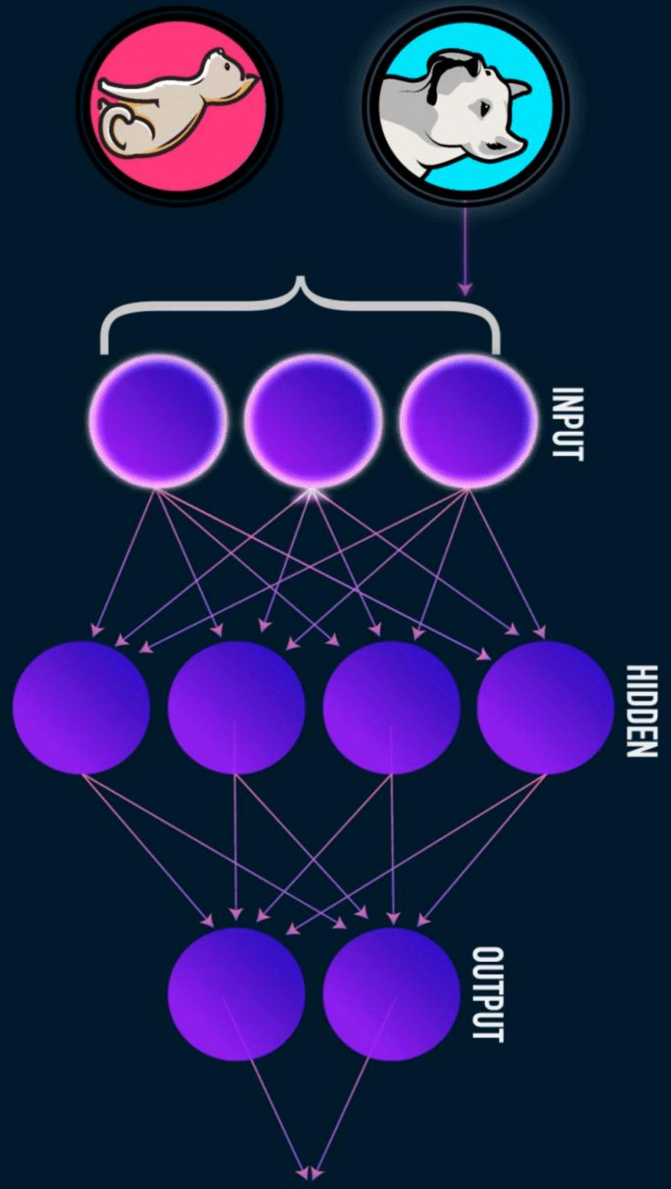
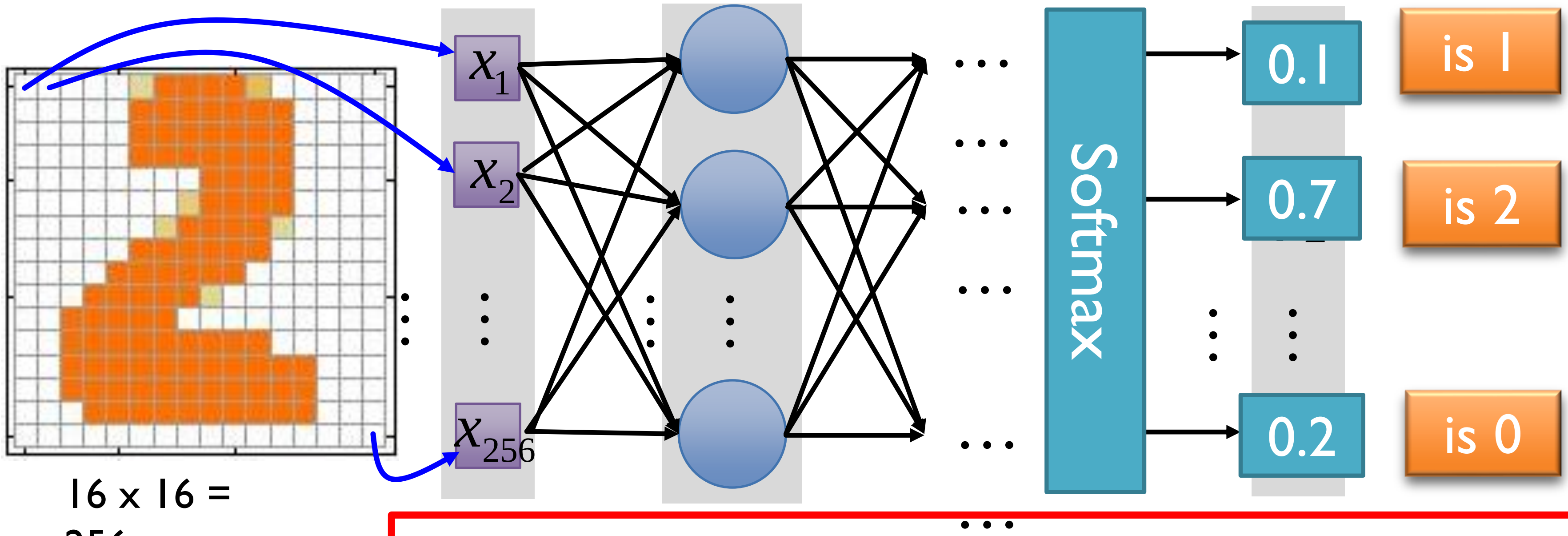


Backpropagation

Deep Network – How to set Network Parameters?

$$\theta = \{W^1, b^1, W^2, b^2, \cdots W^L, b^L\}$$



16 x 16 = 256

Ink → 1
No ink → 0

Set the network parameters θ such that ……

Input: … m value

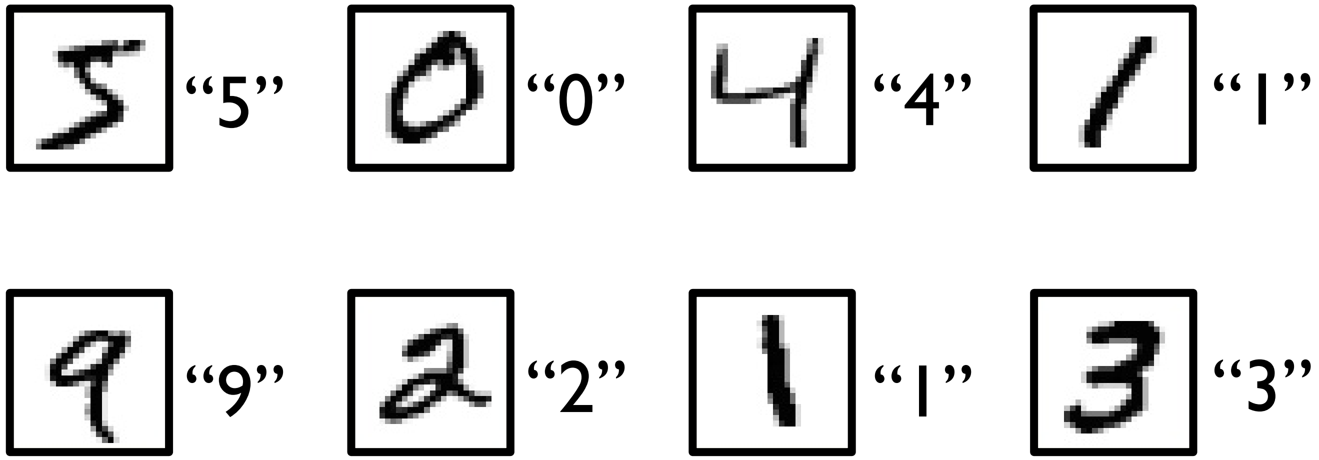
Input: → y_2 has the maximum value

How to let the neural network achieve this

Deep Network – How to set Network Parameters?

Training Data

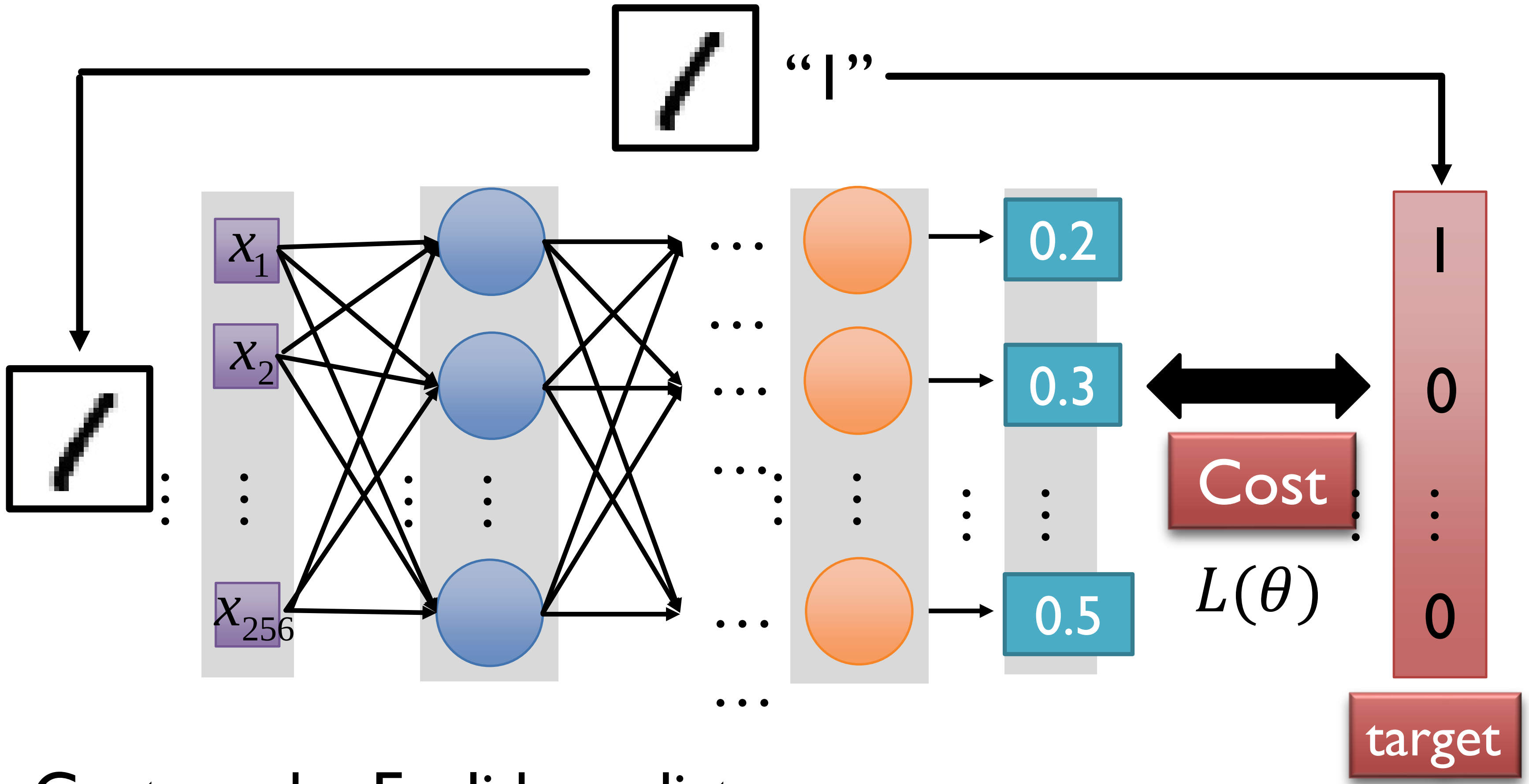
Preparing Training Data: Images And Their Labels



Using the training data to find the network parameters.

