Introduction to Deep Learning

Feng Chen
HPC User Services
LSU HPC & LONI
sys-help@loni.org

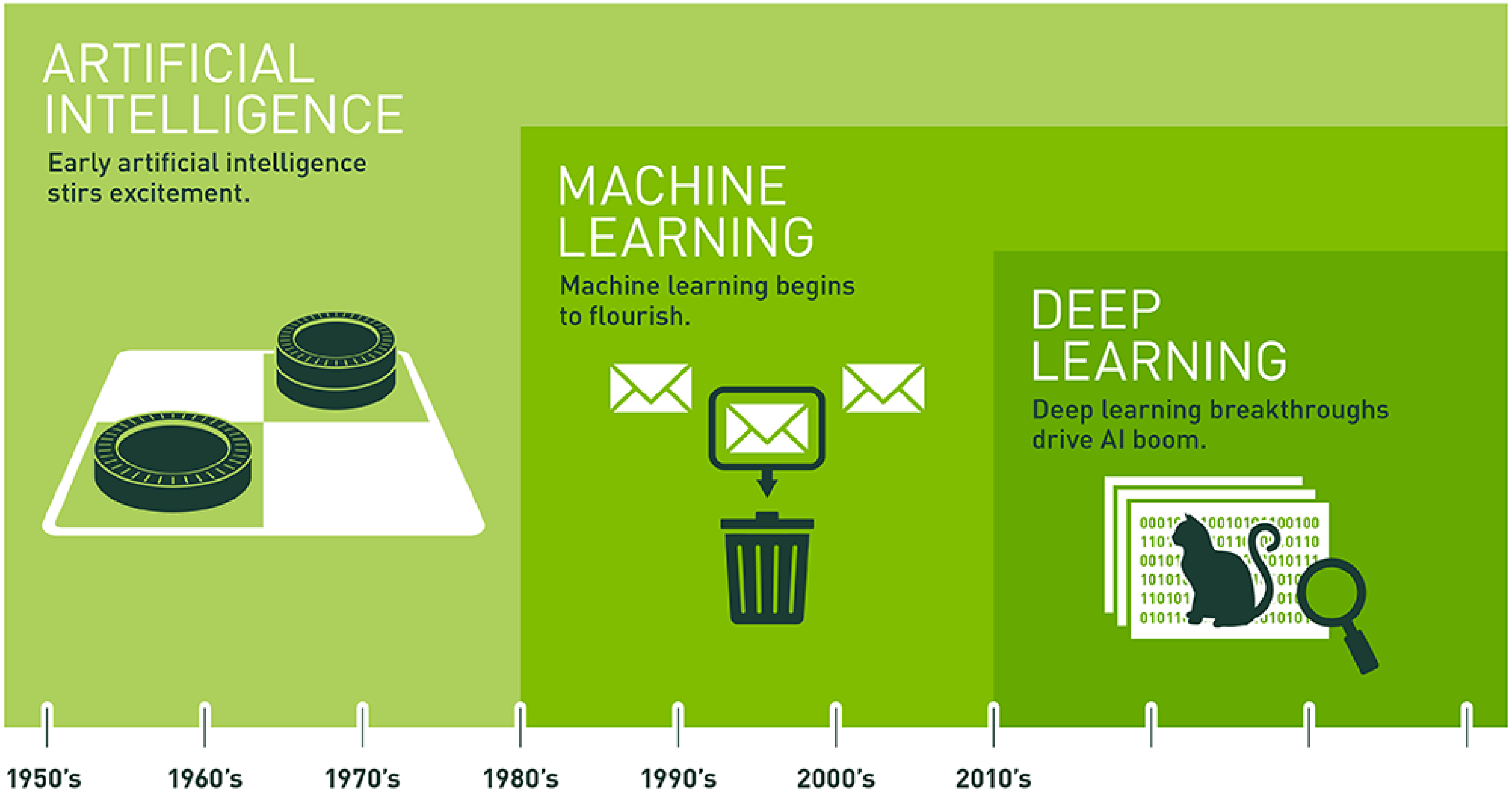
Part of slides referenced from
Nvidia, ***Deep Learning Institute (DLI) Teaching Kit***
Stanford, ***CS231n: Convolutional Neural Networks for Visual Recognition***
Martin Görner, ***Learn TensorFlow and deep learning, without a Ph.D***

Louisiana State University
Baton Rouge
May 28, 2019

Topics To Be Discussed

- **Fundamentals about Machine Learning**
  - What is ***Deep Learning***?
    - What is a (deep) neural network
    - How to train it
- **Build a neural network model using Keras/TensorFlow**
  - MNIST example
    - Softmax classification
    - Cross-entropy cost function
    - A 5 layer deep neural network
    - Dropout
    - Convolutional networks
- **Deep Learning Frameworks**
  - Tensorflow/Keras
  - PyTorch
  - Caffe

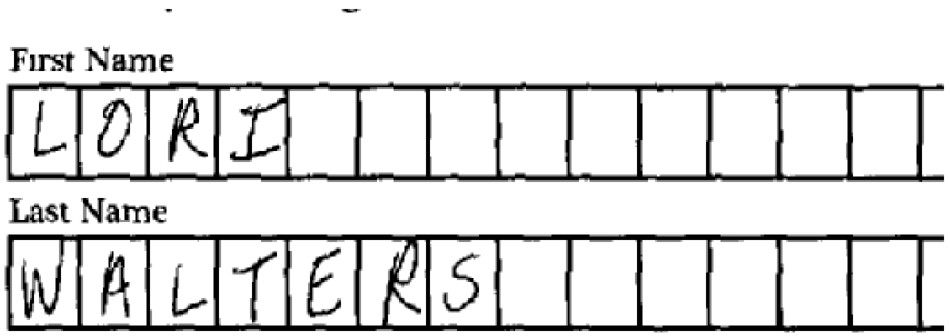
AI, Machine Learning and Deep Learning

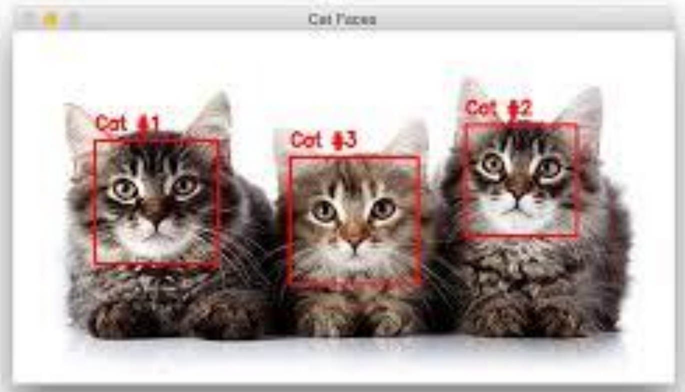

ARTIFICIAL INTELLIGENCE
Early artificial intelligence stirs excitement.
MACHINE LEARNING
Machine learning begins to flourish.
DEEP LEARNING
Deep learning breakthroughs drive AI boom.
1950's
1960's
1970's
1980's
1990's
2000's
2010's

Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

Machine Learning

- **Machine Learning is the science of getting computers to learn, without being explicitly programmed.**
- **Examples are used to train computers to perform tasks that would be difficult to program**


First Name
L O R I
Last Name
W A L T E R S



Applications of Machine Learning

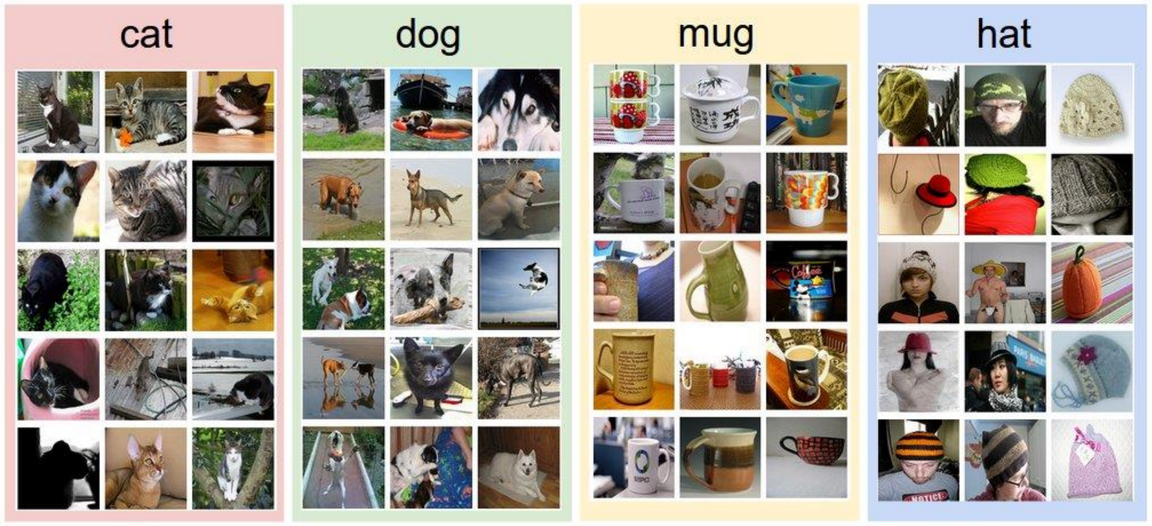
- **Computer Vision (CV)**
  - Image Classification
    - label images with appropriate categories (e.g. Google Photos)
  - Handwriting Recognition
    - convert written letters into digital letters
- **Natural Language Processing (NLP)**
  - Language Translation
    - translate spoken and or written languages (e.g. Google Translate)
  - Speech Recognition
    - convert voice snippets to text (e.g. Siri, Cortana, and Alexa)
- **Autonomous Driving**
  - enable cars to drive

Types of Machine Learning

- **Supervised Learning**
  - Training data is labeled
  - Goal is correctly label new data
- **Unsupervised Learning**
  - Training data is unlabeled
  - Goal is to categorize the observations
- **Reinforcement Learning**
  - Training data is unlabeled
  - System receives feedback for its actions
  - Goal is to perform better actions

Data-driven Approach

- **Instead of trying to specify what every one of the categories of interest look like directly in code, the approach that we will take is not unlike one you would take with a child:**
  - Provide the computer with many examples of each class
  - Develop learning algorithms that look at these examples and learn about the visual appearance of each class.
- **This approach is referred to as a data-driven approach.**

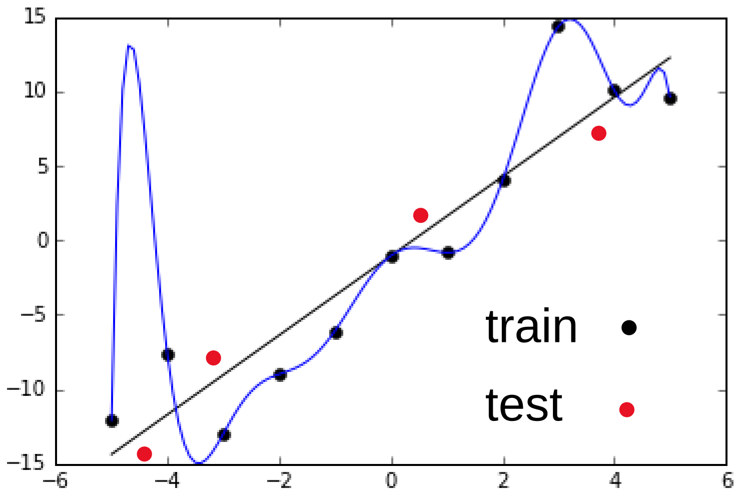


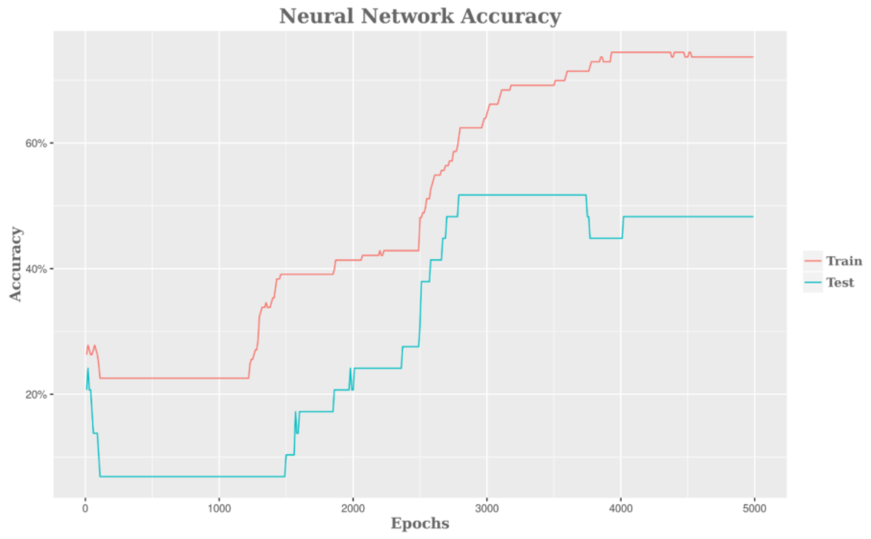
An example training set for four visual categories. In practice we may have thousands of categories and hundreds of thousands of images for each category. ***(From Stanford *CS231n*)**

Training and Test Data

- **Training Data**
  - data used to learn a model
- **Test Data**
  - data used to assess the accuracy of model

- **Overfitting**
  - Model performs well on training data but poorly on test data





Supervised Learning Algorithms

- **Linear Regression**
- **Decision Trees**
- **Support Vector Machines**
- **K-Nearest Neighbor**
- **Neural Networks**
  - Deep Learning is the branch of Machine Learning based on Deep Neural Networks (DNNs, i.e., neural networks composed of more than 1 hidden layer).
  - Convolutional Neural Networks (CNNs) are one of the most popular DNN architectures (so CNNs are part of Deep Learning), but by no means the only one.

Machine Learning Frameworks

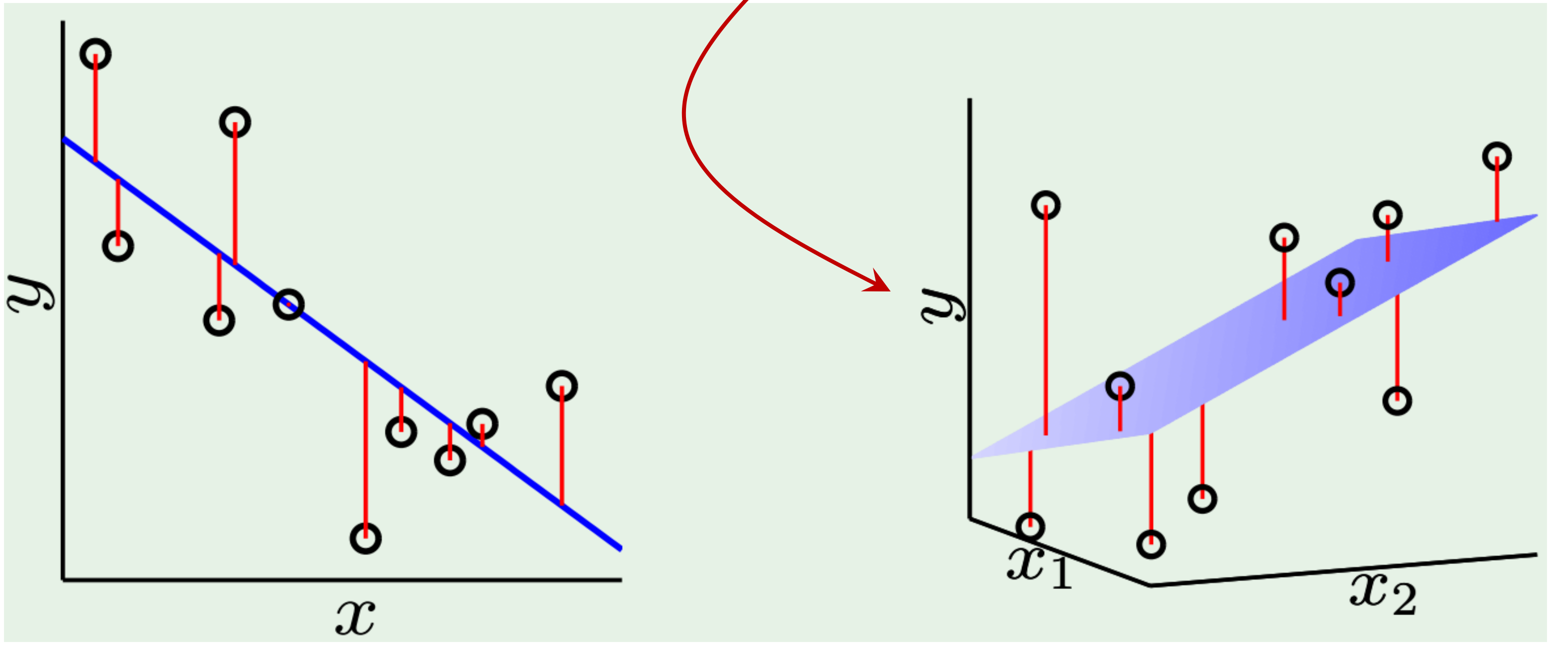
Tool	Uses	Language
Scikit-Learn	Classification, Regression, Clustering	Python
Spark MLlib	Classification, Regression, Clustering	Scala, R, Java
MXNet	Deep learning framework	Python, R, Julia, Scala, Go, Javascript and more
Caffe	Neural Networks	C++, Python
TensorFlow	Neural Networks	Python
PyTorch	Neural Networks	Python

What is Deep Learning

Machine Learning and Deep Learning

Recall From The Least Square Method

- **Start from least square method...** $y = w_1 x_1 + w_2 x_2 + b$
- **Trying to find**
  - ***Parameters* (*w*, *b*)**: minimizes the sum of the squares of the errors
  - ***Errors***: distance between known data points and predictions

