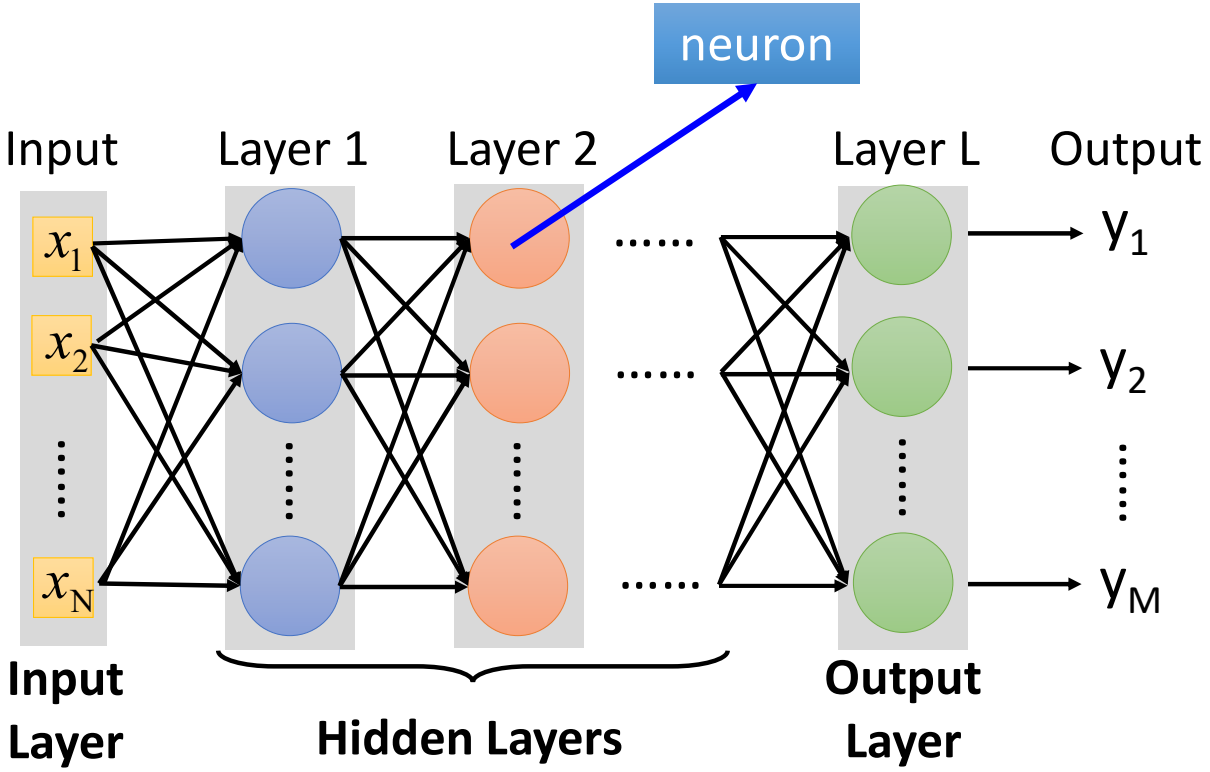
Introduction to Deep Learning (cont'd)

(1) How is ANN trained?
(2) ANN with Keras: some examples in Python

Neural Network (Feedforward)



Deep means many hidden layers

Every node in one layer is connected to every node in the next layer

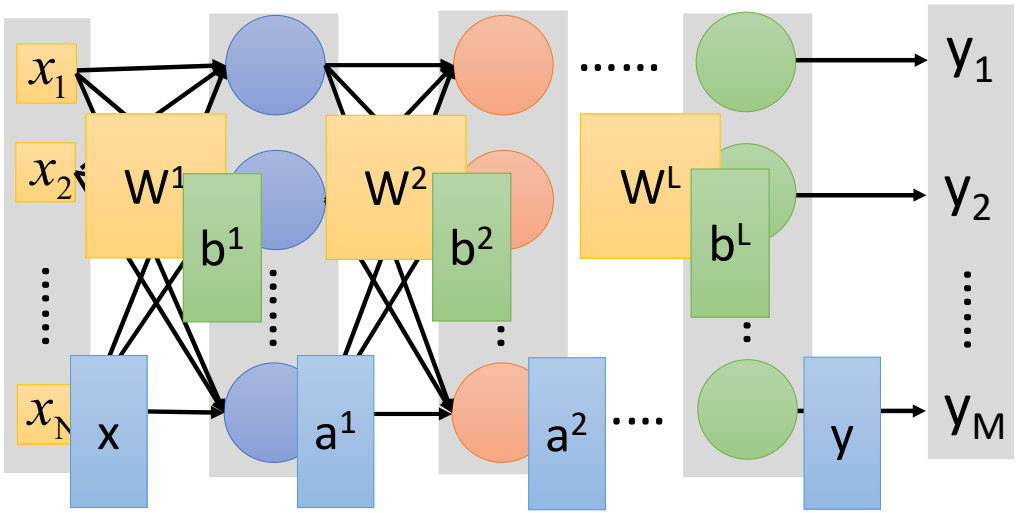
Neural Network: in matrix notation (optional slide)

- The intermediate quantity in the activations of the hidden layers:

$$z_1^{(2)} = w_{10}^{(1)} x_0 + w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + w_{13}^{(1)} x_3$$
$$z_2^{(2)} = w_{20}^{(1)} x_0 + w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{23}^{(1)} x_3$$
$$z_3^{(2)} = w_{30}^{(1)} x_0 + w_{31}^{(1)} x_1 + w_{32}^{(1)} x_2 + w_{33}^{(1)} x_3$$

$$\underbrace{\begin{pmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{pmatrix}}_{z^{(2)}} = \underbrace{\begin{pmatrix} w_{10}^{(1)} & w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{20}^{(1)} & w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \\ w_{30}^{(1)} & w_{31}^{(1)} & w_{32}^{(1)} & w_{33}^{(1)} \end{pmatrix}}_{\mathrm{W}^{(1)}} \underbrace{\begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}}_{x}$$

Neural Network: matrix operation (optional)

