//

Machine Learning III

____

maXbox Starter 65 – Data Science with ML.

There are two kinds of data scientists:
 1) Those who can extrapolate from incomplete data.

The term “machine learning” is used to describe one kind of “artificial intelligence” (AI) where a machine is able to learn and adapt through its own experience.
This tutor introduces the basic idea of back-propagation and optimization learning process with a image classification example. Machine learning teaches machines (**and** me too) to learn to carry out tasks **and** concepts by themselves, so here **is** an overview:

http://www.softwareschule.ch/examples/machinelearning.jpg

maXbox4 ScriptStudio: 807_FANN_XorSample_EKON22.pas
15 807_FANN_XorSample_EKON22.pas

```
    '1') )
    Layers.add('2')
    Layers.add('3')
    Layers.add('1')

    LearningRate := 0.69999998807907100
    ConnectionRate := 1.00000000000000000
    TrainingAlgorithm := taFANN_TRAIN_RPROP
    ActivationFunctionHidden := afFANN_SIGMOID
    ActivationFunctionOutput := afFANN_SIGMOID
    //Left := 192
    //Top := 40
  end
end;
```

```
807_FANN_XorSample_EKON22.pas in maxboxdef.ini stored: 16:18:29
C:\maXbox\mX46210\maxbox4\examples\807_FANN_XorSample_EKON22.pas File stored
807_FANN_XorSample_EKON22.pas in maxboxdef.ini stored: 16:18:39
C:\maXbox\mX46210\maxbox4\examples\807_FANN_XorSample_EKON22.pas F[illegible]
807_FANN_XorSample_EKON22.pas in maxboxdef.ini stored: 16:21:58
```

EKON 22 November 2018 at Duesseldorf Session

Of course, machine learning (often also referred to **as** Artificial Intelligence, Artificial Neural Network, Big Data, Data Mining **or** Predictive Analysis) **is not** that new field **in** itself **as** they want to believe us. For most of the cases you do experience 5 steps **in** different loops in a artificial neural network:

• Data Shape	(Write one or more dataset importing functions)
• Modelclass	(Pre-made Estimators encode best practices,)
• Cost Function	(cost or loss, and then try to minimize error)
• Optimizer	(optimization to constantly modify vars to reduce costs.)
• Classify	(Choosing a model **and** classify algorithm - supervised)
• Accuracy	(Predict **or** report precision **and** drive data to decision)

The MNIST dataset contains images of handwritten digits (0, 1, 2, etc.). The first thing you have to do is to import the library and data too:

```
import tensorflow as tf

# Import MINST data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)
```

The first 2 images represent respectively 7 and 2. The option ***one_hot=True*** deal with one-hot tensor: 10-dimensional vectors indicating which digit class (zero through nine) the corresponding MNIST image belongs to. The 2 previous images would have been represented like

```
[[0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0], …].
```

MNIST images have shape 28px x 28px, so we can define one feature with shape [28, 28]:

```
# mnist data image of shape 28*28=784
     x = tf.placeholder(tf.float32, [None, 784], name='InputData')
     # 0-9 digits recognition => 10 classes
     y = tf.placeholder(tf.float32, [None, 10], name='LabelData')
```

Next step is to define the model:

```
if __name__ == "__main__":
     # Parameters
     learning_rate = 0.01
     training_epochs = 25
     batch_size = 120
     display_epoch = 1
     logs_path = '/tmp/tensorflow_logs/example/'
```

The learning rate (*eps_k*) determines the size of the step that the algorithm takes along the gradient (in the negative direction in the case of minimization and in the positive direction in the case of maximization).

The learning rate is a function of iteration *k* and is a single most important hyper-parameter. A learning rate that is too high (e.g. > 0.1) can lead to parameter updates that miss the optimum value, a learning rate that is too low (e.g. < 1e-5) will result in unnecessarily long training time.

