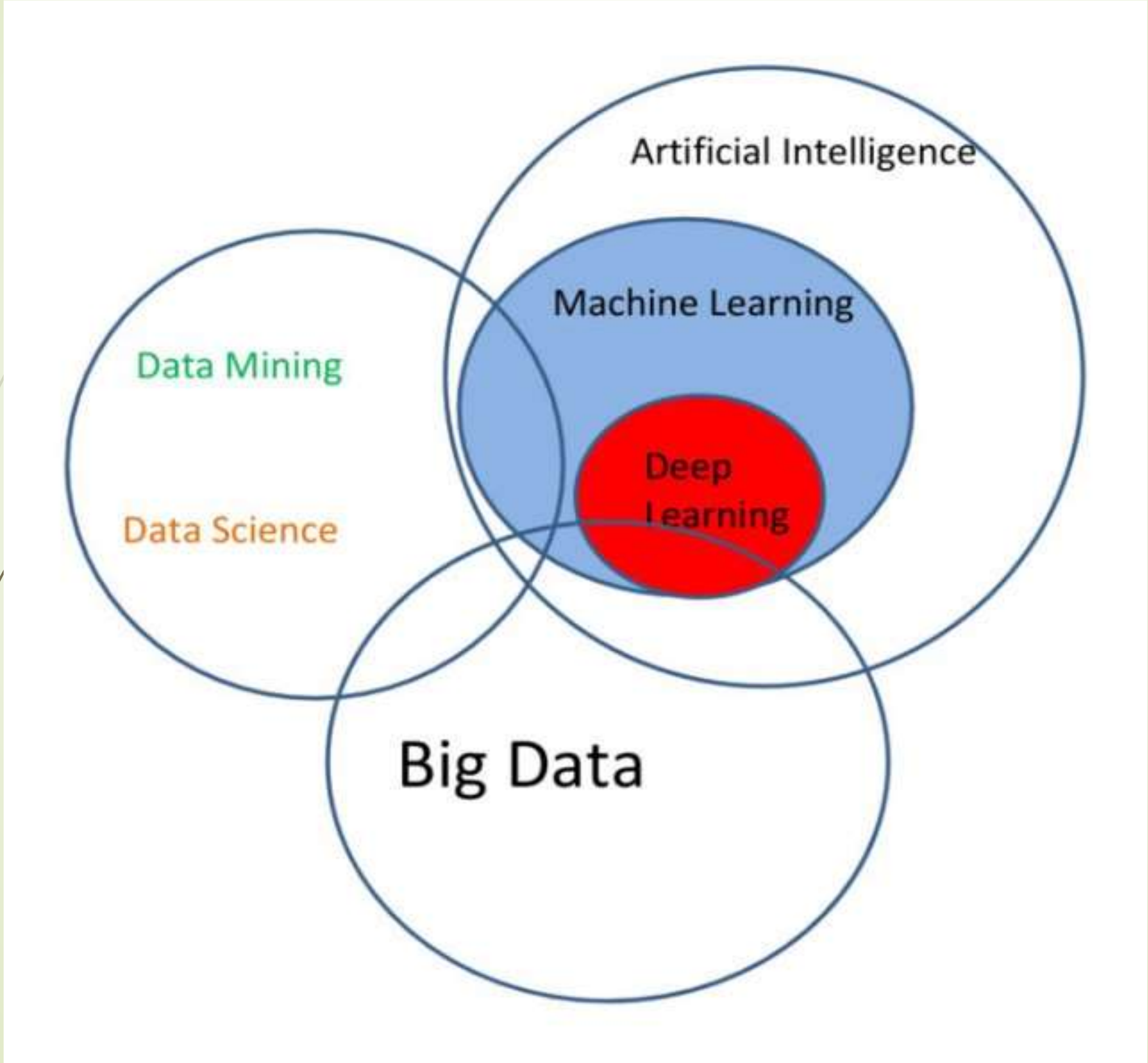
Highlights/Overview

- intro to AI/ML/DL/NNs
- Hidden layers/Initialization/Neurons per layer
- cost function/gradient descent/learning rate
- Dropout rate
- Activation function
- Linear Regression
- What are CNNs
- Filters/ReLU/MaxPooling
- Keras and CNNs
- TensorFlow 1.x

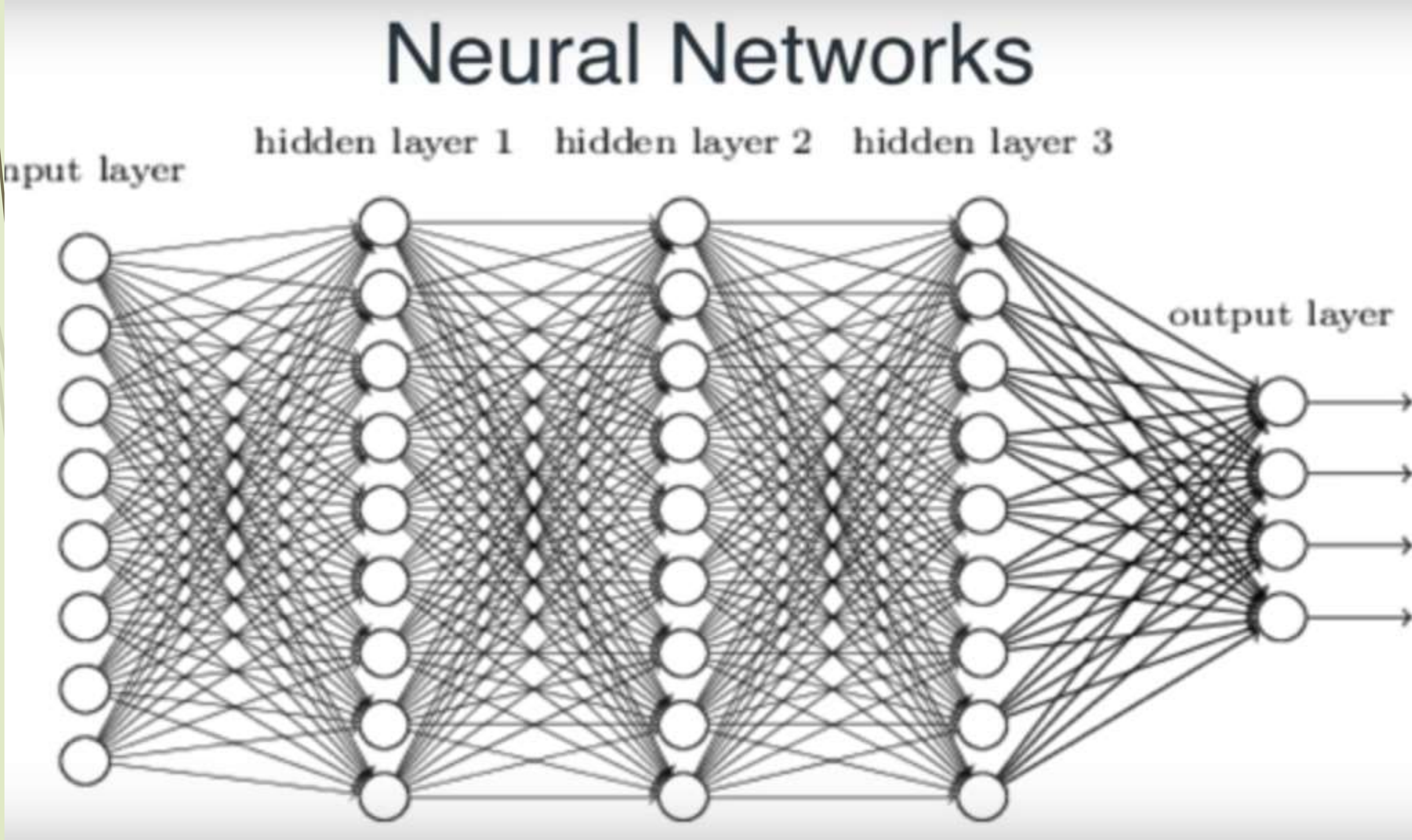
The Data/AI Landscape



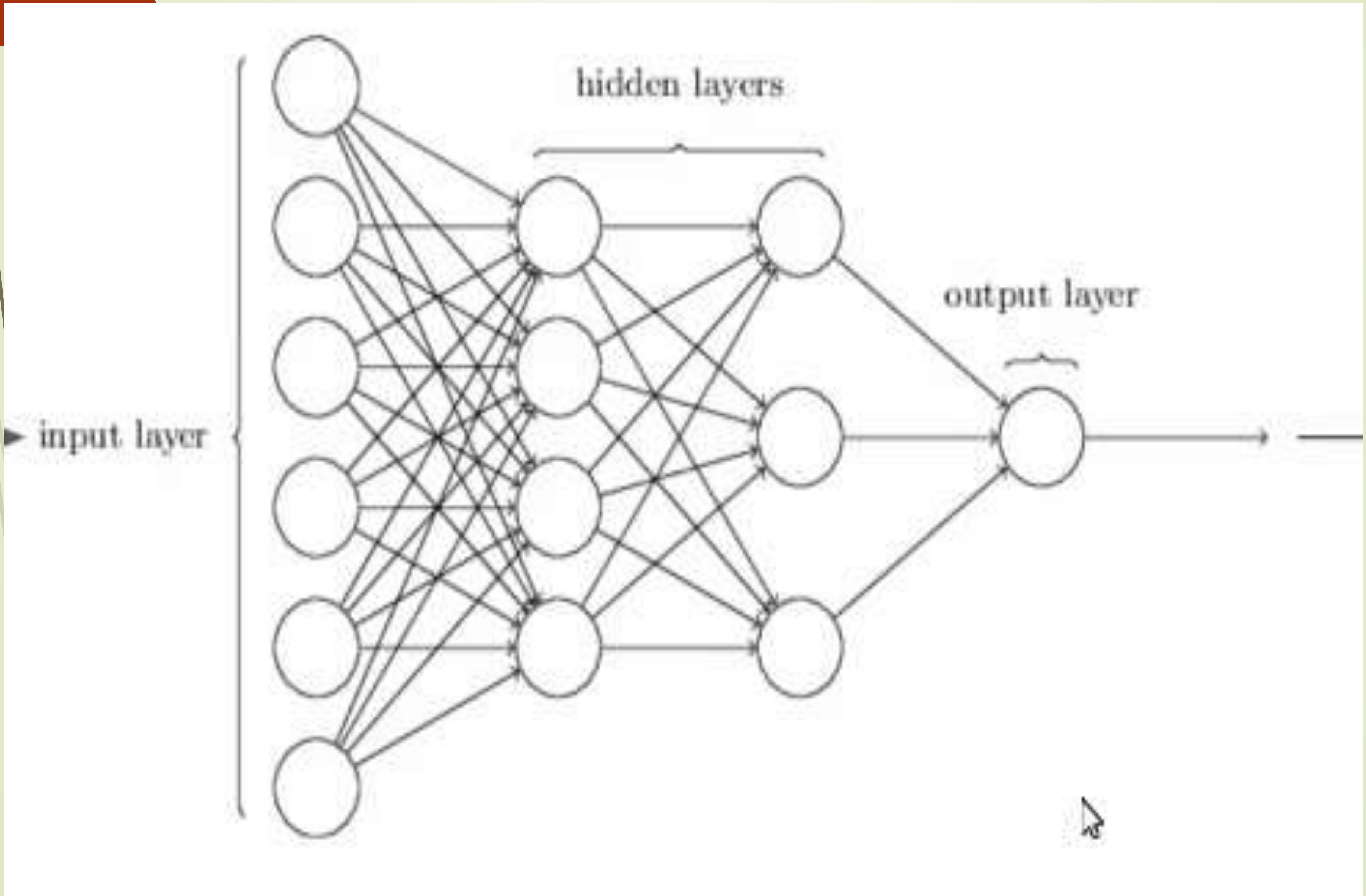
Use Cases for Deep Learning

- computer vision
- speech recognition
- image processing
- bioinformatics
- social network filtering
- drug design
- Recommendation systems
- Bioinformatics
- Mobile Advertising
- Many others

