CS 696 Intro to Big Data: Tools and Methods
Fall Semester, 2016
Doc 16 Neural Nets
Oct 18, 2016

Neural Networks

All you really need to know for the moment is that the universe is a lot more complicated than you might think, even if you start from a position of thinking it's pretty damn complicated in the first place.

--- Douglas Adams, Hitchhikers Guide to the Universe

Example

Apples	Oranges	Total Cost
2	3	5
9	4	16
4	8	10.5

Find w(a) and w(o)

let w(a) = cost of apple

n(a) = number of apples

w(o) = cost of orange

n(o) = number of oranges

t = transaction fee

Total Cost = w(a)*n(a) + w(o)*n(o) + t

Apples	Oranges	Total Cost	Guess
2	3	5	2.5
9	4	16	6.5
4	8	10.5	6

w(a) - guess 0.5
w(o) - guess 0.5
t - guess 0

Too low

Apples	Oranges	Total Cost	Guess
2	3	5	6
9	4	16	14
4	8	10.5	13

w(a) - guess 1
w(o) - guess 1
t - guess 1

Too high in two cases

Apples	Oranges	Total Cost	Guess
2	3	5	4.5
9	4	16	10.5
4	8	10.5	9.75

w(a) - guess 0.75
w(o) - guess 0.75
t - guess 0.75

Too low

Need

Measure of how far off guess is from data

Systematic way to change weights

Loss Function

Measure of how the data differs from estimate

Linear case

$$\mathfrak{L}(\mathbf{W}, \boldsymbol{b}) = \frac{1}{N}\sum_{i=1}^{N}(\hat{Y}_i - Y_i)^2$$

Y_i = data value

Y_i_hat = computed value

Activation Function

Function that we are trying to fit

In example linear function with two independent variables

f(x1,x2) = a*x1 + b*x2 + c

= w1*x1 + w2*x2 + b

w1, w1 are the weights

b is the bias

Bias

Prejudice in favor of one thing

Positive values being for
Negative values being against

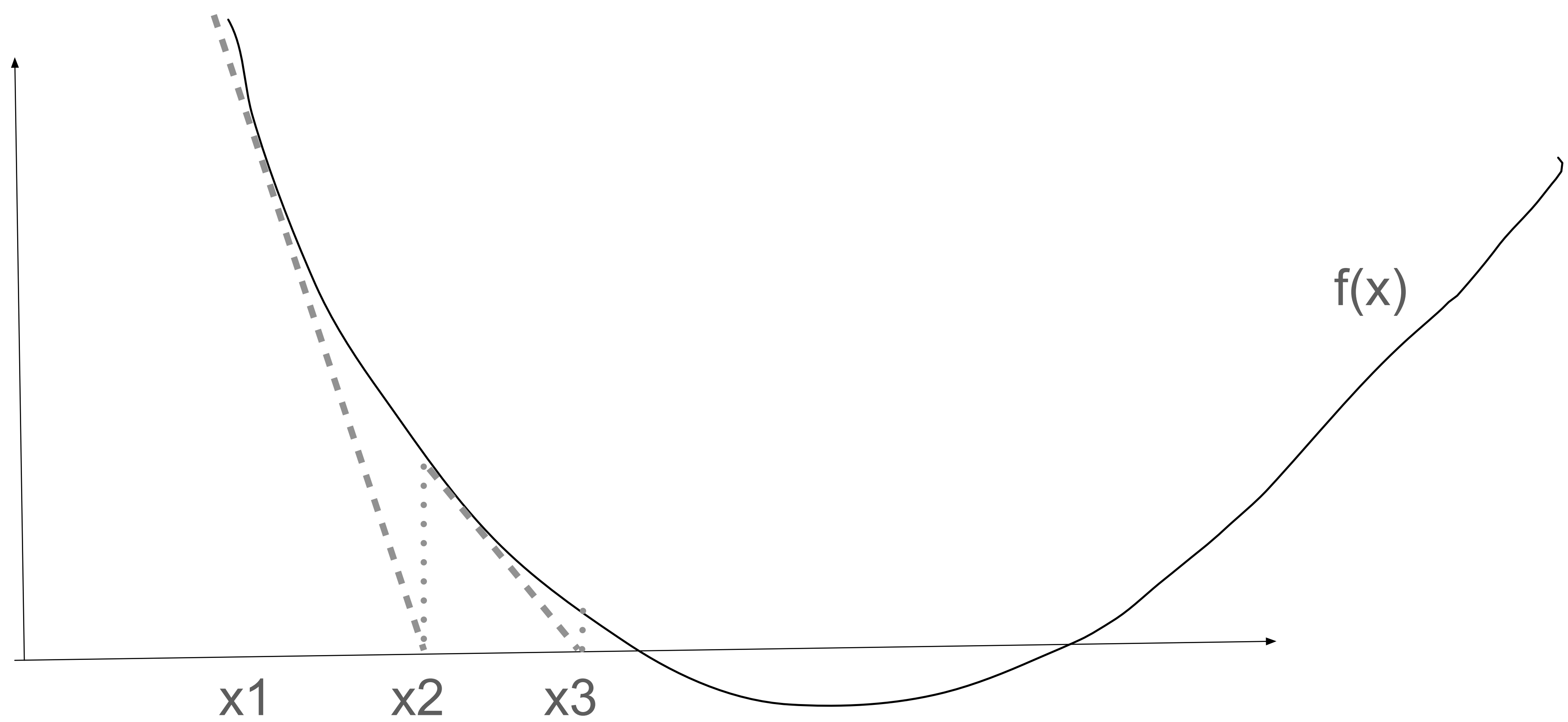
Consider x = 0 neutral input

Then if f is neutral function f(0) == 0

f(x1, x2) = w1*x1 + w2*x2 + b

f(0, 0) = b

So f has a bias



Pick x1

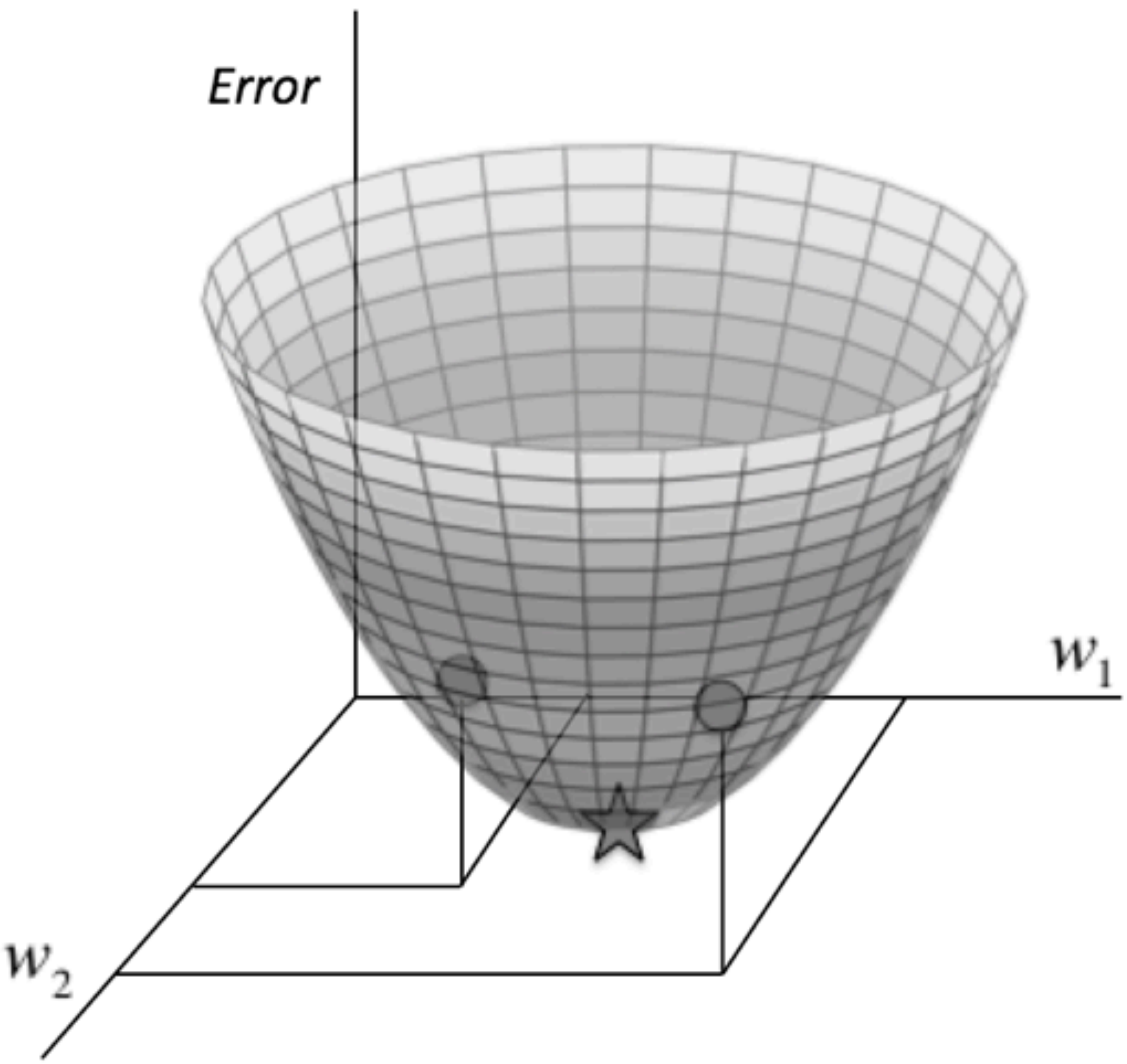
Find the slope at f(x1) ie take derivative

