Machine Learning: TensorFlow

Bolous AbuJaber

Outline

- ML Introduction
- Supervised deep learning
  - Neural network models
  - Back-propagation
  - Training procedures
- Supervised DL for images
  - Neural network architectures for images.

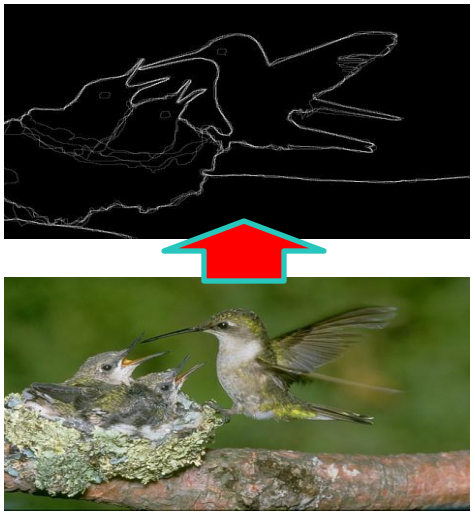
Coding

What do we want ML to do?

- Given image, predict complex high-level patterns:

Object recognition

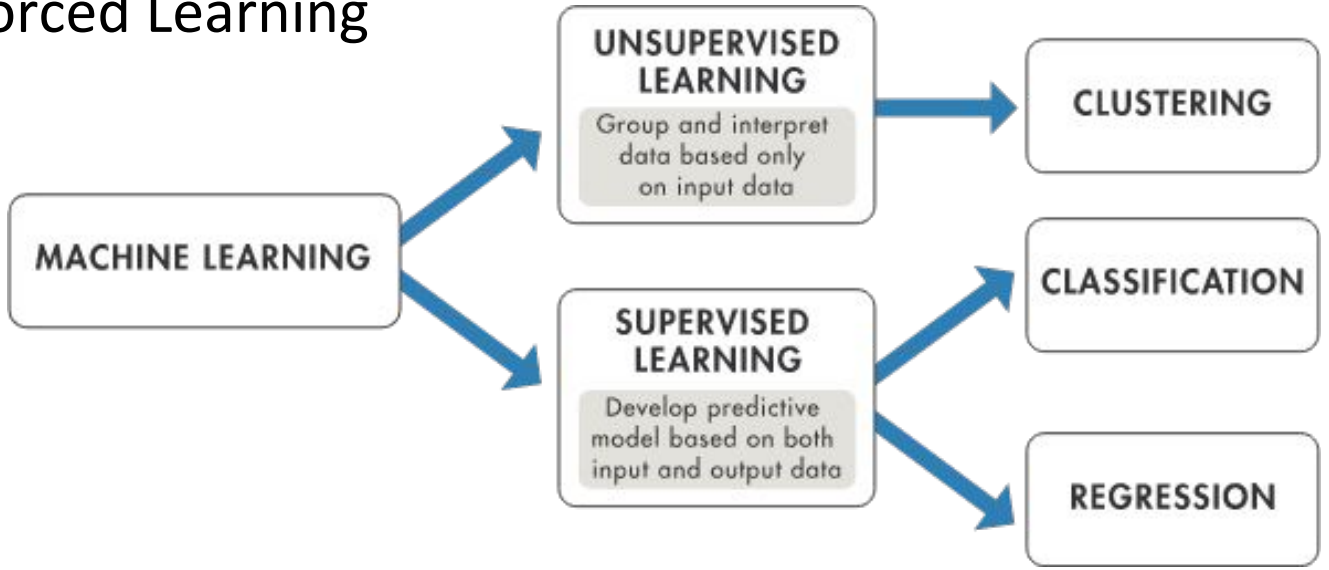
Detection



Segmentation
[Martin et al., 2001]