NN 3 Hidden Layers: Classifier



NN: 2 Hidden Layers (Regression)



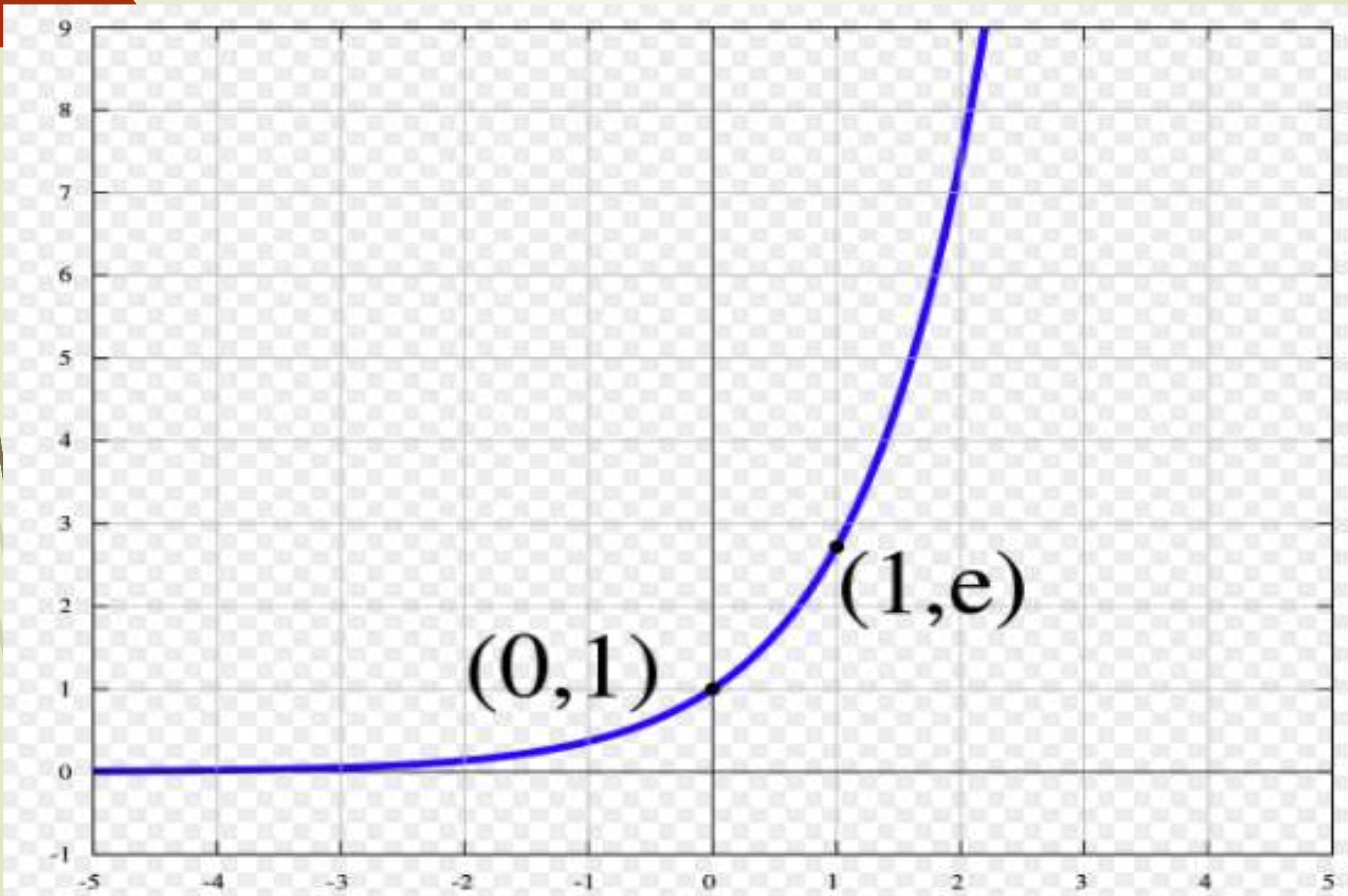
Titanic Dataset (portion)

	A	B	C	D	E	F	G	H	I	J	K	L
1	PassengerI	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
2	1	0	3	Braund, Mr	male	22	1	0	A/5 21171	7.25		S
3	2	1	1	Cumings, M	female	38	1	0	PC 17599	71.2833	C85	C
4	3	1	3	Heikkinen,	female	26	0	0	STON/O2. 3	7.925		S
5	4	1	1	Futrelle, M	female	35	1	0	113803	53.1	C123	S
6	5	0	3	Allen, Mr. \	male	35	0	0	373450	8.05		S
7	6	0	3	Moran, Mr.	male		0	0	330877	8.4583		Q
8	7	0	1	McCarthy,	male	54	0	0	17463	51.8625	E46	S
9	8	0	3	Palsson, M	male	2	3	1	349909	21.075		S
10	9	1	3	Johnson, M	female	27	0	2	347742	11.1333		S
11	10	1	2	Nasser, Mr	female	14	1	0	237736	30.0708		C
12	11	1	3	Sandstrom,	female	4	1	1	PP 9549	16.7	G6	S
13	12	1	1	Bonnell, M	female	58	0	0	113783	26.55	C103	S
14	13	0	3	Saunderco	male	20	0	0	A/5. 2151	8.05		S
15	14	0	3	Andersson,	male	39	1	5	347082	31.275		S