Use slope to estimate where f(x) is zero = x2

Repeat process until f(xn) is really close to 0

Gradient Descent

gradient is the derivative of multi-dimensional function



Fundamentals of Deep Learning, Buduma, O’Reilly Media, Inc., Second Early Release

Systematic way to change weights

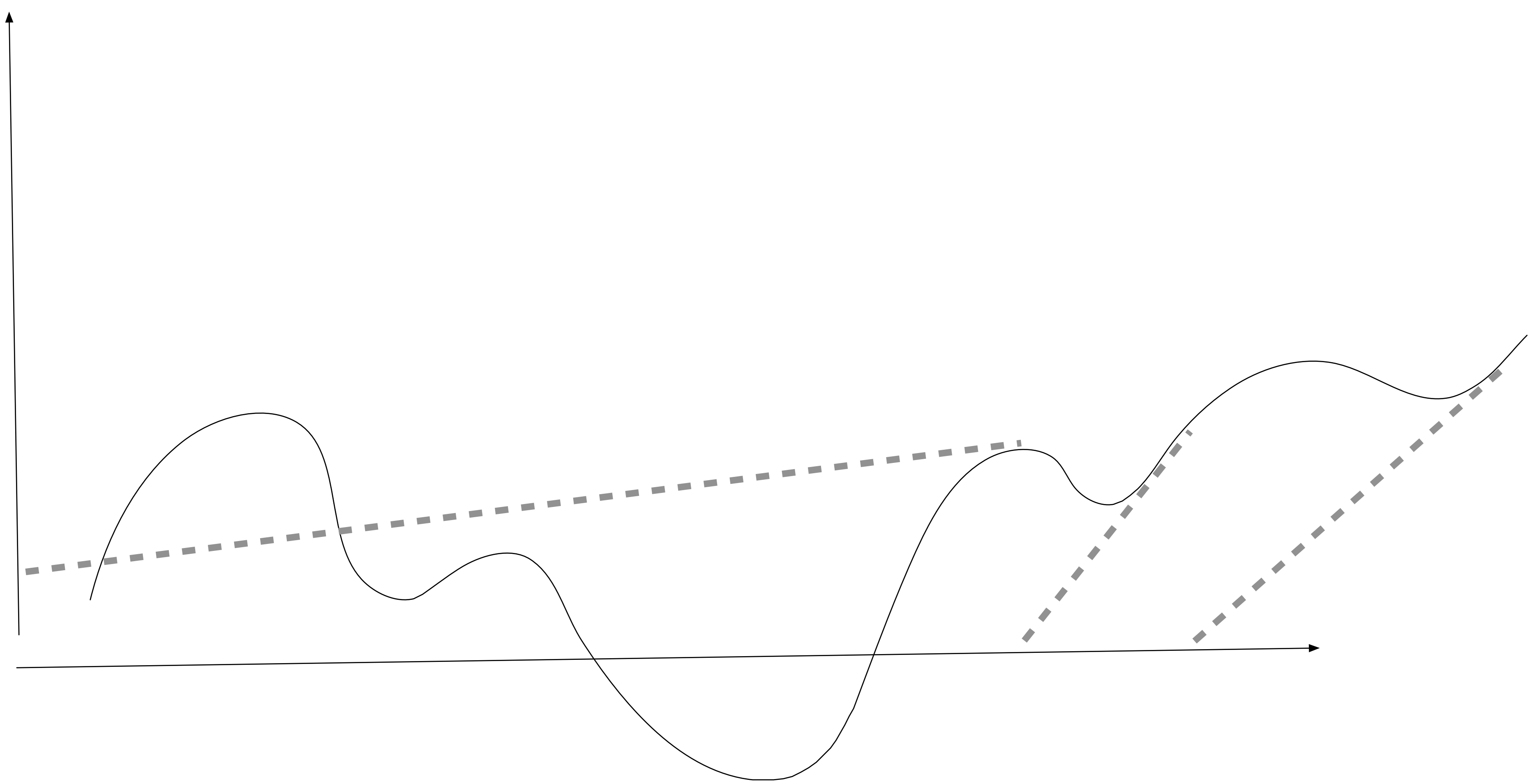
f(x1, x2) = w1*x1 + w2*x2 + b

Take derivative of activation function get gradient

Use the slope in the x1 dimension to adjust w1

Use the slope in the x2 dimension to adjust w2

How far to go?



