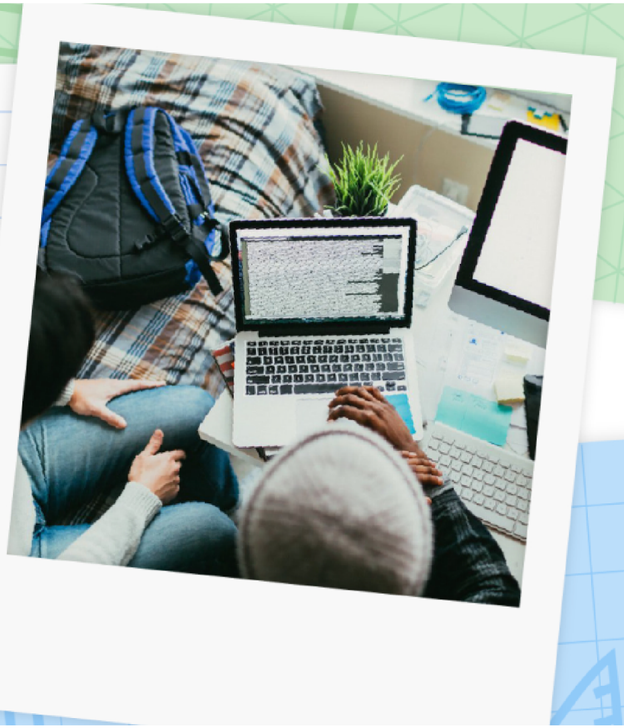
Google Developer Student Clubs

Sup'com - Higher School of Communication



Plan

1-Deep learning history

2-Deep learning Libraries overview

3-Fully connected layers

4- Full CNN layer

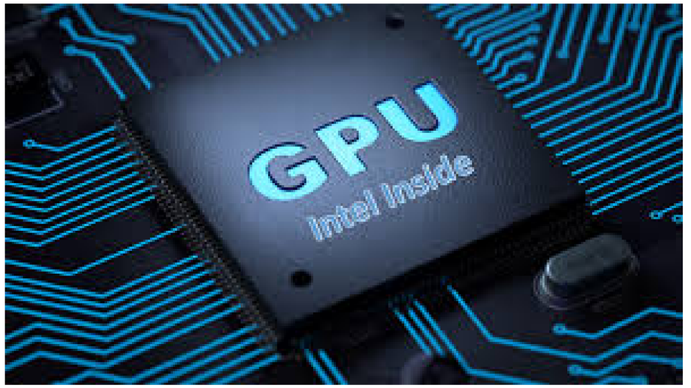
5-Loss Function & Backpropagation

6-Project Overview

Unveiling the Past

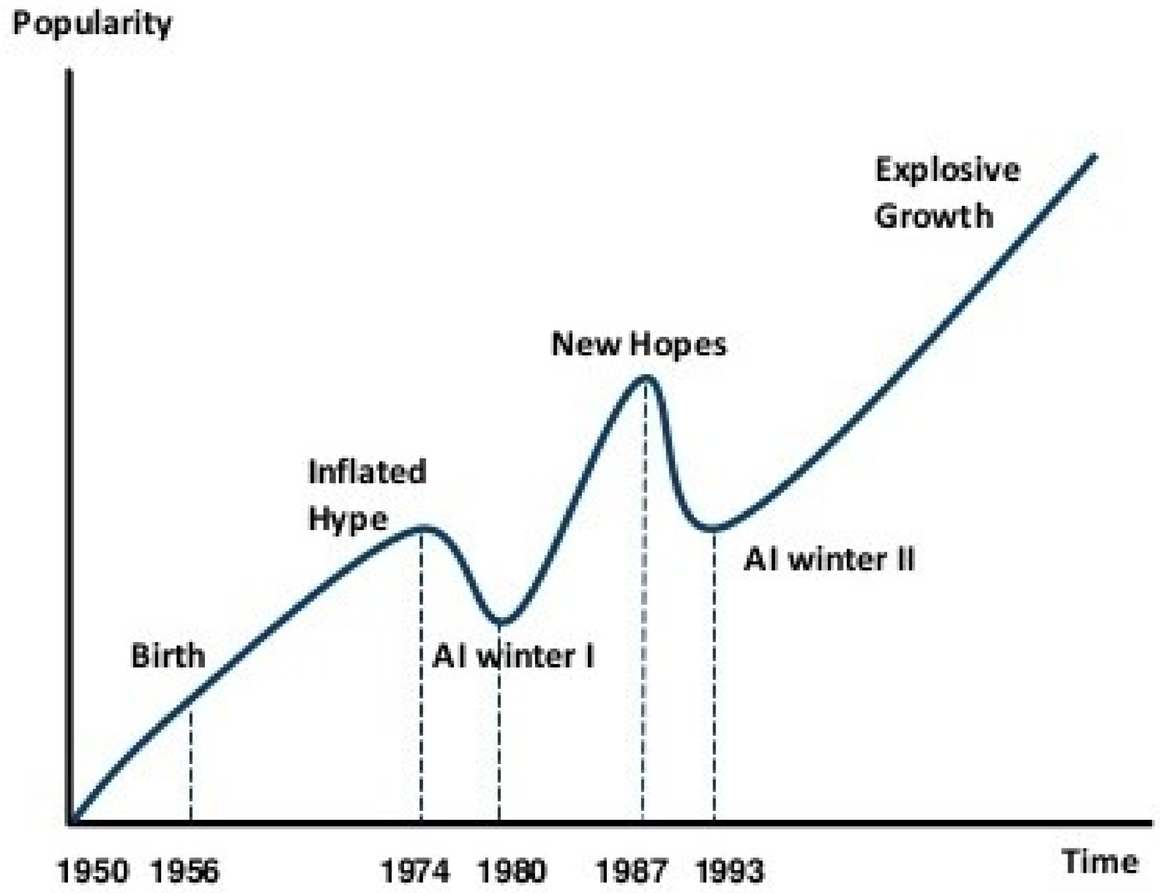
Now is the best time to be a Deep Learning Practitioner