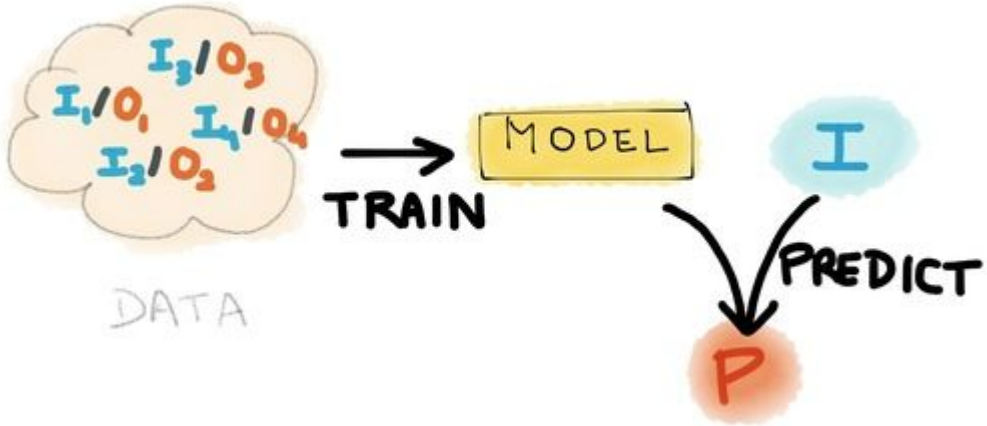
Types of ML

- Unsupervised learning
- Supervised learning (Focus)
- Reinforced Learning



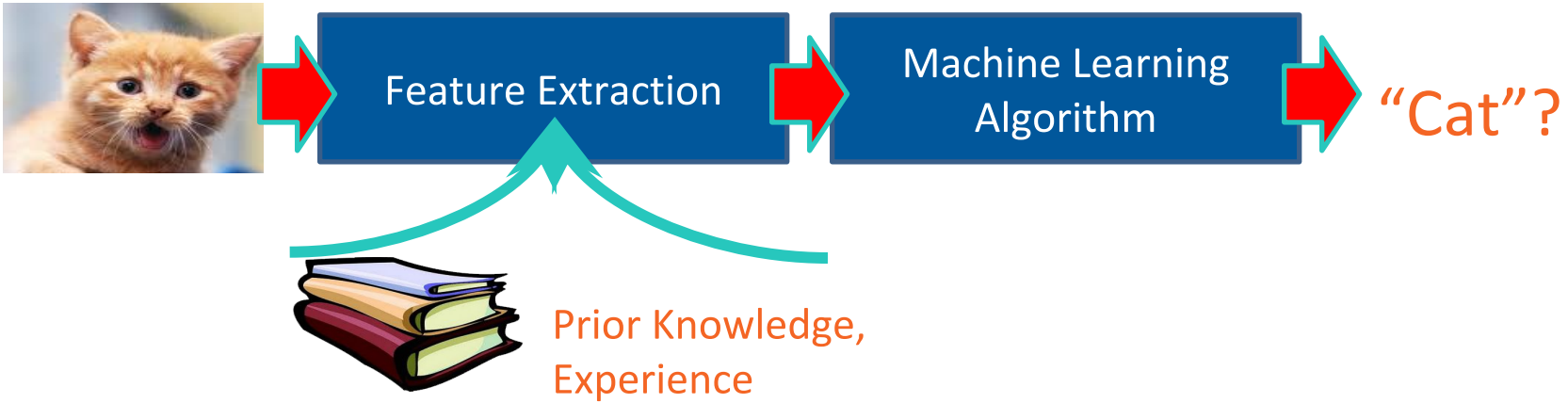
Supervised Learning

- We train a model
- Learn from training examples
- Need labeled dataset
- Lots of examples