Learning Rate

To avoid overshooting multiply the gradient by a factor - say 0.1

This is called the learning rate

Take derivative of activation function get gradient

Use the slope in the x1 dimension * learning rate to adjust w1

Use the slope in the x2 dimension * learning rate to adjust w2

Terms

Loss Function

Activation Function

Learning Rate

Weights

Bias

Basic Algorithm

f(x1, x2) = w1*x1 + w2*x2 + b

Select initial values for w1, w2, b

1. Compute loss function on data to find the error

2. Update w1, w2, b

    Take derivative of activation function get gradient

        w1 = w1 + the slope in the x1 dimension * learning rate * Error

        w2 = w2 + the slope in the x2 dimension * learning rate * Error

        b = b + gradient * learning rate * Error

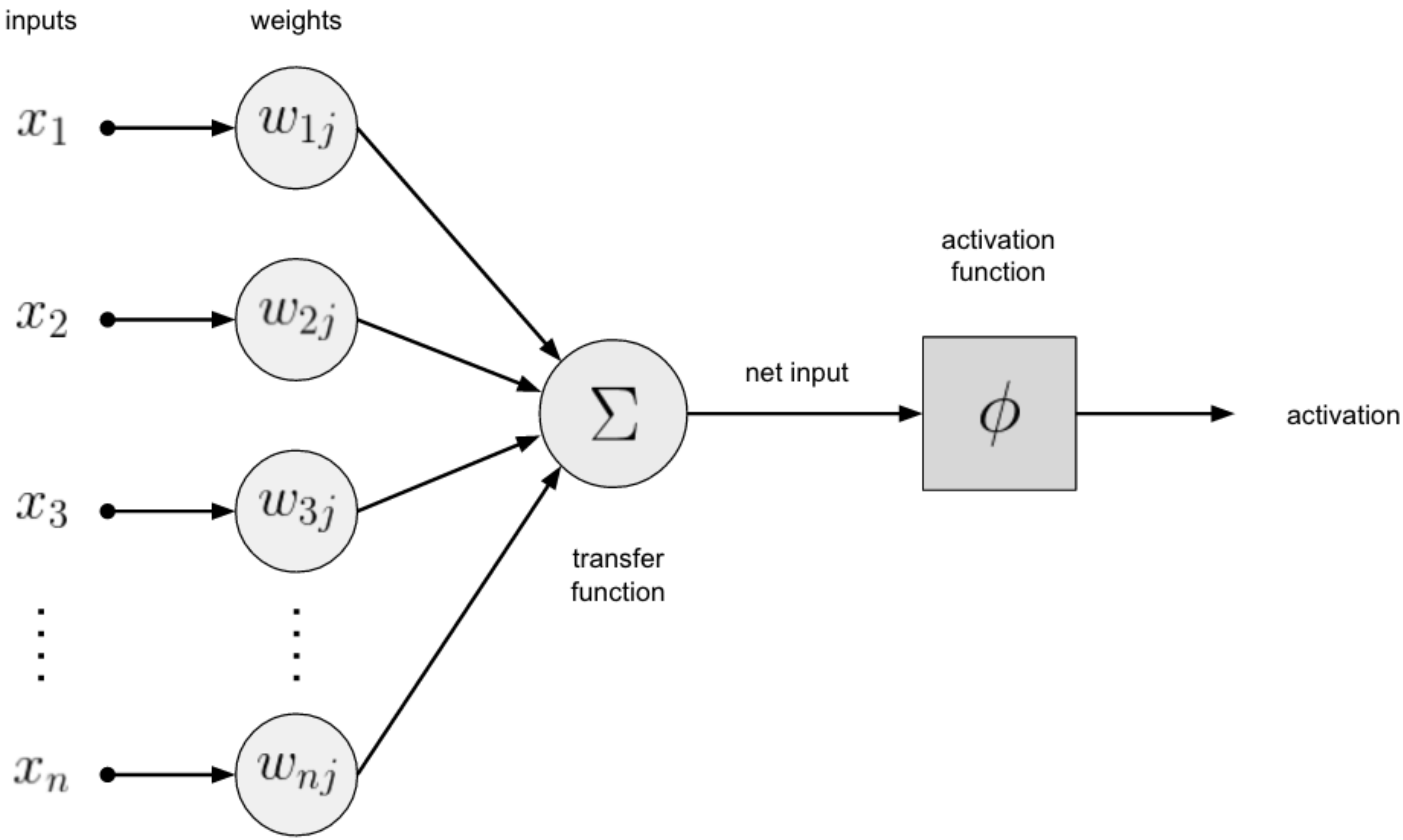
Repeat 1 & 2 until error is acceptable

Learning Rate

If too small then take too long for result to converge

If too large then algorithm will jump arround too much and not converge

Basic Structure of Neuron



Deep Learning, Gibson & Patterson, O'Reilly Media, Inc. , Early Release

Knet.jl

Deep learning framework

Developed at in Koç University in Turkey

Hides some complexity

Can use GPU

Define

    activation (predict) function

    loss function

Then train the data

Linear Knet Example

```
using Knet

activation(w,x) = w[1]*x .+ w[2]

loss(w,x,y) = sumabs2(y - activation(w,x)) / size(y,2)

lossgradient = grad(loss)              # grad computed gradient


function train(w, data; learning_rate=.1)
    for (x,y) in data
        dw = lossgradient(w, x, y)
        for i in 1:length(w)
            w[i] -= learning_rate * dw[i]
        end
    end
end
```

```
x = rand(10)
y = 2 .* x .+ 3          #exact model so we know
x = x'
y = y'
w = [2.5,3.5]

for i in 1:20
  train(w,[(x,y)], learning_rate = 0.1)
  println(loss(w,x,y))
end
```

Loss value

First 0.34

Last 0.001

w:

2.09855

2.94161

Varying Learning Rate

Learning rate 0.01	Loss value	w:
	First 0.43	2.45
	Last 0.26	3.29
Learning rate 0.1	Loss value	w:
	First 0.34	2.10
	Last 0.001	2.94
Learning rate 1.0	Loss value	w:
	First 0.83	53.0
	Last 21299.5	129

Varying Starting Point

Learning rate 0.01 w = [0.0, 0.0]	Loss value	w:
	First 9.1	1.76
	Last 0.005	3.12
Learning rate 0.01 w = [-10.0, -10.0]	Loss value	w:
	First 208	-1.01
	Last 0.76	4.56
Learning rate 0.01 w = [10.0, -10.0]	Loss value	w:
	First 52	10.9
	Last 6.76	-1.81