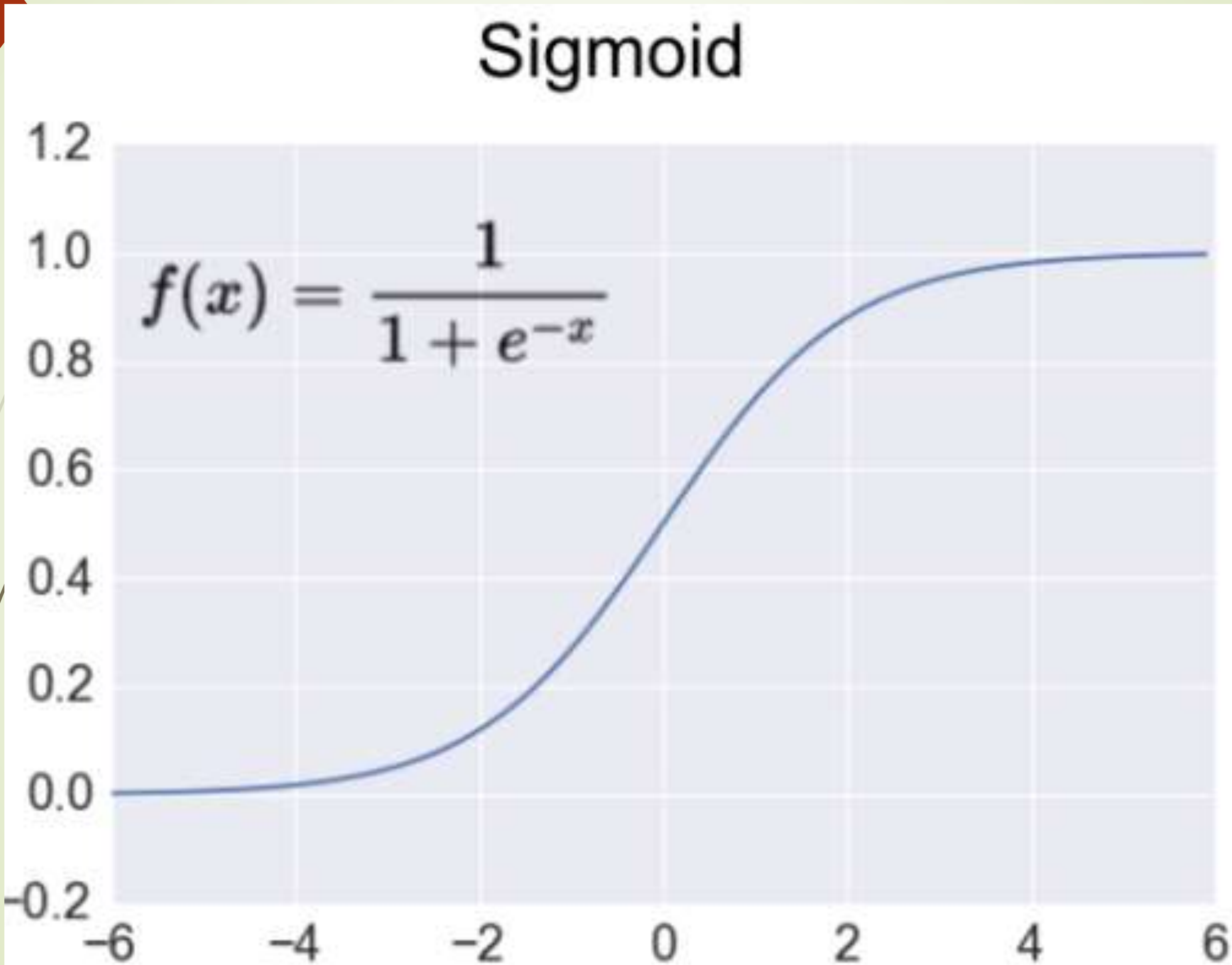
target variable

Classification and Deep Learning

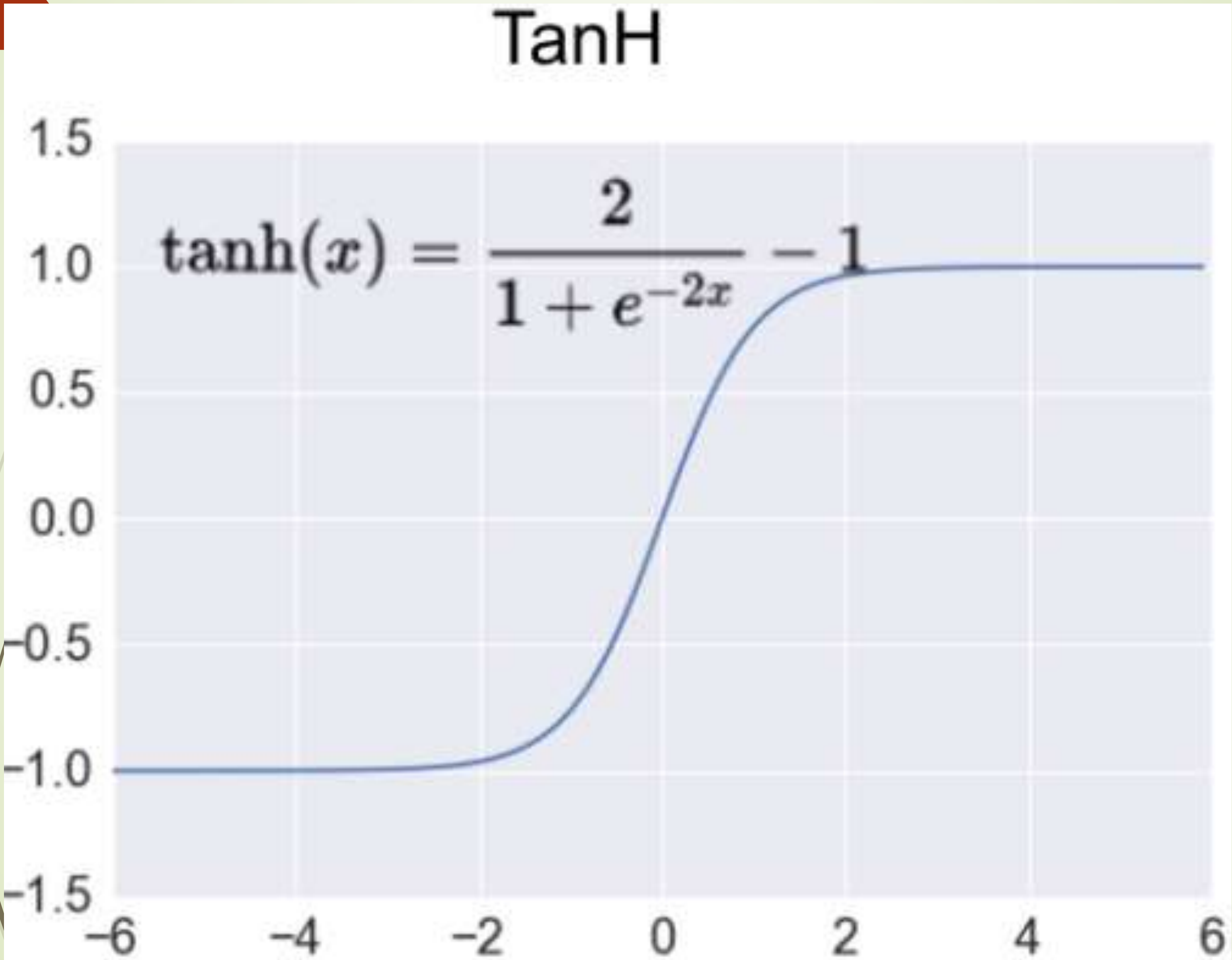
Euler's Function (e: 2.71828. . .)



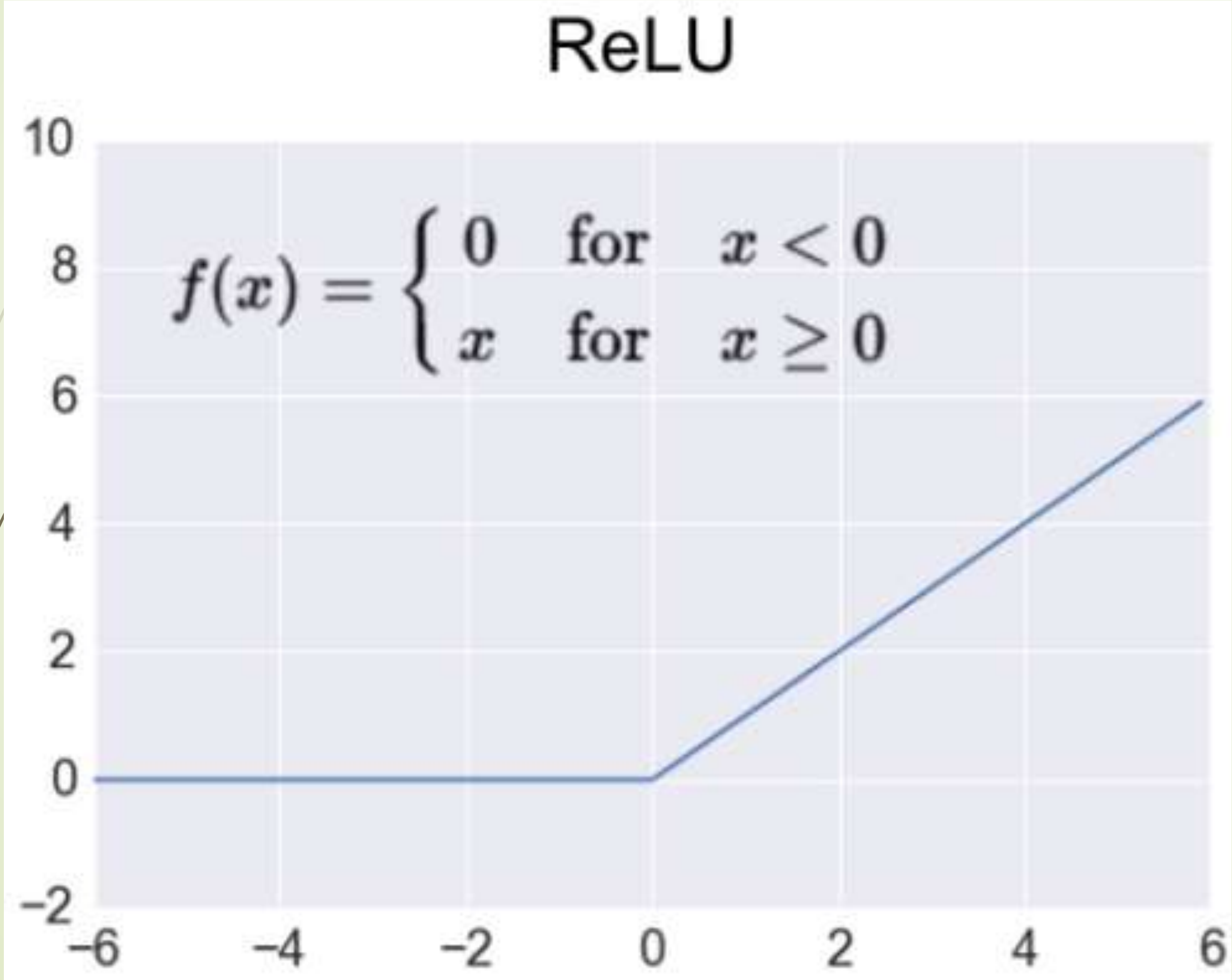
The sigmoid Activation Function



The tanh Activation Function



The ReLU Activation Function



The softmax Activation Function

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

Activation Functions in Python

```
import numpy as np
...
# Python sigmoid example:
z = 1/(1 + np.exp(-np.dot(W, x)))
...
# Python tanh example:
z = np.tanh(np.dot(W,x));

# Python ReLU example:
z = np.maximum(0, np.dot(W, x))
```

What's the "Best" Activation Function?

- Initially: sigmoid was popular
- Then: tanh became popular
- Now: RELU is preferred (better results)
- Softmax: for FC (fully connected) layers
- NB: sigmoid and tanh are used in LSTMs

Linear Regression

- One of the simplest models in ML

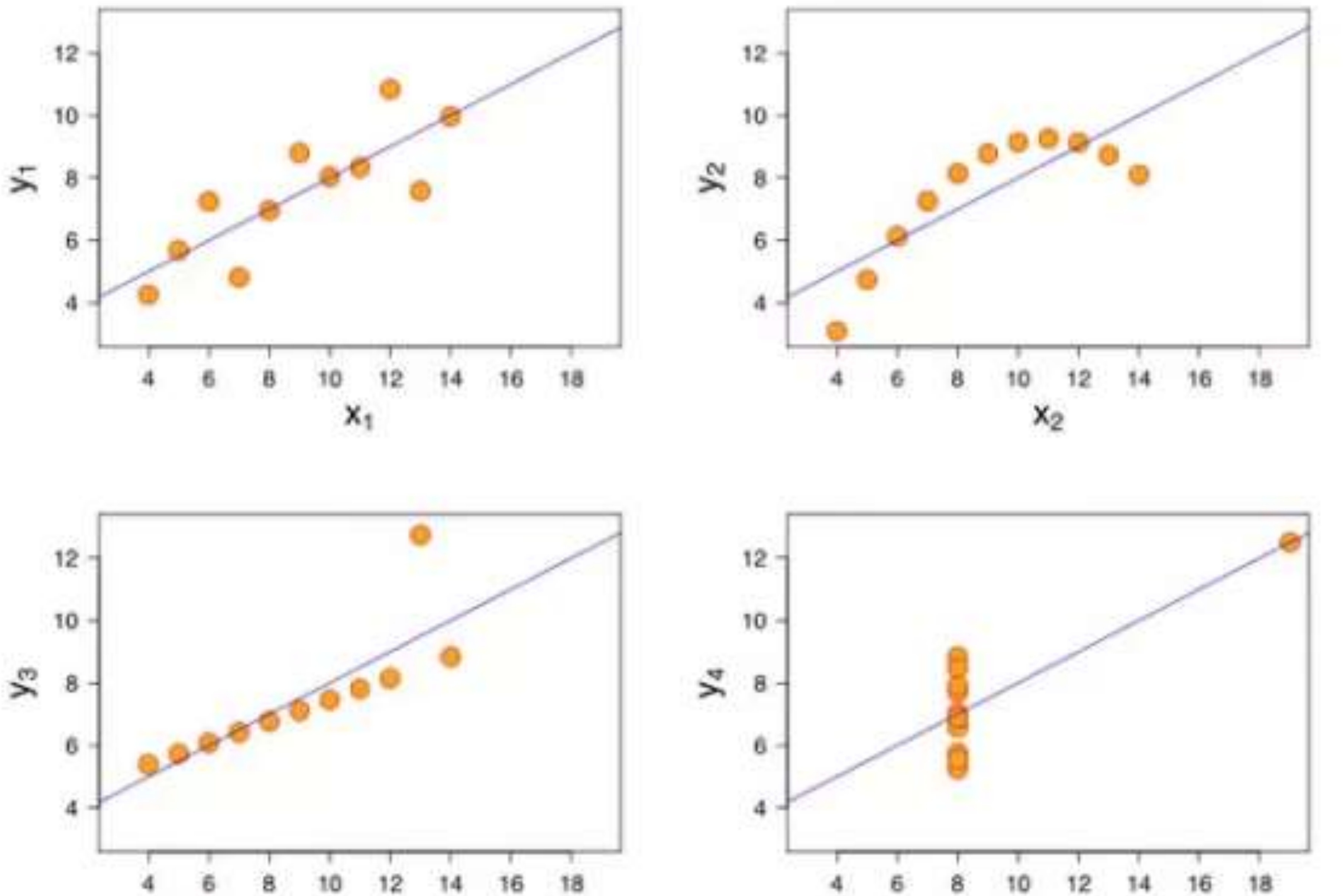
- Fits a line ($y = m*x + b$) to data in 2D

