2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Deep Learning for Design and Verification Engineers

John Aynsley

DOULOS

Deep Learning for Design and Verification Engineers

What is Deep Learning?

Neural Networks

How a Network Learns

Getting Started

Machine Learning – Definition

"Giving computers the ability to learn without being explicitly programmed"

Machine Learning Algorithms

Classification

Regression

Neural networks

Support vector machines

Dimensionality reduction

Bayesian statistics

Hebbian learning

Markov models

Decision tree learning

Random forests

Reinforcement learning

Evolutionary algorithms

Used in statistics and data science

Deep Learning

Deep learning is very simple algorithms applied very intensively to massive datasets and often applied end-to-end

This is not a definition, just an intuitive description!

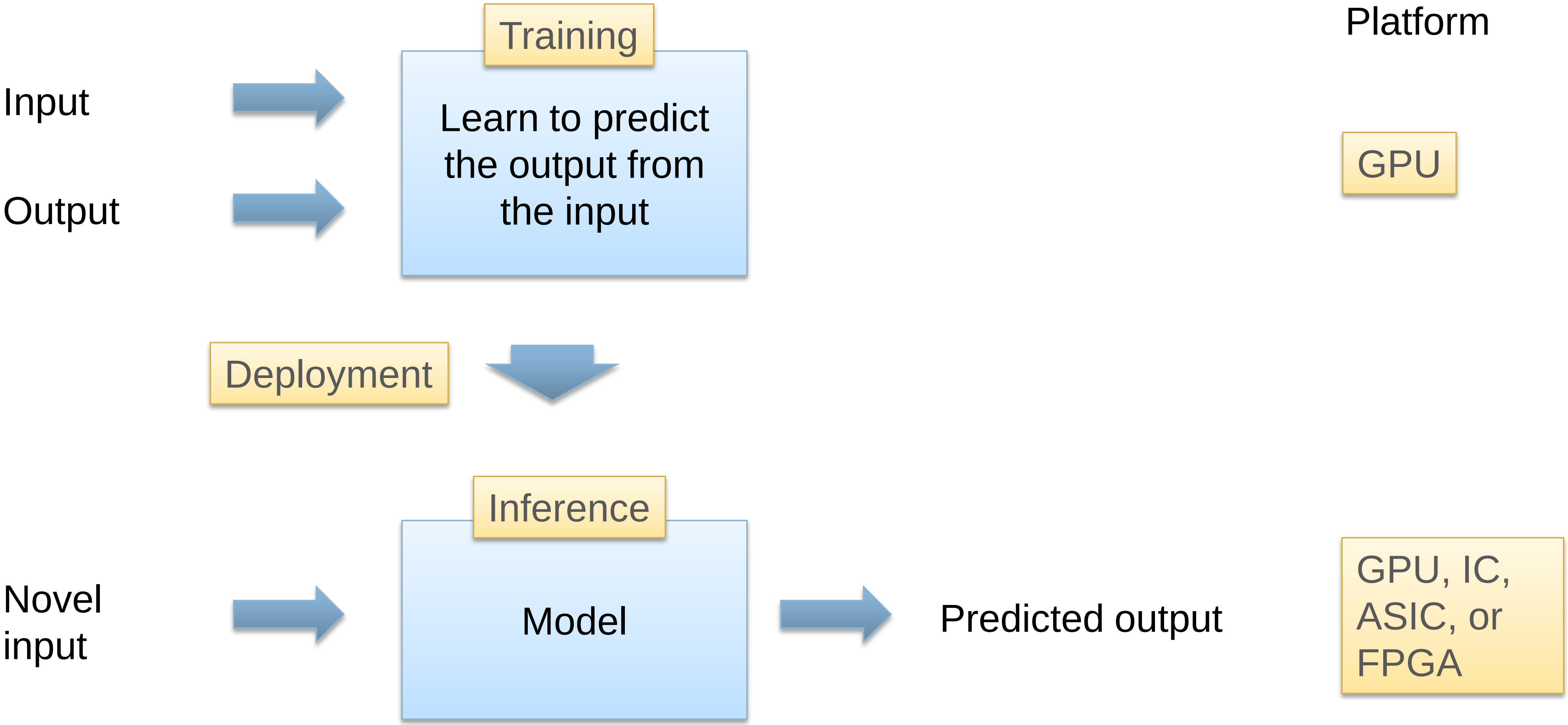
Machine Learning and Deep Learning

Machine learning is not new

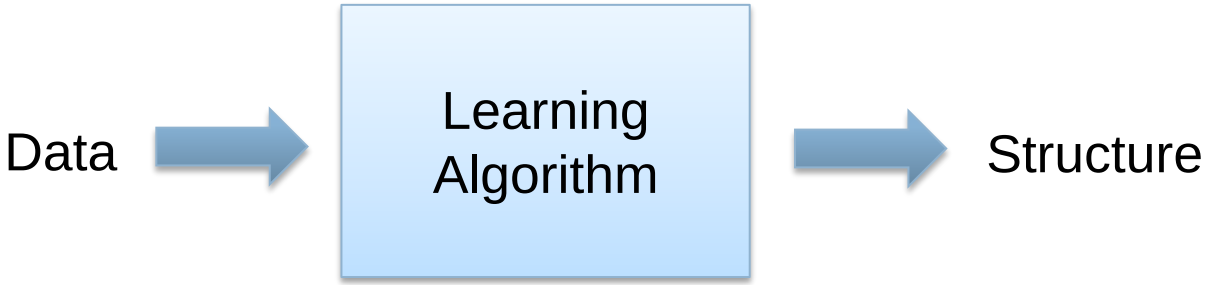
Deep learning is new

Deep learning is growing fast because it really works!

Supervised Learning



Unsupervised Learning



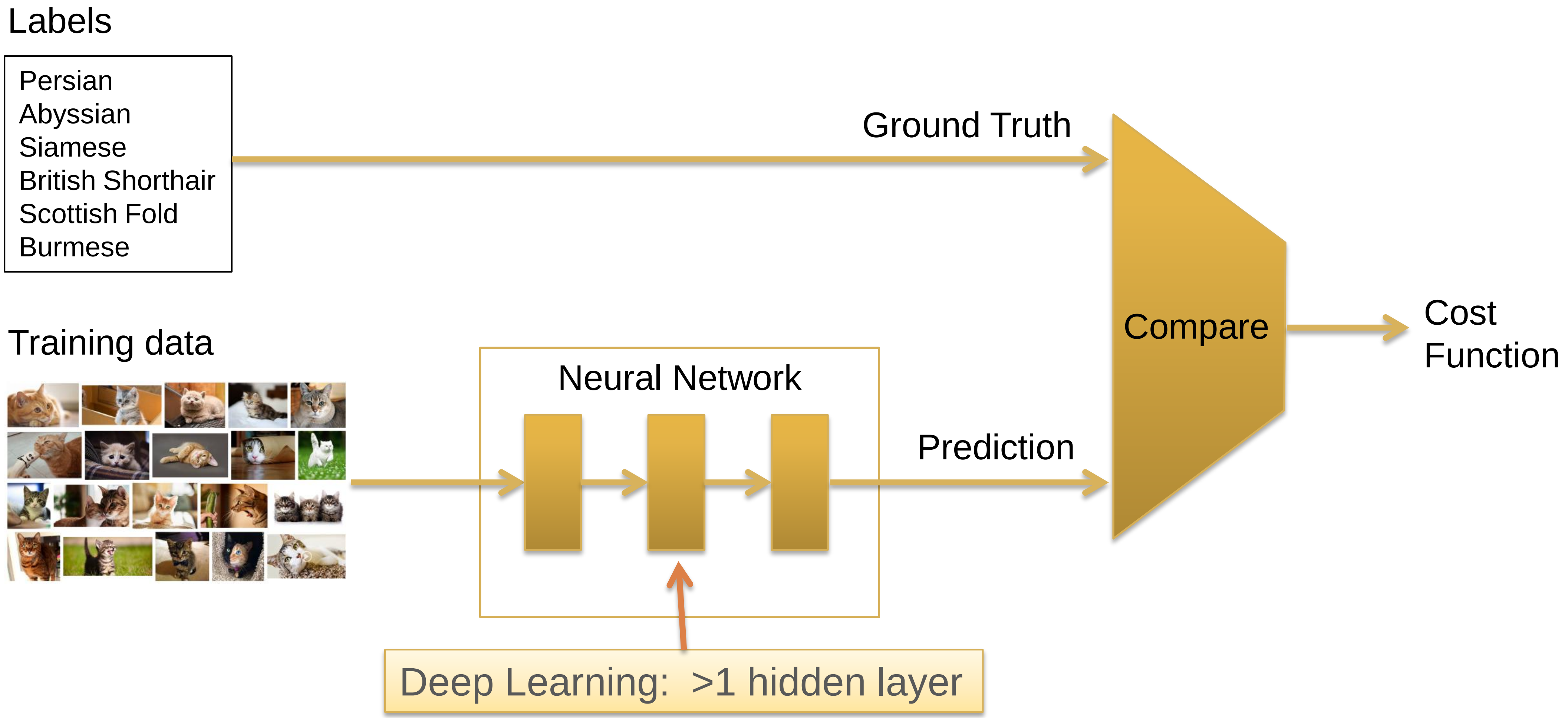
Examples

Partition the data into clusters based on similarity

Find hidden patterns in the data

Pick out anomalous data samples

Supervised Learning with a Neural Network



Why Neural Networks Now?

2012

Bigger datasets

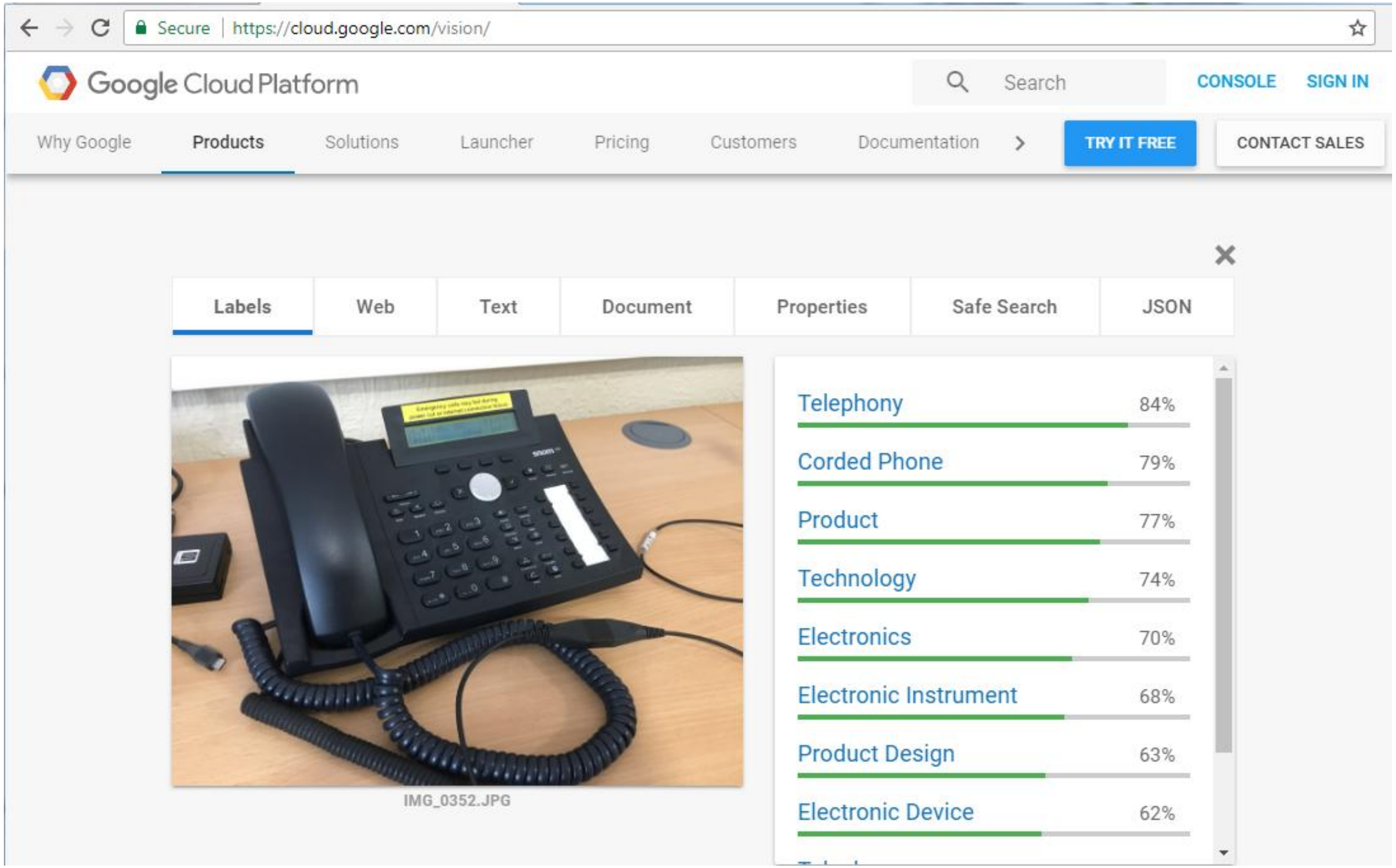
Faster computers

Since 2012

Improved neural network architectures

Neural networks often outperforming previous state-of-the-art

Image Classification



The ImageNet Challenge (ILSVRC)

ImageNet Large Scale Visual Recognition Challenge: 1.2M images in 1000 categories

Depends what you measure!

Year	Network	#Layers	Top-5 Error Rate
2012 winner	AlexNet (CNN)	8	15.3%
2013 winner	ZFNet (CNN)	8	14.8%
2014	VGGNet (CNN)	19	7.3%
2014 winner	GoogLeNet (Inception)	22	6.7%
2015 winner	ResNet (residual)	152	3.6%
2016 winner	CUImage (ensemble)	-	3.0%

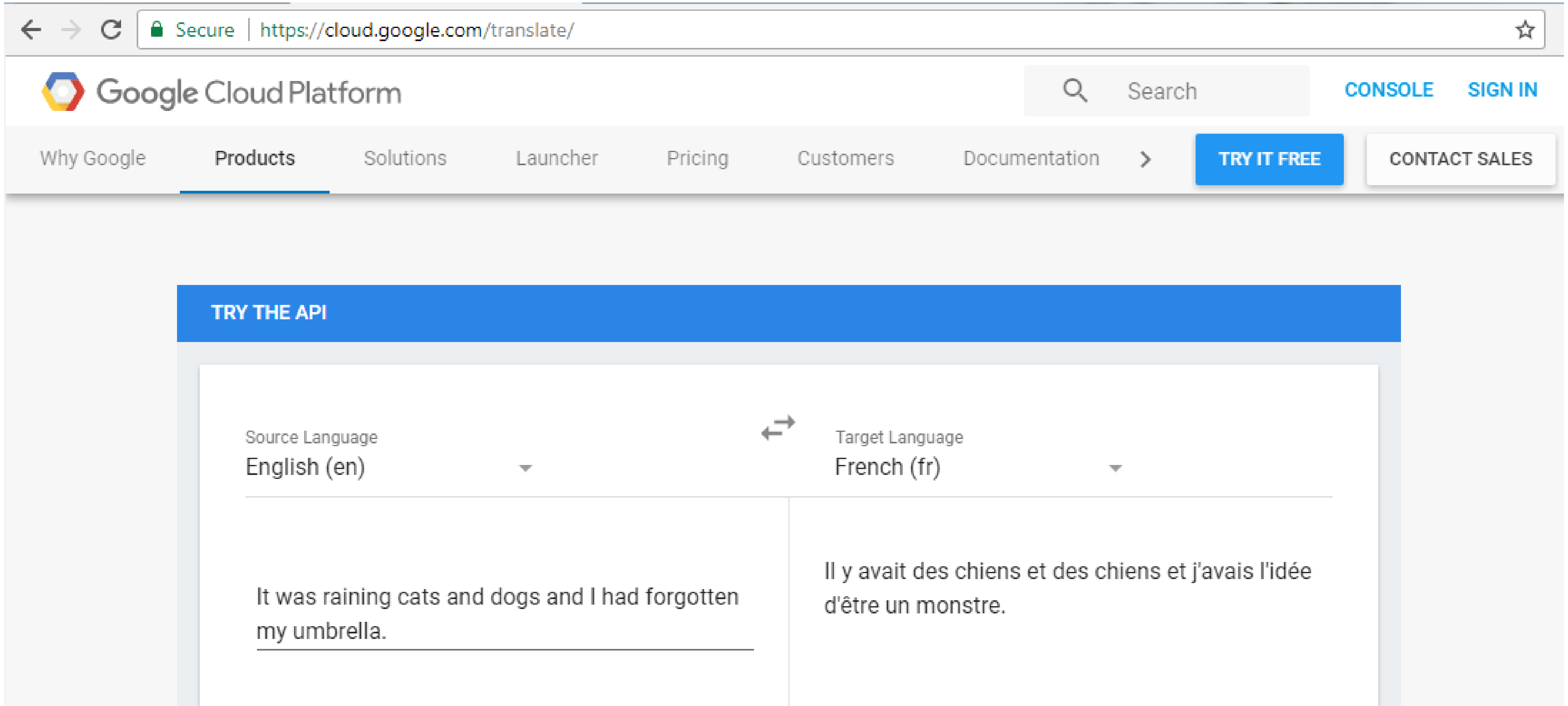
Dramatic improvement

Human error rate ~ 5%

3% bad labels

Training typically takes a few weeks on a few GPUs

Natural Language Translation



Other Exciting Applications

Image segmentation

Adding captions to images

Adding color to black-and-white images

Generating images that mimic other artists or styles

Spotting significant events in videos

Human pose estimation to analyze behavior

Handwriting recognition

Voice recognition and voice generation

Predicting patterns in natural phenomena

Playing games

Cool Stuff

http://www.yaronhadad.com/deep-learning-most-amazing-applications/

https://deepdreamgenerator.com

Deep Learning
for Design and Verification Engineers

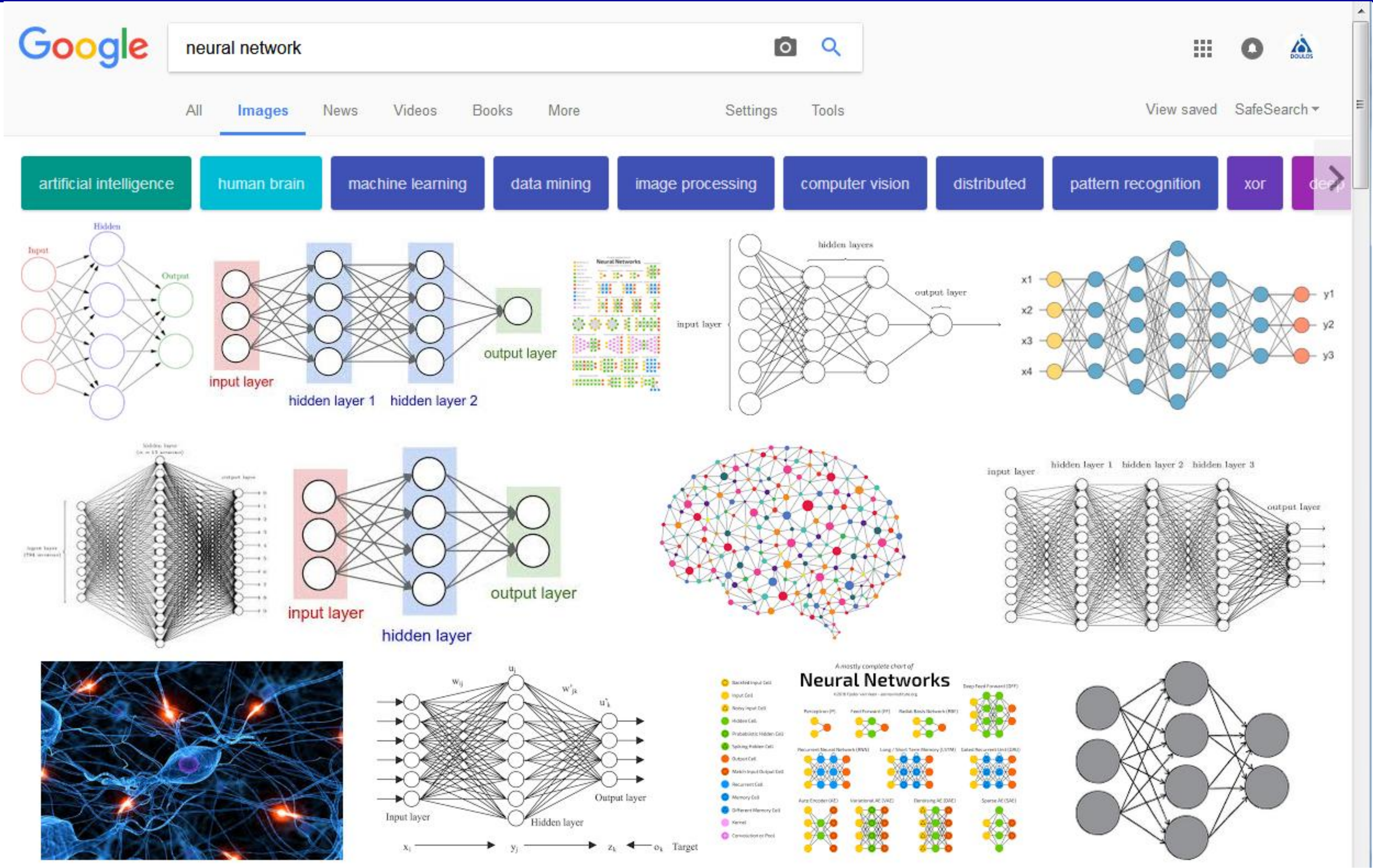
What is Deep Learning?