Neural Networks Parameters

Input weights

Learning Rate

Linear Neurons - Perceptrons

Linear neurons even when combined have limited use

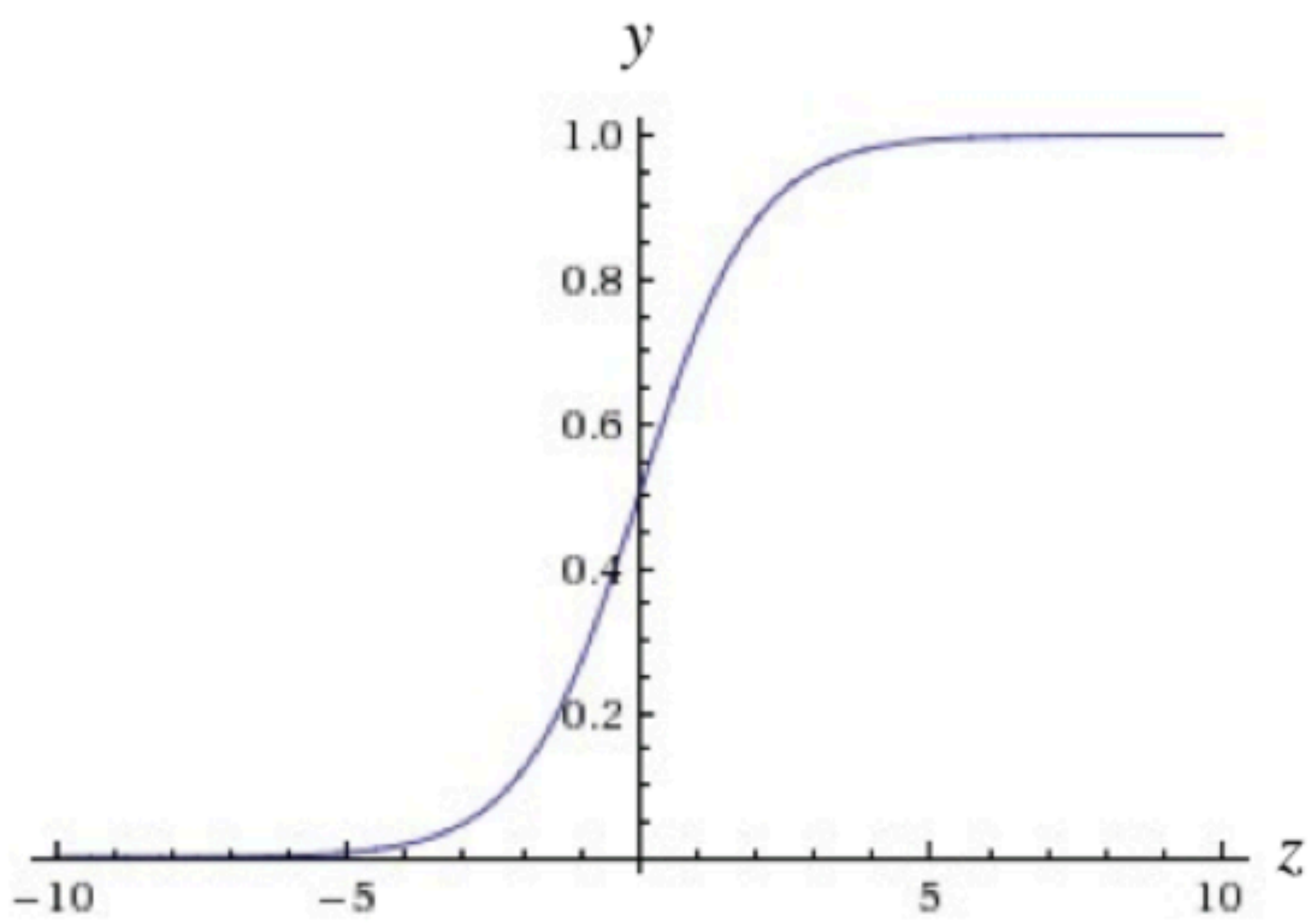
Need more types of neurons

Each type needs gradient function & loss function

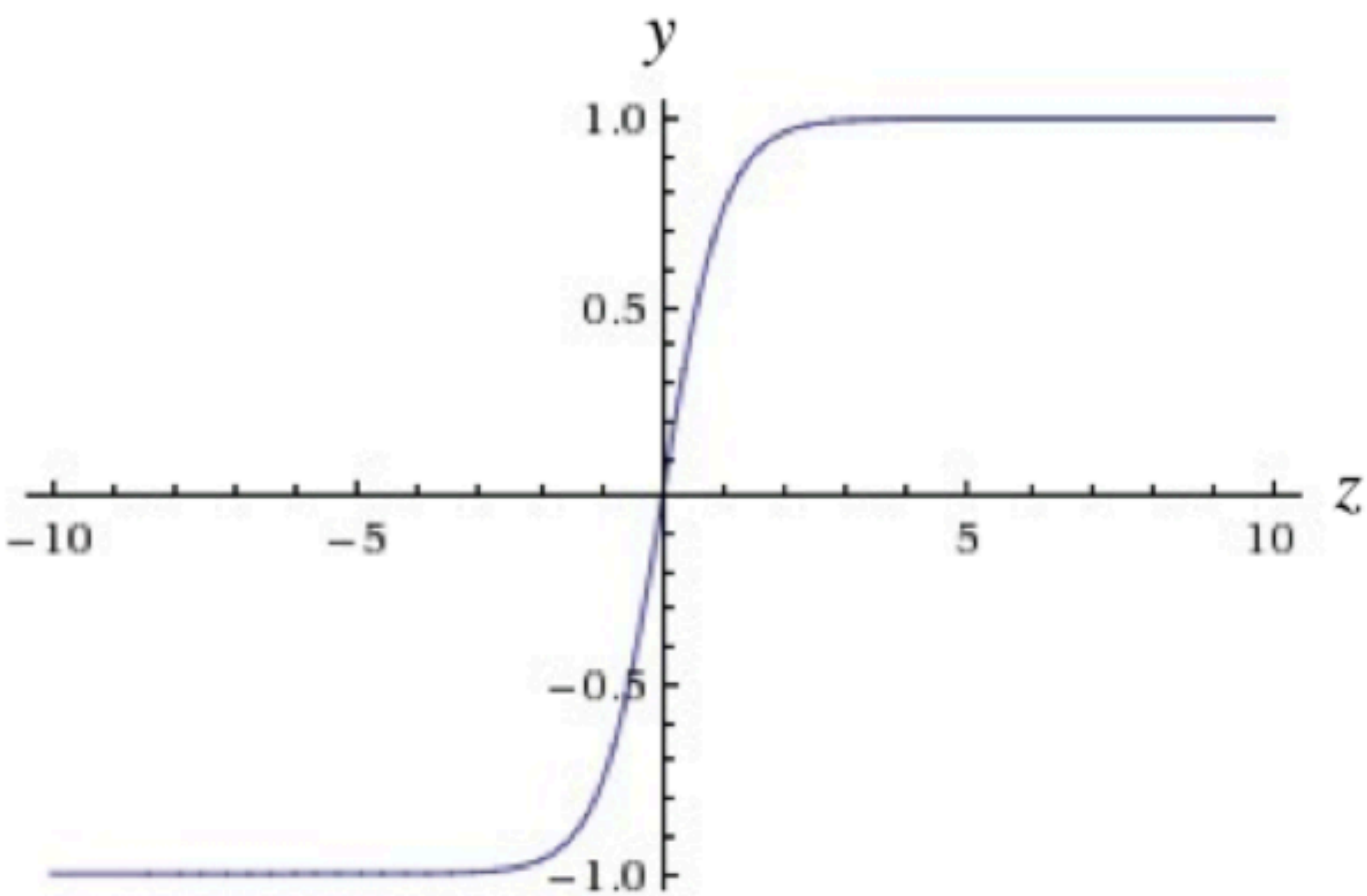
Layers of neurons

Types of Neurons/Activation Functions

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

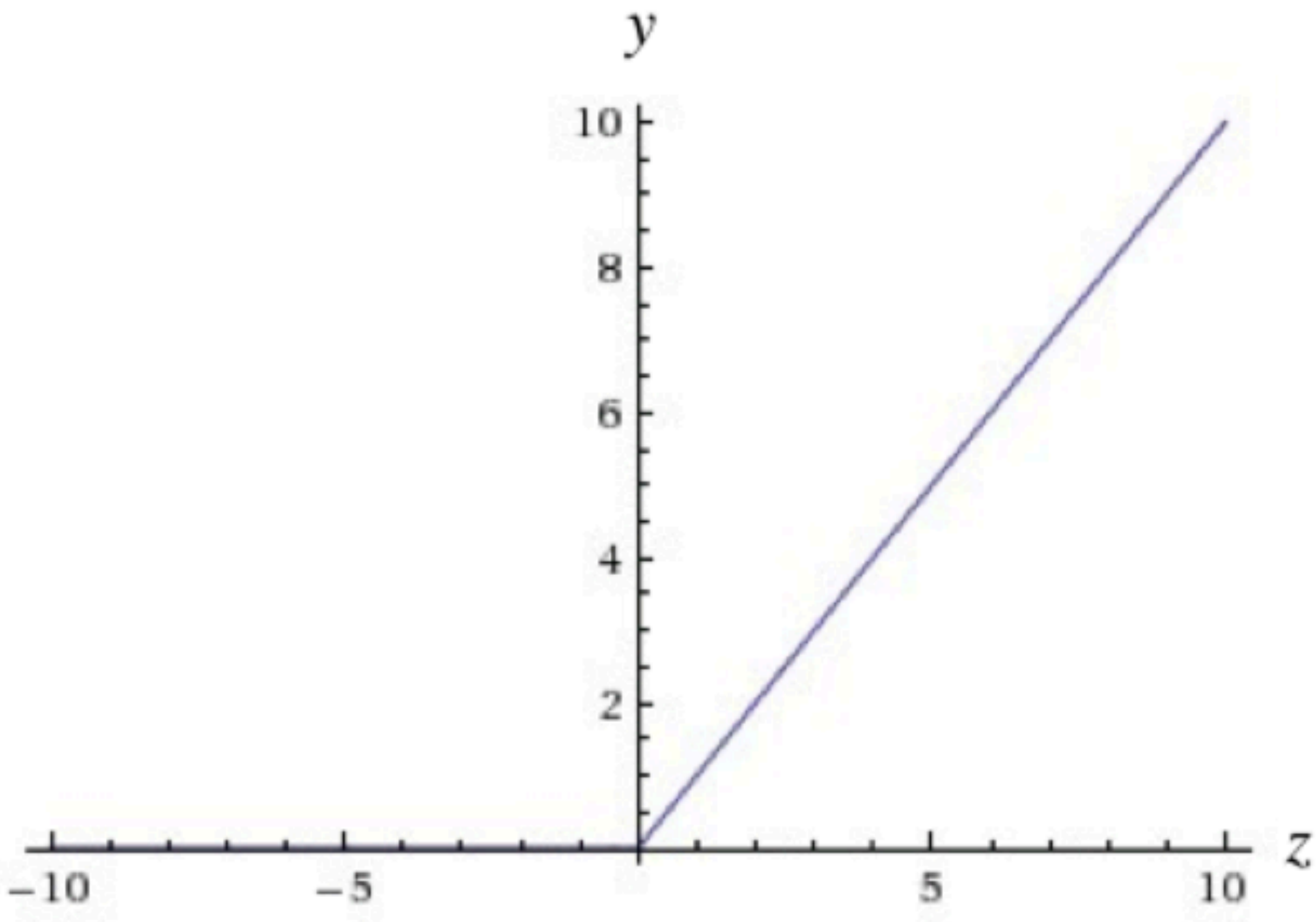


Tanh



Fundamentals of Deep Learning, Buduma, O'Reilly Media, Inc., Second Early Release

Restricted Linear Unit (ReLU)



Softmax

softmax_norm(x) = 1 ./(1 + exp(-(x - mean(x))/std(x)))

Recall from clustering

Often used as output neuron

Loss functions

$$\mathfrak{L}(\mathbf{W}, \boldsymbol{b}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} (\log \hat{y_{ij}} - \log y_{ij})^2$$

mean square log error
MSLE

$$\mathfrak{L}(\mathbf{W}, \boldsymbol{b}) = \frac{1}{N} \sum_{i=1}^{N} max(0, 1 - y_{ij} \times \hat{y_{ij}})$$

Hinge loss
Binary classification

$$\mathfrak{L}(\mathbf{W}, \boldsymbol{b}) = - \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} \times \log \hat{y}_{ij}$$

