

CS 649 Big Data: Tools and Methods
Spring Semester, 2021
Doc 21 Clustering & Deep Learning
Mar 13, 2021

Copyright ©, All rights reserved. 2021 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (<http://www.opencontent.org/openpub/>) license defines the copyright on this document.

Spark ML

Classification

- Logistic Regression
- Decision tree classifier
- Random forest classifier
- Gradient-boosted tree classifier
- Multilayer perceptron classifier**
- Linear Support Vector Machine
- One-vs-Rest classifier
- Naive Bayes

Clustering

- K-means
- Latent Dirichlet allocation
- Gaussian Mixture Model

How to Export Spark Models

Spark.mllib supports PMML for some models

`spark.mllib` model	PMML model
KMeansModel	ClusteringModel
LinearRegressionModel	RegressionModel
RidgeRegressionModel	RegressionModel
LassoModel	RegressionModel
SVMModel	RegressionModel
Binary LogisticRegressionModel	RegressionModel

Predictive Model Markup Language

XML-based predictive model interchange format

ML libraries and Model interoperability

<https://www.andrey-melentyev.com/model-interoperability.html>

Spark k-Means

pyspark.ml.clustering.KMeans

Need to set number of clusters

Can set a seed

iris.txt

```
sepal_length,sepal_width,petal_length,petal_width,species
5.1,3.5,1.4,0.2,setosa
4.9,3.0,1.4,0.2,setosa
4.7,3.2,1.3,0.2,setosa
4.6,3.1,1.5,0.2,setosa
5.0,3.6,1.4,0.2,setosa
5.4,3.9,1.7,0.4,setosa
4.6,3.4,1.4,0.3,setosa
5.0,3.4,1.5,0.2,setosa
```

```
import pyspark.sql as sql
spark = sql.SparkSession.builder \
    .master("local[4]") \
    .appName("Sample") \
    .getOrCreate()
```

```
iris = spark.read.format("csv"). \
    option("header", True). \
    option("inferSchema", True). \
    load("iris.txt")
```

```
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import VectorAssembler
from pyspark.ml import Pipeline
```

```
iris_indexer = StringIndexer(inputCol="species", outputCol="label").fit(iris)
iris_assembler = VectorAssembler(inputCols=["sepal_length", "sepal_width", "petal_length",
"petal_width"], outputCol="features")
```

```
pipeline = Pipeline(stages=[iris_indexer, iris_assembler])
iris_formated = pipeline.fit(iris).transform(iris)
```

iris_formated.show()

sepal_length	sepal_width	petal_length	petal_width	species	label	features
5.1	3.5	1.4	0.2	setosa	2.0	[5.1,3.5,1.4,0.2]
4.9	3.0	1.4	0.2	setosa	2.0	[4.9,3.0,1.4,0.2]
4.7	3.2	1.3	0.2	setosa	2.0	[4.7,3.2,1.3,0.2]
4.6	3.1	1.5	0.2	setosa	2.0	[4.6,3.1,1.5,0.2]
5.0	3.6	1.4	0.2	setosa	2.0	[5.0,3.6,1.4,0.2]
5.4	3.9	1.7	0.4	setosa	2.0	[5.4,3.9,1.7,0.4]
4.6	3.4	1.4	0.3	setosa	2.0	[4.6,3.4,1.4,0.3]
5.0	3.4	1.5	0.2	setosa	2.0	[5.0,3.4,1.5,0.2]
4.4	2.9	1.4	0.2	setosa	2.0	[4.4,2.9,1.4,0.2]
4.9	3.1	1.5	0.1	setosa	2.0	[4.9,3.1,1.5,0.1]
5.4	3.7	1.5	0.2	setosa	2.0	[5.4,3.7,1.5,0.2]
4.8	3.4	1.6	0.2	setosa	2.0	[4.8,3.4,1.6,0.2]
4.8	3.0	1.4	0.1	setosa	2.0	[4.8,3.0,1.4,0.1]


```
from pyspark.ml.clustering import KMeans, KMeansModel
clusters = KMeans(k = 3)
```

```
iris_model = clusters.fit(iris_formated)
```

```
centers = iris_model.clusterCenters()
```

```
print("Cluster Centers: ")
```

```
for center in centers:
```

```
    print(center)
```

```
Cluster Centers:
```

```
[5.88360656 2.74098361 4.38852459 1.43442623]
```

```
[6.85384615 3.07692308 5.71538462 2.05384615]
```

```
[5.006 3.418 1.464 0.244]
```

```
predictions = iris_model.transform(iris_formated)
```

```
predictions.filter(predictions.label != predictions.prediction).show()
```

sepal_length	sepal_width	petal_length	petal_width	species	label	features	prediction
7.0	3.2	4.7	1.4	versicolor	0.0	[7.0,3.2,4.7,1.4]	1
6.9	3.1	4.9	1.5	versicolor	0.0	[6.9,3.1,4.9,1.5]	1
6.7	3.0	5.0	1.7	versicolor	0.0	[6.7,3.0,5.0,1.7]	1
5.8	2.7	5.1	1.9	virginica	1.0	[5.8,2.7,5.1,1.9]	0
4.9	2.5	4.5	1.7	virginica	1.0	[4.9,2.5,4.5,1.7]	0
5.7	2.5	5.0	2.0	virginica	1.0	[5.7,2.5,5.0,2.0]	0
5.8	2.8	5.1	2.4	virginica	1.0	[5.8,2.8,5.1,2.4]	0
6.0	2.2	5.0	1.5	virginica	1.0	[6.0,2.2,5.0,1.5]	0
5.6	2.8	4.9	2.0	virginica	1.0	[5.6,2.8,4.9,2.0]	0
6.3	2.7	4.9	1.8	virginica	1.0	[6.3,2.7,4.9,1.8]	0
6.2	2.8	4.8	1.8	virginica	1.0	[6.2,2.8,4.8,1.8]	0
6.1	3.0	4.9	1.8	virginica	1.0	[6.1,3.0,4.9,1.8]	0
6.3	2.8	5.1	1.5	virginica	1.0	[6.3,2.8,5.1,1.5]	0
6.0	3.0	4.8	1.8	virginica	1.0	[6.0,3.0,4.8,1.8]	0
5.8	2.7	5.1	1.9	virginica	1.0	[5.8,2.7,5.1,1.9]	0
6.3	2.5	5.0	1.9	virginica	1.0	[6.3,2.5,5.0,1.9]	0
5.9	3.0	5.1	1.8	virginica	1.0	[5.9,3.0,5.1,1.8]	0

17/151 incorrect

```
from pyspark.ml.evaluation import ClusteringEvaluator
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(predictions)
print("Silhouette with squared euclidean distance = " + str(silhouette))
```

Silhouette with squared euclidean distance = 0.7342113066202725

Silhouettes

Method for validating clusters of data

$a(i)$ = average distance from i -th point to other points within the same cluster

if i is in wrong cluster $a(i)$ will be high

$b(i, k)$ = average distance from the i -th point to the points in the k -th cluster

$b(i)$ = $\min b(i, k)$ over all k except for $k = i$

if i -th point is in wrong cluster $b(i)$ will be low

$s(i) = (b(i) - a(i)) / \max(a(i), b(i))$

$-1 \leq s(i) \leq 1$

Silhouettes

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

$$-1 \leq s(i) \leq 1$$

$s(i)$ close to 1 indicates i -th point well within cluster

PCA - Principle Component Analysis

Used to reduce the dimensionality of data

Changes the dimension of the data so

First dimension has the greatest variance

Second dimension has second greatest variance

...

Can then select first K dimensions to work with

Data is transformed into different coordinate system

```
from pyspark.ml.feature import PCA
pca = PCA(k=3, inputCol="features", outputCol="pca_features")
model = pca.fit(iris_formated)
iris_transformed = model.transform(iris_formated)
```

```
model.explainedVariance
```

```
DenseVector([0.9246, 0.053, 0.0172])
```

```
Sum      0.9948
```

```
pca = PCA(k=2, inputCol="features", outputCol="pca_features")
model.explainedVariance
```

```
DenseVector([0.9246, 0.053])
```

```
Sum      0.9776
```

```
iris_transformed.select("features", "pca_features").show(n=20, truncate=60)
```

```
+-----+-----+
|          features          |          pca_features          |
+-----+-----+
|[5.1, 3.5, 1.4, 0.2] | [-2.827135972679027, -5.641331045573321, 0.6642769315107171] |
|[4.9, 3.0, 1.4, 0.2] | [-2.7959524821488437, -5.145166883252896, 0.8462865195142029] |
|[4.7, 3.2, 1.3, 0.2] | [-2.6215235581650584, -5.177378121203909, 0.6180558535097703] |
|[4.6, 3.1, 1.5, 0.2] | [-2.7649059004742402, -5.003599415056946, 0.605093119223434] |
|[5.0, 3.6, 1.4, 0.2] | [-2.7827501159516603, -5.648648294377395, 0.5465353947341569] |
|[5.4, 3.9, 1.7, 0.4] | [-3.231445736773378, -6.062506444034077, 0.46843947549237885] |
|[4.6, 3.4, 1.4, 0.3] | [-2.690452415602345, -5.232619219784267, 0.37851400931804624] |
|[5.0, 3.4, 1.5, 0.2] | [-2.8848611044591563, -5.485129079769225, 0.6585666047730699] |
|[4.4, 2.9, 1.4, 0.2] | [-2.6233845324473406, -4.743925704477345, 0.6154296883942059] |
|[4.9, 3.1, 1.5, 0.1] | [-2.8374984110638537, -5.208032027056187, 0.8342983942443872] |
```