- We’re in the Golden Age of AI
- Fast Graphics Processing Unit (GPU) hardware is relatively cheap and readily available
- Cheap even free cloud GPUs are available
- Deep Learning Libraries are mature and relatively easy to use



History of Deep Learning

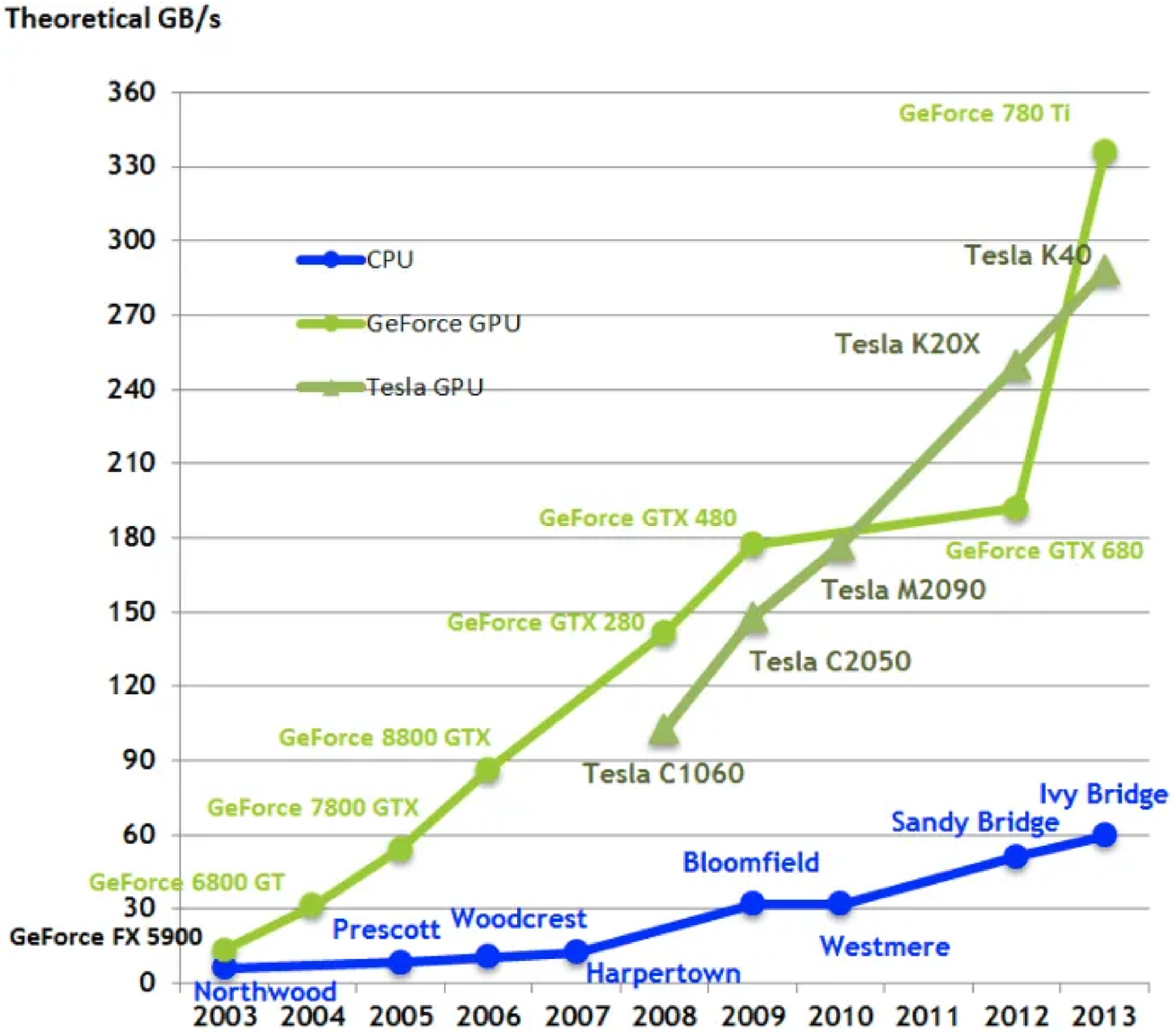
AI HAS A LONG HISTORY OF BEING "THE NEXT BIG THING"...