Logisitic loss

Neural Networks Parameters

Input weights

Learning Rate

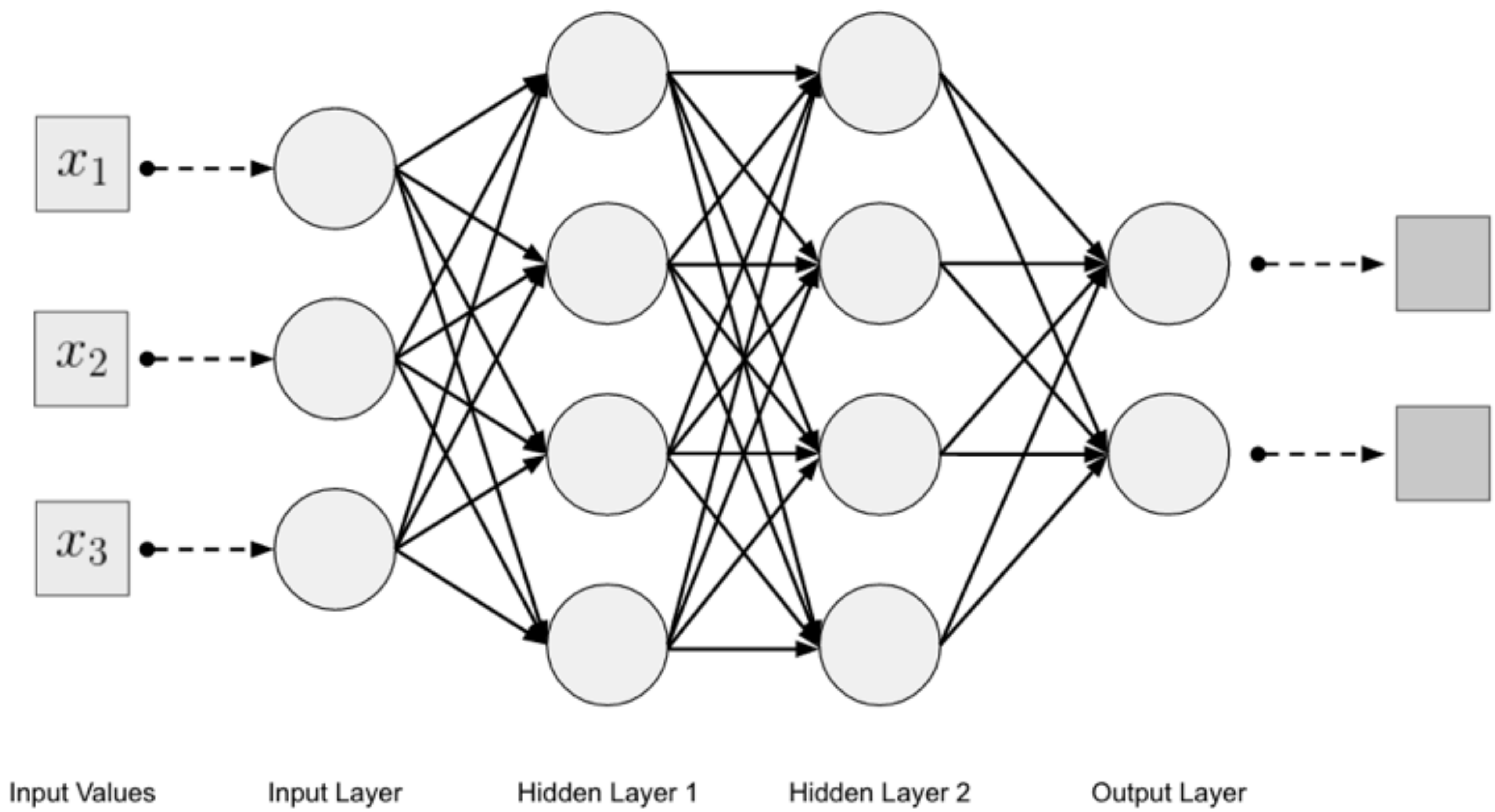
Loss function

Activation function

Layers

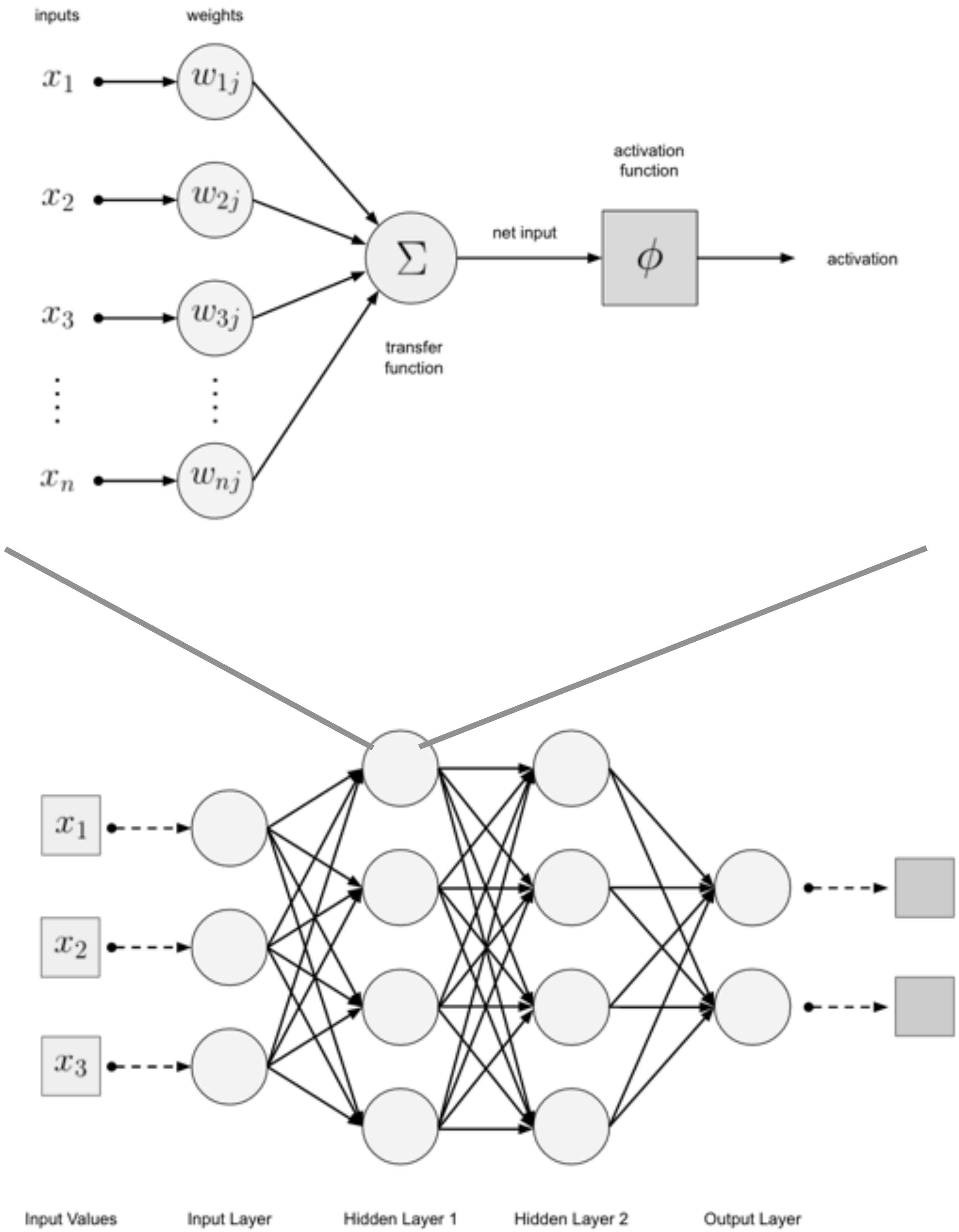
Even with different types of neurons single neurons are not very useful

Create layers of neurons



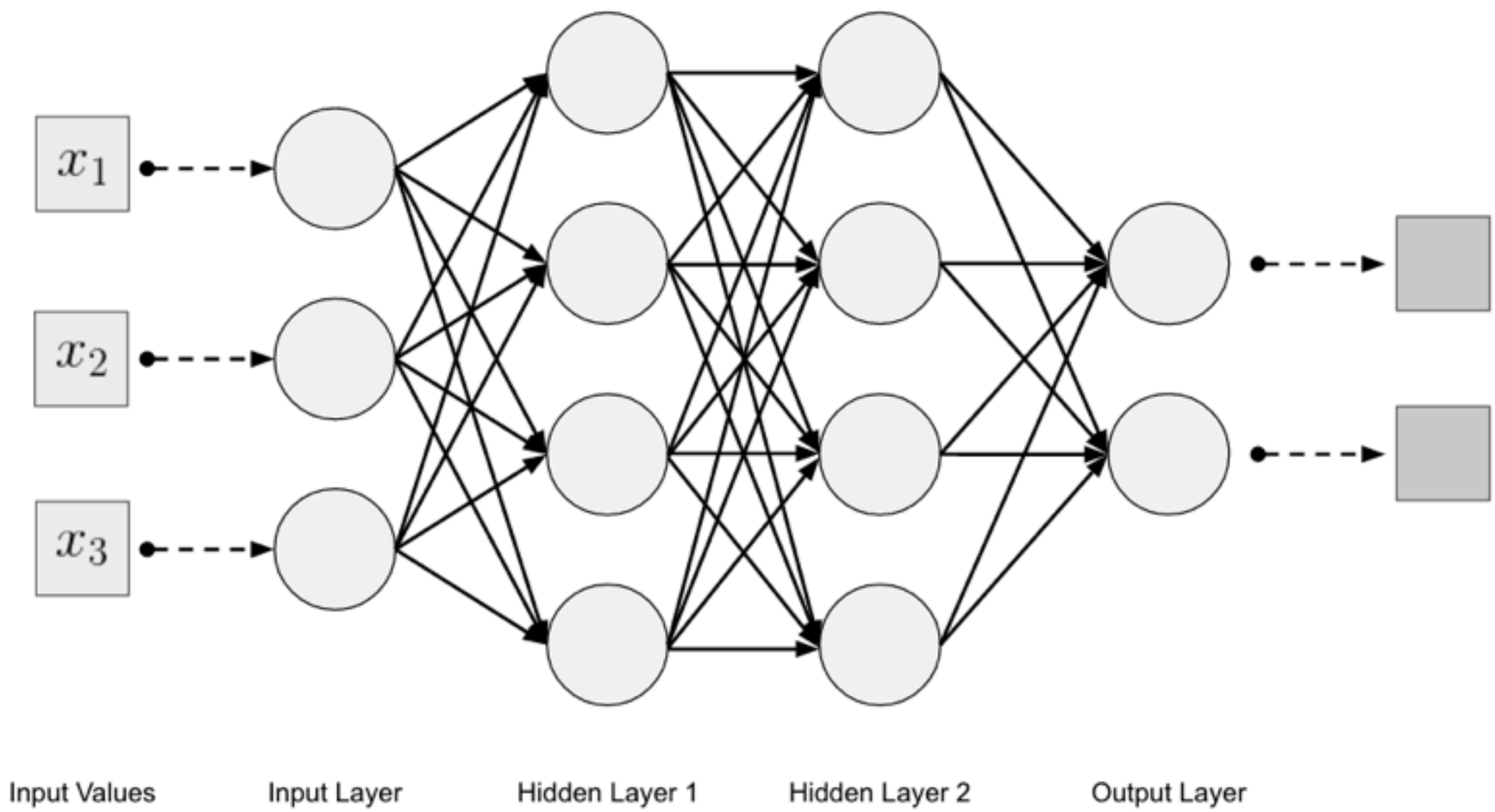
Deep Learning, Gibson & Patterson, O'Reilly Media, Inc. , Early Release