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

$$\text{Total Cost} = w(a) \cdot n(a) + w(o) \cdot n(o) + t$$

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

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

Y_i = data value

\hat{Y}_i = computed value

Activation Function

Function that we are trying to fit

In example linear function with two independent variables

$$f(x_1, x_2) = a \cdot x_1 + b \cdot x_2 + c$$

$$= w_1 \cdot x_1 + w_2 \cdot x_2 + b$$

w_1, w_2 are the weights

b is the bias

Bias

Prejudice in favor of one thing

$$f(x_1, x_2) = w_1 * x_1 + w_2 * x_2 + b$$

Positive values being for

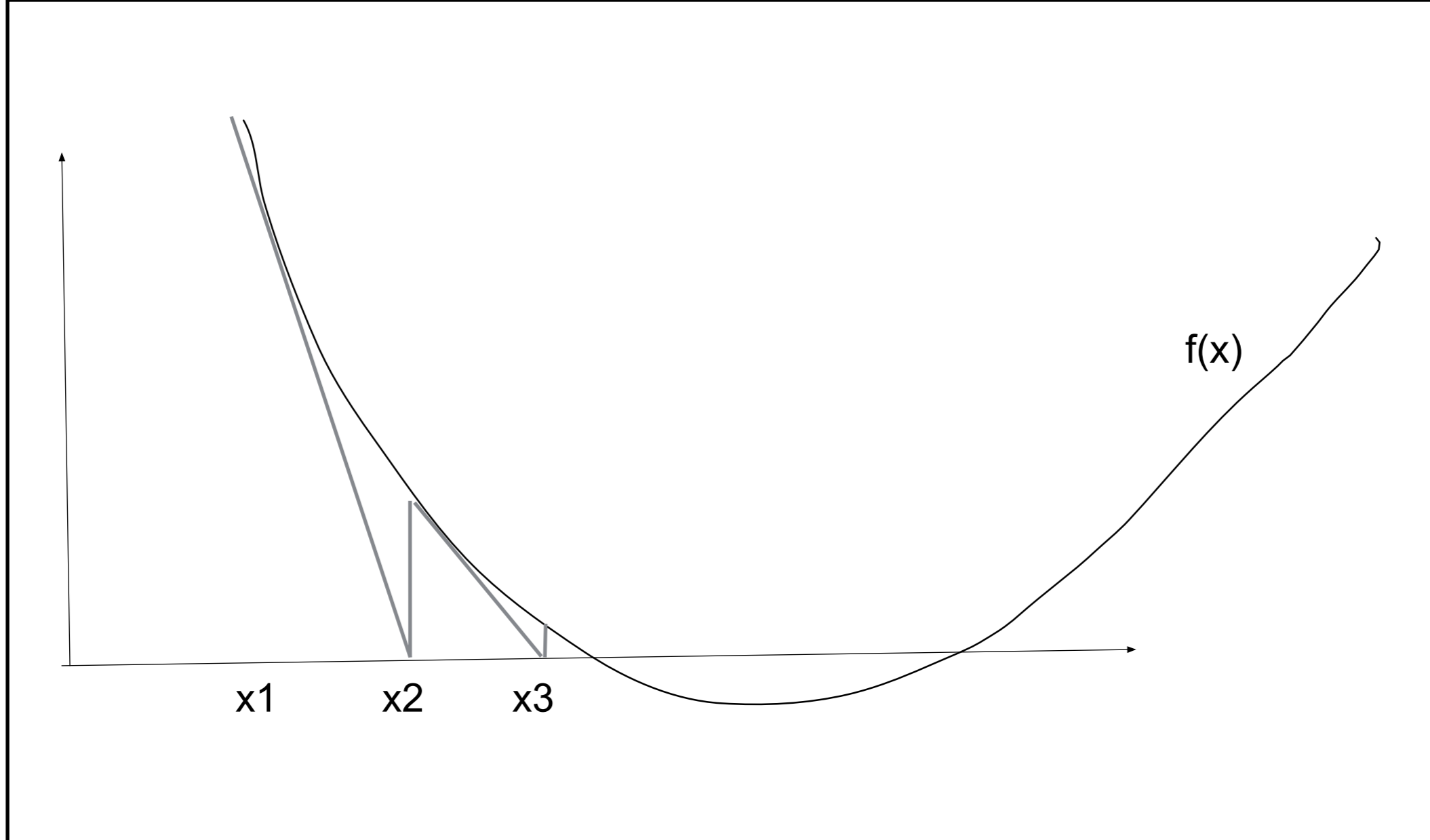
$$f(0, 0) = b$$

Negative values being against

So f has a bias

Consider $x = 0$ neutral input

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



Pick x_1

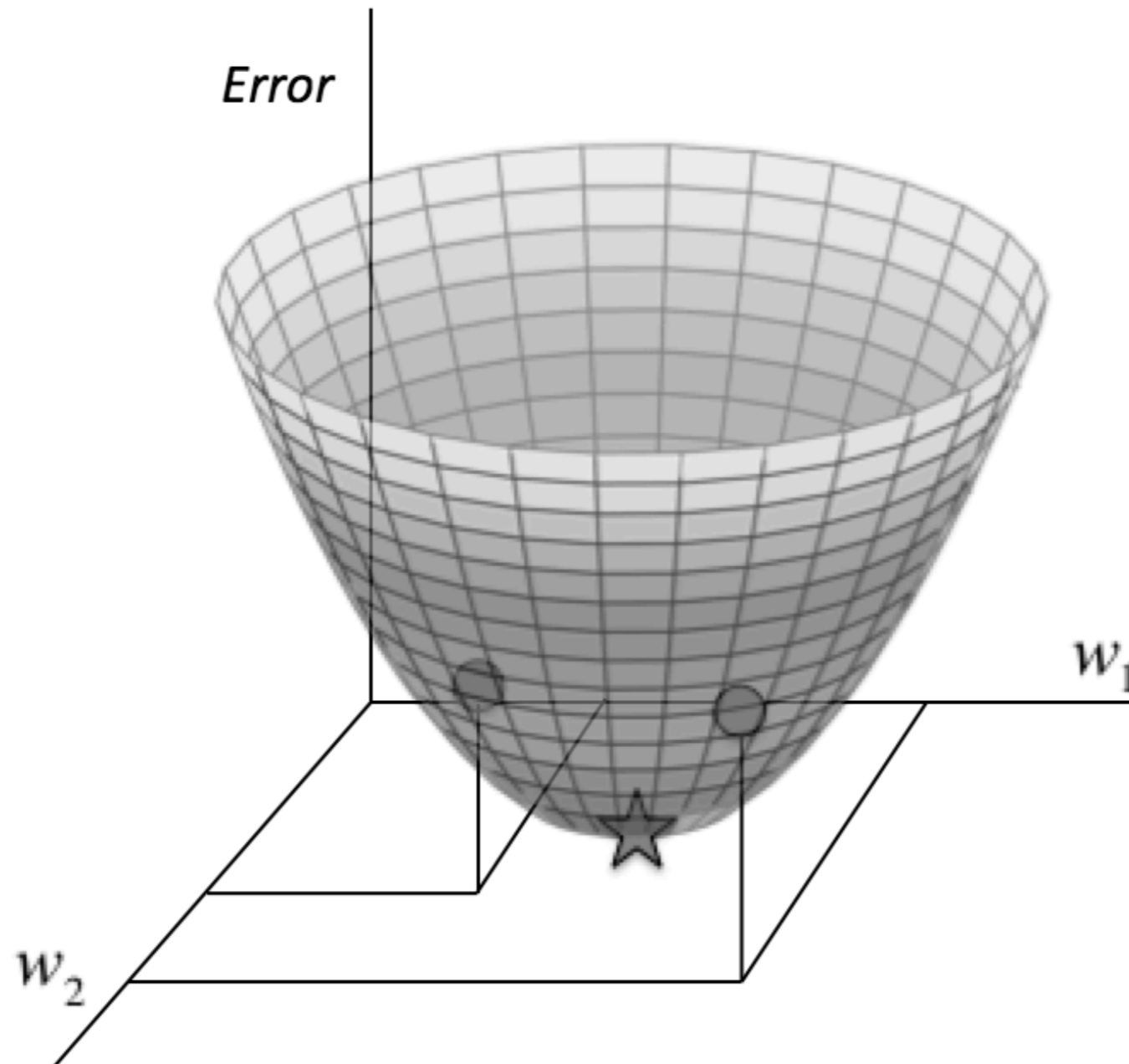
Find the slope at $f(x_1)$ ie take derivative

Use slope to estimate where $f(x)$ is zero = x_2

Repeat process until $f(x_n)$ is really close to 0

Gradient Descent

gradient is the derivative of multi-dimensional function



Differentiable programming

Programming paradigm where

- Program differentiates a function automatically

- Allows for gradient based optimization

- Neural nets are subset

Static compile graph based

- TensorFlow, Theano, MXNet

- Scales well but limited interactivity & program structure (loops, recursion)