Timeline of AI Development

- **1950s-1960s**: First AI boom - the age of reasoning, prototype AI developed
- **1970s**: AI winter I
- **1980s-1990s**: Second AI boom: the age of Knowledge representation (appearance of expert systems capable of reproducing human decision-making)
- **1990s**: AI winter II
- **1997**: Deep Blue beats Gary Kasparov
- **2006**: University of Toronto develops Deep Learning
- **2011**: IBM's Watson won Jeopardy
- **2016**: Go software based on Deep Learning beats world's champions

GPUs have come a long way



Deep Learning Frameworks

PyTorch, TensorFlow, Keras and others!

While PyTorch and TensorFlow/Keras have come out on top, many others vied for this title such as Theano, Caffe, MXNet, PaddlePaddle

TensorFlow

Developed by Google

- Free and open-source Deep Learning Framework
- Developed internally by Google Brain and open-sourced in 2015
- Written in C++ and CUDA (NVIDIA’s Compute Unified Device Architecture API for programming on GPUs)
- Primarily used in Python but APIs/wrappers exist for several other languages (C++, Go, Java, JavaScript, Swift)
- TensorFlow 2.0 was released in 2019

Pytorch

Developed by Facebook

Free and open-source Deep Learning Library

Developed by Facebook's AI Research Team

It is a C-based tensor Library written with CUDA capabilities

PyTorch is Python based scientific computing package aimed at:

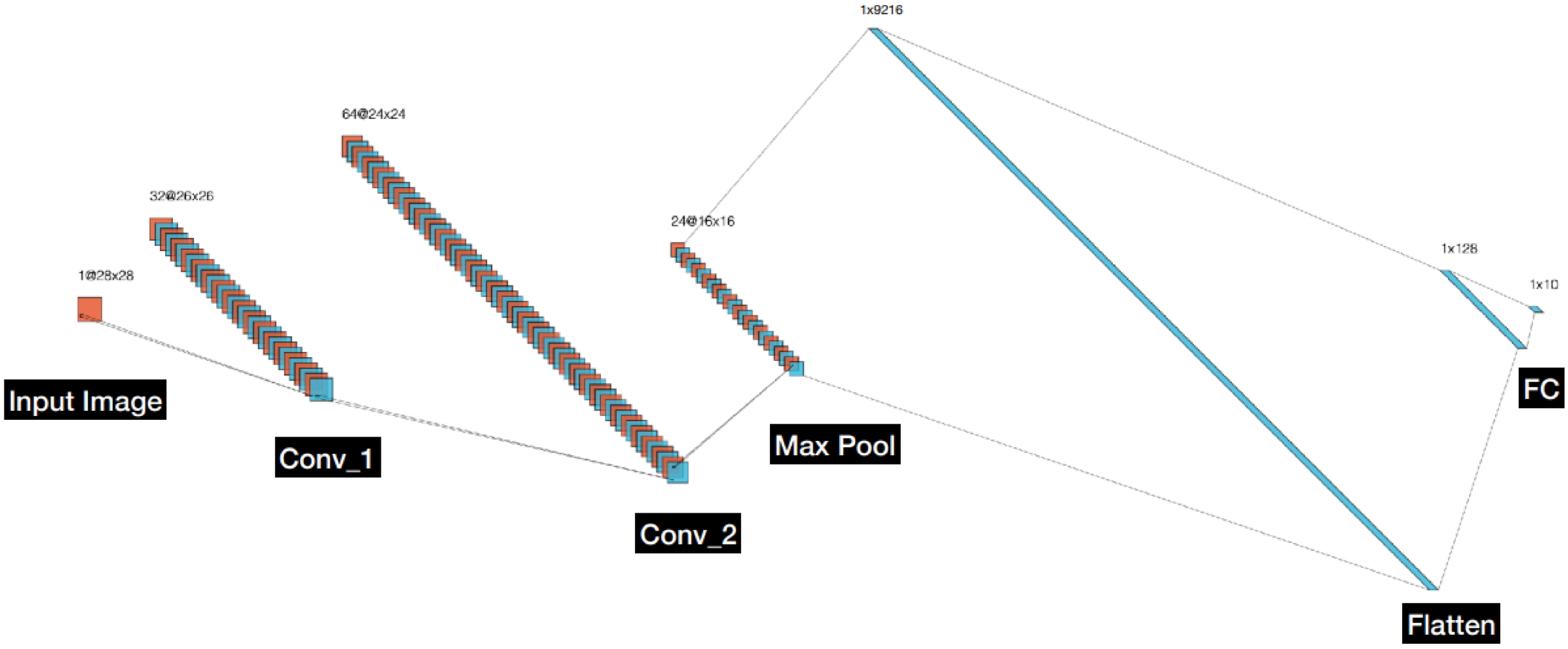
1. A replacement for NumPy using the power of GPUs
2. A Deep Learning research platform providing both flexibility and speed