One neuron


inputs
weights
x_1
x_2
x_3
x_n
w_{1j}
w_{2j}
w_{3j}
w_{nj}
Σ
transfer function
net input
activation function
ϕ
activation
x_1
x_2
x_3
Input Values
Input Layer
Hidden Layer 1
Hidden Layer 2
Output Layer

Forward Propagation

Input data goes to input layer
Each neuron passes its ouput to the next layer
Below is a fullly connected neural network



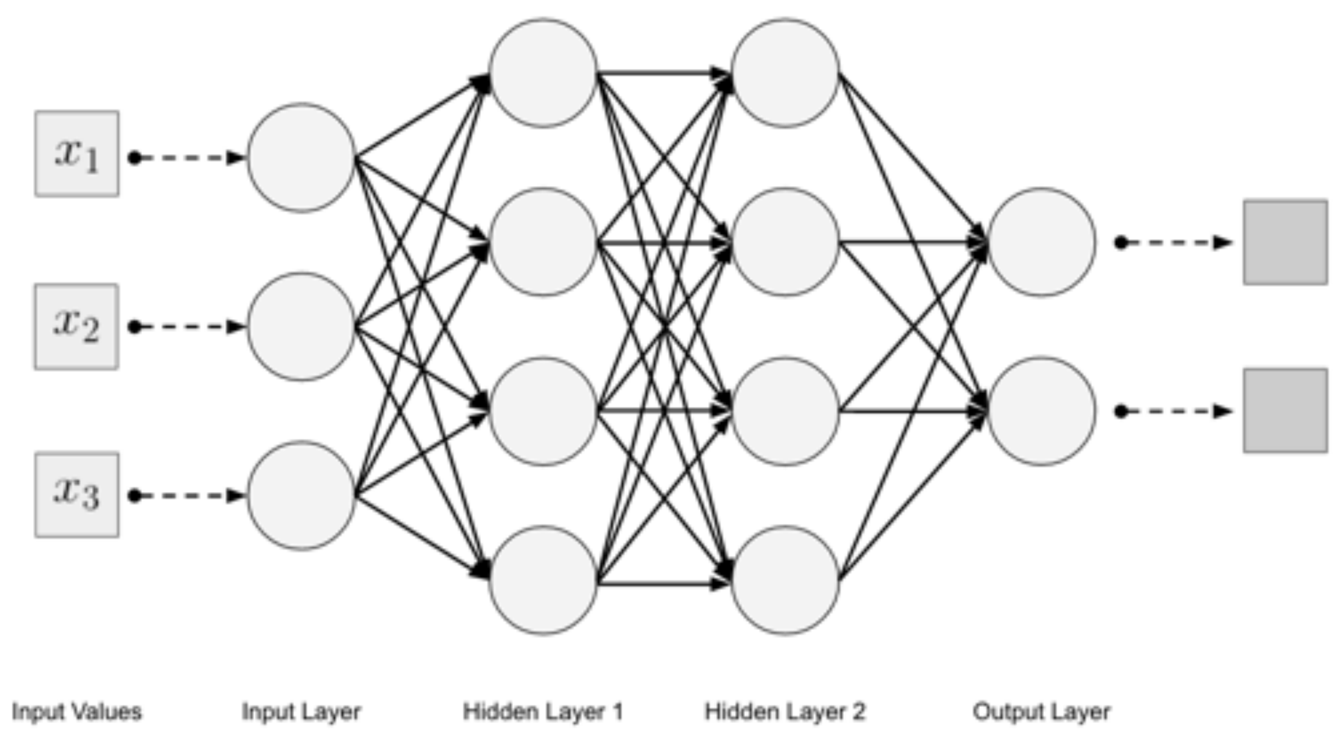
Backpropagation

How to adjust weights for each neuron?

Adjust the weights of the last layer as before

Using these weights we can compute what the inputs to last layer should be

We can now use those estimates to adjust the previous layers weights



Neural Networks Parameters

Input weights per neuron

Learning rate per neuron

Loss function per neuron

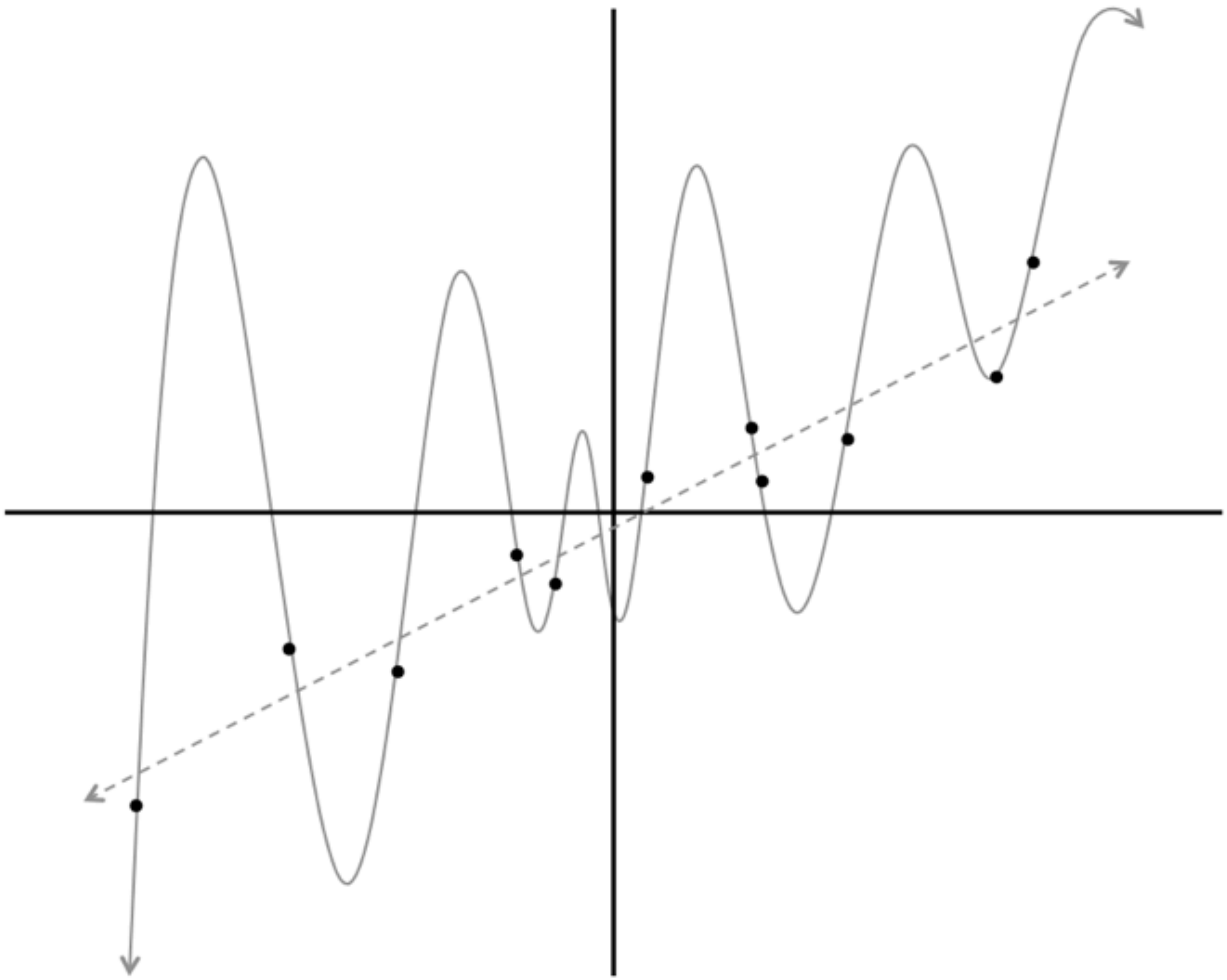
Activation function per neuron

Number of layers

Number of neurons per layer

How neurons are connected

Overfitting



Fundamentals of Deep Learning, Buduma, O'Reilly Media, Inc., Second Early Release

