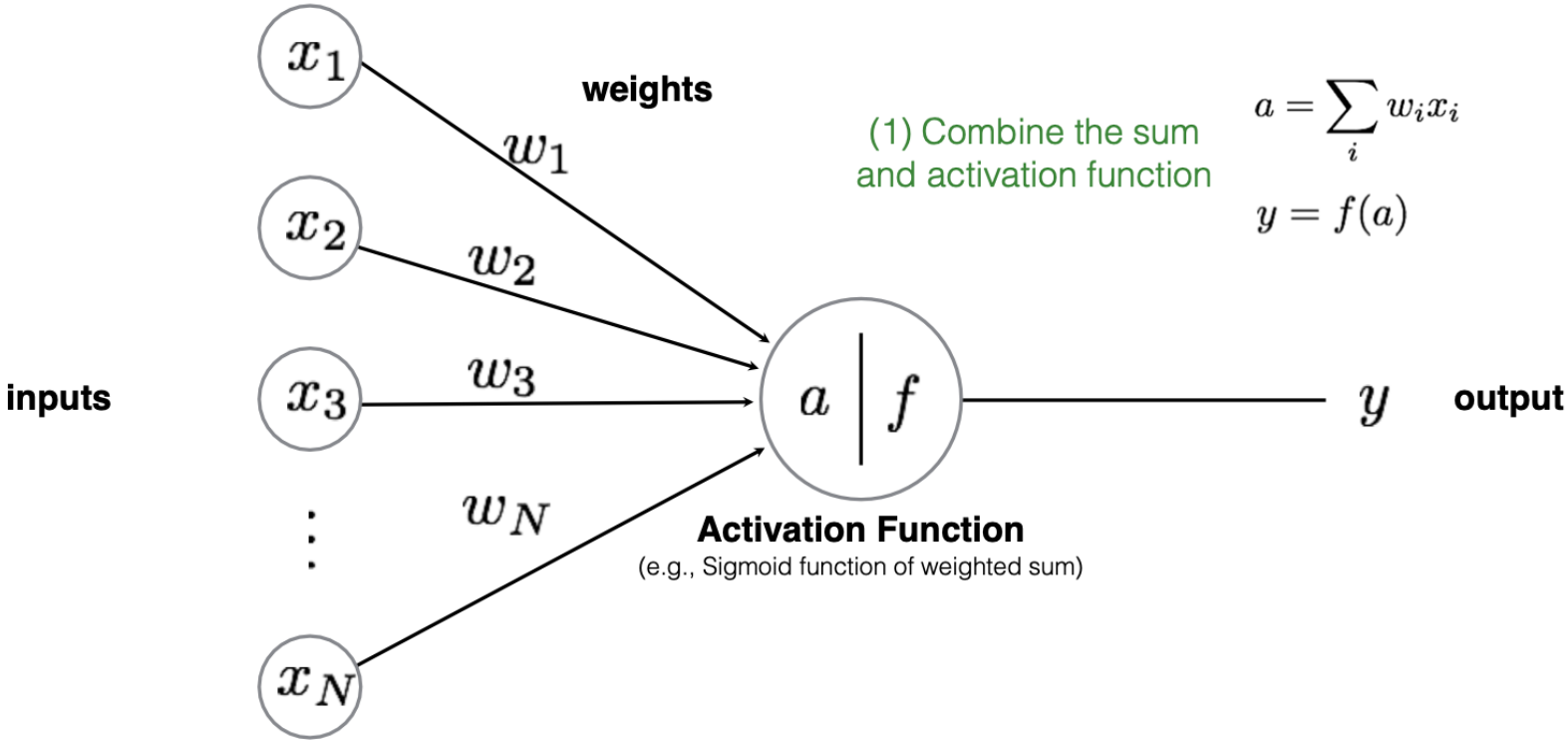
† To whom correspondence should be addressed.

The Perceptron



The Perceptron

- Another representation



Perceptron Training by SGD

1: **function** PERCEPTRON ALGORITHM

2: $\boldsymbol{w}^{(0)} \leftarrow \mathbf{0}$

3: **for** $t = 1, \ldots, T$ **do**

4: RECEIVE$(\boldsymbol{x}^{(t)})$ $\boldsymbol{x} \in \{0,1\}^N$ N-d binary vector

5: $\hat{y}_A^{(t)} = \underset{\text{sign of zero is +1}}{\text{sign}}\left(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle \right)$ Classification result

6: RECEIVE(y^t) $y \in \{1, -1\}$

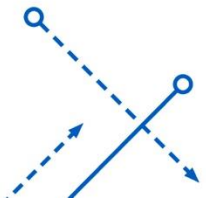
7: $w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$ Update the parameters

Code to train a Perceptron

- Just one line of code!

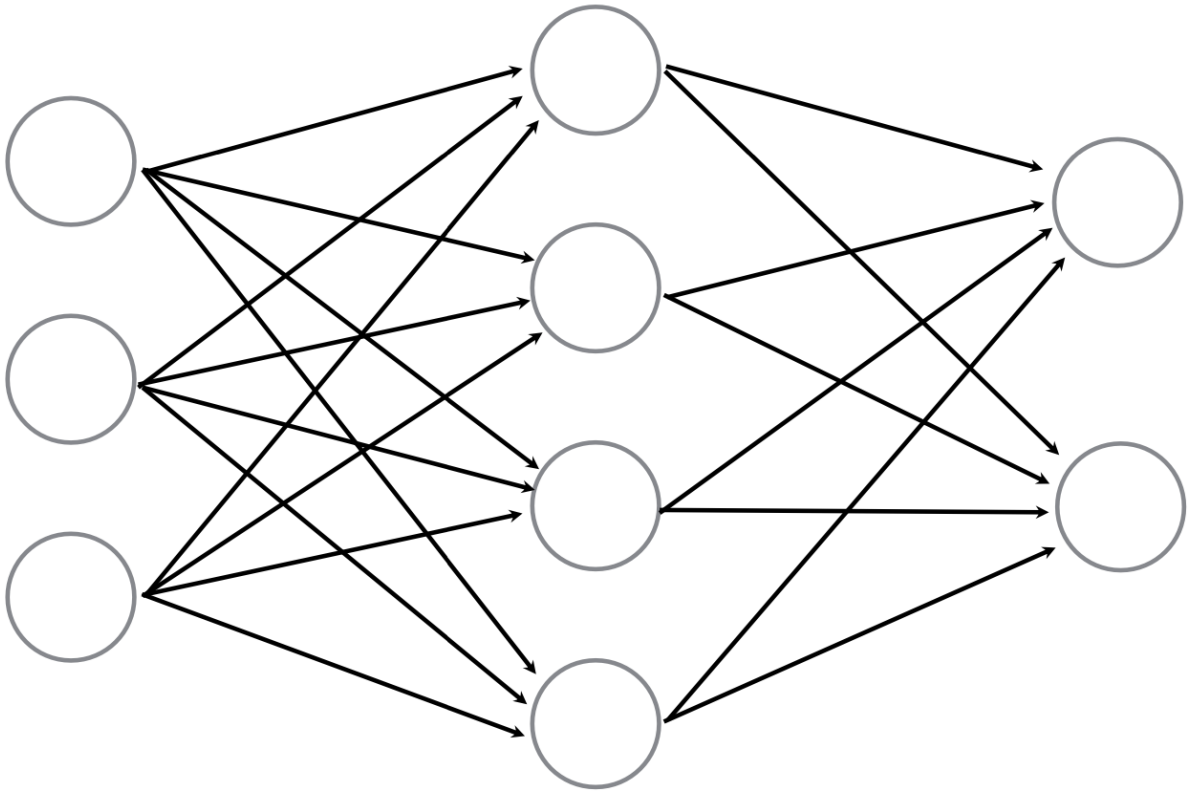
$$\text{for } n = 1 \ldots N$$

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



Neural Network

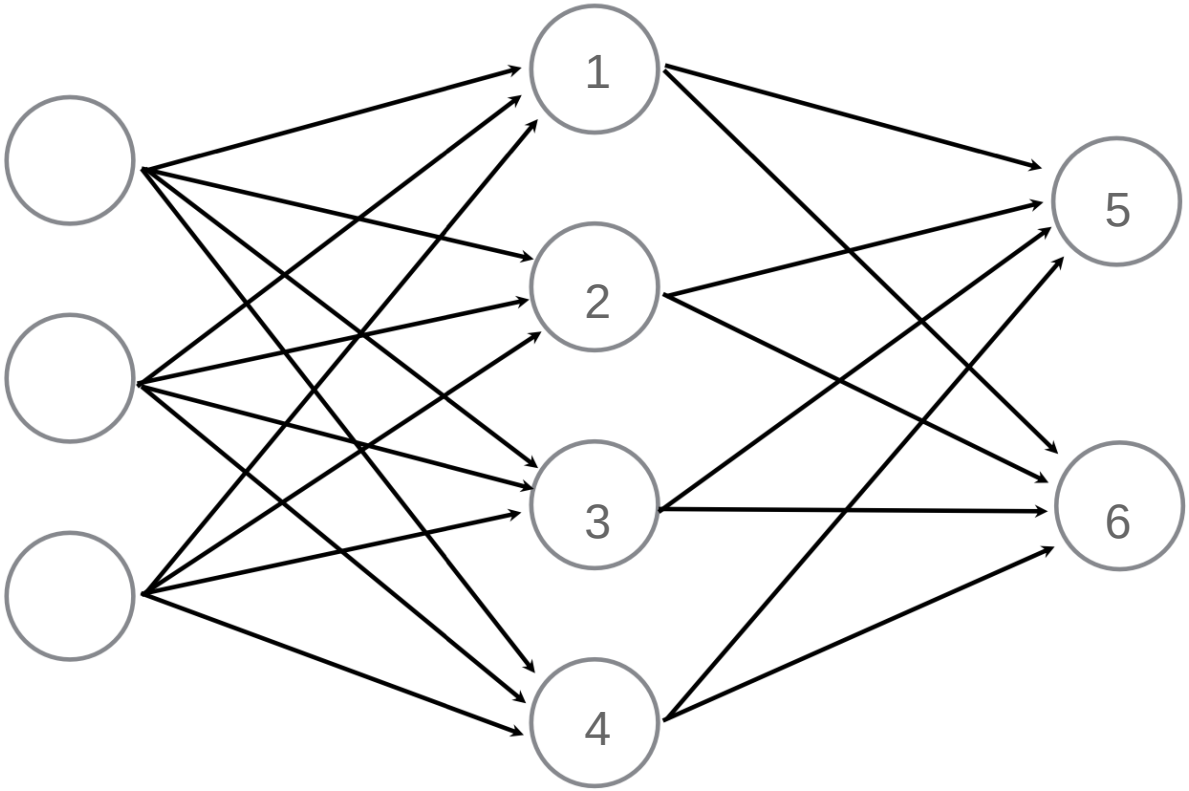
- A collection of connected perceptrons



This is also called Multi-layer perceptron (**MLP**)