Keras

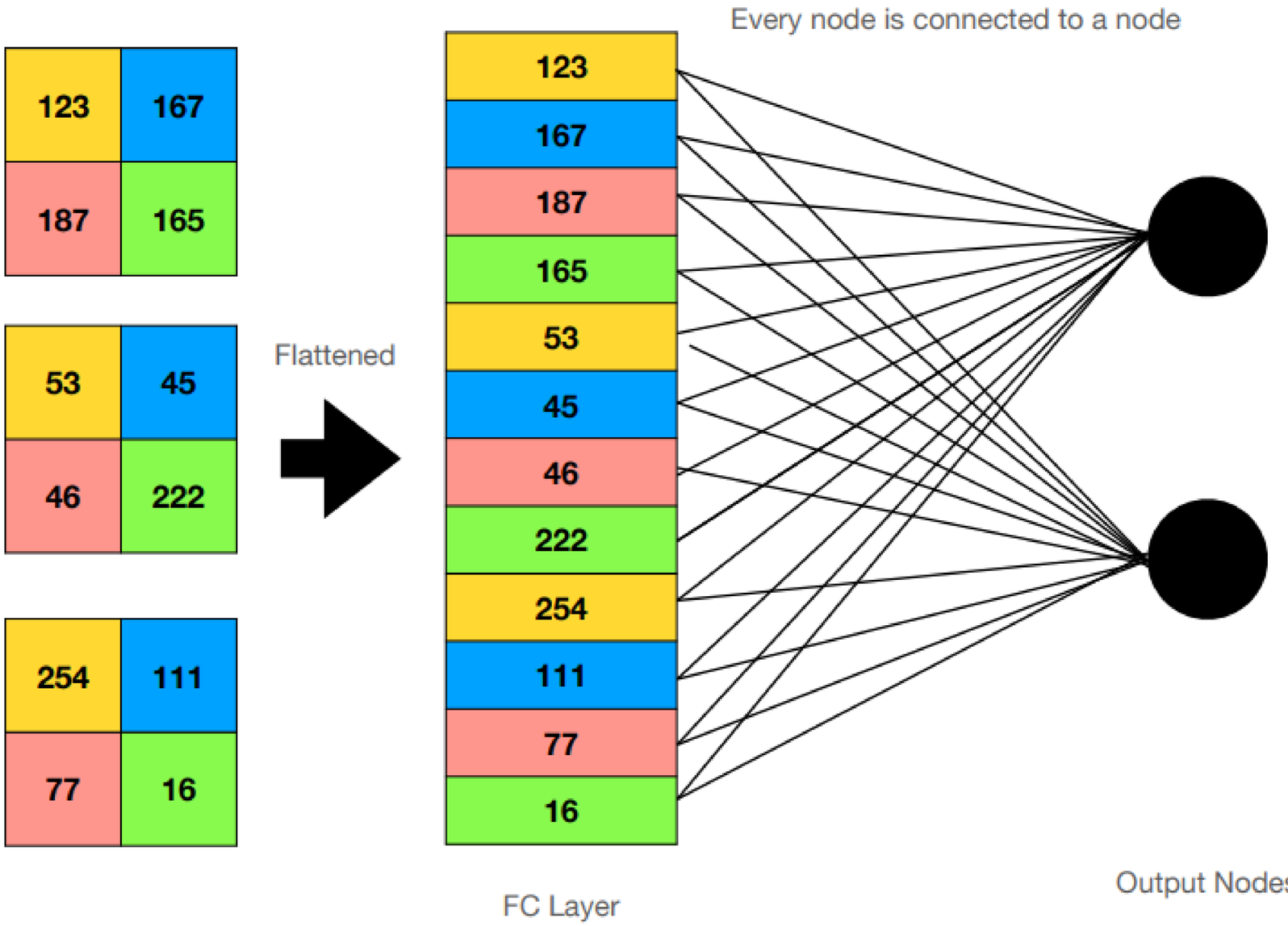
Developed by François Chollet

- Free and open-source Deep Learning API Interface for Python
- Initially supported several Deep Learning Backends (TensorFlow & Theano), as of Version 2.3+, its sole focus was TensorFlow.
- It also contained several implementations of NN layers and tools
- As for 2019, Keras is now bundled with TensorFlow2.0

A simple CNN


1x9216
64@24x24
32@26x26
24@16x16
1x128
1x10
1@28x28
Input Image
Conv_1
Conv_2
Max Pool
Flatten
FC