Neural Networks

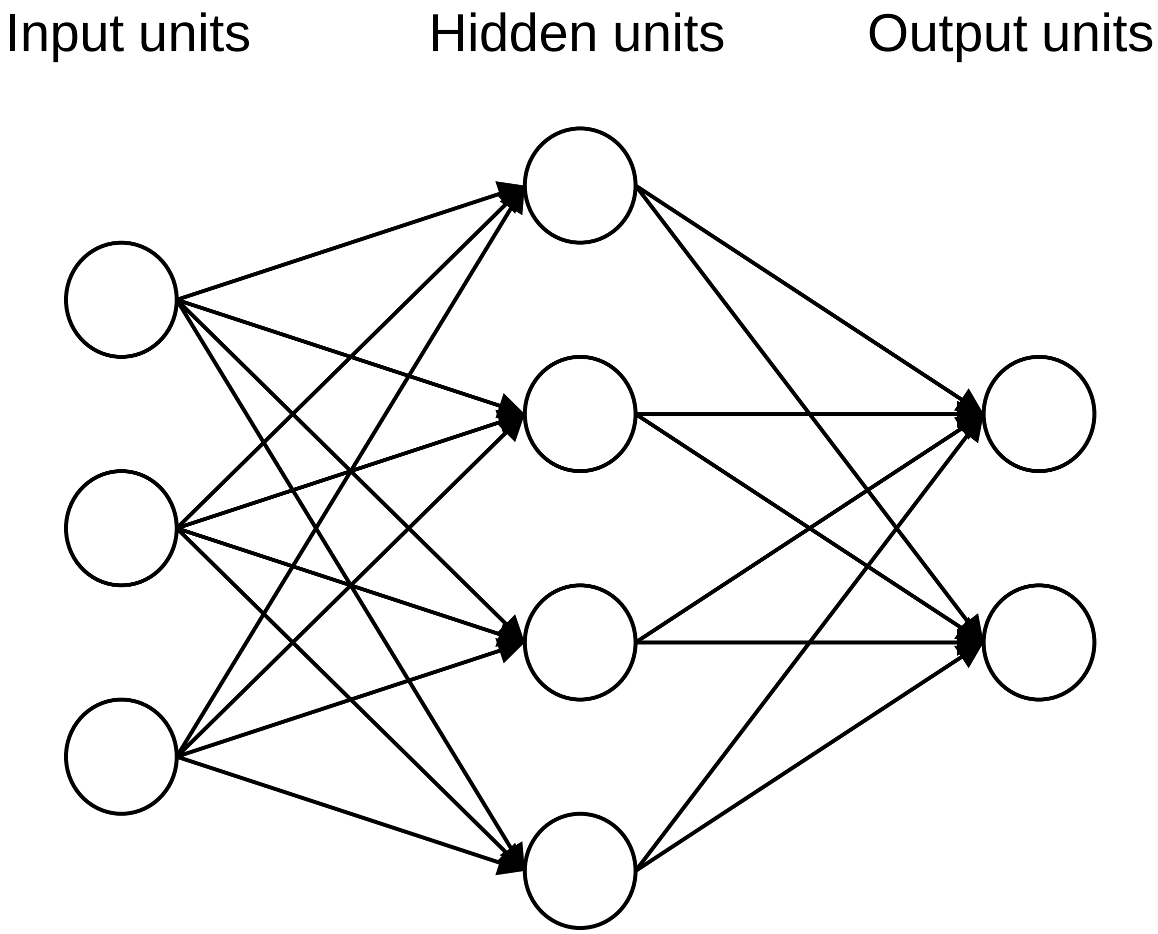
How a Network Learns

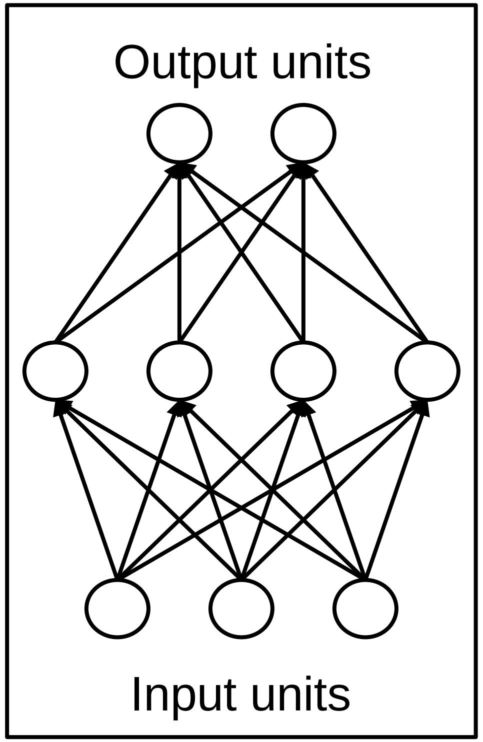
Getting Started

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES



A Neural Network?

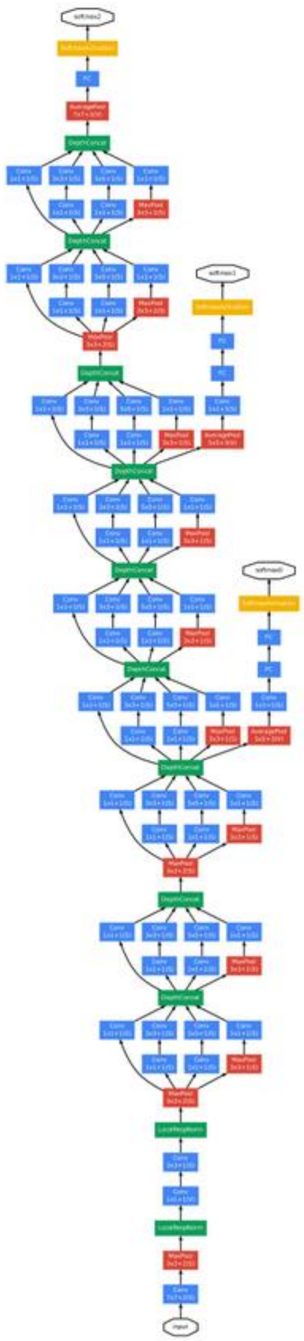




In the academic literature

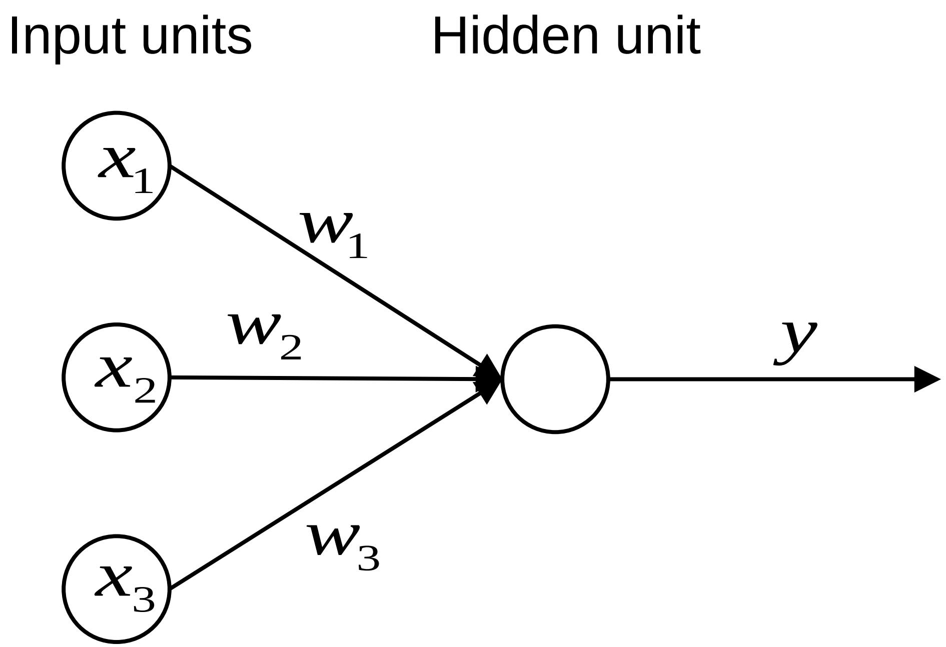
E.g. GoogLeNet

Inception network



Auxiliary outputs

An Artificial Neuron



$$u = \sum_{i=1}^{3} w_i x_i + b$$

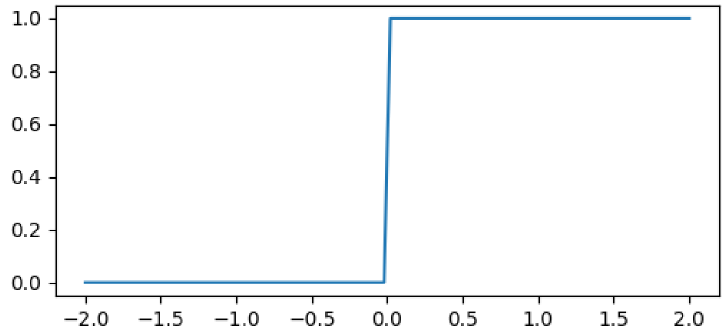
Linear function

$$y = activation(u)$$

Non-linear function

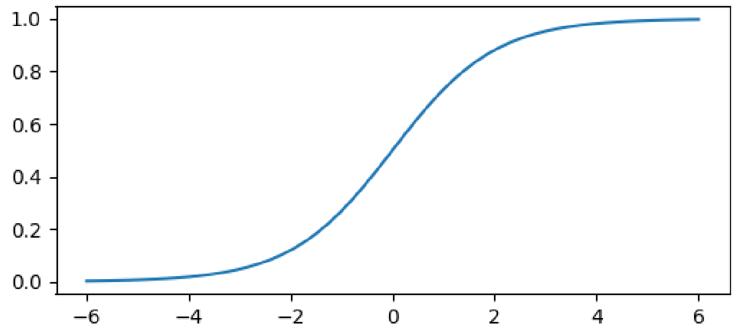
Common Activation Functions

Step



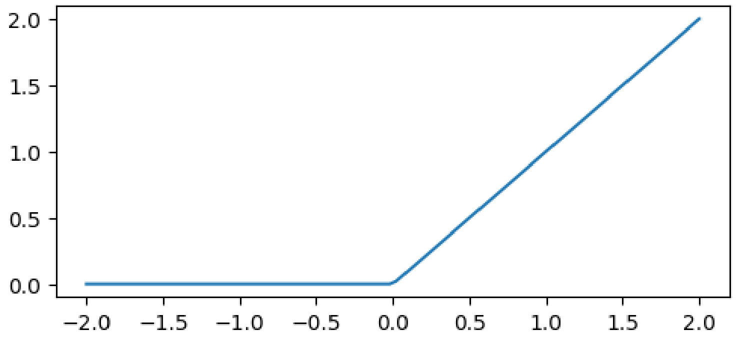
The Perceptron

Sigmoid
aka
logistic



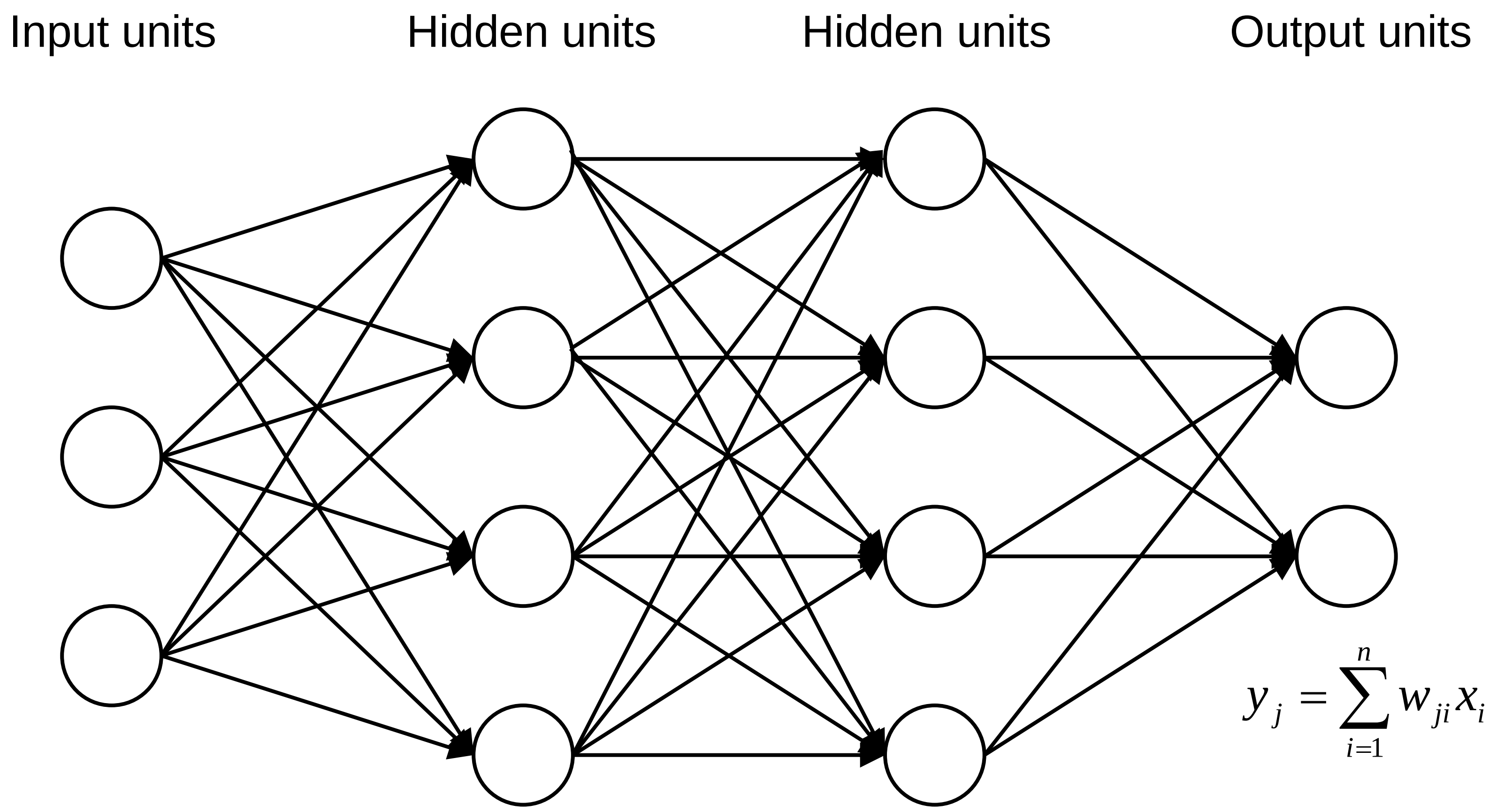
Early neural networks and still used today

ReLU



Popular today

A Deep Neural Network



$$y_j = \sum_{i=1}^{n} w_{ji} x_i + b_j$$

$$y_j = RELU\left(\sum_{i=1}^{3} w_{ji} x_i + b_j\right) \qquad y_j = RELU\left(\sum_{i=1}^{n} w_{ji} x_i + b_j\right)$$

Kinds of Neural Network

ANN – Artificial Neural Network

CNN – Convolutional Neural Network (e.g. image processing)

R-CNN – Regional CNN (image segmentation)

RNN – Recursive Neural Network (e.g. natural language processing)

GAN – Generative Adversarial Network

Deep Learning
for Design and Verification Engineers

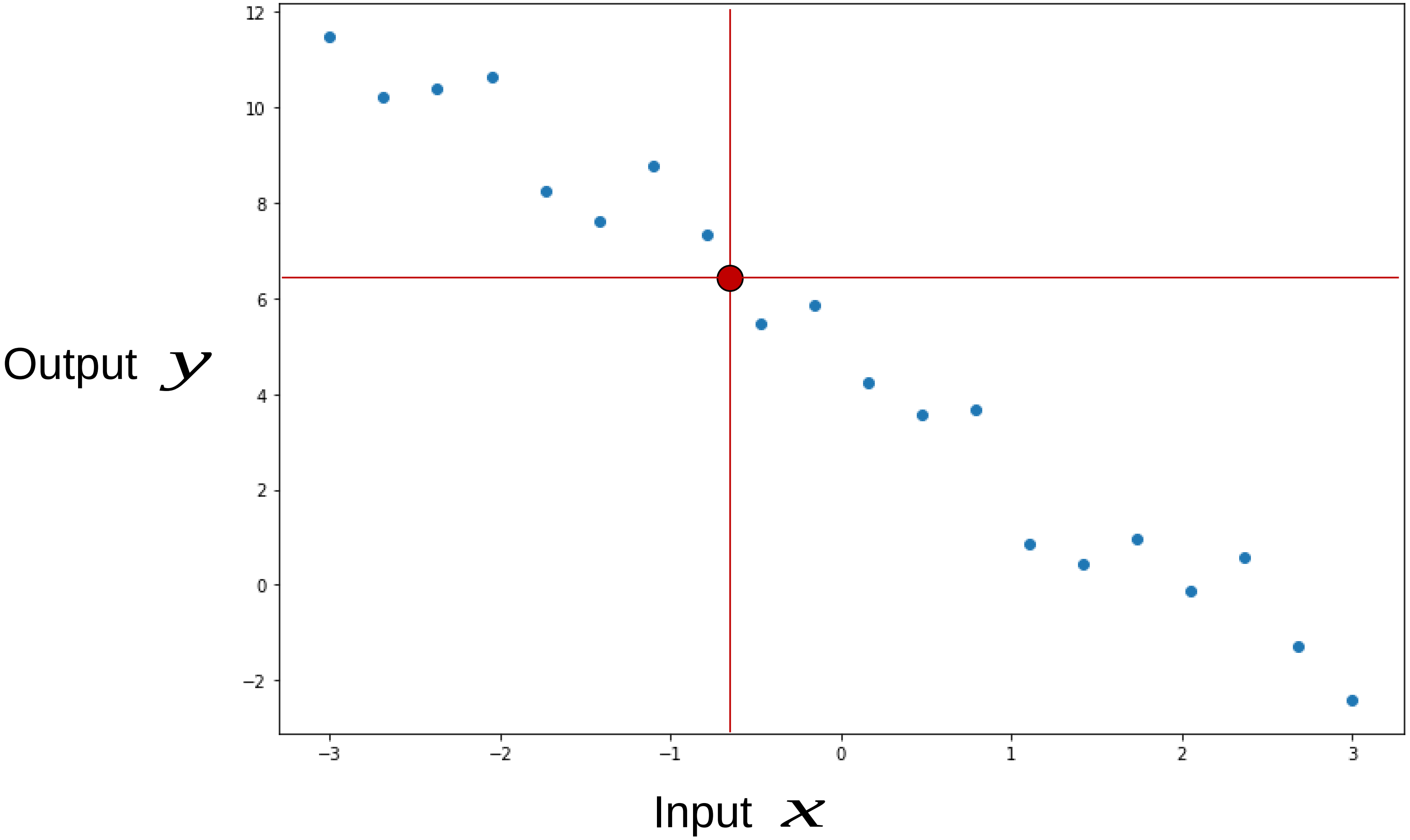
What is Deep Learning?