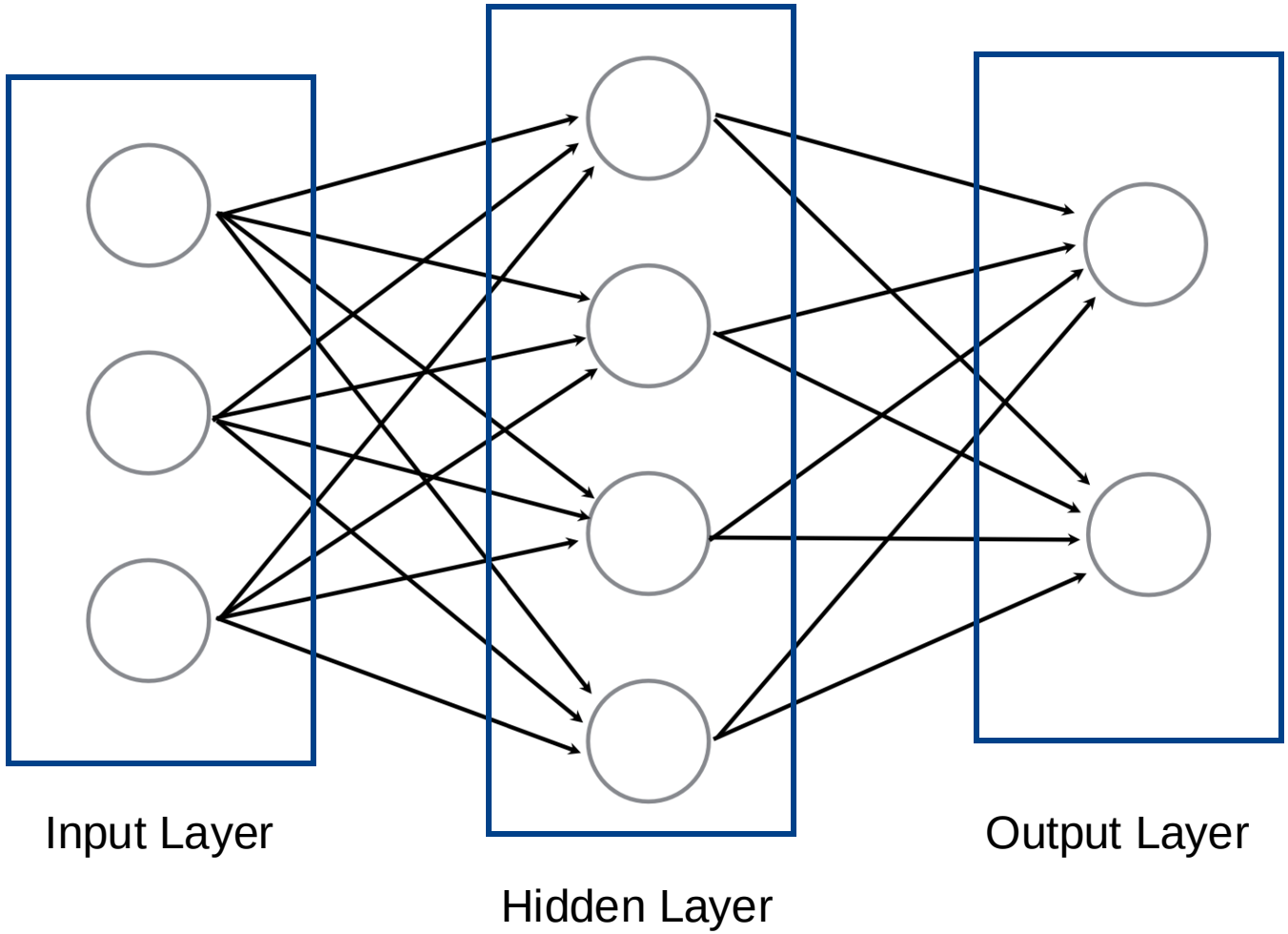
Multi-layered perceptron (MLP)

- How many perceptrons in this MLP?



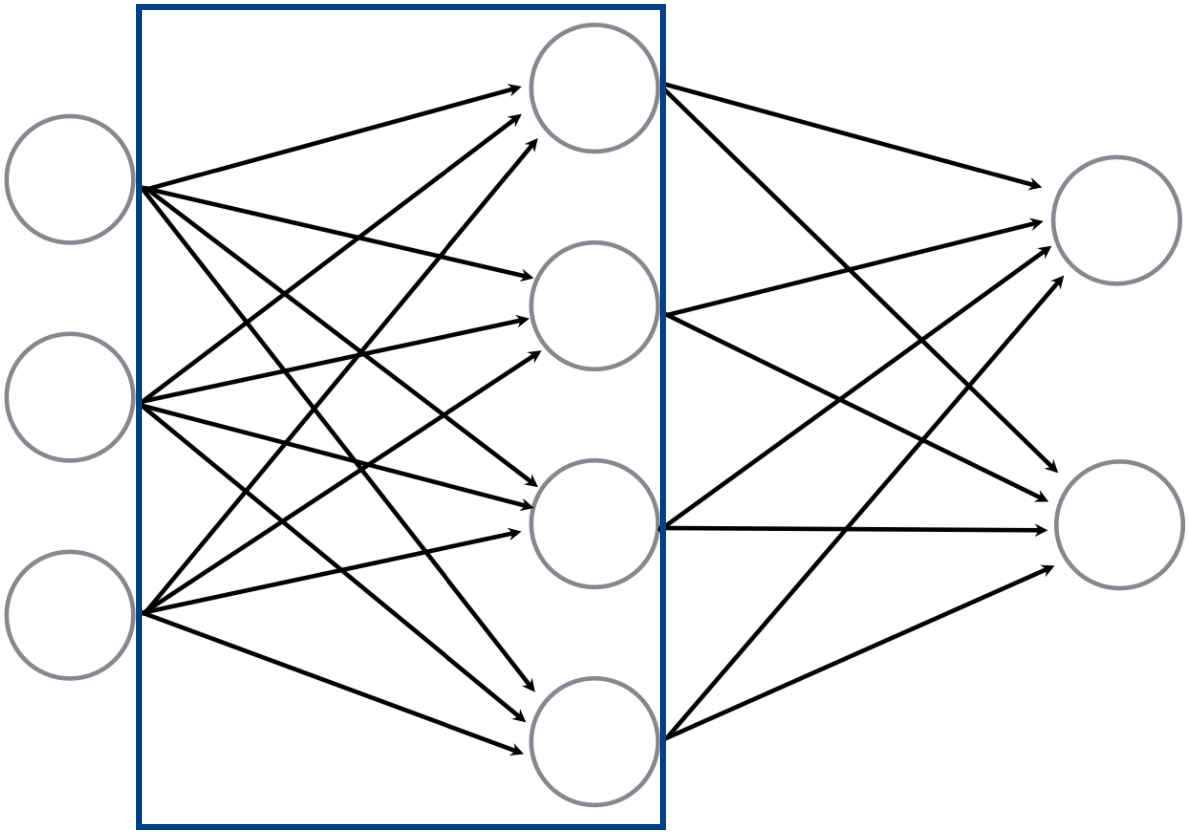
Multi-layered perceptron (MLP)

- Some terminology



Fully Connected Layer

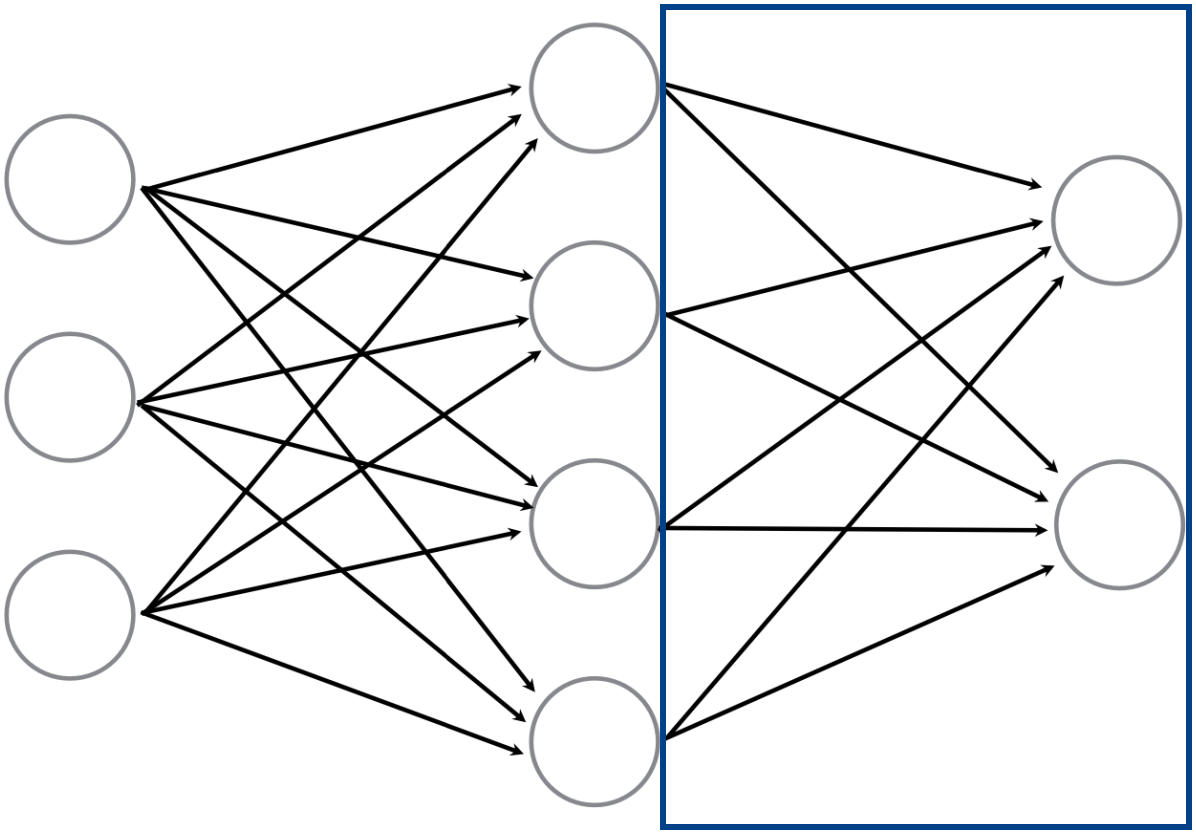
- All pairwise neurons between layers are connected



This is a fully connected Layer

Fully Connected Layer

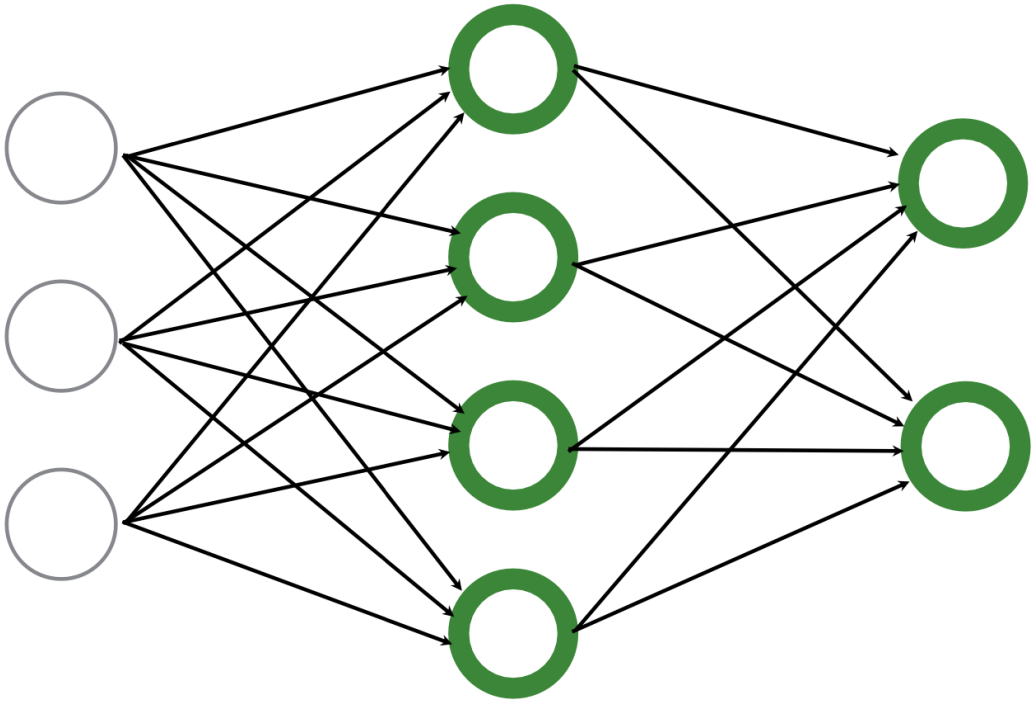
- All pairwise neurons between layers are connected



This is also a fully connected layer

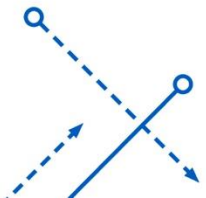
Multi-layered perceptron (MLP)

- How many neurons (perceptrons)? 4 + 2 = 6
- How many weights (edges)? (3 x 4) + (4 x 2) = 20
- How many learnable parameters total? 20 + 4 + 2 = 26



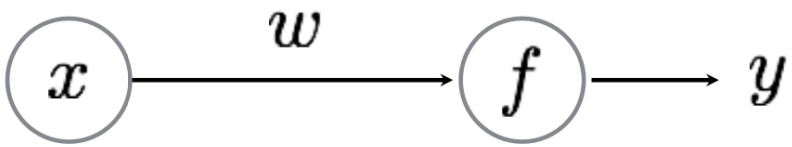
Multi-layered perceptron (MLP)

- Performance usually tops out at 2-3 layers, deeper networks don’t really improve performance.
- We often use convolutional networks to extract features for images, but use MLP as the final decision layer.