The Fully Connected Layer

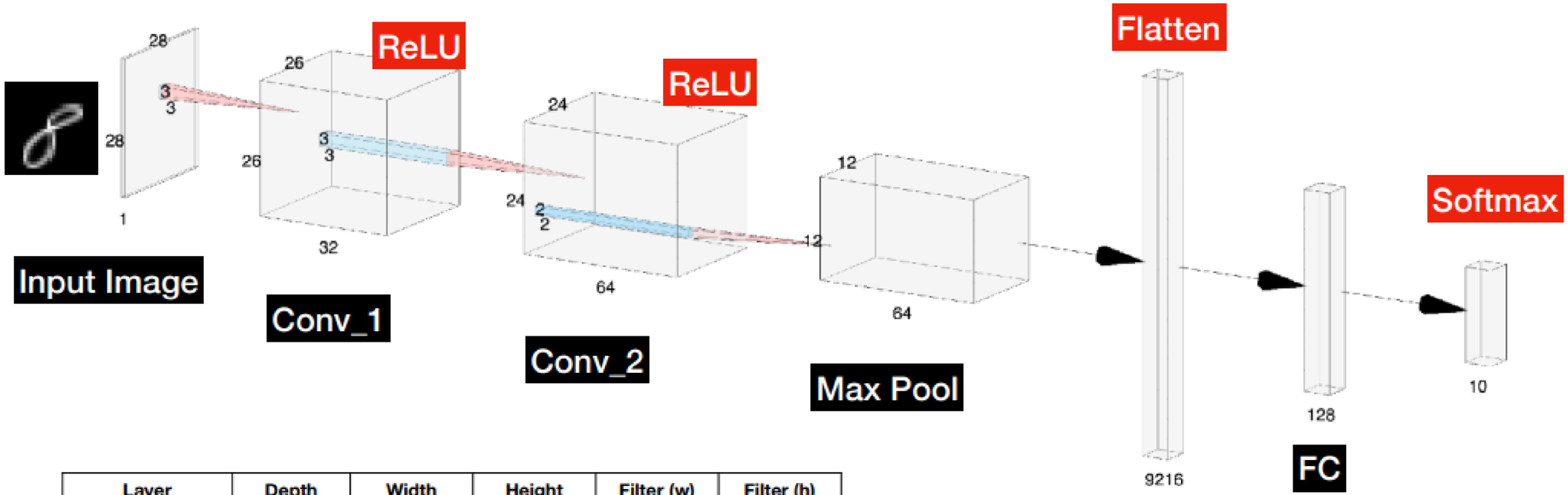


Softmax Layer

- We now need to produce probability outcomes for each class in Network.
- Softmax converts the Logs into Probabilities
- It takes the exponents of every output and the normalises each output by the
- sum of the exponents.
- It guarantees a well behaved distribution i.e. all sum to 1 and no values are
- zero.

$$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j))}$$

Another representation



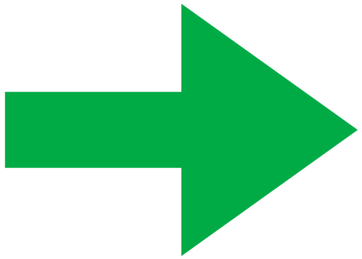
Layer	Depth	Width	Height	Filter (w)	Filter (h)
Input	1	28	28		
Conv_1	32	26	26	3	3
Conv_2	64	24	24	3	3
Max Pool	64	12	12	2	2
Flatten	9216	1	1		
Fully Connected	128	1	1		
Output	10	1	1		

What happens during training?

- **Initialise random weights** values for our trainable parameters
- **Forward propagate** an image or batch of images through our network
- Calculate the **total error**
- Use **Back Propagation** to update our gradients (weights) via Gradient Descent
- **Propagate more images** (or batch) and update weights, until all images have been propagated (one epoch)
- **Repeat a few more epochs** (i.e. passing all image batches through our Network) until our loss reaches satisfactory values

We need to Learn from our Results

- How correct are our results?
- We need a way tell the model it needs to do better



Loss Function

- How bad are the probabilities we predicted?
- How do we quantify the degree our prediction is off by?

Cross Entropy Loss

- Cross Entropy Loss uses two distributions, our ground truth distribution p(x) and q(x) our predicted distribution.

- Where y is the ground truth vector,y is the predicted distribution

Class	Predicted Probabilities	Ground Truth
0	0.3	0
1	0.6	1
2	0.1	0

- $L = -\, y \,.\, log(\hat{y})$
- $L = -\,(0 \times log(0.3) + 1 \times log(0.6) + 0 \times log(0.1)$
- $L = -\,(0 + 1 \times -0.222 + 0) = 0.222$

Back Propagation

This is what makes Neural Networks Trainable

Backpropagation is the optimization algorithm used to update the weights of a neural network during training. It involves a forward pass to make predictions, computation of the loss, a backward pass to calculate gradients, and weight updates to minimize the error by adjusting the weights in the opposite direction of the gradient.