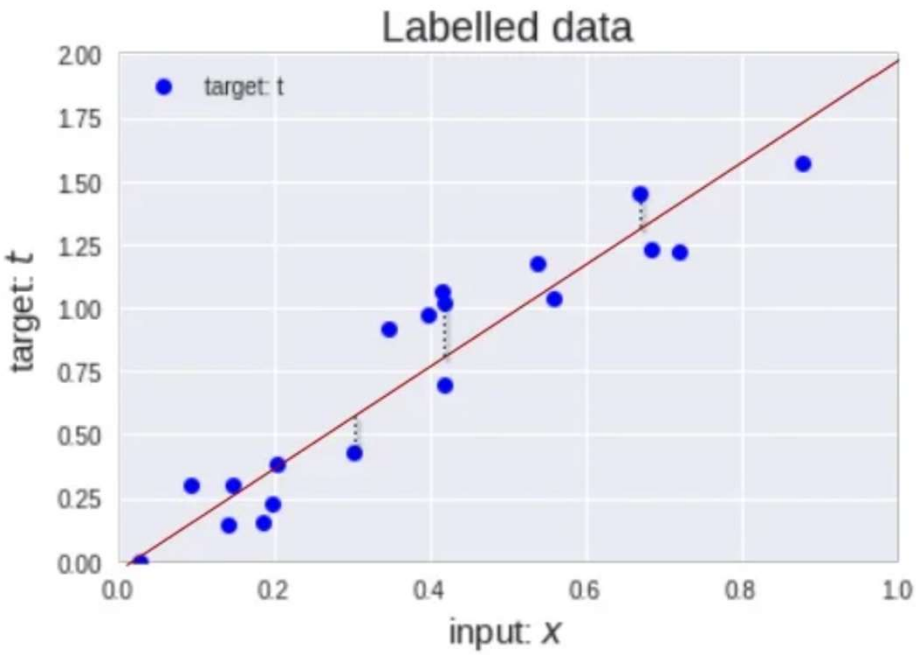


$$\mathbf{y} = f(\mathbf{x})$$

Using parallel computing techniques to speed up matrix operation

$$= \sigma(\mathbf{W}^L \cdots \sigma(\mathbf{W}^2 \sigma(\mathbf{W}^1 \mathbf{x} + \mathbf{b}^1) + \mathbf{b}^2) \cdots + \mathbf{b}^L)$$

Cost function



For this linear regression example, to determine the best p (slope of the line) for

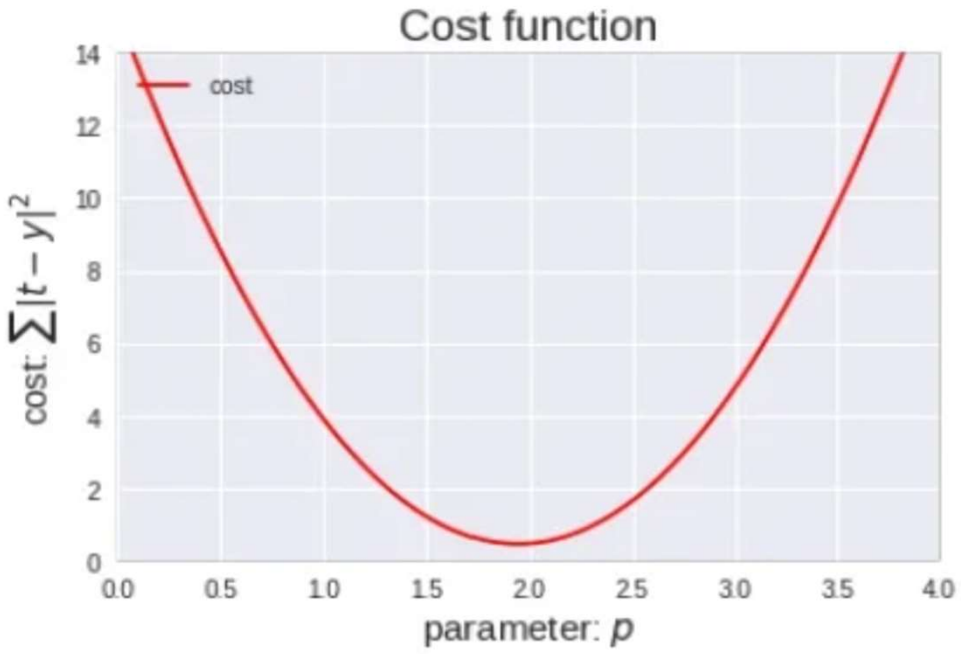
$$y = x \cdot p$$

we can calculate the cost function, such as Mean Square Error, Mean absolute error, Mean bias error, SVM Loss, etc.

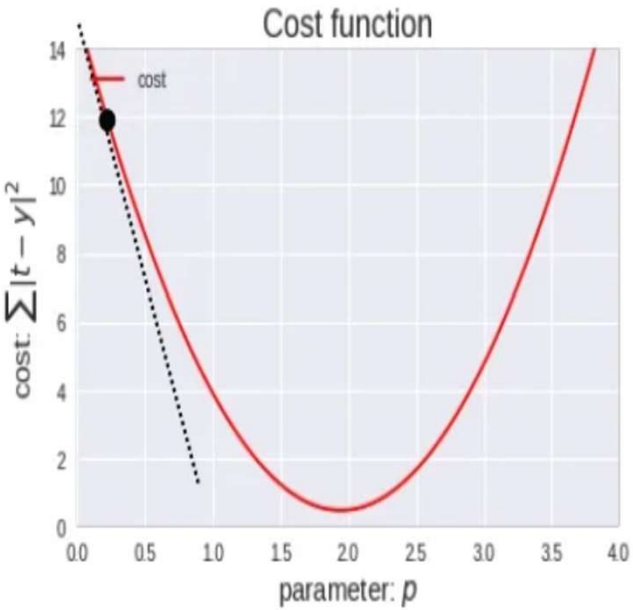
For this example, we'll use sum of squared absolute differences