Operator overloading, dynamic graph

- PyTorch, AutoGrad

- Not as scalable

Julia's Zygote package

- Overcomes limitations of other systems

- Differentiates code at compile time

$$f(x) = 2x^2 + 3x - 2$$

$$f'(2)$$

$$df(x) = \text{gradient}(f, x)[1]$$

Systematic way to change weights

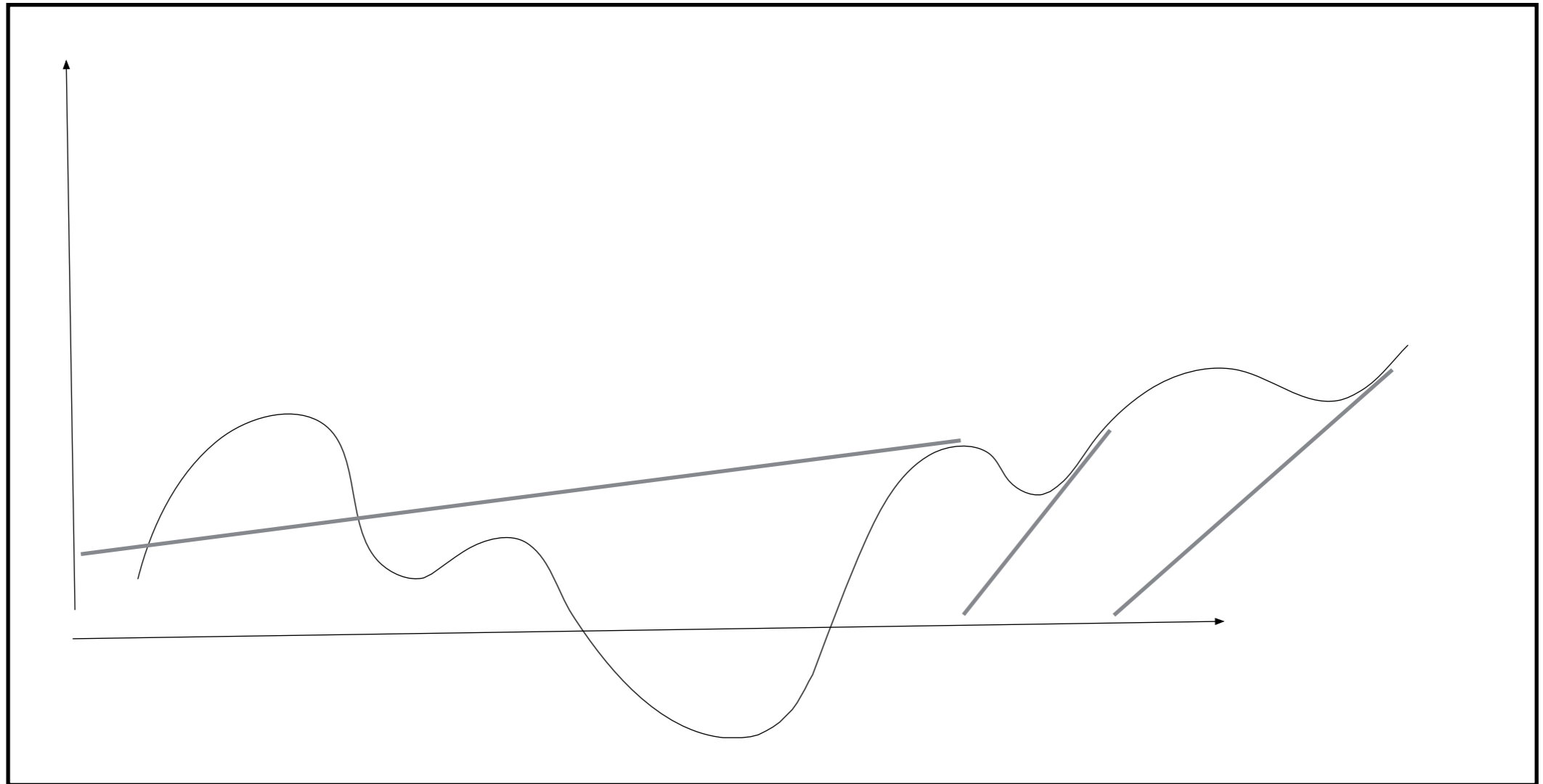
$$f(x_1, x_2) = w_1 * x_1 + w_2 * x_2 + b$$

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?



28

D21

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 x_1 dimension * learning rate to adjust w_1

Use the slope in the x_2 dimension * learning rate to adjust w_2

Terms

Loss Function

Activation Function

Learning Rate

Weights

Bias

Basic Algorithm

$$f(x_1, x_2) = w_1 * x_1 + w_2 * x_2 + b$$

Select initial values for w_1 , w_2 , b

1. Compute loss function on data to find the error
2. Update w_1 , w_2 , b

Take derivative of activation function get gradient

$$w_1 = w_1 + \text{the slope in the } x_1 \text{ dimension} * \text{learning rate} * \text{Error}$$

$$w_2 = w_2 + \text{the slope in the } x_2 \text{ dimension} * \text{learning rate} * \text{Error}$$

$$b = b + \text{gradient} * \text{learning rate} * \text{Error}$$

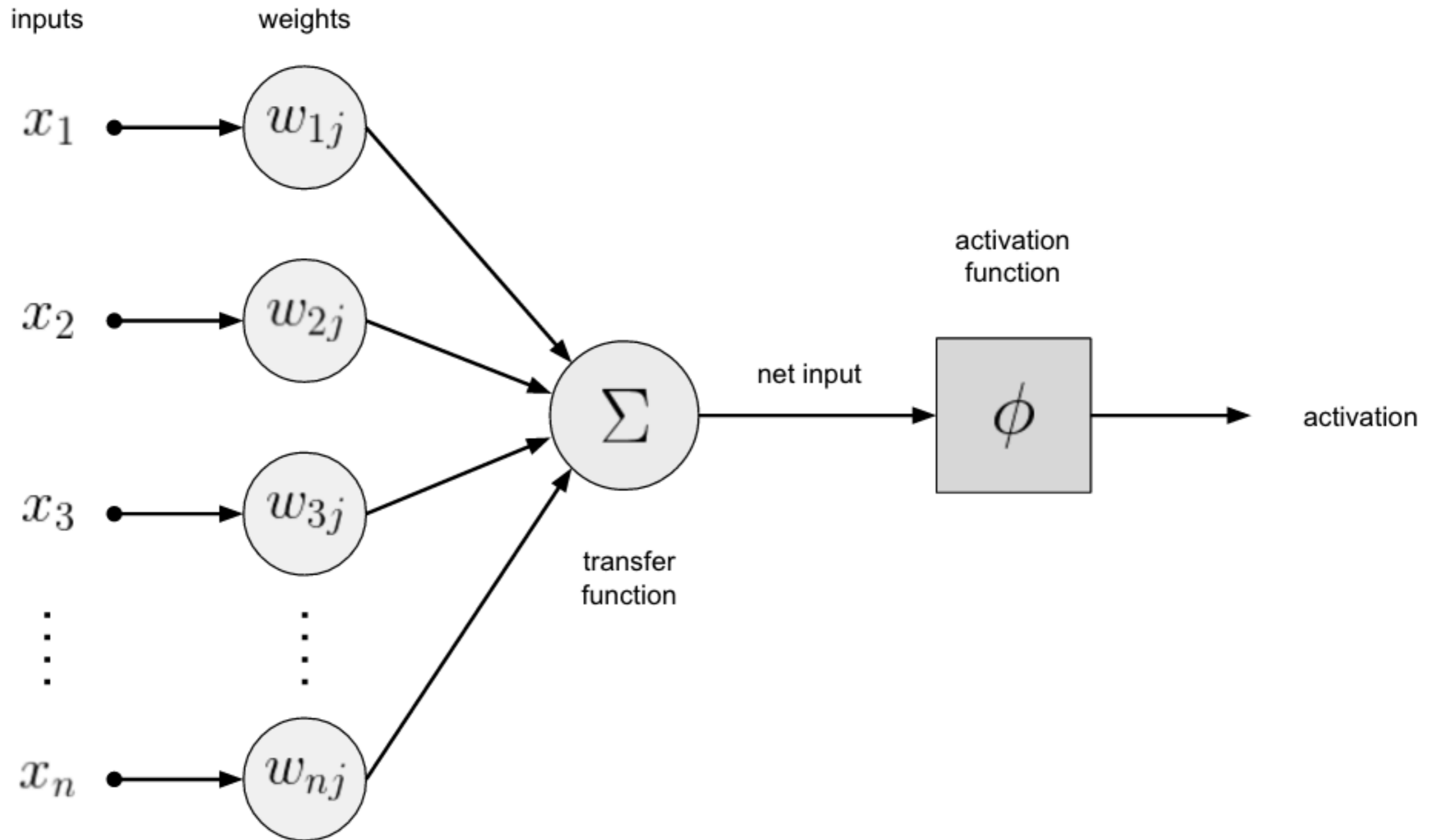
Repeat 1 & 2 until error is acceptable

Learning Rate

If too small then take too long for result to converge

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

Basic Structure of Neuron



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

First 9.1

Last 0.005

w:

1.76

3.12

Learning rate 0.01
 $w = [-10.0, -10.0]$

Loss value

First 208

Last 0.76

w:

-1.01

4.56

Learning rate 0.01
 $w = [10.0, -10.0]$

Loss value

First 52

Last 6.76

w:

10.9

-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

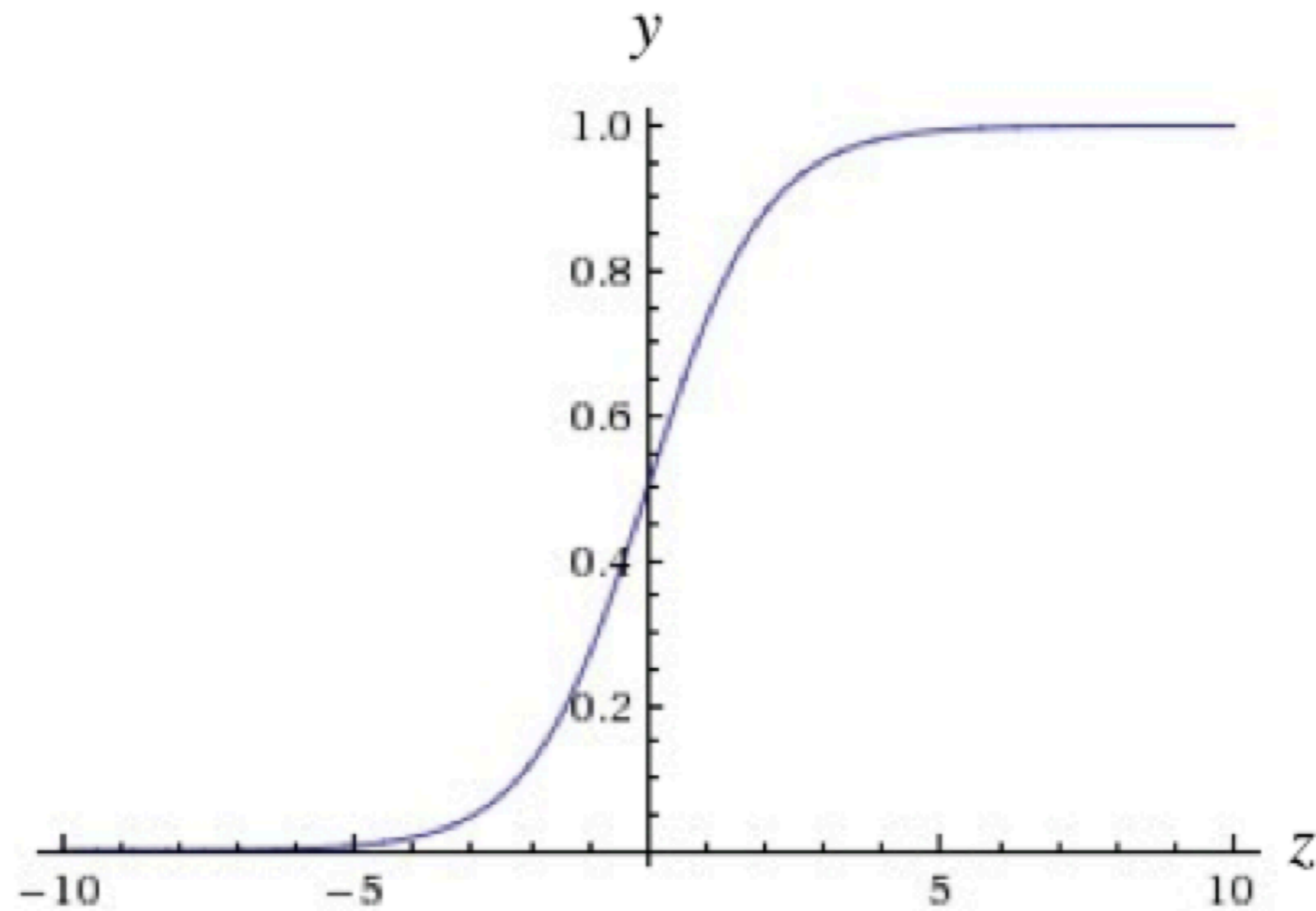
Each type needs gradient function & loss function

Layers of neurons

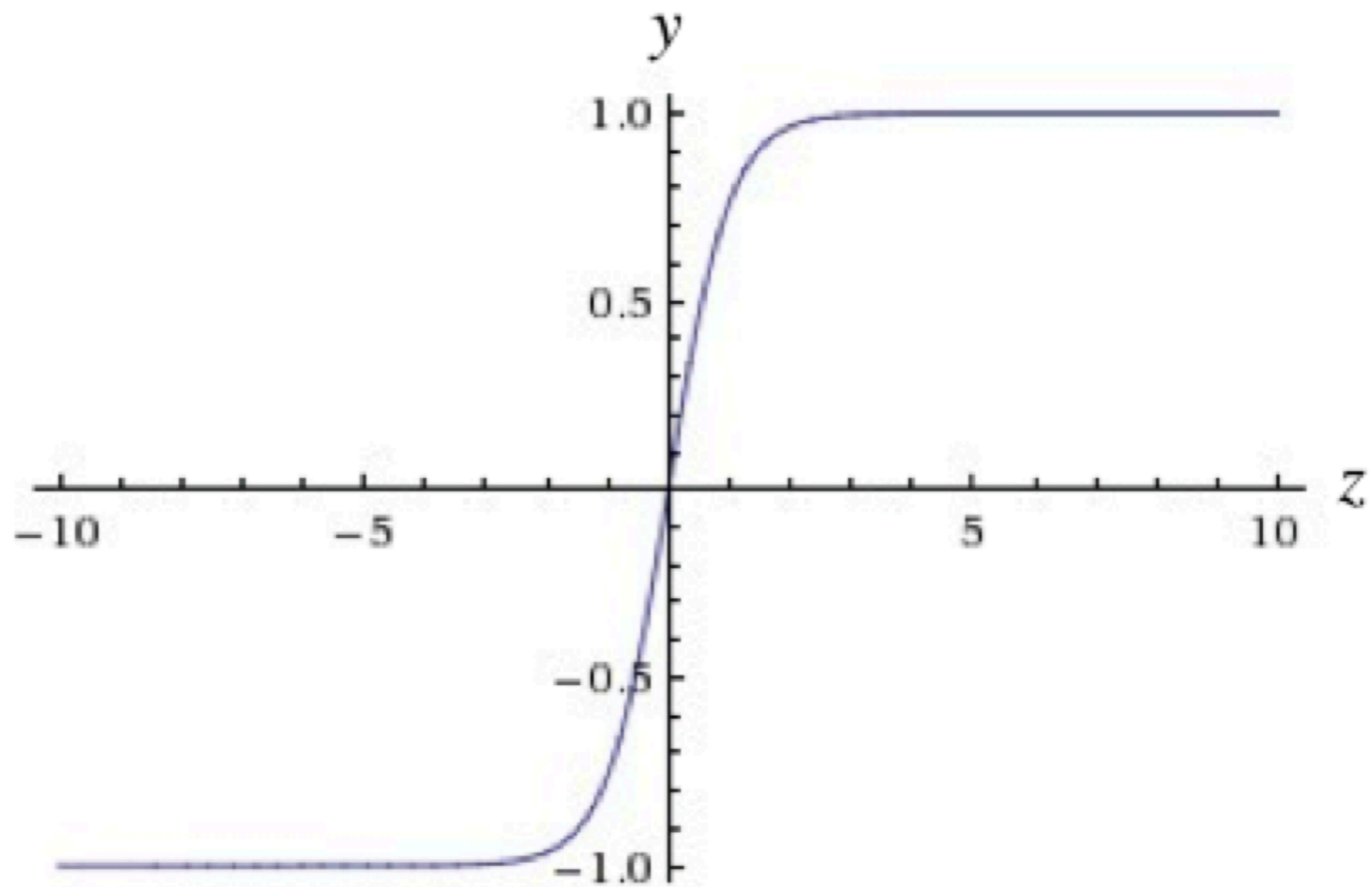
Types of Neurons/Activation Functions

Sigmoid

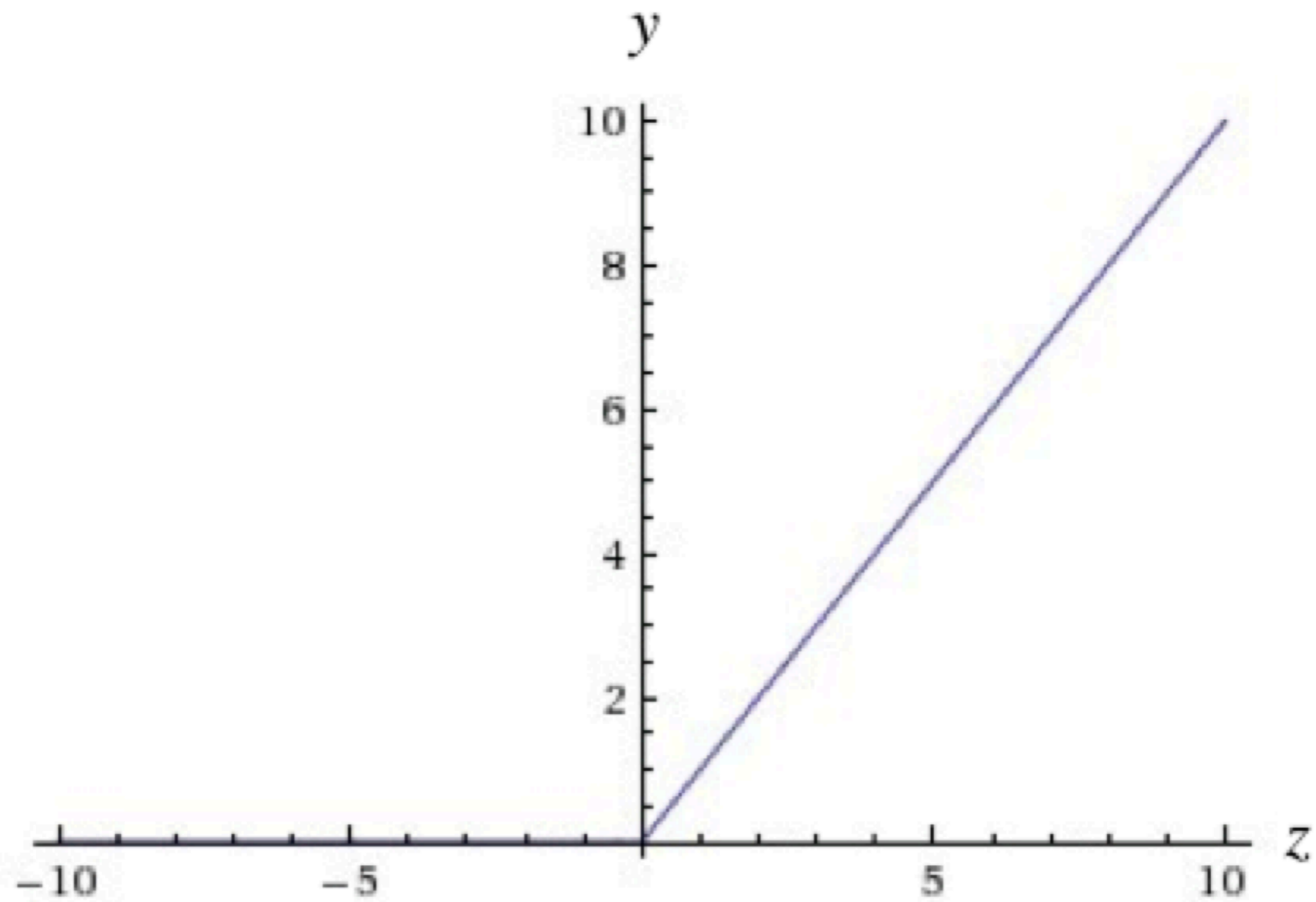
$$f(z) = \frac{1}{1 + e^{-z}}$$



Tanh



Restricted Linear Unit (ReLU)