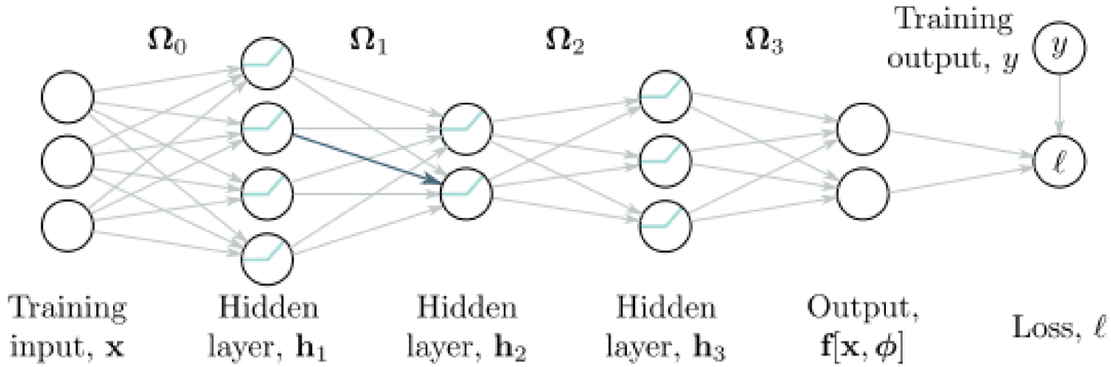
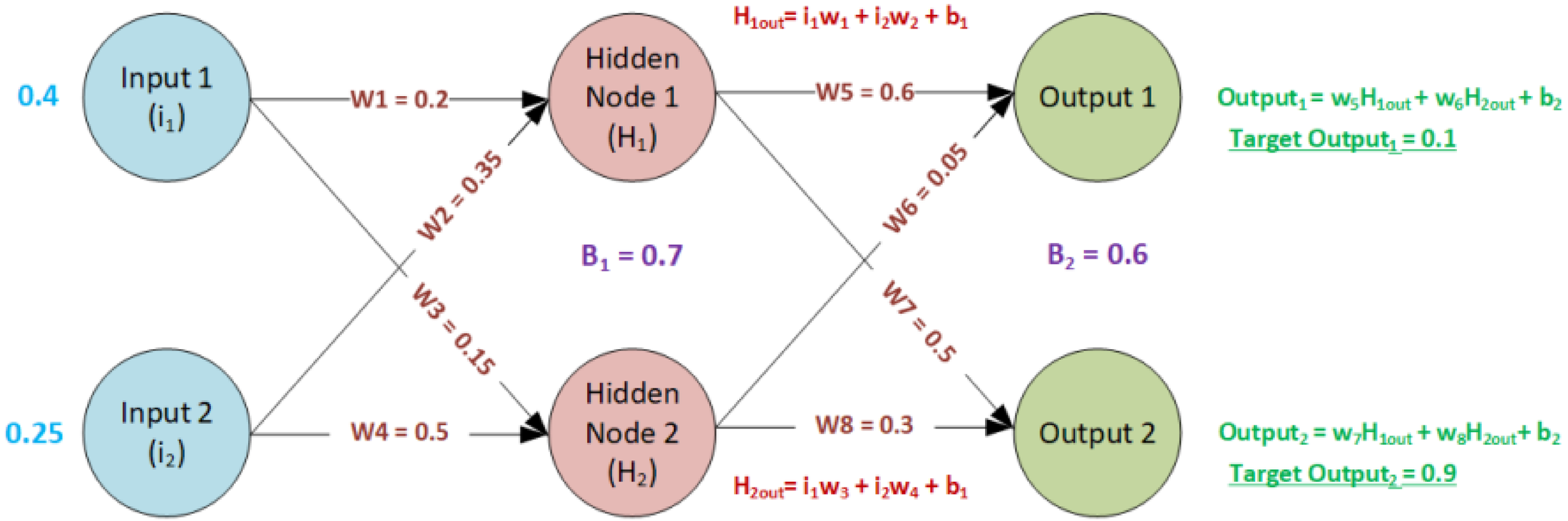


Figure 7.1 Backpropagation forward pass. The goal is to compute the derivatives of the loss ℓ with respect to each of the weights (arrows) and biases (not shown). In other words, we want to know how a small change to each parameter will affect the loss. Each weight multiplies the hidden unit at its source and contributes the result to the hidden unit at its destination. Consequently, the effects of any small change to the weight will be scaled by the activation of the source hidden unit. For example, the blue weight is applied to the second hidden unit at layer 1; if the activation of this unit doubles, then the effect of a small change to the blue weight will double too. Hence, to compute the derivatives of the weights, we need to calculate and store the activations at the hidden layers. This is known as the *forward pass* since it involves running the network equations sequentially.