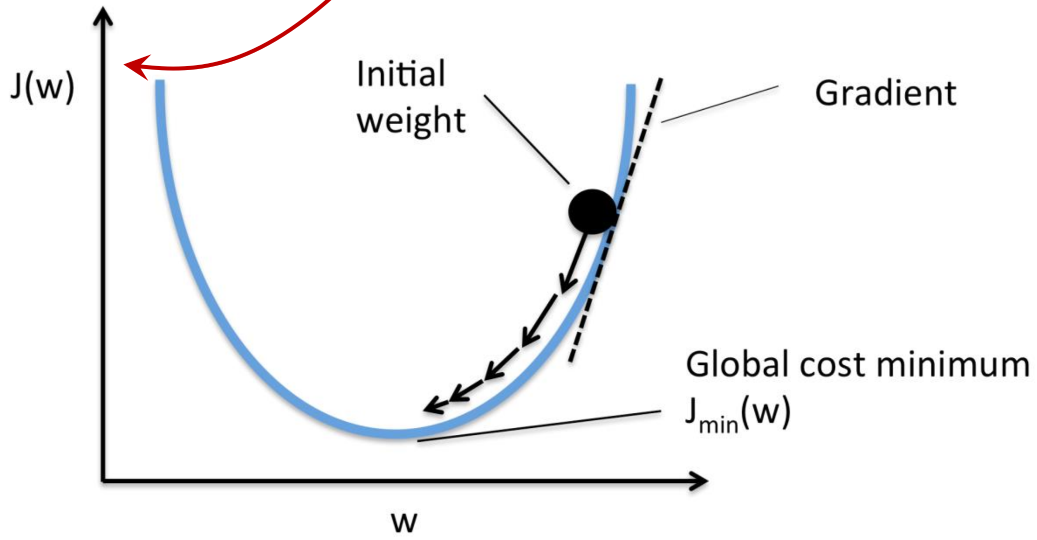


❖ from Yaser Abu-Mustafa "Learning From Data" Lecture 3

To The Machine Learning Language

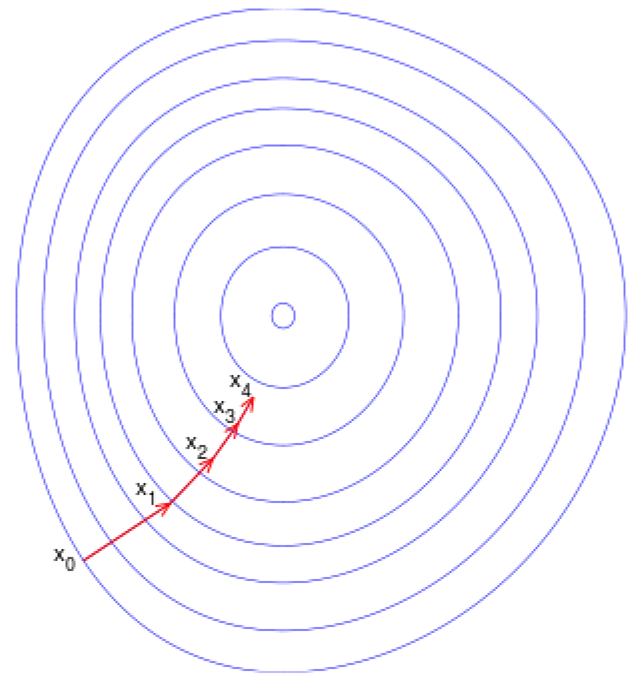
- **Error**
  - Cost Function (Loss): *J*(*w*), *C*, *L*
- **Parameters**
  - Weights and Biases: (*w*, *b*)

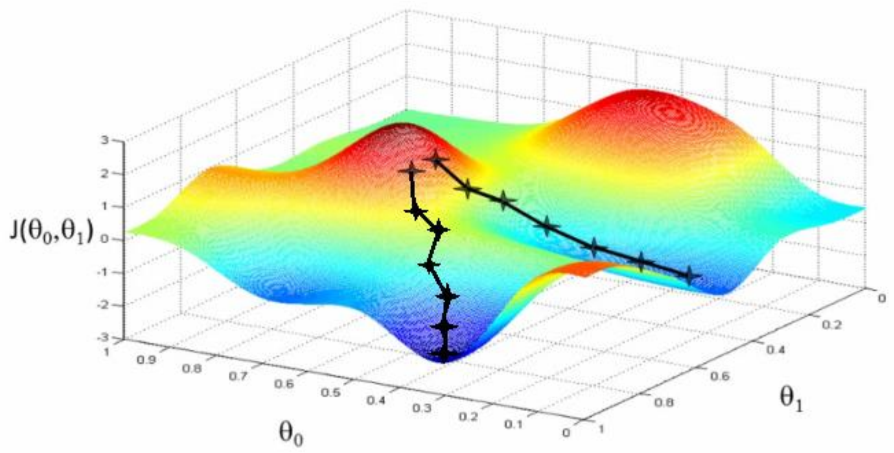
- **Define the cost function of your problem**
- **Find the set of weights that minimizes the cost function (loss)**



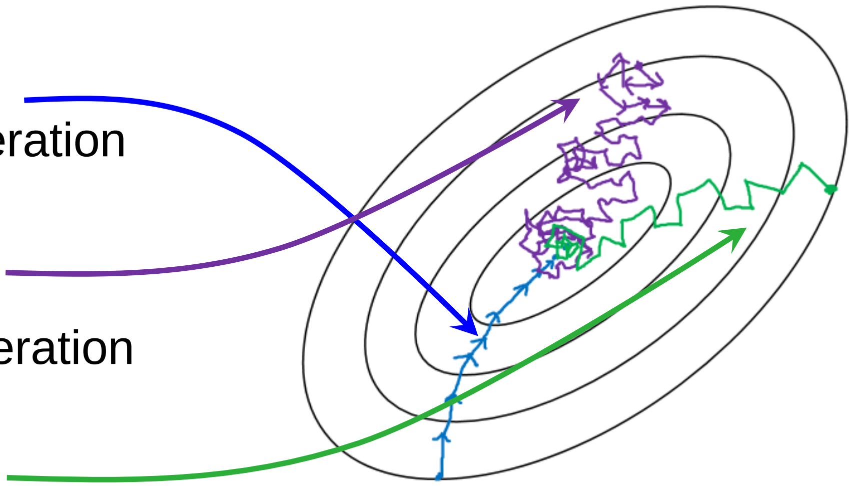
Theory: Gradient Descent

- Gradient descent is a first-order iterative optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point.





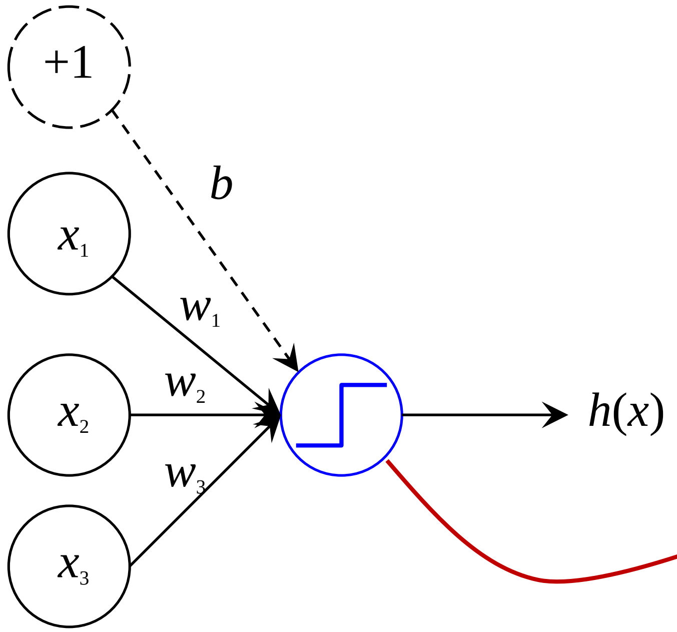
Mini-batch Gradient Descent



- **Batch gradient descent:**
  - Use all examples in each iteration
- **Stochastic gradient descent:**
  - Use one example in each iteration
- **Mini-batch gradient descent**
  - Splits the training dataset into small batches (size *b*) that are used to calculate model error and update model coefficients.
- **In the neural network terminology:**
  - one ***epoch*** consists of one full training cycle on the training set.
  - Using all your batches once is 1 ***epoch***. If you have 10 epochs it mean that you will use all your data 10 times (split in batches).

What is a Neural Network?

- **Start from a perceptron**



x1	age	23
x2	gender	male
x3	annual salary	$30,000
b	threshold	some value

h(x)	Approve credit if: `h(x)>0`

Activation function:

$\sigma(z)=sign(z)$

Denote as: Z

Feature vector: $\mathbf{x}$

$$\mathbf{x}=\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Weight vector: $\mathbf{w}$

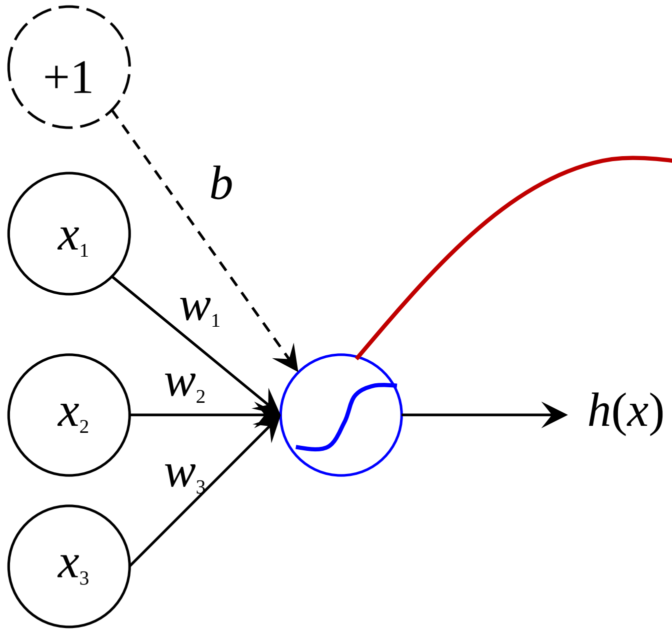
$$\mathbf{w}=\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

$$h(x)=\text{sign}(w_1x_1+w_2x_2+w_3x_3+b)$$
$$=\text{sign}\left(\sum_i w_ix_i+b\right)$$
$$=\text{sign}\left(\mathbf{w}^T\mathbf{x}+b\right)$$

Hypothesis (Prediction: y)

Perceptron To Neuron

- Replace the sign to sigmoid



Activation function:
$\sigma(z) = sigmoid(z)$

$$h(x) = \text{sigmoid}\left(w_1x_1 + w_2x_2 + w_3x_3 + b\right)$$
$$= \text{sigmoid}\left(\sum_i w_ix_i + b\right)$$
$$= \text{sigmoid}\left(\mathbf{w}^T\mathbf{x} + b\right)$$

Feature vector: $\mathbf{x}$ Weight vector: $\mathbf{w}$

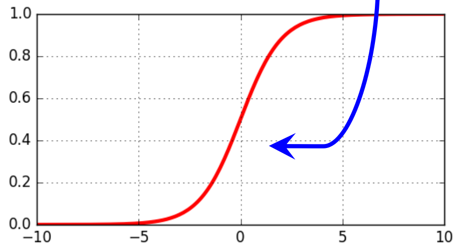
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

$$z = \mathbf{w}^T\mathbf{x} + b$$
$$y = h(x) = \sigma(z)$$

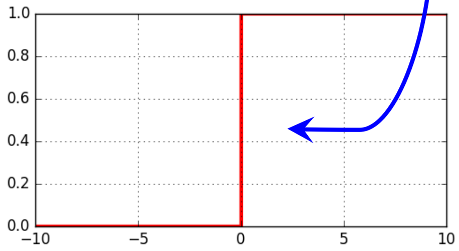
Sigmoid Neurons

- **Sigmoid activation Function**
  - In the field of Artificial Neural Networks, the sigmoid function is a type of activation function for artificial neurons.

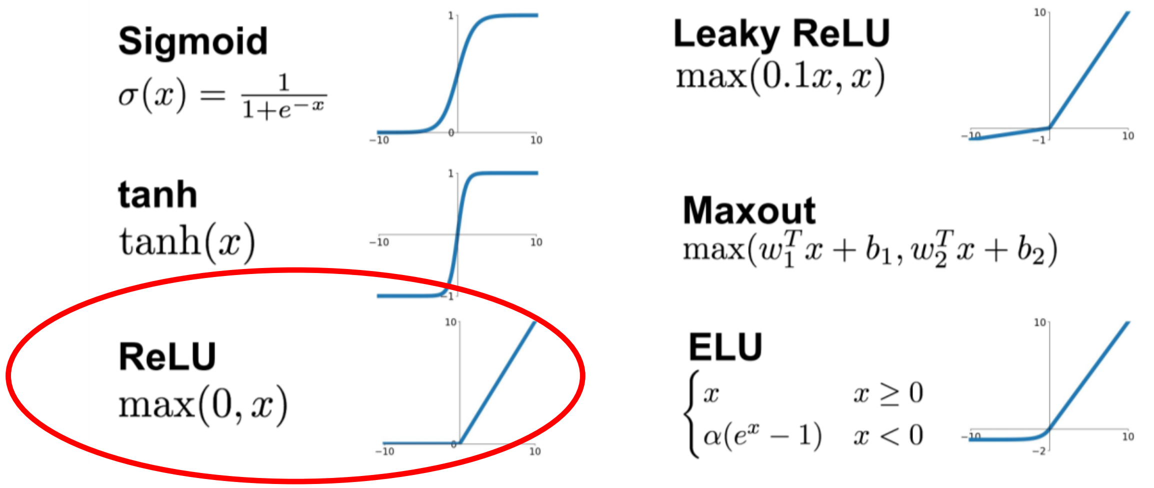
$$\sigma(z) = \frac{1}{1+e^{-z}}$$



$$\sigma(z) = sign(z)$$

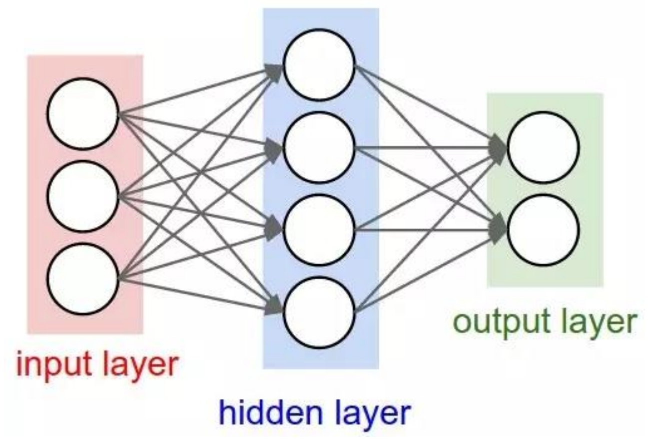


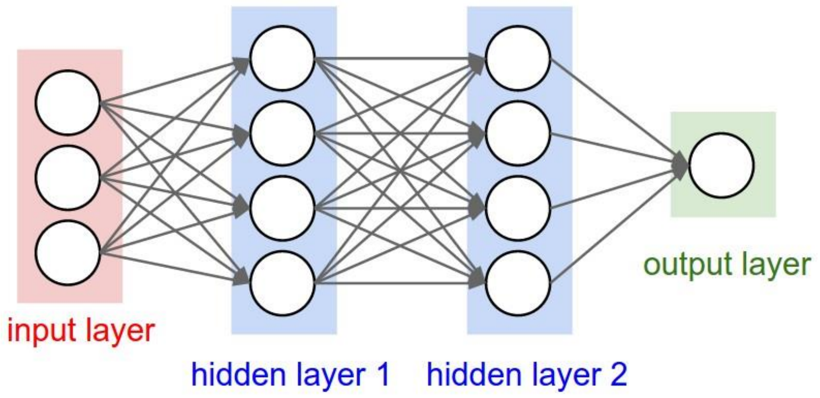
- **There are many other activation functions. (We will touch later.)**



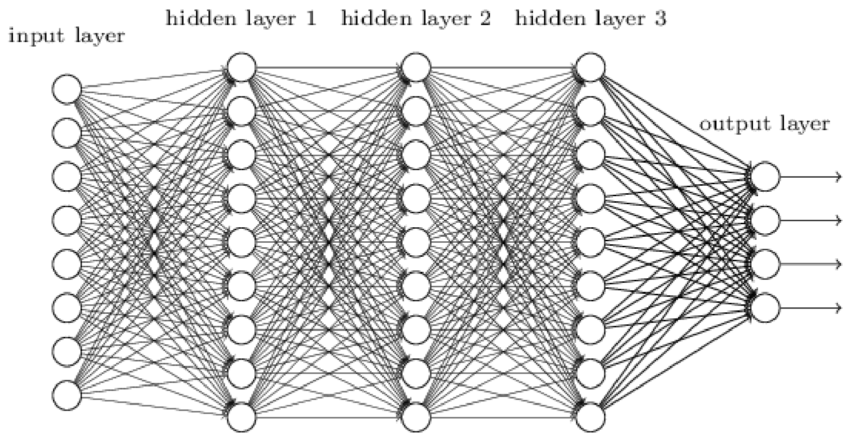
Network Of Neurons

- A complex network of neurons could make quite subtle decisions

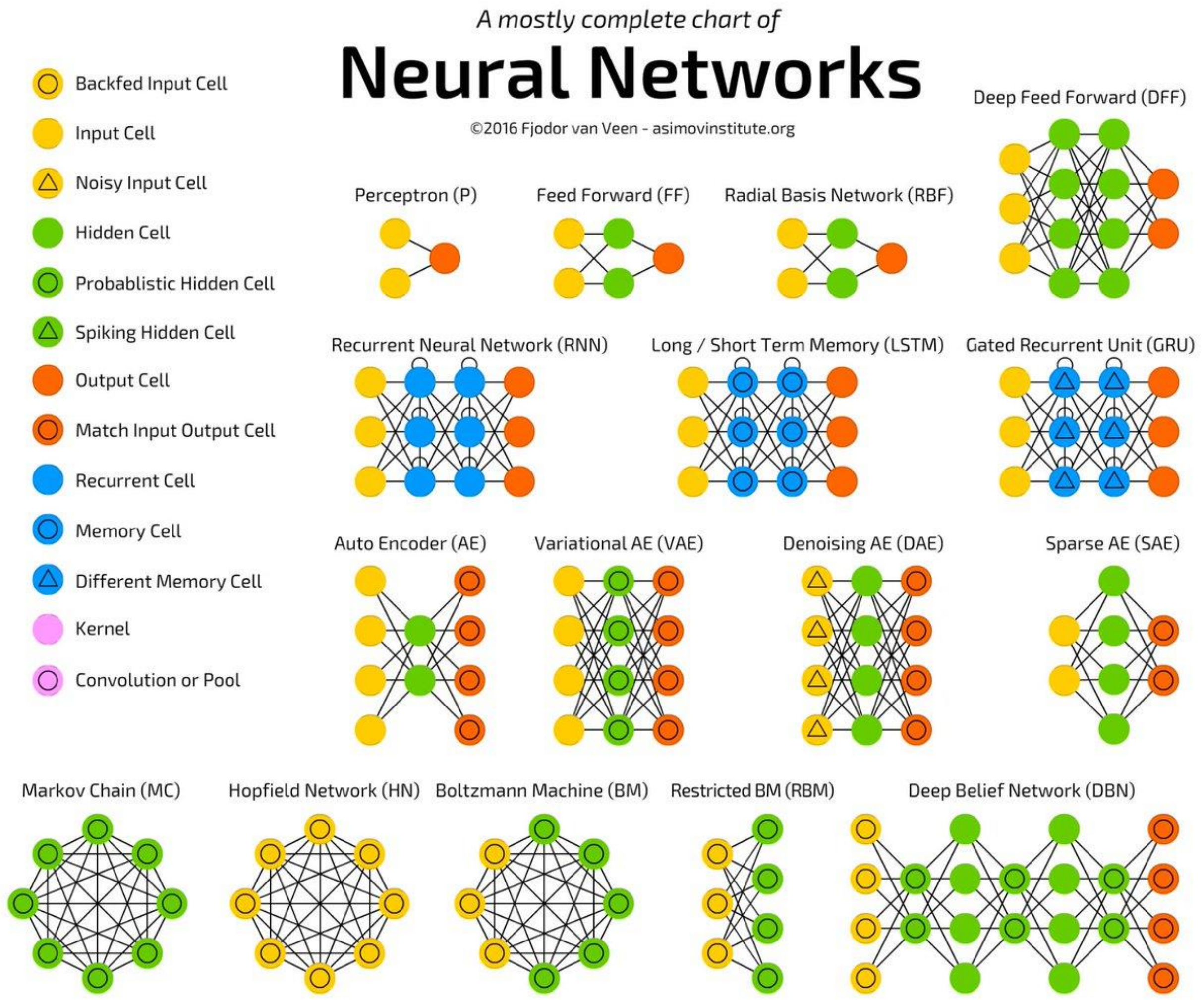




- Deep Neuron Network: Number of hidden layers >1



Types of Neural Networks



Ref: http://www.asimovinstitute.org/neural-network-zoo/

How to Train DNN?