Our model also requires weights and offset values, of course, we can use them as a further input (placeholders), but there is a better way TensorFlow to represent them: Variable. One Variable represents a modifiable tensor that exists in TensorFlow diagram for describing interactive operations. They can be used to calculate input values and can be modified in calculations.

```
# Set model weights
     W = tf.Variable(tf.zeros([784, 10]), name='Weights')
     b = tf.Variable(tf.zeros([10]), name='Bias')
```

You can run Estimator-based models on a local host or on a distributed multi-server environment without changing your model. Furthermore, you can run Estimator-based models on CPUs, GPUs, or TPUs without recoding your model.

```
# Construct model and encapsulating all ops into scopes, making
     # Tensorboard's Graph visualization more convenient
     with tf.name_scope('Model'):
         # Model
         pred = tf.nn.softmax(tf.matmul(x, W) + b) # Softmax
     with tf.name_scope('Loss'):
         # Minimize error using cross entropy
         cost = \
         tf.reduce_mean(-tf.reduce_sum(y * tf.log(pred),
                                  reduction_indices=1))
     with tf.name_scope('SGD'):
         # Gradient Descent
         optimizer = \

tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
     with tf.name_scope('Accuracy'):
         # Accuracy
         acc = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
         acc = tf.reduce_mean(tf.cast(acc, tf.float32))
```

So whats behind pred = tf.nn.softmax(tf.matmul(x, W) + b)

In order to get evidence that a given picture belongs to a specific digital class, we perform a weighted sum on the picture pixel values.
If this pixel has strong evidence that the picture does not belong to this class, the corresponding weight is negative, and conversely, if the pixel has favorable evidence to support this picture belongs to this class, then the weight is a positive number.
This is a classic case using a **softmax** regression model. The softmax model can be used to assign probabilities to different objects.

First of all, we use the tf.matmul(X, W) expression x multiplication W, corresponding to the previous equation wx, where x is a 2-dimensional tensor with multiple inputs. Then add band enter the **tf.nn.softmax** function into the function.

https://github.com/tensorflow/tensorflow/blob/r1.10/tensorflow/python/ops/nn_ops.py

When writing an application with Estimators, you must separate the data input pipeline from the model. This separation simplifies experiments with different data sets.
Next step is to prepare the model for training:

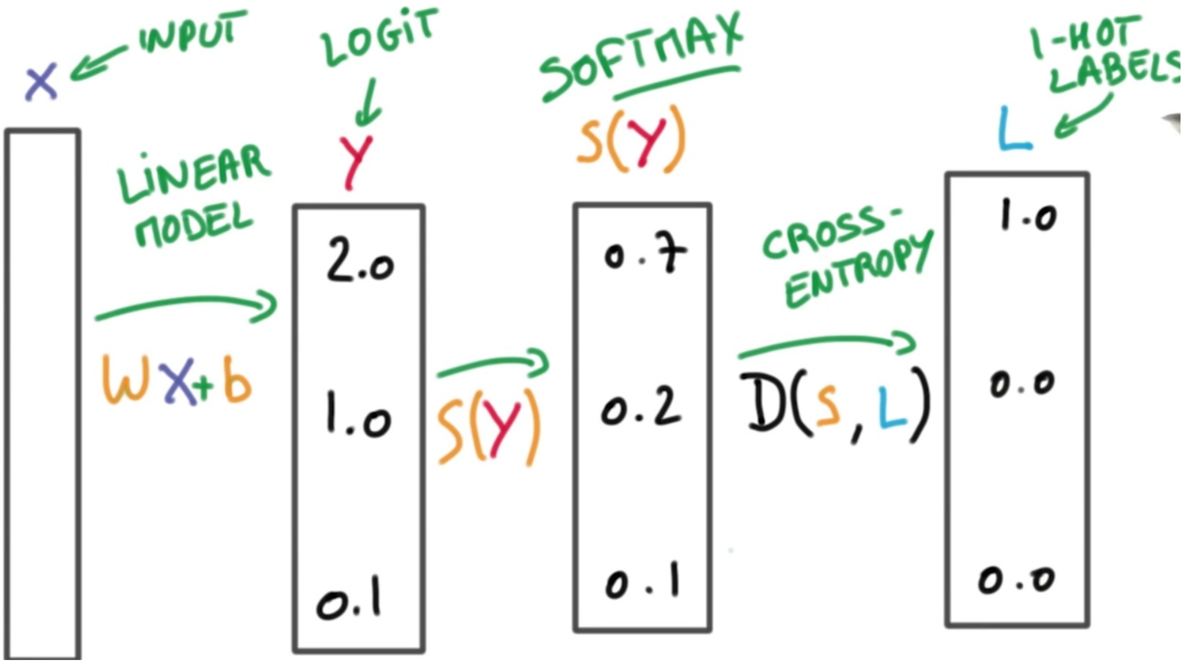
```
# Initializing the variables
     init = tf.global_variables_initializer()

     # Create a summary to monitor cost tensor
     tf.summary.scalar("loss", cost)
     # Create a summary to monitor accuracy tensor
     tf.summary.scalar("accuracy", acc)
     # Merge all summaries into a single op
     merged_summary_op = tf.summary.merge_all()
```