Softmax

$$\text{softmax_norm}(x) = 1 ./ (1 + \exp(-(x - \text{mean}(x))/\text{std}(x))))$$

Recall from clustering

Often used as output neuron

Loss functions

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

mean square log error
MSLE

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

Hinge loss
Binary classification

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

Logisitic loss

Neural Networks Parameters

Input weights

Learning Rate

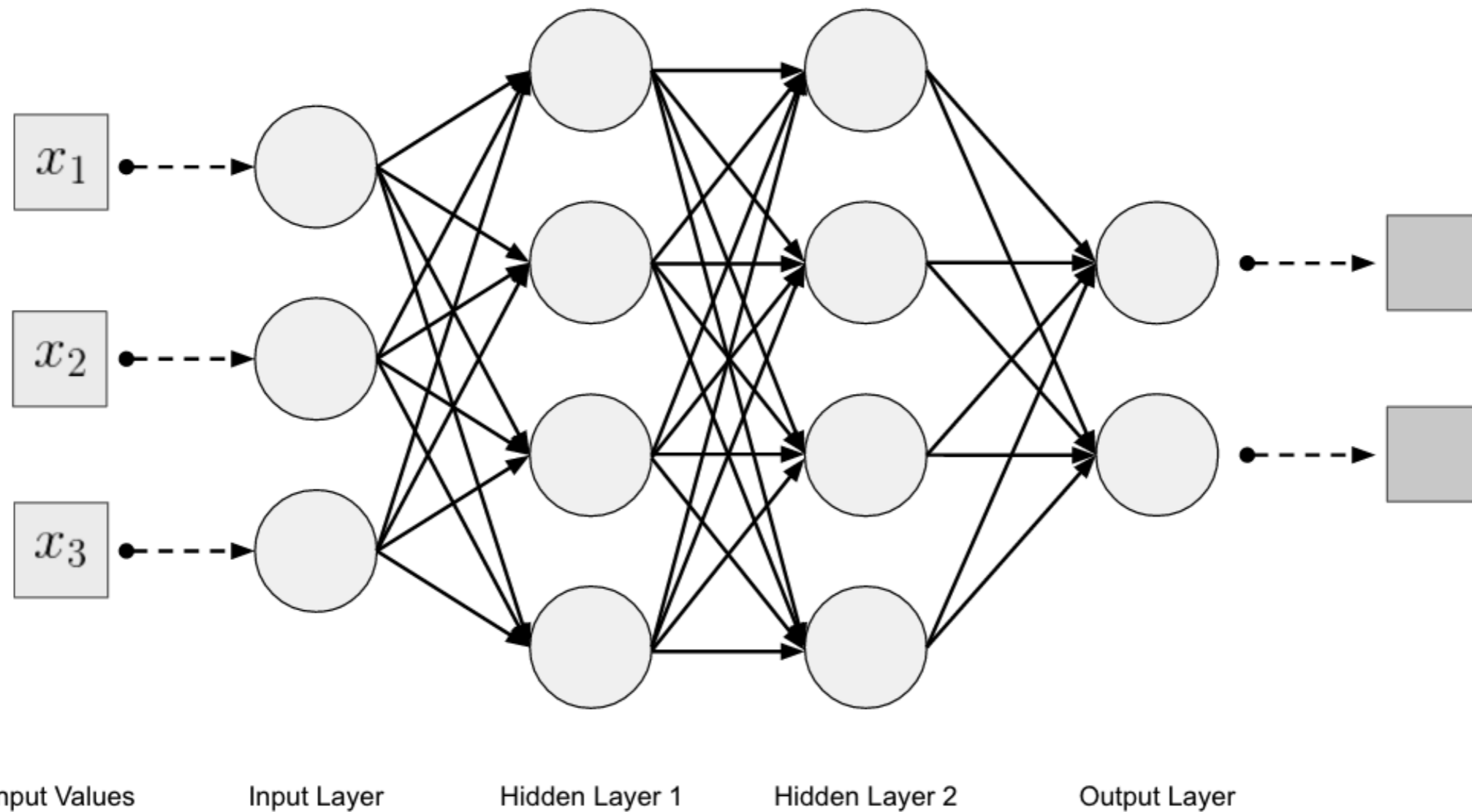
Loss function

Activation function

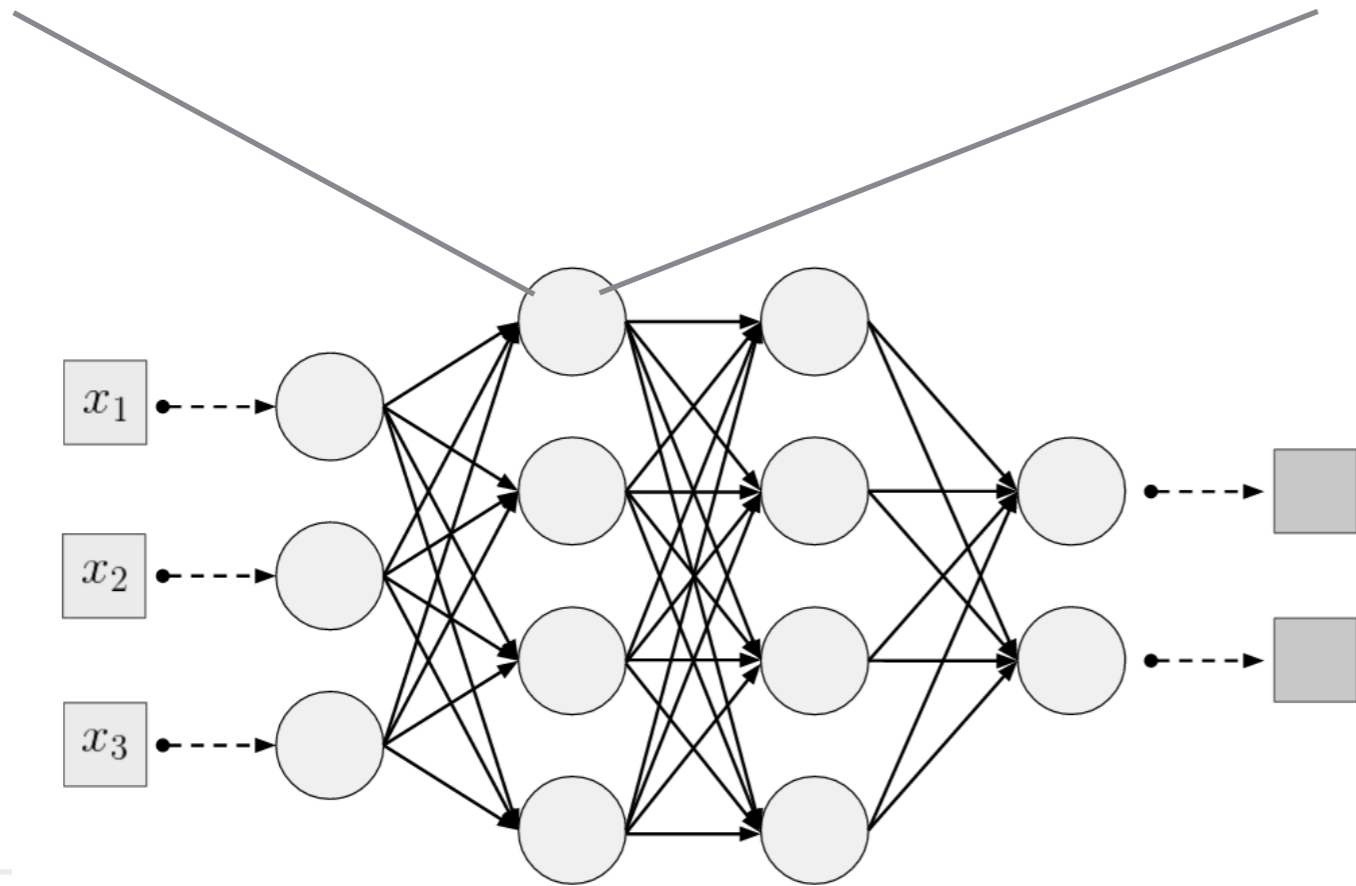
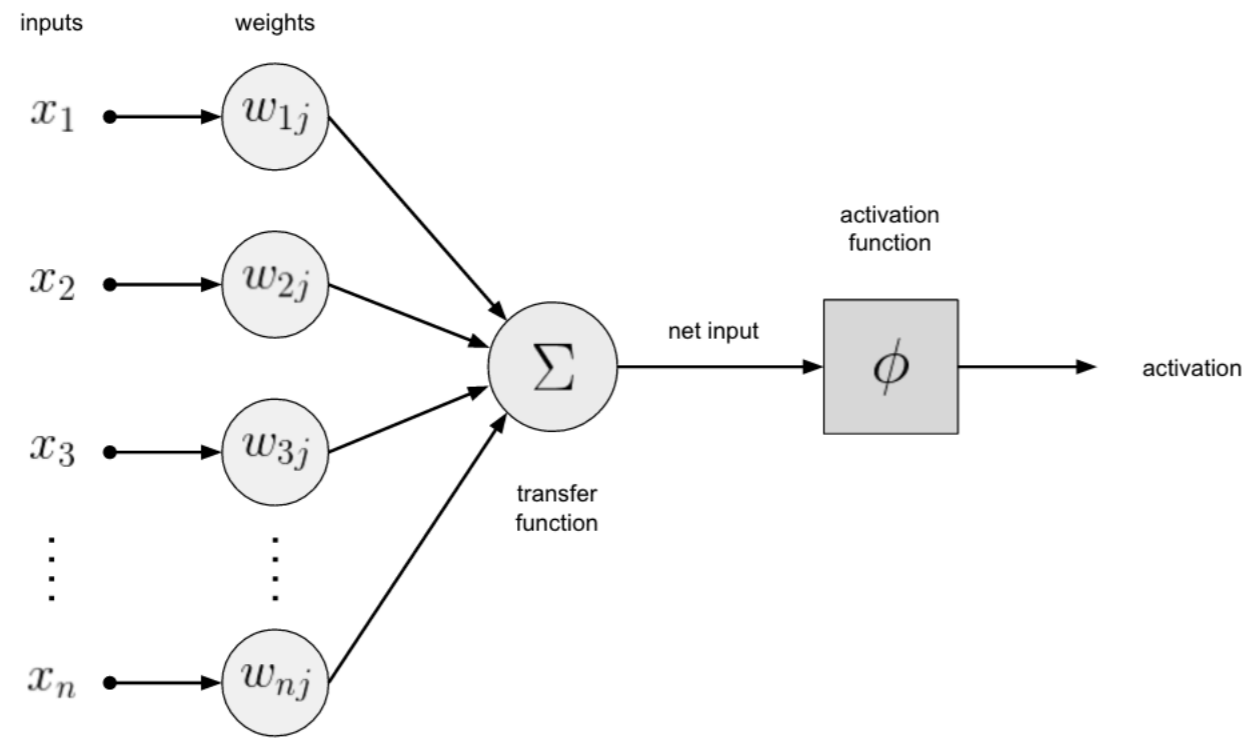
Layers