Deep Network – How to set Network Parameters?

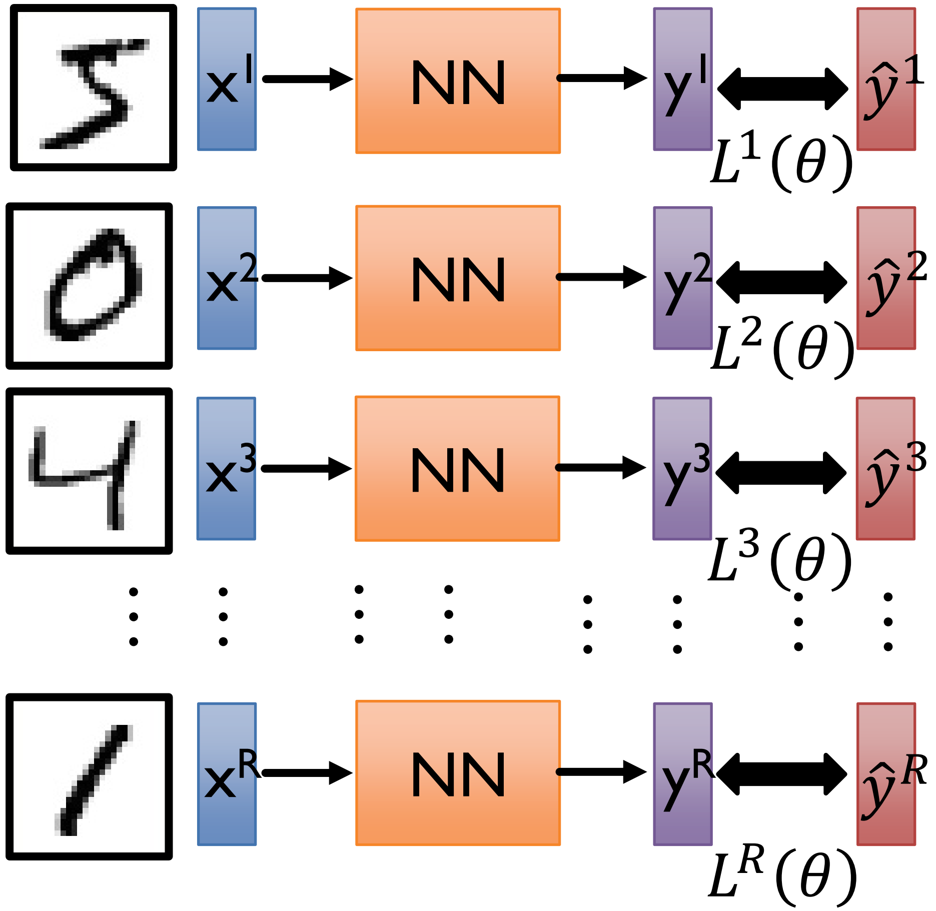
Cost - Given A Set Of Network Parameters θ, Each Example Has A Cost Value.



Cost can be Euclidean distance or cross entropy of the network output and target

Deep Network – How to set Network Parameters?

Total Cost – For all training data



Total Cost:

$$C(\theta) = \sum_{r=1}^{R} L^r(\theta)$$

How bad the network parameters θ is on this task

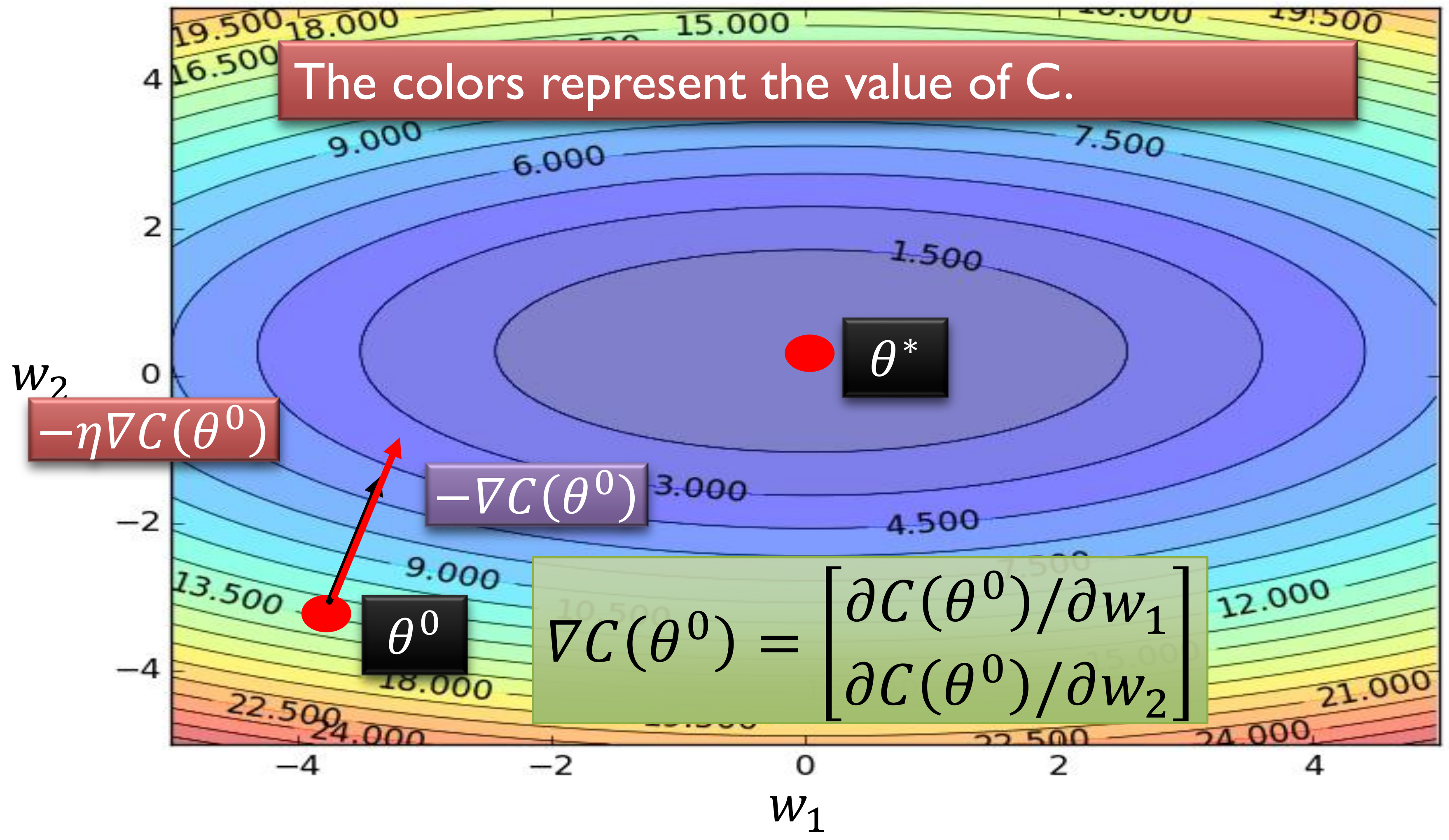
Find the network parameters θ^* that minimize this value

Gradient Descent

Assume there are only two parameters w_1 and w_2 in a network.

$$\theta = \{w_1, w_2\}$$

Error Surface



$$\nabla C(\theta^0) = \begin{bmatrix} \partial C(\theta^0)/\partial w_1 \\ \partial C(\theta^0)/\partial w_2 \end{bmatrix}$$