https://datascienceplus.com/mnist-for-machine-learning-beginners-with-softmax-regression/

At its core, most algorithms should have a proof of classification **and** this **is** nothing more than keeping track of which feature gives evidence to which **class**. The way the features are designed determines the model that **is** used to learn. This can be a confusion matrix, a certain confidence interval, a T-Test statistic, p-value **or** something **else** used **in** hypothesis[1] testing.

http://www.softwareschule.ch/examples/decision.jpg

Lets start with some code snippets to grape the 5 steps, assuming that you have Python or maXbox already installed (everything at least **as** recent **as** 2.7 should be fine **or** better 3.6 **as** we do), we need to install NumPy **and** SciPy **for** numerical operations, **as** well **as** matplotlib **and** sklearn **for** visualization:



Pre-made Estimators enable you to work at a much higher conceptual level than the base TensorFlow APIs. You no longer have to worry about creating the

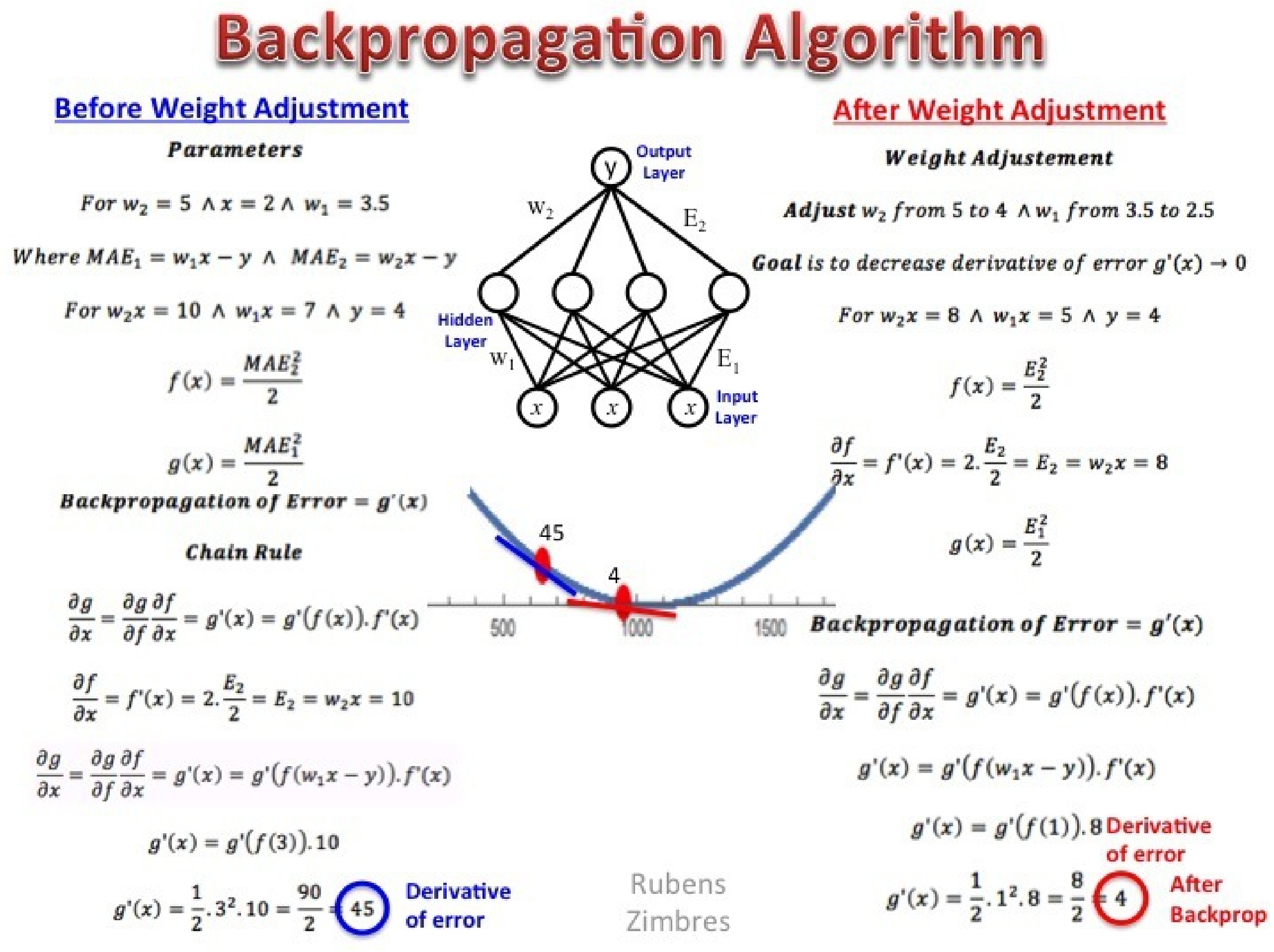
1 A thesis with evidence

computational graph or sessions since Estimators handle all the "plumbing" for you. Estimators are themselves built on tf.keras.layers, which simplifies customization.
https://stackoverflow.com/questions/34240703/what-is-logits-softmax-and-softmax-cross-entropy-with-logits

TensorFlow provides several operations that help you perform classification.

- tf.nn.sigmoid_cross_entropy_with_logits
- tf.nn.softmax
- tf.nn.log_softmax
- tf.nn.softmax_cross_entropy_with_logits
- tf.nn.softmax_cross_entropy_with_logits_v2 - identical to the base version, except it allows gradient propagation into labels.