- **Backward Propagation**
  - The backward propagation of errors or backpropagation, is a common method of training artificial neural networks and used in conjunction with an optimization method such as gradient descent.

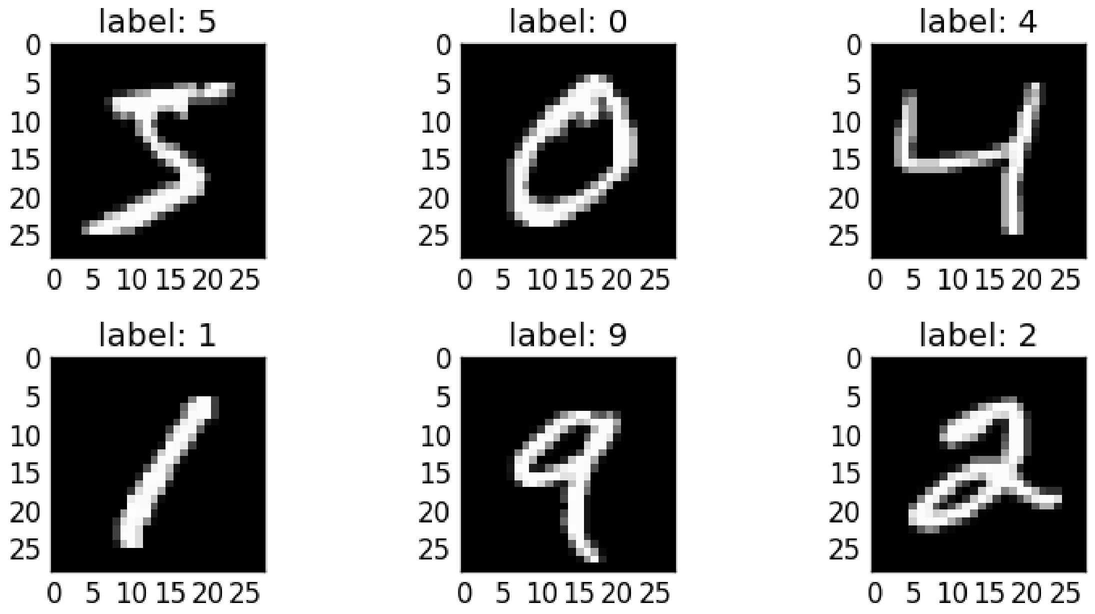
- **Deep Neural Networks are hard to train**
  - learning machines with lots of (typically in range of million) parameters
  - Unstable gradients issue
    - Vanishing gradient problem
    - Exploding gradient problem
  - Choice of network architecture and other hyper-parameters is also important.
  - Many factors can play a role in making deep networks hard to train
  - Understanding all those factors is still a subject of ongoing research

Deep Learning Example

Hello World of Deep Learning: Recognition of MNIST

Introducing the MNIST problem

- **MNIST (Mixed National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems.**
- **It consists of images of handwritten digits like these:**



- **The MNIST database contains 60,000 training images and 10,000 testing images.**

Example Problem - MNIST

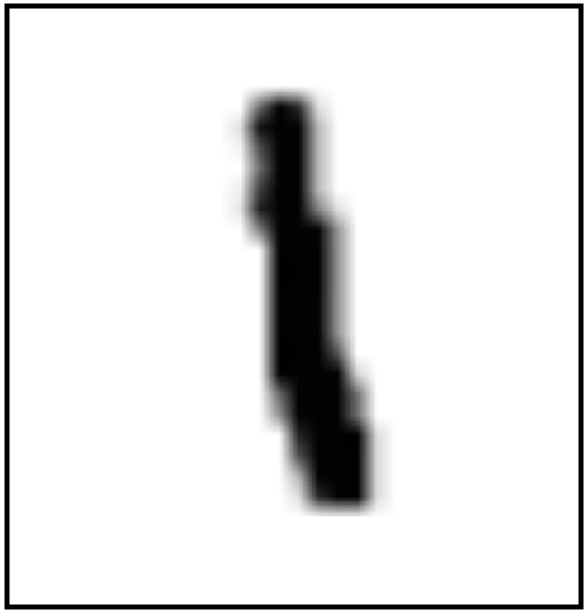
- Recognizes handwritten digits.
- We uses the MNIST dataset, a collection of 60,000 labeled digits that has kept generations of PhDs busy for almost two decades. You will solve the problem with less than 100 lines of Python/Keras/TensorFlow code.
- We will gradually enhance the neural network to achieve above 99% accuracy by using the mentioned techniques.

Steps for MNIST

- **Understand the MNIST data**
- **Softmax regression layer**
- **The cost function**

The MNIST Data

- Every MNIST data point has two parts: an image of a handwritten digit and a corresponding label. We'll call the images "x" and the labels "y". Both the training set and test set contain images and their corresponding labels;
- Each image is 28 pixels by 28 pixels. We can interpret this as a big array of numbers:

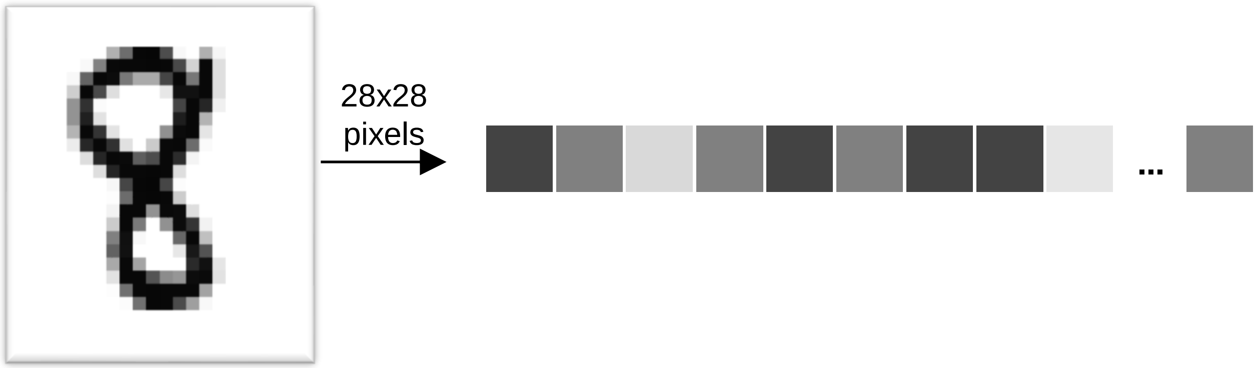


$$\simeq \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .6 & .8 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .5 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .7 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .9 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .3 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

LSU INFORMATION TECHNOLOGY SERVICES

One Layer NN for MNIST Recognition

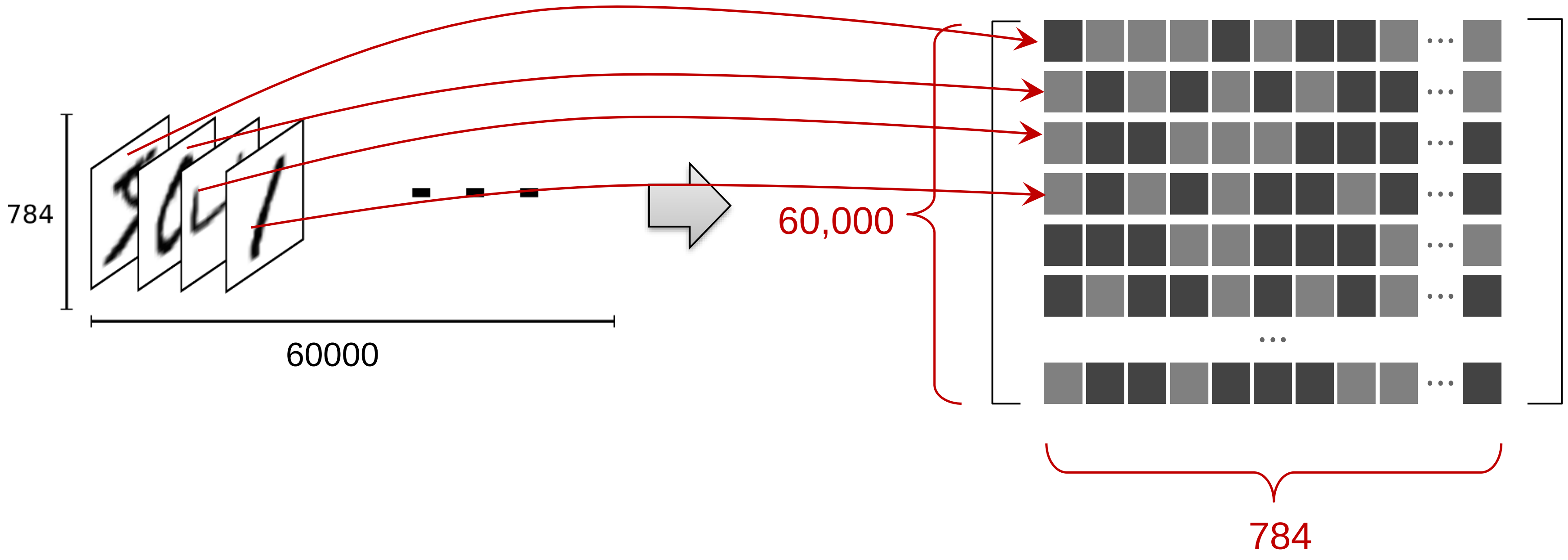
- We will start with a very simple model, called Softmax Regression.
- We can flatten this array into a vector of 28x28 = 784 numbers. It doesn't matter how we flatten the array, as long as we're consistent between images.
- From this perspective, the MNIST images are just a bunch of points in a 784-dimensional vector space.



❖ **What are we missing here?**

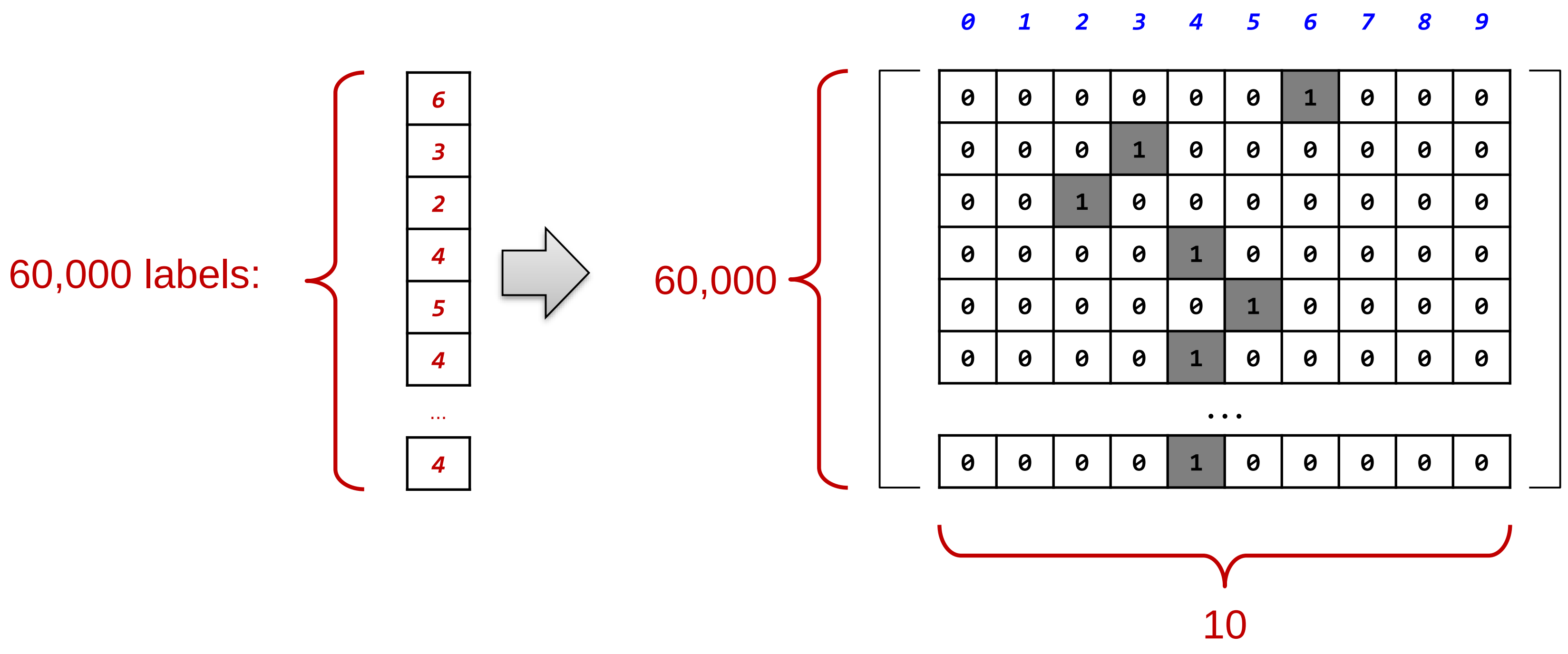
Result of the Flatten Operation

- The result is that the training images is a matrix (tensor) with a shape of `[60000, 784]`.
- The first dimension is an index into the list of images and the second dimension is the index for each pixel in each image.
- Each entry in the tensor is a pixel intensity between 0 and 1, for a particular pixel in a particular image.



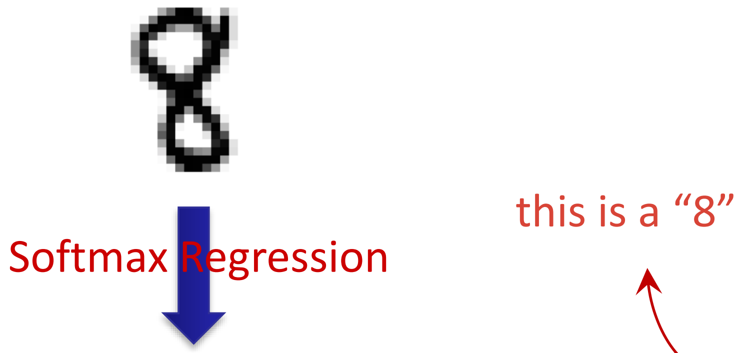
One-hot Vector (One vs All)

- For the purposes of this tutorial, we label the y's as "one-hot vectors".
- A one-hot vector is a vector which is 0 in most dimensions, and 1 in a single dimension.
- How to label an "8"?
  - `[0,0,0,0,0,0,0,0,1,0]`
- What is the dimension of our y matrix (tensor)?



Softmax Regressions

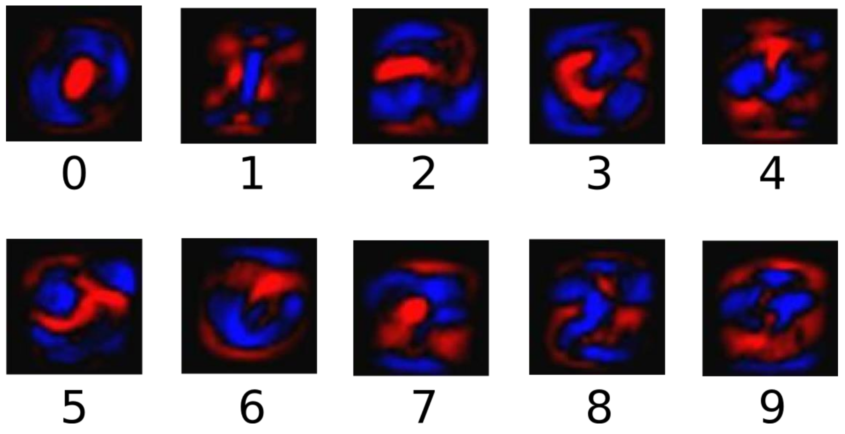
- Every image in MNIST is of a handwritten digit between 0 and 9.
- So there are only ten possible things that a given image can be. We want to be able to look at an image and give the probabilities for it being each digit.
- For example, our model might look at a picture of an eight and be 80% sure it's an 8, but give a 6% chance to it being a 4 (because of the top loop) and a bit of probability to all the others because it isn't 100% sure.



0	1	2	3	4	5	6	7	8	9
0.04	0.02	0.01	0.01	0.06	0.03	0.01	0.01	0.80	0.01

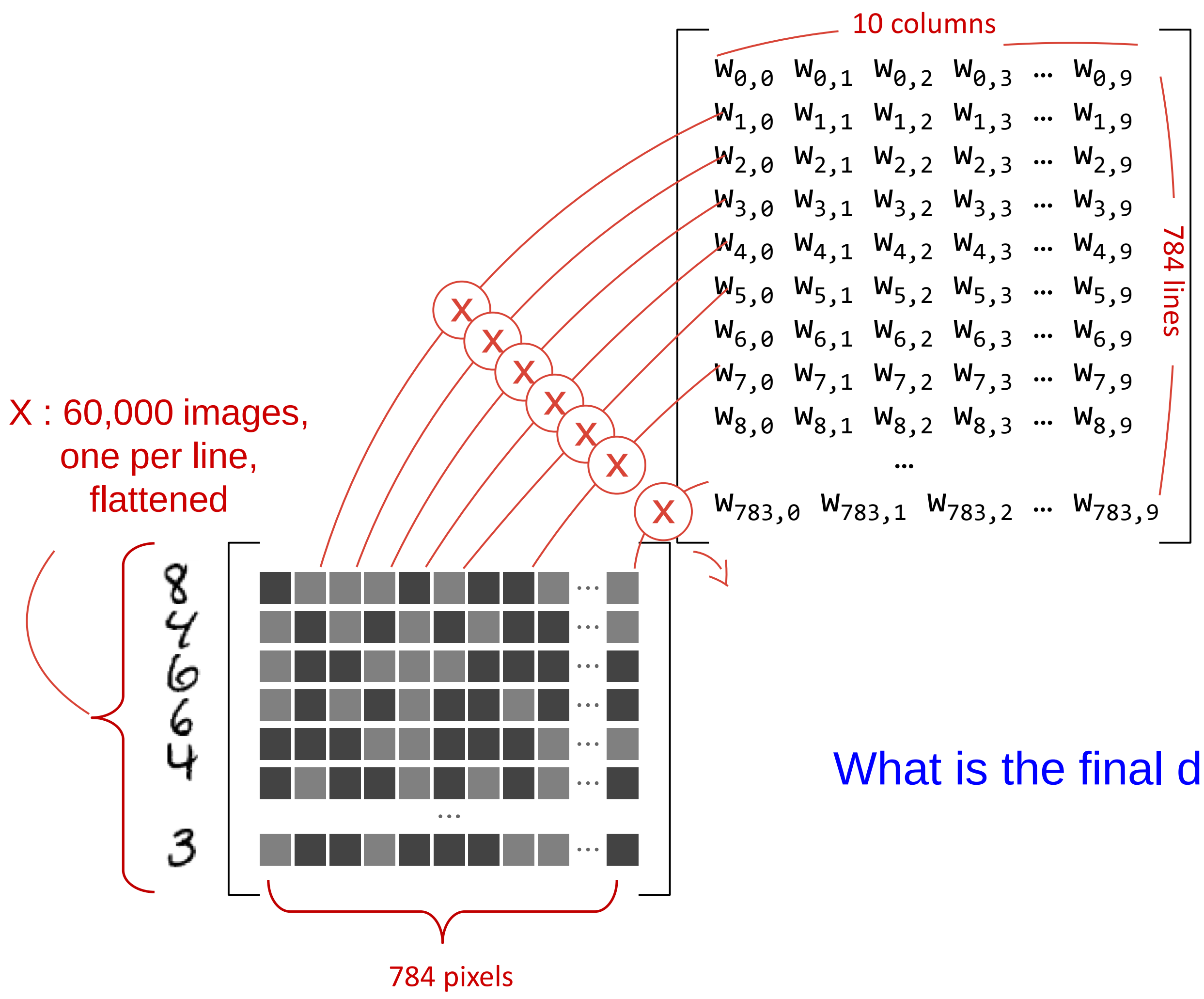
2 steps in softmax regression - Step 1

- **Step 1: Add up the evidence of our input being in certain classes.**
  - Do a weighted sum of the pixel intensities. The weight is negative if that pixel having a high intensity is evidence against the image being in that class, and positive if it is evidence in favor.



$$z_i = \sum_j W_{i,j} x_j + b_i$$

Matrix Representation of softmax layer


10 columns
784 lines
X : 60,000 images, one per line, flattened
784 pixels
W0,0 W0,1 W0,2 W0,3 … W0,9
W783,0 W783,1 W783,2 … W783,9

broadcast

$$Y = \mathrm{softmax}(X \cdot W + b)$$

What is the final dimension for *X*W*?