Hyperparameters

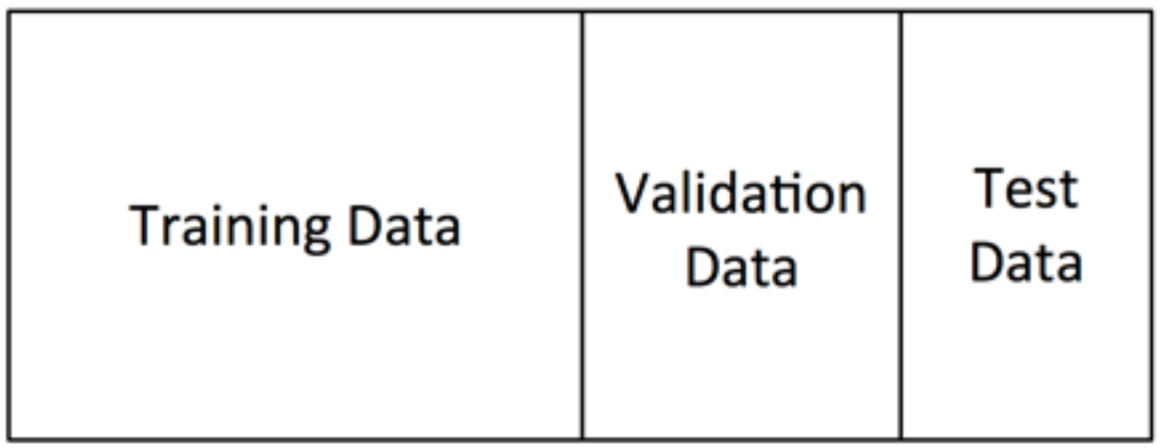
Things we can change to make neural networks train better

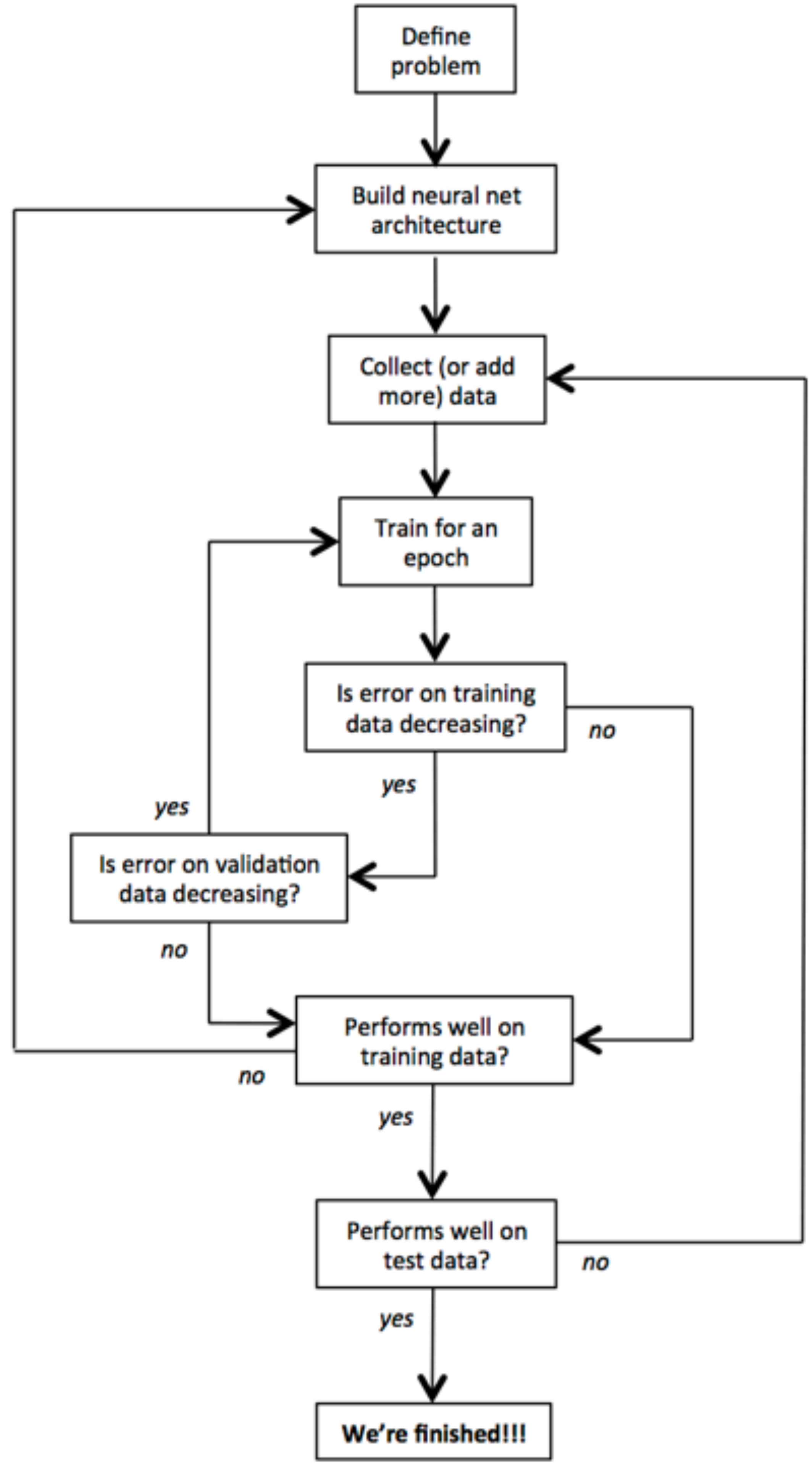
Learning Rate
Activation functions
Weight initalization strategies
Loss functions
Normalization
Layer size & number of layers

mini-batch size
Regularization
Momentum
Sparsity

Work Flow

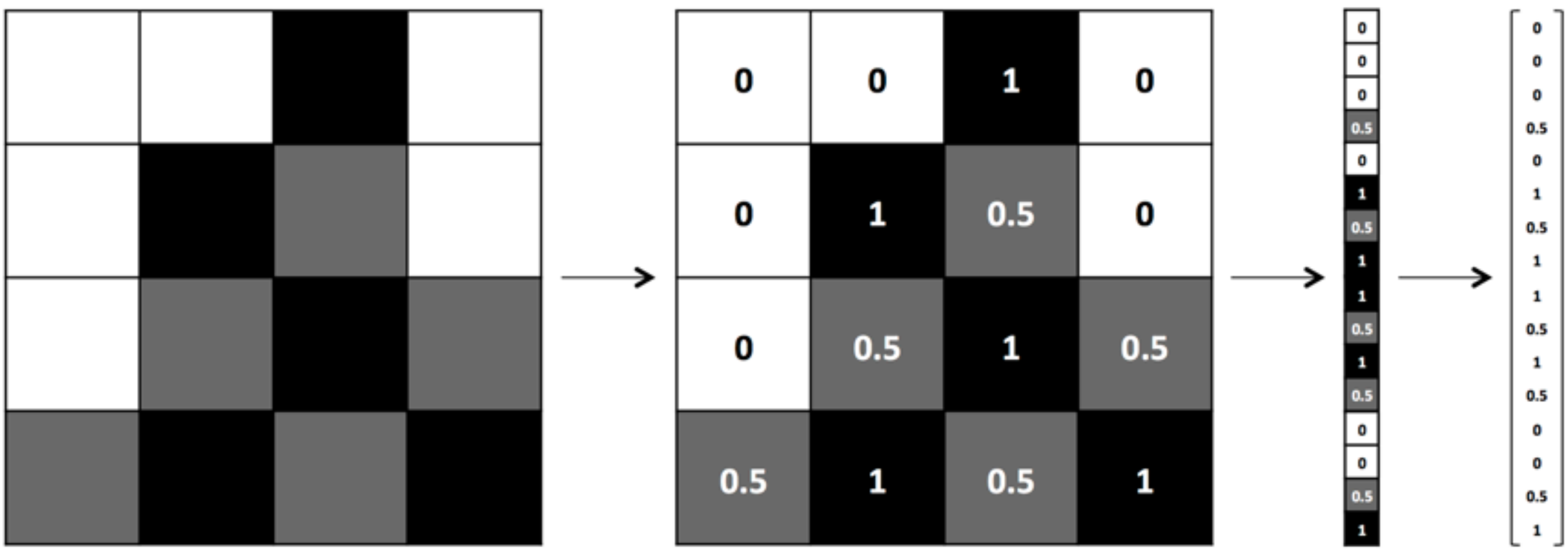
Full Dataset:





Input

Need to map input into vector



Fundamentals of Deep Learning, Buduma, O'Reilly Media, Inc., Second Early Release

Images & Scaling

Image of 32 pixels by 32 pixels with 3 color channels (RGB)

Fully connected neuron needs 32*32*3 = 3,072 weights

Image of 200 pixels by 200 pixels with 3 color channels (RGB)

Fully connected neuron needs 200*200*3 = 120,000 weights

Image researchers use up to 150 layers

Deep Learning

More neurons than previous networks
More complex ways of connecting layers
Explosion of computing power to train
Automatic feature extraction