- tf.nn.sparse_softmax_cross_entropy_with_logits
- tf.nn.weighted_cross_entropy_with_logits

Backpropagation is a method used in artificial neural networks to calculate a gradient that is needed in the calculation of the weights to be used in the network.[1]
Backpropagation is shorthand for "the backward propagation of errors," since an error is computed at the output and distributed backwards throughout layers.



Before Weight Adjustment

Parameters

$For\ w_2 = 5 \wedge x = 2 \wedge w_1 = 3.5$

$Where\ MAE_1 = w_1x - y \wedge MAE_2 = w_2x - y$

$For\ w_2x = 10 \wedge w_1x = 7 \wedge y = 4$

$$f(x) = \frac{MAE_2^2}{2}$$

$$g(x) = \frac{MAE_1^2}{2}$$

Backpropagation of Error $= g'(x)$

Chain Rule

$$\frac{\partial g}{\partial x} = \frac{\partial g}{\partial f}\frac{\partial f}{\partial x} = g'(x) = g'(f(x)).f'(x)$$

$$\frac{\partial f}{\partial x} = f'(x) = 2.\frac{E_2}{2} = E_2 = w_2x = 10$$

$$\frac{\partial g}{\partial x} = \frac{\partial g}{\partial f}\frac{\partial f}{\partial x} = g'(x) = g'(f(w_1x - y)).f'(x)$$

$$g'(x) = g'(f(3)).10$$

$$g'(x) = \frac{1}{2}.3^2.10 = \frac{90}{2} = \boxed{45}$$ Derivative of error

After Weight Adjustment

Weight Adjustement

Adjust w_2 *from* 5 *to* 4 $\wedge\ w_1$ *from* 3.5 *to* 2.5

Goal *is to decrease derivative of error* $g'(x) \to 0$

$For\ w_2x = 8 \wedge w_1x = 5 \wedge y = 4$

$$f(x) = \frac{E_2^2}{2}$$

$$\frac{\partial f}{\partial x} = f'(x) = 2.\frac{E_2}{2} = E_2 = w_2x = 8$$

$$g(x) = \frac{E_1^2}{2}$$

Backpropagation of Error $= g'(x)$

$$\frac{\partial g}{\partial x} = \frac{\partial g}{\partial f}\frac{\partial f}{\partial x} = g'(x) = g'(f(x)).f'(x)$$