Learning a perceptron

- Consider the smallest perceptron, $y = wx$,



line equation
linear regression

- Given a set of samples and a perceptron

$$\{x_i, y_i\}$$

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

- Estimate the parameters of the Perceptron

$$w$$

Gradient Descent

- Given a set of samples

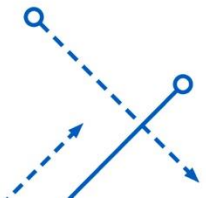
$$\{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

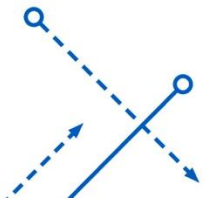
- Modify weight w such that $\hat{y}$ gets "**closer**" to y.

Perceptron parameter **Perceptron output** ***what does this mean?*** **true label**



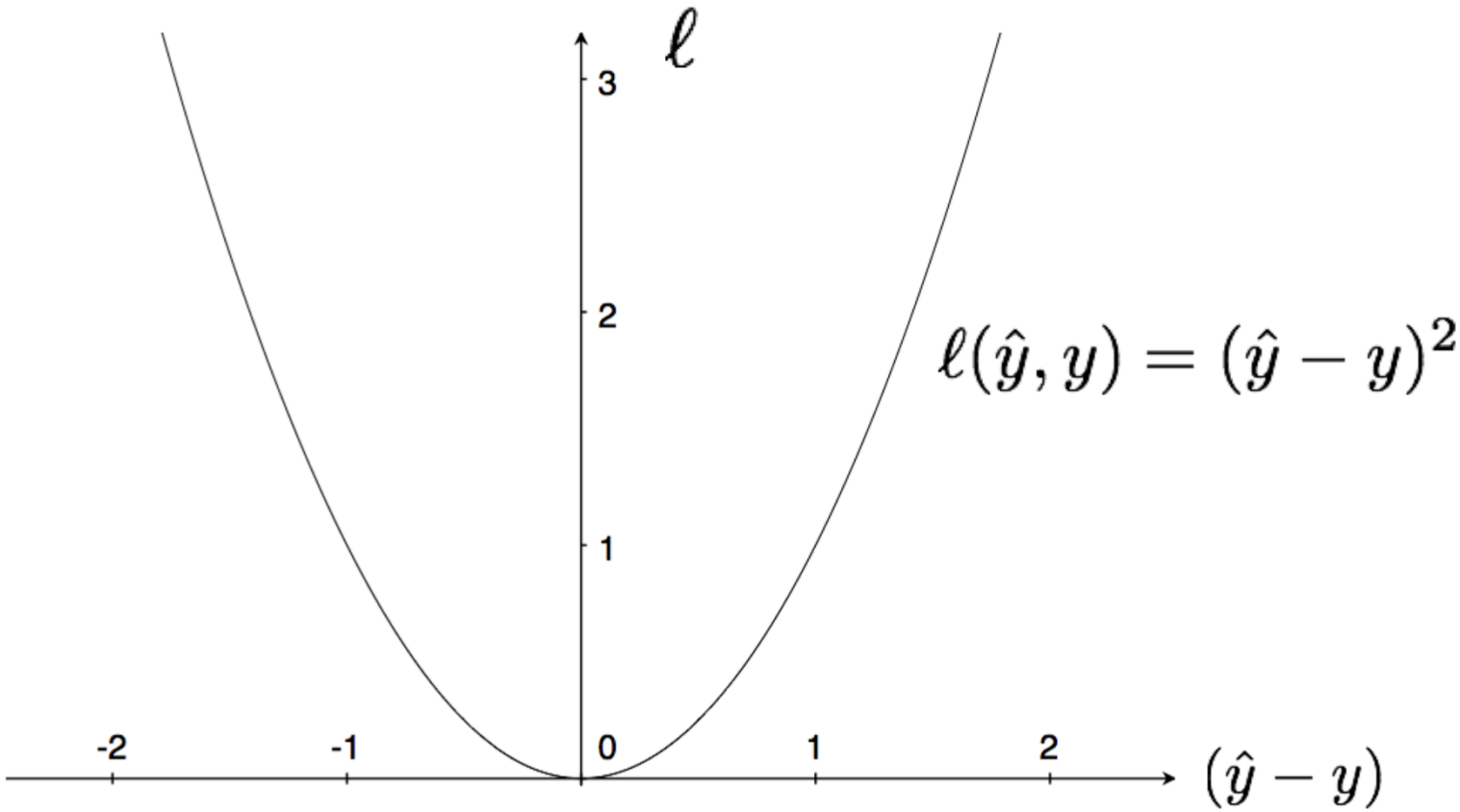
Loss function

- Before dive into gradient descent, we need…
- Loss Function
  - defines what is means to be close to the true solution
- We need to chose the loss function!
  - some are better than others depending on what want to do.



Loss function:

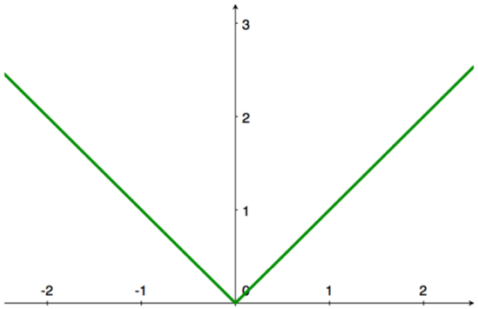
- Squared Error (L2): a popular loss function.



Loss functions

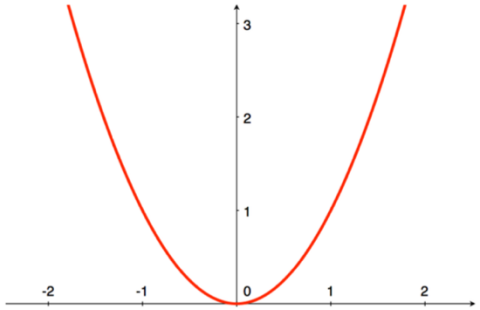
L1 Loss

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



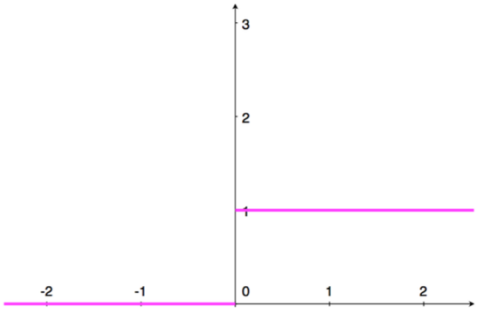
L2 Loss

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



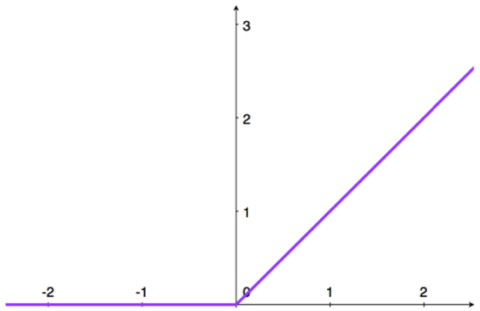
Zero-One Loss

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



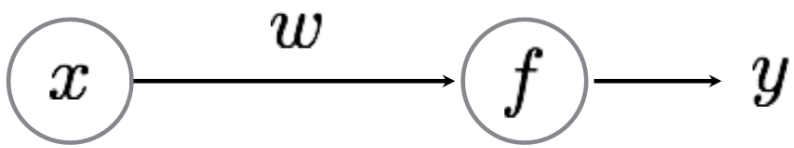
Hinge Loss

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



Training perceptron

- Back to the smallest perceptron



$$\hat{y} = wx$$

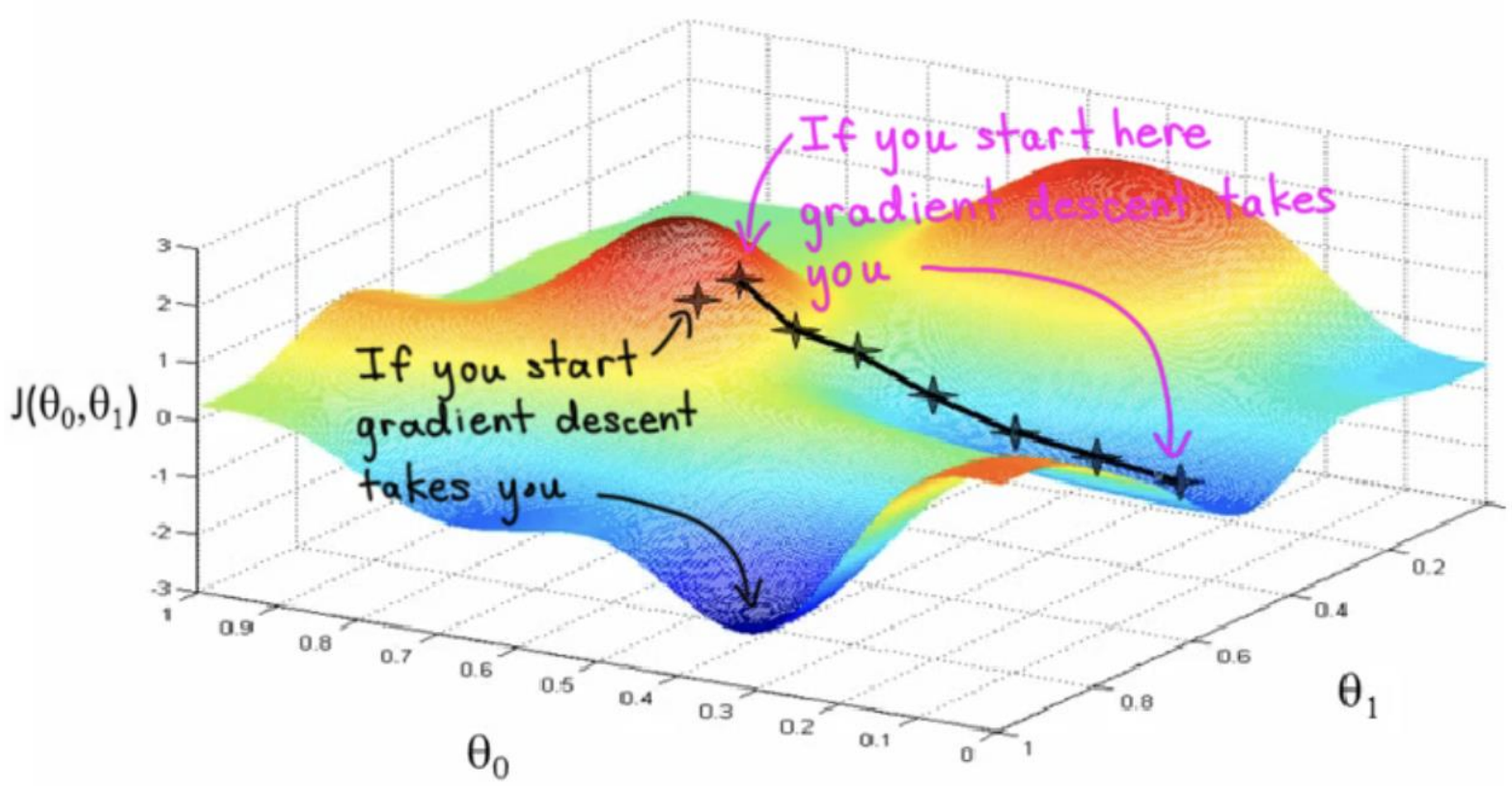
function of ONE parameter!