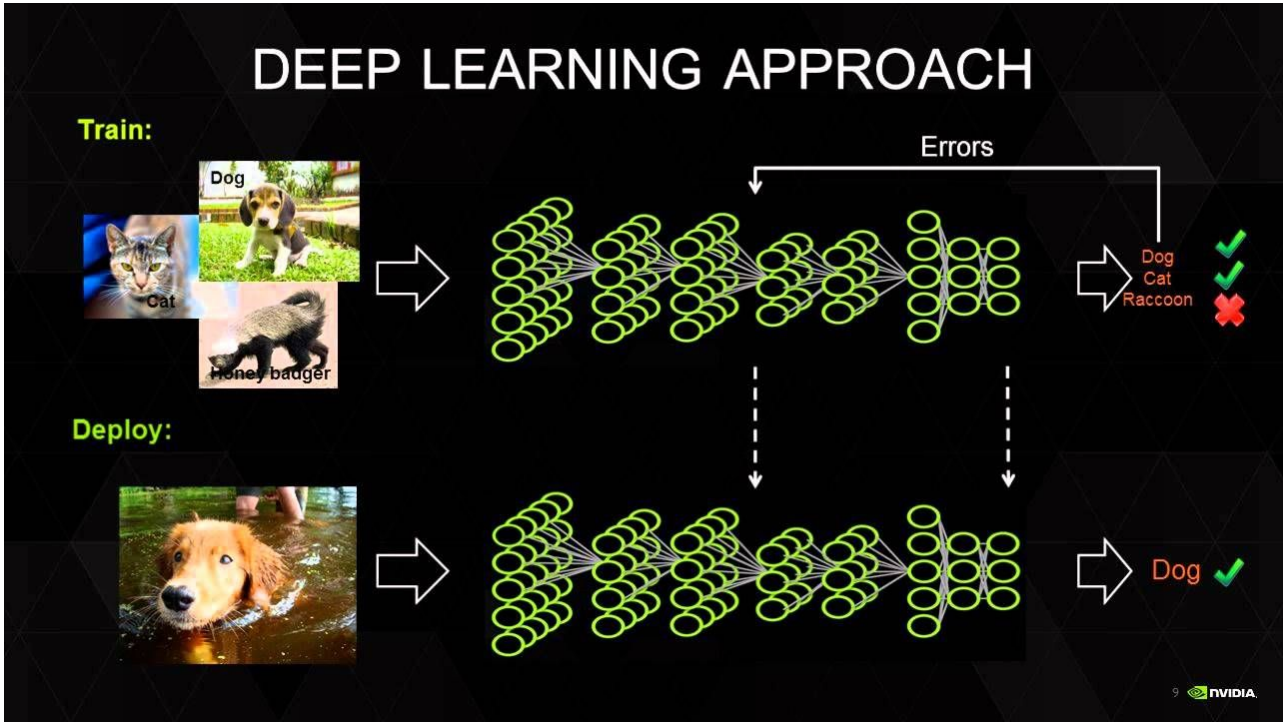


How is ML done?

- Machine learning often uses common pipeline with hand-designed feature extraction.
- Better models use End to End Approaches (CNN)