2 steps in softmax regression - Step 2

- **Step 2: Convert the evidence tallies into our predicted probabilities y using the "softmax" function:**

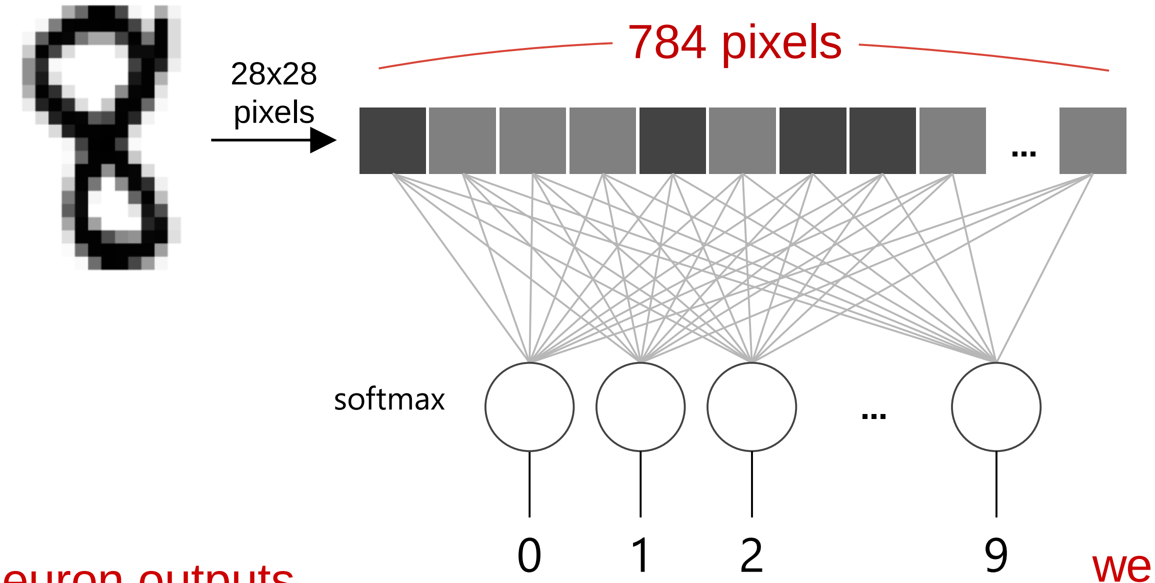
$$h(\mathbf{x}_i) = \text{softmax}(z_i) = \text{softmax}\left(\sum_j W_{i,j} x_j + b_i\right)$$

- **Here softmax is serving as an "activation" function, shaping the output of our linear function a probability distribution over 10 cases, defined as:**

$$\text{softmax}(z_i) = \text{normalize}(\exp(z)) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

The softmax layer

- **The output from the softmax layer is a set of probability distribution, positive numbers which sum up to 1.**



neuron outputs

weighted sum of all pixels + biases

$$\mathrm{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Softmax on a batch of images

- **More compact representation for “softmaxing” on all the images**

Predictions Images Weights Biases

`Y[60000, 10]` `[60000, 784]` `W[784,10]` `b[10]`

$$Y = \text{softmax}(X \cdot W + b)$$

applied on each line

matrix multiply

broadcast on all lines

The Cross-Entropy Cost Function

- **For classification problems, the Cross-Entropy cost function works better than quadratic cost function.**
- **We define the cross-entropy cost function for the neural network by:**

0	0	0	0	0	0	1	0	0	0

"one-hot" encoded ground truth "6"

Cross entropy

$$C = -\sum_i y_i' \cdot \log P\left(Y = y_i \mid X = x_i\right)$$

this is a "6"

computed probabilities

0	1	2	3	4	5	6	7	8	9
0.01	0.01	0.01	0.01	0.01	0.01	**0.90**	0.01	0.02	0.01

Short Summary

- **How MNIST data is organized**
  - X:
    - Flattened image pixels matrix
  - Y:
    - One-hot vector
- **Softmax regression layer**
  - Linear regression
  - Output probability for each category
- **Cost function**
  - Cross-entropy

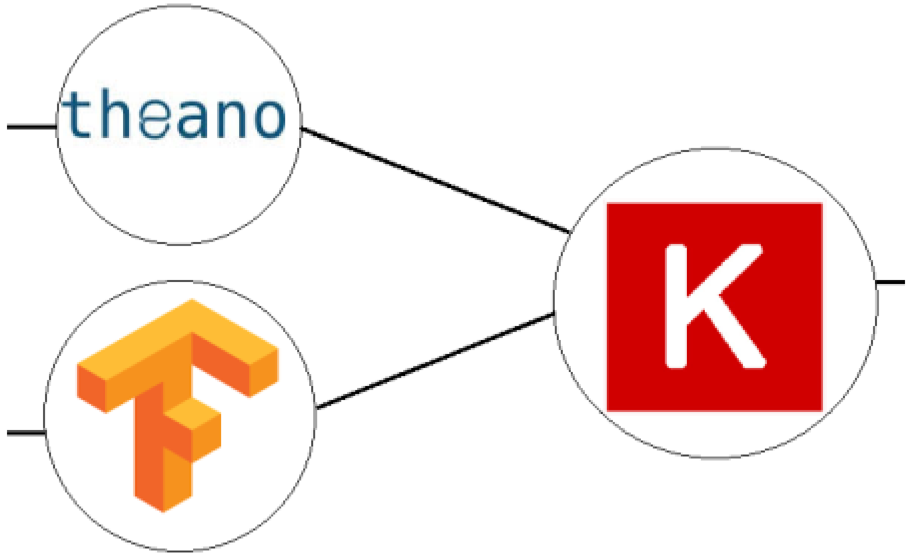
❖ **How to implement them?**

Deep Learning Example

Implementation in Keras/Tensorflow

Few Words about Keras, Tensorflow and Theano

- **Keras is a high-level neural networks library, written in Python and capable of running on top of either TensorFlow or Theano.**
- **TensorFlow is an open source software library for numerical computation using data flow graphs.**
- **Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently.**



Introducing Keras

- **Keras is a high-level neural networks library,**
- **Written in Python and capable of running on top of either TensorFlow or Theano.**
- **It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.**
- **See more at: https://github.com/fchollet/keras**

Typical Code Structure

- **Load the dataset (MNIST)**
- **Build the Neural Network/Machine Learning Model**
- **Train the model**

Software Environment

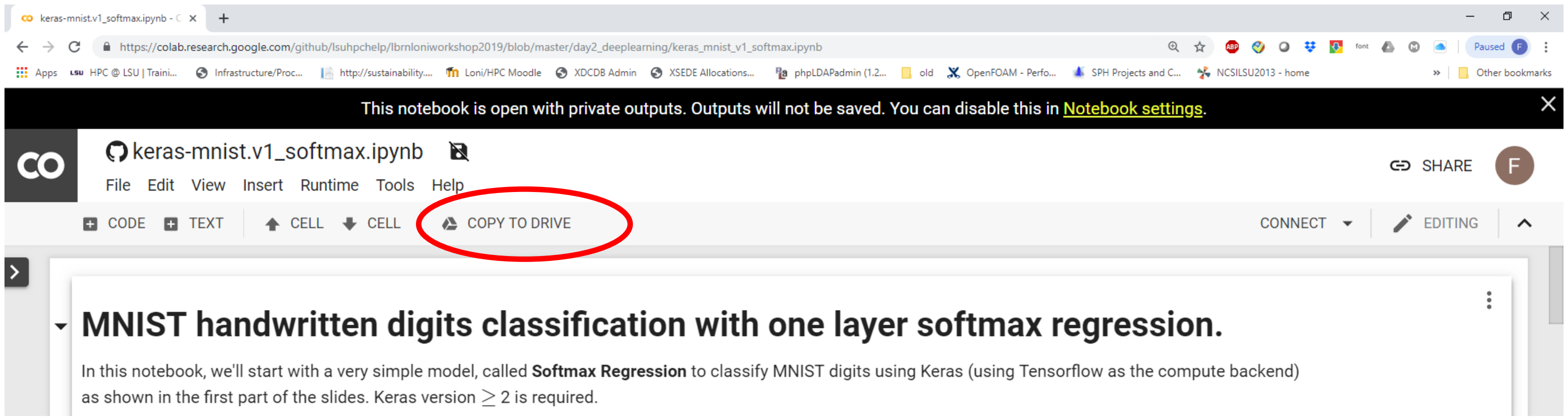
- **What you'll need**
  - Python 2 or 3 (Python 3 recommended)
  - TensorFlow and Keras
  - Matplotlib (Python visualization library)
- **We will use the Google Colaboratory**
  - Colaboratory is a research tool for machine learning education and research. It’s a Jupyter notebook environment that requires no setup to use. You can refer to: https://research.google.com/colaboratory/faq.html for more information.
  - See next slide for starting the Colab notebook

Open Colab Notebook from Github

- **Open the below link:**
  - https://github.com/lsuhpchelp/lbrnloniworkshop2019/blob/master/day2_deeplearning/keras_mnist_v1_softmax.ipynb
- **Or navigate yourself in the github repo:**
  - https://github.com/lsuhpchelp/lbrnloniworkshop2019
  - Select "`day2_deeplearning > keras_mnist_v1_softmax.ipynb`"
- **Click the "Open in Colab" link:**

  Open in Colab

- **After the Colab notebook is laid out, you need one more step, save the Colab notebook to your google drive by "COPY TO DRIVE", or you will be editing the notebook in "Playground" (read only) mode:**


keras-mnist.v1_softmax.ipynb
https://colab.research.google.com/github/lsuhpchelp/lbrnloniworkshop2019/blob/master/day2_deeplearning/keras_mnist_v1_softmax.ipynb
keras-mnist.v1_softmax.ipynb
File Edit View Insert Runtime Tools Help
SHARE
CODE TEXT CELL CELL COPY TO DRIVE
CONNECT EDITING

This notebook is open with private outputs. Outputs will not be saved. You can disable this in Notebook settings.

MNIST handwritten digits classification with one layer softmax regression.

In this notebook, we'll start with a very simple model, called **Softmax Regression** to classify MNIST digits using Keras (using Tensorflow as the compute backend) as shown in the first part of the slides. Keras version ≥ 2 is required.

Keras - Initialization

```
# import necessary modules
# The Sequential model is a linear stack of layers.
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D, MaxPooling2D
from keras.utils import np_utils
from keras import backend as K
```

Load The MNIST Dataset

```
# load the mnist dataset
import cPickle
import gzip
f = gzip.open('mnist.pkl.gz', 'rb')
# load the training and test dataset
# download https://s3.amazonaws.com/img-datasets/mnist.pkl.gz
# to use in this tutorial
X_train, y_train, X_test, y_test = cPickle.load(f)
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

Output of the print line:

```
(60000, 28, 28) (60000,) (10000, 28, 28) (10000,)
```

Preprocessing the MNIST Dataset

```
# Flatten the image to 1D
X_train = X_train.reshape(X_train.shape[0], img_rows*img_cols)
X_test = X_test.reshape(X_test.shape[0], img_rows*img_cols)
input_shape = (img_rows*img_cols,)
```

Flatten 28x28 image to 1D

```
# convert all data to 0.0-1.0 float values
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
```

All grayscale values to 0.0-1.0

```
# convert class vectors to binary class matrices
Y_train = np_utils.to_categorical(y_train, nb_classes)
Y_test = np_utils.to_categorical(y_test, nb_classes)
```

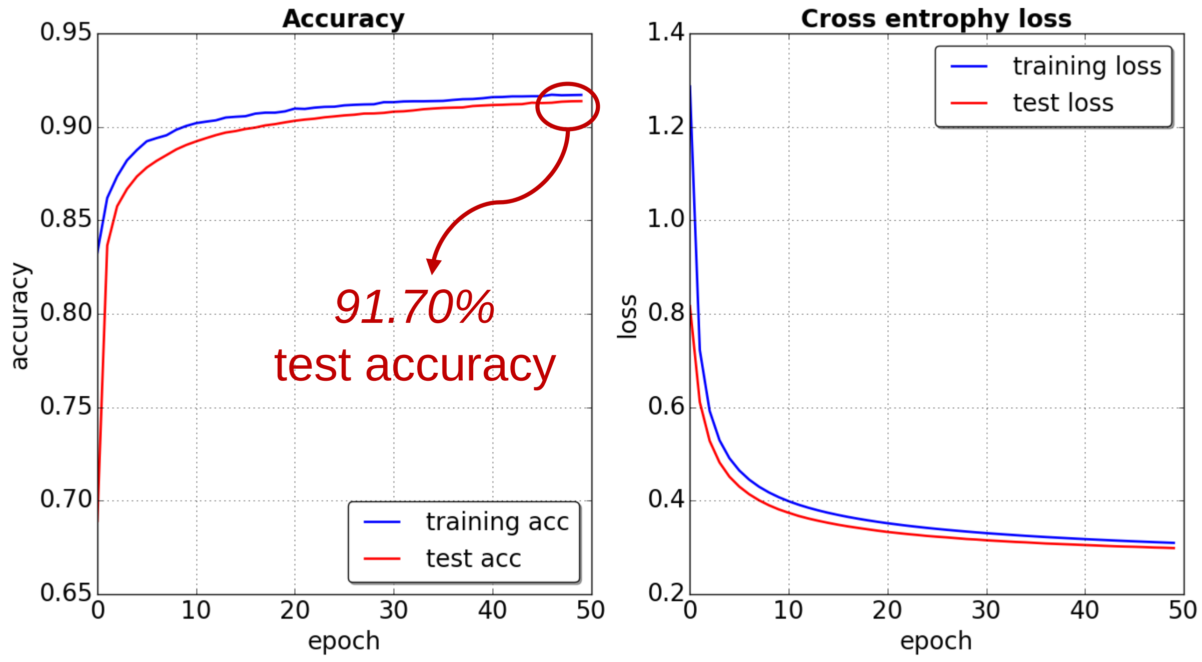
One-hot encoding

Build The First softmax Layer