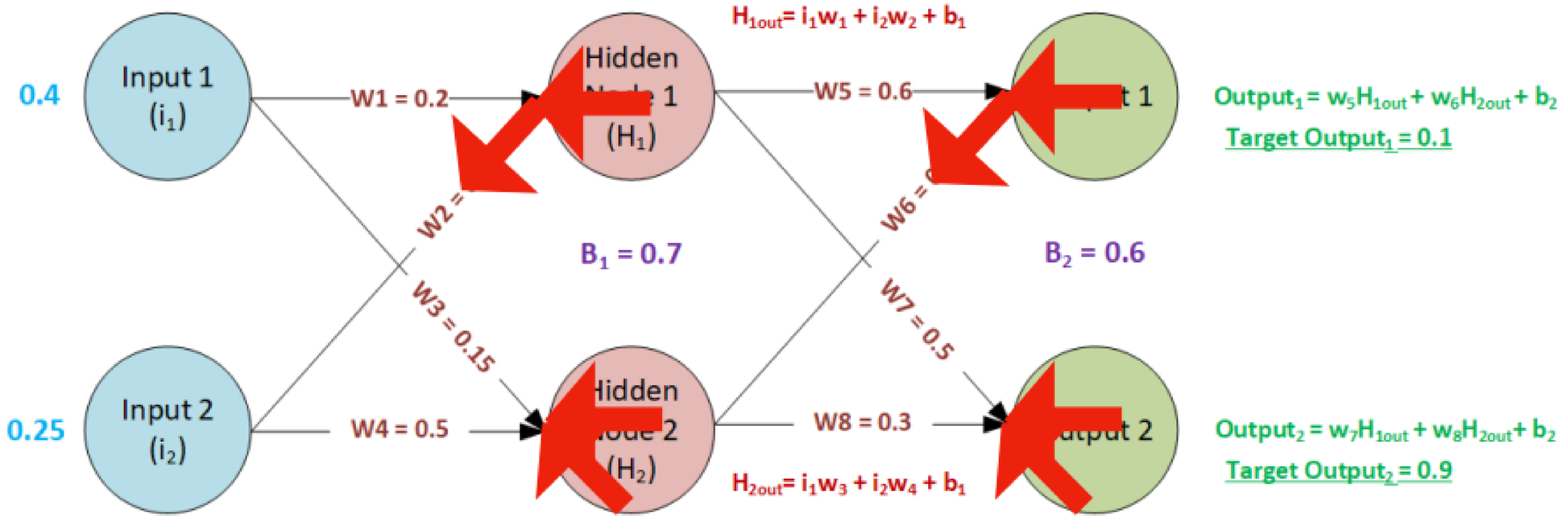
Back Propagation

This is what makes Neural Networks Trainable



Back Propagation

This is what makes Neural Networks Trainable



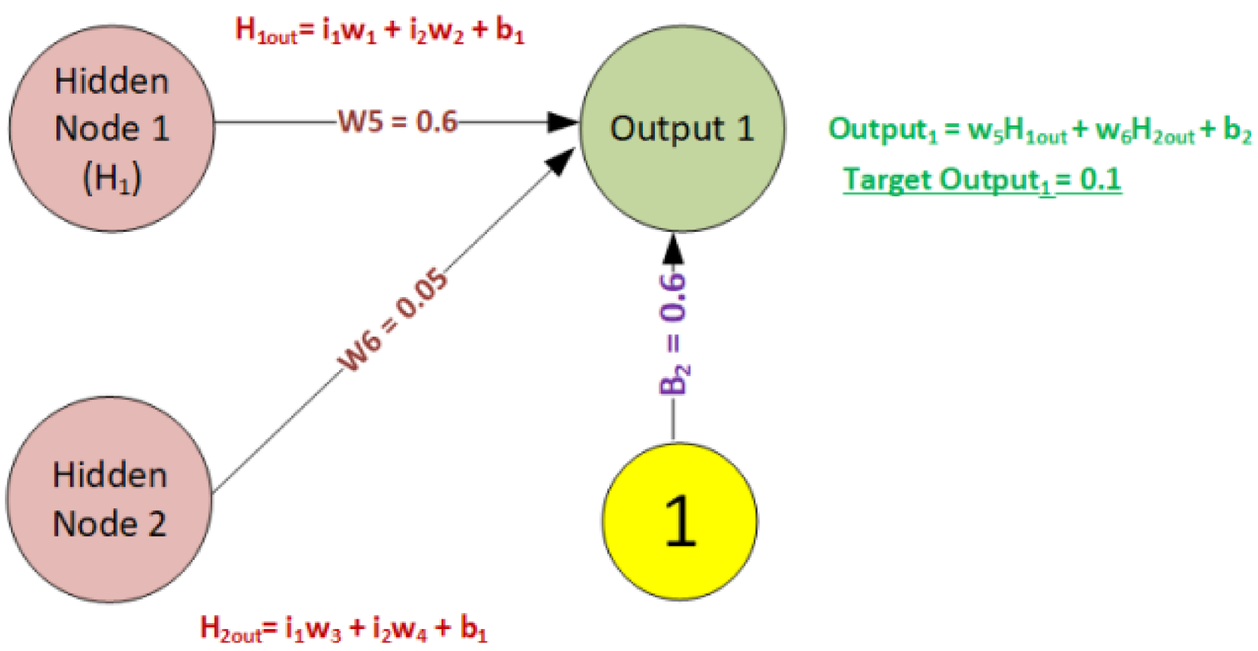
Back Propagation

This is what makes Neural Networks Trainable

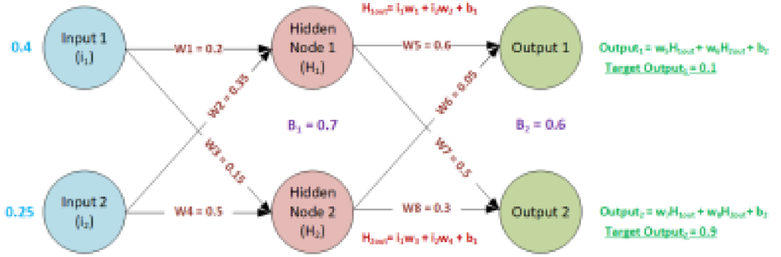
- **Chain Rule!**
- If we have two functions $y = f(u)$ and $u = g(x)$ then the derivative of y is:
  - $$\frac{dy}{dx} = \frac{dy}{du} \times \frac{du}{dx}$$

Back Propagation

This is what makes Neural Networks Trainable

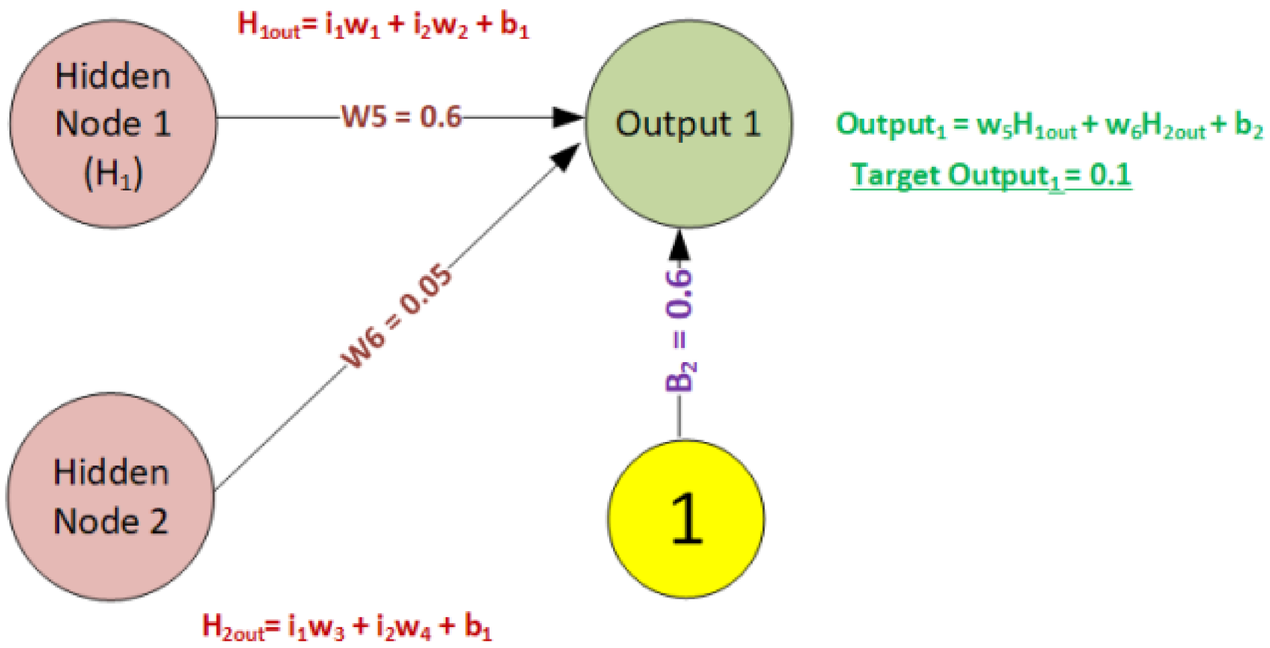


- We want to know how much changing W_5 changes the **Total Error**.
- That is given by:
  - $\frac{dE_T}{dW_5}$
  - Where E_T is the sum of the error from Outputs 1 and 2 (see below)



Back Propagation

This is what makes Neural Networks Trainable



New $W_5 = -\lambda \times \frac{dE_T}{dW_5}$

- Note we introduced a new parameter λ
- λ is our learning rate
- It controls how a big a jump (positive or negative) we take when updating W_5
- Large learning rates train faster, but can get stuck in a Global Minimum
- Small learning rates train more slowly

Move on to the practice

Thank you for your attention!