Recall:

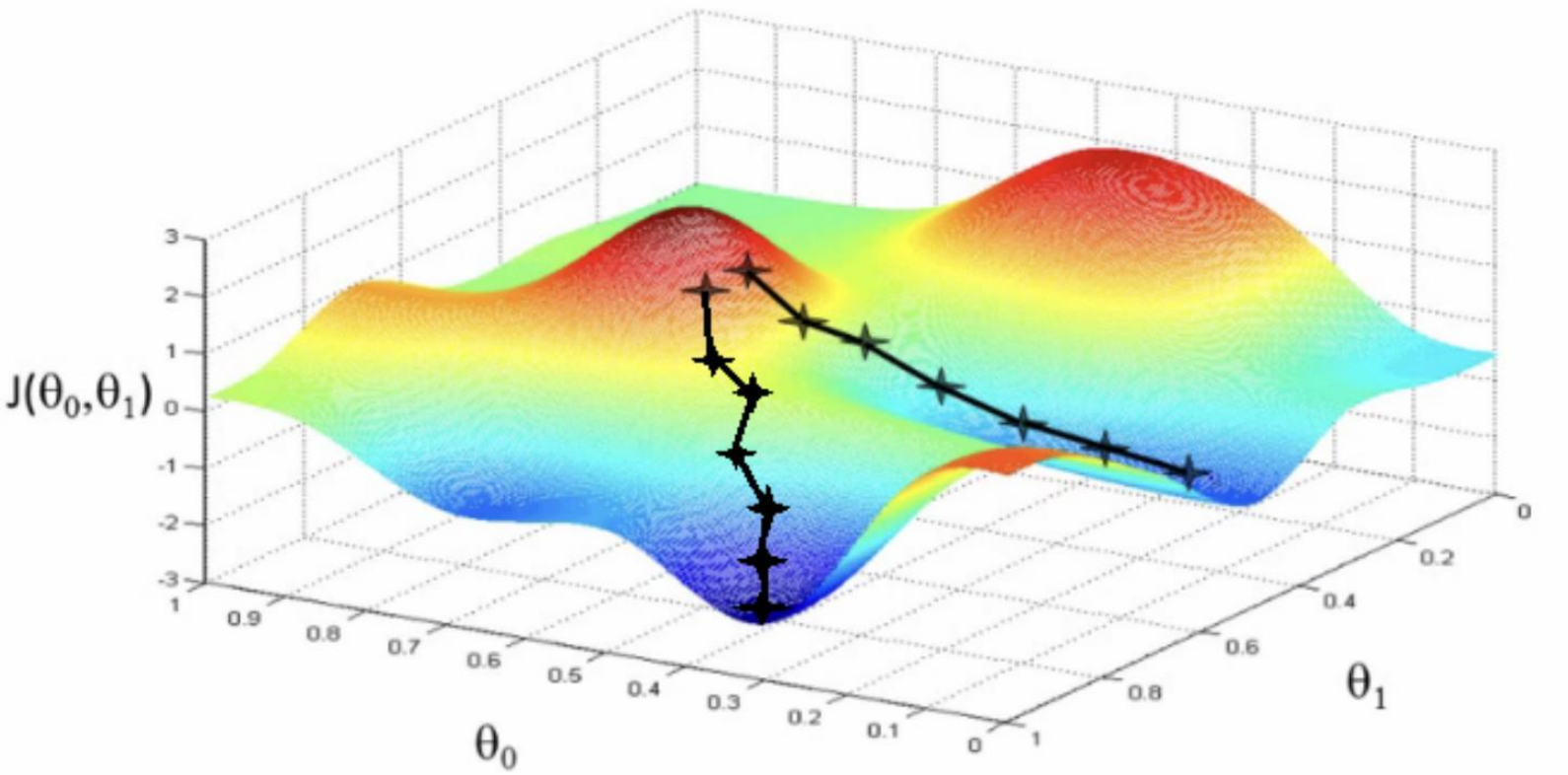
Code to train your perceptron:

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

Visualizing Gradient Descent

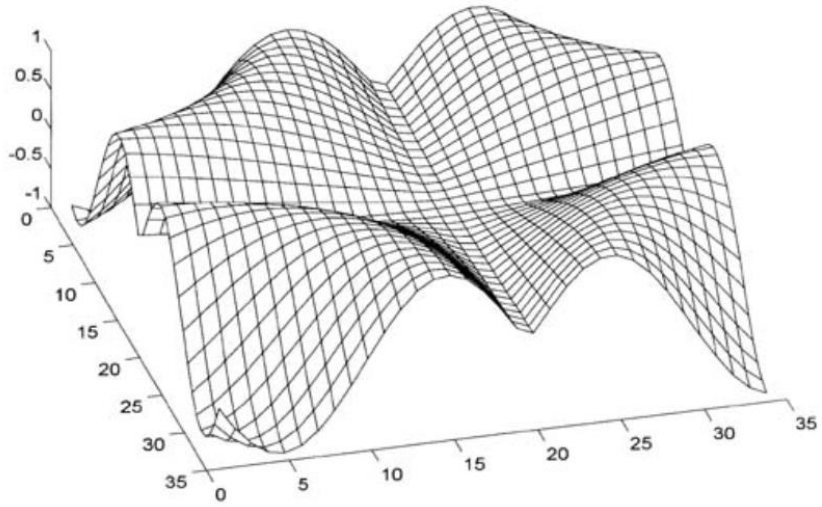


Visualizing Gradient Descent

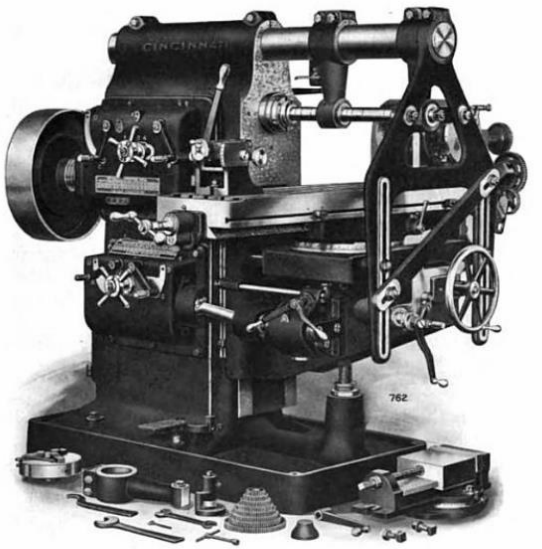


Partial Derivatives

- Tell us how much one variable affects function value.
- Two ways to think about them.