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

Create layers of neurons



One neuron

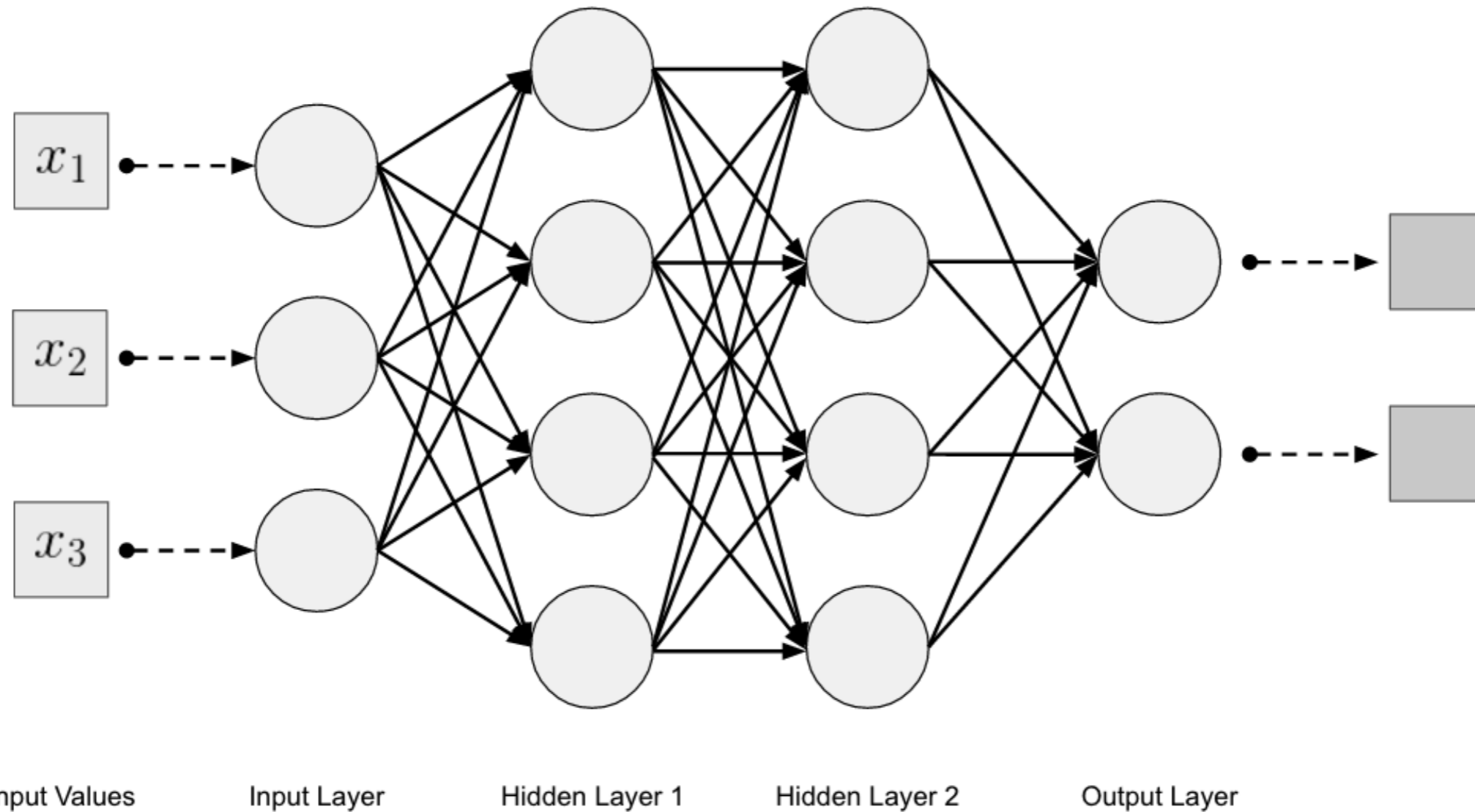


Forward Propagation

Input data goes to input layer

Each neuron passes its output to the next layer

Below is a fully connected neural network



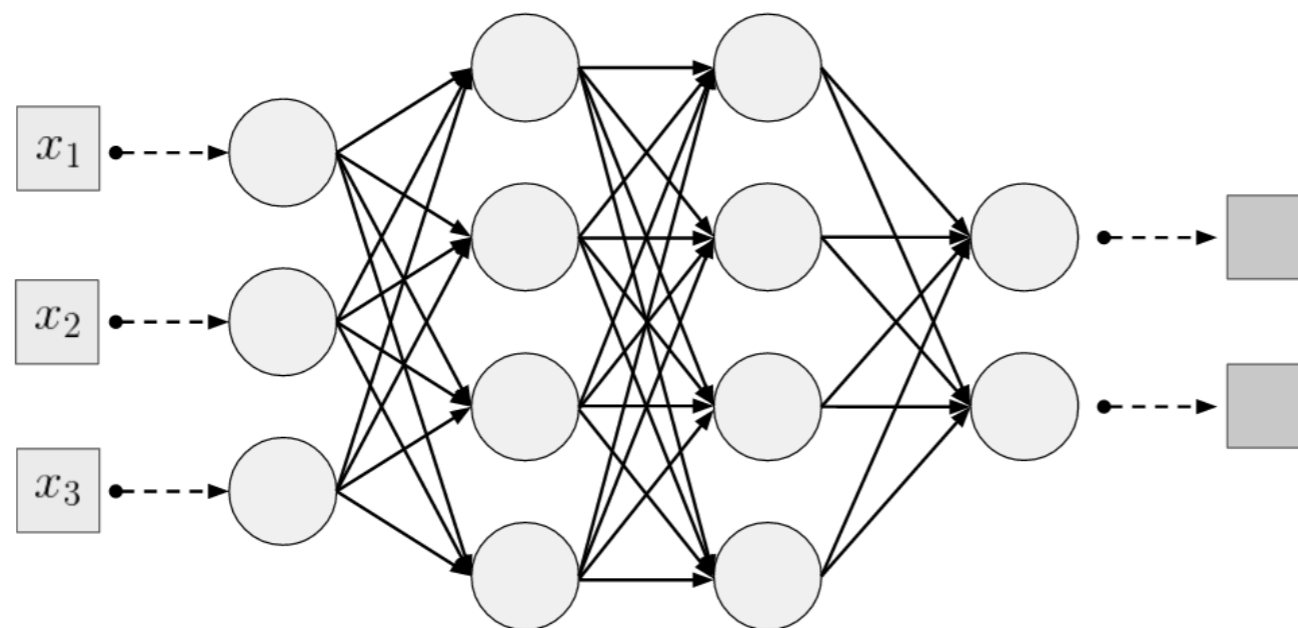
Back Propagation

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

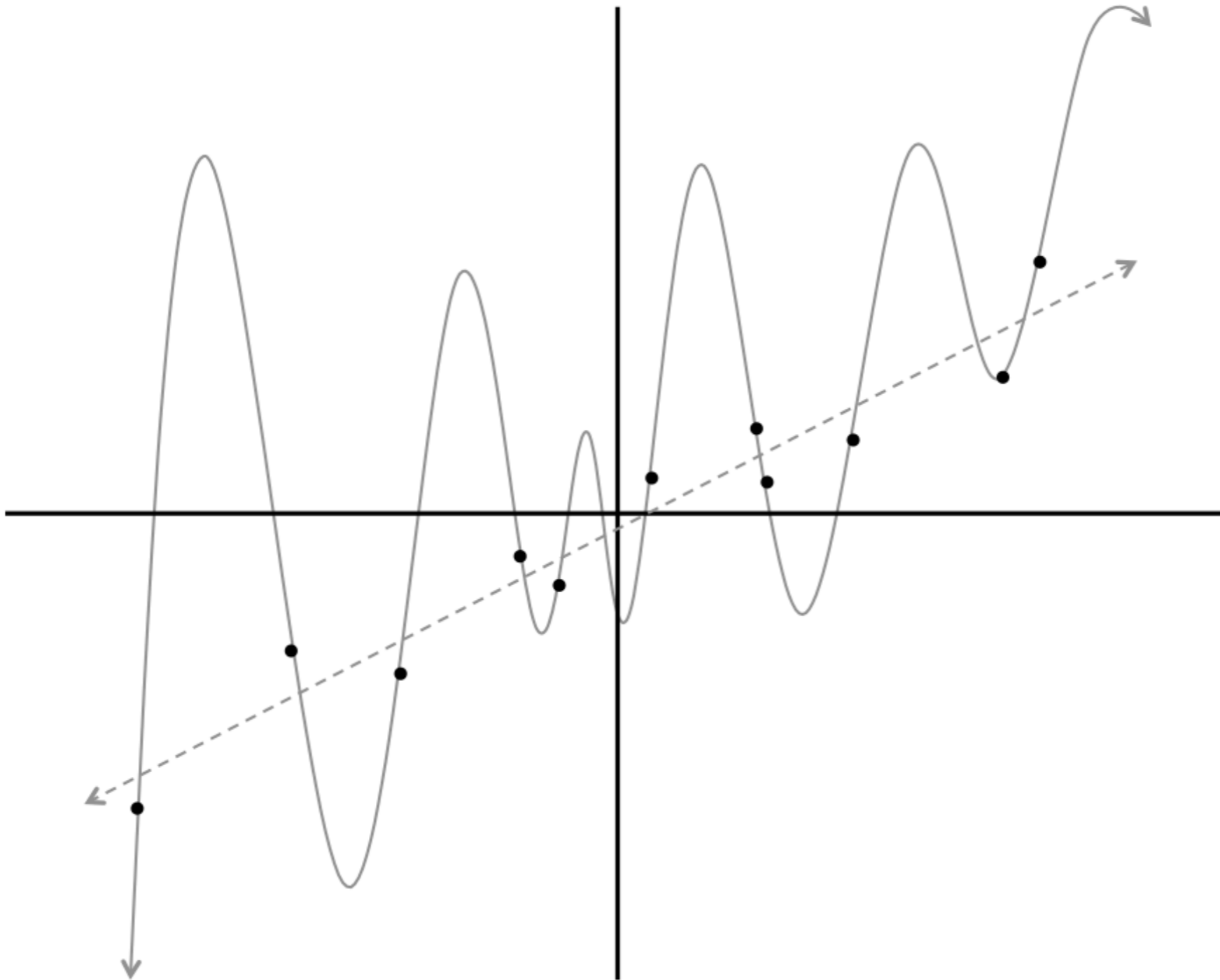