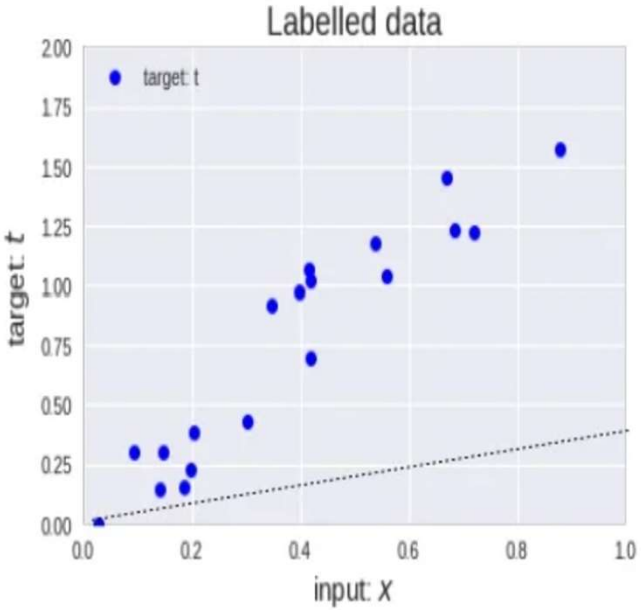
$$cost = \sum |t - y|^2$$

Visualize the cost function

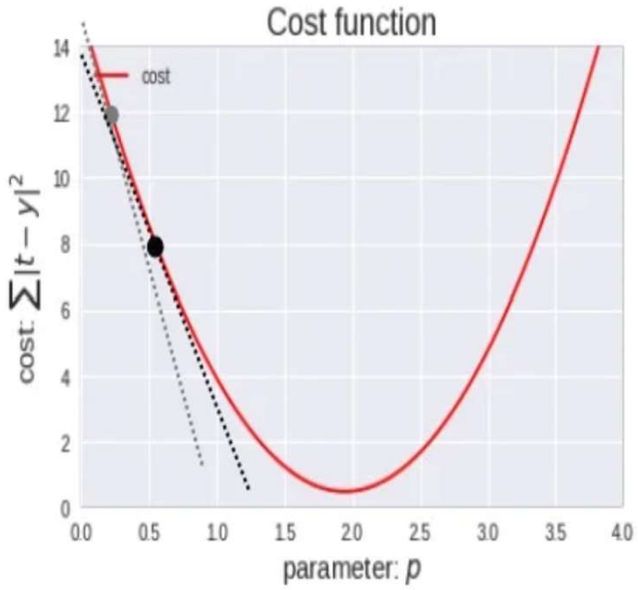


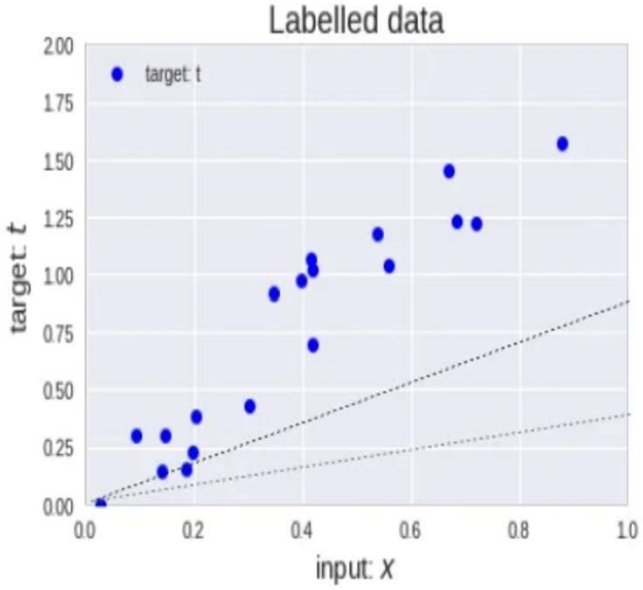
Calculate its derivative





Gradient descent





Simple gradient descent

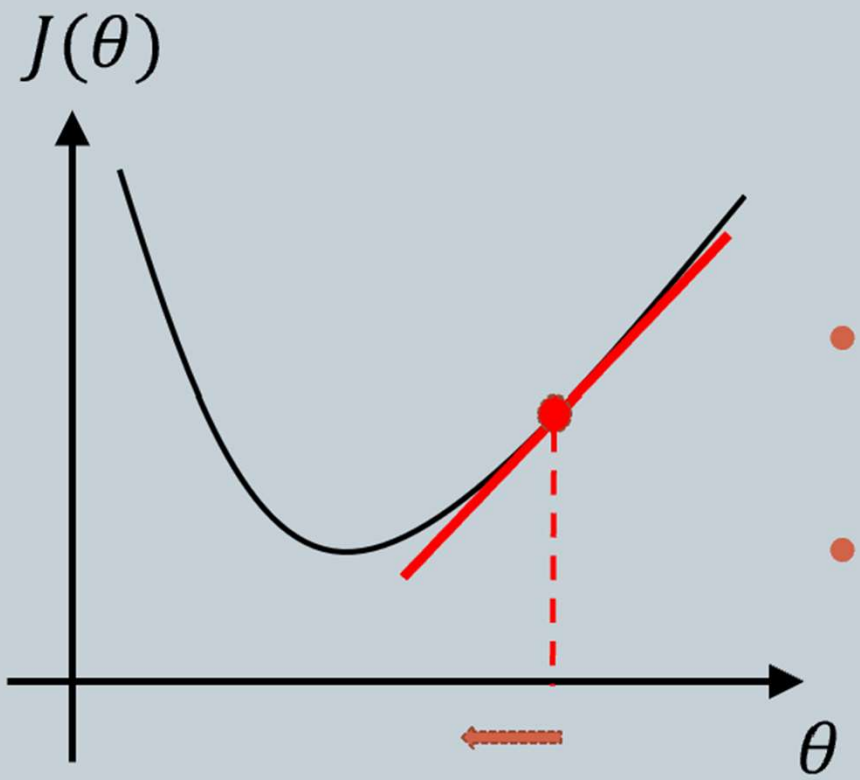
- Find $\theta \in R$ such that

$$\text{Min } J(\theta)$$

- In iteration t, perform the following update

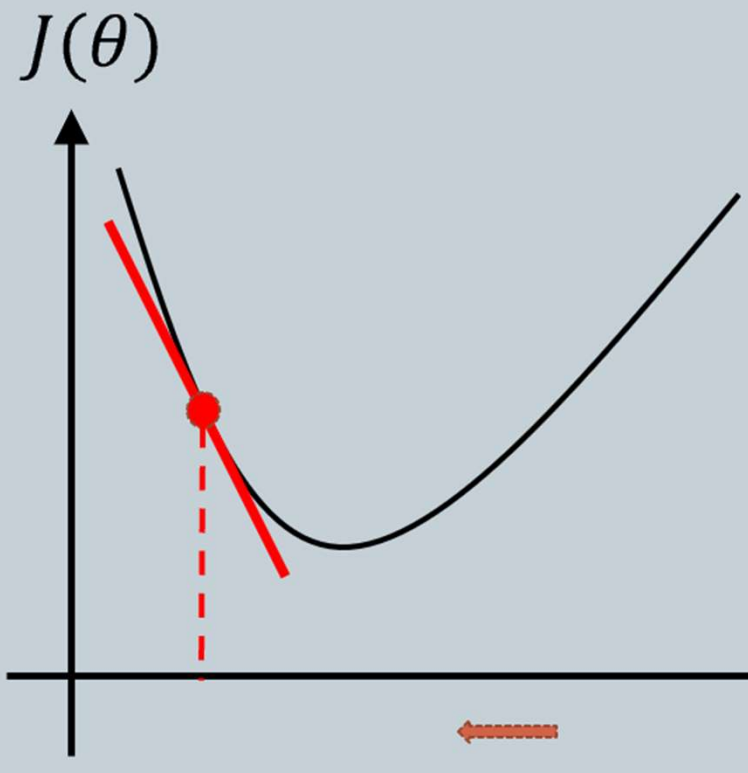
$$\theta^{(t+1)} = \theta^{(t)} - \eta \frac{d}{d\theta} J(\theta^{(t)})$$

where $\eta > 0$ is called step-size and $\frac{d}{d\theta} J\left(\theta^{(t)}\right)$ is the derivative at $\theta^{(t)}$.