Neural Networks

How a Network Learns

Getting Started

Regression Task



Define a Hypothesis / Model / Network

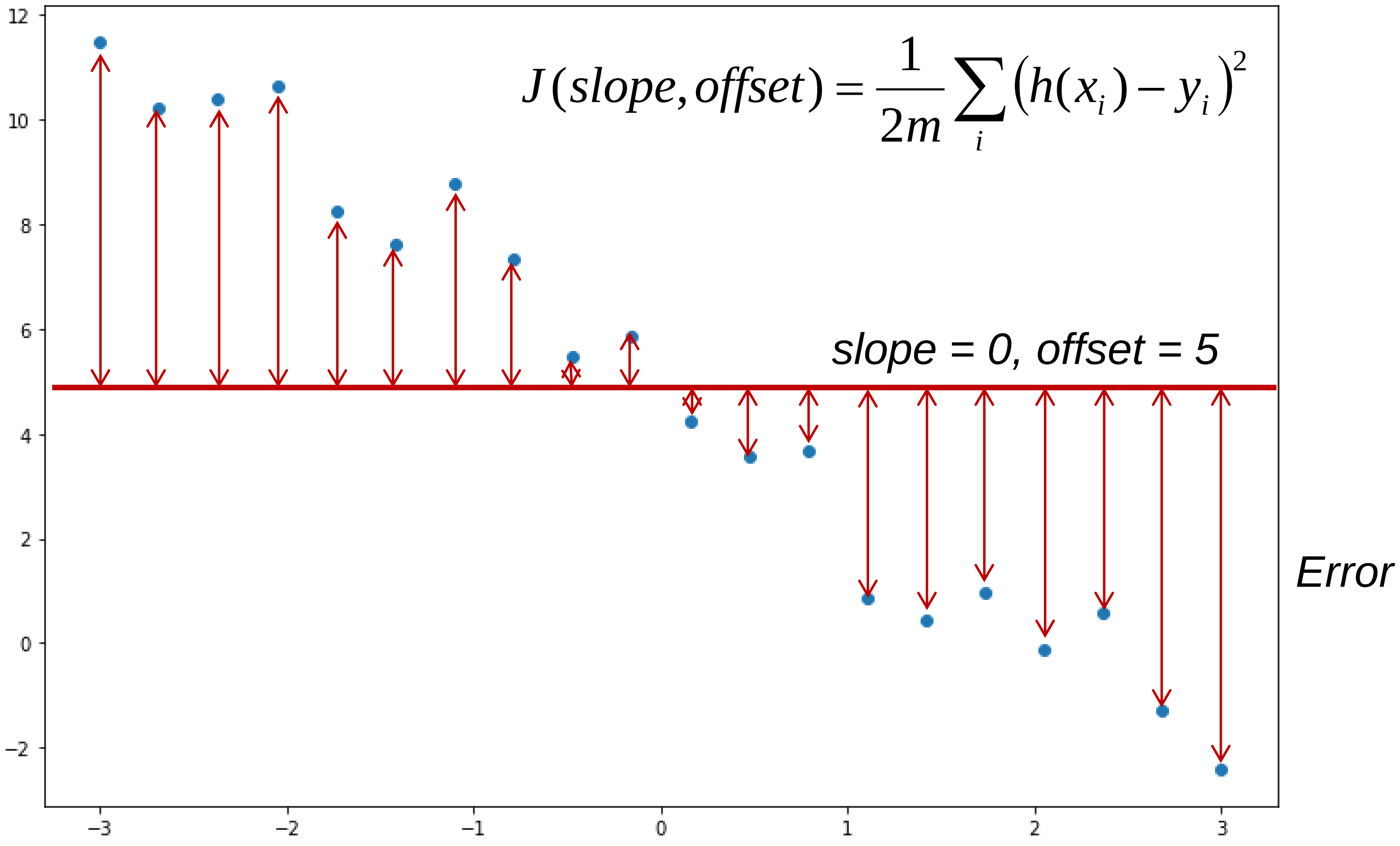
$$h_{slope,offset}(x) = slope \cdot x + offset$$

Input

Learnable parameters

```
def hypothesis(x, slope, offset):
    return slope * x + offset
```

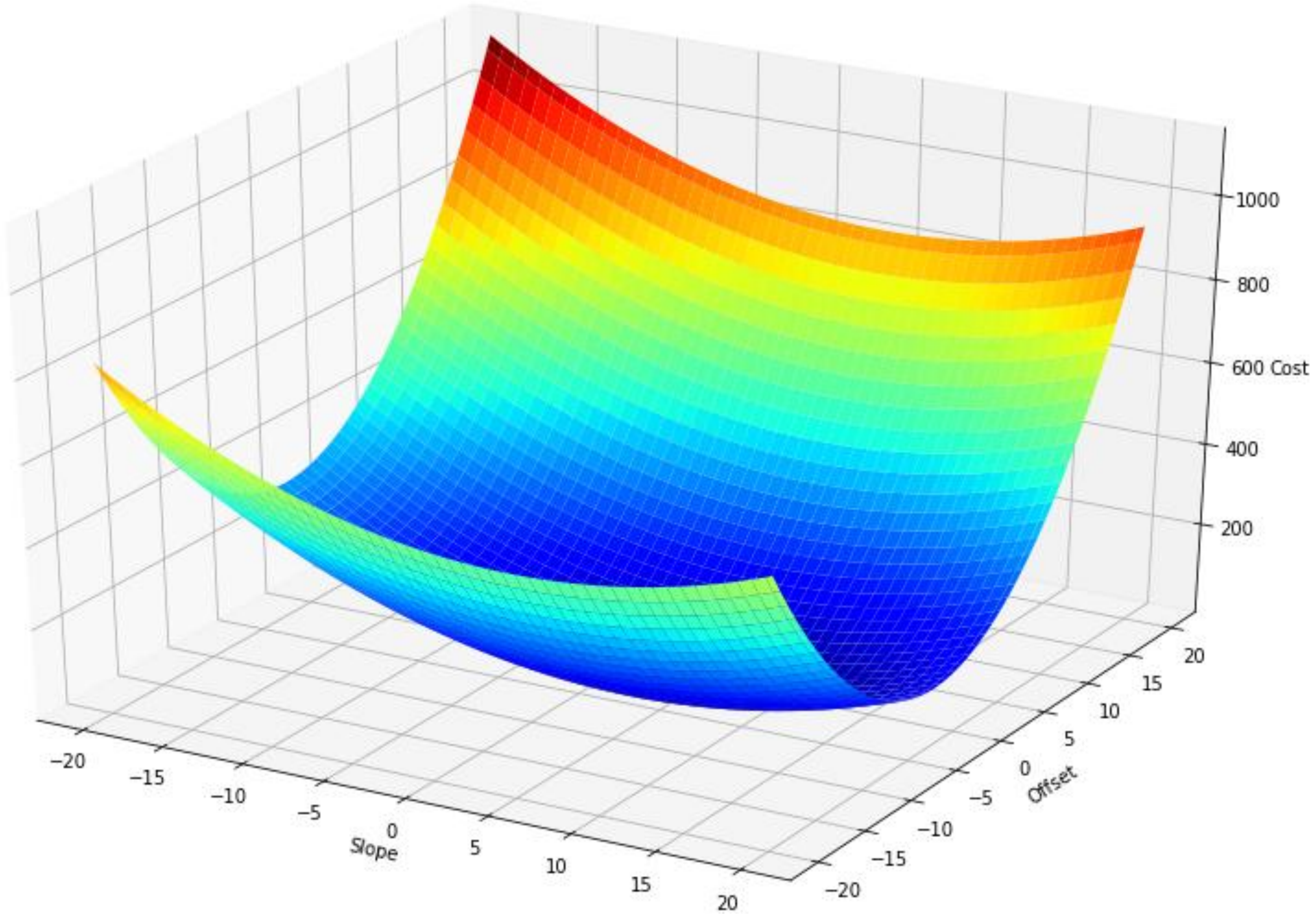
Cost Function



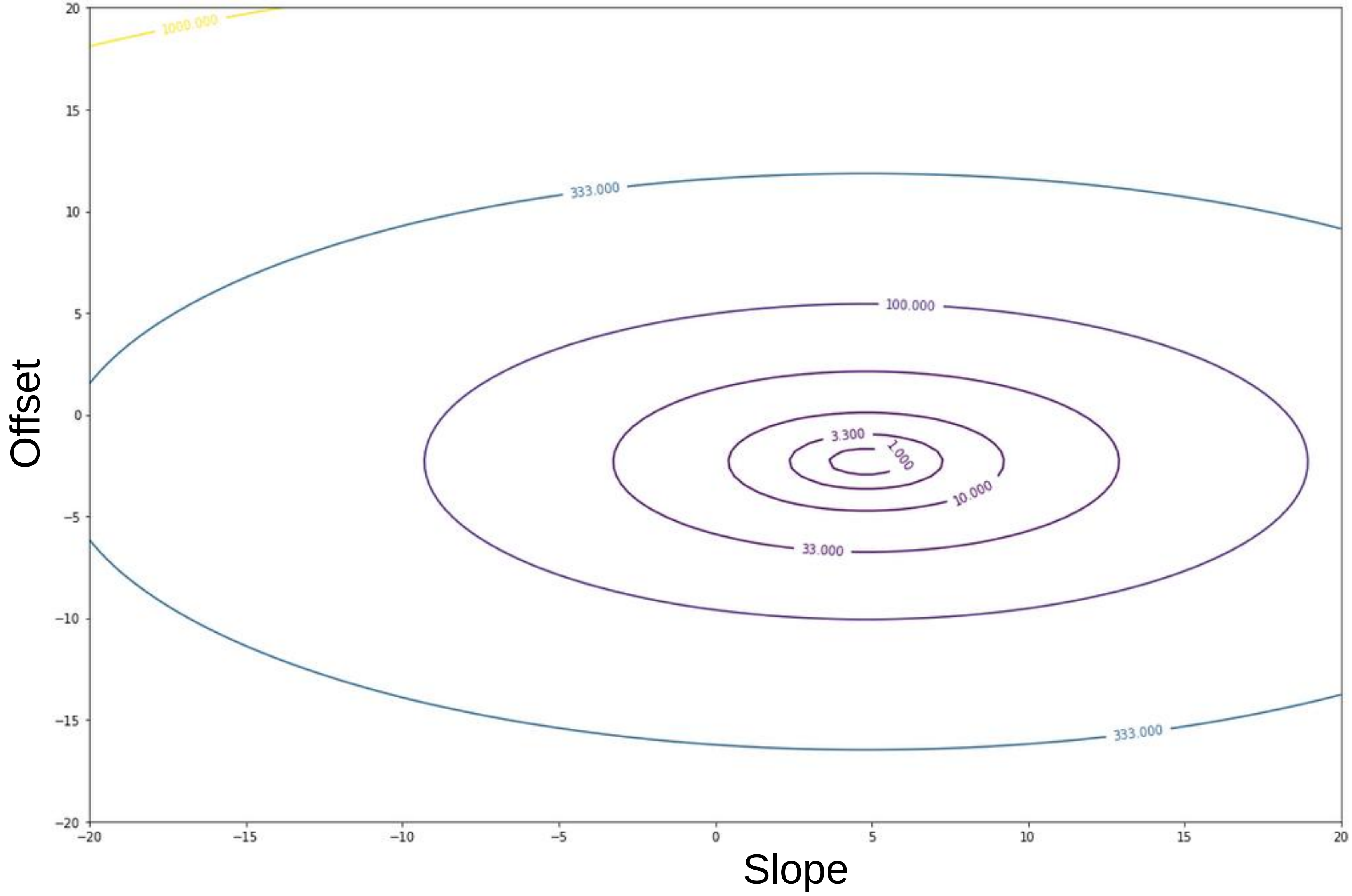
$$J(slope, offset) = \frac{1}{2m}\sum_i \left(h(x_i) - y_i\right)^2$$

```
def cost(x, slope, offset, y):
    return np.mean(np.square(hypothesis(x, slope, offset) - y)) / 2.0
```

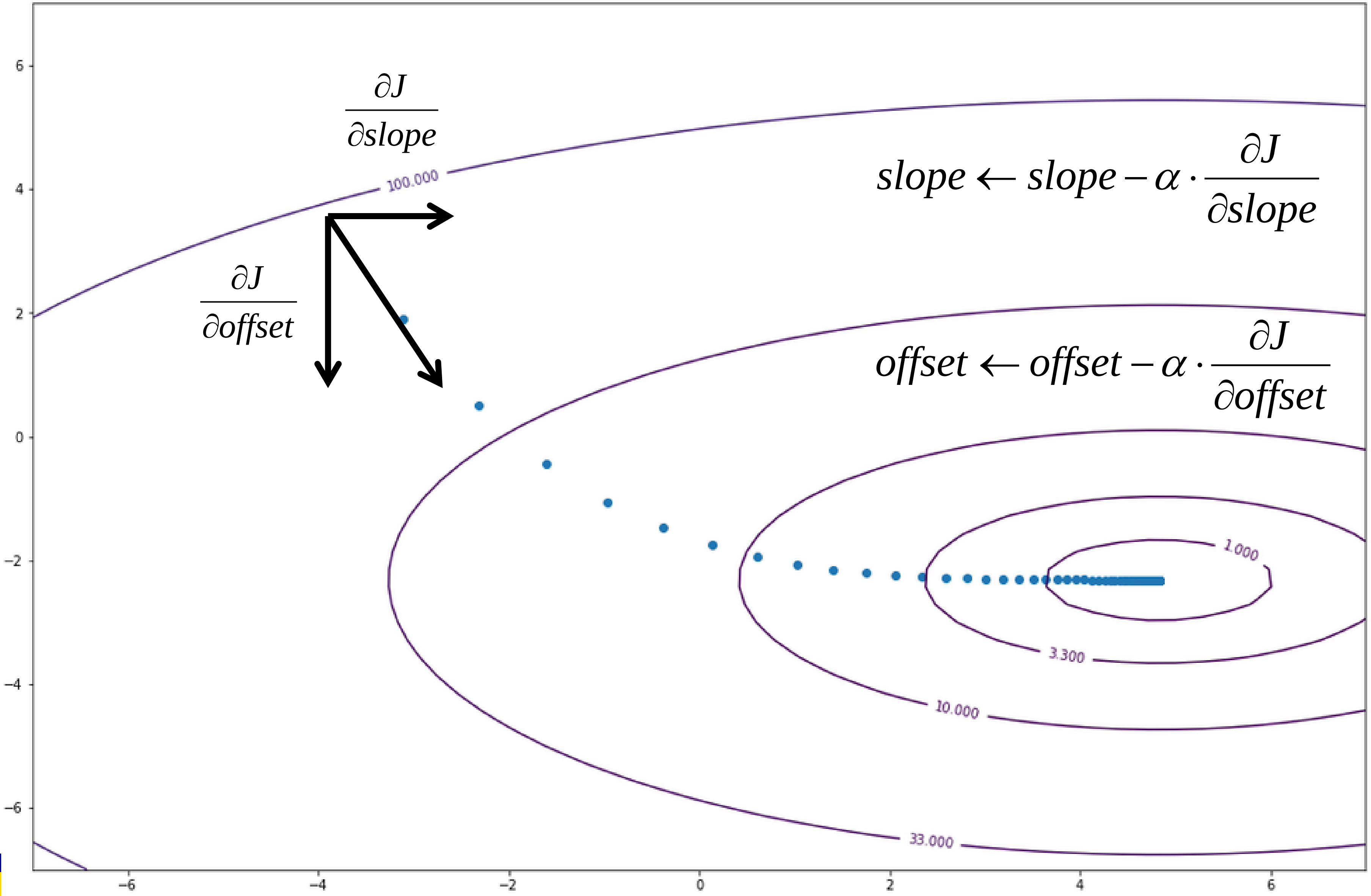
Cost as a Function of Slope and Offset



Contour Plot of Cost Function



Gradient Descent



$$\frac{\partial J}{\partial slope}$$

$$\frac{\partial J}{\partial offset}$$

$$slope \leftarrow slope - \alpha \cdot \frac{\partial J}{\partial slope}$$

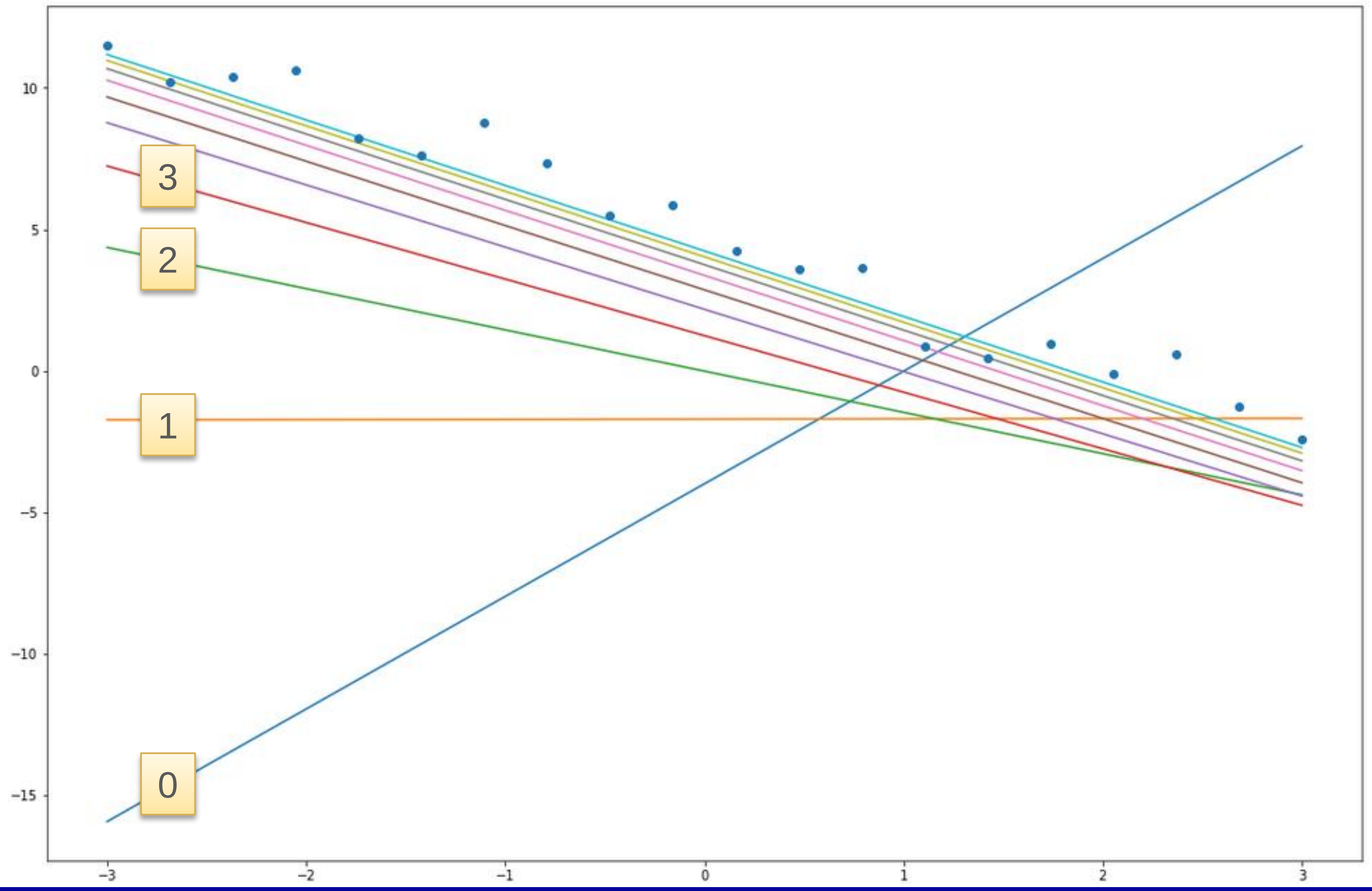
$$offset \leftarrow offset - \alpha \cdot \frac{\partial J}{\partial offset}$$

Gradient Descent Algorithm

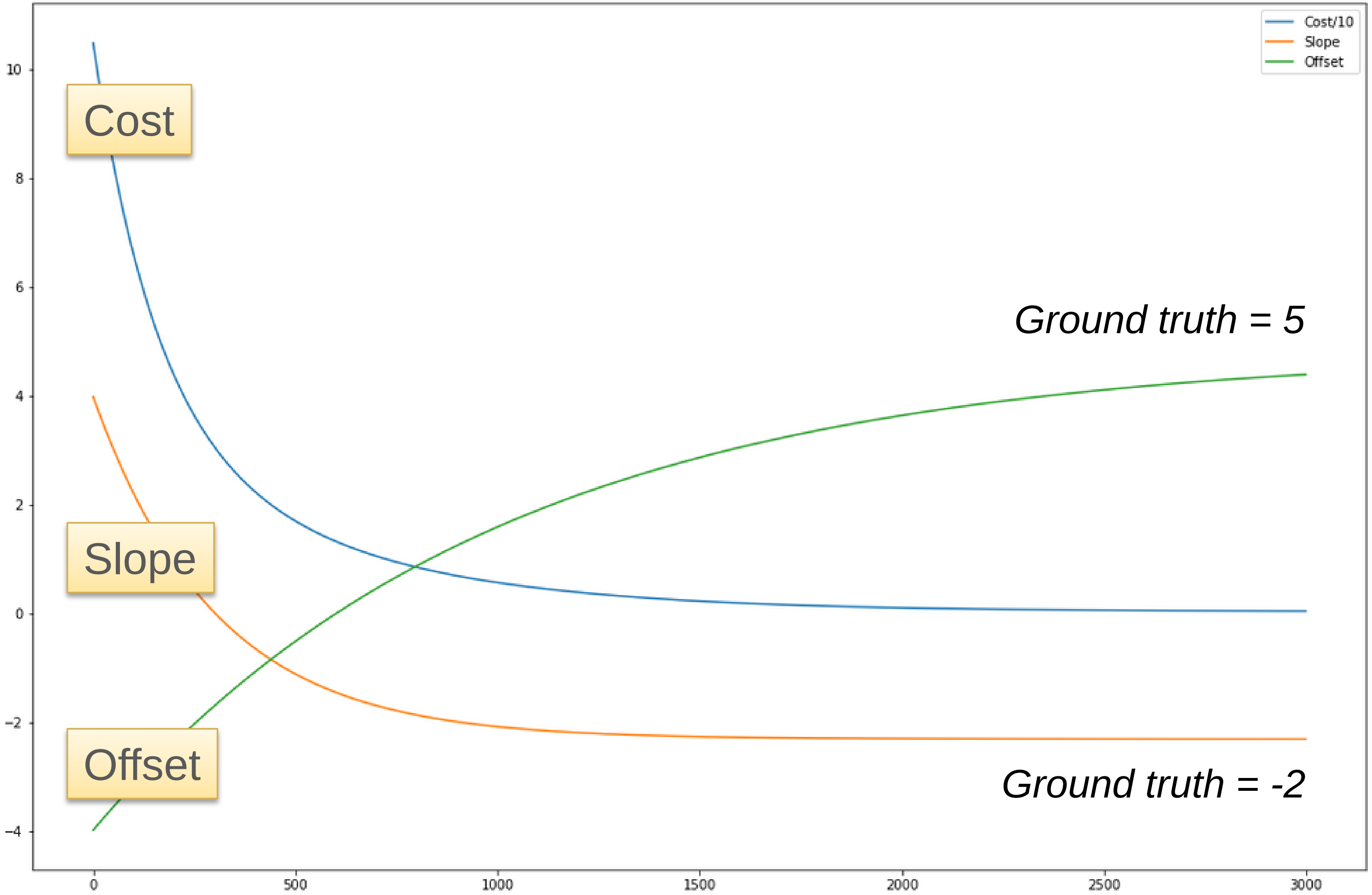
```
num_steps     = 3000
learning_rate = 0.001

slope  = 4.0
offset = -4.0

def derivative_wrt_slope():
    return np.mean((hypothesis(x, slope, offset) - y) * x)

def derivative_wrt_offset():
    return np.mean((hypothesis(x, slope, offset) - y))

for step in range(num_steps):
    new_slope  = slope  - learning_rate * derivative_wrt_slope()
    new_offset = offset - learning_rate * derivative_wrt_offset()
    slope  = new_slope
    offset = new_offset
```