- Finds best line by minimizing MSE:

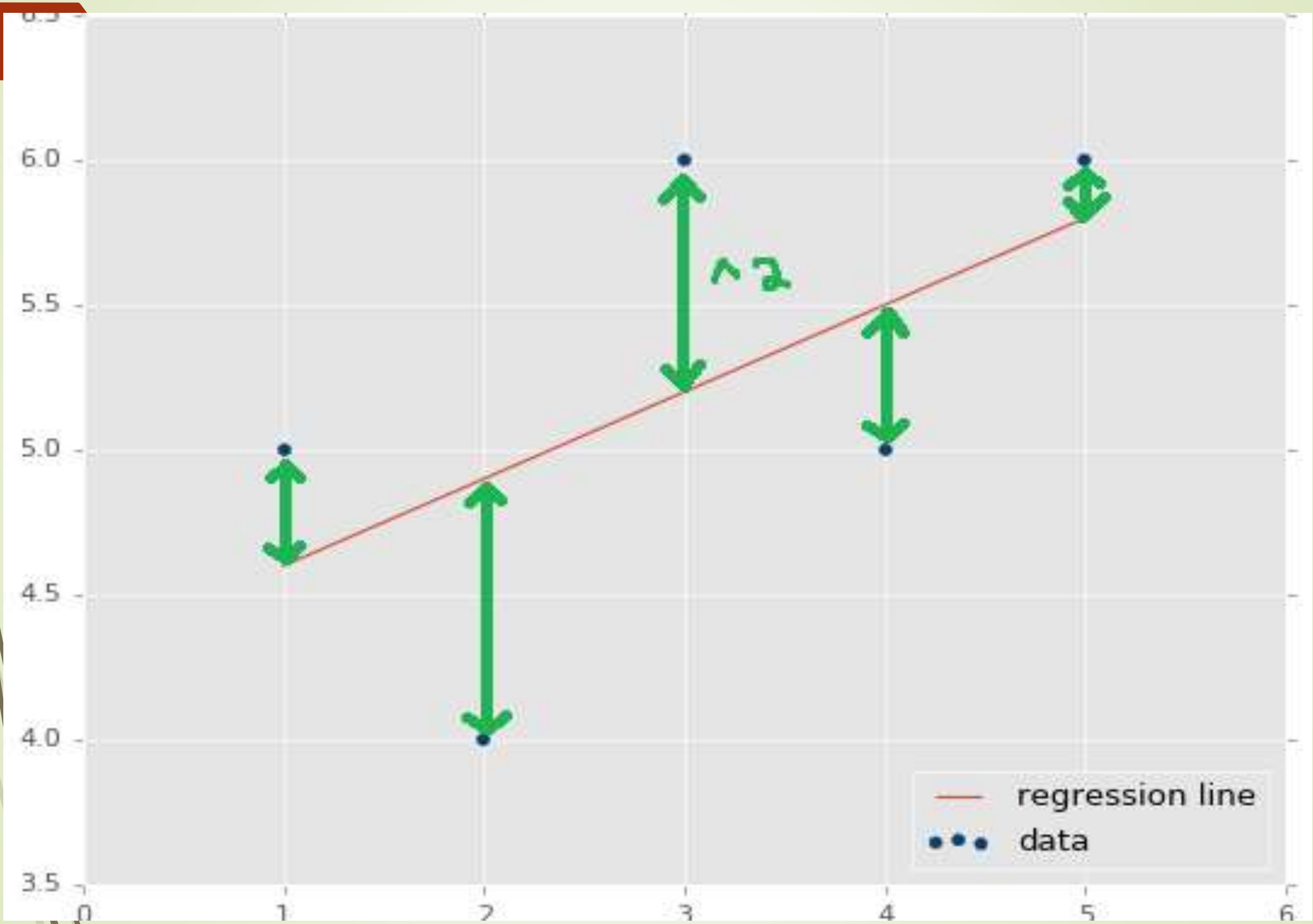
m = slope of the best-fitting line

b = y-intercept of the best-fitting line

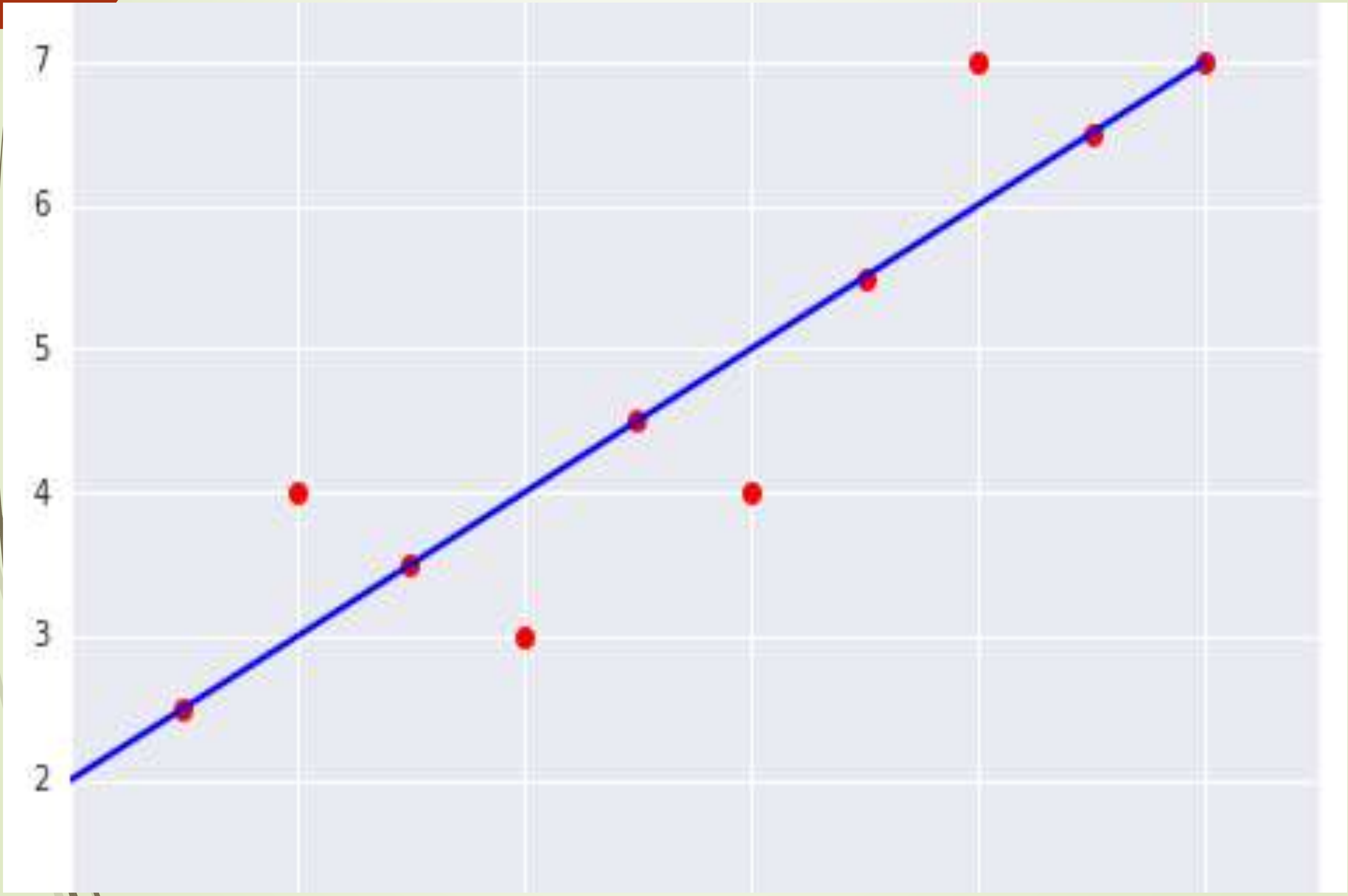
Which is Good for Linear Regression?