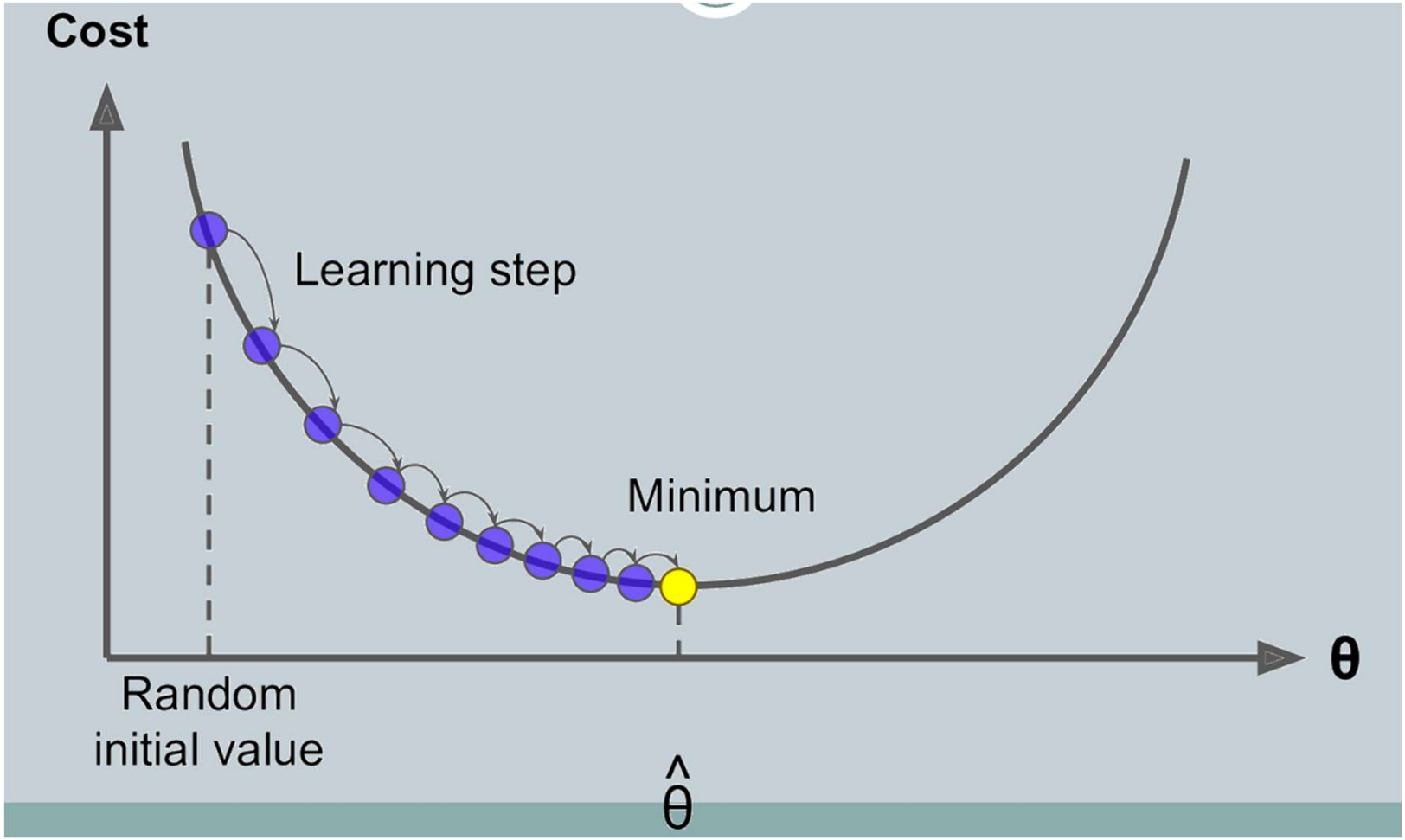
$$\theta^{(t+1)} = \theta^{(t)} - \eta \frac{d}{d\theta} J(\theta^{(t)})$$

- The derivative is positive
- We will move to the left

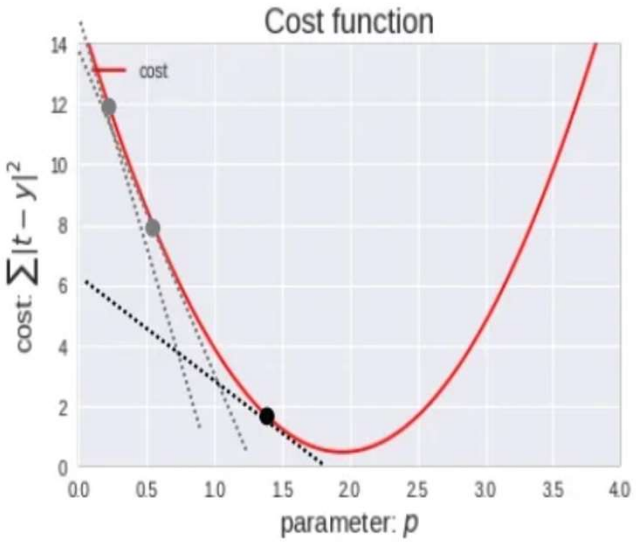


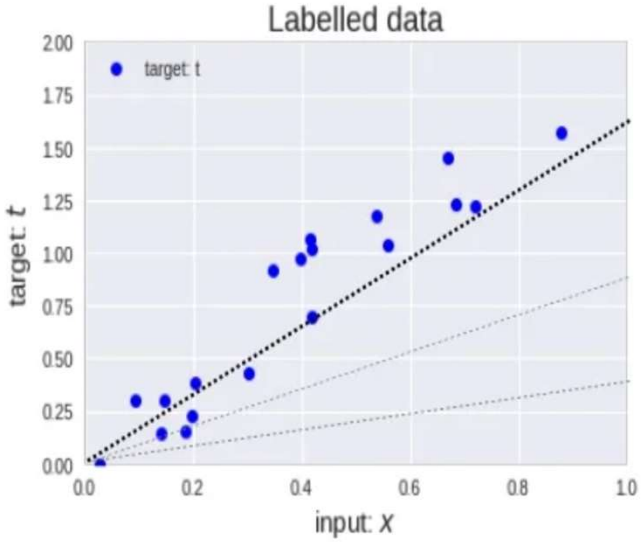
$$\theta^{(t+1)} = \theta^{(t)} - \eta \frac{d}{d\theta} J(\theta^{(t)})$$