$$g'(x) = g'(f(w_1x - y)).f'(x)$$

$$g'(x) = g'(f(1)).8$$ Derivative of error

$$g'(x) = \frac{1}{2}.1^2.8 = \frac{8}{2} = \boxed{4}$$ After Backprop

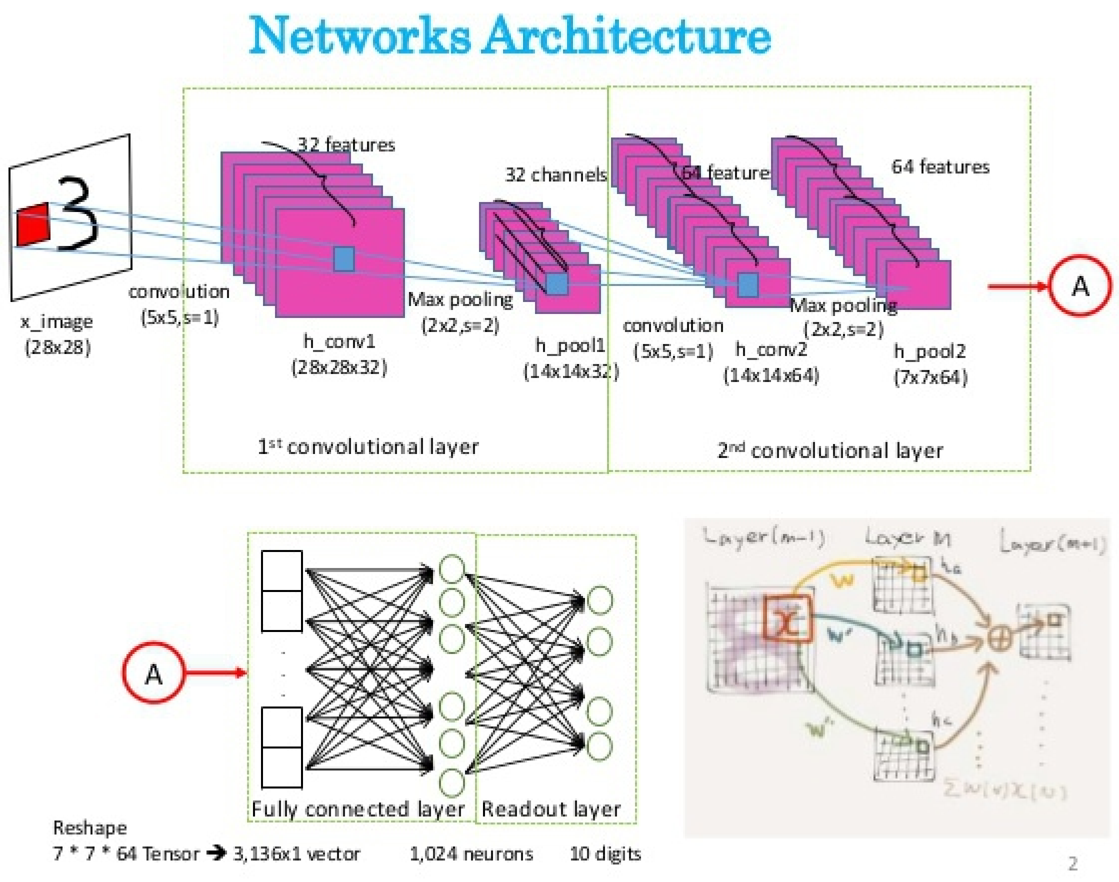
"Classification"

Now we will use linear SVC to partition our graph into clusters and split the data into a training set and a test set for further predictions.

```
 X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# Run classifier, using a model that is too regularized (C too low) to see
# the impact on the results

 classifier = svm.SVC(kernel='linear', C=0.01)
 y_pred = classifier.fit(X_train, y_train).predict(X_test)
```