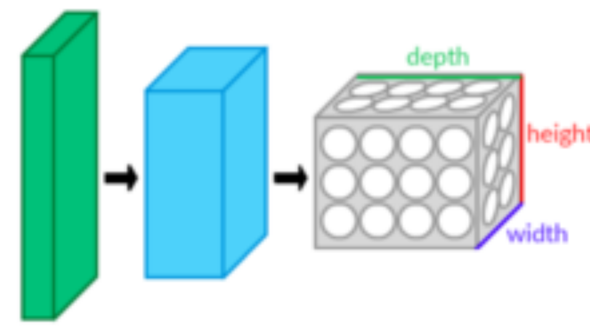
Some Deep Learning Networks

- Unsupervised Pre-Trained Networks
- Convolutional Neural Networks
  - Common for image Analysis
- Recurrent Neural Networks
  - Time series analysis
- Recursive Neural Networks

Convolutional Neural Network

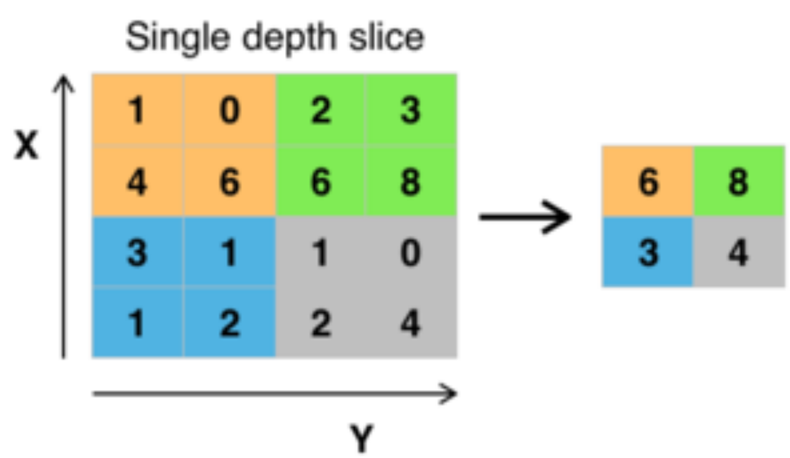
Convolutional Layer

- 3-D network of neurons
- Only locally connected
- Each 2-D slice in depth share same weight

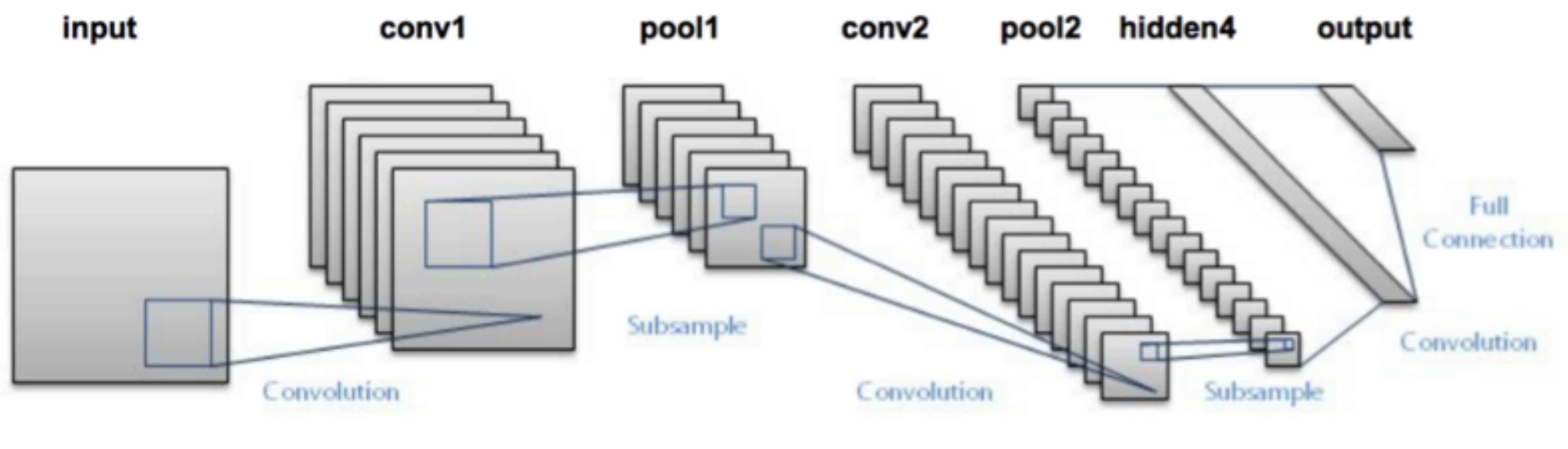


Pooling Layer

- Down-sampling layer

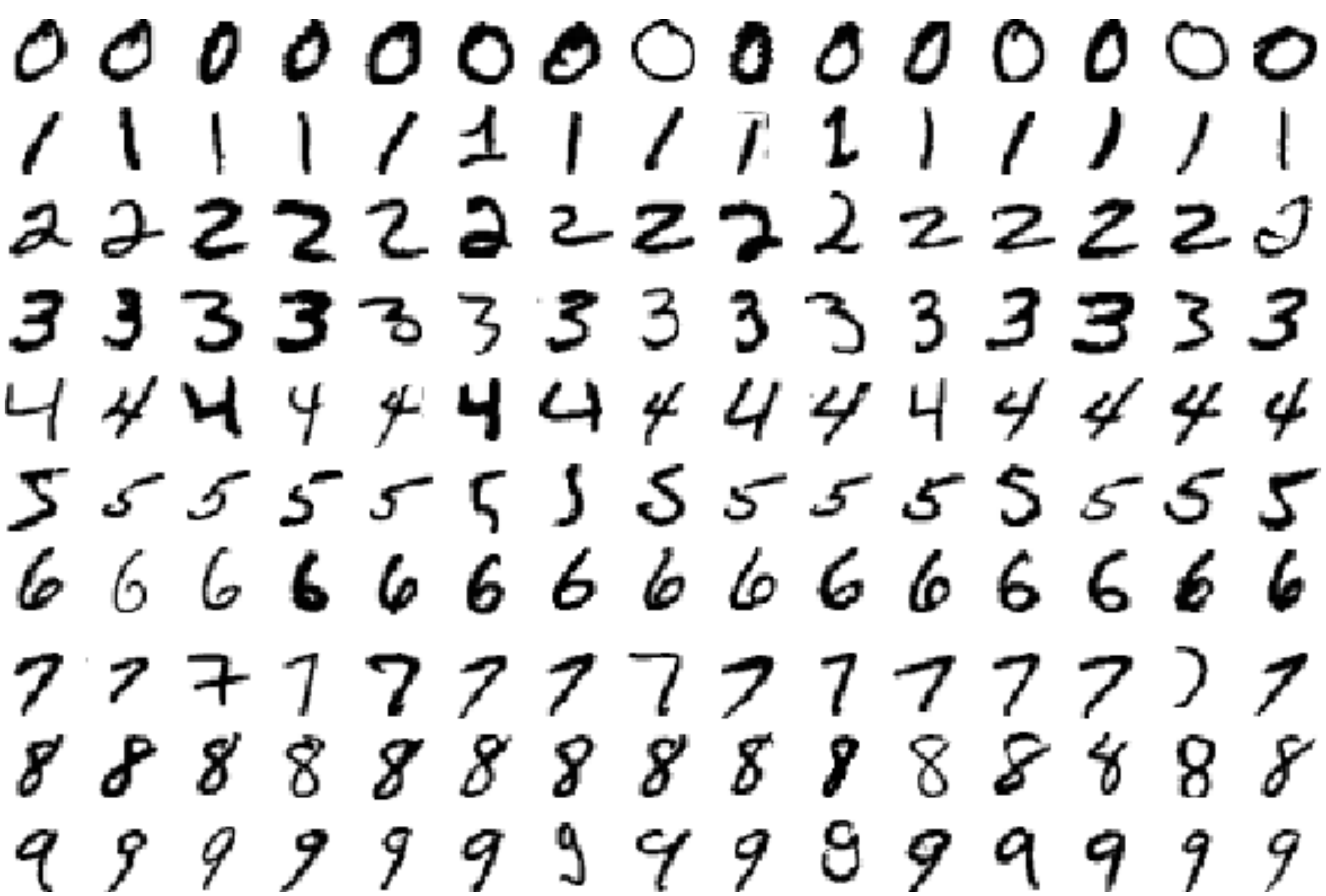


https://en.wikipedia.org/wiki/Convolutional_neural_network


input
conv1
pool1
conv2
pool2
hidden4
output
Convolution
Subsample
Convolution
Subsample
Convolution
Full
Connection

Hello World of Deep Learning

Mixed National Institute of Standards & Technology database of handwritten digits
60,000 training images
Normallized to 20x20 pixels with grayscale



Different Methods with Error Rate

https://en.wikipedia.org/wiki/MNIST_database