Randomly pick a starting point θ^0

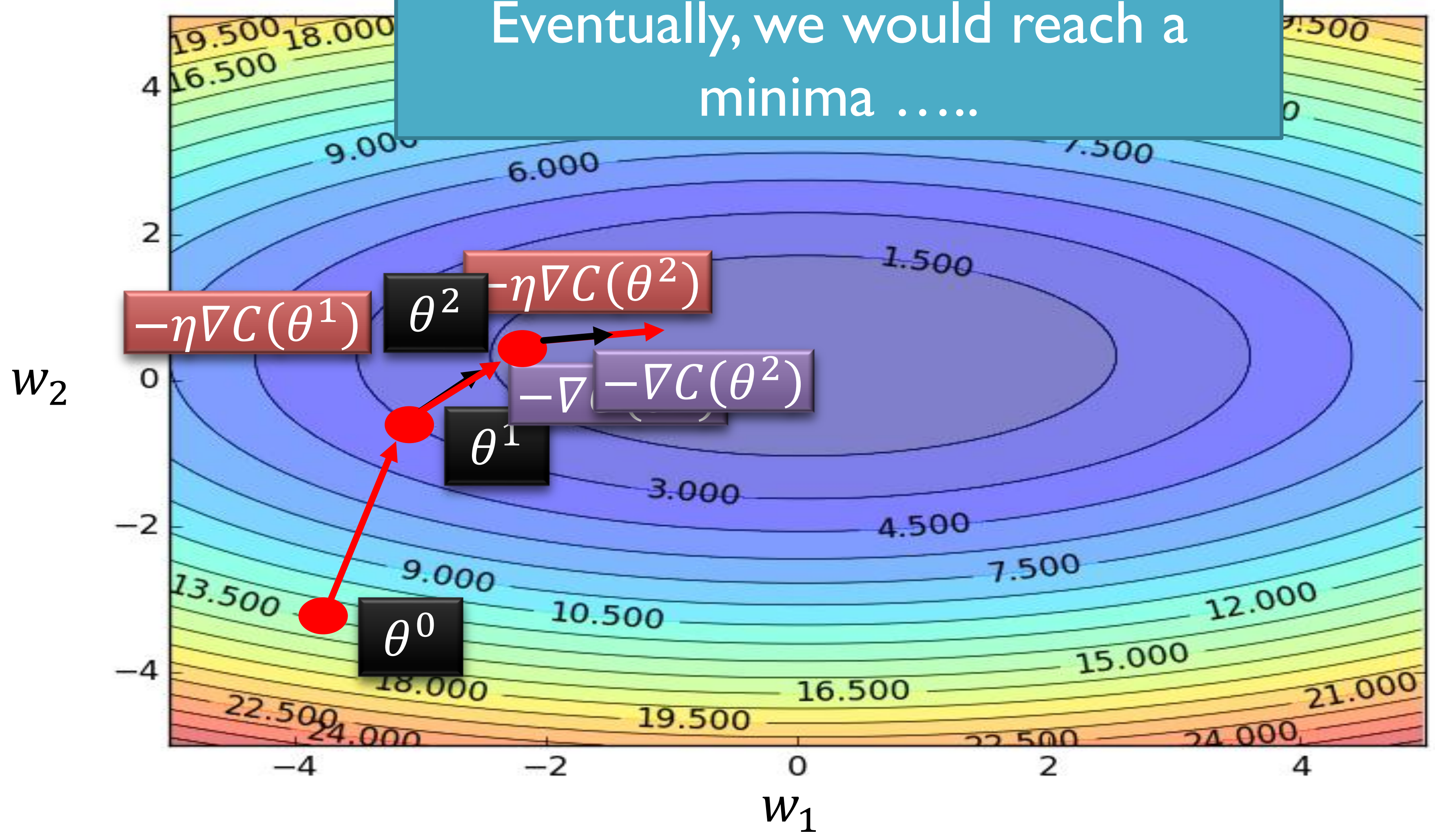
Compute the negative gradient at θ^0

$\rightarrow -\nabla C(\theta^0)$

Times the **learning rate** η

$\rightarrow -\eta\nabla C(\theta^0)$

Gradient Descent



Randomly pick a starting point θ^0

Compute the negative gradient at θ^0

$\Rightarrow -\nabla C(\theta^0)$

Times the learning rate η

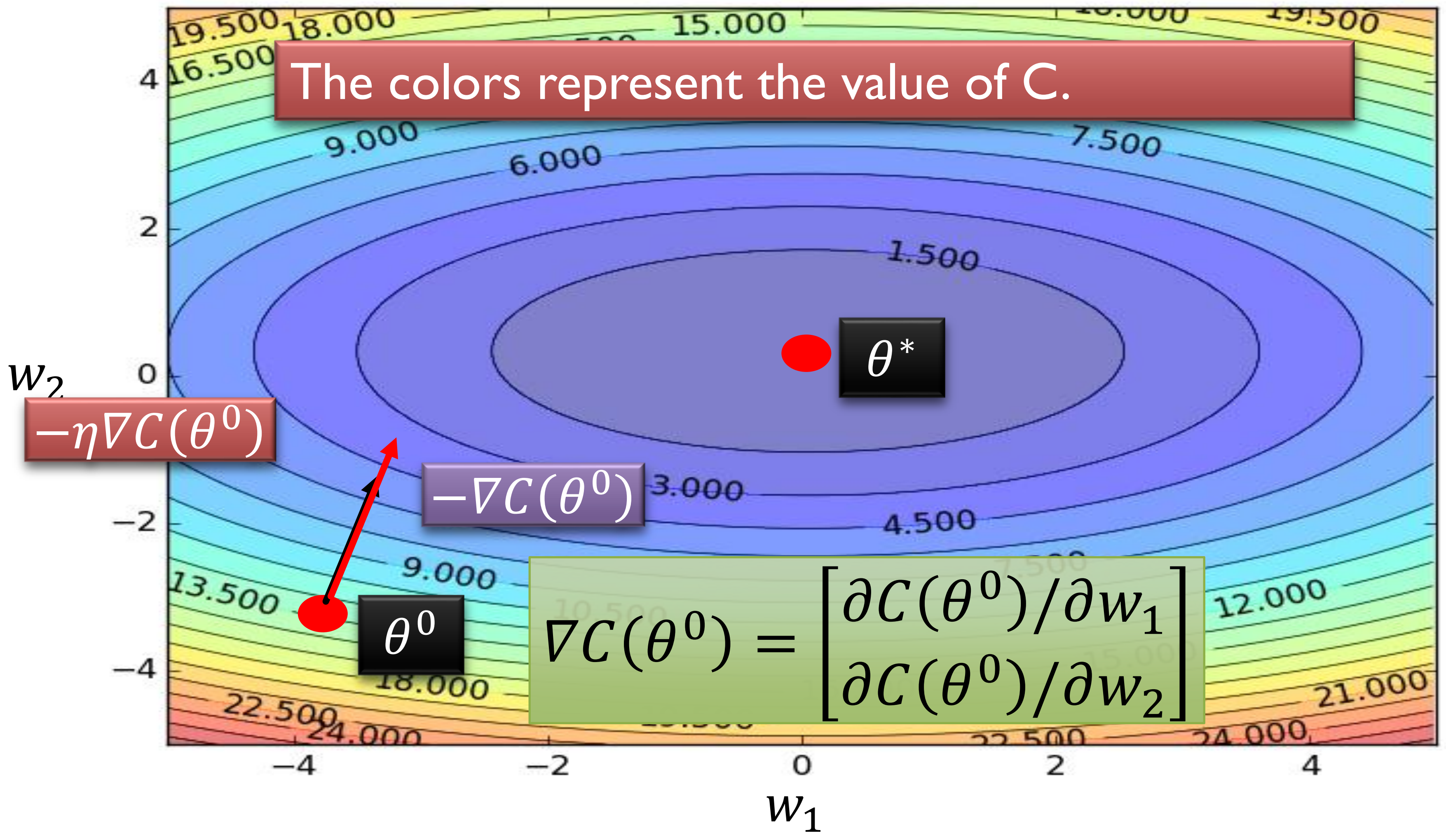
$\Rightarrow -\eta \nabla C(\theta^0)$

Gradient Descent

Assume there are only two parameters w_1 and w_2 in a network.

$$\theta = \{w_1, w_2\}$$

Error Surface



$$\nabla C(\theta^0) = \begin{bmatrix} \partial C(\theta^0)/\partial w_1 \\ \partial C(\theta^0)/\partial w_2 \end{bmatrix}$$