Converging on the Minimum

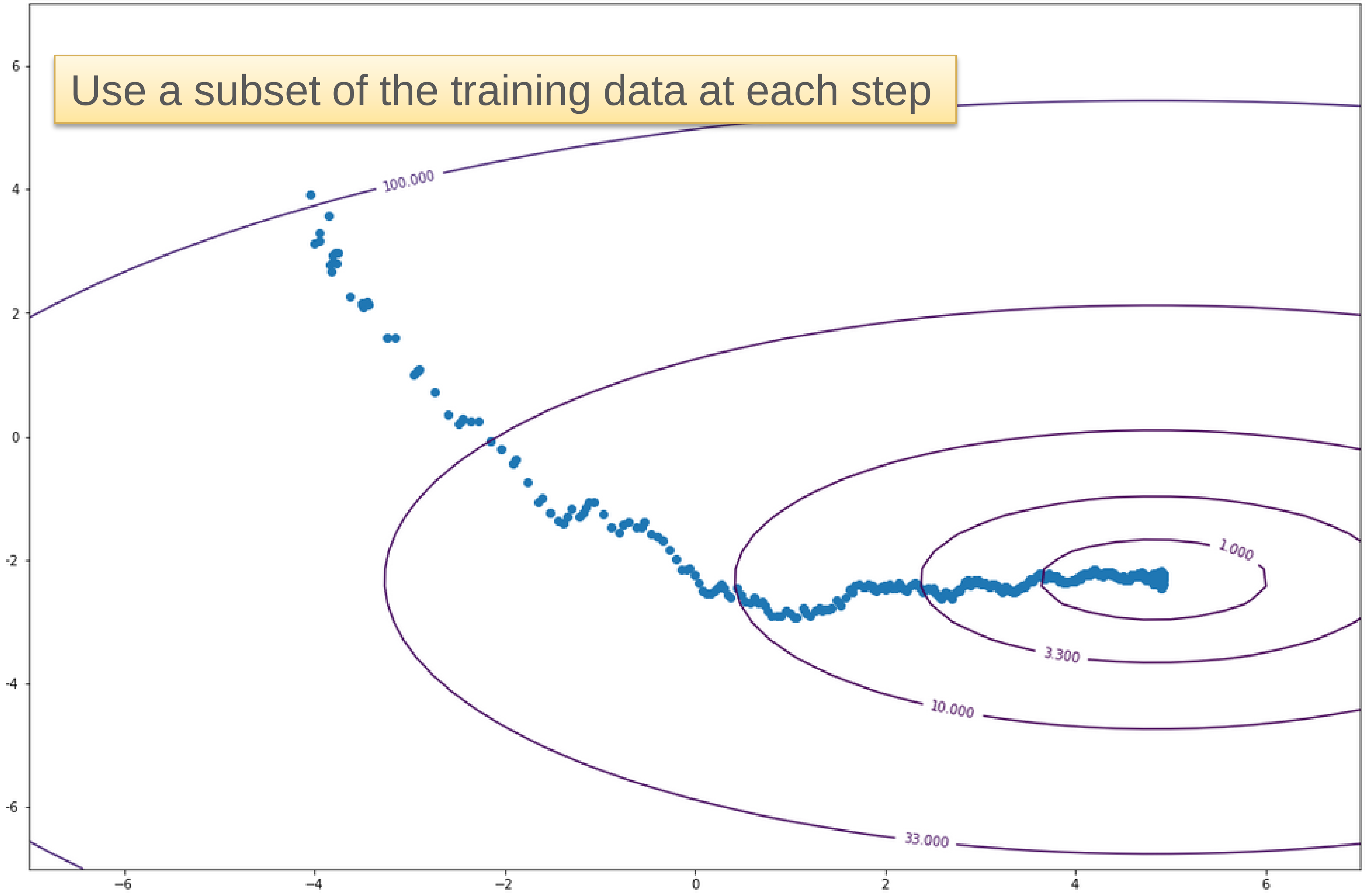
Final slope = -2.31499114425 offset = 4.38980555415



Cost, Slope, Offset against Step

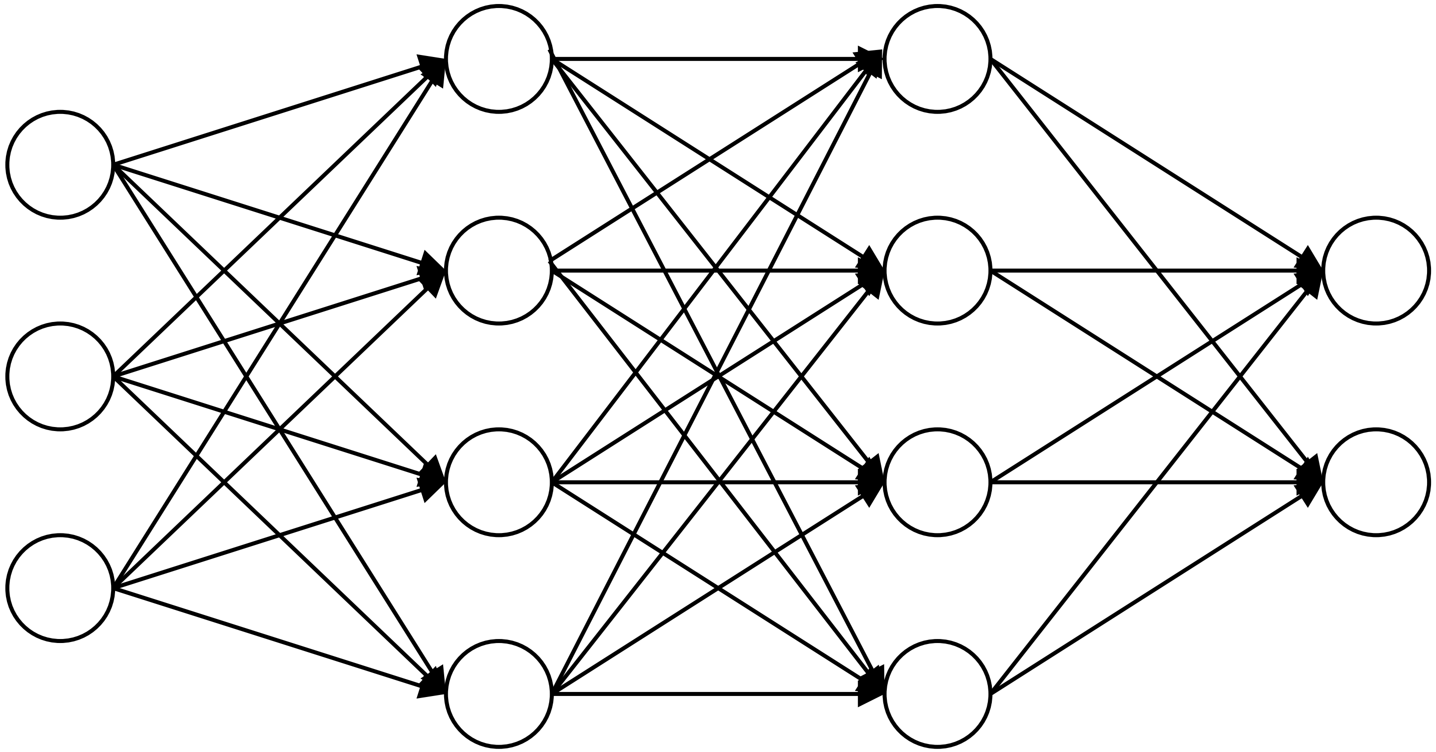


Stochastic Gradient Descent



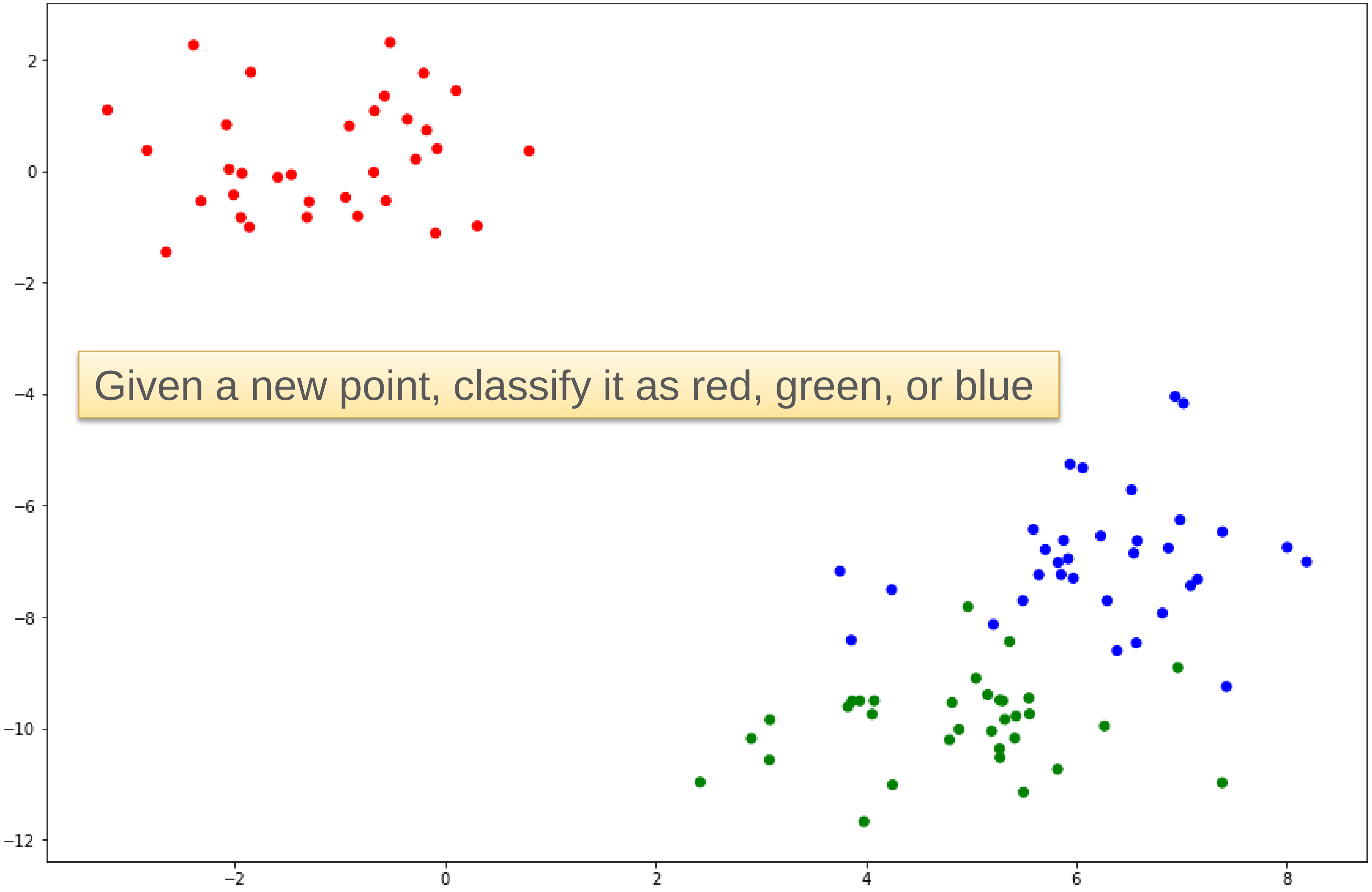
Forward and Back-Propagation

Forward propagation calculates weighted sums and activation function



Back propagation calculates gradients

Classification Task



One-Hot Labels

Category:	Red	Green	Blue	
Numeric label (single output):	1	2	3	Would introduce a bias
One-hot label (3 outputs):	$\begin{pmatrix}1\\0\\0\end{pmatrix}$	$\begin{pmatrix}0\\1\\0\end{pmatrix}$	$\begin{pmatrix}0\\0\\1\end{pmatrix}$	No bias

The Hypothesis or Model

Output

Input

$$\vec{y} = W\vec{x} + \vec{b}$$

$$y_j = \sum_{i=1}^{3} w_{ji} x_i + b_j$$

Trainable parameters

Weights

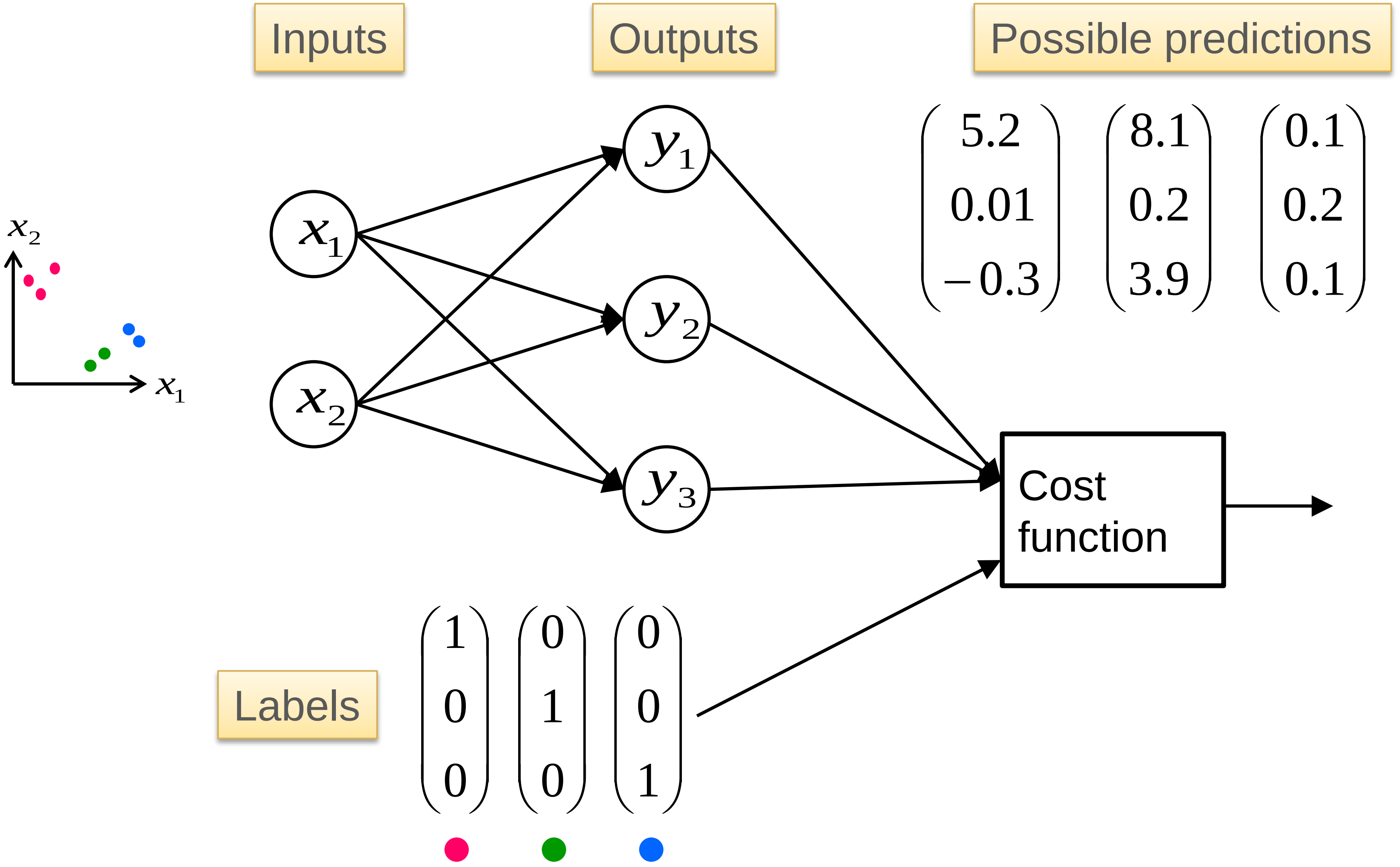
Biases

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

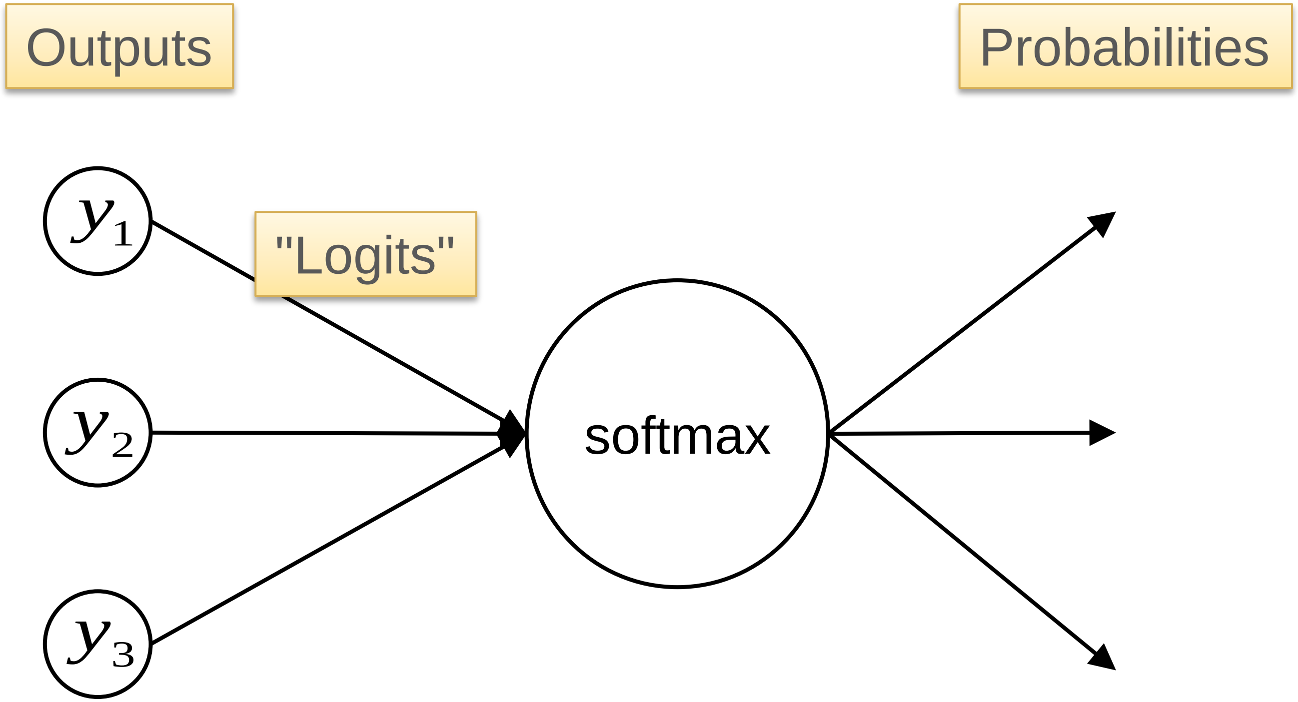
3 labels red, green, blue

2 features

Calculating the Cost Function



Converting Scores to Probabilities



$$\begin{pmatrix} 5.2 \\ 0.01 \\ -0.3 \end{pmatrix} \qquad \begin{pmatrix} 0.9849 \\ 0.0004 \\ 0.0148 \end{pmatrix}$$

The probabilities of a given label occurring

The Softmax Function

$$\sigma(y)_j = \frac{e^{y_j}}{\sum_k e^{y_k}}$$

$$\begin{pmatrix} -0.1 \\ 0.0 \\ +0.1 \end{pmatrix} \rightarrow \begin{pmatrix} 0.301 \\ 0.332 \\ 0.367 \end{pmatrix} \xleftrightarrow{compare} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} 1.0 \\ 2.0 \\ 3.0 \end{pmatrix} \rightarrow \begin{pmatrix} 0.090 \\ 0.245 \\ 0.665 \end{pmatrix} \xleftrightarrow{compare} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} 10 \\ 20 \\ 30 \end{pmatrix} \rightarrow \begin{pmatrix} 2\times10^{-9} \\ 5\times10^{-5} \\ 9.999 \end{pmatrix} \xleftrightarrow{compare} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