```
# The Sequential model is a linear stack of layers in Keras
model = Sequential()
#build the softmax regression layer
model.add(Dense(nb_classes,input_shape=input_shape))
model.add(Activation('softmax'))
                                    nb_classes=10           input_shape=(784,)
# Before training a model,
# configure the learning process via the compile method.
# using the cross-entropy loss function (objective)
model.compile(loss='categorical_crossentropy',
              #using the stochastic gradient descent (SGD)
              optimizer='sgd',
              # using accuracy to judge the performance of your model
              metrics=['accuracy'])
# fit the model, the training process
h = model.fit(X_train, Y_train, batch_size=batch_size, nb_epoch=nb_epoch,
          verbose=1, validation_data=(X_test, Y_test))
```

Results Of The First softmax Regression

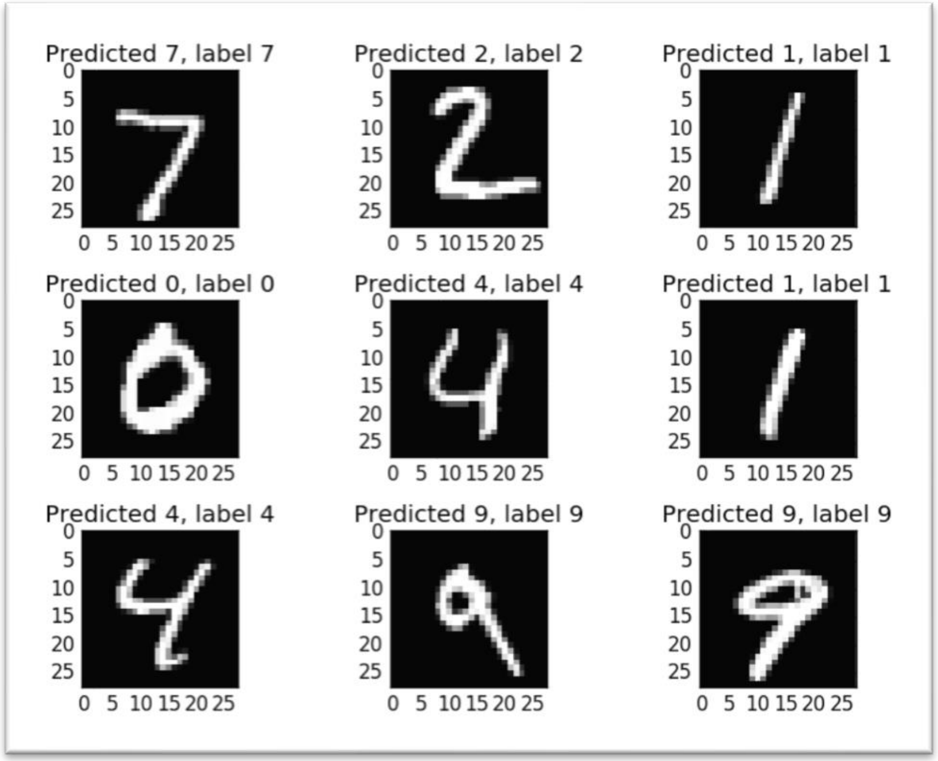
- **Training accuracy vs Test accuracy, loss function**
- **We reach a test accuracy at *91.7%***

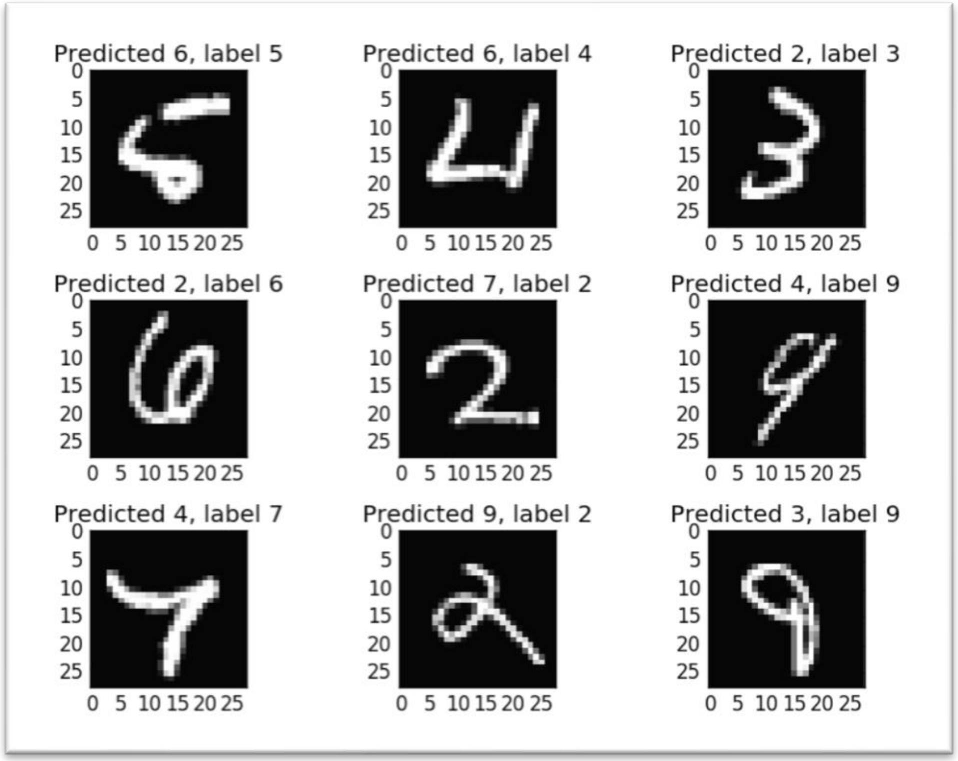


Review The Classified Results

Correctly classified

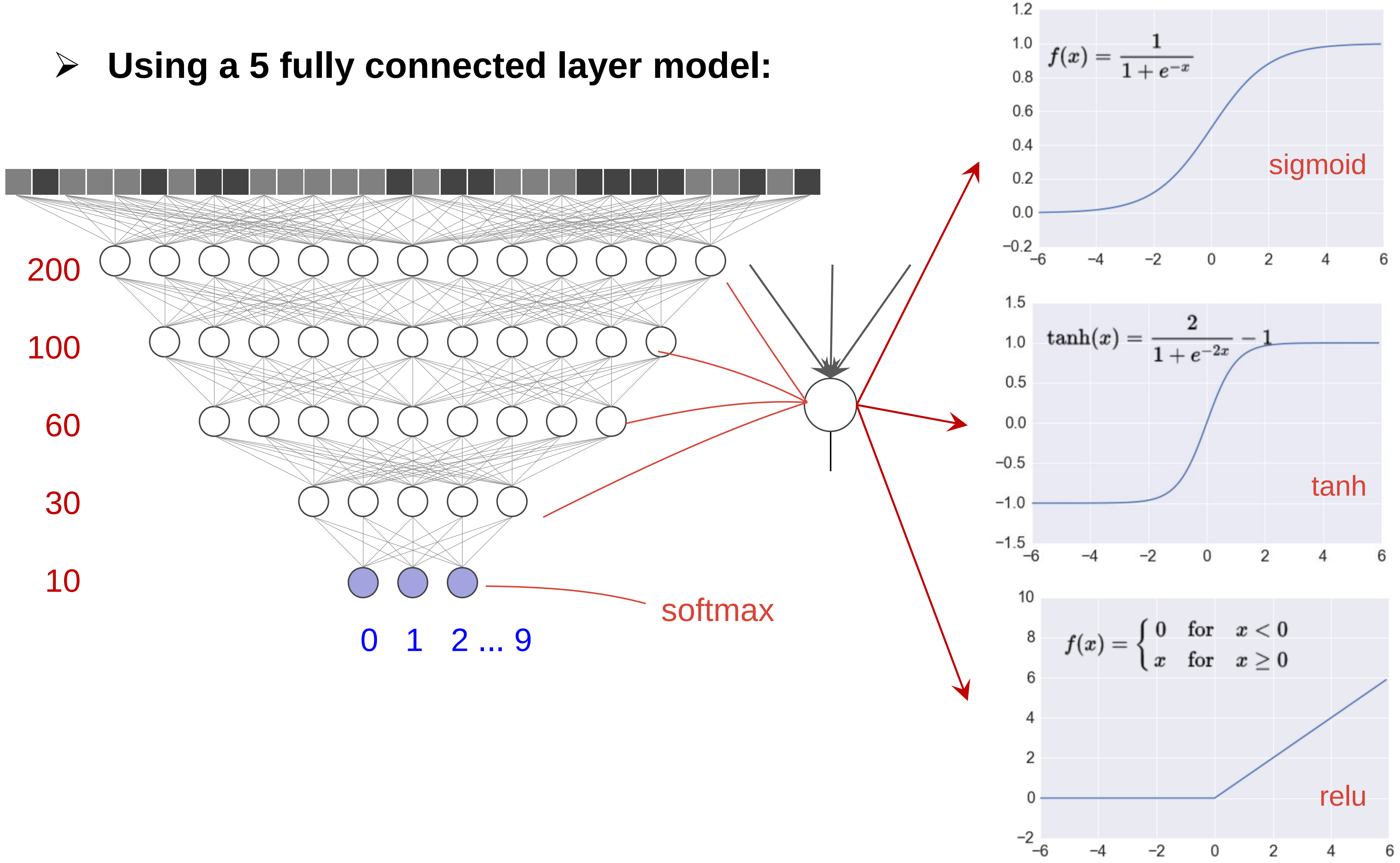
Incorrectly classified





Adding More Layers?

- **Using a 5 fully connected layer model:**



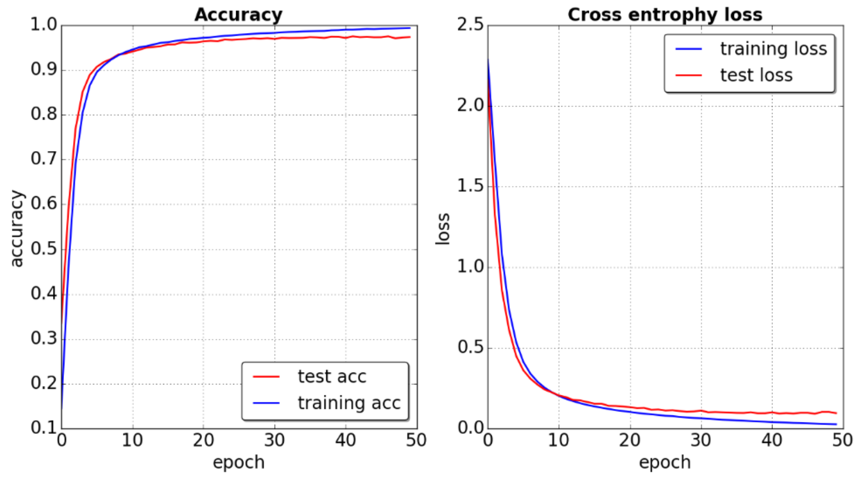
200
100
60
30
10
softmax
0 1 2 ... 9
$f(x) = \frac{1}{1+e^{-x}}$
sigmoid
$\tanh(x) = \frac{2}{1+e^{-2x}} - 1$
tanh
$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
relu

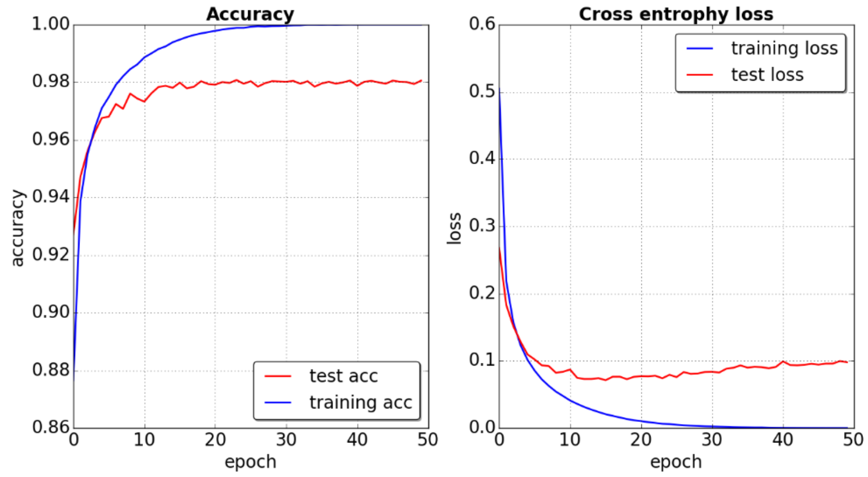
5 Layer Model In Keras

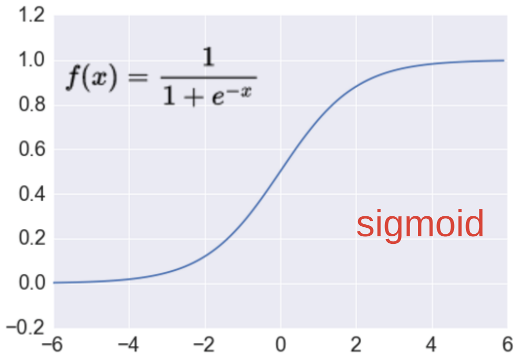
```
model = Sequential()
# try also tanh, sigmoid
act_func='relu'
model.add(Dense(200,activation=act_func,input_shape=input_shape))
model.add(Dense(100,activation=act_func))
model.add(Dense( 60,activation=act_func))
model.add(Dense( 30,activation=act_func))
model.add(Dense(nb_classes,activation='softmax'))
model.compile(loss=‘categorical_crossentropy’,optimizer='sgd',
              metrics=['accuracy'])
h = model.fit(X_train, Y_train, batch_size=batch_size,nb_epoch=nb_epoch,
          verbose=1, validation_data=(X_test, Y_test))
```

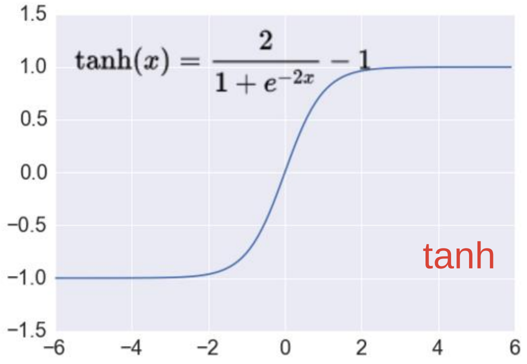
5 Layer Regression – Different Activation

- **Training accuracy vs Test accuracy, loss function**
- **We reach a Test accuracy at *97.35% (sigmoid), 98.06% (tanh)***







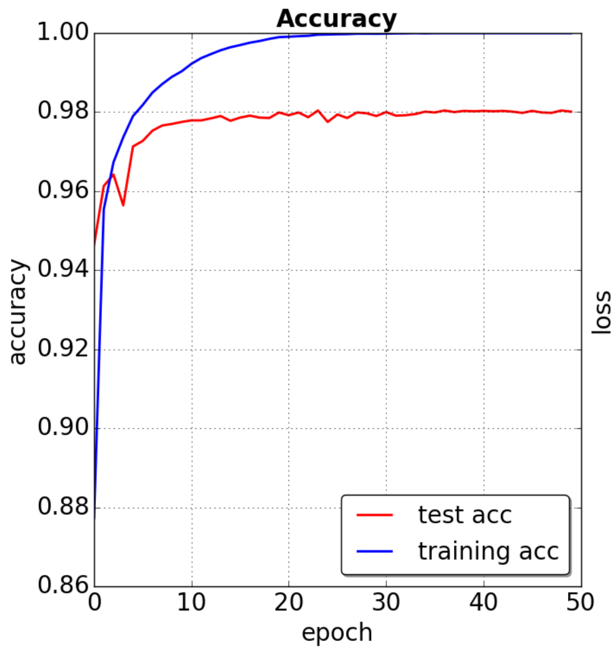


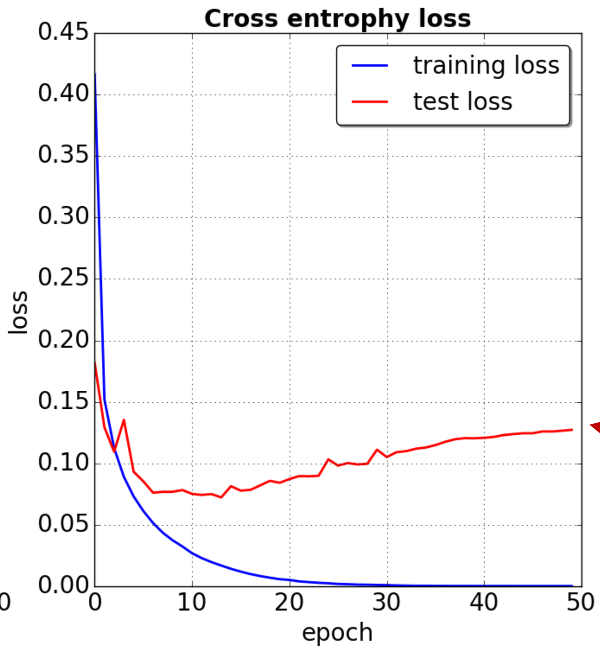
Rectified Linear Unit (ReLU) activation function

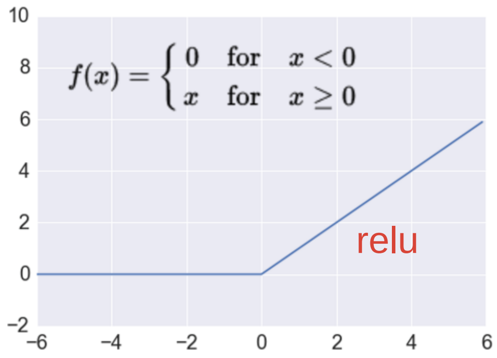
- **ReLU - The Rectified Linear Unit has become very popular in the last few years:**

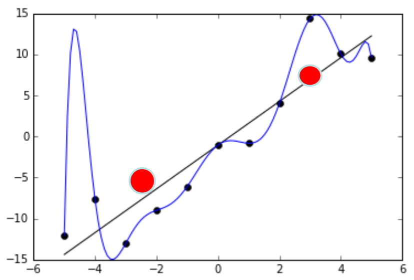
$$f(z) = \max(0, z)$$

- **We get a test accuracy of *98.07%* with ReLU**





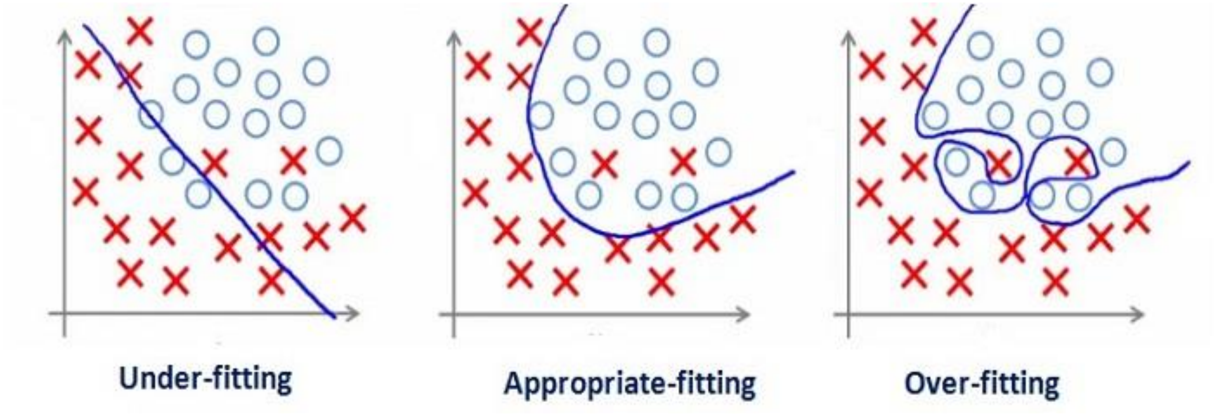




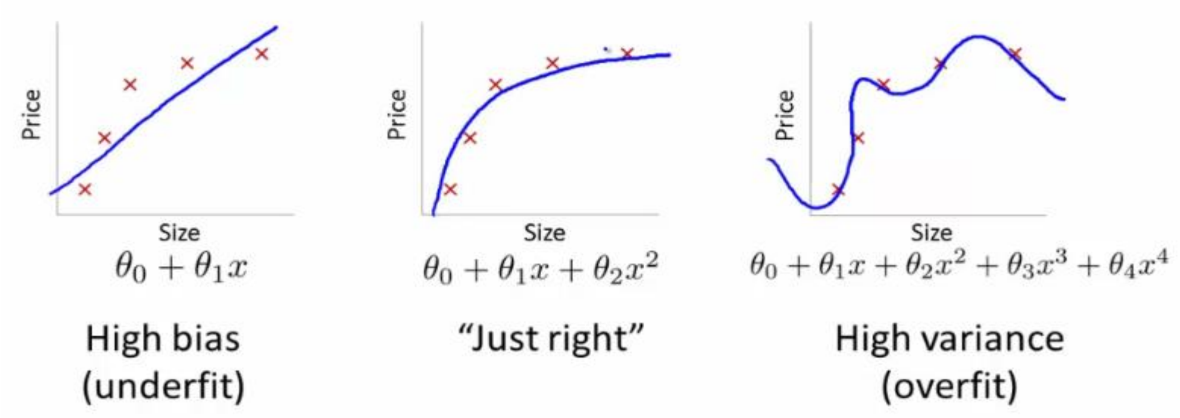
Overfitting

- Overfitting occurs when a model is excessively complex, such as having too many parameters relative to the number of observations. A model that has been overfit has poor predictive performance, as it overreacts to minor fluctuations in the training data.

Classification:

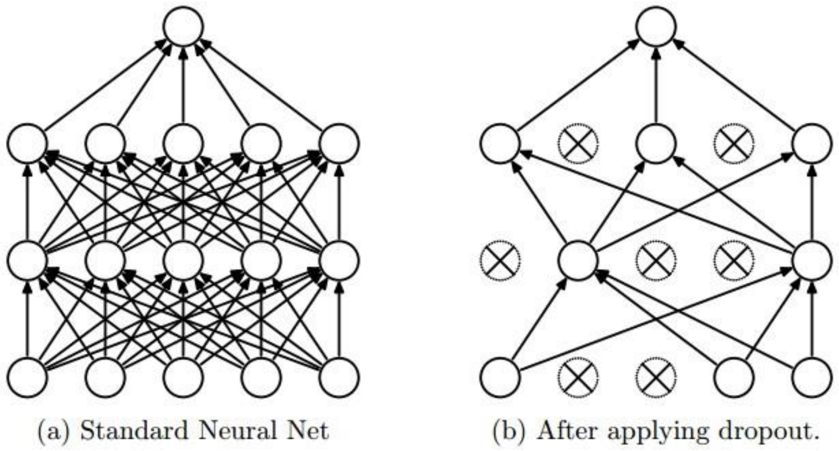


Regression:



Regularization - Dropout

- Dropout is an extremely effective, simple and recently introduced regularization technique by Srivastava et al (2014).



- While training, dropout is implemented by only keeping a neuron active with some probability p (a hyperparameter), or setting it to zero otherwise.
- It is quite simple to apply dropout in Keras.

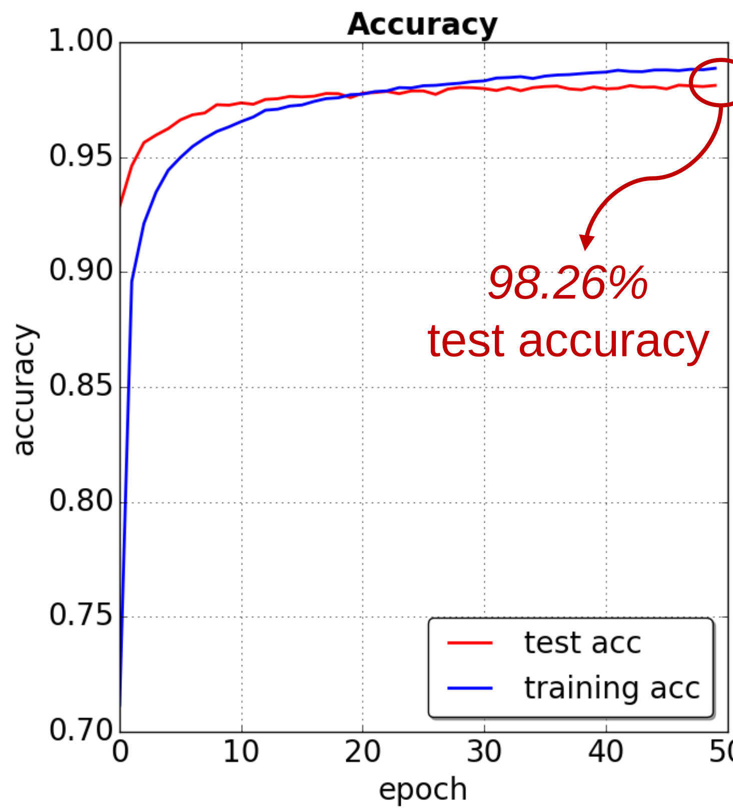
```
# apply a dropout rate 0.25 (drop 25% of the neurons)
model.add(Dropout(0.25))
```

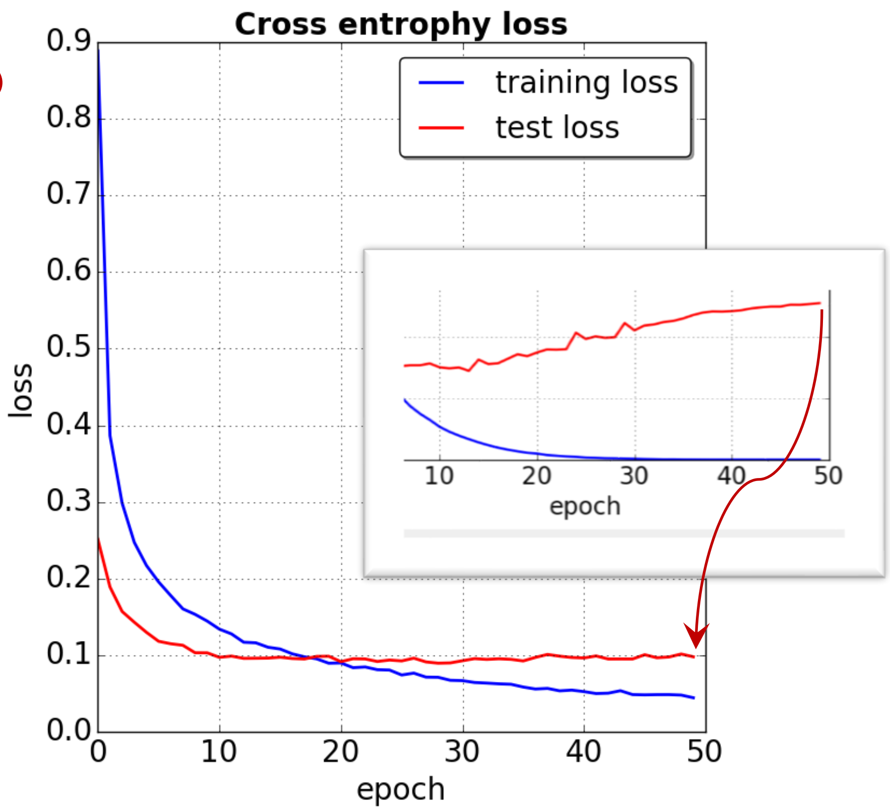
Apply Dropout To The 5 Layer NN