- The derivative is negative
- We will move to the right


Cost
Learning step
Minimum
θ
Random
initial value
θ̂

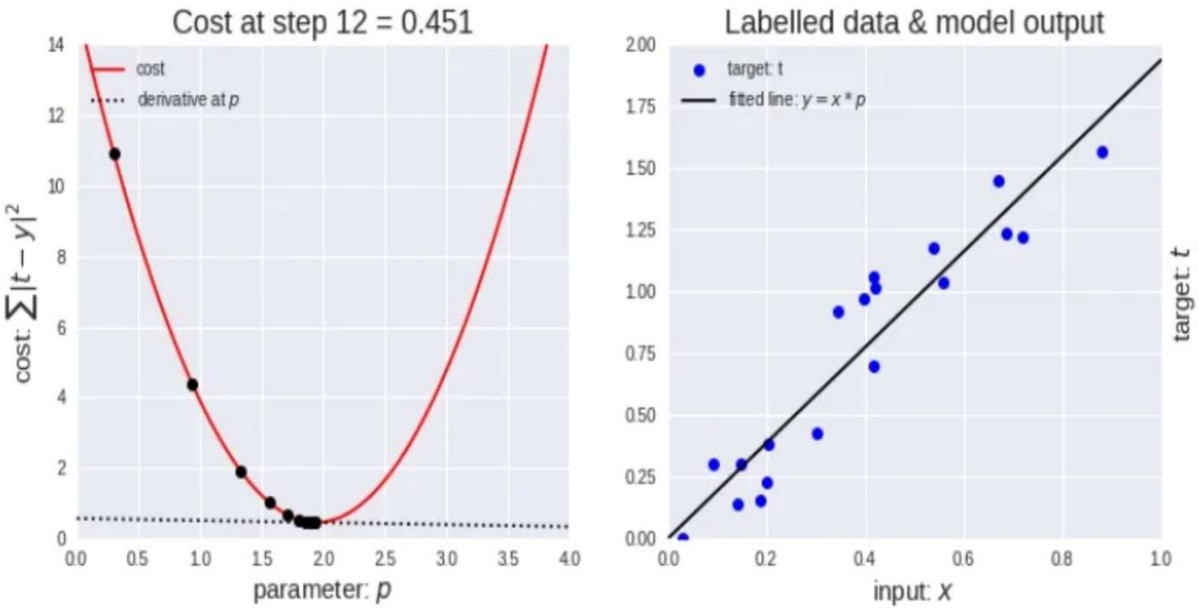
Gradient descent



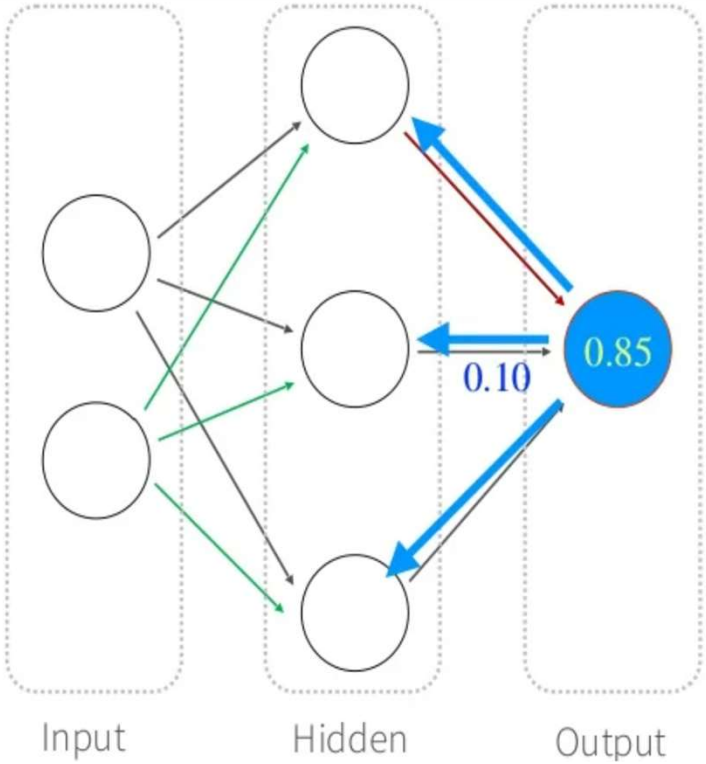


The goal is to find the lowest point of the cost function (i.e. least amount of cost or the minimum or *minima*).
Gradient descent iteratively (i.e. step by step) descends the cost function curve to find the minima.

Gradient descent optimization



Backpropagation



- Backpropagation: calculate the gradient of the cost function in a neural network
- Used by gradient descent optimization algorithm to adjust weight of neurons
- Also known as backward propagation of errors *as the error is calculated and distributed back through the network of layers*

Back-propagation training

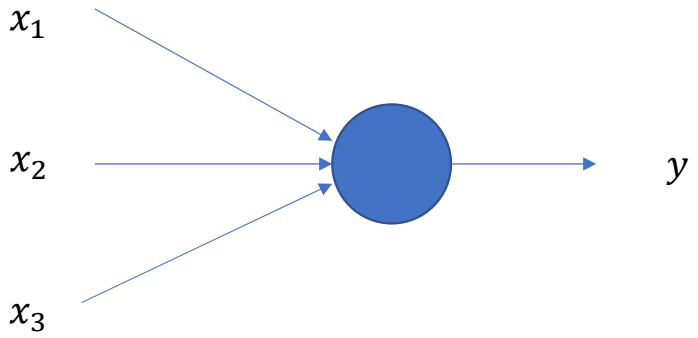
- first adjust last set of weights
- propagate error back to each previous layer
- adjust their weights