Supervised Learning

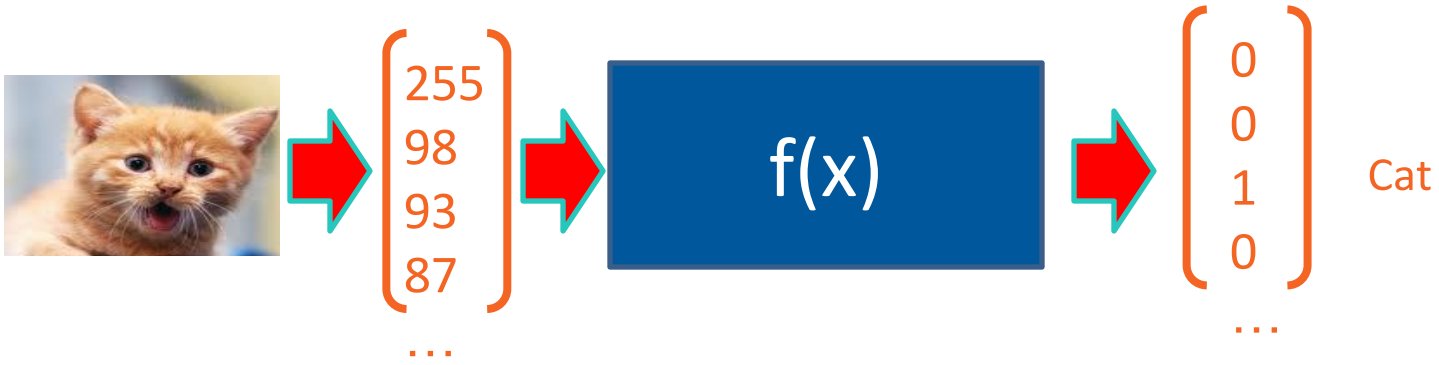


Supervised Learning

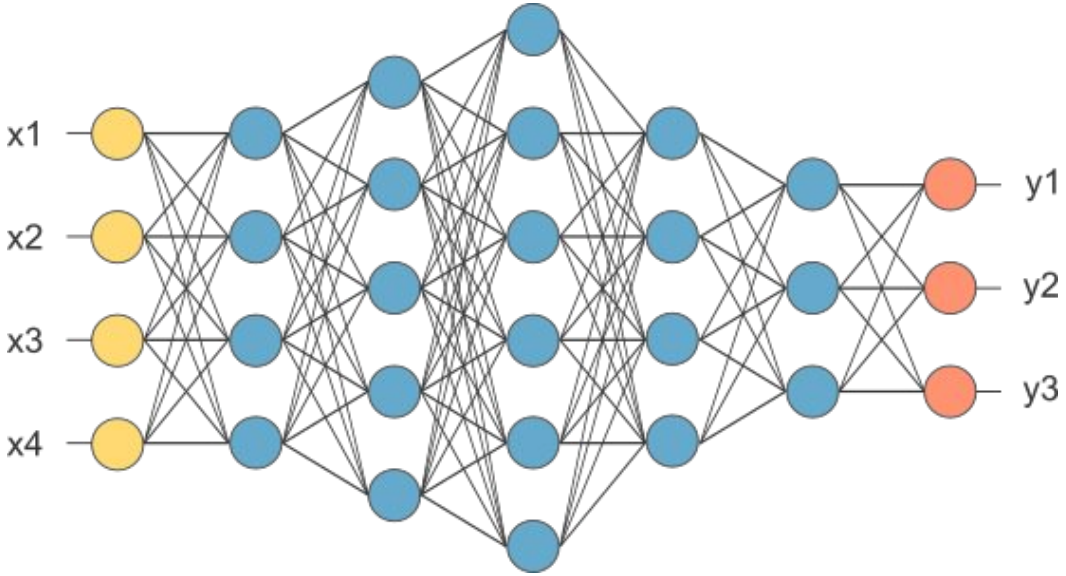
- Given *labeled* training examples:

$$\mathcal{X} = \{(x^{(i)}, y^{(i)}) : i = 1, \ldots, m\}$$

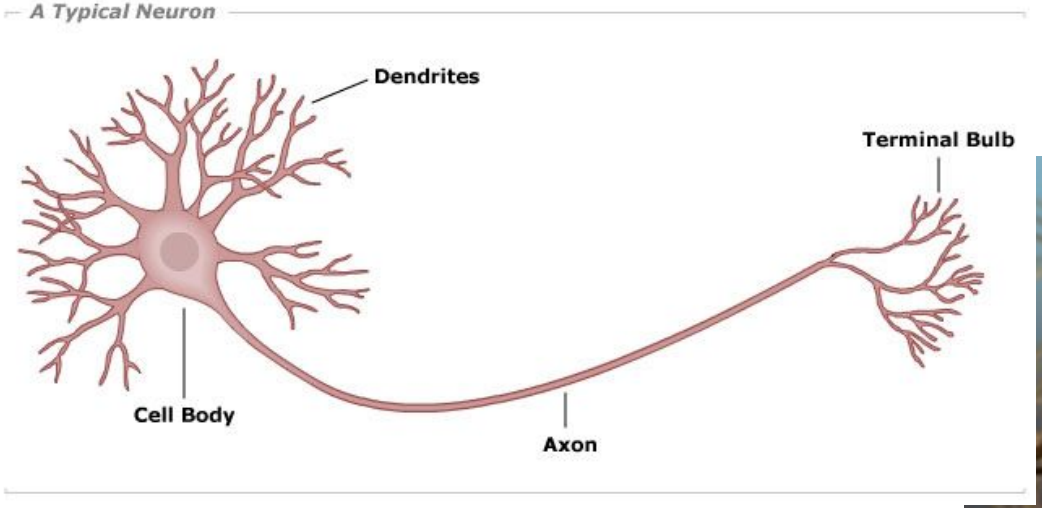
- Goal: find f(x) to predict y from x on training data.
  - Hopefully: learned predictor works on "test" data.