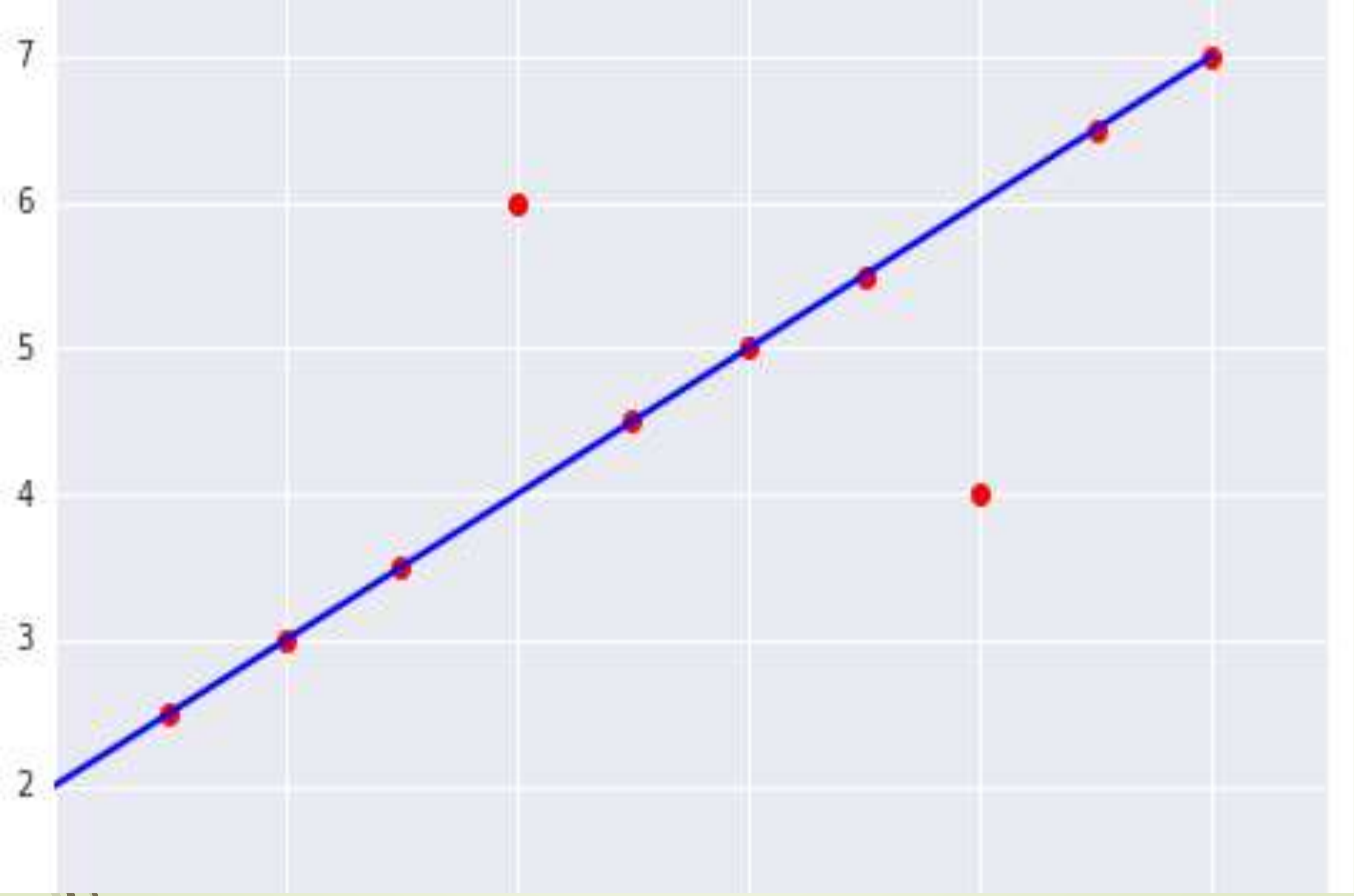
Linear Regression in 2D: example



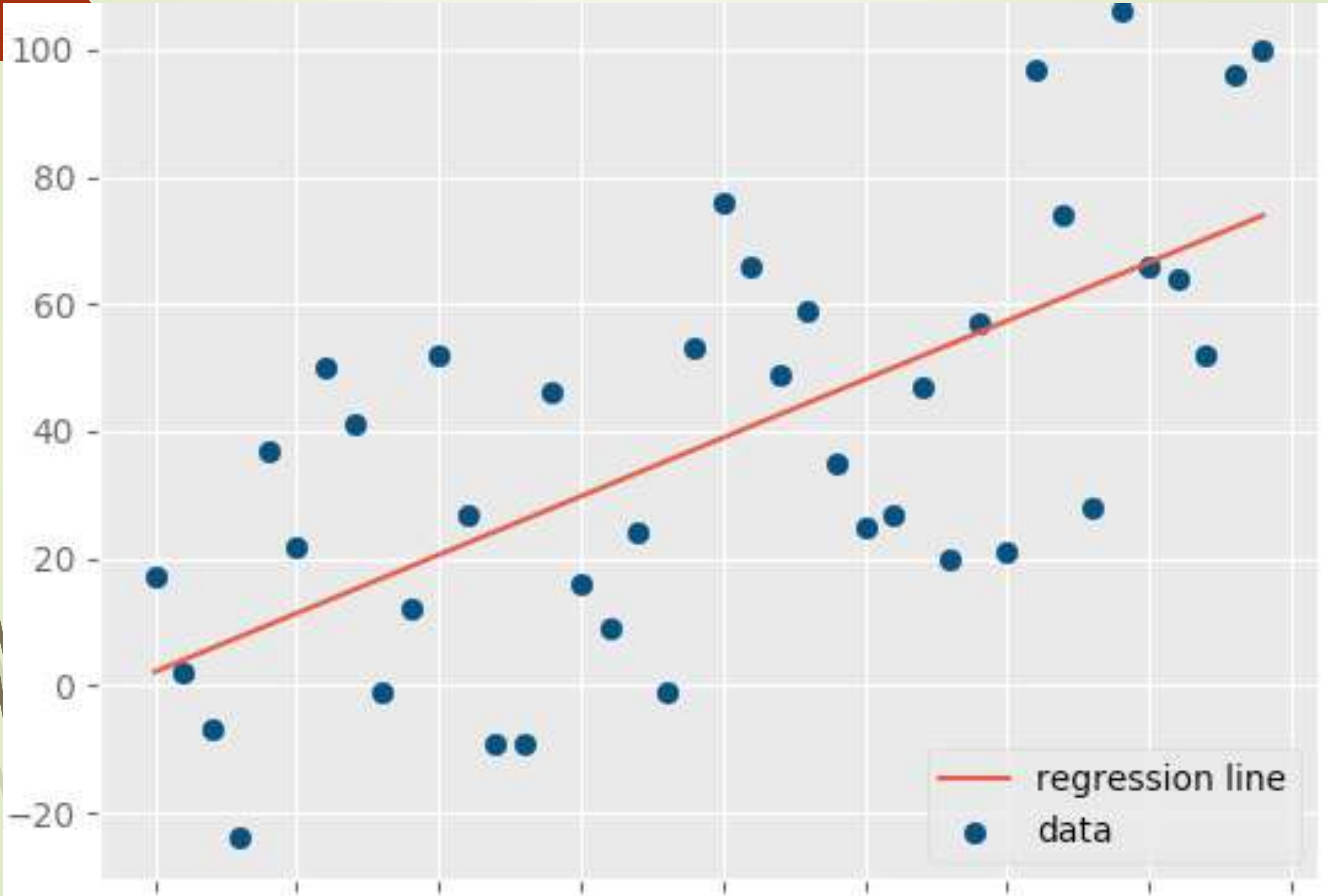
Example #1: MSE = ?



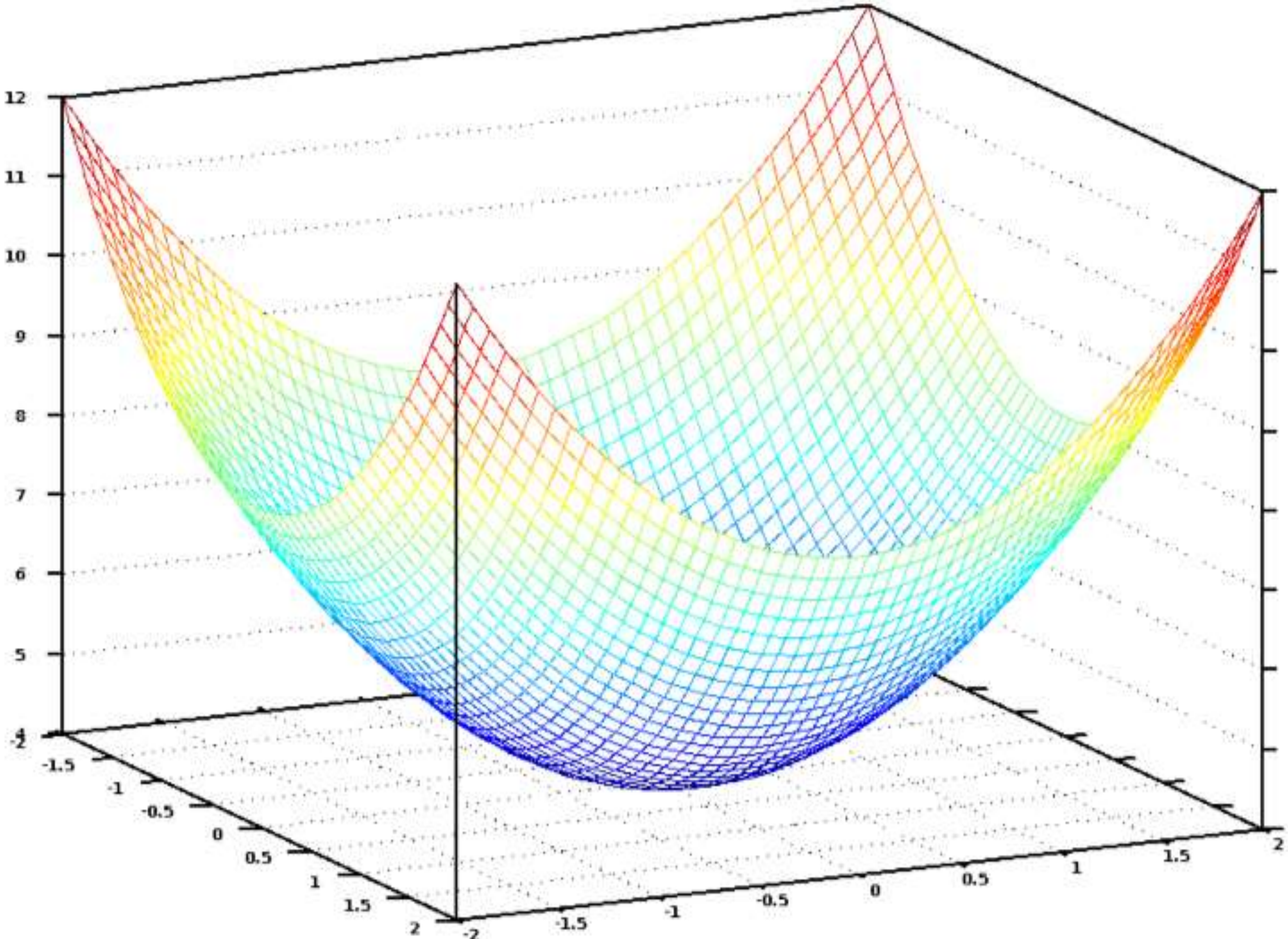
Example #2: MSE = ?