By setting up a dense mesh of points in the grid and classifying all of them, we can render the regions of each cluster as distinct colors:

```
def plotPredictions(clf):
        xx, yy = np.meshgrid(np.arange(0, 250000, 10),
                        np.arange(10, 70, 0.5))
        Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

        plt.figure(figsize=(8, 6))
        Z = Z.reshape(xx.shape)
        plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
        plt.scatter(X[:,0], X[:,1], c=y.astype(np.float))
        plt.show()
```



It returns coordinate matrices from coordinate vectors. Make N-D coordinate arrays for vectorized evaluations of N-D scalar/vector fields over N-D grids, given one-dimensional coordinate arrays x1, x2,..., xn.

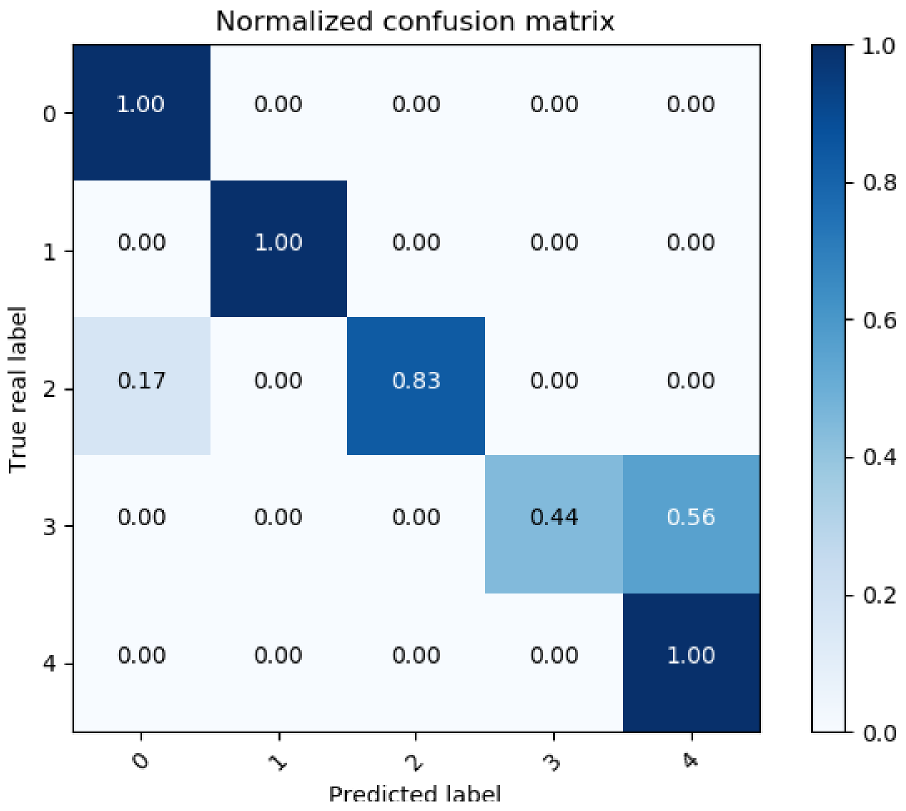
At last we start to train the model:

```
# Start Training ------------------------------------------
      print('Start Train with examples: ',mnist.train.num_examples)
      #http://maxbox8:6006
      #>>> len(mnist.train.images)
      with tf.Session() as sess:
          sess.run(init)

          # op to write logs to Tensorboard
          summary_writer = \
           tf.summary.FileWriter(logs_path, graph=tf.get_default_graph())

          # Training cycle
          for epoch in range(training_epochs):
              avg_cost = 0.
              total_batch = int(mnist.train.num_examples / batch_size)
              # Loop over all batches
              for i in range(total_batch):
                  batch_xs, batch_ys = mnist.train.next_batch(batch_size)
                  # Run optimization op (backprop), cost op (to get loss value)
                  # and summary nodes
                  _, c, summary = sess.run([optimizer, cost, merged_summary_op],
                                           feed_dict={x: batch_xs, y: batch_ys})
                  # Write logs at every iteration
                  summary_writer.add_summary(summary, epoch * total_batch + i)
                  # Compute average loss
                  avg_cost += c / total_batch
              # Display logs per epoch step
              if (epoch+1) % display_epoch == 0:
                  print("Epoch:", '%04d' % (epoch+1),
                              "cost=", "{:.9f}".format(avg_cost))

          print("EKONization Optimization Finished!")

          # Test model
          # Calculate accuracy
          print("Accuracy:", acc.eval({x: mnist.test.images,
                                              y: mnist.test.labels}))
```