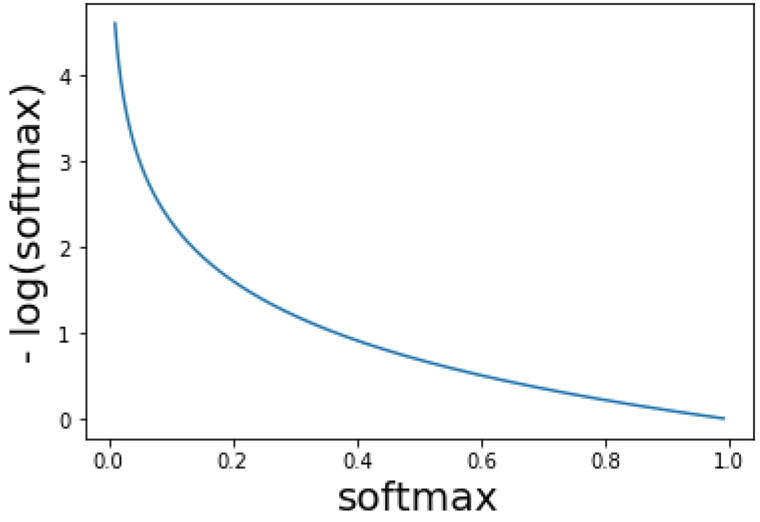
Compare using Cross-Entropy

$$H(\vec{\sigma}, \vec{\ell}) = -\sum_i \ell_i \log(\sigma_i)$$

Labels

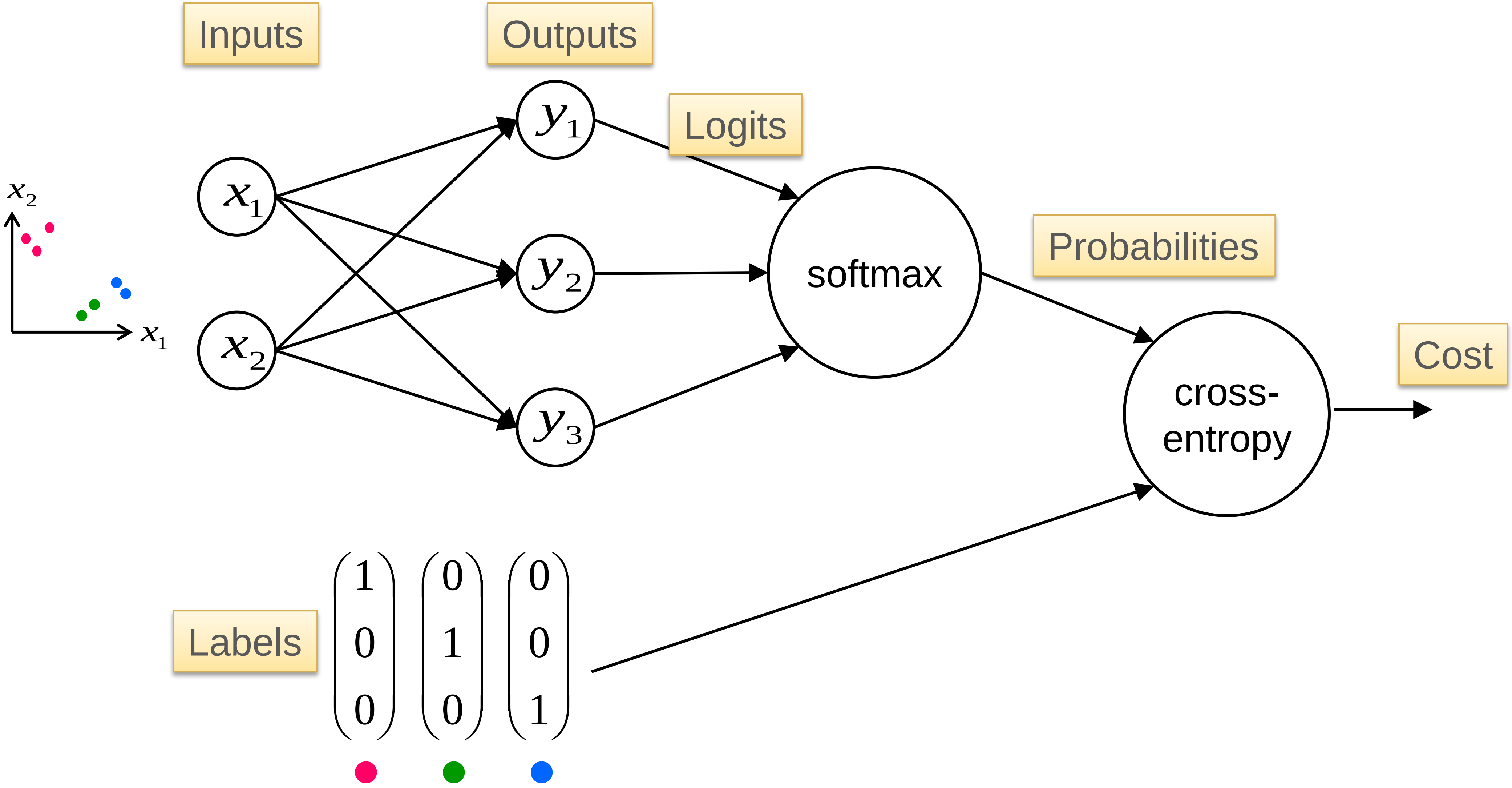
Softmax

$$\begin{pmatrix}1\\0\\0\end{pmatrix} \begin{pmatrix}0\\1\\0\end{pmatrix} \begin{pmatrix}0\\0\\1\end{pmatrix} \qquad \begin{pmatrix}0.1\\0.2\\0.7\end{pmatrix} \quad \sigma_i \in (0,1)$$

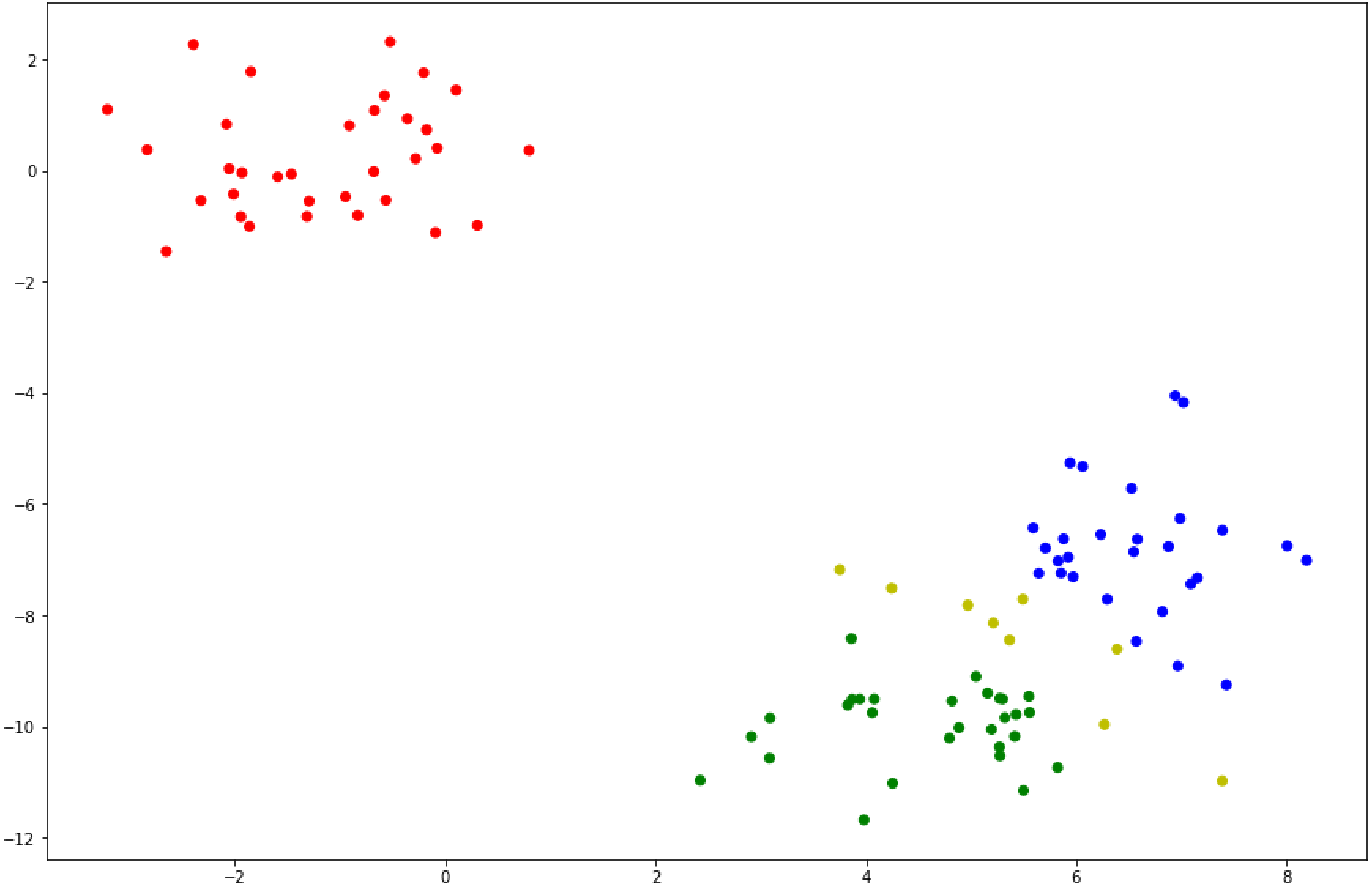


The closer the softmax value corresponding to the given label is to 1, the closer the cost is to 0.