Slope of a function

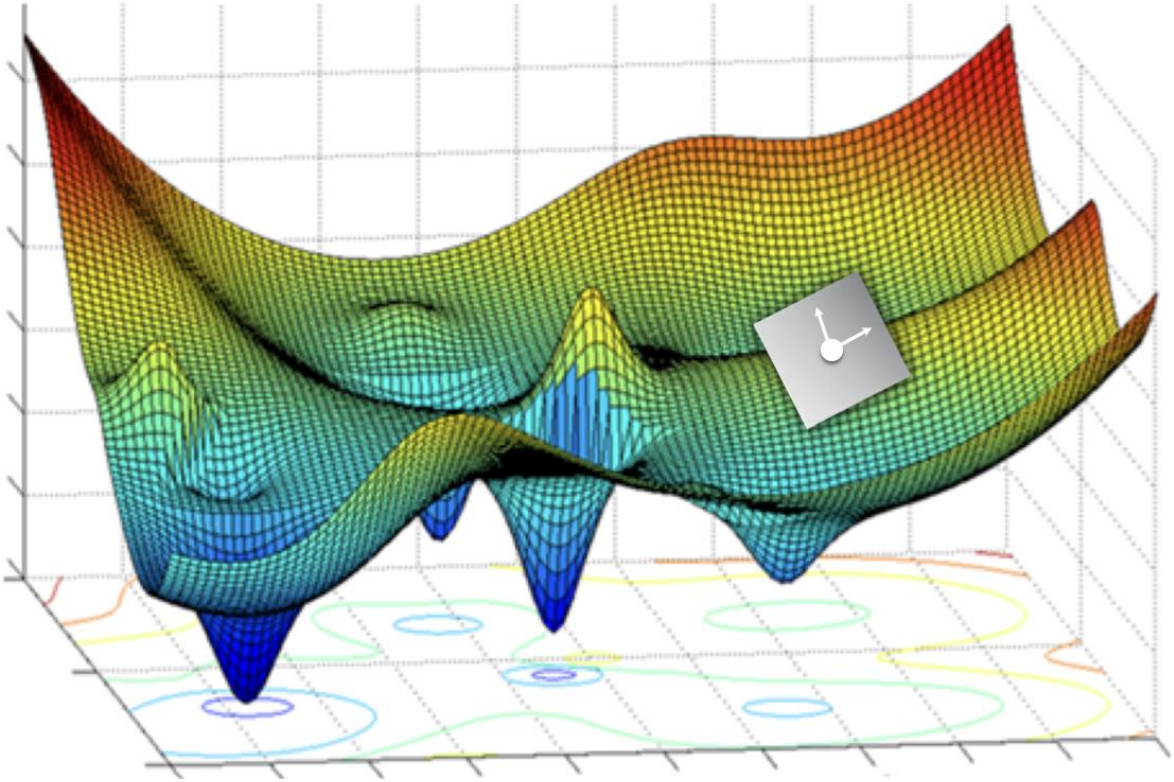


Knobs on a machine

Slope of a function

- Describes the slope around a point

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



Knobs on a machine

input $\boldsymbol{x}$

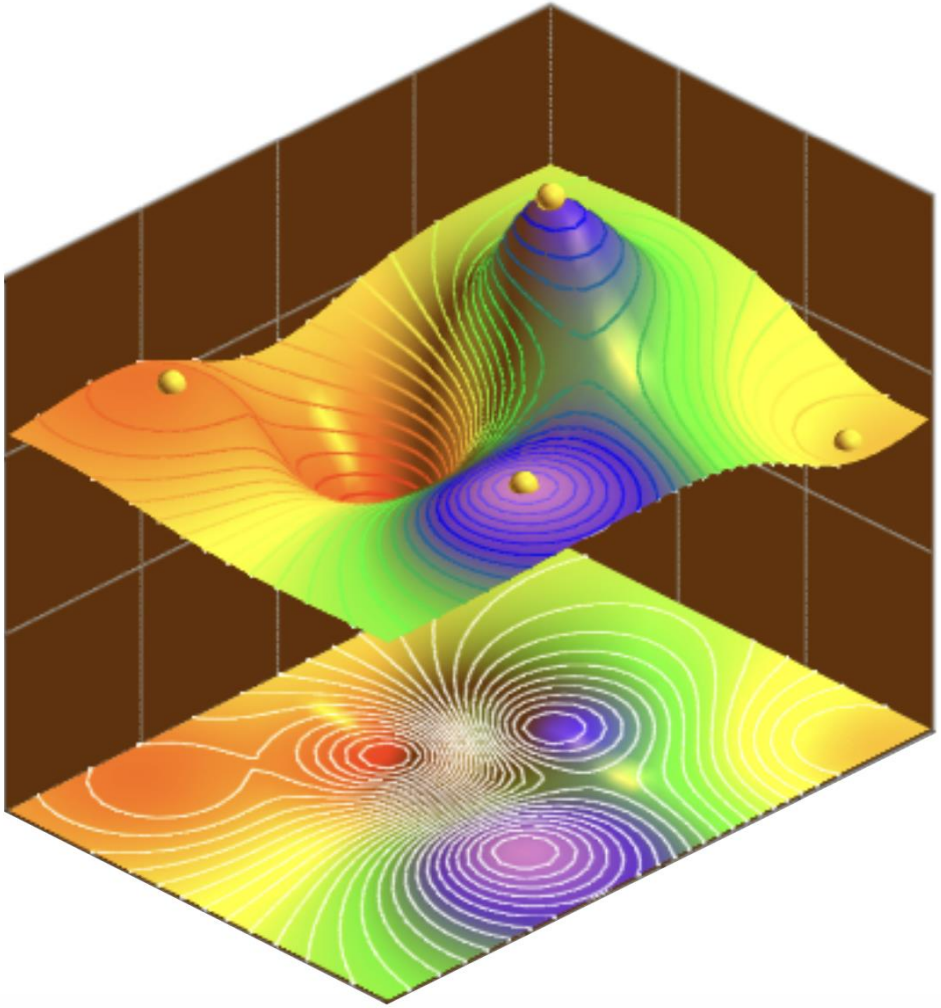
output $f(\boldsymbol{x}; \boldsymbol{w})$

describes how each 'knob' affects the output

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

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

Contour map visualization

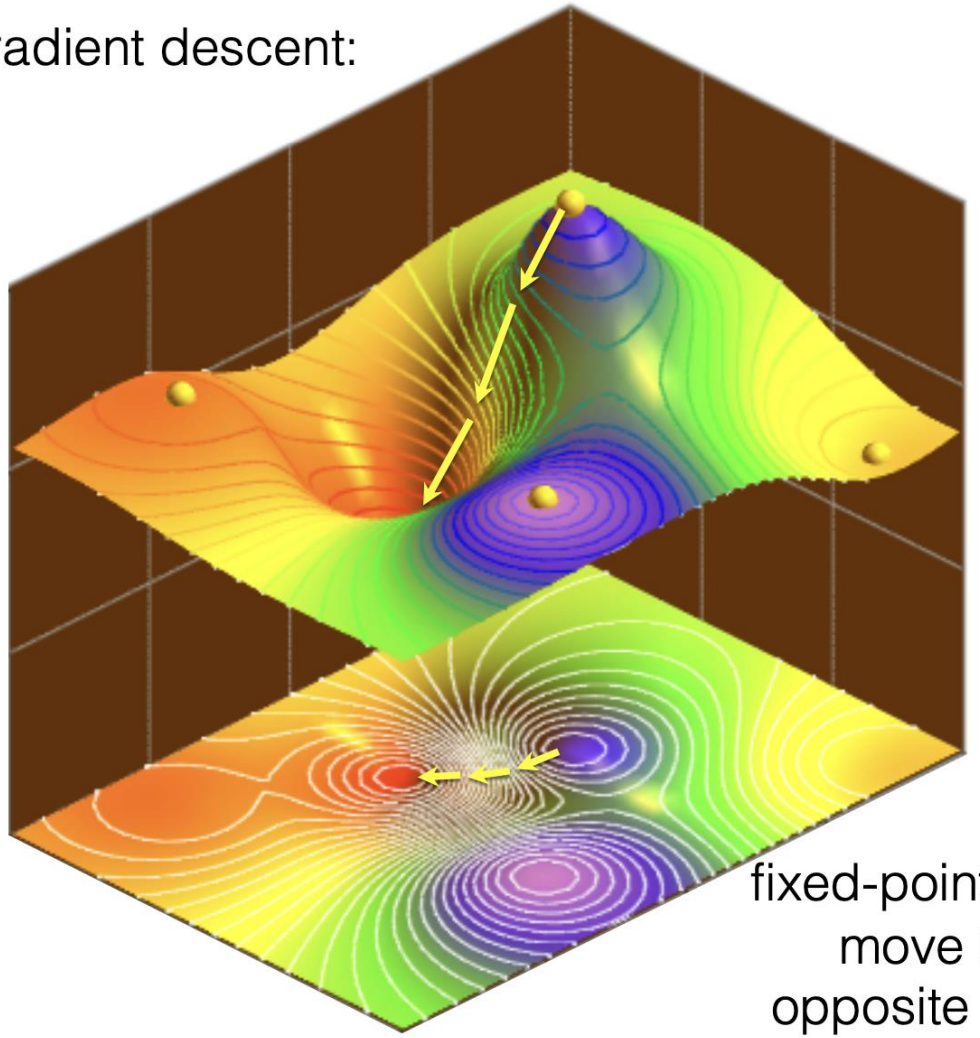


Loss function in 3D

Contour map visualization of the same thing

Gradient descent

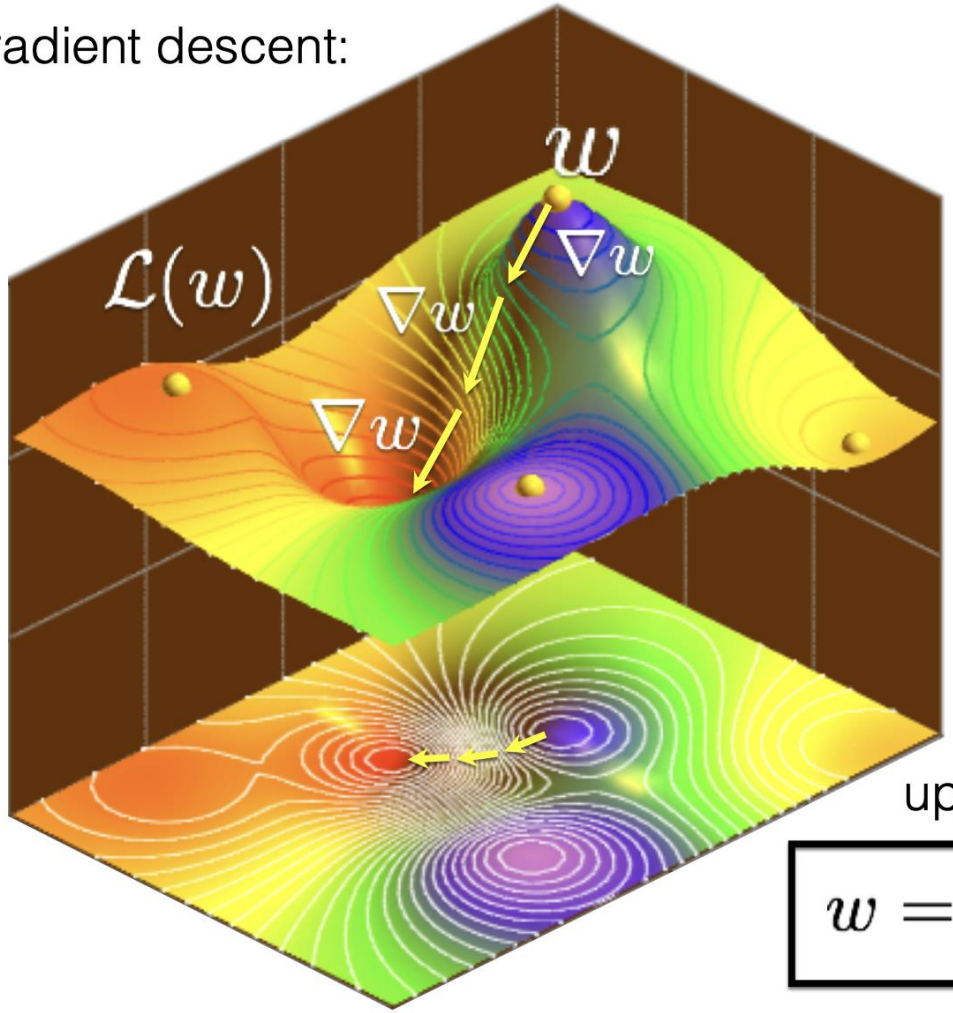
Gradient descent:



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

Gradient descent

Gradient descent:



update rule:

$$w = w - \nabla w$$

SAIR

Spatial AI & Robotics Lab

MULTI-LAYERED PERCEPTRON

Back-Propagation

Training of perceptron

- Back to the smallest perceptron

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

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

This is just gradient descent, that means…

this should be the gradient of the loss function

- Now where does this come from?

Training of perceptron

- We have the loss function:

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

$\frac{d\mathcal{L}}{dw}$ …is the rate at which **this** will change…

$$\hat{y} = wx$$

Per weight parameter

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

- Then the weight update for gradient descent is:

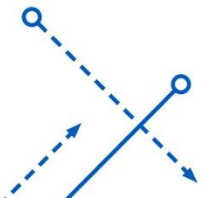
$$w = w - \nabla w$$ move in direction of negative gradient

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

Stochastic Gradient Descent

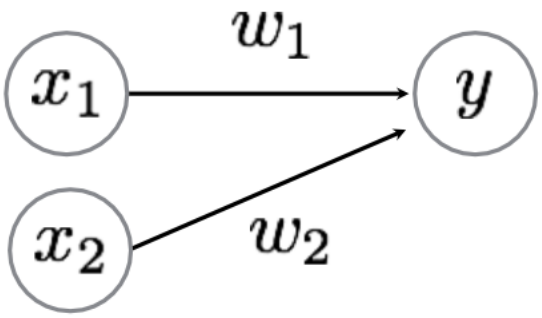
Smallest perceptron: For each sample $\{x_i, y_i\}$

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



A little more complicated

- Consider the 2nd smallest perceptron
  - Function of two parameters!

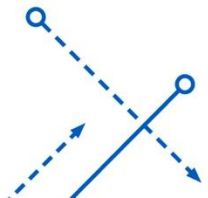


Stochastic Gradient Descent

2nd smallest perceptron: For each sample $\{x_i, y_i\}$

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

we just need to compute partial derivatives for this network



Back-Propagation

- We have partial derivative now.