Activation function per neuron

Number of layers

Number of neurons per layer

How neurons are connected

Overfitting



Hyperparameters

Things we can change to make neural networks train better

Learning Rate

Activation functions

Weight initialization strategies

Loss functions

Normalization

Layer size & number of layers

mini-batch size

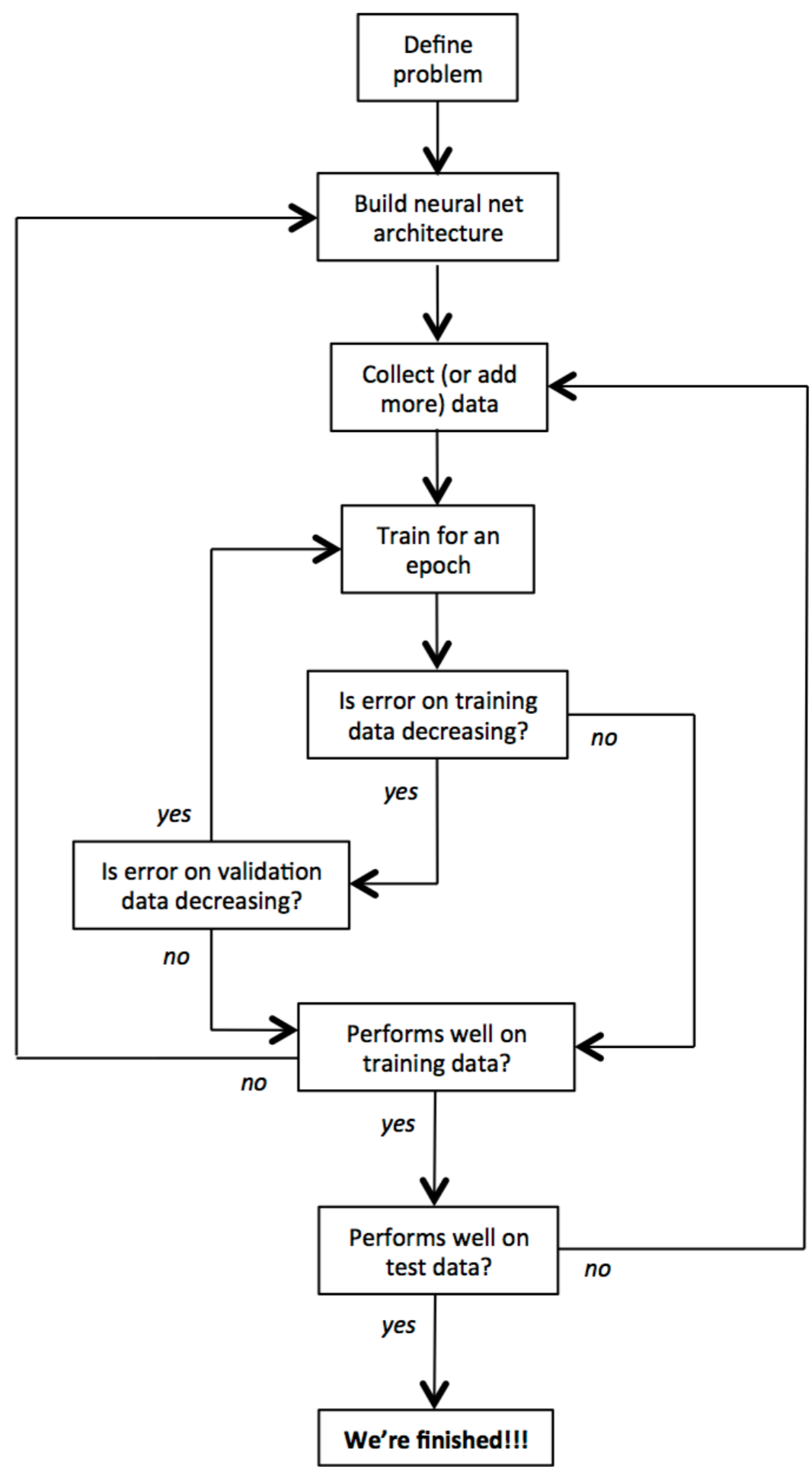
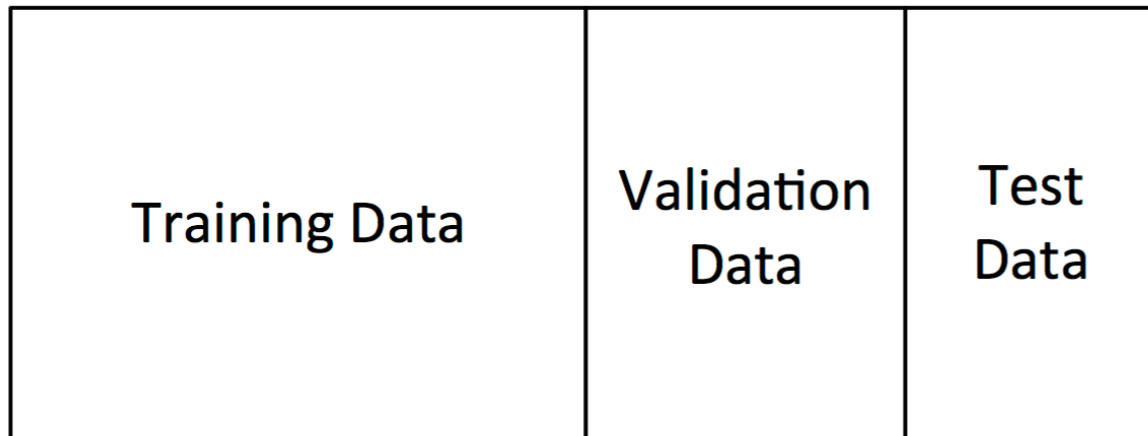
Regularization