Neural network

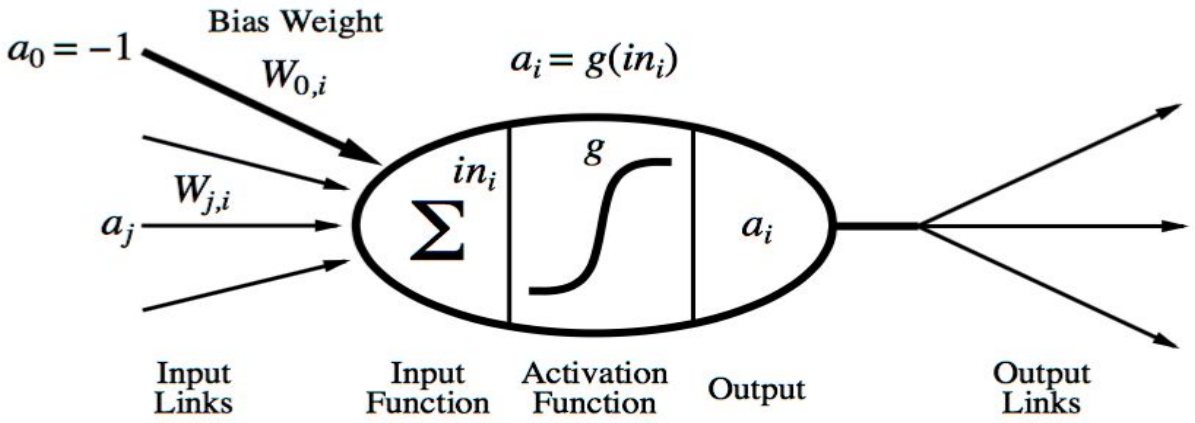


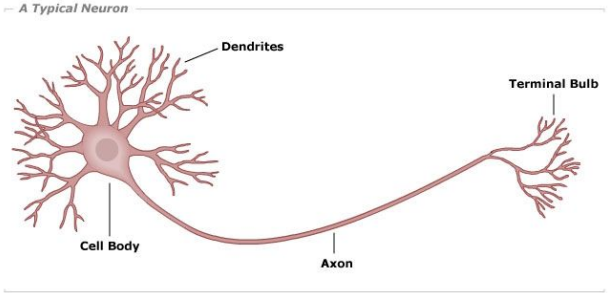
Neurons, Inspiration



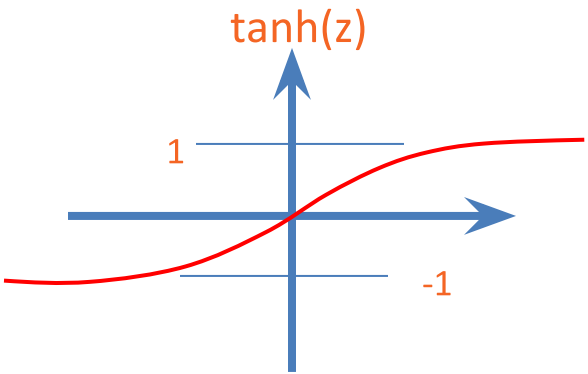
Neuron (Perceptron)

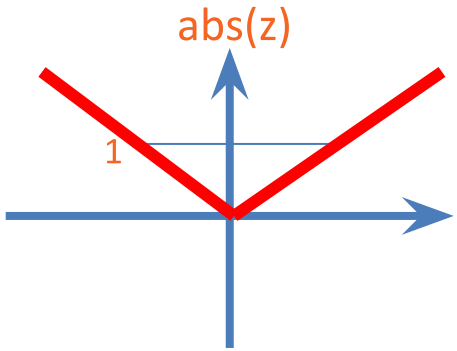
$$a_i \leftarrow g(in_i) = g\left(\Sigma_j W_{j,i} a_j\right)$$

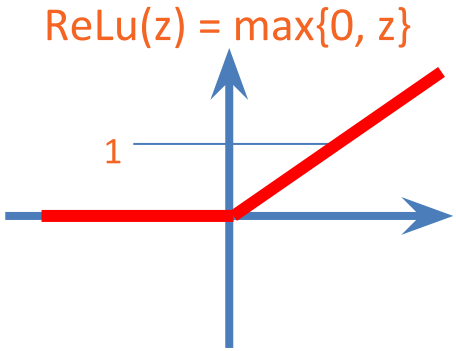




Activations







"Rectified Linear Unit"
→ Increasingly popular.