Randomly pick a starting point θ^0

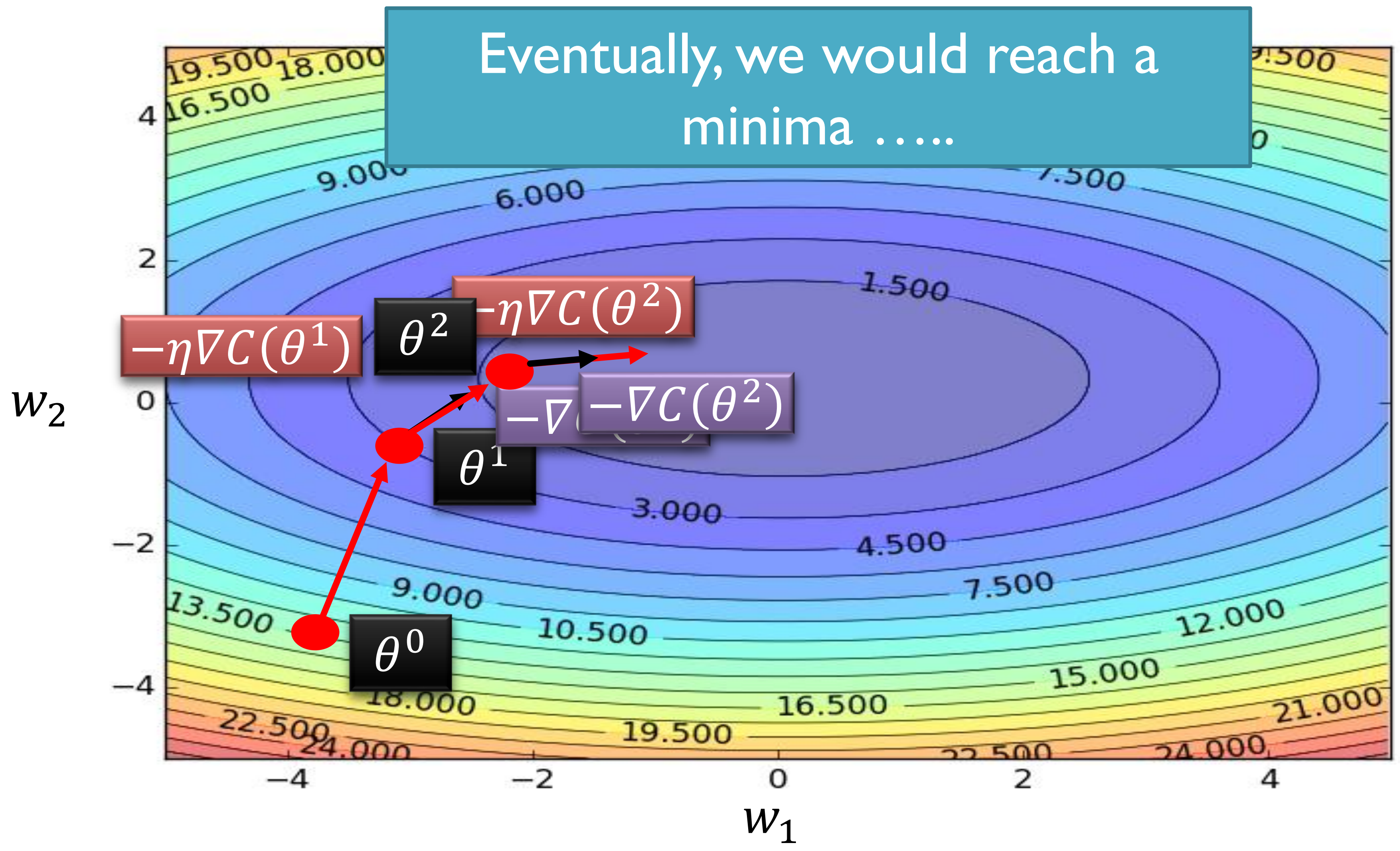
Compute the negative gradient at θ^0

$\Rightarrow -\nabla C(\theta^0)$

Times the learning rate η

$\Rightarrow -\eta \nabla C(\theta^0)$

Gradient Descent



Randomly pick a starting point θ^0

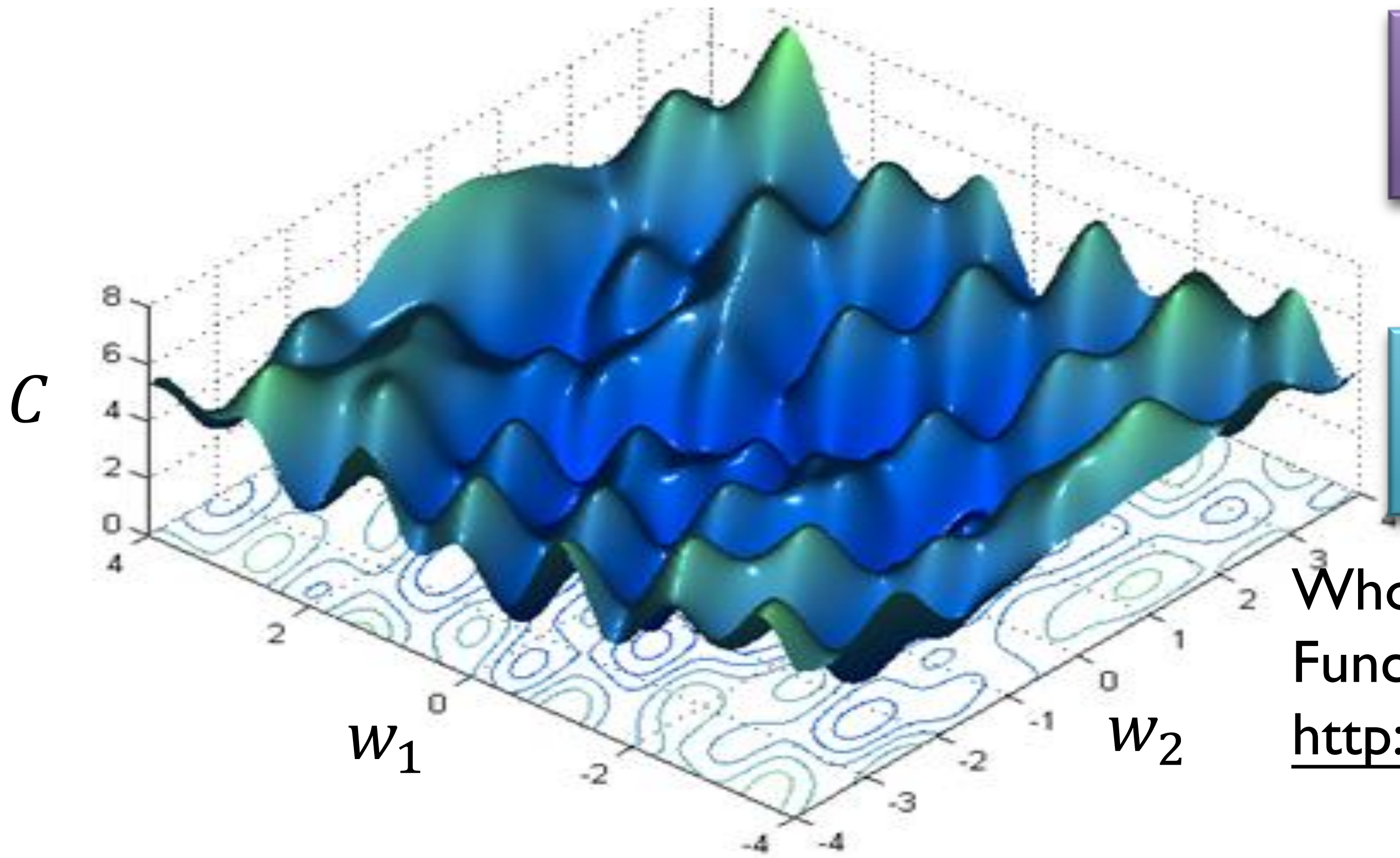
Compute the negative gradient at θ^0

$\Rightarrow -\nabla C(\theta^0)$

Times the learning rate η

$\Rightarrow -\eta\nabla C(\theta^0)$

Local Minima

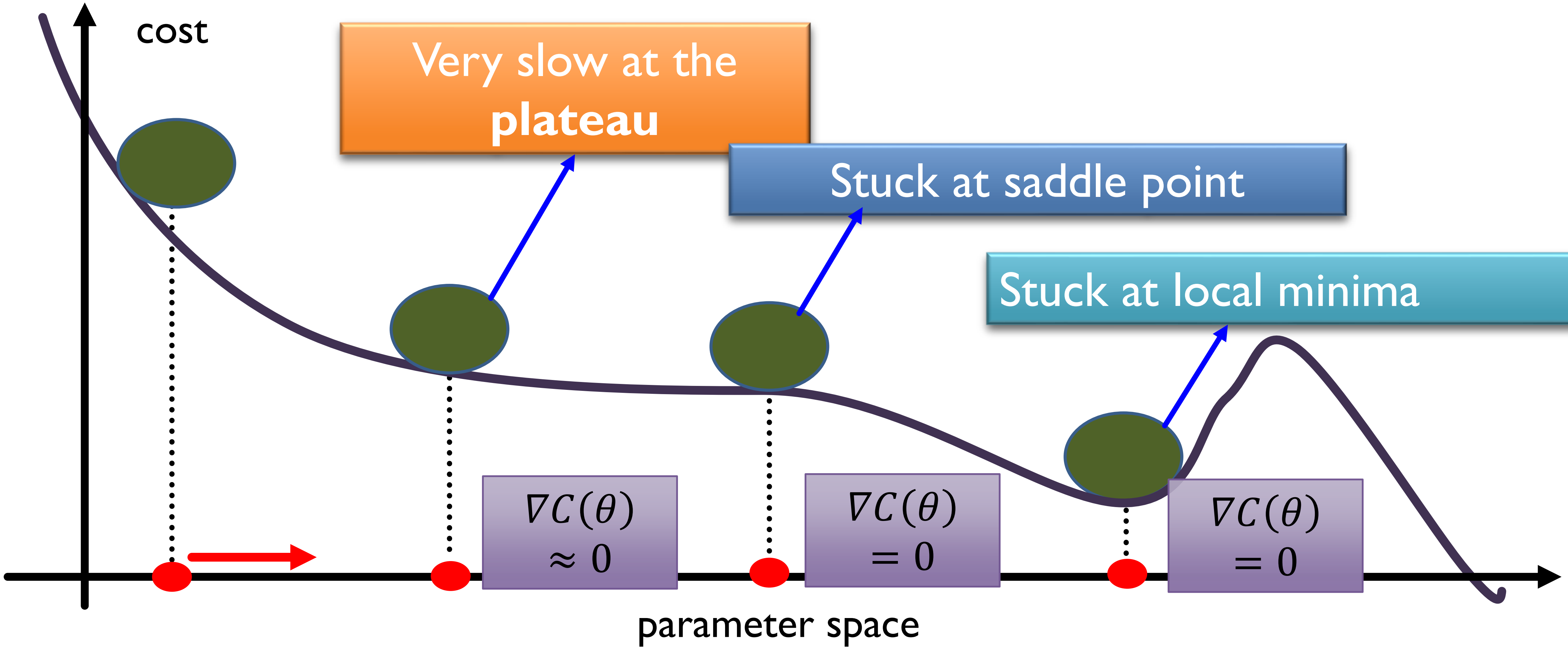


Different initial point θ^0

Reach different minima, so different results

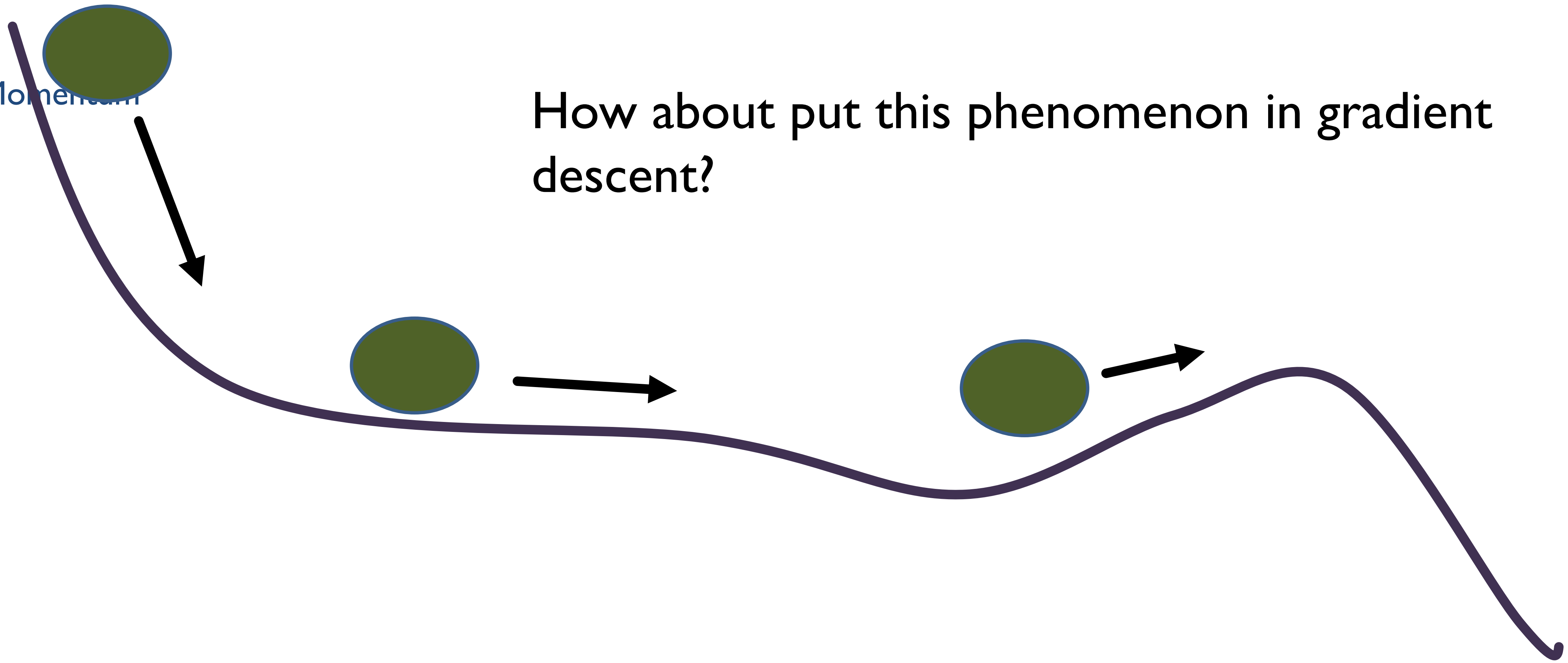
Who is Afraid of Non-Convex Loss Functions?
http://videolectures.net/eml07_lecun_wia/

Besides local minima ……



In physical world ……

- Momentum

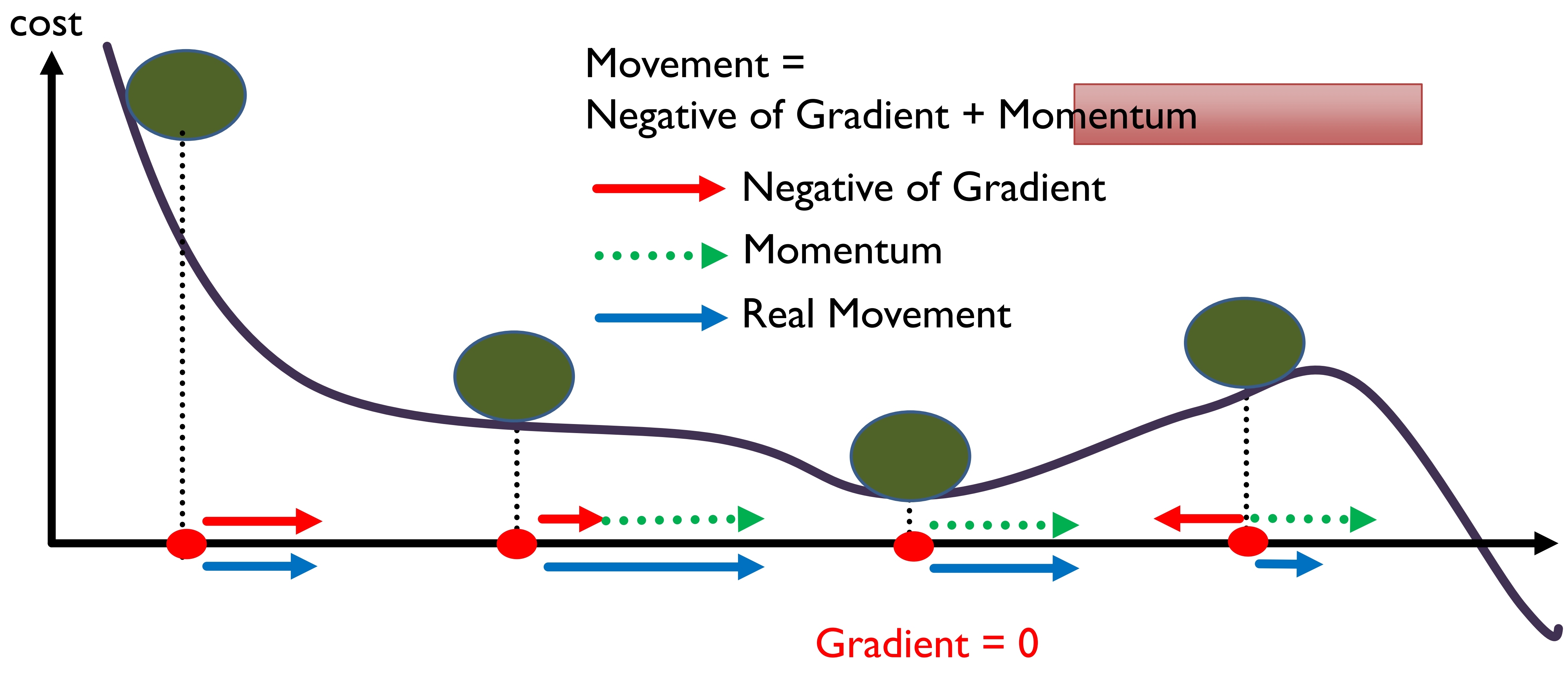


How about put this phenomenon in gradient descent?