```
model = Sequential()
act_func='relu'
p_dropout=0.25 # apply a dropout rate 25 %
model.add(Dense(200,activation=act_func,input_shape=input_shape))
model.add(Dropout(p_dropout))
model.add(Dense(100,activation=act_func))
model.add(Dropout(p_dropout))
model.add(Dense( 60,activation=act_func))
model.add(Dropout(p_dropout))
model.add(Dense( 30,activation=act_func))
model.add(Dropout(p_dropout))
model.add(Dense(nb_classes,activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer=‘sgd',
              metrics=['accuracy'])
h = model.fit(X_train, Y_train, batch_size=batch_size,nb_epoch=nb_epoch,
          verbose=1, validation_data=(X_test, Y_test))
```

Results Using p_dropout=0.25

- **Resolve the overfitting issue**
- **Sustained** *98.26%* **accuracy**



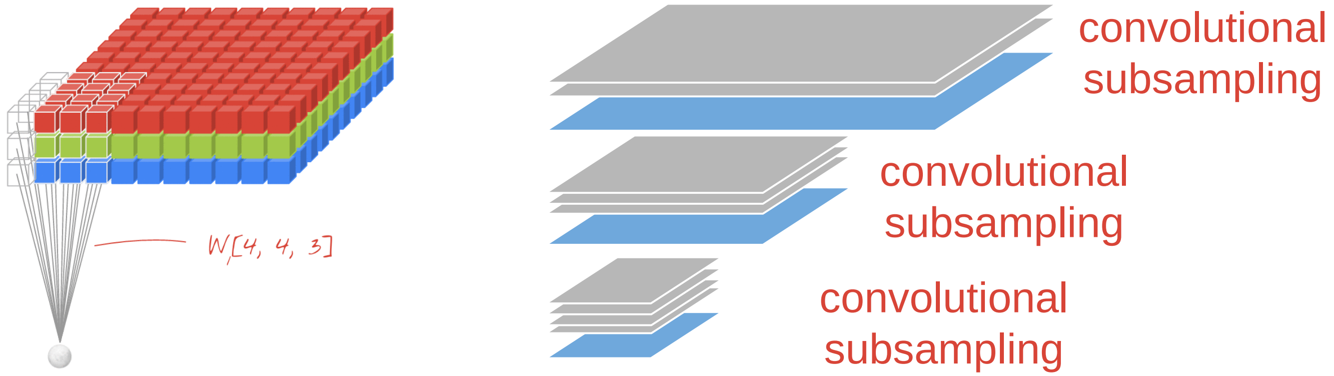


Why Using Fully Connected Layers?

- **Such a network architecture does not take into account the spatial structure of the images.**
  - For instance, it treats input pixels which are far apart and close together on exactly the same weight.
- **Spatial structure must instead be inferred from the training data.**

❖ **Is there an architecture which tries to take advantage of the spatial structure?**

Convolution Neuron Network (CNN)

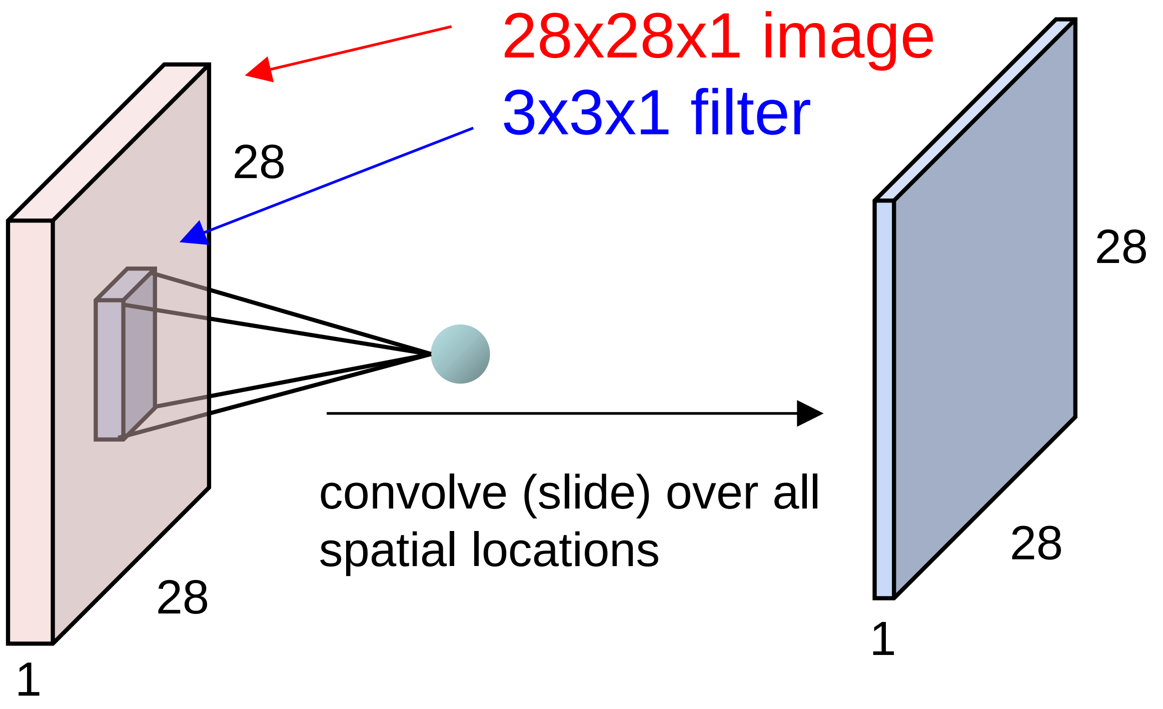

W[4, 4, 3]
convolutional subsampling
convolutional subsampling
convolutional subsampling

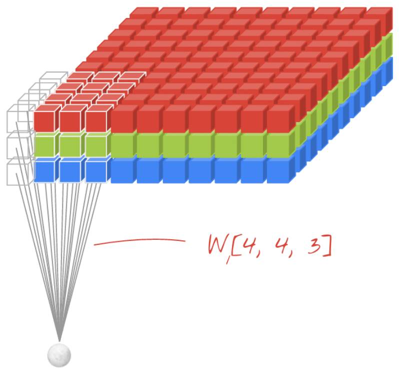
from Martin Görner Learn TensorFlow and deep learning, without a Ph.D

- **Deep convolutional network is one of the most widely used types of deep network.**
- **In a layer of a convolutional network, one "neuron" does a weighted sum of the pixels just above it, across a small region of the image only. It then acts normally by adding a bias and feeding the result through its activation function.**
- **The big difference is that each neuron reuses the same weights whereas in the fully-connected networks seen previously, each neuron had its own set of weights.**

activation map

How Does CNN Work?

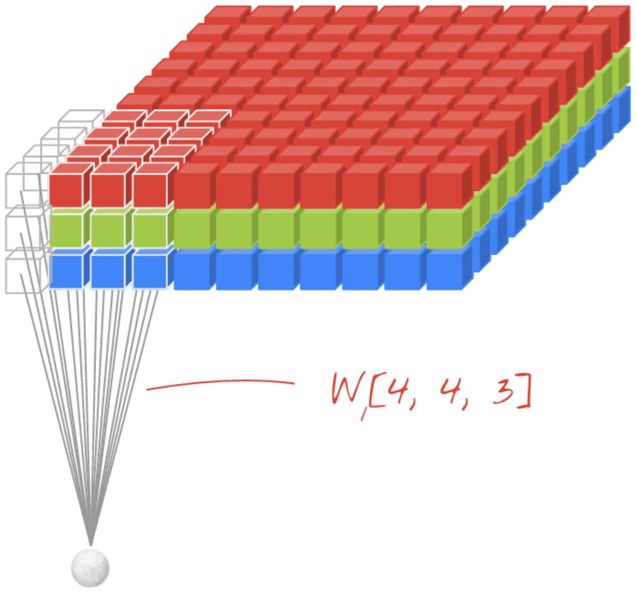


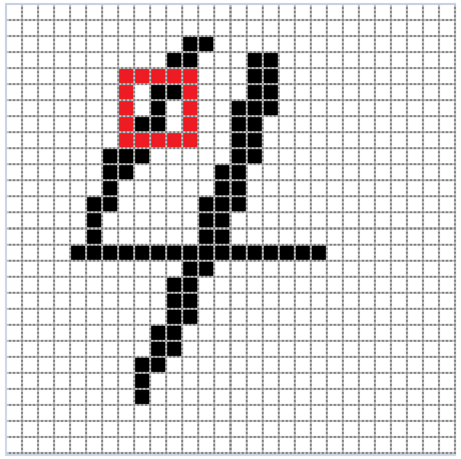


- By sliding the patch of weights (filter) across the image in both directions (a convolution) you obtain as many output values as there were pixels in the image (some padding is necessary at the edges).

Three basic ideas about CNN

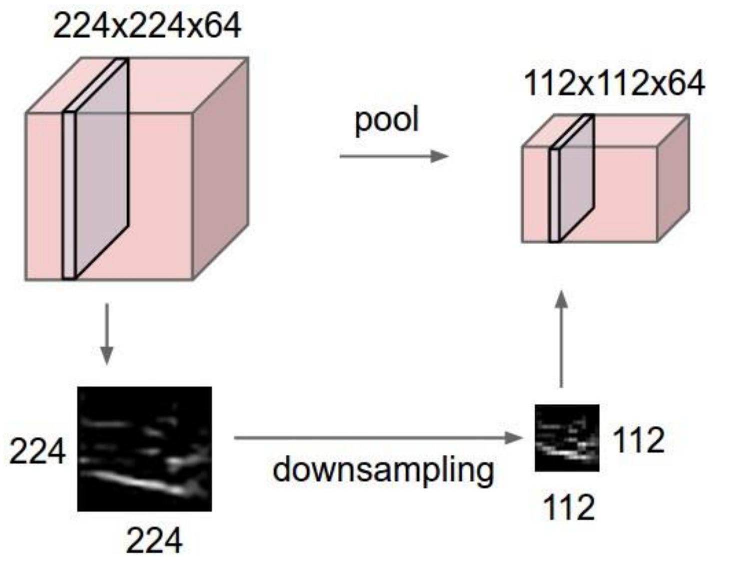
- **Local receptive fields**
- **Shared weights and biases:**
- **Pooling**

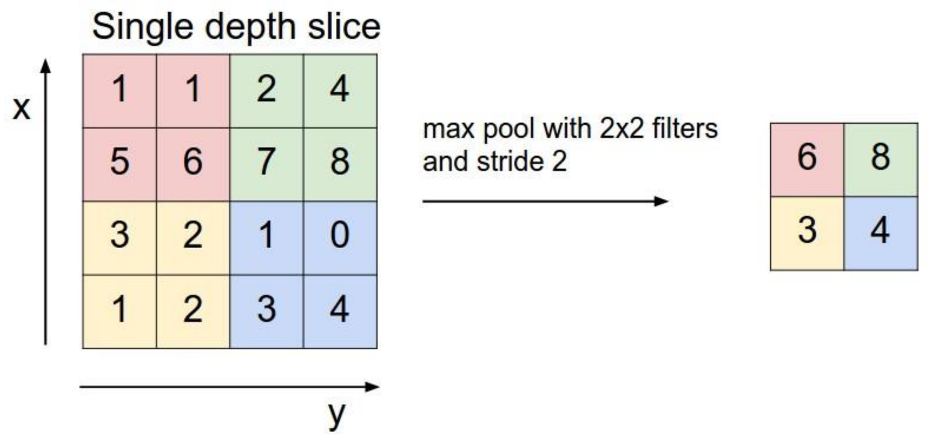




Pooling Layer

- **Convolutional neural networks also contain pooling layers. Pooling layers are usually used immediately after convolutional layers.**
- **What the pooling layers do is simplify the information in the output from the convolutional layer.**
- **We can think of max-pooling as a way for the network to ask whether a given feature is found anywhere in a region of the image. It then throws away the exact positional information.**

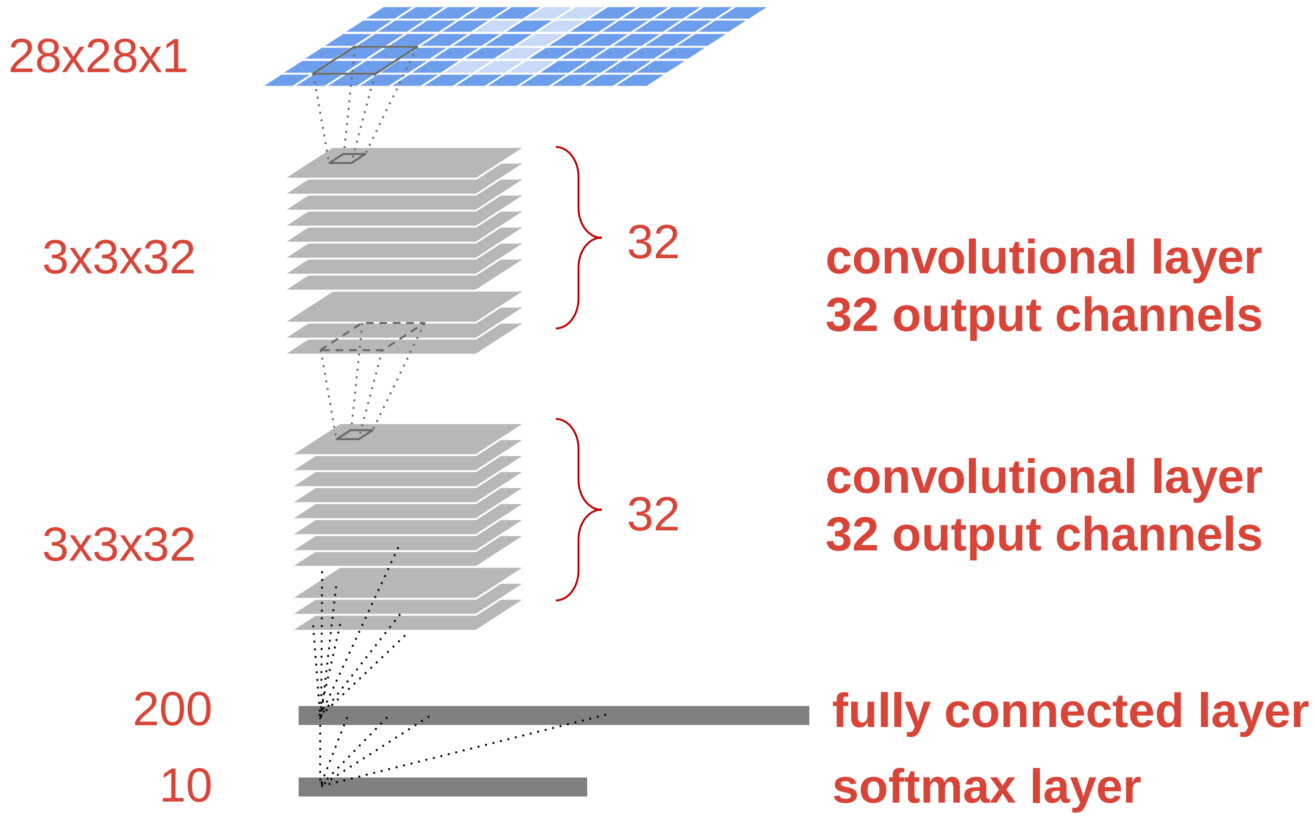




1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

6	8
3	4

Convolutional Network With Fully Connected Layers



Stacking And Chaining Convolutional Layers in Keras

nb_filters=32

kernel_size=(3,3)

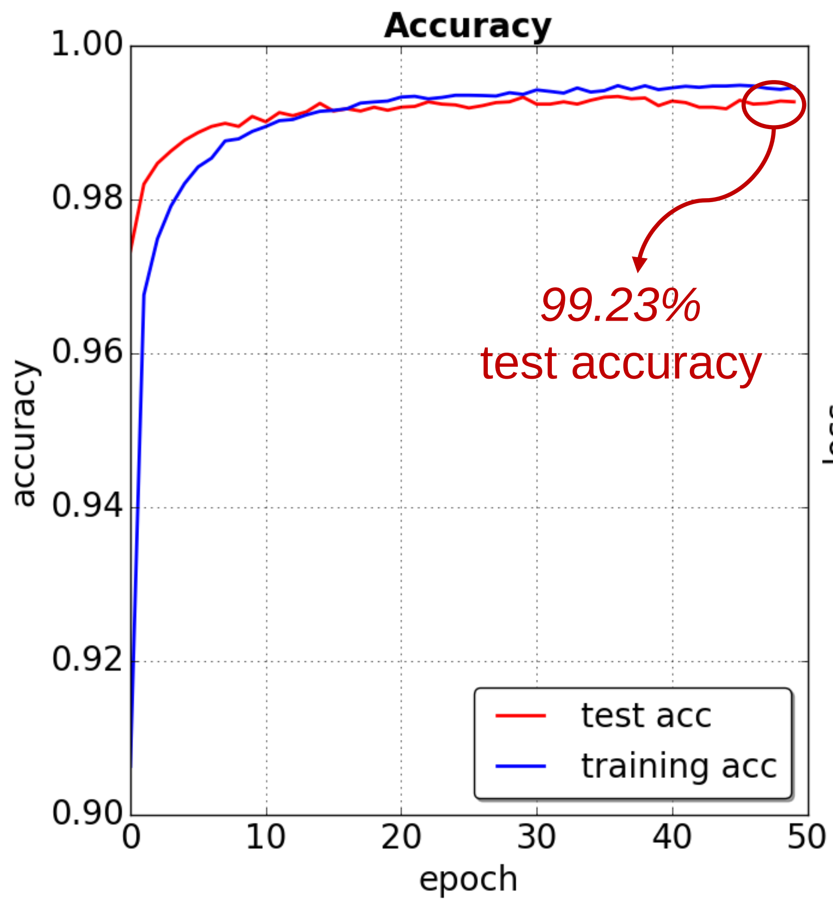
input_shape=(28,28)

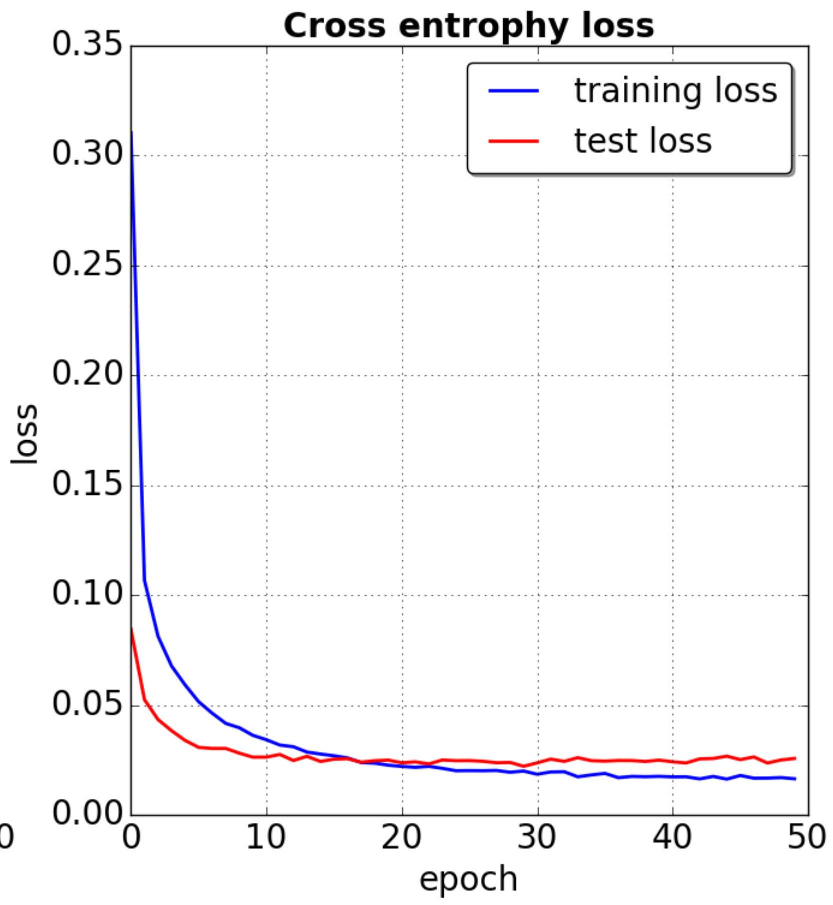
```
model = Sequential()
# Adding the convulation layers
model.add(Convolution2D(nb_filters, kernel_size[0], kernel_size[1],
                        border_mode='valid',
                        input_shape=input_shape))
model.add(Activation('relu'))
model.add(Convolution2D(nb_filters, kernel_size[0], kernel_size[1]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=pool_size))
model.add(Dropout(0.25))
# Fully connected layers
model.add(Flatten())
model.add(Dense(256,activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(nb_classes,activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adadelta',
          metrics=['accuracy'])