Why is vision so hard?

"Coffee Mug"

Pixel Intensity

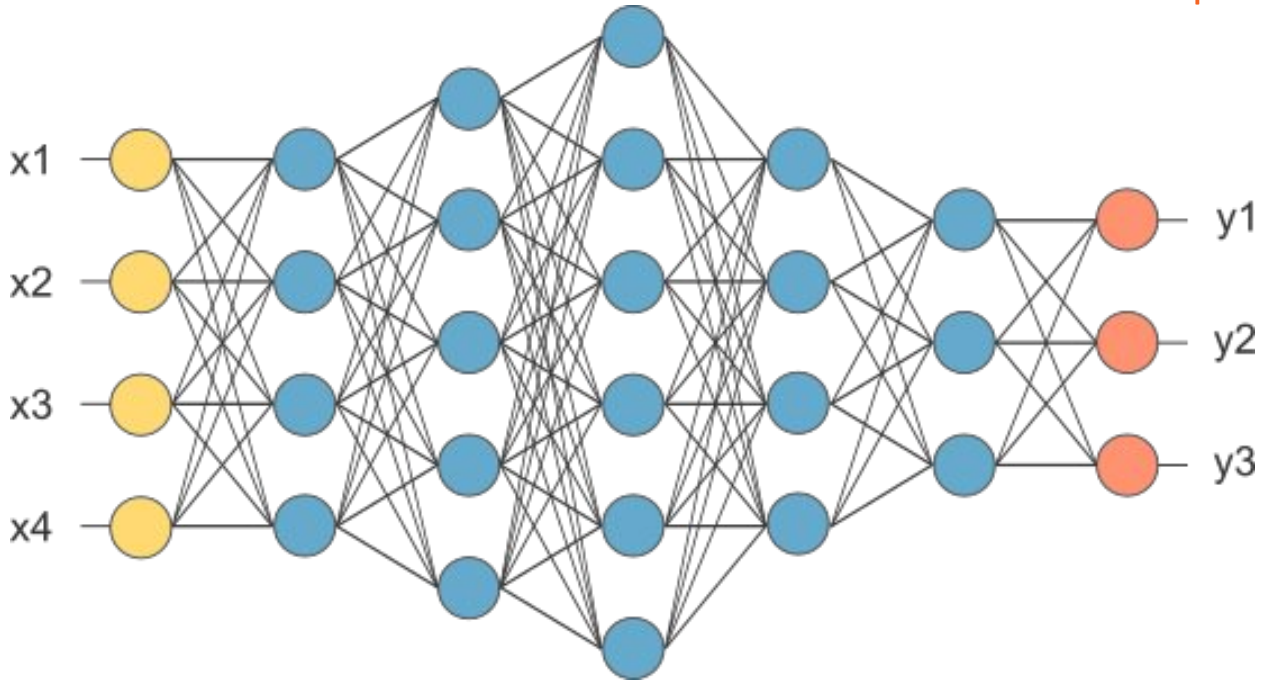
177	153	118	91	85	100	124	145
151	124	93	77	86	115	148	168
115	93	78	83	108	145	177	191
88	79	84	104	136	168	190	197
82	85	103	127	152	170	180	182
91	101	120	138	150	157	159	159
103	114	127	136	140	140	140	141
111	119	126	130	130	129	128	130

Pixel intensity is a very poor representation.

Neural network

- Can stack up several layers:

Must learn multiple stages of internal “representation”.