Example #3 (Random Points)



Sample Cost Function #1 (MSE)



Linear Regression: example #1

- One feature (independent variable):

X = number of square feet

- Predicted value (dependent variable):

Y = cost of a house

- A very “coarse grained” model
- We can devise a much better model

Linear Regression: example #2

- Multiple features:
- X1 = # of square feet
- X2 = # of bedrooms
- X3 = # of bathrooms (dependency?)
- X4 = age of house
- X5 = cost of nearby houses
- X6 = corner lot (or not): Boolean

- a much better model (6 features)

Linear Multivariate Analysis

- General form of multivariate equation:

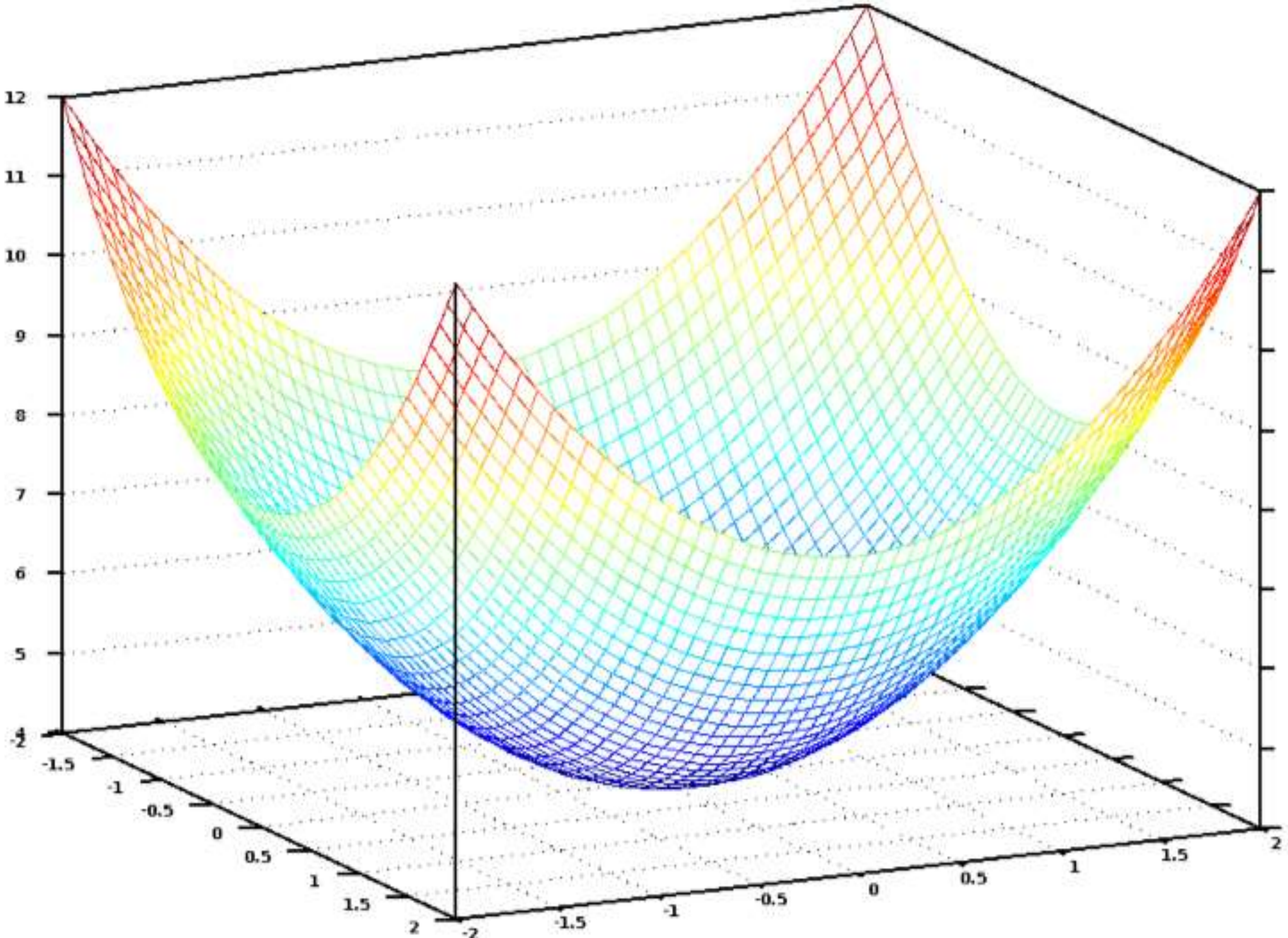
- $Y = w1*x1 + w2*x2 + . . . + wn*xn + b$
- w1, w2, . . . , wn are numeric values
- x1, x2, . . . , xn are variables (features)

- Properties of variables:

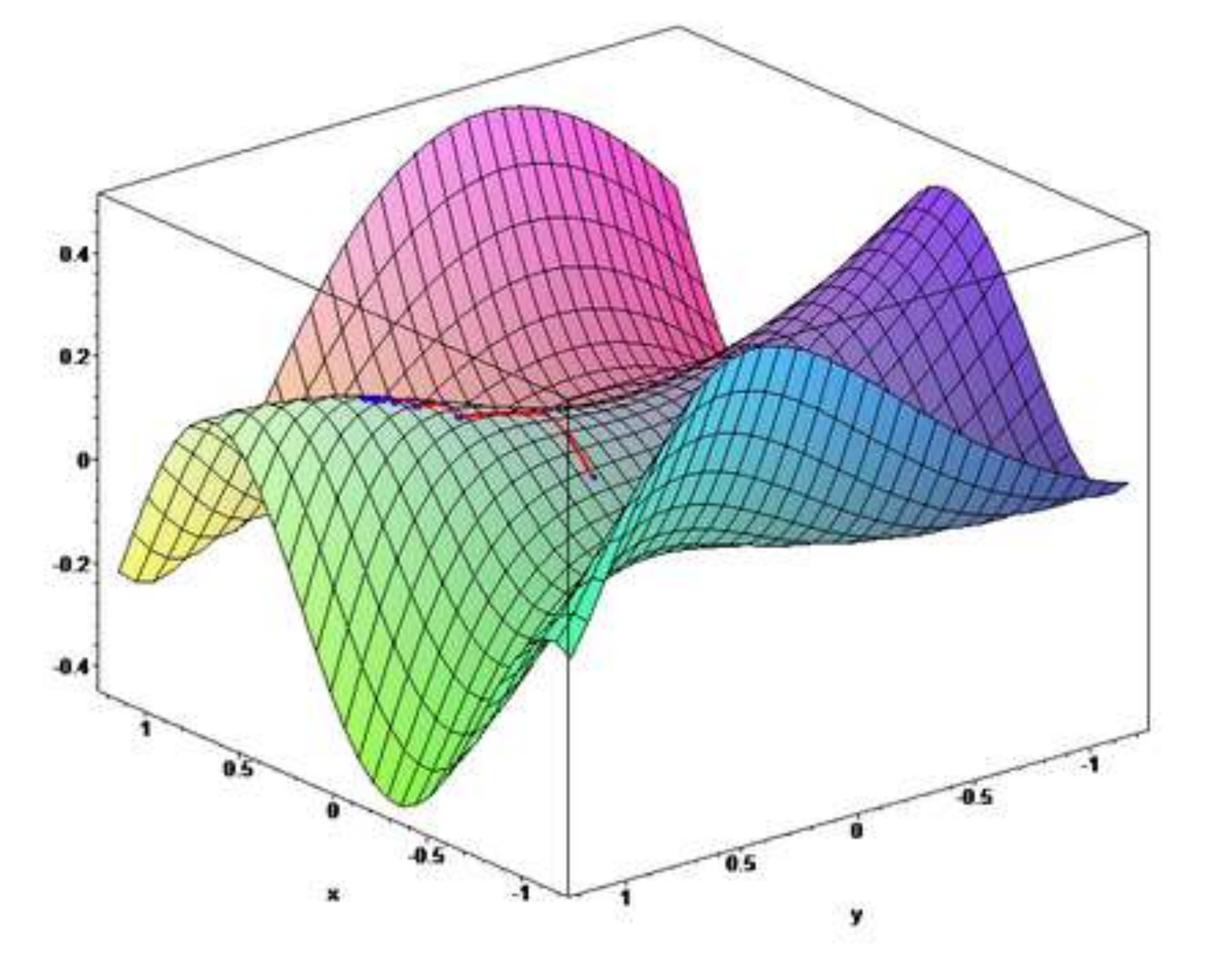
Can be independent (Naïve Bayes)

weak/strong dependencies can exist

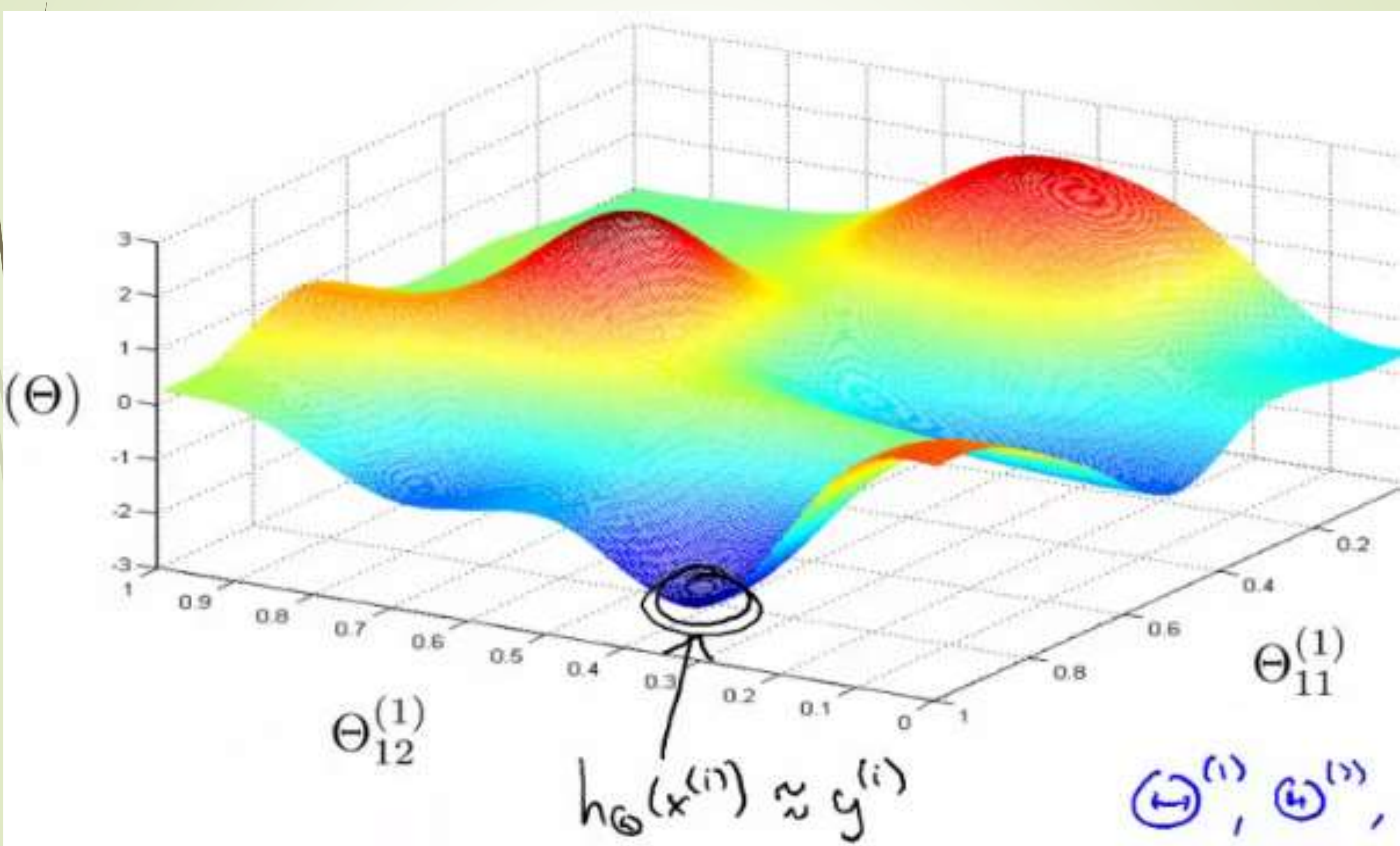
Sample Cost Function #1 (MSE)