Variations of Gradient descent

- Batch Gradient Descent
  - Involves calculations over the full data set
  - Uses the full data set at every step
  - Terribly slow on large training sets
- Stochastic Gradient Descent
  - Picks a random observation in the training data and computes gradients at every step
  - Much faster
- Mini-batch Gradient Descent
  - Calculates gradients based on a subset or mini-batch of the training data

Activation function: Sigmoid

- A sigmoid neuron closely approximates a perceptron



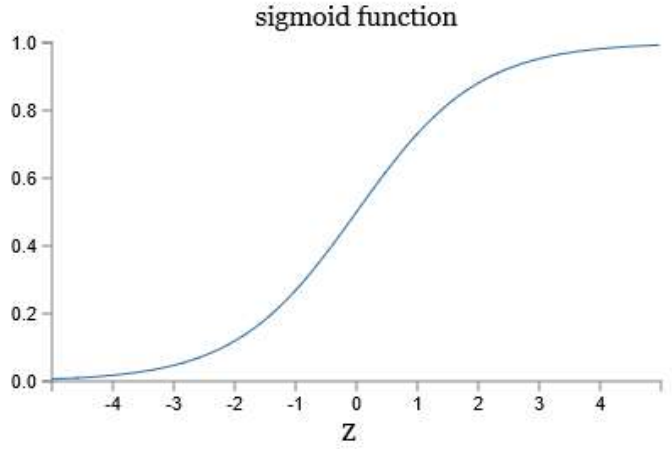
Sigmoid function:

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

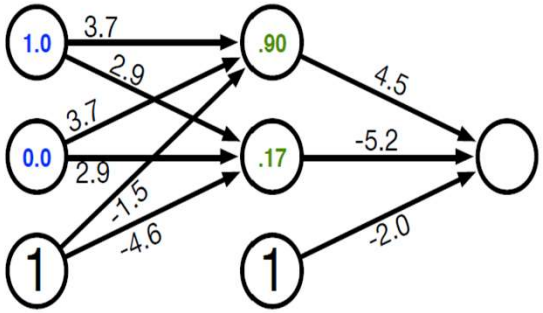
- If $\theta^T x$ is large, then output ≈ 1, if $\theta^T x$ is very negative, then output ≈ 0. Only when $\theta^T x$ is in between, then sigmoid neuron differs from perceptrons

Sigmoid Neurons

- Sigmoid neurons: small changes in their weights cause only a small change in their output.



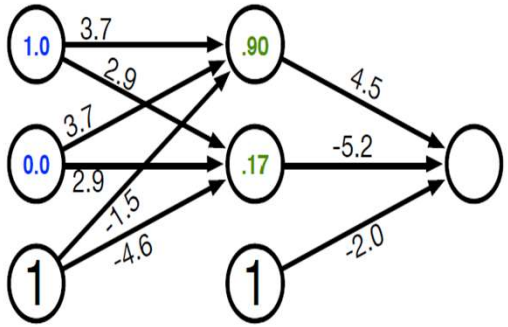
- Crucial fact which will allow a network of sigmoid neurons to learn.



- Try out two input values
- Hidden unit computation

$$\text{sigmoid}(1.0 \times 3.7 + 0.0 \times 3.7 + 1 \times -1.5) = \text{sigmoid}(2.2) = \frac{1}{1+e^{-2.2}} = 0.90$$

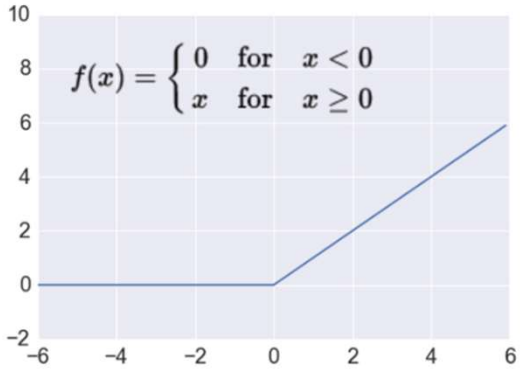
$$\text{sigmoid}(1.0 \times 2.9 + 0.0 \times 2.9 + 1 \times -4.5) = \text{sigmoid}(-1.6) = \frac{1}{1+e^{1.6}} = 0.17$$



- Output unit computation

$$\text{sigmoid}(.90 \times 4.5 + .17 \times -5.2 + 1 \times -2.0) = \text{sigmoid}(1.17) = \frac{1}{1+e^{-1.17}} = 0.76$$

Activation: ReLU (rectifier linear unit)



http://adilmoujahid.com/images/activation.png

Takes a real-valued number and thresholds it at zero $\mathrm{f}(x) = \max(0, x)$

$$R^n \rightarrow R^n_+$$

Most Deep Networks use ReLU nowadays

☺ Trains much **faster**
- accelerates the convergence of SGD
- due to linear, non-saturating form