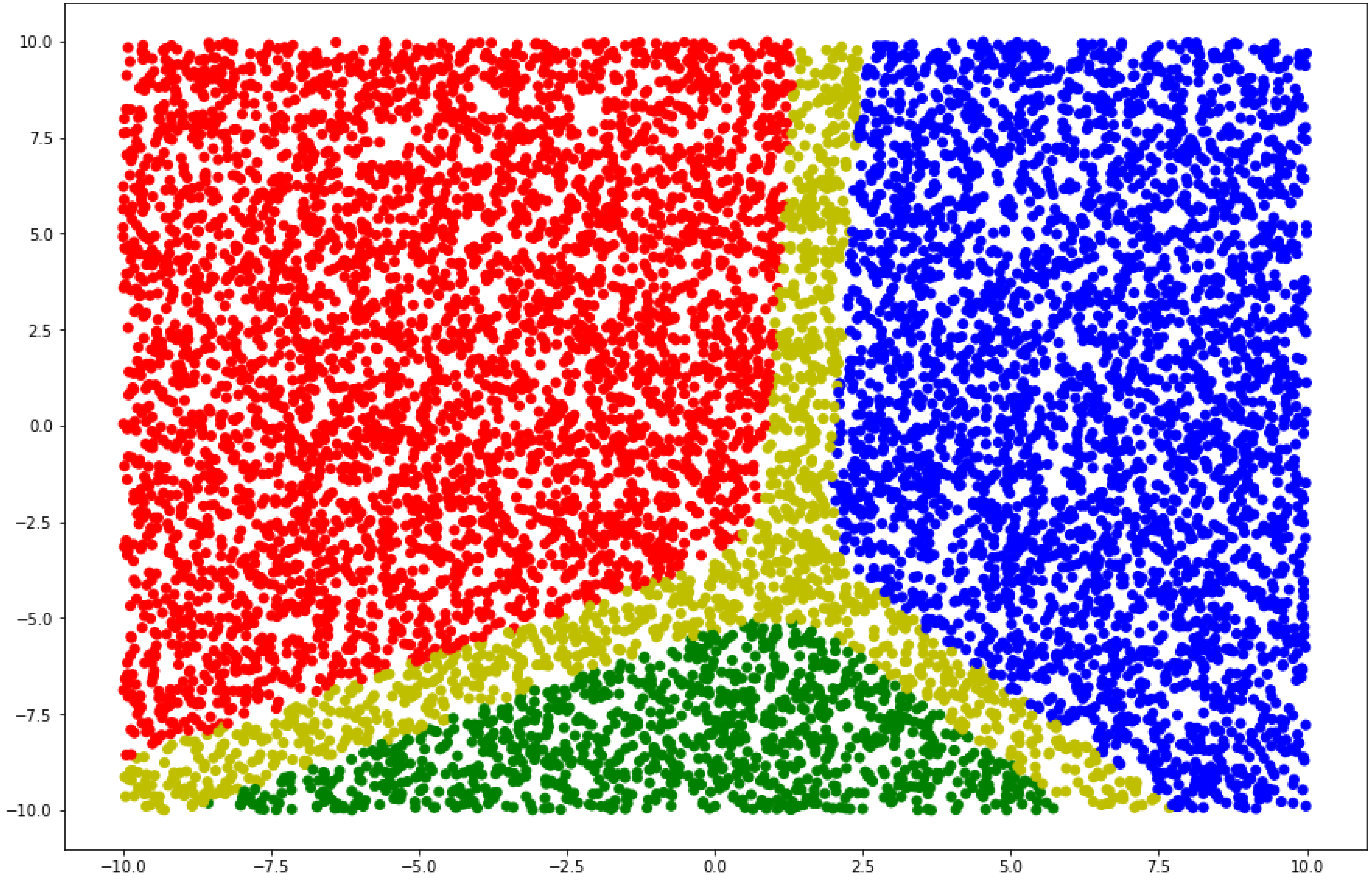
A Neural Network for Classification



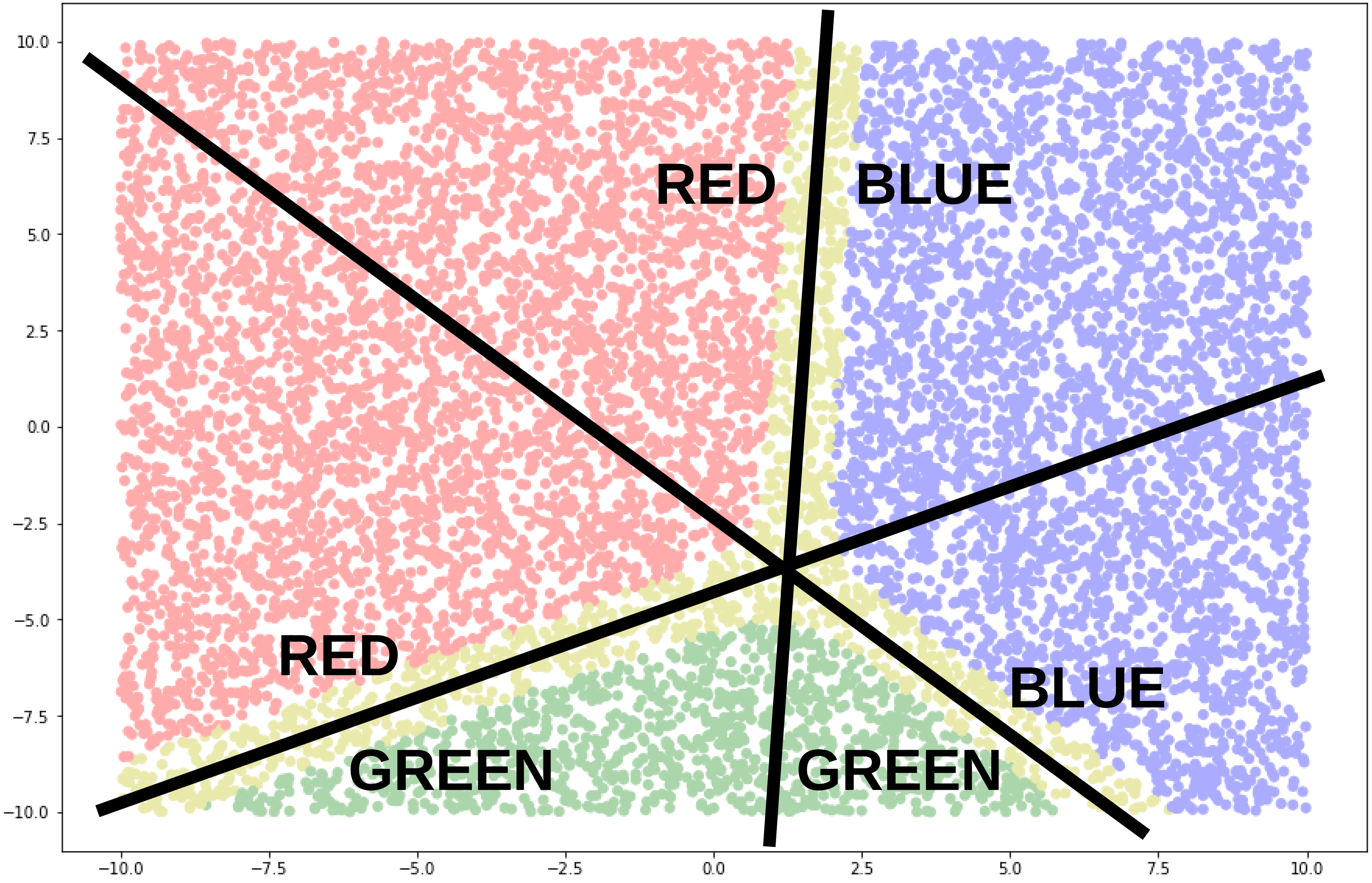
Predictions, Confidence > 0.8



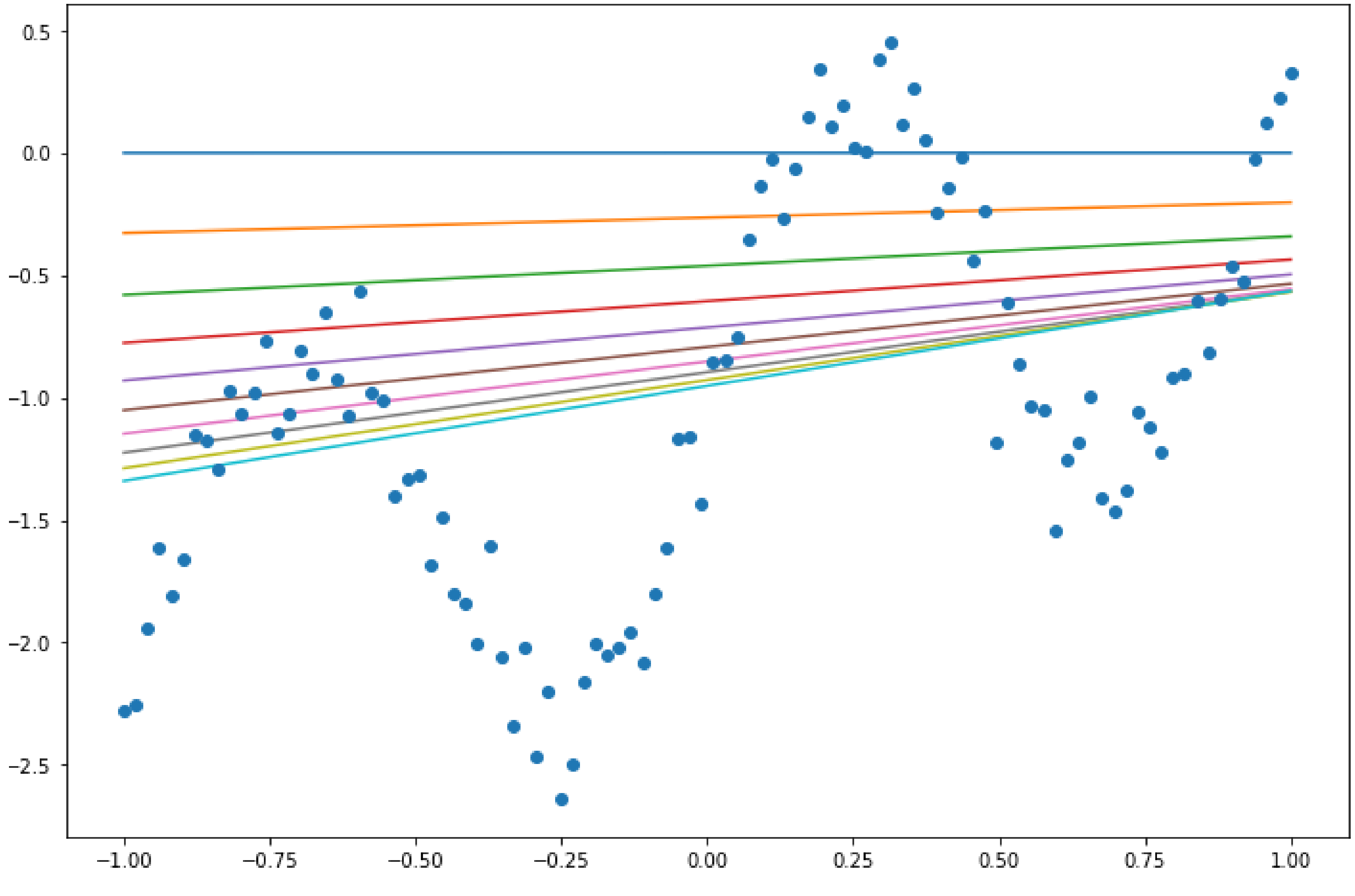
The Decision Boundary



Classes are Linearly Separable

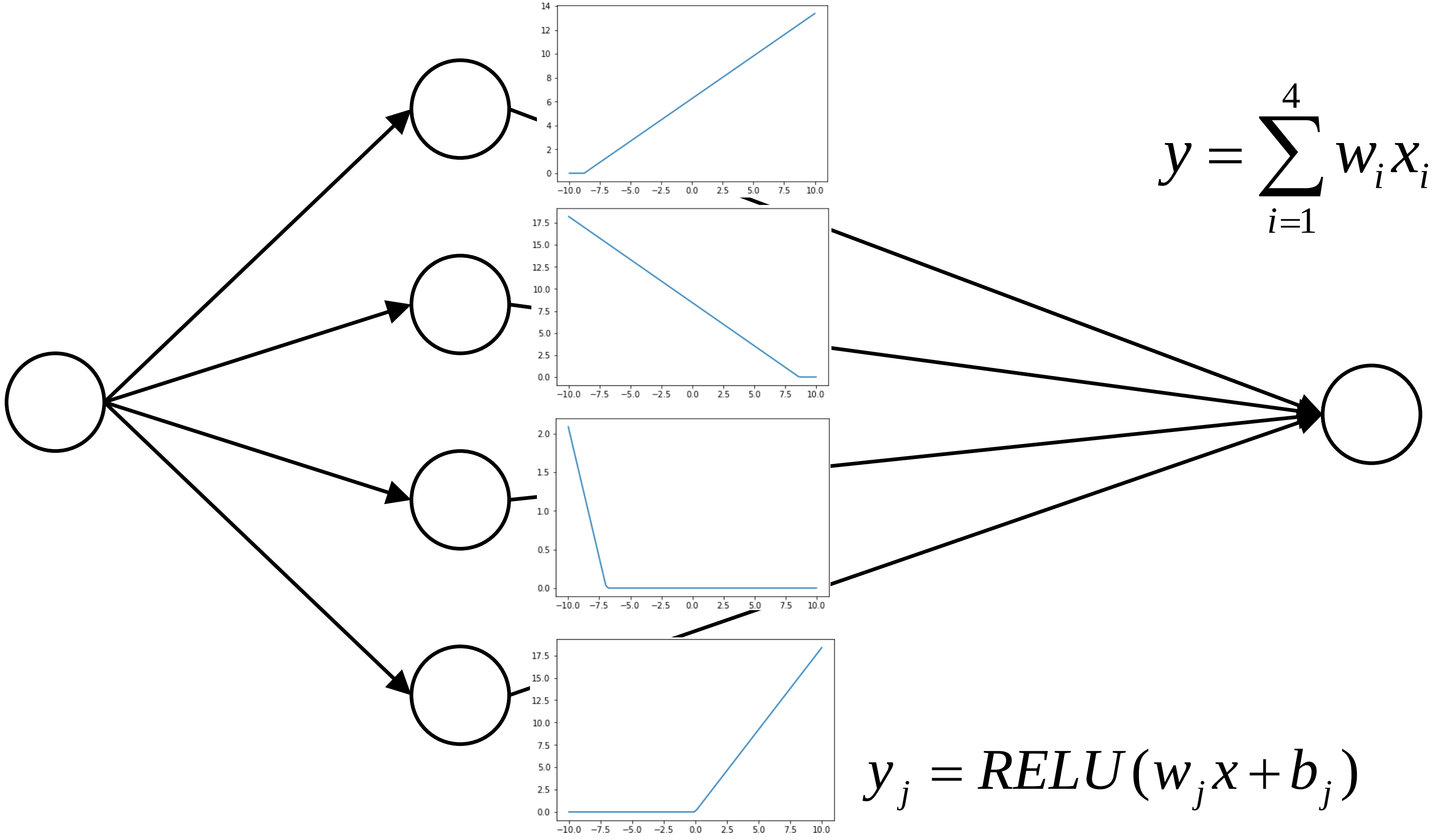


Non-Linear Regression and Classification

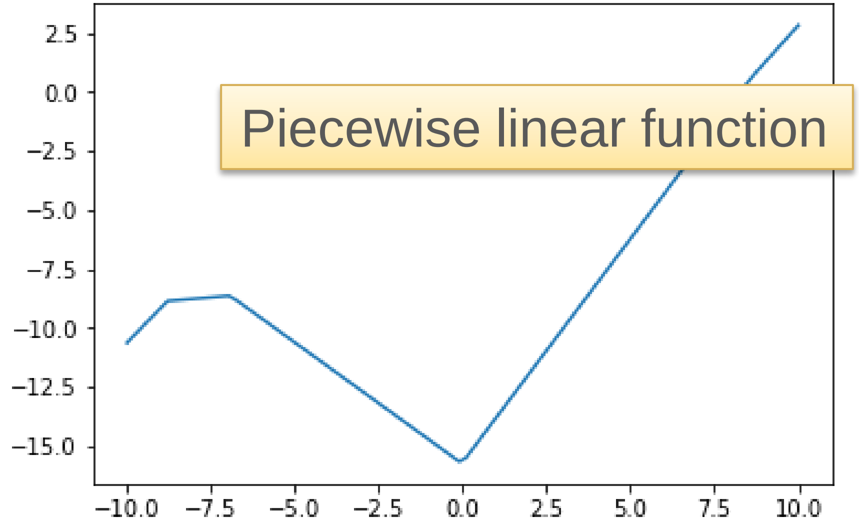


Piecewise Linear Approximation

Input unit Hidden units Output unit

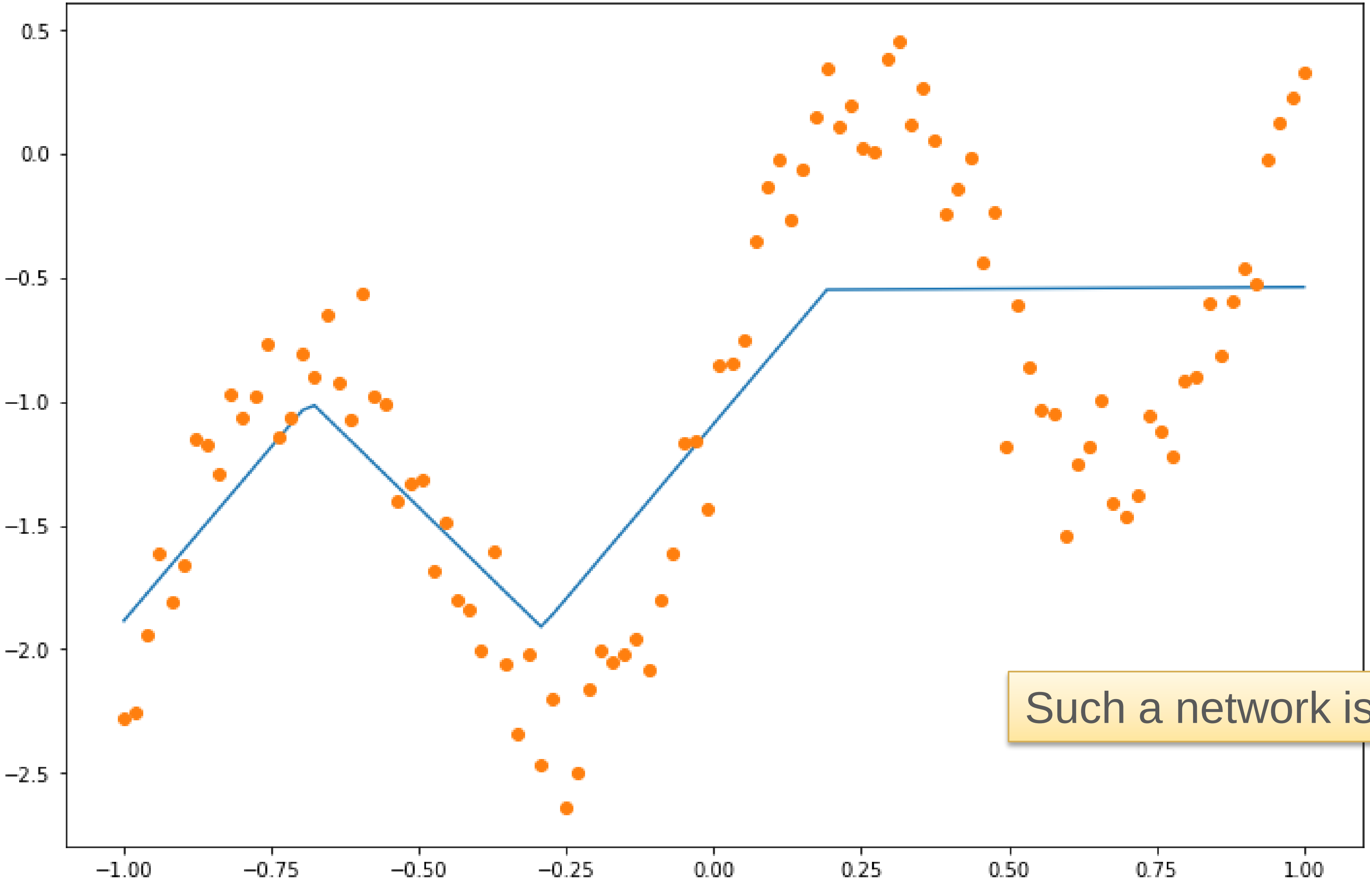


$$y = \sum_{i=1}^{4} w_i x_i + b$$



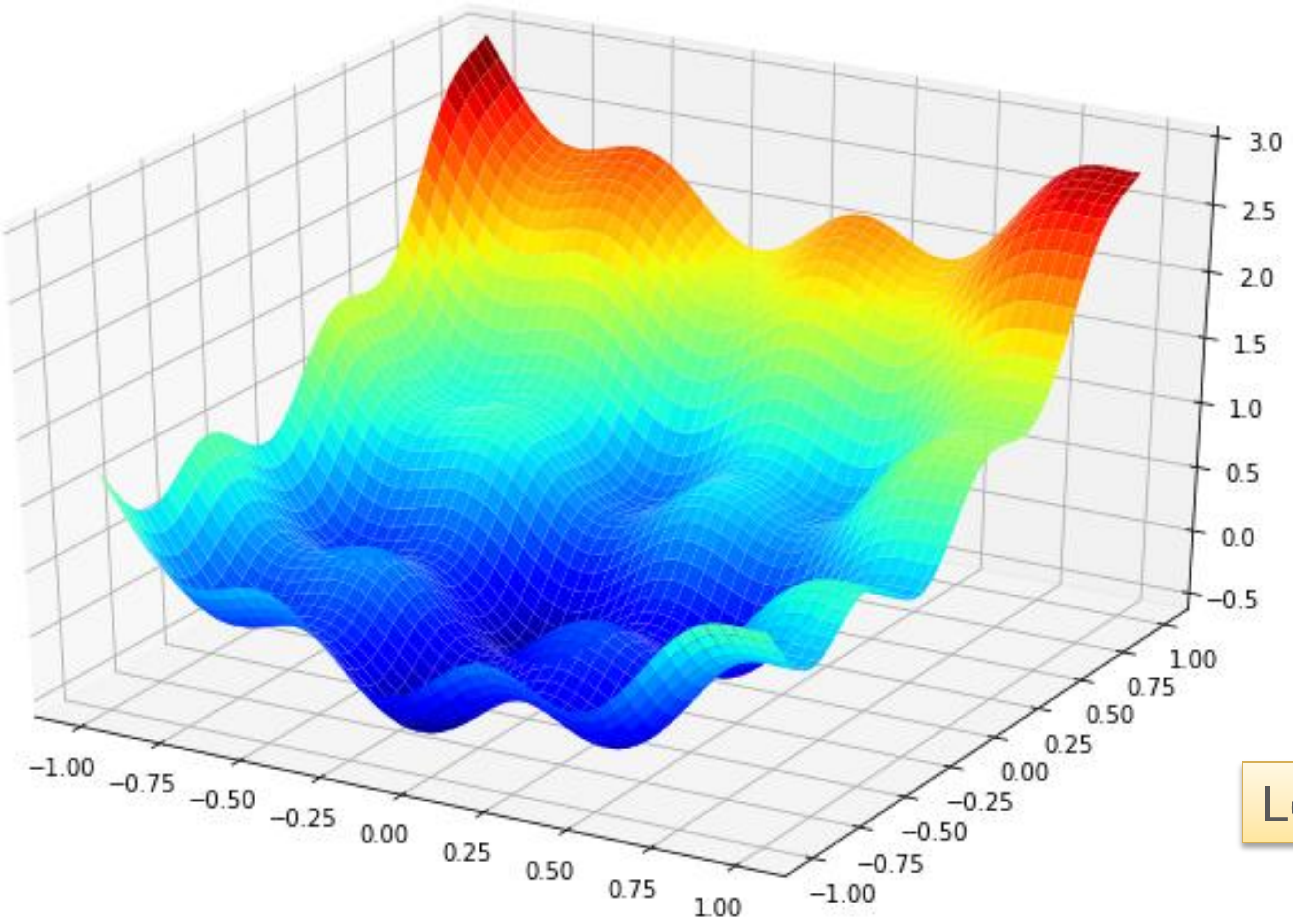
$$y_j = RELU(w_j x + b_j)$$

The Predicted Output



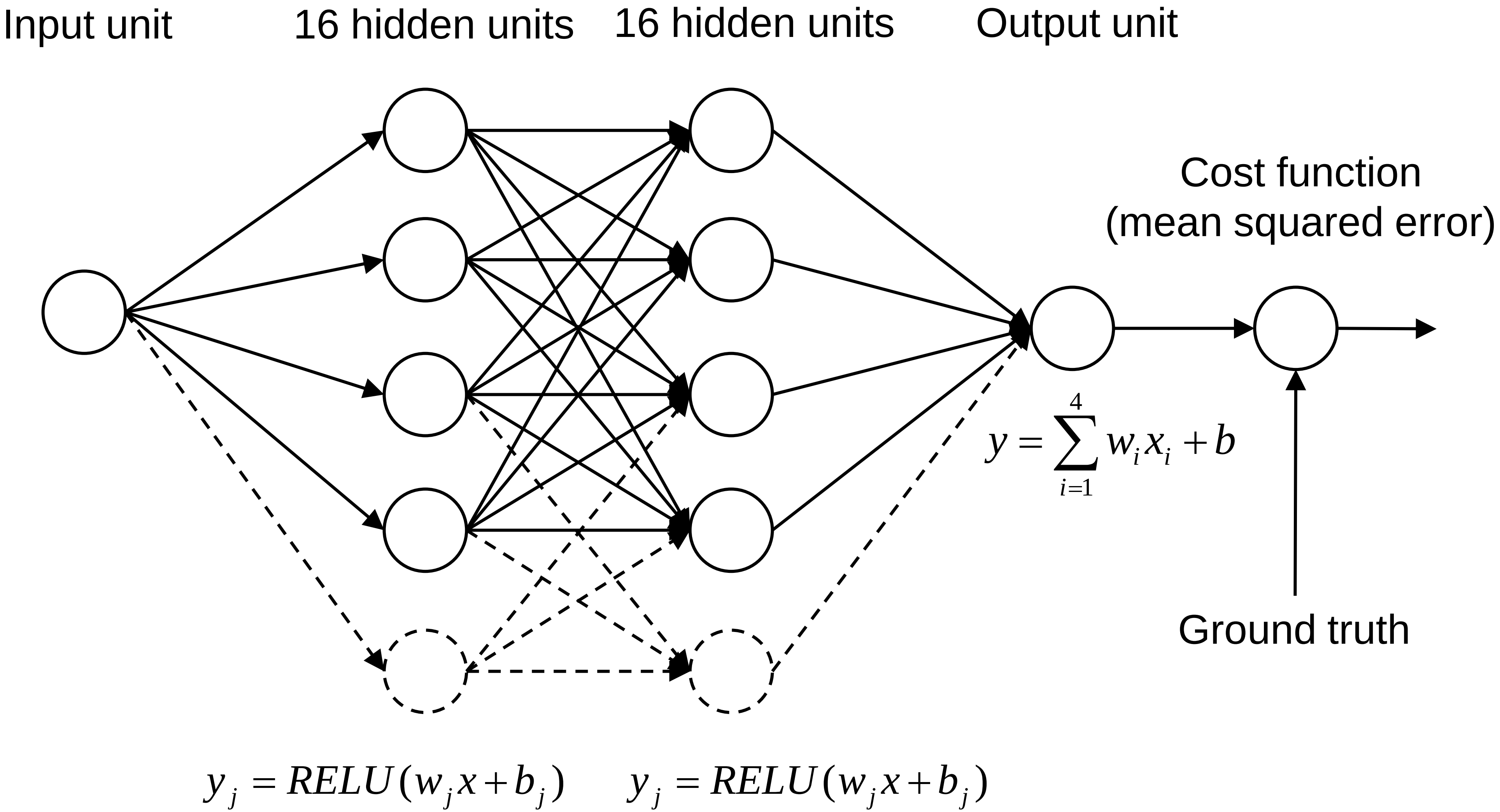
Such a network is hard to train

The Landscape of the Cost Function

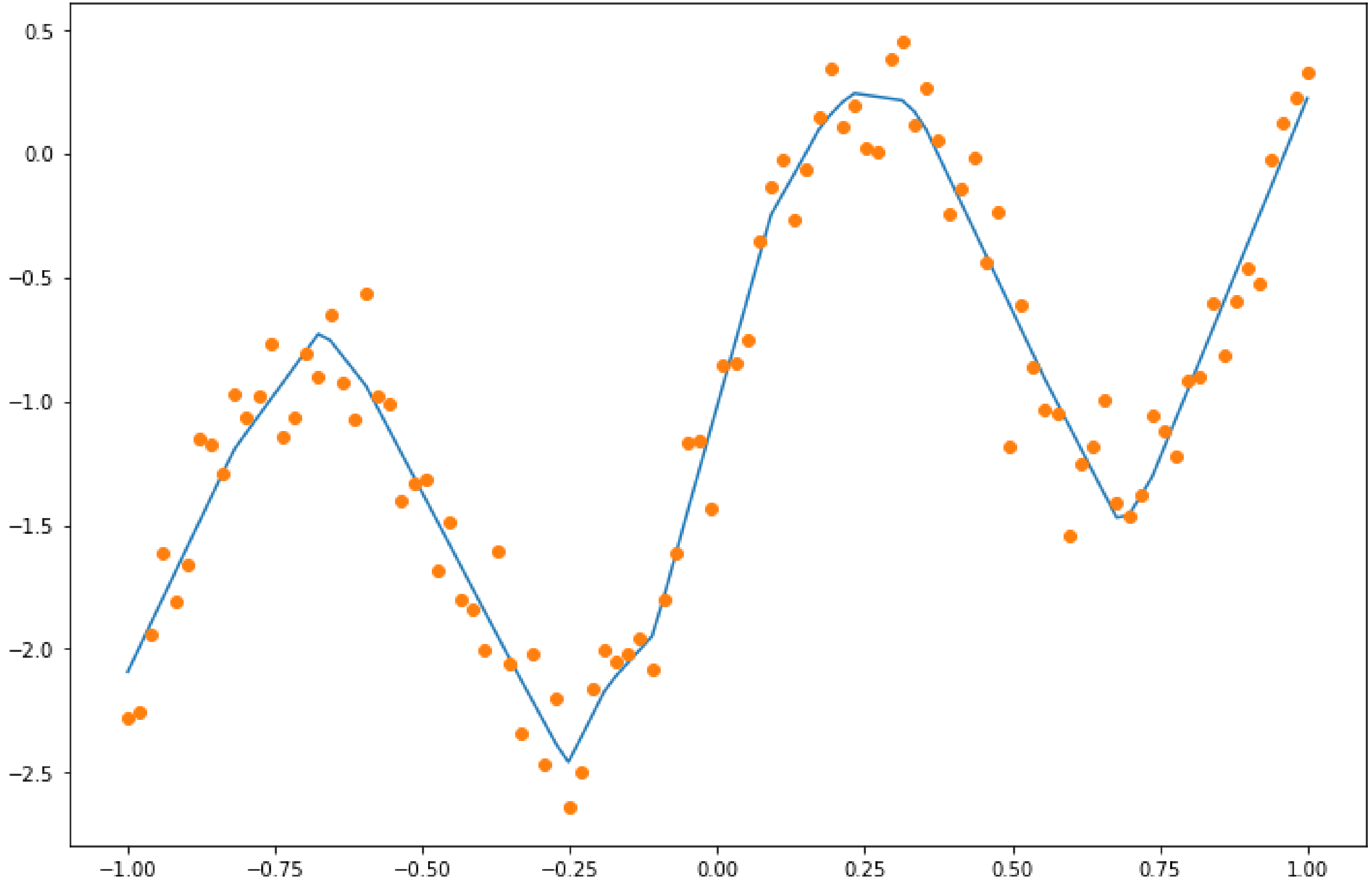


Local and global minima

A Deep Neural Network



The Predicted Output



The Magic of Deep Neural Networks

Deep neural networks have many degrees of freedom / degenerate / redundant

Gradient descent tends not to get stuck in local minima

Gradient descent tends to find a good global minimum

Why?

Most stationary points are saddle points, not local minima?

Deep Learning
for Design and Verification Engineers

What is Deep Learning?

Neural Networks

How a Network Learns

Getting Started

Libraries and Frameworks for Training

	Scikit-learn	Not deep learning
University of Montreal	**Theano**	Runs on GPU
	Pylearn2	(Theano)
	Lasagne	(Theano)
Berkeley AI Research	**Caffe**	Runs on GPU
Facebook, Twitter, ...	**Torch, PyTorch**	Runs on GPU
Google	**TensorFlow**	Runs on GPU
Microsoft	**CNTK**	Cognitive Tool Kit
	Keras	(Theano, TensorFlow, CNTK)
Skymind	**DL4J**	Deep Learning for Java
Apache	**MXNet**	
Intel	**Neon**	Optimized for Intel CPUs
MathWorks	**MATLAB**	Various toolboxes