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

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

Gradient Update

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

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

Stochastic Gradient Descent

2nd smallest perceptron: For each sample $\{x_i, y_i\}$

1. Predict
    1. Forward pass $\hat{y} = f_{PER}(x_i; w)$
    2. Compute Loss $\mathcal{L}_i = \frac{1}{2}(y_i - \hat{y})^2$
2. Update

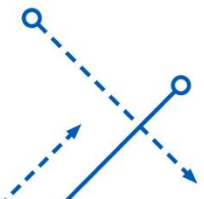
two BP lines now

    1. Back Propagation

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

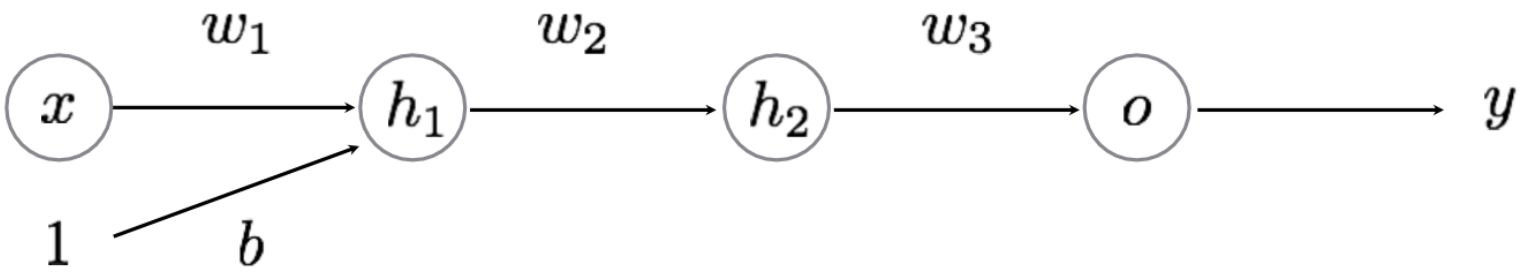
    2. Gradient update

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

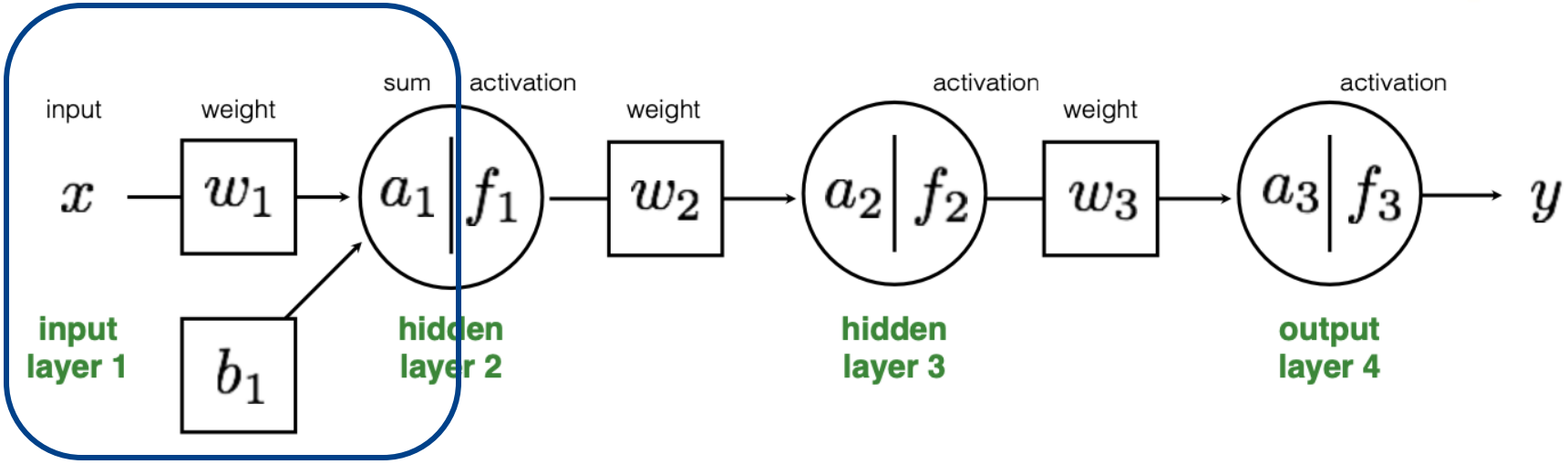


Multi-layer perceptron

- We haven't seen a lot of 'propagation' yet because our perceptrons only had one layer
- Multi-layer perceptron
  - Function of four parameters and four layers