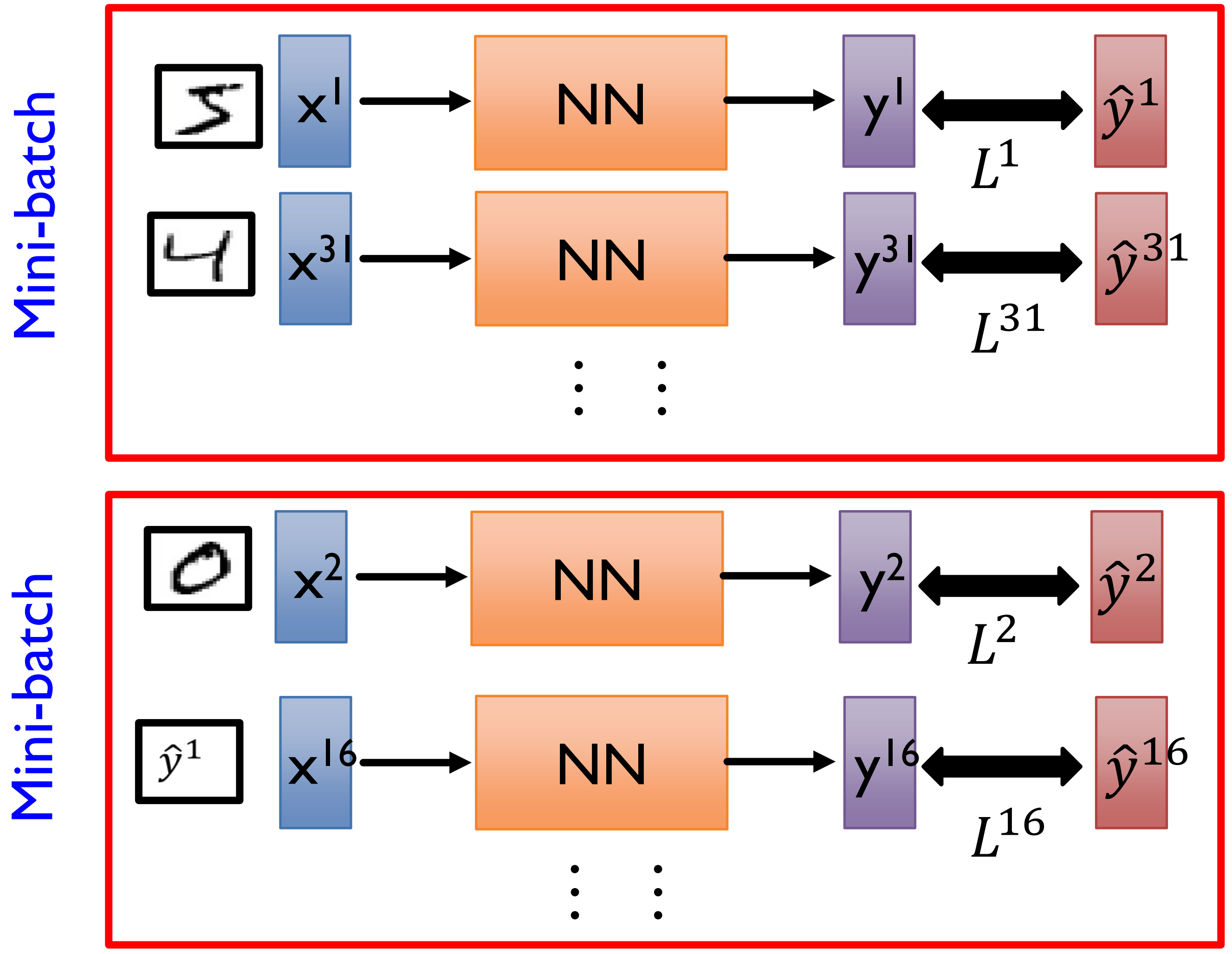
Momentum

Still not guarantee reaching global minima, but give some hope ……

Movement =
Negative of Gradient + Momentum



Mini-batch



- Randomly initialize θ^0
- Pick the 1st batch

$$C = L^1 + L^{31} + \cdots$$

$$\theta^1 \leftarrow \theta^0 - \eta \nabla C(\theta^0)$$

- Pick the 2nd batch

$$C = L^2 + L^{16} + \cdots$$

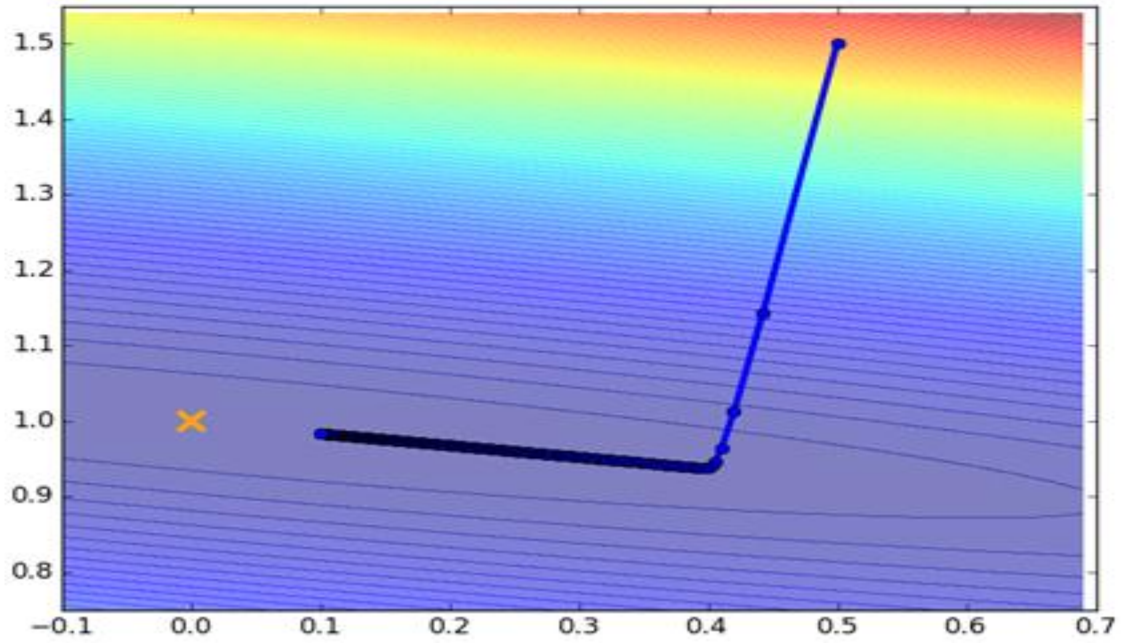
$$\theta^2 \leftarrow \theta^1 - \eta \nabla C(\theta^1)$$

$$\vdots$$

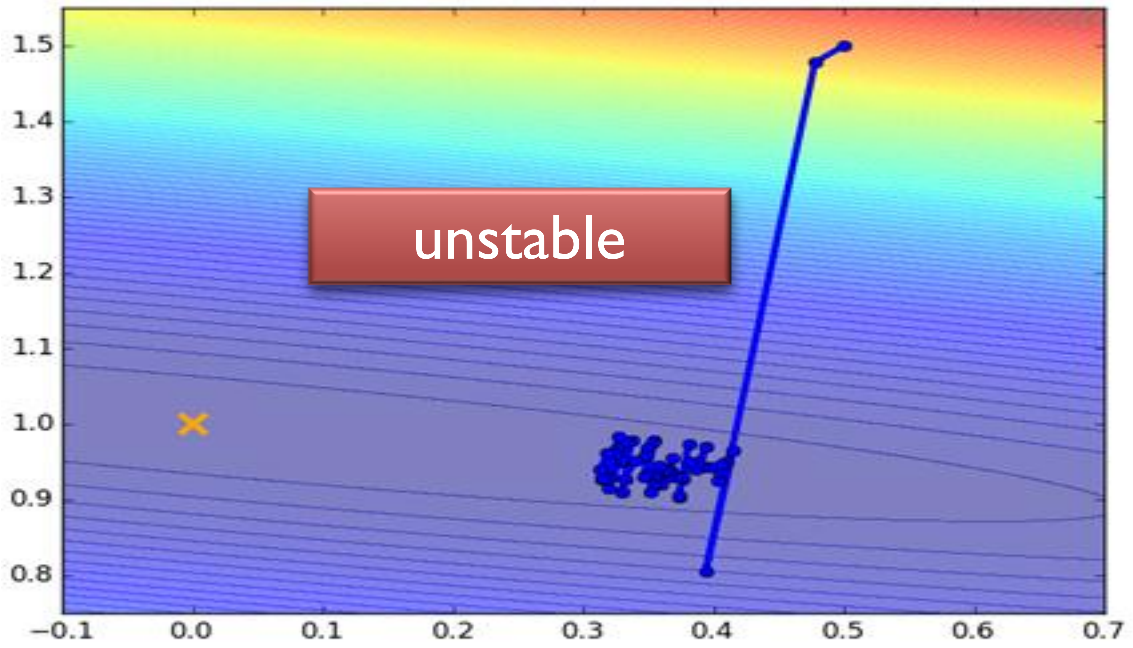
C is different each time when we update parameters!

Mini-batch

Original Gradient Descent

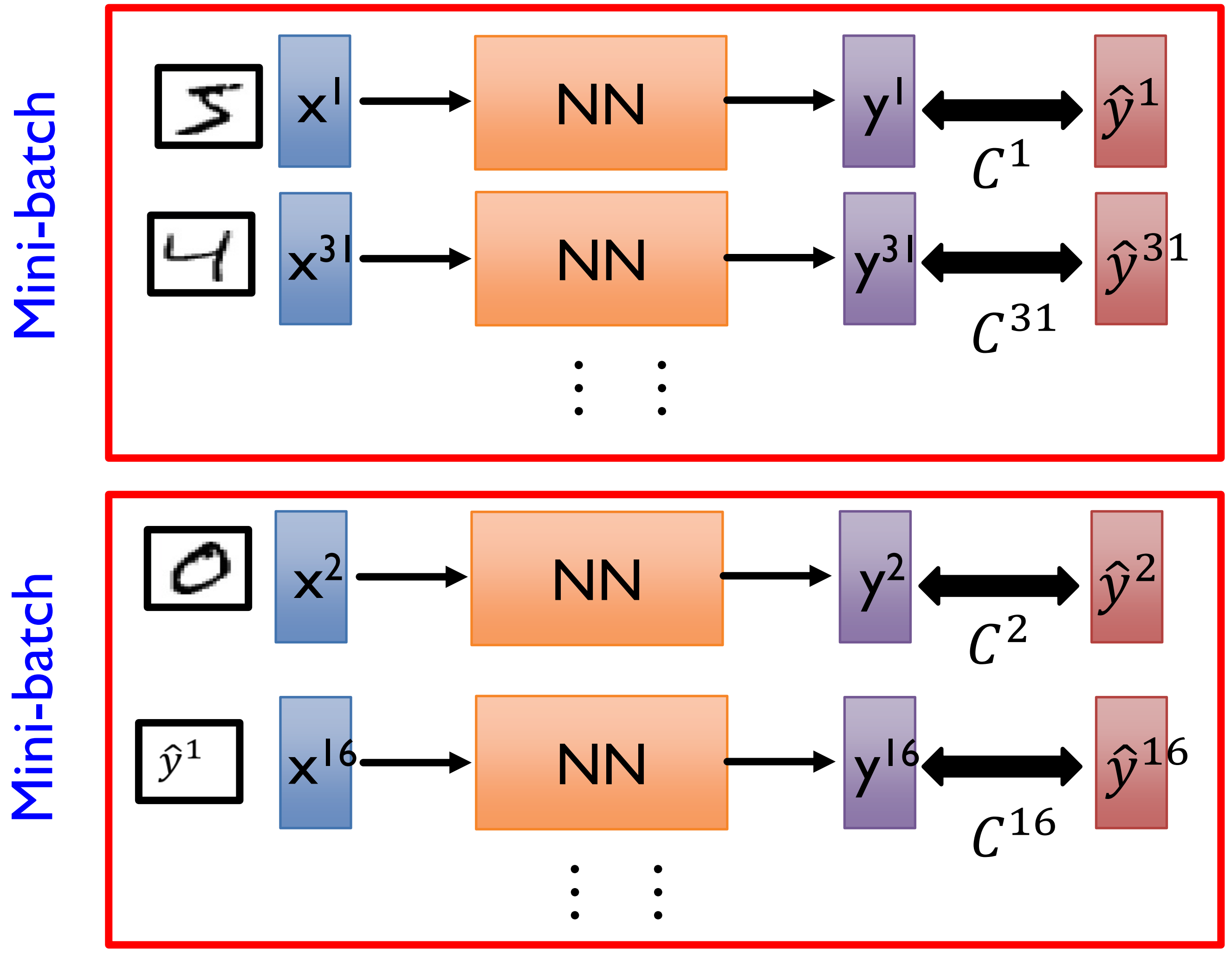


With Mini-batch



The colors represent the total C on all training data.