Momentum

Sparsity

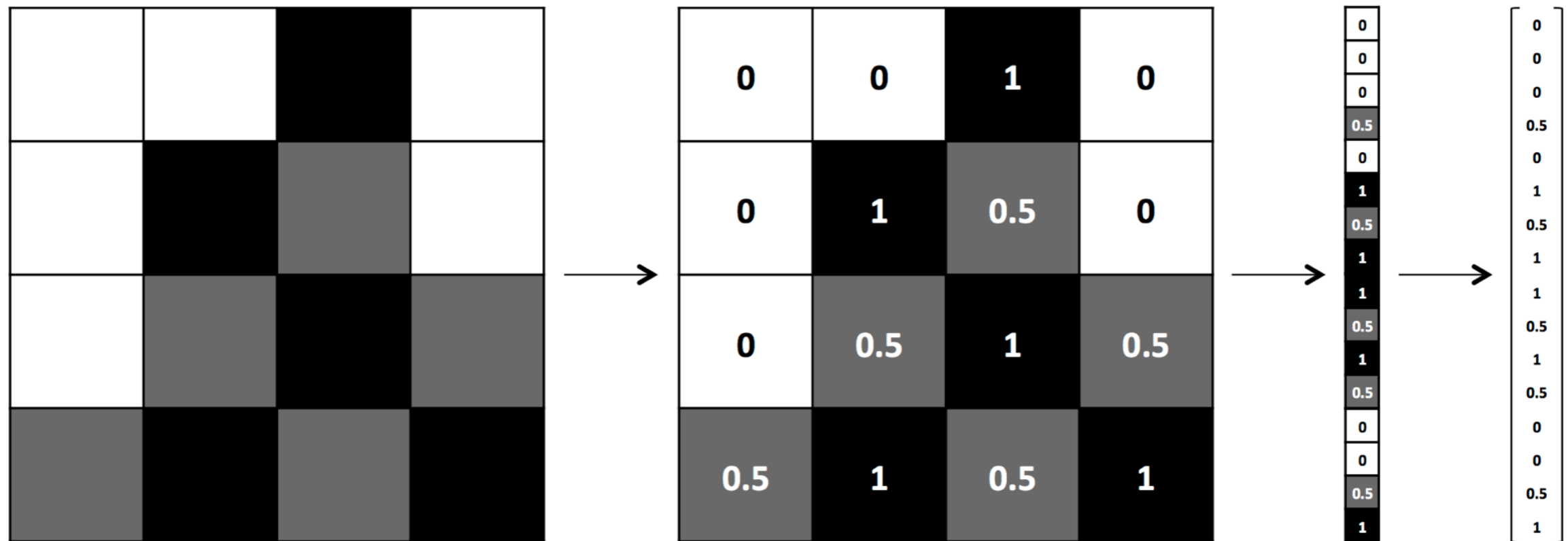
Work Flow

Full Dataset:



Input

Need to map input into vector



Images & Scaling

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

Fully connected neuron needs $32 \times 32 \times 3 = 3,072$ weights

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

Fully connected neuron needs $200 \times 200 \times 3 = 120,000$ 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

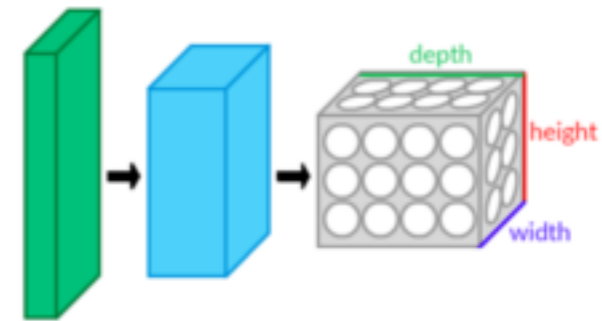
Convolutional Neural Network

Convolutional Layer

3-D network of neurons

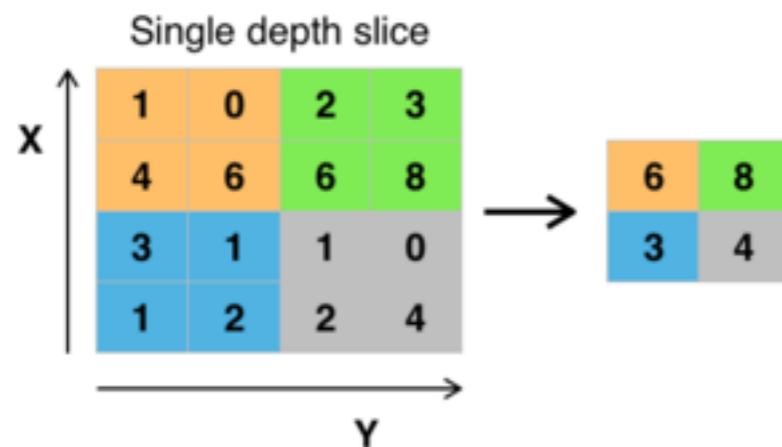
Only locally connected

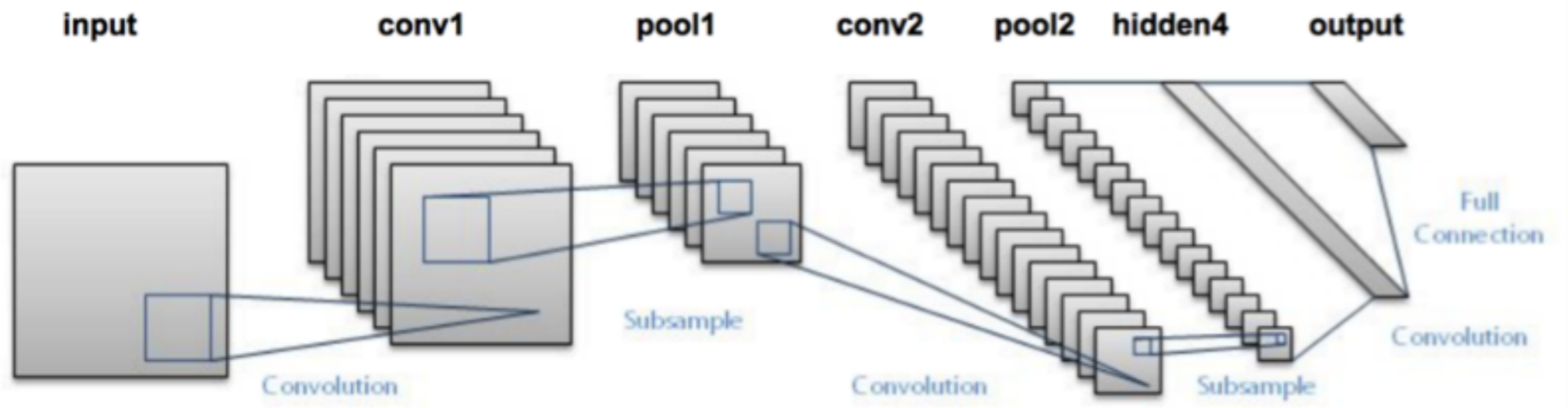
Each 2-D slice in depth share same weight



Pooling Layer

Down-sampling layer





Hello World of Deep Learning

Mixed National Institute of Standards & Technology database of handwritten digits

60,000 training images

Normalized to 20x20 pixels with grayscale



Different Methods with Error Rate

Type	Classifier	Error rate (%)
Linear classifier	Pairwise linear classifier	7.6
K-Nearest Neighbors	K-NN with non-linear deformation (P2DHMDM)	0.52
Neural network	2-layer 784-800-10	1.6
Neural network	2-layer 784-800-10	0.7
Deep neural network	6-layer 784-2500-2000-1500-1000-500-10	0.35
Convolutional neural network	Committee of 35 conv. net, 1-20-P-40-P-150-10	0.23

Spark DeepLearning

MultilayerPerceptronClassifier

feedforward artificial neural network

backpropagation for learning the model

Parameters

Number of Layers

Neurons per layer

Tolerance of iteration

Block size of the learning

Seed size

Max iteration number

```
iris = spark.read.format("csv"). \
    option("header", True). \
    option("inferSchema", True). \
    load("iris.txt")
```

```
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import VectorAssembler
from pyspark.ml import Pipeline
```

```
iris_indexer = StringIndexer(inputCol="species", outputCol="label").fit(iris)
```

```
iris_assembler = VectorAssembler(inputCols=["sepal_length", "sepal_width", "petal_length",
"petal_width"], outputCol="features")
```

```
pipeline = Pipeline(stages=[iris_indexer, iris_assembler])
iris_formated = pipeline.fit(iris).transform(iris)
```

```
from pyspark.ml.classification import MultilayerPerceptronClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

```
(train, test) = iris_formated.randomSplit([0.6, 0.4], 1234)
```

```
# specify layers for the neural network:
# input layer of size 4 (features), two intermediate of size 5 and 4
# and output of size 3 (classes)
layers = [4, 5, 4, 3]

# create the trainer and set its parameters
trainer = MultilayerPerceptronClassifier(maxIter=100, layers=layers, blockSize=128, seed=1234)

# train the model
model = trainer.fit(train)

result = model.transform(test)
predictionAndLabels = result.select("prediction", "label")
evaluator = MulticlassClassificationEvaluator(metricName="accuracy")
print("Test set accuracy = " + str(evaluator.evaluate(predictionAndLabels)))
```

```
Test set accuracy = 0.9607843137254902
```

```
result \  
  .filter(result.label != result.prediction) \  
  .select("label","rawPrediction","probability", "prediction") \  
  .show()
```

```
+-----+-----+-----+-----+  
|label|      rawPrediction|      probability|prediction|  
+-----+-----+-----+-----+  
|  1.0|[38.1539357915169...|[0.9999999999999719...|      0.0|  
|  1.0|[37.2094508219486...|[0.999999999999999...|      0.0|  
+-----+-----+-----+-----+
```