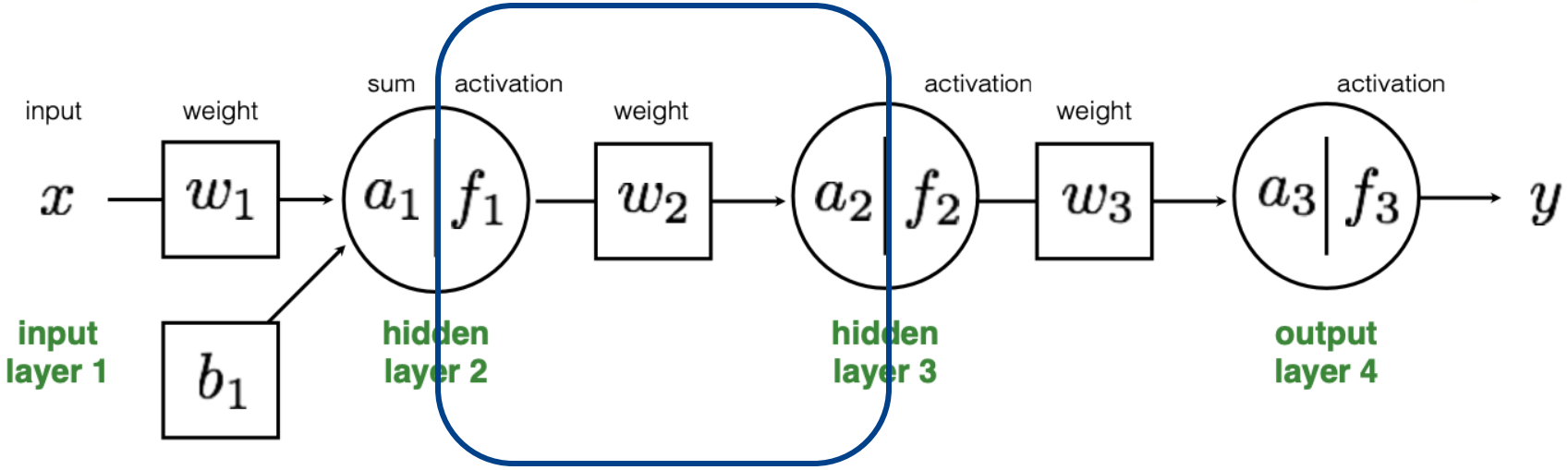


Multi-layer perceptron



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

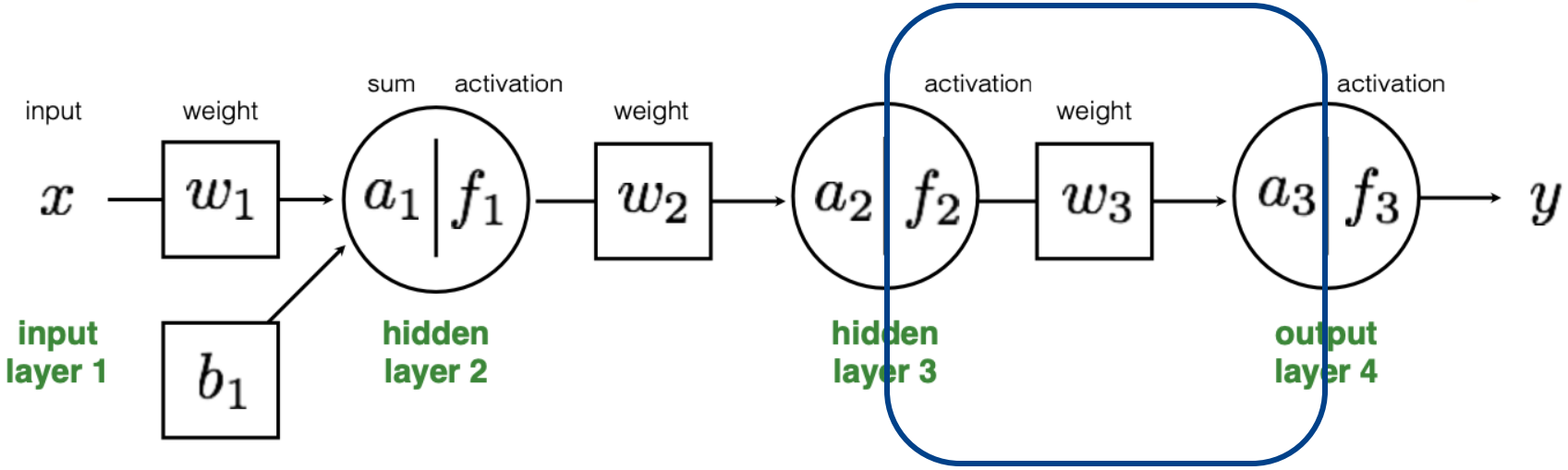
Multi-layer perceptron



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

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

Multi-layer perceptron

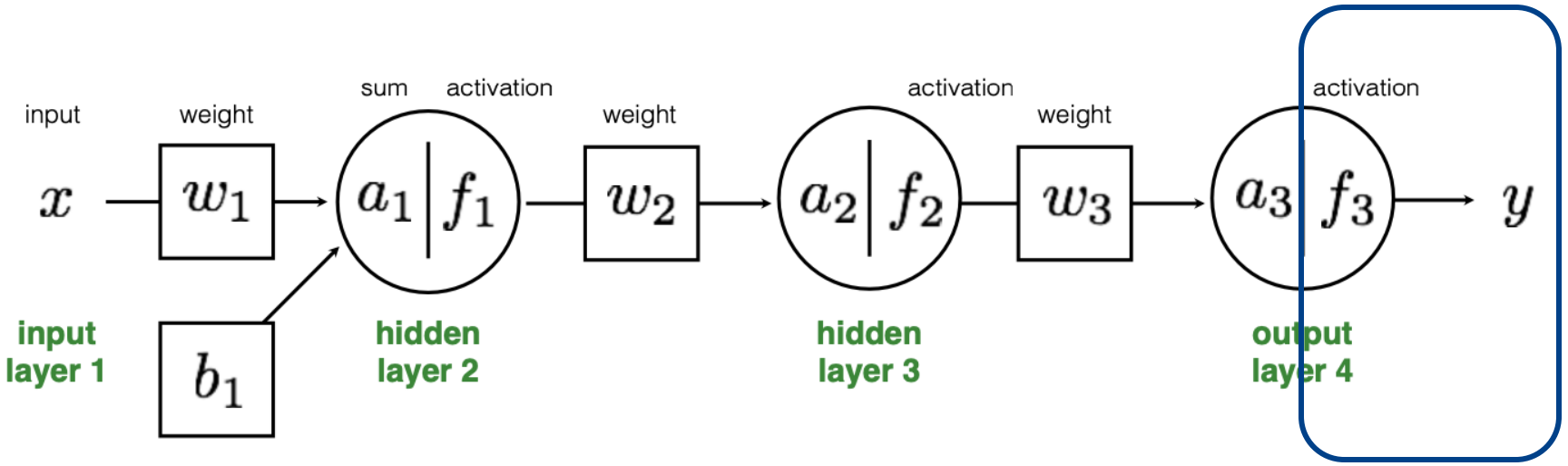


$$a_1 = w_1 \cdot x + b_1$$

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

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

Multi-layer perceptron



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

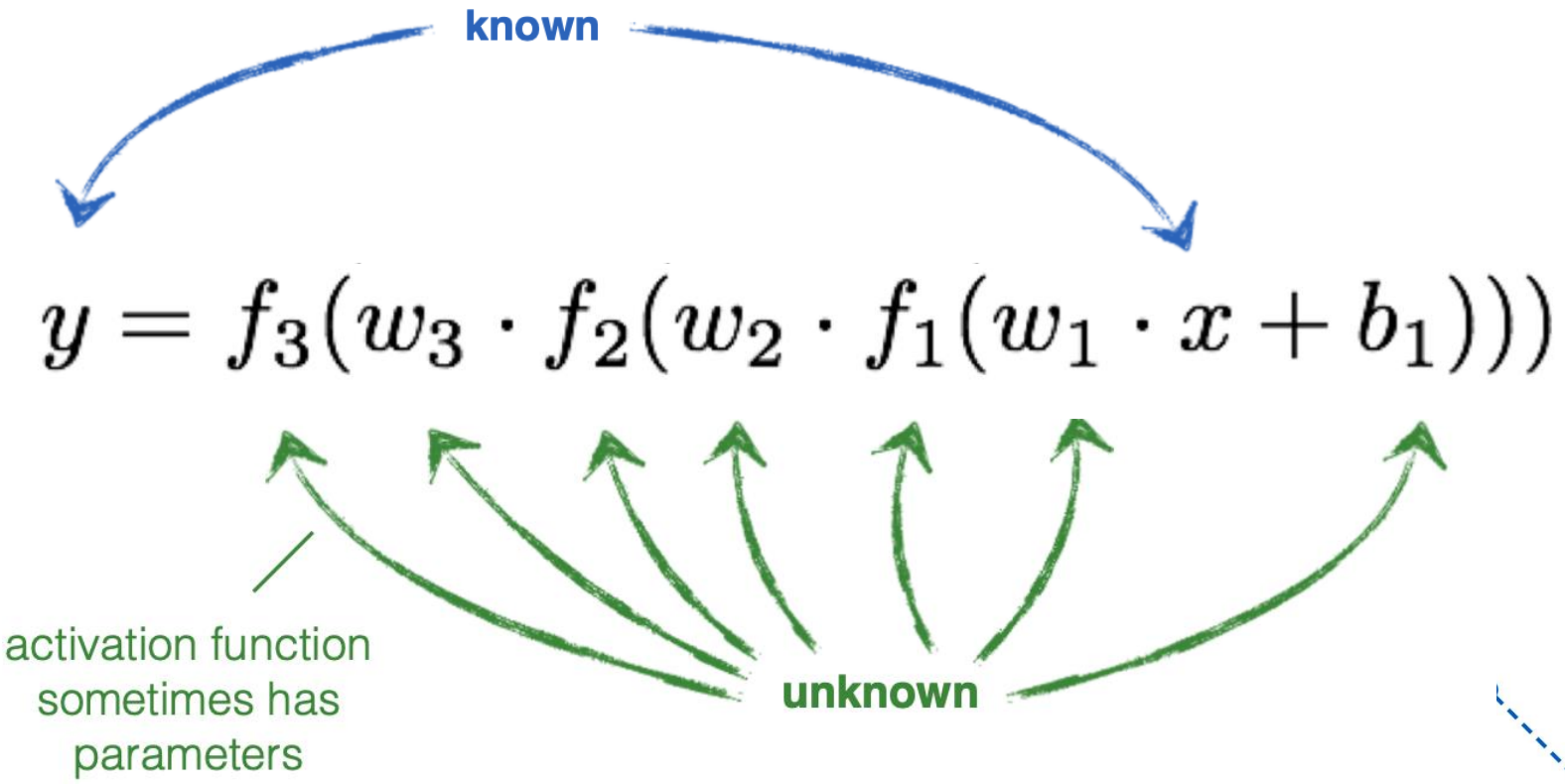
$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

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

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

Multi-layer perceptron

- Entire network can be written out as one equation
  - We need to train the network:
  - What is known? What is unknown?



Learning an MLP

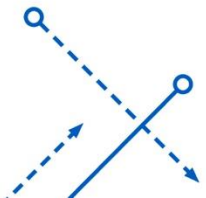
- Given a set of samples and an MLP

$$\{x_i, y_i\}$$

$$y = f_{MLP}(x; w)$$

- Estimate the parameters of the Perceptron

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

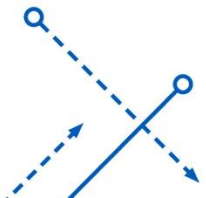


Stochastic Gradient Descent

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

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

vector of parameter update equations

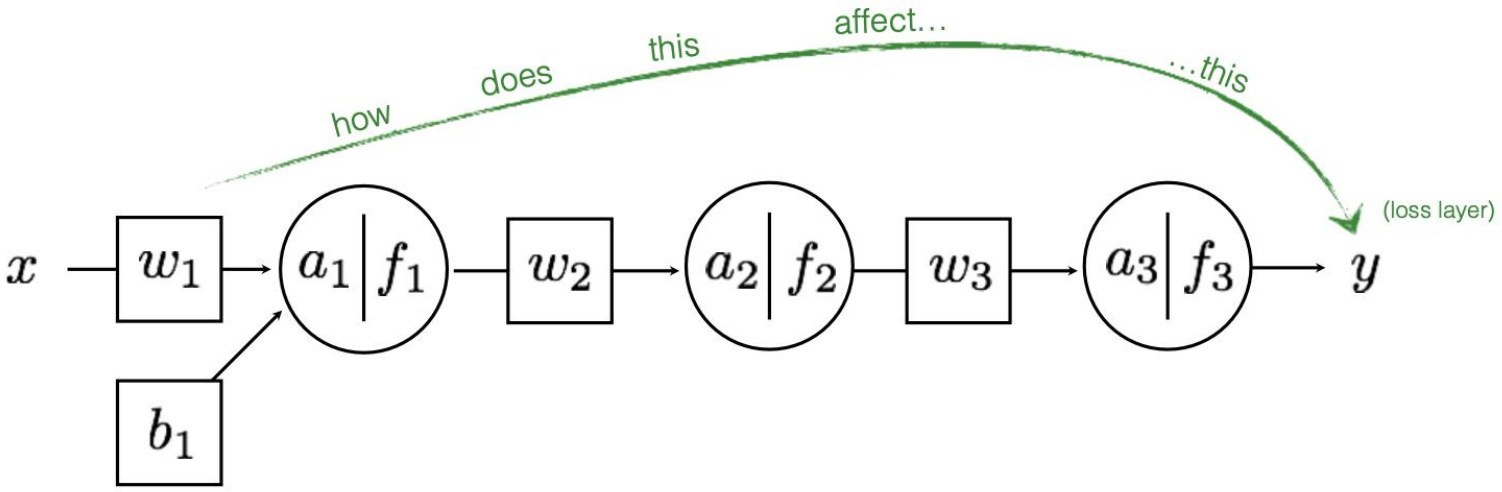


Stochastic Gradient Descent

- 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



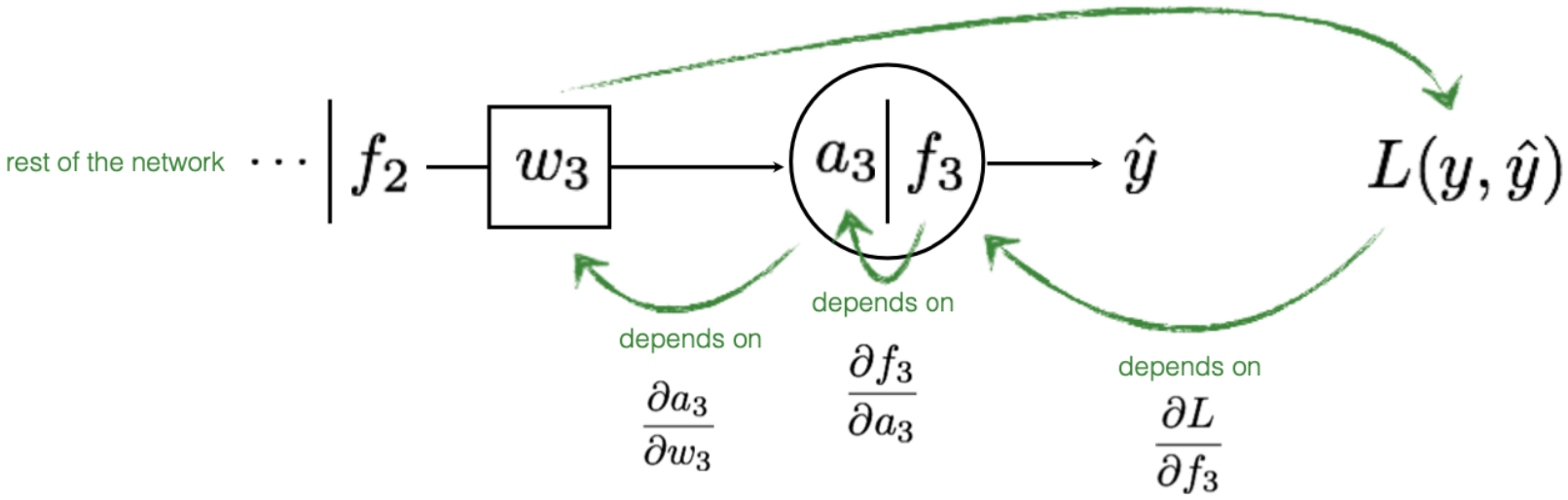
So, how do you compute it?

The Chain Rule

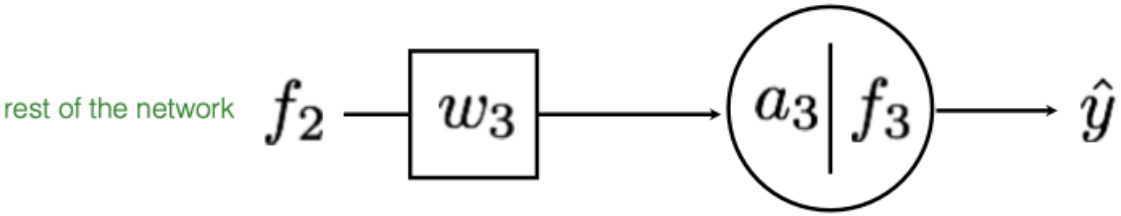
- According to the chain rule...

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

- The effect of weight on loss function: $\frac{\partial L}{\partial w_3}$



The Chain Rule

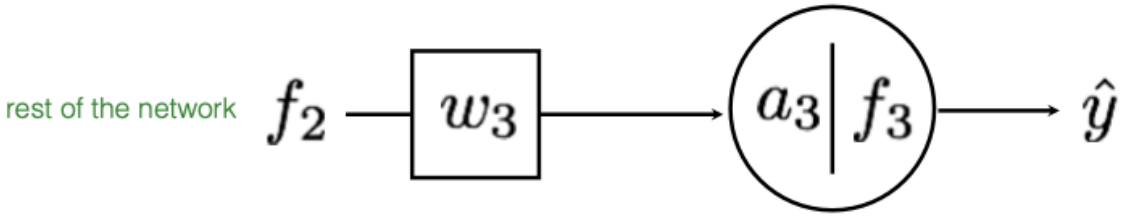


$$\mathcal{L} = \frac{1}{2}(y - \hat{y})^2$$

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

Chain Rule!