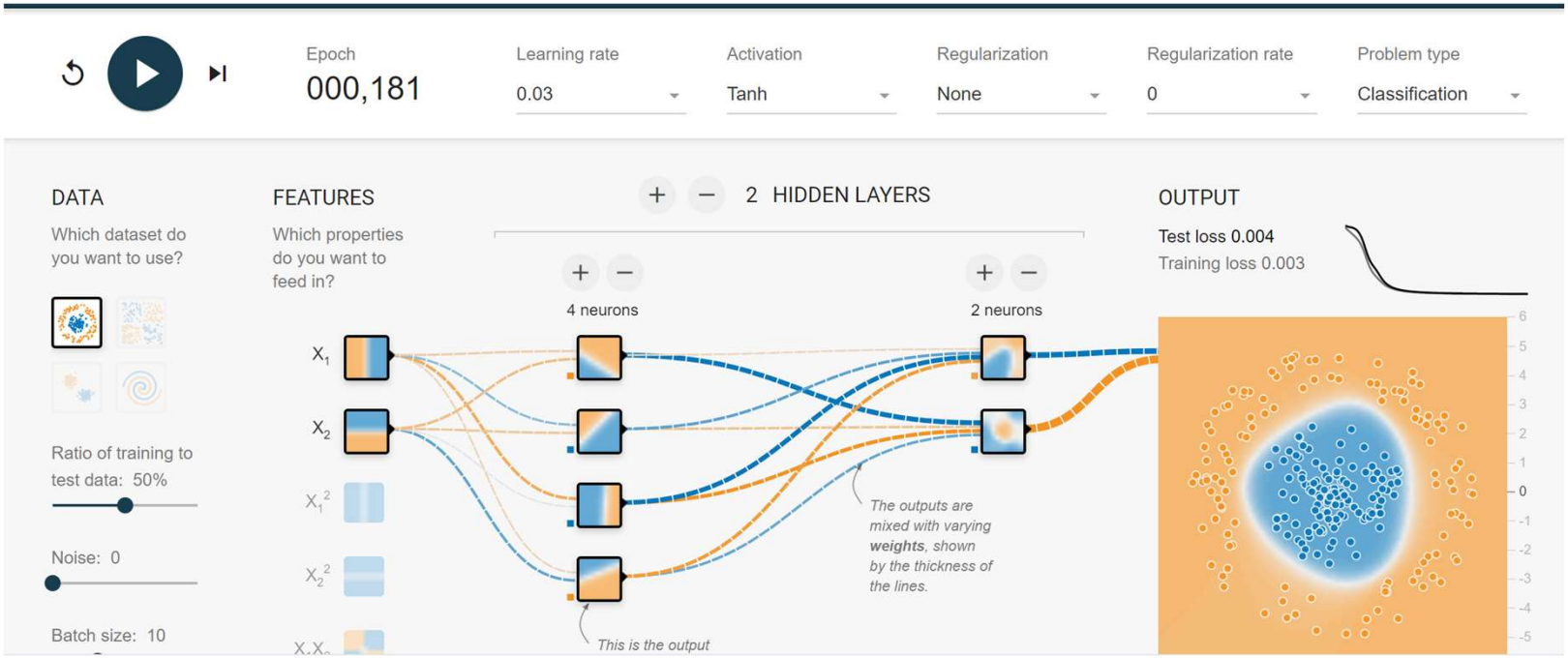
☺ Less expensive operations
- compared to sigmoid/tanh (exponentials etc.)
- implemented by simply thresholding a matrix at zero

☺ More **expressive**

☺ Prevents the **gradient vanishing problem**

Let’s create our first ANN:

https://playground.tensorflow.org/

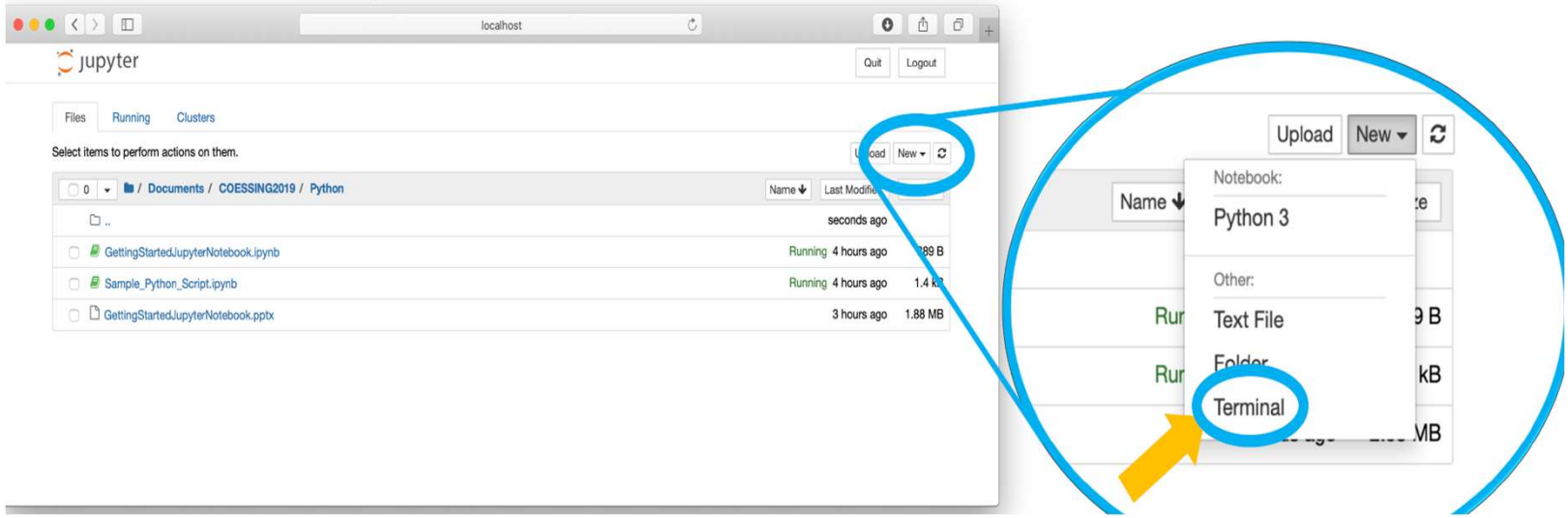


How to install other libraries to Anaconda

- There are some libraries that my be useful that do not come with Anaconda. But, we can install them directly to our conda library!
- There are two methods to install these libraries:

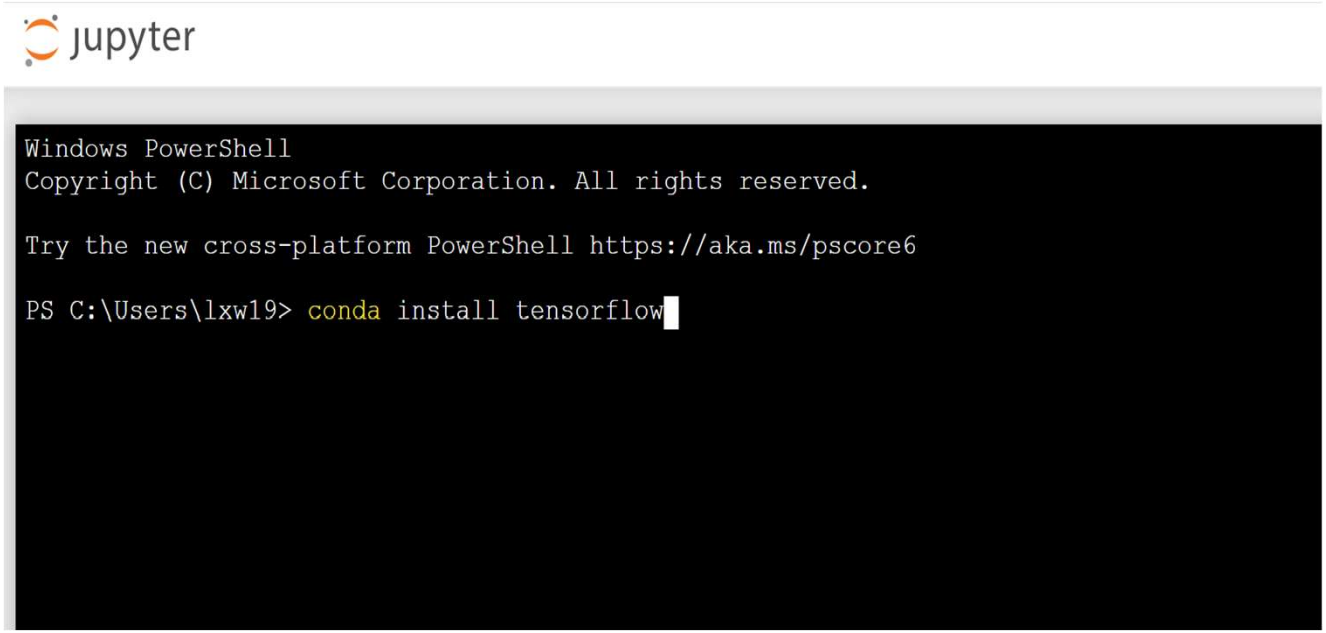
1. Open Anaconda Prompt

2. Open a “Terminal” instance from Jupyter Notebook (see below image for instructions!)



Install Tensorflow

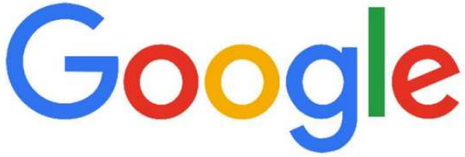
With either method, you will see something like this:



Keras: Deep Learning Library

- High-level neural networks API, written in Python
- Models described in Python code. No separate models config files
- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility)
- Runs seamlessly on CPUs and GPUs

Who makes Keras? Contributors and backers



What's special about Keras?

- A focus on user experience.
- Large adoption in the industry and research community.
- Multi-backend, multi-platform.
- Easy productization of models
- This ease of use does not come at the cost of reduced flexibility: because Keras integrates with lower-level deep learning languages (in particular TensorFlow), it enables you to implement anything you could have built in the base language.

Keras vs Tensorflow

Parameters	Keras	Tensorflow
Type	High-Level API Wrapper	Low-Level API
Complexity	Easy to use if you Python language	You need to learn the syntax of using some of Tensorflow function
Purpose	Rapid deployment for making model with standard layers	Allows you to make an arbitrary computational graph or model layers
Tools	Uses other API debug tool such as TFDBG	You can use Tensorboard visualization tools
Community	Large active communities	Large active communities and widely shared resources

Important note

- You could not have Keras without TensorFlow, and if you installed Keras on your system, you were also installing TensorFlow.

Developing the model

- Number of Layers
- Types of these Layers
- Number of units (neurons) in each Layer
- Activation Functions of each Layer
- Input and output size

- Here, we've used Keras' Sequential() to instantiate a model. It takes a group of sequential layers and stacks them together into a single model. Into the Sequential() constructor, we pass a list that contains the layers we want to use in our model.
- The number of neurons in the input layer is the same as the number of features in our data.

- We'll be using *Dense* and *Dropout* layers. Dense layers are the most common and popular type of layer - it's just a regular neural network layer where each of its neurons is connected to the neurons of the previous and next layer.
- Each *dense layer* has an activation function that determines the output of its neurons based on the inputs and the weights of the synapses.
- *Dropout layers* are just regularization layers that randomly drop some of the input units to 0. This helps in reducing the chance of overfitting the neural network.

Simple deep MLP with Keras

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential()
model.add(Dense(64, activation='relu', input_dim=50)) #input shape of 50
model.add(Dense(28, activation='relu')) #input shape of 50
model.add(Dense(10, activation='softmax'))
```

Compiling the Model

- The Loss Function -The lower the error, the closer the model is to the goal. Different problems require different loss functions to keep track of progress. Here's a list of supported loss functions.
- The Optimizer - The optimizing algorithm that helps us achieve better results for the loss function.
- Metrics - Metrics used to evaluate the model.

The three most common loss functions

- ‘binary_crossentropy‘ for binary classification.
- ‘sparse_categorical_crossentropy‘ for multi-class classification.
- ‘mse‘ (mean squared error) for regression.

A full list of loss functions:

https://www.tensorflow.org/api_docs/python/tf/keras/losses

- Assume you have a dataset with 200 samples (rows of data) and you choose a batch size of 5 and 1,000 epochs.

- This means that the dataset will be divided into 40 batches, each with five samples. The model weights will be updated after each batch of five samples.

- This also means that one epoch will involve 40 batches or 40 updates to the model.

- With 1,000 epochs, the model will be exposed to or pass through the whole dataset 1,000 times. That is a total of 40,000 batches during the entire training process.

- Stochastic gradient descent is an iterative learning algorithm that uses a training dataset to update a model.
- The **batch size** is a hyperparameter of gradient descent that controls the number of training samples to work through before the model's internal parameters are updated.
- The number of **epochs** is a hyperparameter of gradient descent that controls the number of complete passes through the training dataset.

What Is a Batch?

- Think of a batch as a for-loop iterating over one or more samples and making predictions. At the end of the batch, the predictions are compared to the expected output variables and an error is calculated. From this error, the update algorithm is used to improve the model, e.g. move down along the error gradient.

- A training dataset can be divided into one or more batches.

- When all training samples are used to create one batch, the learning algorithm is called batch gradient descent. When the batch is the size of one sample, the learning algorithm is called stochastic gradient descent. When the batch size is more than one sample and less than the size of the training dataset, the learning algorithm is called mini-batch gradient descent.

What Is an Epoch?

- One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters. An epoch is comprised of one or more batches. For example, as above, an epoch that has one batch is called the batch gradient descent learning algorithm.

- You can think of a for-loop over the number of epochs where each loop proceeds over the training dataset. Within this for-loop is another nested for-loop that iterates over each batch of samples, where one batch has the specified “batch size” number of samples.

What Is an Epoch? (cont'd)

- The number of epochs is traditionally large, often hundreds or thousands, allowing the learning algorithm to run until the error from the model has been sufficiently minimized. You may see examples of the number of epochs in the literature and in tutorials set to 10, 100, 500, 1000, and larger.

What Is the Difference Between Batch and Epoch?

- The batch size is a number of samples processed before the model is updated.
- The number of epochs is the number of complete passes through the training dataset.
- The size of a batch must be more than or equal to one and less than or equal to the number of samples in the training dataset.
- The number of epochs can be set to an integer value between one and infinity. You can run the algorithm for as long as you like and even stop it using other criteria besides a fixed number of epochs, such as a change (or lack of change) in model error over time.