TensorFlow

- Deep Learning framework developed by Google
- The idea is to give a simple Framework for building Neural Networks
- Python
- C++ Acceleration
- CUDA + GPU acceleration
- Open source

Tip

TensorFlow has amazing tutorials online on YouTube

Installation

- Install Python3.5 (Windows), add Environment variables
- Use pip3 to install packages
- pip3 install --upgrade tensorflow
- pip3 install --upgrade tflearn

Tip

TensorFlow has amazing tutorials online on YouTube

Steps

Build a model

First we need to define our model, using TFLearn classes

Train the model

We will use the MNIST dataset in-order to train our model to recognize digits

Use the model

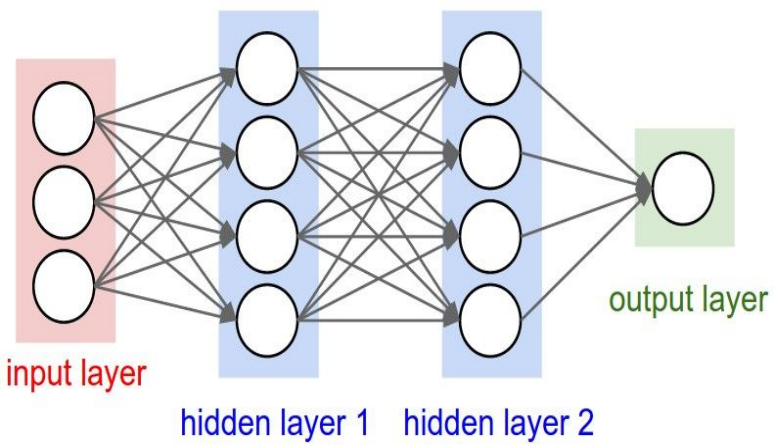
Use our trained model to detect digits.

Building the model

- Input Layer (28*28 = 784)
- Fully Connected (64), tanh
- Fully Connected (64), tanh
- Fully Connected (10), softmax

Tip

Deeper and larger networks tend to give better results, but that is not always the case. Start small and build up.




```
# Building deep neural network
input_layer = tflearn.input_data(shape=[None, 784])
dense1 = tflearn.fully_connected(input_layer, 64, activation='tanh',
                                 regularizer='L2', weight_decay=0.001)
dropout1 = tflearn.dropout(dense1, 0.8)
dense2 = tflearn.fully_connected(dropout1, 64, activation='tanh',
                                 regularizer='L2', weight_decay=0.001)
dropout2 = tflearn.dropout(dense2, 0.8)
softmax = tflearn.fully_connected(dropout2, 10, activation='softmax')
```

Training the model

- Use MNIST dataset - we download it from the web
- 55k Images - labeled and separated to training and validation
- One-hot : Labels are encoded as binary vectors

```
# Data loading and preprocessing
import tflearn.datasets.mnist as mnist
X, Y, testX, testY = mnist.load_data(one_hot=True)
```

- SGD Optimizer: Loss update function (Training)
- Cross Entropy loss: Loss calculation function

```
sgd = tflearn.SGD(learning_rate=0.1, lr_decay=0.96, decay_step=1000)
top_k = tflearn.metrics.Top_k(3)
net = tflearn.regression(softmax, optimizer=sgd, metric=top_k,
                         loss='categorical_crossentropy')
```

Tip

Try our different learning_rates, large rates may cause the network to diverge really fast. Low rates may slow the learning process

Running the session

- Training set: our Training dataset. Images and labels
- Validation set: Small part of the data for validation
- N_epoch: Number of times we train the model

```
# Training
model = tflearn.DNN(net, tensorboard_verbose=0)
model.fit(X, Y, n_epoch=20, validation_set=(testX, testY),
          show_metric=True, run_id="dense_model")
```

Tip

The more epochs we do the better the model becomes, up to a certain point. Then we might risk the situation of over fitting the data. (we dont want that)

Saving the network

- Save a model to a .model file (Saves network + weights)
- Load model weights from a file

```
model.save('mnist.model')
model.load('mnist.model')
```

Tip

Large networks may take a huge space on the disk.

Using the network

- Load a trained model
- Load a new image for detection
- Reshape image to suit our input
- Use the trained model to predict the image label

```
def loadImage(path):
    return np.reshape(Image.open(path).resize((28, 28)), [28*28*3])

image = loadImage("images/test/test2.jpg")
model.predict([image])
```

Tip

Scaling the images too small may reduce the quality of the prediction.