Mini-batch



Faster

Better!

- Randomly initialize θ^0
- Pick the 1st batch

$$C = C^1 + C^{31} + \cdots$$

$$\theta^1 \leftarrow \theta^0 - \eta \nabla C(\theta^0)$$

- Pick the 2nd batch

$$C = C^2 + C^{16} + \cdots$$

$$\theta^2 \leftarrow \theta^1 - \eta \nabla C(\theta^1)$$

$$\vdots$$

- Until all mini-batches have been picked

one epoch

Repeat the above process

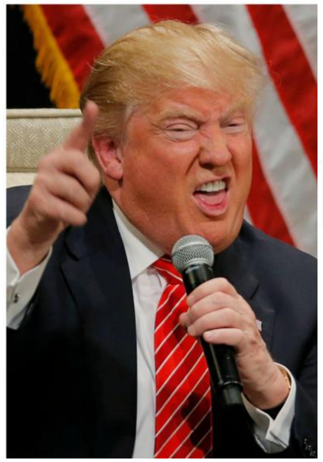
Training the network

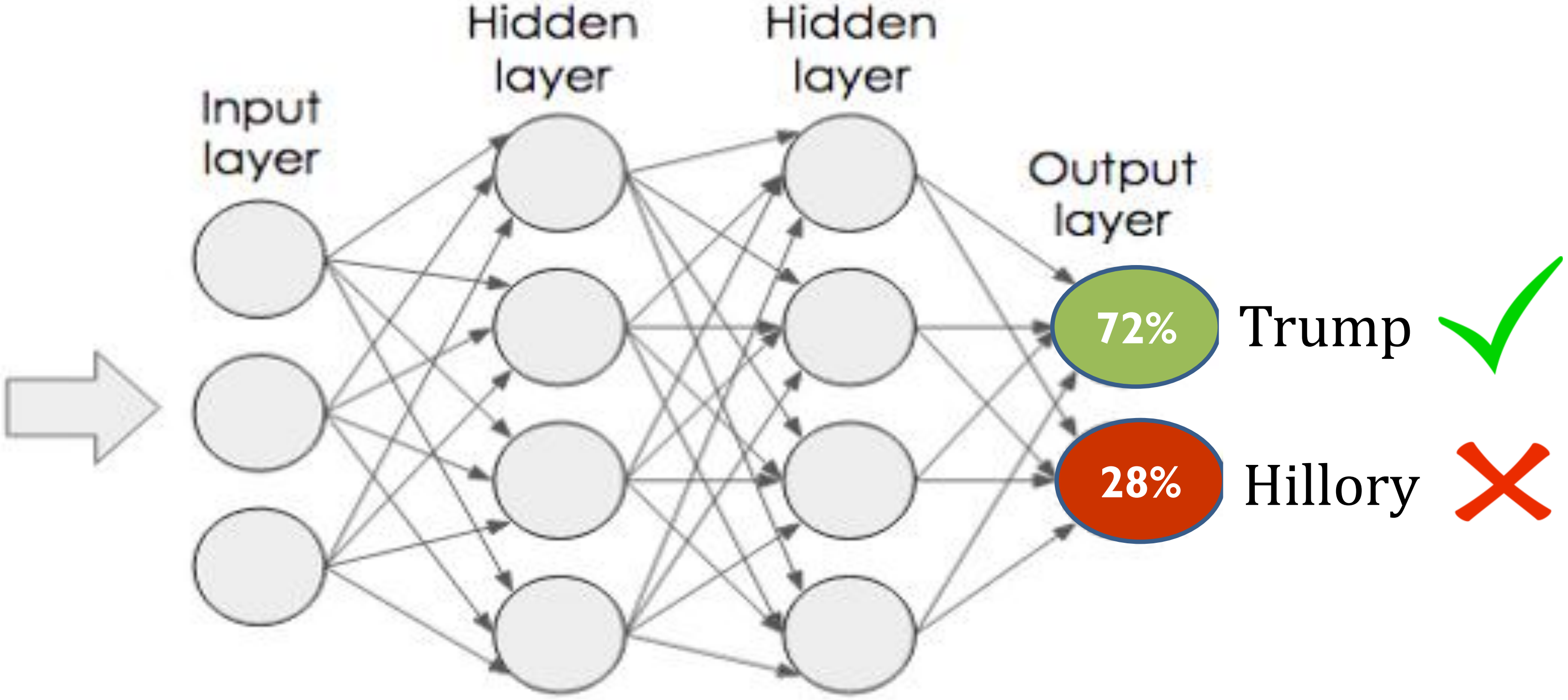
1. Randomly initialise the network weights and biases
2. **Training:**
   - Get a ton of labelled training data (e.g. pictures of cats labelled ‘cats’ and pictures of other stuff also correctly labelled)
   - For every piece of training data, feed it into the network
3. **Testing:**
   - Check whether the network gets it right (given a picture labelled ‘cat’, is the result of the network also ‘cat’ or is it ‘dog’, or something else?)
   - If not, how wrong was it? Or, how right was it? (What probability did it assign to its guess?)
4. **Tuning /Improving:** Nudge the weights a little to increase the probability of the network more probability getting the answer right.

Example – Classify the images – Trump & Hillory

Training the network

Predict

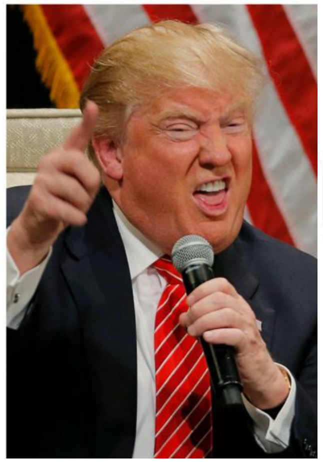


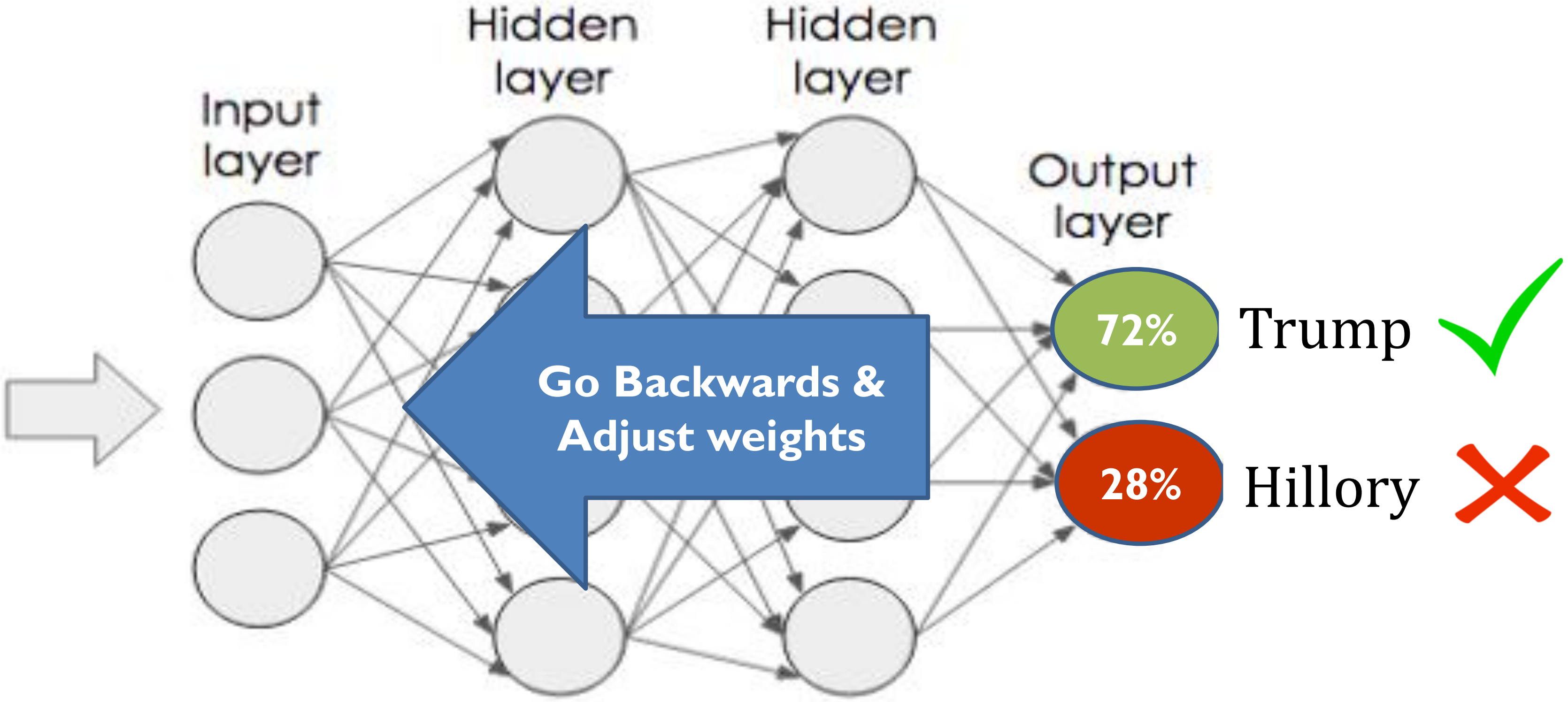


1. Two output neurons (for each class one neuron)
2. O/P: A probability of 72% that the image is "Trump" → 72 % is Not sufficient

How to increase the accuracy?