Deep Learning Platforms and Toolkits

Amazon Deep Learning AMIs

Au-Zone DeepView

Google Cloud Machine Learning Engine

IBM Watson

Intel Nervana Cloud

Microsoft Azure Machine Learning Studio

MVTec HALCON

NVIDIA TensorRT

Qualcomm Snapdragon NPE SDK

Xilinx reVISION

And many more! All trademarks acknowledged

Deep Learning IP and Chips

BrainChip

Cadence Tensilica Vision DSP

Google TPU

Graphcore

IBM TrueNorth

Intel Loihi

Intel Movidius

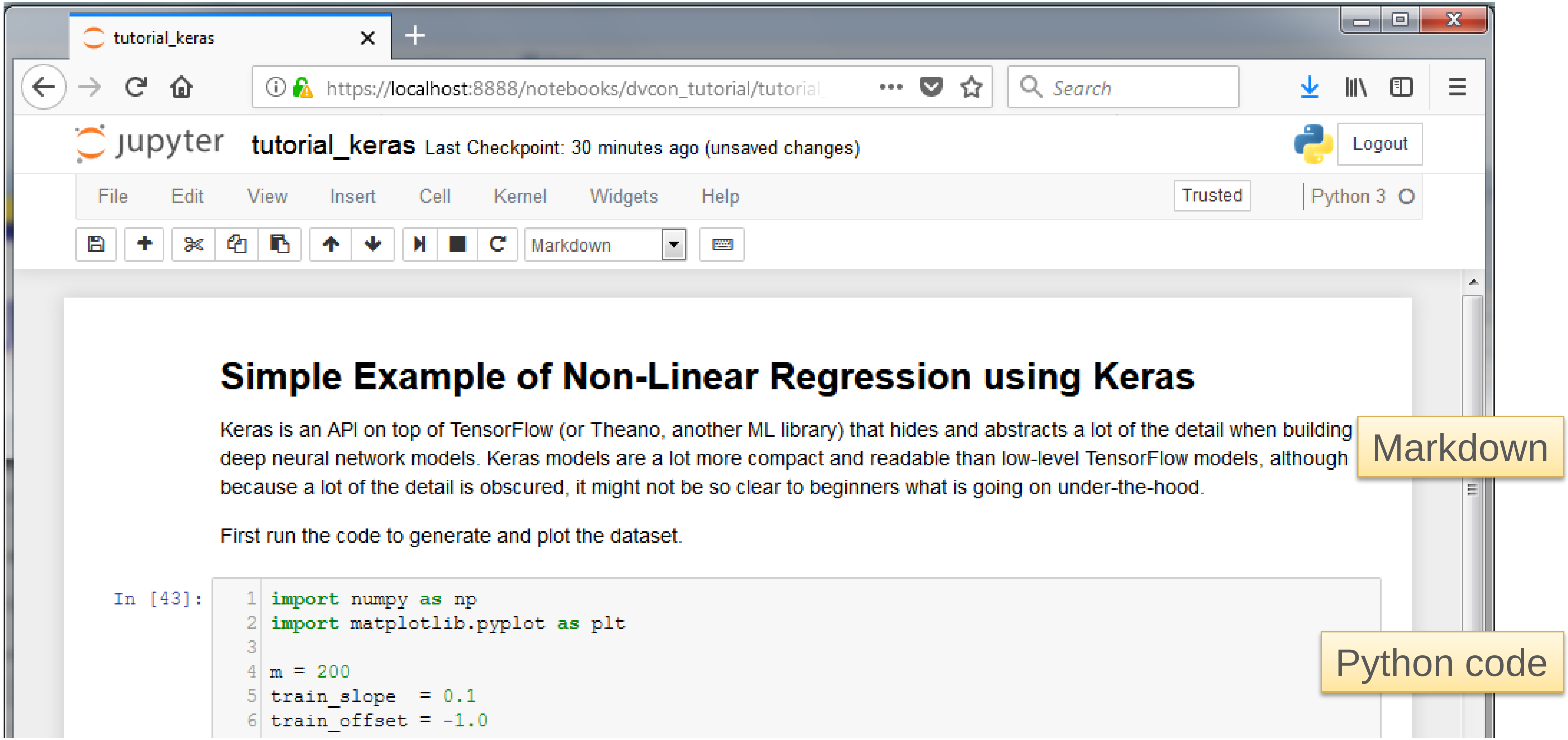
KAIST MVLSI Laboratory

KALRAY MPPA

Synopsys DesignWare EV6x

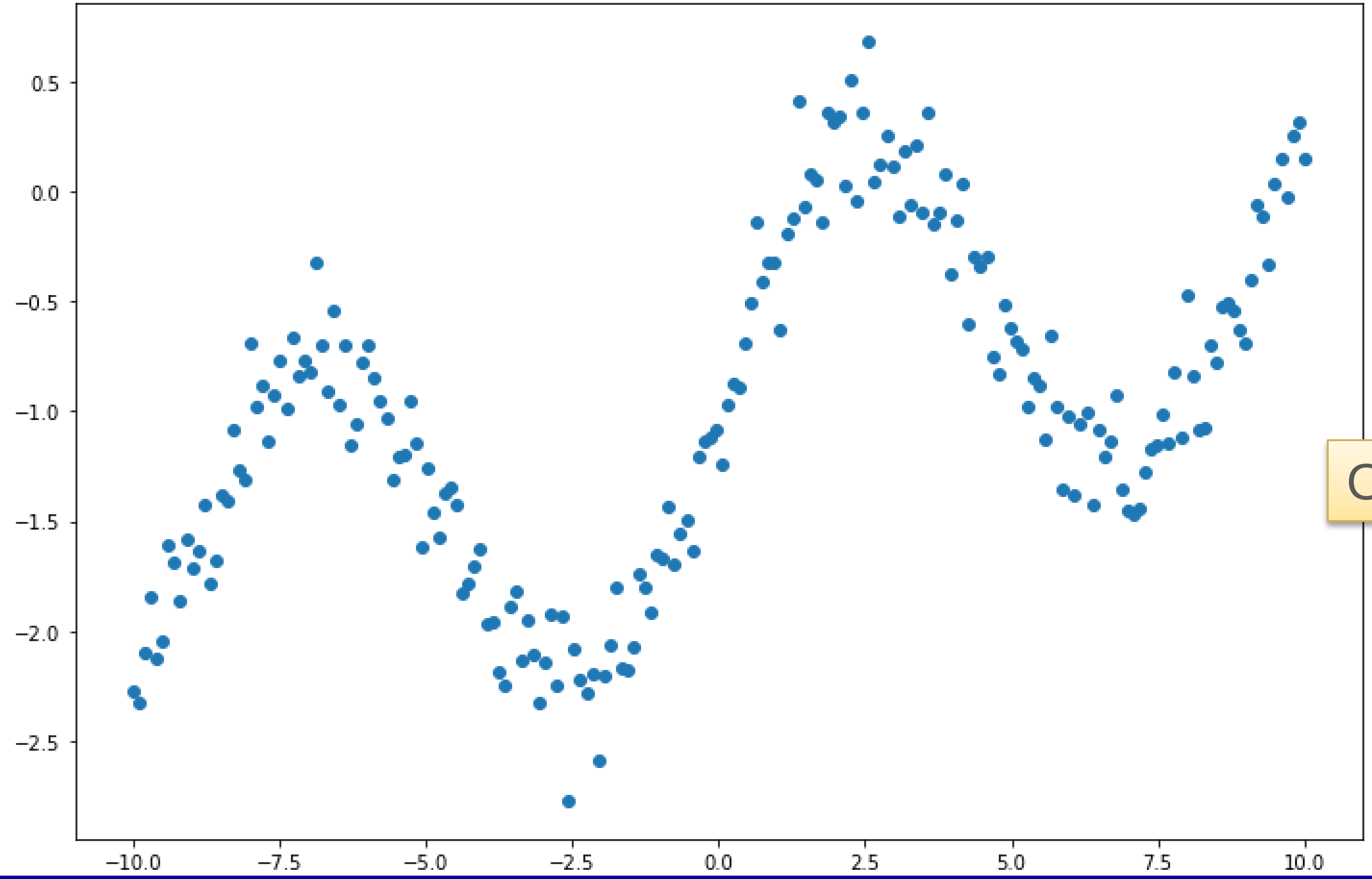
And many more! All trademarks acknowledged

Python and Jupyter Notebook



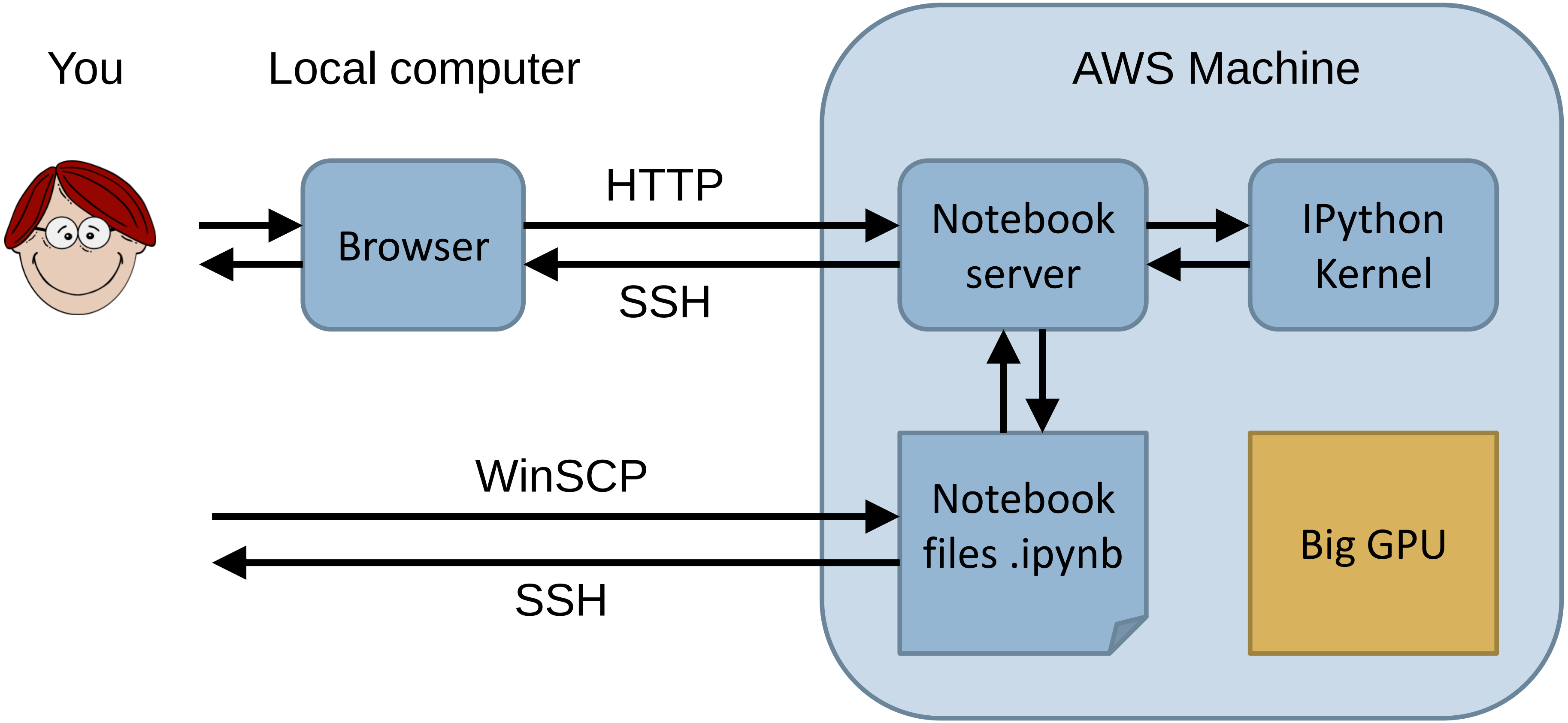
```
train_slope  = 0.1
train_offset = -1.0

train_x = np.linspace(-10, 10, m).astype(np.float32)
rng = np.random.RandomState(seed=42)
train_y = (train_slope * train_x + np.sin(train_x / 1.5) + train_offset +
           rng.normal(0.0, 0.2, size=len(train_x))).astype(np.float32)

plt.rcParams["figure.figsize"] = (12, 8)
plt.plot(train_x, train_y, 'o');
plt.show()
```

Python code

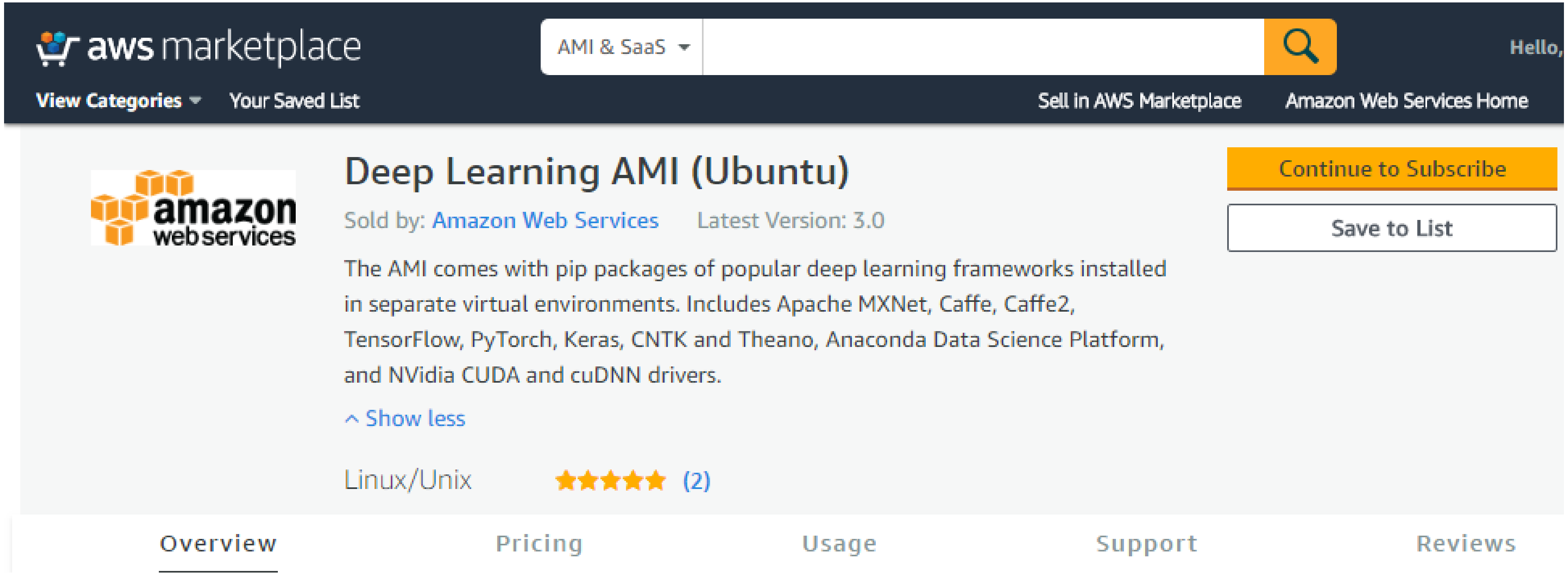


Output

Jupyter Notebook Architecture



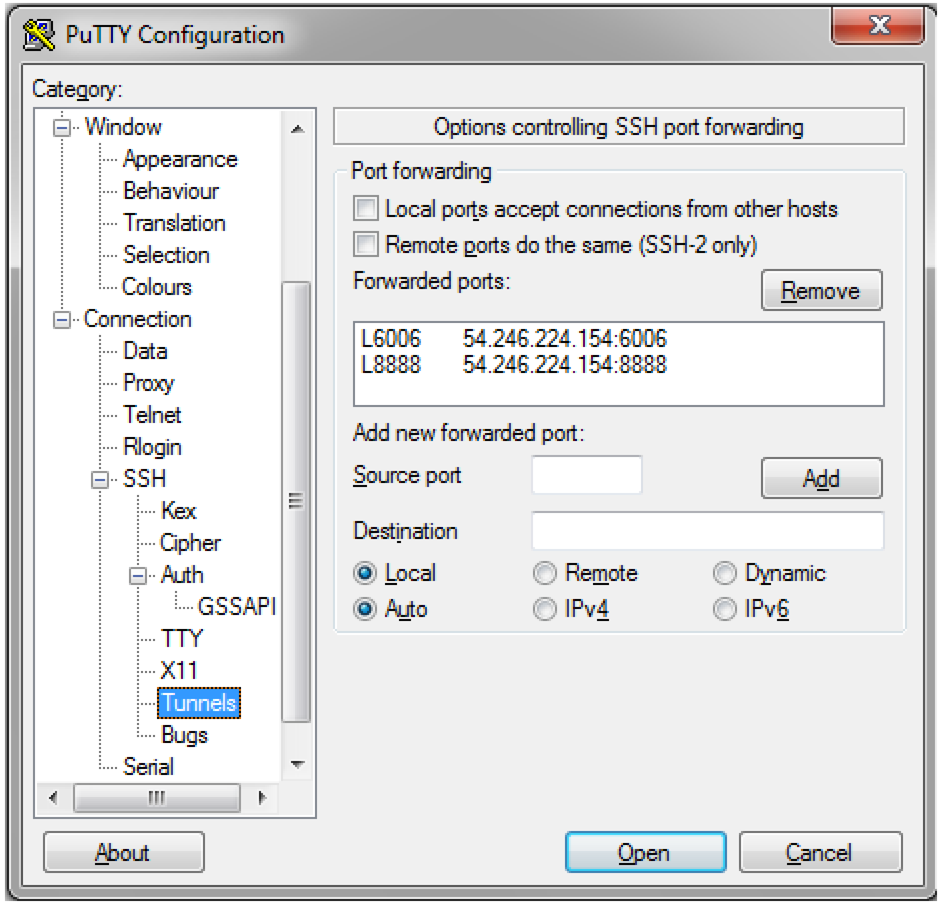
AWS Deep Learning AMI



Apache MXNet, Caffe2, TensorFlow, PyTorch, Keras, CNTK, Theano

SSH Port Forwarding

Jupyter Notebook in local web browser



Amazon EC2 instance with NVIDIA Tesla GPU

Build a neural network model with two hidden layers using the Keras API.

In [47]:
```python
import keras
from keras.models     import Sequential
from keras.layers     import Dense

keras.backend.clear_session()
n_hidden = 32

model = Sequential()
model.add(Dense(input_dim=1, units=n_hidden, activation='relu'))
model.add(Dense(units=n_hidden, activation='relu'))
model.add(Dense(units=1))
model.summary()
model.compile(loss='mean_squared_error', optimizer='sgd', metrics=['accuracy'])
```

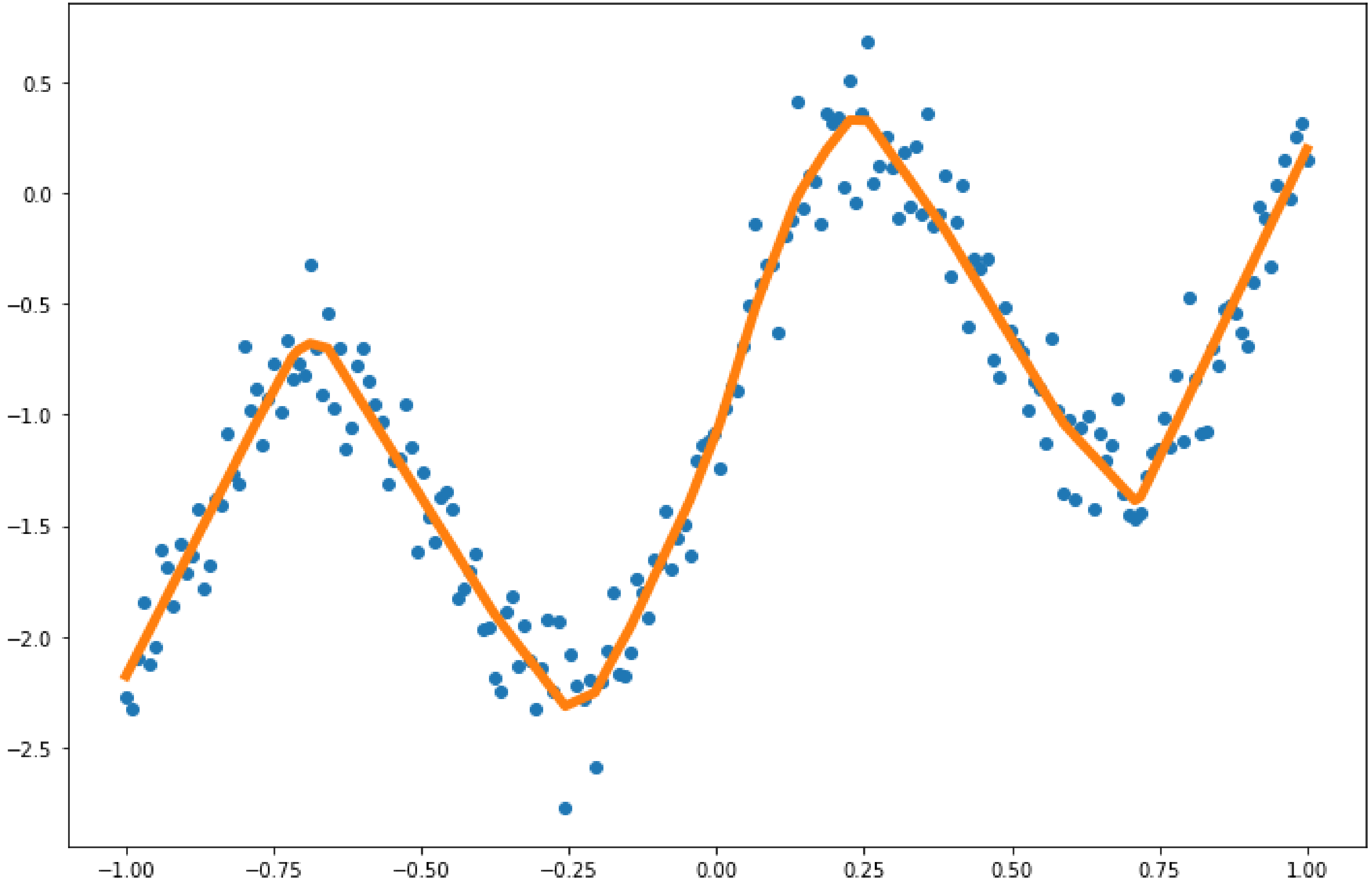
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 32)                64
_________________________________________________________________
dense_2 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,153
Trainable params: 1,153
Non-trainable params: 0
_________________________________________________________________
```

In [44]:
```python
model.fit(train_x, train_y, epochs=10000, batch_size=m, verbose=0)
```

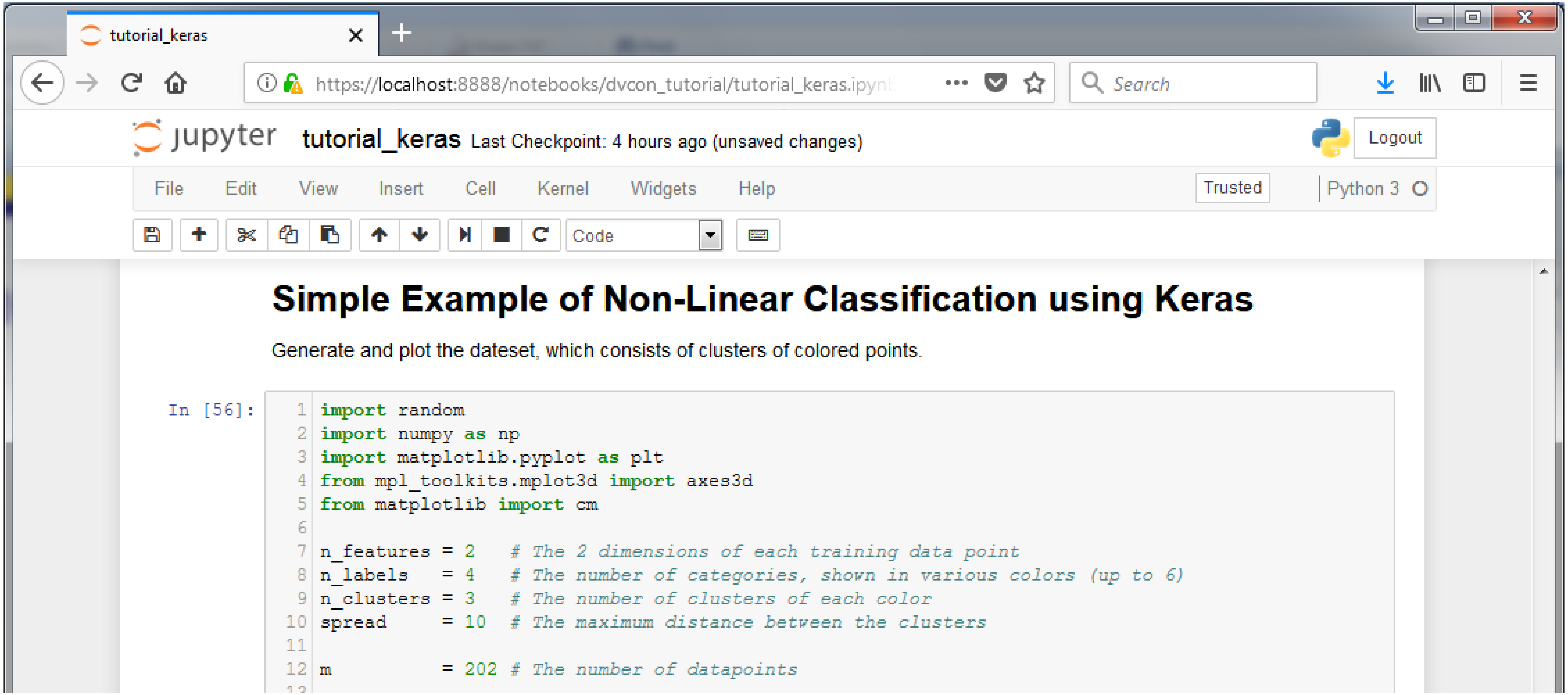
Plot the predicted curve learnt by the model.

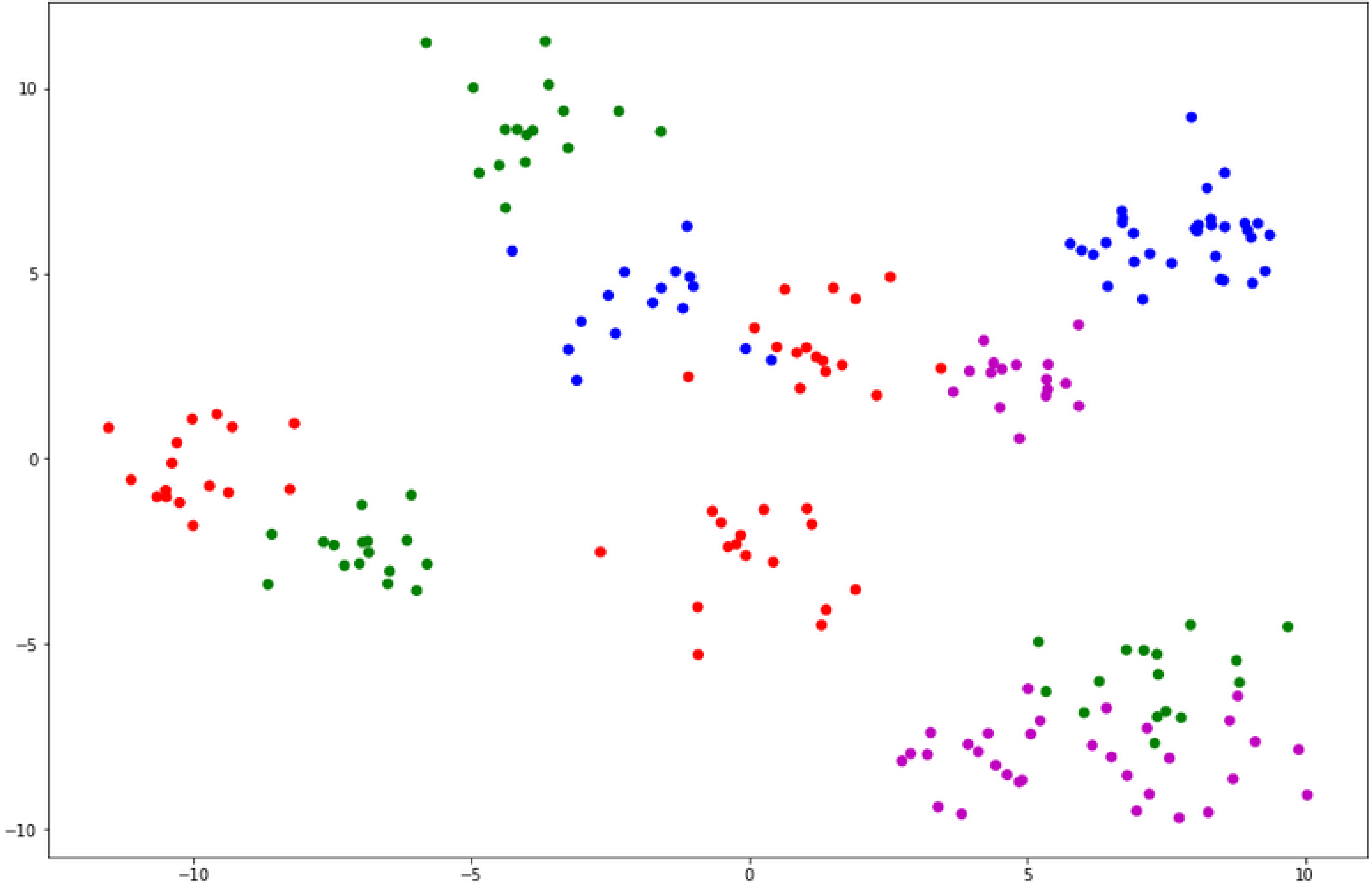
In [42]:

```
y = model.predict(train_x)
plt.plot(train_x, train_y, 'o')
plt.plot(train_x, y, linewidth=5)
plt.show()
```



Classification




10
5
0
−5
−10
−10
−5
0
5
10

In [98]:

```python
import keras
from keras.models     import Sequential
from keras.layers     import Dense
from keras.optimizers import sgd

keras.backend.clear_session()
n_hidden = 16

model = Sequential()
model.add(Dense(input_dim=n_features, units=n_hidden, activation='relu'))
model.add(Dense(units=n_hidden, activation='relu'))
model.add(Dense(units=n_hidden, activation='relu'))
model.add(Dense(units=n_labels, activation='softmax'))
model.summary()
model.compile(loss='categorical_crossentropy', optimizer=sgd(lr=0.05), metrics=['accuracy'])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 16)                48
_________________________________________________________________
dense_2 (Dense)              (None, 16)                272
_________________________________________________________________
dense_3 (Dense)              (None, 16)                272
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 68
=================================================================
Total params: 660
```

Run and Evaluate the Model

Run gradient descent.

In [101]:
```
model.fit(train_x, train_y, epochs=5000, batch_size=m, shuffle=False, verbose=0)
```

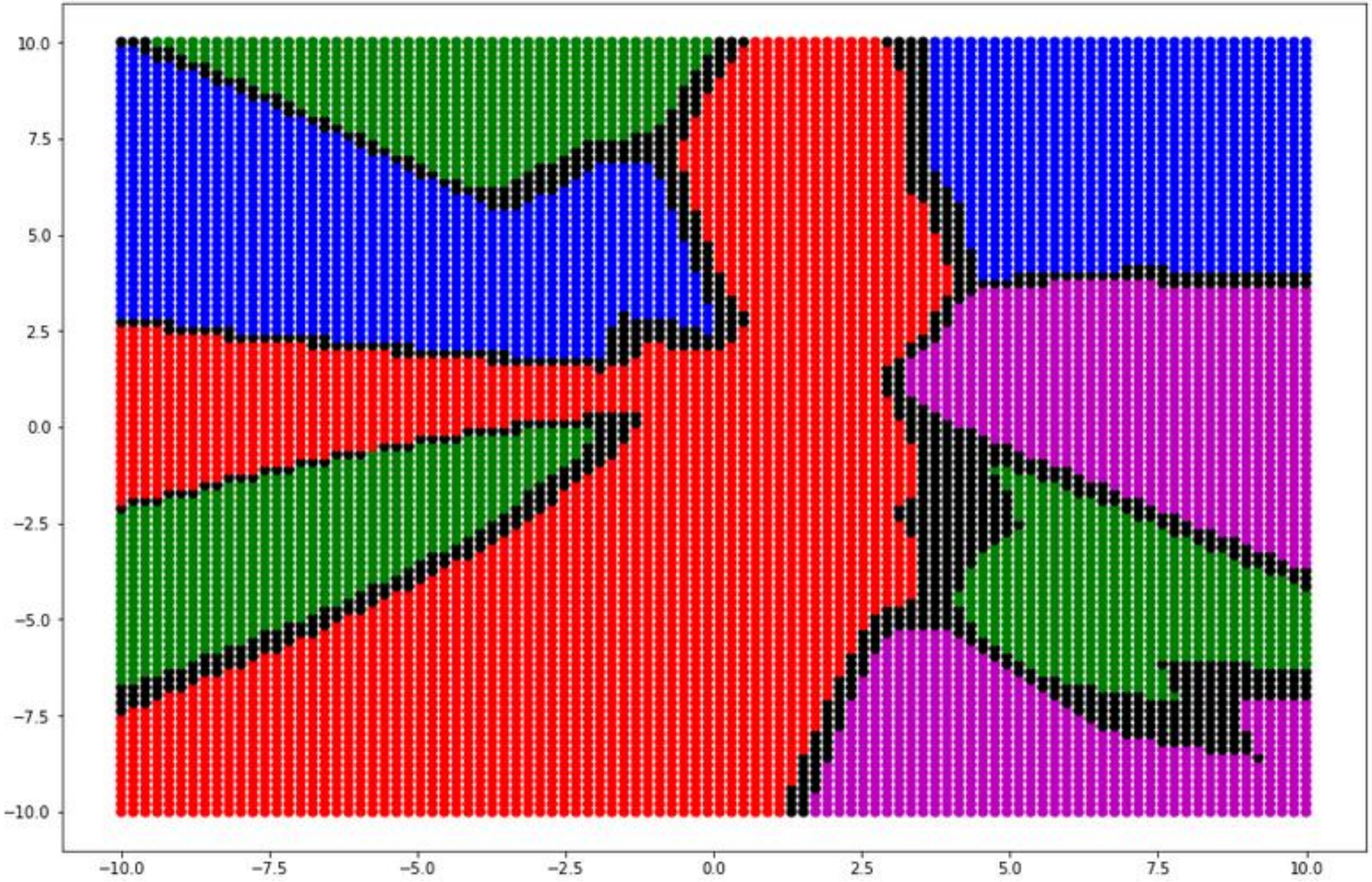
Out[101]: `<keras.callbacks.History at 0x7f416fe2d390>`

Evaluate the trained model on the training data.

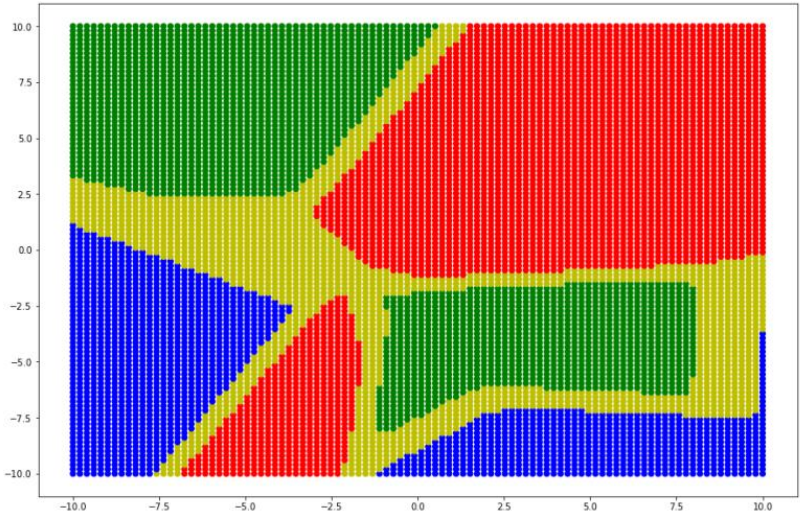
In [102]:
```
loss_and_acc = model.evaluate(train_x, train_y, batch_size=m, verbose=0)
print('Accuracy = {:4.2f}'.format(loss_and_acc[1]))
```

```
Accuracy = 0.98
```

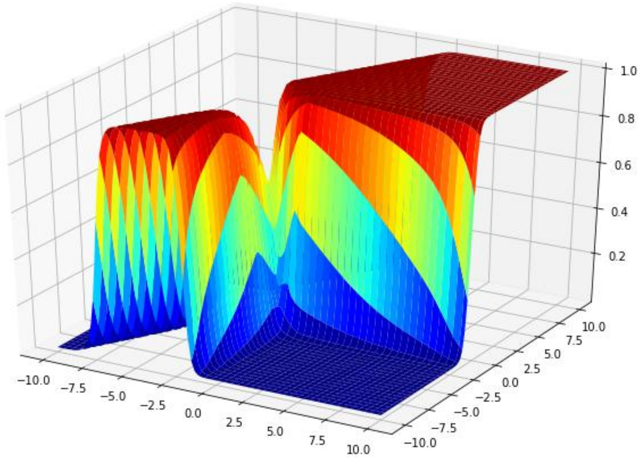
The Decision Boundary



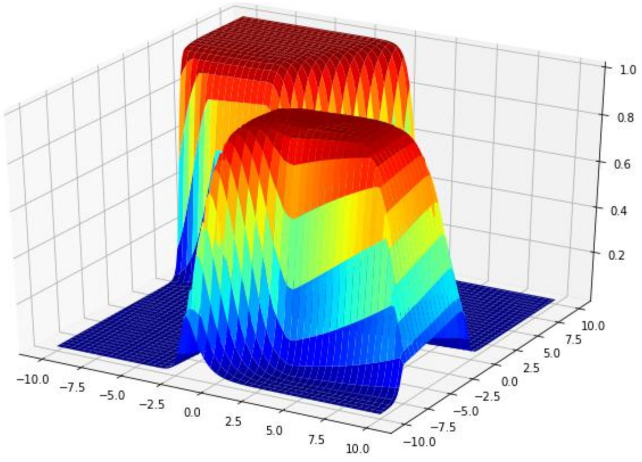
Softmax Probability for each Label



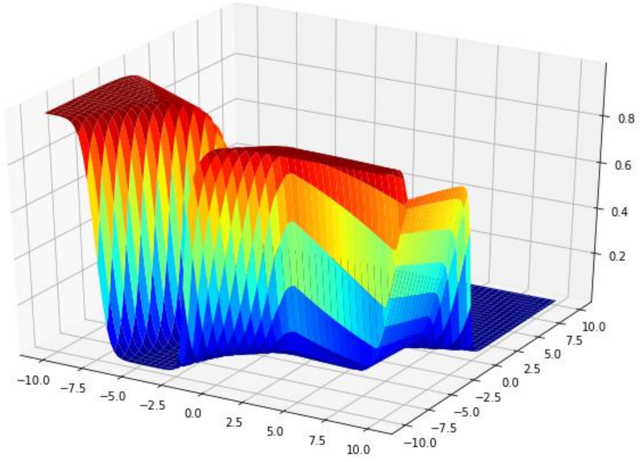
$$\begin{pmatrix}1\\0\\0\end{pmatrix}$$



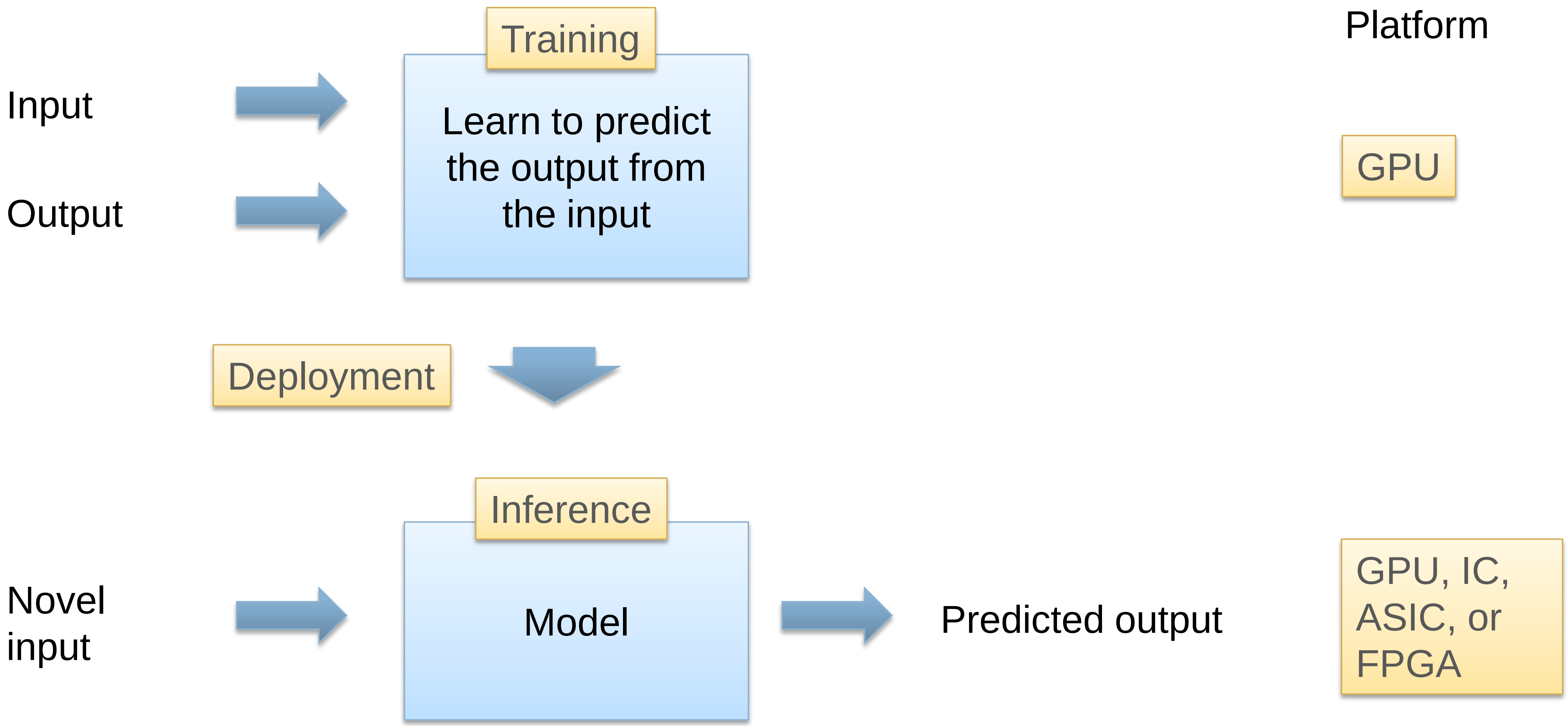
$$\begin{pmatrix}0\\1\\0\end{pmatrix}$$



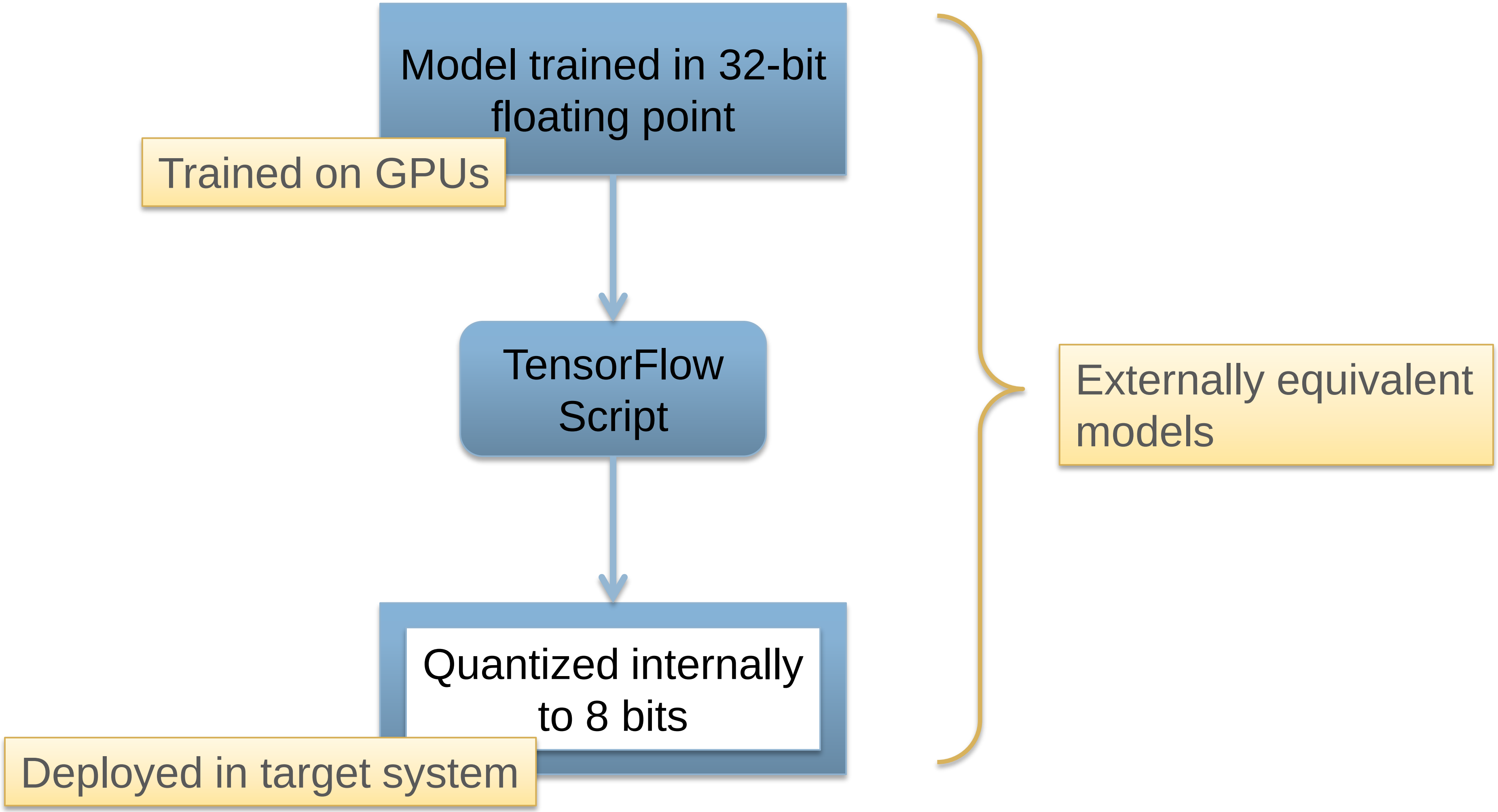
$$\begin{pmatrix}0\\0\\1\end{pmatrix}$$



Training versus Inference



Weight Quantization



For More Information

Example Jupyter Notebook:

www.doulos.com/downloads/dvcon_dl.ipynb

john.aynsley@doulos.com