Sample Cost Function #2



Sample Cost Function #3



Types of Optimizers

- SGD
- rmsprop
- Adagrad
- Adam
- Others

http://cs229.stanford.edu/notes/cs229-notes1.pdf

Deep Neural Network: summary

- input layer, multiple hidden layers, and output layer
- nonlinear processing via activation functions
- perform transformation and feature extraction
- gradient descent algorithm with back propagation
- each layer receives the output from previous layer
- results are comparable/superior to human experts

CNNs versus RNNs

- CNNs (Convolutional NNs):

Good for image processing

2000: CNNs processed 10-20% of all checks

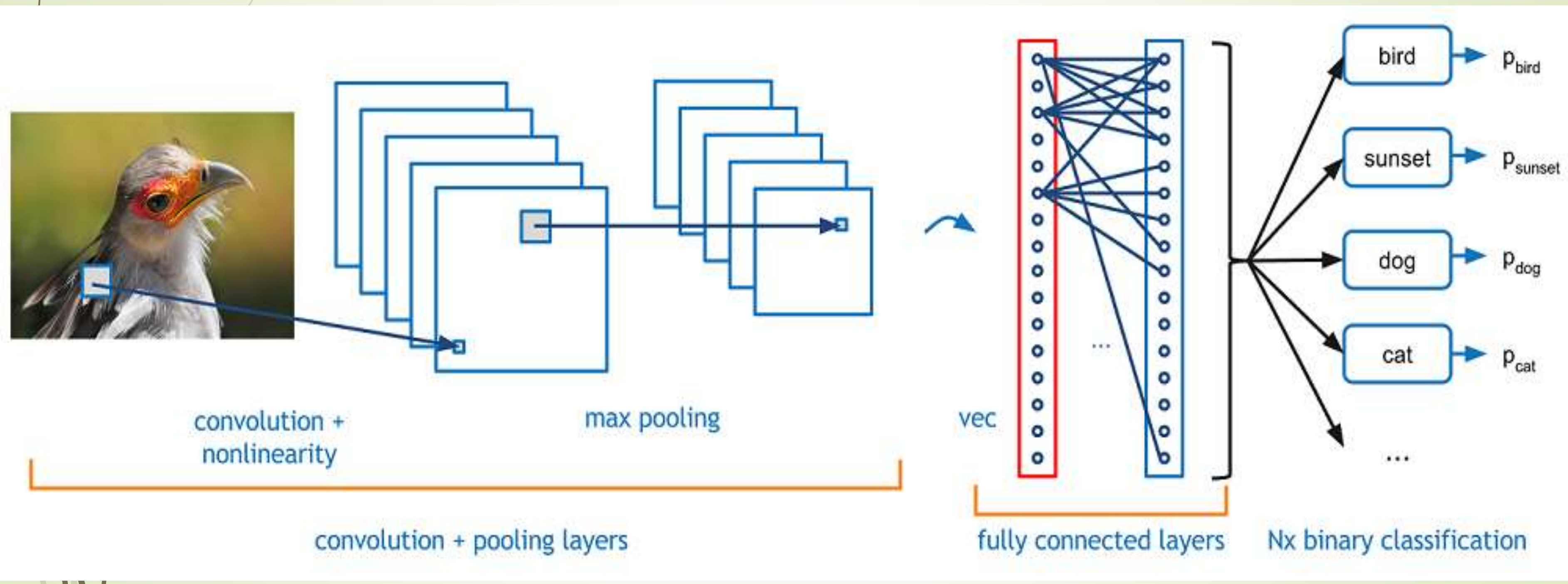
=> Approximately 60% of all NNs

- RNNs (Recurrent NNs):

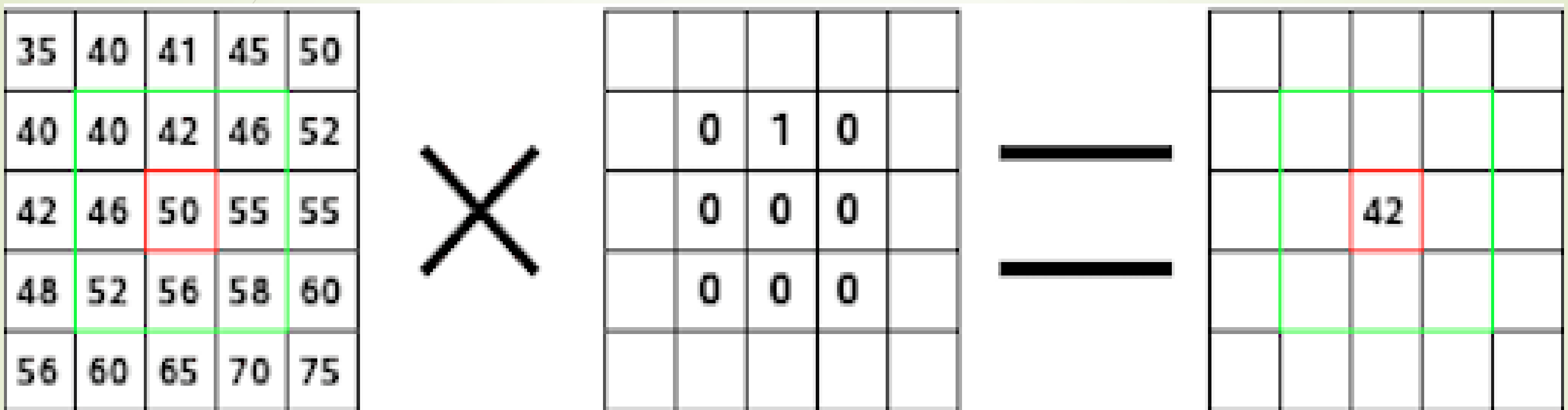
Good for NLP and audio

Used in hybrid networks

CNNs: Convolution, ReLU, and Max Pooling



CNNs: Convolution Calculations



- https://docs.gimp.org/en/plug-in-convmatrix.html

CNNs: Convolution Matrices (examples)

- Sharpen:

0	0	0	0	0
0	0	-1	0	0
0	-1	5	-1	0
0	0	-1	0	0
0	0	0	0	0

- Blur:

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

CNNs: Convolution Matrices (examples)

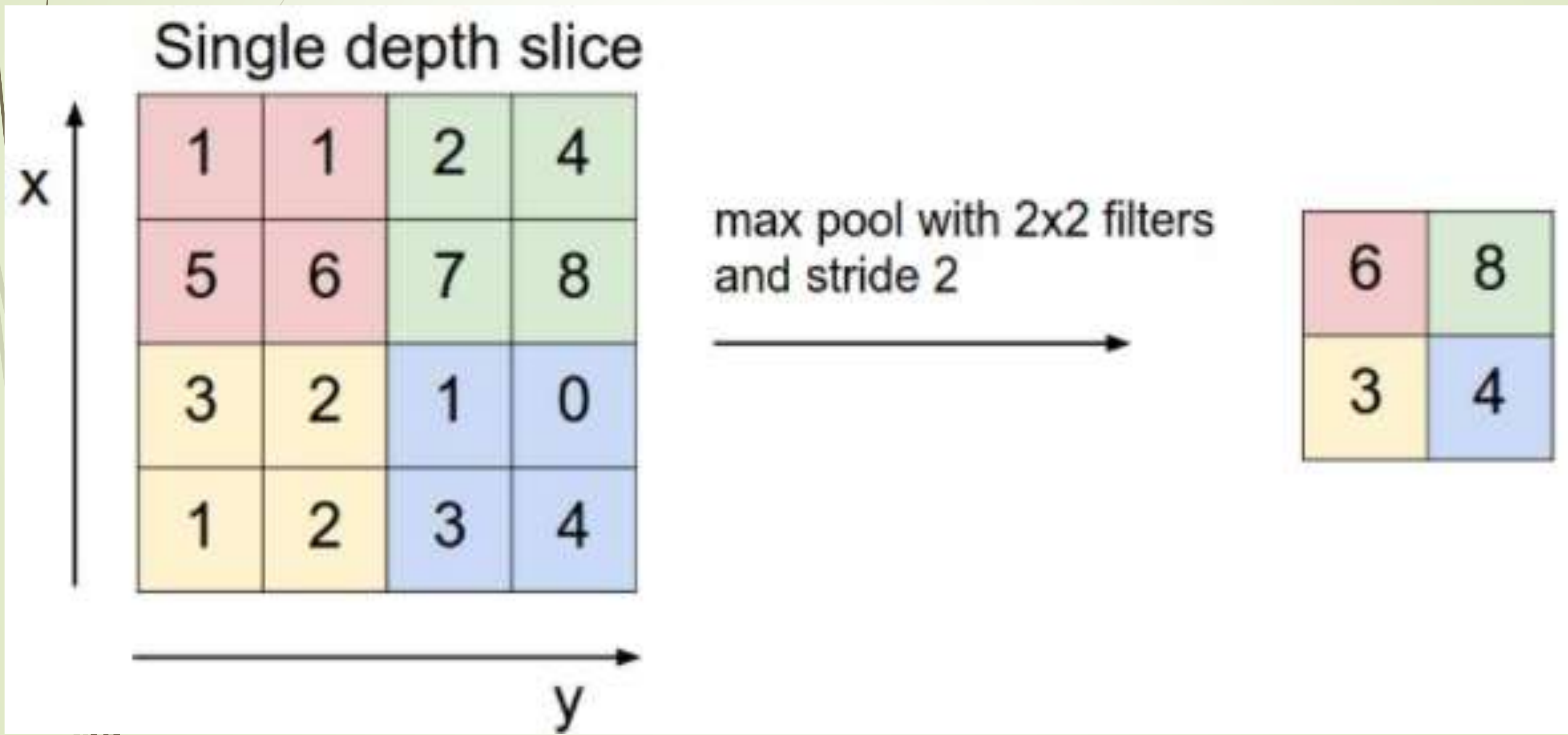
- Edge detect:

	0	1	0	
	1	-4	1	
	0	1	0	

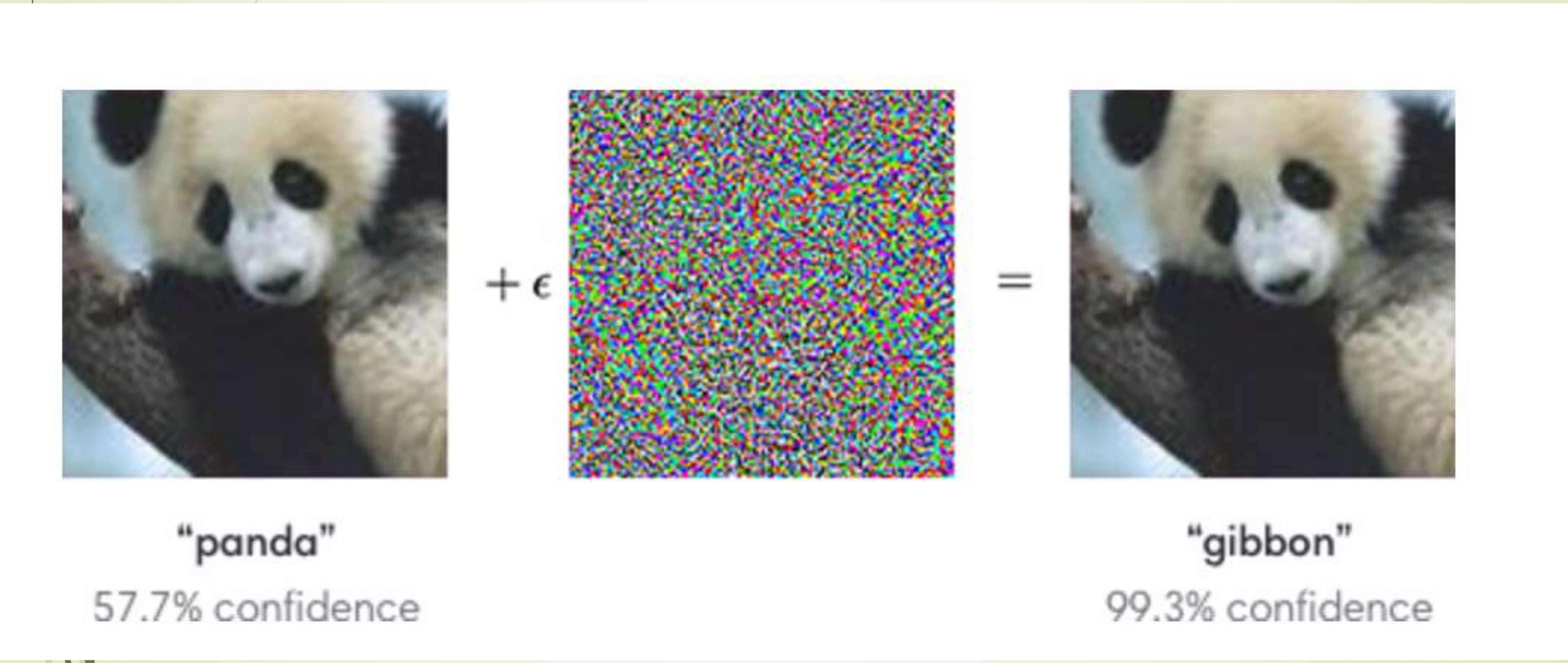
- Emboss:

	-2	-1	0	
	-1	1	1	
	0	1	2	

CNNs: Max Pooling Example



GANs: Generative Adversarial Networks



GANs: Generative Adversarial Networks

- Make imperceptible changes to images
- Can consistently defeat all NNs
- Can have extremely high error rate
- Some images create optical illusions
- https://www.quora.com/What-are-the-pros-and-cons-of-using-generative-adversarial-networks-a-type-of-neural-network

GANs: Generative Adversarial Networks

- Create your own GANs:

https://www.oreilly.com/learning/generative-adversarial-networks-for-beginners

https://github.com/jonbruner/generative-adversarial-networks

- GANs from MNIST:

http://edwardlib.org/tutorials/gan

- GANs and Capsule networks?

CNN in Python/Keras (fragment)

- from keras.models import Sequential
- from keras.layers.core import Dense, Dropout, Activation
- from keras.layers.convolutional import Conv2D, MaxPooling2D
- from keras.optimizers import Adadelta

```
input_shape = (3, 32, 32)
nb_classes = 10

model = Sequential()
model.add(Conv2D(32,(3, 3),padding='same',
                    input_shape=input_shape))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

What is TensorFlow?

- An open source framework for ML and DL
- A “computation” graph
- Created by Google (released 11/2015)
- Evolved from Google Brain
- Linux and Mac OS X support (VM for Windows)

- TF home page: https://www.tensorflow.org/

What is TensorFlow?

- Support for Python, Java, C++
- Desktop, server, mobile device (TensorFlow Lite)
- CPU/GPU/TPU support

- Visualization via TensorBoard
- Can be embedded in Python scripts
- Installation: pip install tensorflow

TensorFlow cluster:

https://www.tensorflow.org/deploy/distributed

TensorFlow Use Cases (Generic)

- Image recognition
- Computer vision
- Voice/sound recognition
- Time series analysis
- Language detection
- Language translation
- Text-based processing
- Handwriting Recognition

Aspects of TensorFlow

- Graph: graph of operations (DAG)
- Sessions: contains Graph(s)
- lazy execution (default)
- operations in parallel (default)
- Nodes: operators/variables/constants
- Edges: tensors

- => graphs are split into subgraphs and executed in parallel (or multiple CPUs)

TensorFlow Graph Execution

- Execute statements in a tf.Session() object
- Invoke the “run” method of that object
- “eager” execution is available (>= v1.4)
- included in the mainline (v1.7)
- Installation: pip install tensorflow

What is a Tensor?

- TF tensors are n-dimensional arrays
- TF tensors are very similar to numpy ndarrays

- scalar number: a zeroth-order tensor
- vector: a first-order tensor
- matrix: a second-order tensor
- 3-dimensional array: a 3rd order tensor

- https://dzone.com/articles/tensorflow-simplified-examples

TensorFlow “primitive types”

- tf.constant:

+ initialized immediately

+ immutable

- tf.placeholder (a function):

+ initial value is not required

+ can have variable shape

+ assigned value via feed_dict at run time

+ receive data from “external” sources

TensorFlow “primitive types”

- tf.Variable (a class):

+ initial value is required

+ updated during training

+ maintain state across calls to “run()”

+ in-memory buffer (saved/restored from disk)

+ can be shared in a distributed environment

+ they hold learned parameters of a model

TensorFlow: constants (immutable)

- import tensorflow as tf

- aconst = tf.constant(3.0)
- print(aconst)

\# output: Tensor("Const:0", shape=(), dtype=float32)

- sess = tf.Session()
- print(sess.run(aconst))

\# output: 3.0

- sess.close()

- \# => there's a better way☺

TensorFlow: constants

- import tensorflow as tf

- aconst = tf.constant(3.0)
- print(aconst)

- Automatically close “sess”
- with tf.Session() as sess:
-   print(sess.run(aconst))

TensorFlow Arithmetic

```
import tensorflow as tf

a = tf.add(4, 2)
b = tf.subtract(8, 6)
c = tf.multiply(a, 3)
d = tf.div(a, 6)

with tf.Session() as sess:
  print(sess.run(a)) # 6
  print(sess.run(b)) # 2
  print(sess.run(c)) # 18
  print(sess.run(d)) # 1
```

TF placeholders and feed_dict

```
import tensorflow as tf

a = tf.placeholder("float")
b = tf.placeholder("float")
c = tf.multiply(a,b)

# initialize a and b:
feed_dict = {a:2, b:3}

# multiply a and b:
with tf.Session() as sess:
  print(sess.run(c, feed_dict))
```

TensorFlow: Simple Equation

- `import tensorflow as tf`

- `# W and x are 1d arrays`
- `W = tf.constant([10,20],       name='W')`
- `X = tf.placeholder(tf.int32, name='x')`
- `b = tf.placeholder(tf.int32, name='b')`

- `Wx = tf.multiply(W, x, name='Wx')`
- `y  = tf.add(Wx, b, name='y') OR`
- `y2 = tf.add(tf.multiply(W,x),b)`

TensorFlow fetch/feed_dict

```
with tf.Session() as sess:
  print("Result 1: Wx = ",
        sess.run(Wx, feed_dict={x:[5,10]}))

  print("Result 2: y  = ",
        sess.run(y,feed_dict={x:[5,10],b:[15,25]}))
```

- Result 1: Wx = [50 200]
- Result 2: y = [65 225]

Saving Graphs for TensorBoard

```
import tensorflow as tf
x = tf.constant(5,name="x")
y = tf.constant(8,name="y")
z = tf.Variable(2*x+3*y, name="z")
init = tf.global_variables_initializer()

with tf.Session() as session:
  writer = tf.summary.FileWriter("./tf_logs",session.graph)
  session.run(init)
  print 'z = ',session.run(z) # =>  z = 34

# launch: tensorboard -logdir=./tf_logs
```

TensorFlow Eager Execution

- An imperative interface to TF
- Fast debugging & immediate run-time errors
- Eager execution is “mainline” in v1.7 of TF
- => requires Python 3.x (not Python 2.x)

TensorFlow Eager Execution

- integration with Python tools
- Supports dynamic models + Python control flow
- support for custom and higher-order gradients
- Supports most TensorFlow operations
- => Default mode in TensorFlow 2.0 (2019)

- https://research.googleblog.com/2017/10/eager-execution-imperative-define-by.html

TensorFlow Eager Execution

- import tensorflow as tf
- import tensorflow.contrib.eager as tfe

- tfe.enable_eager_execution()

- x = [[2.]]
- m = tf.matmul(x, x)
- print(m)

tf.Tensor([[4.]], shape=(1, 1), dtype=float32)

What is tensorflow.js?

- an ecosystem of JS tools for machine learning
- TensorFlow.js also includes a Layers API
- a library for building machine learning models
- tools to port TF SavedModels & Keras HDF5 models
- => https://js.tensorflow.org/

What is tensorflow.js?

- tensorflow.js evolved from deeplearn.js
- deeplearn.js is now called TensorFlow.js Core

- TensorFlow.js Core: a flexible low-level API

- TensorFlow.js Layers:

a high-level API similar to Keras

- TensorFlow.js Converter:

tools to import a TF SavedModel to TensorFlow.js

async keyword

- keyword placed before JS functions
- For functions that return a Promise
- Trivial example:

```
async function f() {
    return 1;
}
```

await keyword

- Works only inside async JS functions

- Trivial example:
- let value = await mypromise;

async/await example

```
async function f() {
  let promise = new Promise((resolve, reject) => {
      setTimeout(() => resolve("done!"), 1000)
  });

  // wait till the promise resolves
  let result = await promise
  alert(result)
}

f()
```

Tensorflow.js Samples

1) tfjs-example.html (linear regression)

2) js.tensorflow.org (home page)

3) https://github.com/tensorflow/tfjs-examples-master

a)cd mnist-core

b) yarn

c) yarn watch

Deep Learning and Art/"Stuff"

- "Convolutional Blending" images:

=> 19-layer Convolutional Neural Network

www.deepart.io

- https://www.fastcodesign.com/90124942/this-google-engineer-taught-an-algorithm-to-make-train-footage-and-its-hypnotic