Predict

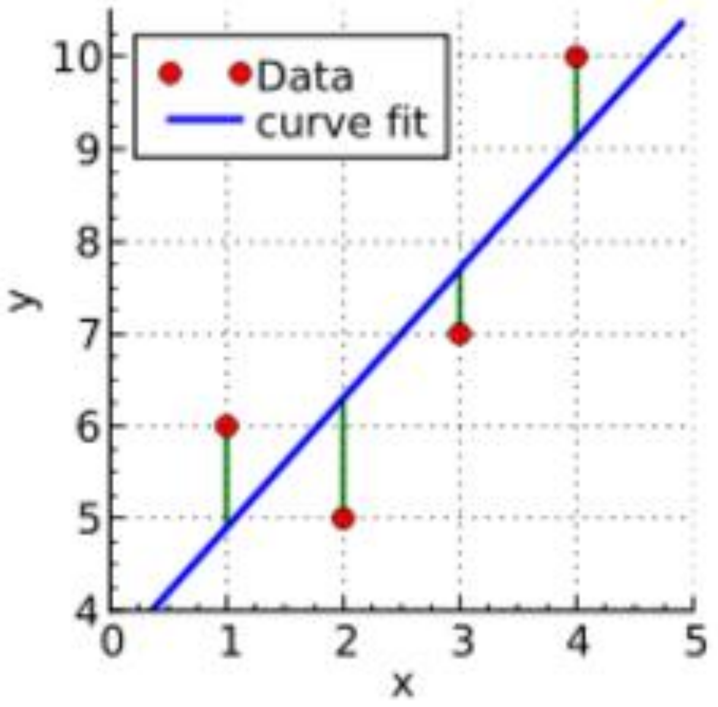




1. Two output neurons for each class
2. O/P: A probability of **72%** that the image is "Trump" → 72 % is **Not sufficient**

How to increase the accuracy?

- Measure the network's output and the correct output using the '**loss function**'
- Which Loss Function??
  - Depends on the data
  - **Example**: Mean Square Error (MSE)



How to increase the accuracy?

- Measure the network's output and the correct output using the '**loss function**'
- Which Loss Function??
  - Depends on the data
  - **Example**: Mean Square Error

The **goal of training** is to find the weights and biases that **minimizes the loss function**.

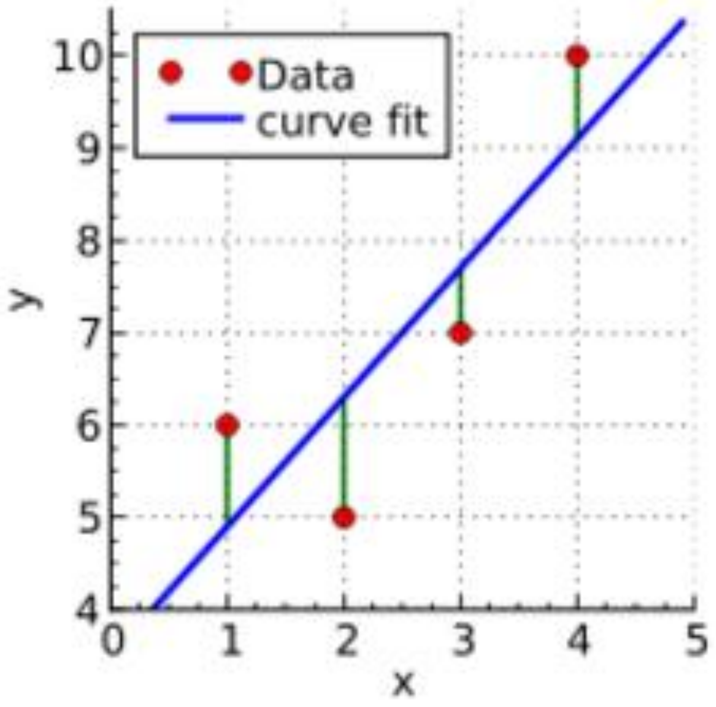


Image source: firsttimeprogrammer.blogspot.co.uk

How to increase the accuracy?

- We can plot the loss against the weights.
- To do this accurately, we would need to be able to visualise tons of dimensions, to account for the many weights and biases in the network.
- Because I find it difficult to visualise more than three dimensions,
- Let's pretend we only need to find two weight values. We can then use the third dimension for the loss.
- Before training the network, the weights and biases are randomly initialised so the loss function is likely to be high as the network will get a lot of things wrong.
- Our aim is to find the lowest point of the loss function, and then see what weight values that corresponds to. (See Next Slide).

How to increase the accuracy?

- Here, we can easily see where the lowest point is and could happily read off the corresponding weights values.
- Unfortunately, it's not that easy in reality. The network doesn't have a nice overview of the loss function, it can only know what its current loss is and its current weights and biases are.

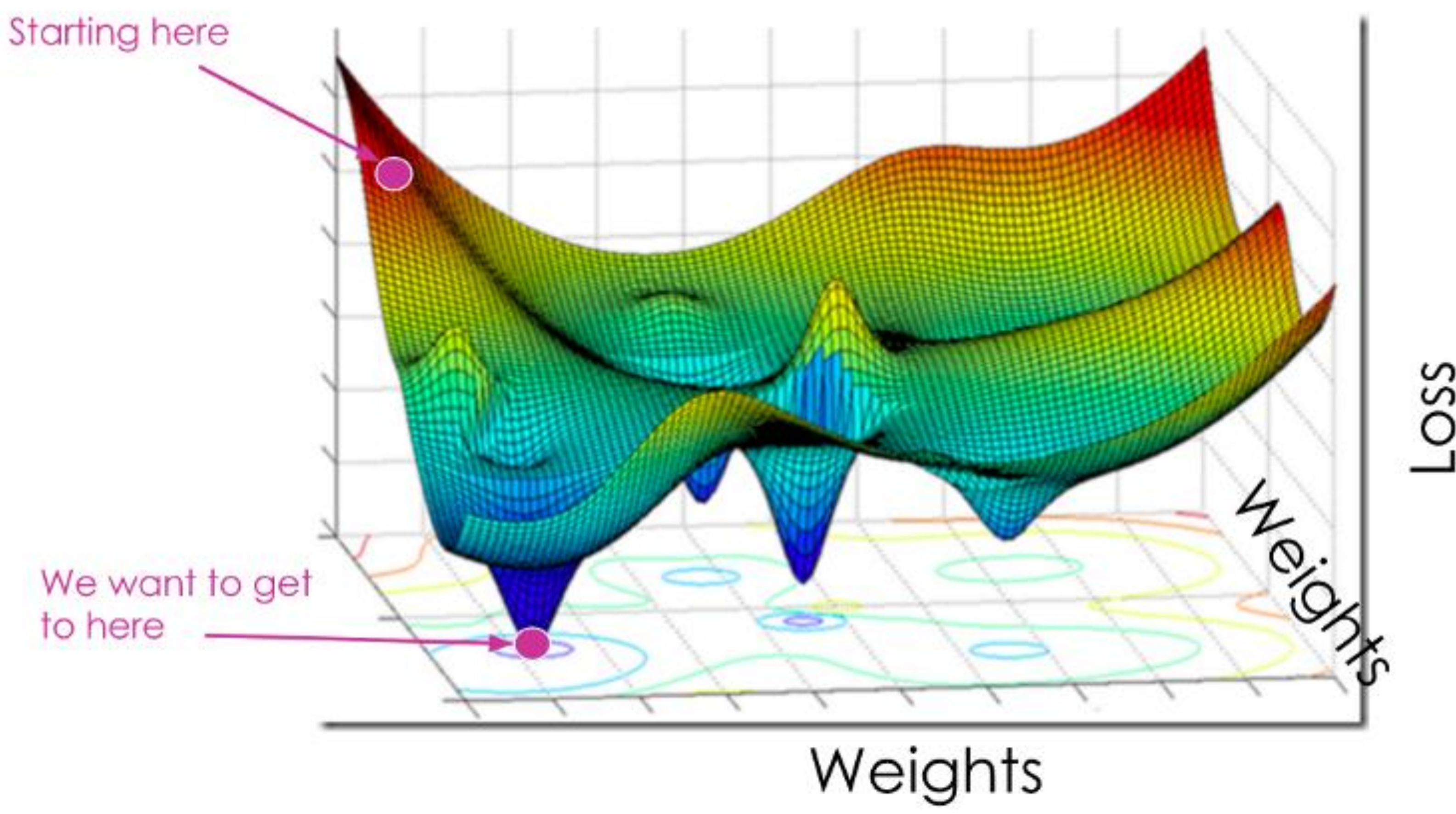


Image source: firsttimeprogrammer.blogspot.co.uk

How to increase the accuracy – Gradient Descent

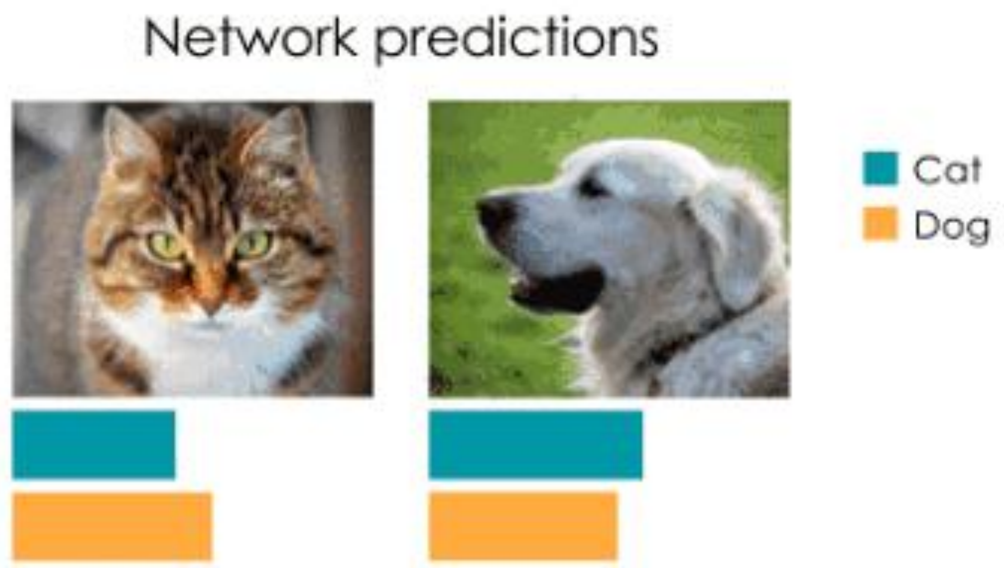


Image source: firsttimeprogrammer.blogspot.co.uk

Back Propagation

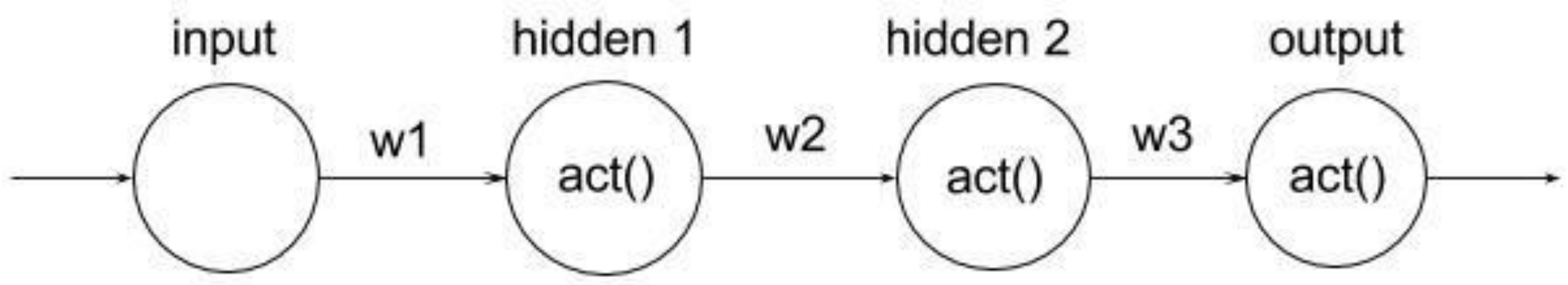
How a neuron works?

$$a_n = act(W_1 x_1 + \ldots + W_n x_n)$$

1. Each neuron computes a weighted sum of the outputs of the previous layer (which, for the inputs, we'll call **x**),
2. applies an activation function, before forwarding it on to the next layer.
3. Each neuron in a layer has its own set of weights—so while each neuron in a layer is looking at the same inputs, their outputs will all be different.
4. The final layer's activations are the predictions that the network actually makes.