The Chain Rule



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

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

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

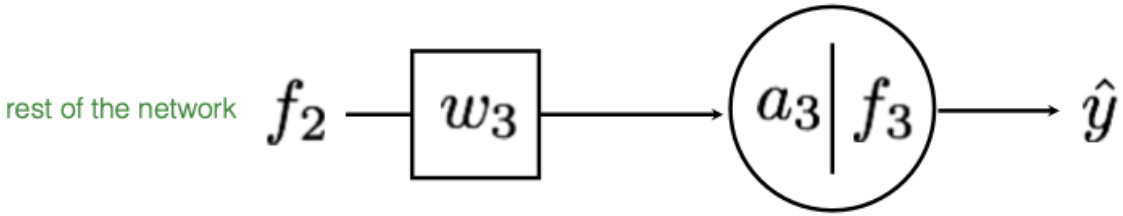
$$S(x) = \frac{1}{1 + e^{-x}}$$

Let's use a Sigmoid function

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

Just the partial derivative of L2 loss

The Chain Rule

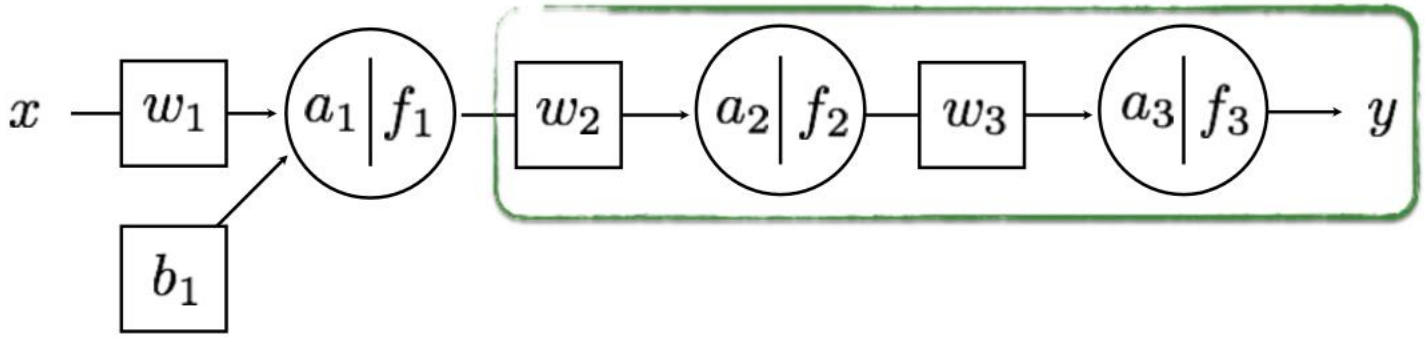


$$\mathcal{L} = \frac{1}{2}(y - \hat{y})^2$$

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

The Chain Rule

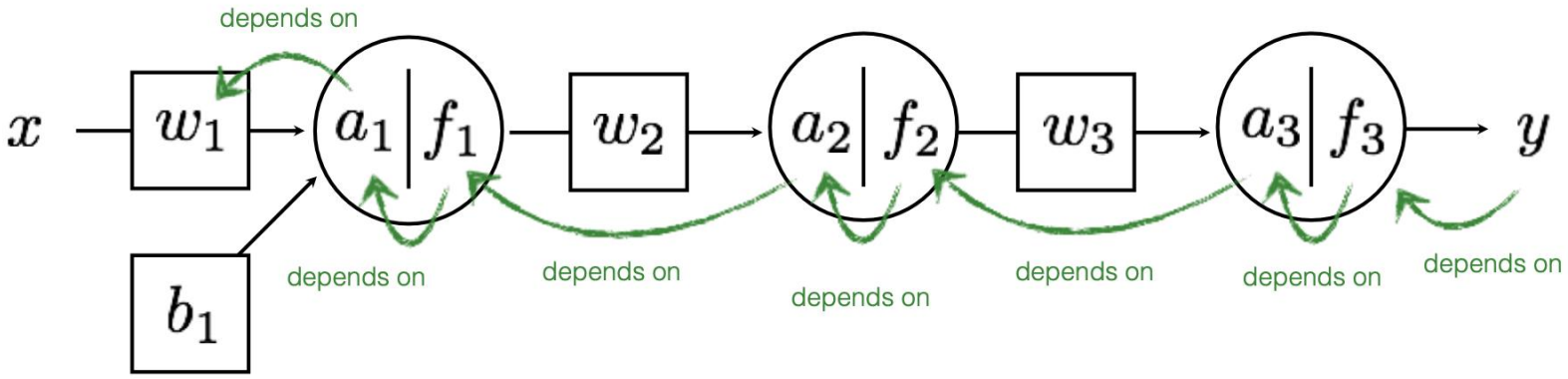
- Back-propagation



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

already computed.
re-use (propagate)!

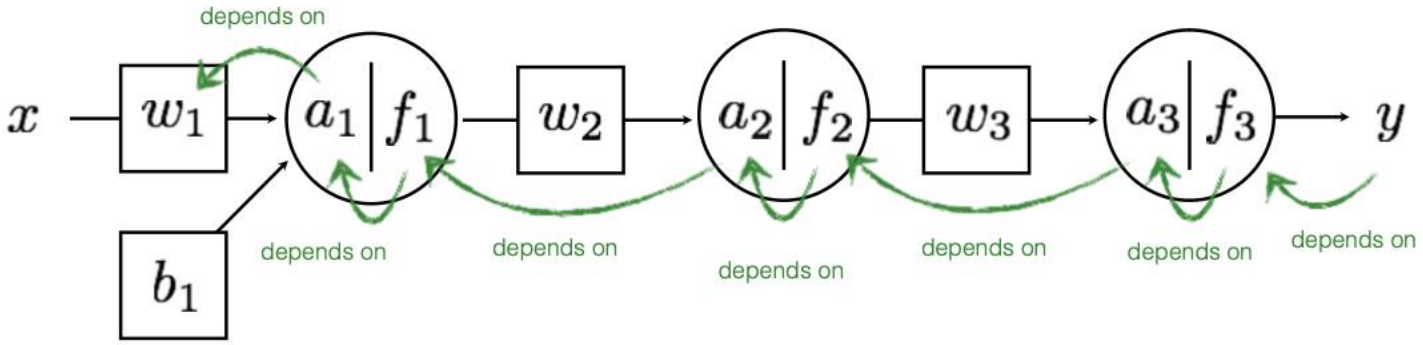
The Chain Rule



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

already computed.
re-use (propagate)!

Back-propagation



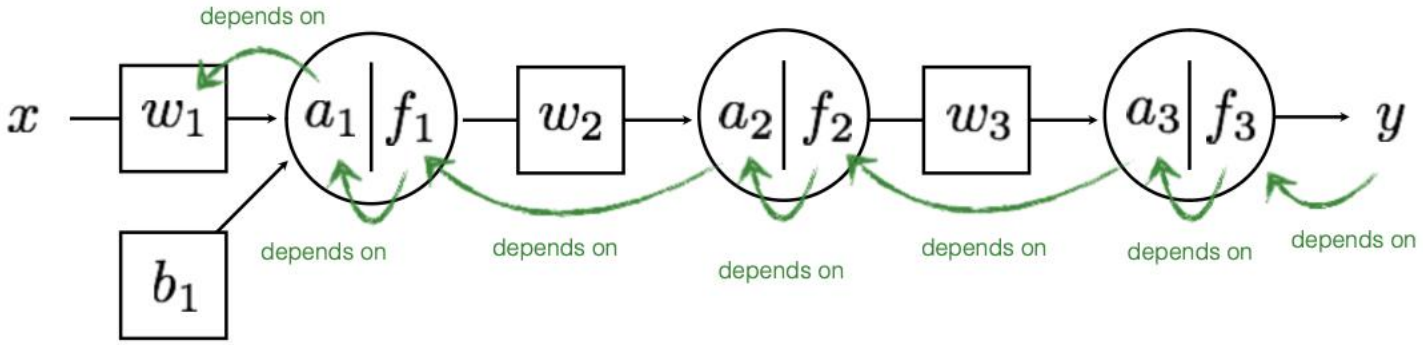
$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

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

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

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

Back-propagation



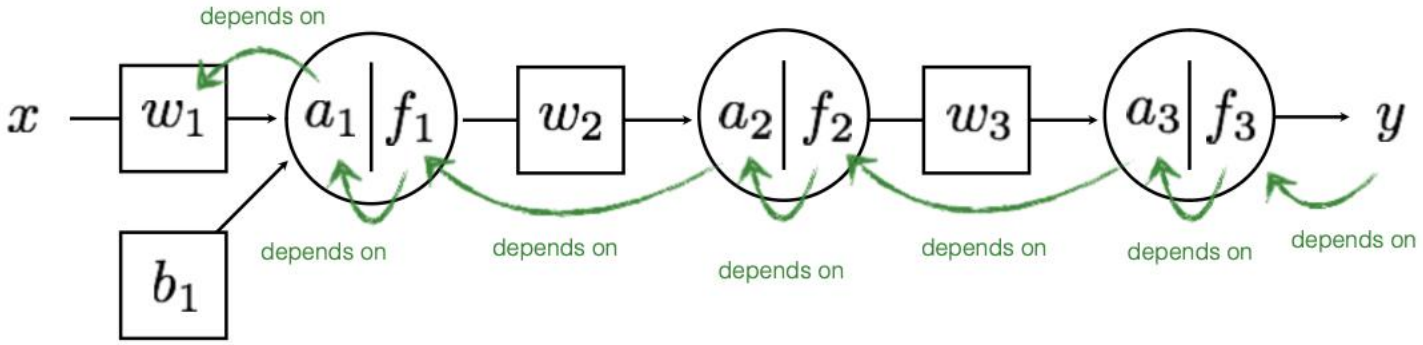
$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

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

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

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

Back-propagation



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

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

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

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

Stochastic Gradient Descent

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

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

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

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

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

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

Stochastic Gradient Descent

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

1. Predict
    1. Forward pass
    2. Compute Loss

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

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

2. Update
    1. Back Propagation
    2. Gradient update

$$\frac{\partial \mathcal{L}}{\partial \theta}$$

vector of parameter partial derivatives

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

vector of parameter update equations

Other frequently used loss function

- Cross-entropy loss

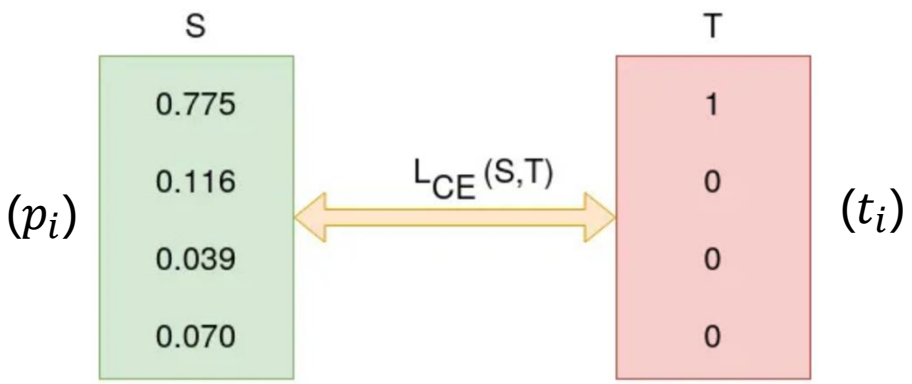
Softmax function

$$p\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

$$L_{\mathrm{CE}} = -\sum_{i=1}^{n} t_i \log(p_i), \quad \text{for n classes,}$$

where t_i is the truth label and p_i is the Softmax probability for the i^{th} class.

- Target as one-hot vector



Logits(S) and one-hot encoded truth label(T) with Categorical Cross-Entropy loss function used to measure the 'distance' between the predicted probabilities and the truth labels. (Source: Author)

Why do we call it cross-entropy?

- We need to define Information.
  - High Information (surprising): low probability
  - Low Information (unsurprising): high probability
- We need map probability $(0, 1)$ to information $(\infty, 0)$
- So, we define information as

$$I(x) = -\log p(x)$$

- Recall the expectation of an event x.

$$\int xp(x)\,dx \quad \text{or} \quad \sum x_i p(x_i)$$

- The **entropy** (expected information) is

$$-\int p(x)\log p(x)\,dx \quad \text{or} \quad -\sum_i p(x_i)\log p(x_i)$$

Cross-entropy

- Cross-entropy: change $p(x_i)$ or p_i in **entropy** to distribution t_i:

$$L_{\mathrm{CE}} = -\sum_{i=1}^{n} t_i \log(p_i), \quad \text{for n classes,}$$

where t_i is the truth label and p_i is the Softmax probability for the i^{th} class.

- Expected information of p_i with respect to distribution t_i.

- A measure of the **difference between two probability distributions**, typically between a predicted probability distribution and a true probability distribution.

Why is Cross-entropy often used?

- Easy to compute:
  - For binary classification:

$$L = -\left(y \log(p) + (1-y) \log(1-p)\right)$$

  - Gradient:

$$\frac{\partial L}{\partial p} = p - y$$

  - Multi-class
    - p_c is the multiclass probability
    - y_c is one-hot vector

$$\frac{\partial L}{\partial p_c} = p_c - y_c$$

- Useful Gradient information:
  - If predictions are correct → $p_c - 1$, pushing p_c to 1.
  - If predictions are wrong → p_c , pushing p_c to 0.

Important Concepts

- MLP
- Cross-Entropy Loss
- Backpropagation!
- Backpropagation!!
- Backpropagation!!!