h = model.fit(X_train, Y_train, batch_size=batch_size, nb_epoch=nb_epoch,
          verbose=1,callbacks=[history], validation_data=(X_test, Y_test))
```

Challenging The 99% Testing Accuracy

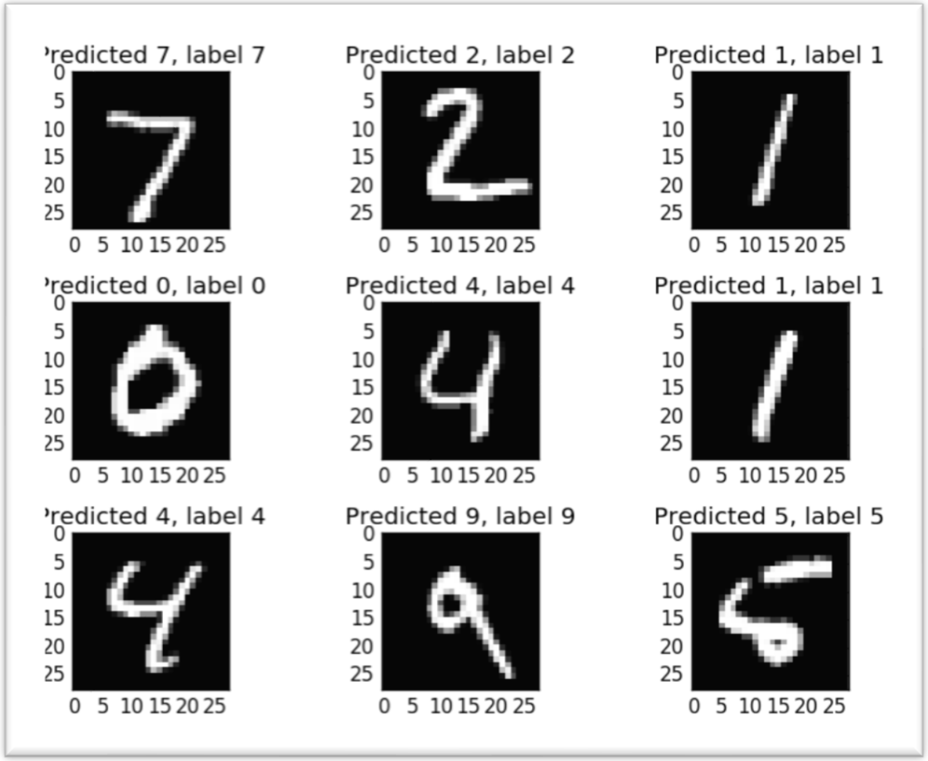
- **By using the convolution layer and the fully connected layers, we reach a test accuracy of** *99.23%*



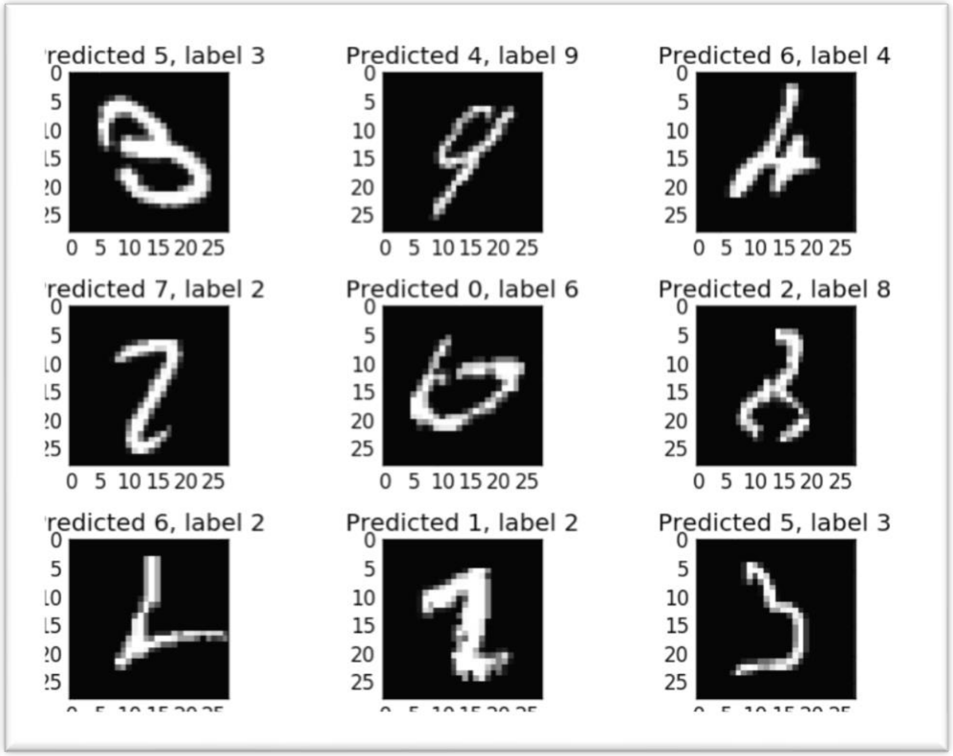


Review The Classified Results of CNN

Correctly classified

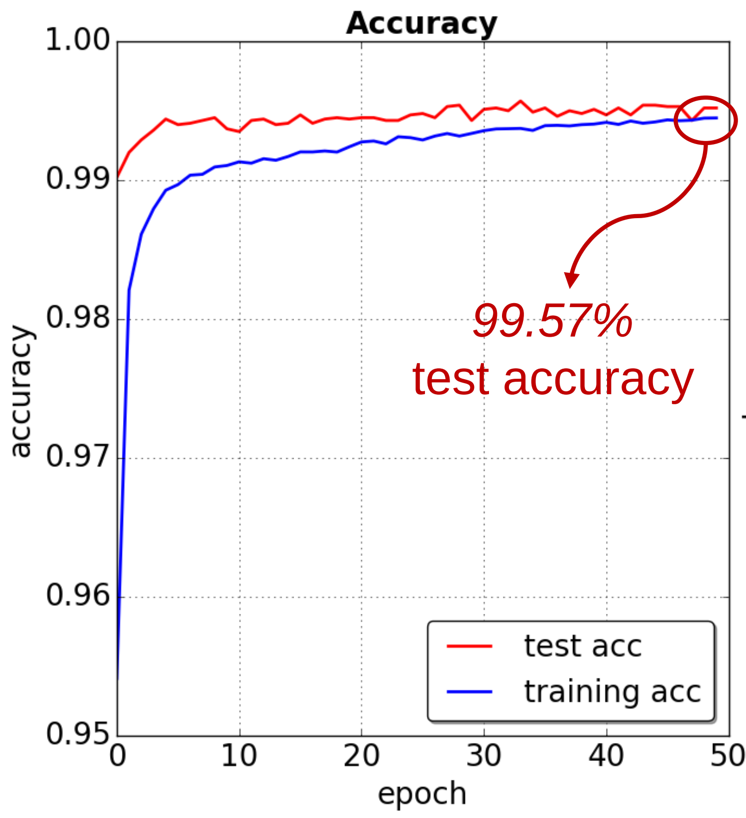


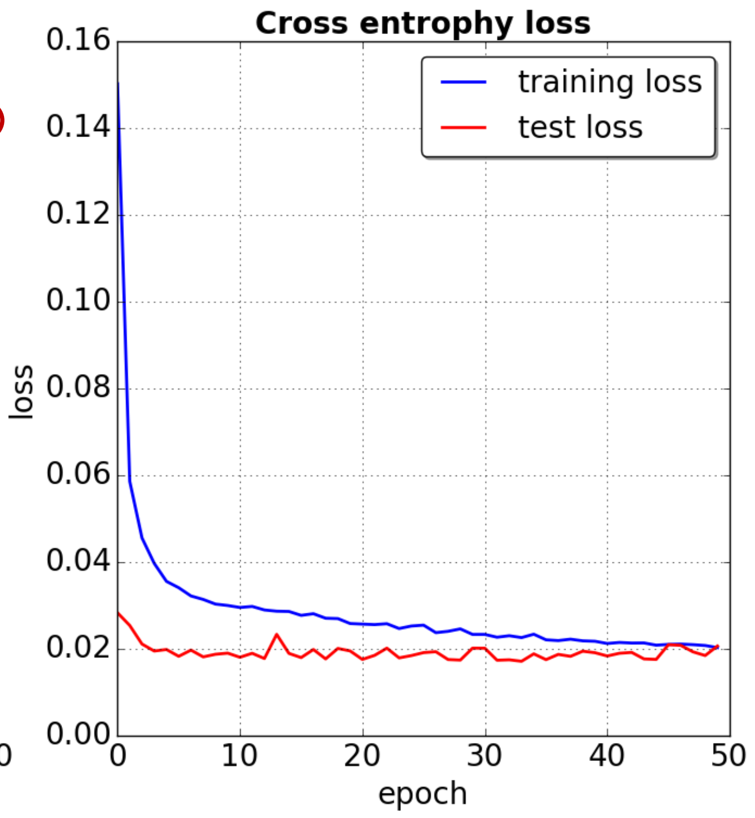
Incorrectly classified



Feed More Data: Using Expanded Dataset

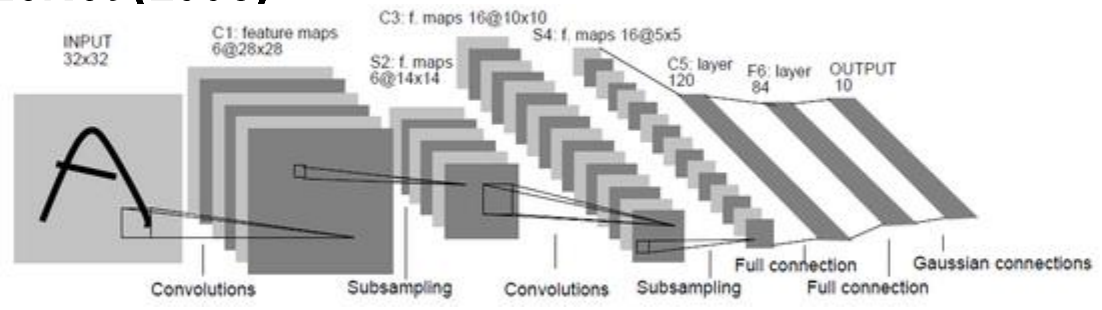
- **We can further increase the test accuracy by expanding the mnist.pkl.gz dataset, reaching a nearly** *99.6%* **test accuracy**



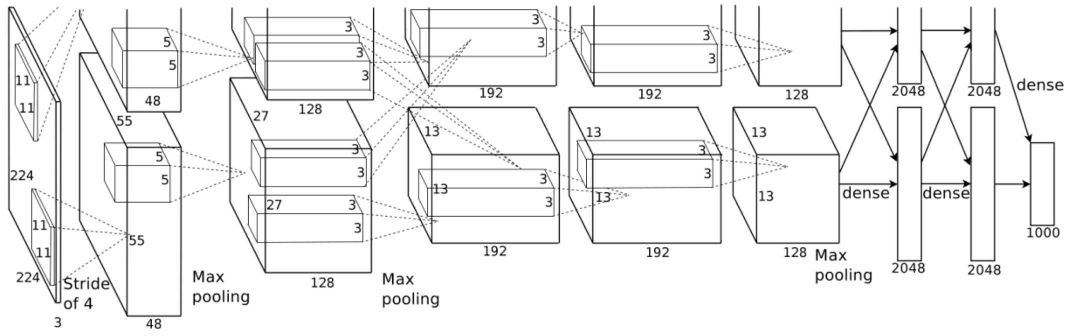


Examples of Convolution NN

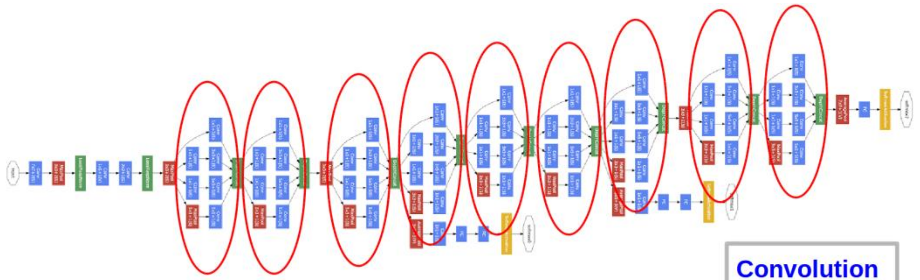
- **LeNet (1998)**



- **AlexNet (2012)**



- **GoogleLeNet (2014)**



Machine Learning Courses List

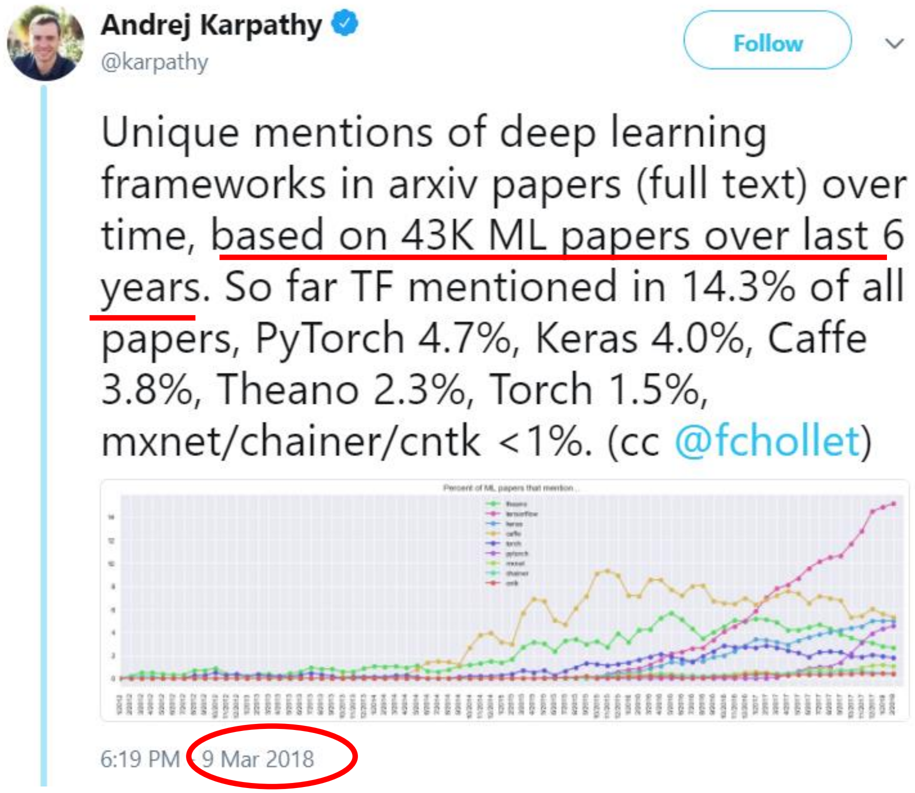
- **Machine Learning in Coursera**
  https://www.coursera.org/learn/machine-learning
- **Learning from Data (Caltech)**
  https://work.caltech.edu/telecourse.html
- **Convolutional Neural Networks for Visual Recognition**
  http://cs231n.github.io/
- **Deep Learning for Natural Language Processing**
  https://cs224d.stanford.edu/

Deep Learning Frameworks

An Overview of Deep Learning Frameworks

Overview of Deep Learning Frameworks

Name	Percentage
TensorFlow	14.3
PyTorch	4.7
Keras	4.0
Caffe	3.8
Theano	2.3
Torch	1.5
MXNet/Chainer/CNTK	<1



Andrej Karpathy
@karpathy

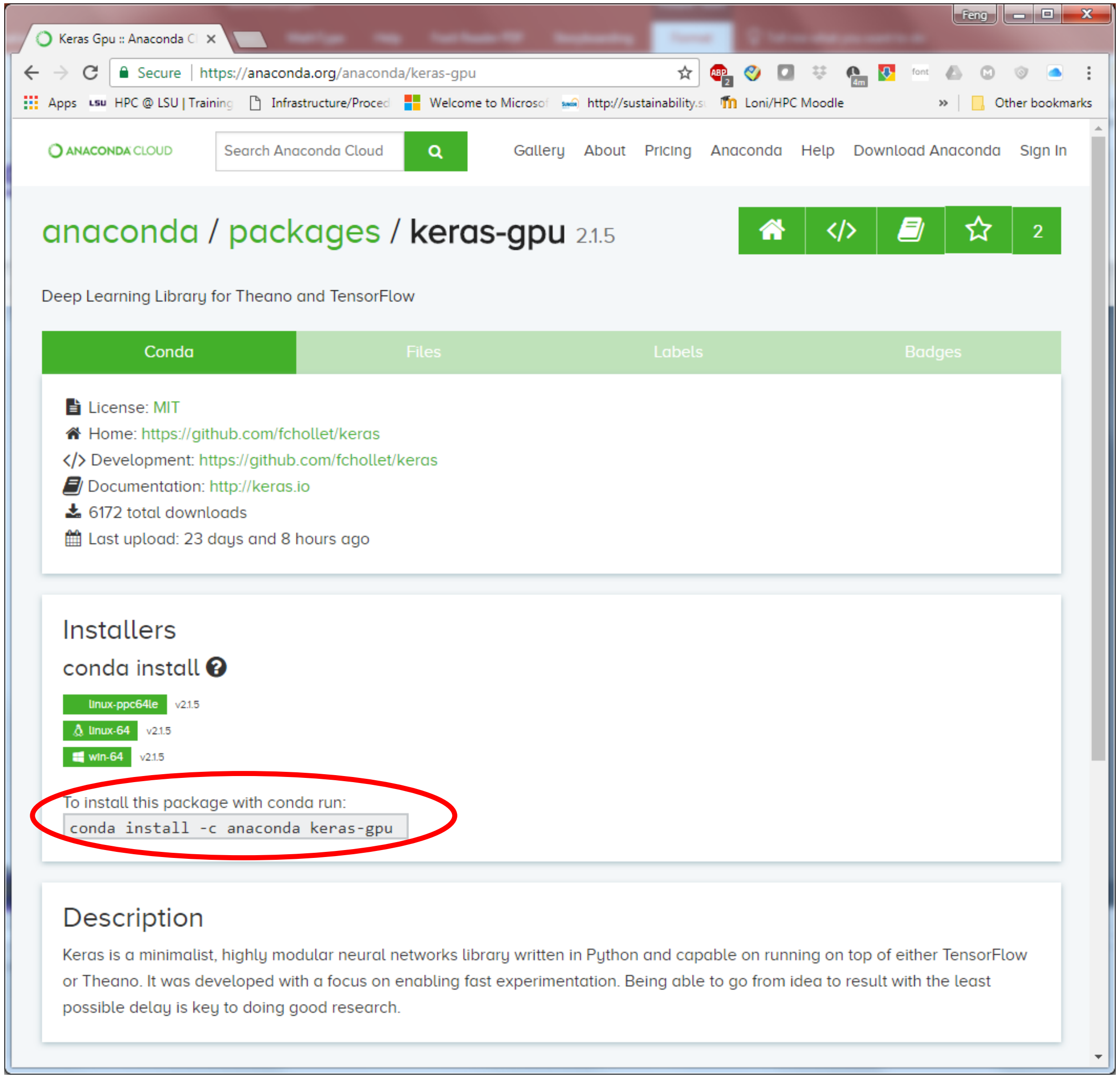
Unique mentions of deep learning frameworks in arxiv papers (full text) over time, based on 43K ML papers over last 6 years. So far TF mentioned in 14.3% of all papers, PyTorch 4.7%, Keras 4.0%, Caffe 3.8%, Theano 2.3%, Torch 1.5%, mxnet/chainer/cntk <1%. (cc @fchollet)

6:19 PM - 9 Mar 2018

Deep Learning Software

Tensorflow/Keras

Using Anaconda Cloud


Keras Gpu :: Anaconda C

Secure https://anaconda.org/anaconda/keras-gpu

Apps HPC @ LSU | Training Infrastructure/Proced Welcome to Microso http://sustainability.s Loni/HPC Moodle Other bookmarks

ANACONDA CLOUD Search Anaconda Cloud Gallery About Pricing Anaconda Help Download Anaconda Sign In

anaconda / packages / keras-gpu 2.1.5

Deep Learning Library for Theano and TensorFlow

Conda Files Labels Badges

License: MIT
Home: https://github.com/fchollet/keras
Development: https://github.com/fchollet/keras
Documentation: http://keras.io
6172 total downloads
Last upload: 23 days and 8 hours ago

Installers

conda install

linux-ppc64le v2.1.5
linux-64 v2.1.5
win-64 v2.1.5

To install this package with conda run:

```
conda install -c anaconda keras-gpu
```

Description

Keras is a minimalist, highly modular neural networks library written in Python and capable on running on top of either TensorFlow or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Install latest tensorflow/keras-gpu using conda virtual environment

1. **Create conda virtual environment:**

```
[fchen14@qb001 fchen14]$ pwd
/work/fchen14
[fchen14@qb001 fchen14]$ conda create --prefix /work/fchen14/myenv python=2.7
Fetching package metadata ...........
Solving package specifications: .

Package plan for installation in environment /work/fchen14/myenv:

The following NEW packages will be INSTALLED:
ca-certificates: 2018.03.07-0
    certifi:         2018.1.18-py27_0
...
Proceed ([y]/n)? y
...
pip-9.0.3-py27 100% |#############################################| Time: 0:00:00  11.82 MB/s
#
# To activate this environment, use:
# > source activate /work/fchen14/myenv
#
# To deactivate an active environment, use:
# > source deactivate
```

Install keras-gpu using anaconda channel

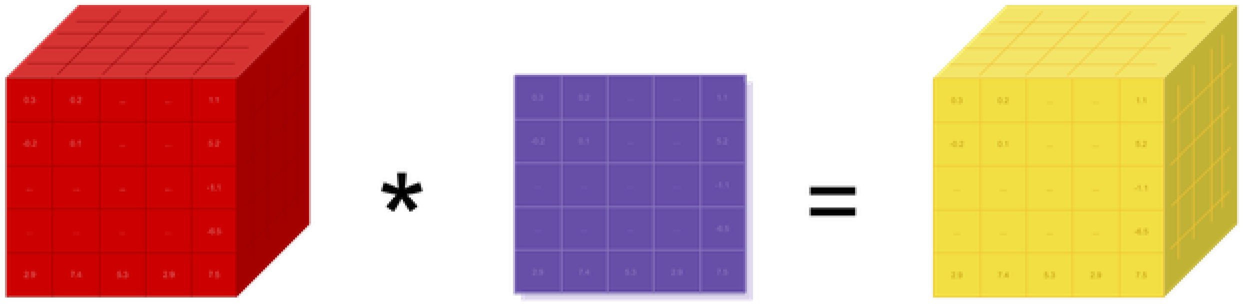
```
[fchen14@qb001 ~]$ source activate /work/fchen14/myenv/
(/work/fchen14/myenv/) [fchen14@qb001 ~]$ conda install -c anaconda keras-gpu
Fetching package metadata .............
Solving package specifications: .
Package plan for installation in environment /work/fchen14/myenv:
The following NEW packages will be INSTALLED:
...
    keras-gpu:              2.1.5-py27_0         anaconda
...
    scipy:                  1.0.1-py27hfc37229_0 anaconda
    tensorflow-gpu:         1.4.1-0              anaconda
    tensorflow-gpu-base:    1.4.1-py27h01caf0a_0 anaconda
...
Proceed ([y]/n)? y
...
tensorflow-gpu 100% |###########################| Time: 0:00:10  11.45 MB/s
...
keras-gpu-2.1. 100% |###########################| Time: 0:00:00  11.59 MB/s
(/work/fchen14/myenv/) [fchen14@qb001 ~]$ python
Python 2.7.14 |Anaconda, Inc.| (default, Mar 27 2018, 17:29:31)
[GCC 7.2.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
>>> tf.__version__
'1.4.1'
```

Deep Learning Software

Torch/PyTorch

About PyTorch (from http://pytorch.org/about/)

- **PyTorch is a python package that provides two high-level features:**
  - Tensor computation (like numpy) with strong GPU acceleration
  - Deep Neural Networks built on a tape-based autodiff system
- **Usually one uses PyTorch either as:**
  - A replacement for numpy to use the power of GPUs.
  - a deep learning research platform that provides maximum flexibility and speed
- A GPU-ready Tensor library
  - If you use numpy, then you have used Tensors (a.k.a ndarray).
  - PyTorch provides Tensors that can live either on the CPU or the GPU, and accelerate compute by a huge amount.



Installation of PyTorch on QB2

➢ **Create conda virtual environment:**

```
[fchen14@qb001 fchen14]$ pwd
/work/fchen14
[fchen14@qb001 fchen14]$ conda create --prefix /work/fchen14/myenv python=2.7
Fetching package metadata ...........
Solving package specifications: .

Package plan for installation in environment /work/fchen14/myenv:

The following NEW packages will be INSTALLED:
ca-certificates: 2018.03.07-0
    certifi:         2018.1.18-py27_0
...
Proceed ([y]/n)? y
...
pip-9.0.3-py27 100% |##############################################| Time: 0:00:00  11.82 MB/s
#
# To activate this environment, use:
# > source activate /work/fchen14/myenv
#
# To deactivate an active environment, use:
# > source deactivate
#
```

Installation of PyTorch on QB2

- **Activate the virtual environment and install PyTorch, torchvision**

```
[fchen14@qb001 fchen14]$ source activate /work/fchen14/myenv
(/work/fchen14/myenv) [fchen14@qb001 fchen14]$ echo $PYTHONPATH
/usr/local/packages/python/2.7.12-anaconda/lib/python2.7/site-packages/
(/work/fchen14/myenv) [fchen14@qb001 fchen14]$ unset PYTHONPATH # remove previous PYTHONPATH
# do *not* use the “–c pytorch” which has the GLIBC2.17 issue on QB2
(/work/fchen14/myenv) [fchen14@qb001 fchen14]$ conda install -c soumith pytorch torchvision
Fetching package metadata .............
...
The following NEW packages will be INSTALLED:
    cudnn:   7.0.5-cuda8.0_0
    pytorch: 0.3.1-py27_cuda8.0.61_cudnn7.0.5_2 soumith
    torchvision: 0.1.9-py27hdb88a65_1 soumith
Proceed ([y]/n)? y
cudnn-7.0.5-cu 100% |###############################| Time: 0:00:22  11.72 MB/s
pytorch-0.3.1- 100% |###############################| Time: 0:00:18  11.47 MB/s
torchvision-0. 100% |###############################| Time: 0:00:00 868.61 kB/s
(/work/fchen14/myenv) [fchen14@qb001 fchen14]$ python
Python 2.7.14 |Anaconda, Inc.| (default, Mar 27 2018, 17:29:31)
[GCC 7.2.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch # test we are able to use the pytorch module
>>>
```

Run PyTorch Example

```
# scripts from the repository:
# https://github.com/pytorch/examples/tree/master/mnist
(/work/fchen14/myenv) [fchen14@qb001 pytorch]$ cd
/project/fchen14/dlsoftware/mnist/pytorch
(/work/fchen14/myenv) [fchen14@qb001 pytorch]$ ls
mnist_pytorch.ipynb  mnist_pytorch.py  README.md  requirements.txt
# first time run will need to download mnist dataset, be patient
(/work/fchen14/myenv) [fchen14@qb001 pytorch]$ python mnist_pytorch.py
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.373651
Test set: Average loss: 0.2042, Accuracy: 9409/10000 (94%)
Train Epoch: 2 [0/60000 (0%)]   Loss: 0.482378
Test set: Average loss: 0.1279, Accuracy: 9581/10000 (96%)
Train Epoch: 3 [0/60000 (0%)]   Loss: 0.524040
Test set: Average loss: 0.1000, Accuracy: 9689/10000 (97%)
...
Train Epoch: 8 [0/60000 (0%)]   Loss: 0.277341
Test set: Average loss: 0.0578, Accuracy: 9819/10000 (98%)
Train Epoch: 9 [0/60000 (0%)]   Loss: 0.072081
Test set: Average loss: 0.0535, Accuracy: 9839/10000 (98%)
Train Epoch: 10 [0/60000 (0%)]  Loss: 0.103900
Test set: Average loss: 0.0487, Accuracy: 9845/10000 (98%)
```

PyTorch MNIST Code Segments (1)

```
class Net(nn.Module): # define NN structure
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

PyTorch MNIST Code Segments (2)

```
def train(epoch): # define the train function
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        if args.cuda:
            data, target = data.cuda(), target.cuda()
        data, target = Variable(data), Variable(target)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.data[0]))
```

Deep Learning Software

Caffe

Caffe Introduction

- **Caffe (http://caffe.berkeleyvision.org/) is a deep learning framework made with expression, speed, and modularity in mind. It is developed by Berkeley AI Research (BAIR) and by community contributors.**

- **Yangqing Jia created the project during his PhD at UC Berkeley. Caffe is released under the BSD 2-Clause license.**

- **Switch between CPU and GPU by setting a single flag to train on a GPU machine then deploy to commodity clusters or mobile devices.**
- **Extensible code fosters active development.**

- **Speed makes Caffe perfect for research experiments and industry deployment. Caffe can process over 60M images per day with a single NVIDIA K40 GPU*.**

Steps to run MNIST on QB2 with Caffe

```
[fchen14@qb001 caffe]$ pwd
/project/fchen14/dlsoftware/mnist/caffe
[fchen14@qb001 caffe]$ module purge
[fchen14@qb001 caffe]$ module load caffe/1.0 # load the caffe module on QB2
[fchen14@qb001 caffe]$ ls
create_mnist.sh  get_mnist.sh  lenet_solver.prototxt  lenet_train_test.prototxt
train_lenet.sh
[fchen14@qb001 caffe]$ ./get_mnist.sh # download mnist dataset
Downloading...
[fchen14@qb001 caffe]$ ./create_mnist.sh # create lmdb format using original data
Creating lmdb...
Done.
[fchen14@qb001 caffe]$ ./train_lenet.sh # train the LeNet using Caffe
I0411 19:23:08.499141 41865 caffe.cpp:218] Using GPUs 0
I0411 19:23:08.579864 41865 caffe.cpp:223] GPU 0: Tesla K40m
I0411 19:23:08.892921 41865 solver.cpp:44] Initializing solver from parameters:
test_iter: 100
...
I0411 19:23:45.728441 41865 solver.cpp:397]     Test net output #1: loss =
0.0271103 (* 1 = 0.0271103 loss)
I0411 19:23:45.728446 41865 solver.cpp:315] Optimization Done.
I0411 19:23:45.728451 41865 caffe.cpp:259] Optimization Done.
```

Future Trainings

- **This is the last training for this semester**
  - Keep an eye on future HPC trainings at:
    - http://www.hpc.lsu.edu/training/tutorials.php#upcoming

- **Programming/Parallel Programming workshops in Summer**
  - See http://www.hpc.lsu.edu/training/workshop.php#upcoming

- **Visit our webpage: www.hpc.lsu.edu**