How A Network Works?

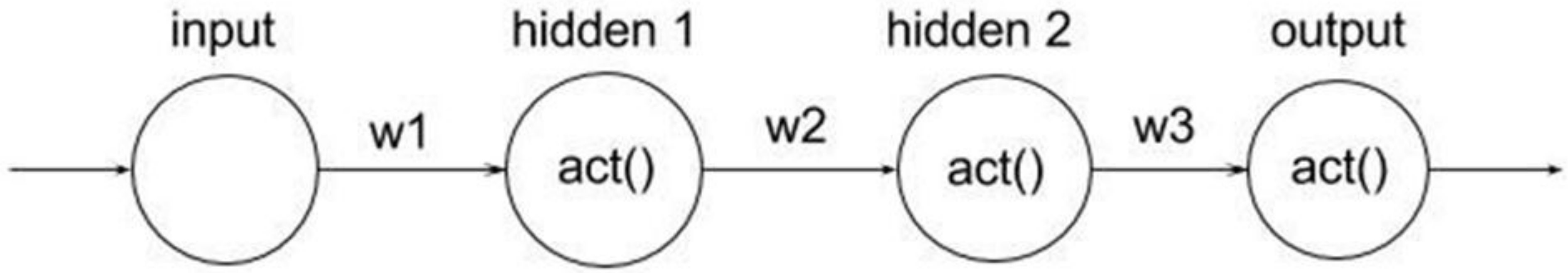
1. we want to find a set of weights **W** that minimizes on the output of **J(W)**.
2. Consider a simple double layer Network.



3. each neuron is a function of the previous one connected to it. In other words, if one were to change the value of **w1**, both "hidden 1" *and* "hidden 2" (and ultimately the output) neurons would change.

How a network works?

we can mathematically formulate the output as an extensive composite function:

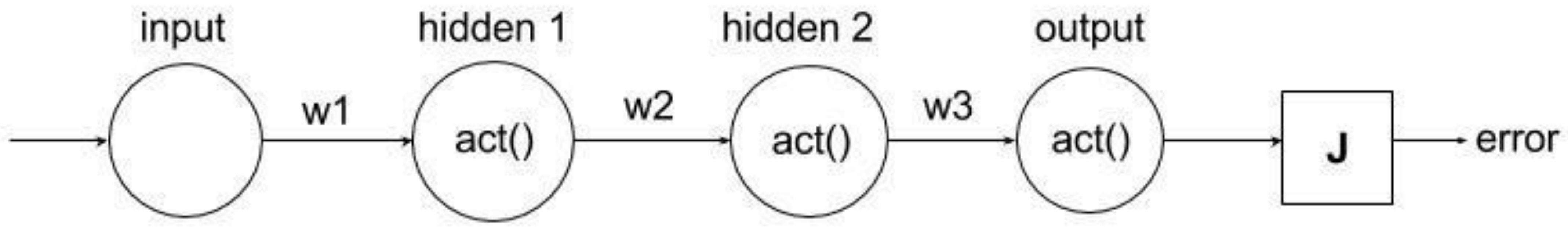


$$output = act(w3 * hidden2)$$
$$hidden2 = act(w2 * hidden1)$$
$$hidden1 = act(w1 * input)$$

$$output = act(w3 * act(w2 * act(w1 * input)))$$

Introduce Error

we can mathematically formulate the output as an extensive composite function:



J is a function of → input, weights, activation function and output

How to minimize the error – back propagation

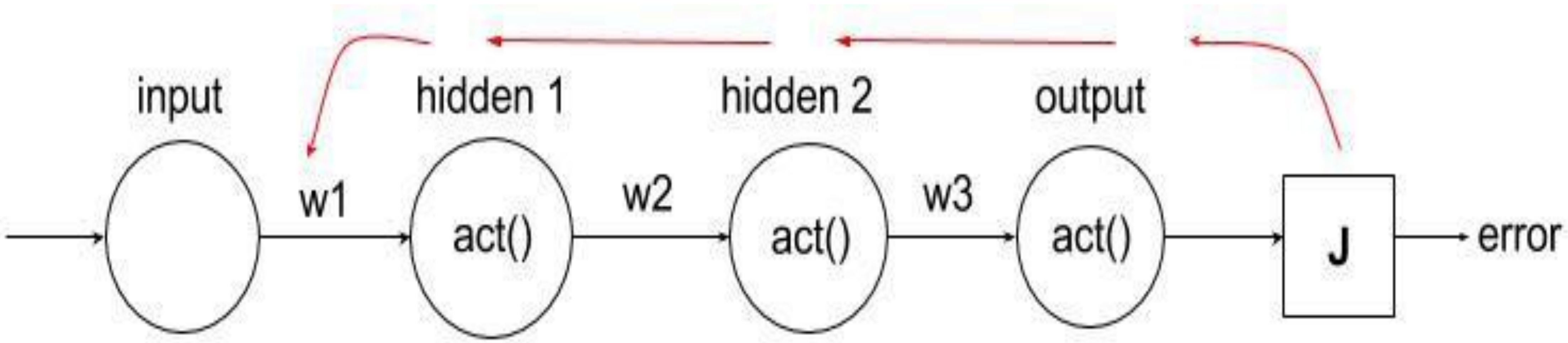
The error derivative can be used in gradient descent (as discussed earlier) to iteratively improve upon the weight.

We compute the error derivatives w.r.t. every other weight in the network and apply gradient descent in the same way.

$$\frac{\partial error}{\partial w1} = \frac{\partial error}{\partial output} * \frac{\partial output}{\partial hidden2} * \frac{\partial hidden2}{\partial hidden1} * \frac{\partial hidden1}{\partial w1}$$

Back propagation

Find out the best "w1" by reducing the error from the last layer to first layer….



Back propagation

How Backpropagation works?

Refer the document – backprop.docx

Deep Learning Tools

MACHINE LEARNING TOOLS

Python

scikits learn
machine learning in Python

Java

MALLET
machine learning for language toolkit

.net

Accord .NET Framework

We need scalable Machine Learning/Data Mining Algorithms

mahout

Java, Scala

Spark
MLlib

Python, Scala

R

Statistical Analysis

C, FORTRAN Languages

MACHINE LEARNING TOOLS/FRAMEWORKS

Caffe

Caffe is a deep learning framework made with expression, speed, and modularity in mind. Caffe is developed by the Berkeley Vision and Learning Center (BVLC), as well as community contributors and is popular for computer vision.
Caffe supports cuDNN v5 for GPU acceleration.

Supported interfaces: C, C++, Python,.

Caffe2

Caffe2 is a deep learning framework enabling simple and flexible deep learning. Built on the original Caffe, Caffe2 is designed with expression, speed, and modularity in mind, allowing for a more flexible way to organize computation.
Caffe2 supports cuDNN v5.1 for GPU acceleration.

Supported interfaces: C++, Python

MACHINE LEARNING TOOLS/FRAMEWORKS

The Microsoft Toolkit — previously known as CNTK— is a unified deep-learning toolkit from Microsoft Research that makes it easy to train and combine popular model types across multiple GPUs and servers. Microsoft Cognitive Toolkit implements highly efficient CNN and RNN training for speech, image and text data. Microsoft Cognitive Toolkit supports cuDNN v5.1 for GPU acceleration.

Supported interfaces: Python, C++, C# and Command line interface

TensorFlow is a software library for numerical computation using data flow graphs, developed by Google's Machine Intelligence research organization.

TensorFlow supports cuDNN v5.1 for GPU acceleration.

Supported interfaces: C++, Python

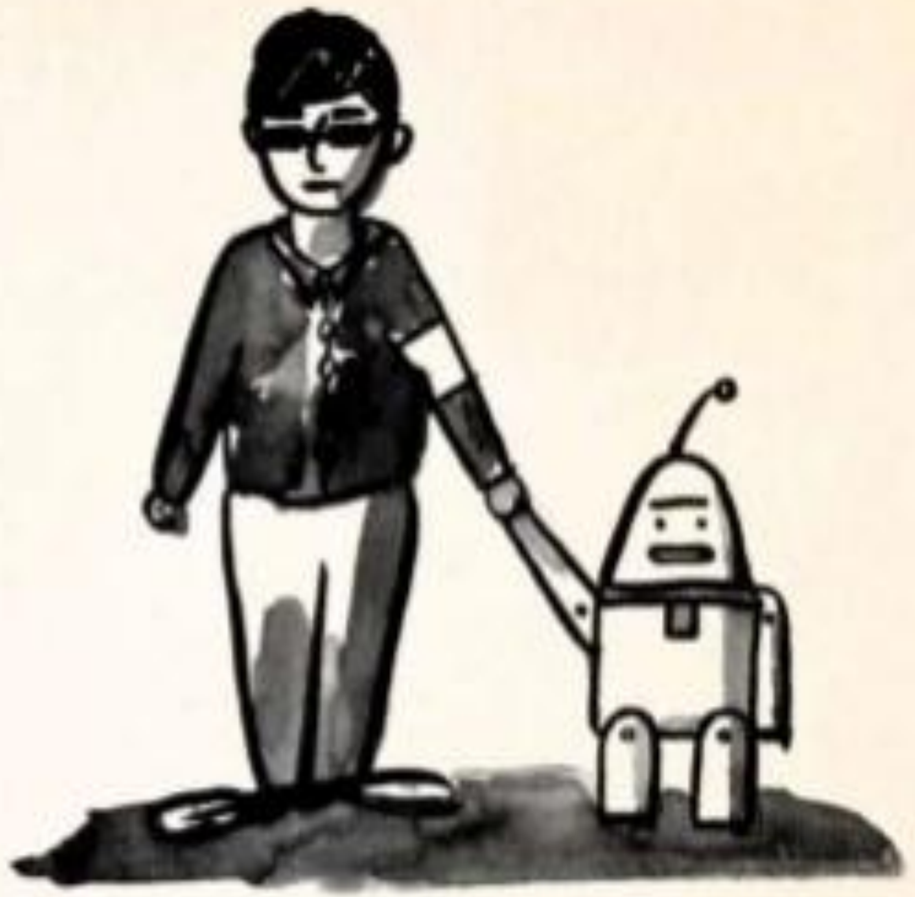
MACHINE LEARNING TOOLS/FRAMEWORKS

Theano is a math expression compiler that efficiently defines, optimizes, and evaluates mathematical expressions involving multi-dimensional arrays.
Theano supports cuDNN v5 for GPU acceleration.

Supported interfaces: Python

Keras is a minimalist, highly modular neural networks library, written in Python, and capable of running on top of either TensorFlow or Theano. Keras was developed with a focus on enabling fast experimentation.
cuDNN version depends on the version of TensorFlow and Theano installed with Keras.

Supported Interfaces: Python



THANK YOU!

QUESTIONS?