Write some code, run some tests, fiddle with features, run a test, fiddle with features, realize everything is slow, and decide to use more layers or factors.

A simple CNN architecture was trained on MNIST dataset using TensorFlow with 1e-3 learning rate and cross-entropy loss using four different optimizers: SGD, Nesterov Momentum, RMSProp and Adam.
We compared different optimizers used in training neural networks and gained intuition for how they work. We found that SGD with Nesterov Momentum and Adam produce the best results when training a simple CNN on MNIST data in TensorFlow.

https://sourceforge.net/projects/maxbox/files/Docu/EKON_22_machinelearning_slides_scripts.zip/download



Last note concerning PCA and Data Reduction or Factor Analysis:

```
As PCA simply transforms the input data, it can be applied both to
classification and regression problems. In this section, we will use a
classification task to discuss the method.

The script can be found at:
  http://www.softwareschule.ch/examples/811_mXpcatest_dmath_datascience.pas
  ..\examples\811_mXpcatest_dmath_datascience.pas

  It may be seen that:

• High correlations exist between the original variables, which are
  therefore not independent

• According to the eigenvalues, the last two principal factors may be
  neglected since they represent less than 11 % of the total variance. So,
  the original variables depend mainly on the first two factors

• The first principal factor is negatively correlated with the second and
  fourth variables, and positively correlated with the third variable

• The second principal factor is positively correlated with the first
  variable

• The table of principal factors show that the highest scores are usually
  associated with the first two principal factors, in agreement with the
  previous results

Const
  N    = 11;   { Number of observations }
  Nvar = 4;    { Number of variables }
```

Of course, its not always this and that simple. Often, we dont know what number of dimensions is advisable in upfront. In such a case, we leave n_components or Nvar parameter unspecified when initializing PCA to let it calculate the full transformation. After fitting the data, explained_variance_ratio_ contains an array of ratios in decreasing order: The first value is the ratio of the basis vector describing the direction of the highest variance, the second value is the ratio of the direction of the second highest variance, and so on.

Being a linear method, PCA has, of course, its limitations when we are faced with strange data that has non-linear relationships. We wont go into much more details here, but its sufficient to say that there are extensions of PCA.

Ref:
    Building Machine Learning Systems with Python
    Second Edition March 2015
    DMath Math library for Delphi, FreePascal and Lazarus May 14, 2011

All slides and scripts of the above article:

https://sourceforge.net/projects/maxbox/files/Docu/EKON_22_machinelearning_slides_scripts.zip/download

    http://www.softwareschule.ch/box.htm

    http://fann.sourceforge.net
    http://neuralnetworksanddeeplearning.com/chap1.html
    http://people.duke.edu/~ccc14/pcfb/numpympl/MatplotlibBarPlots.html

Doc:
    Neural Networks Made Simple: Steffen Nissen
    http://fann.sourceforge.net/fann_en.pdf
    http://www.softwareschule.ch/examples/datascience.txt
    https://maxbox4.wordpress.com
    https://www.tensorflow.org/

https://sourceforge.net/projects/maxbox/files/Examples/13_General/811_mXpcatest_dmath_datascience.pas/download
https://sourceforge.net/projects/maxbox/files/Examples/13_General/809_FANN_XorSample_traindata.pas/download

Although, PCA tries to use optimization for retained variance, multidimensional scaling (MDS) tries to retain the relative distances as much as possible when reducing the dimensions. This is useful when we have a high-dimensional dataset and want to get a visual impression.

Machine learning is the science of getting computers to act without being explicitly programmed. In the past decade, machine learning has given us self-driving cars, practical speech recognition, effective web search, and a vastly improved understanding of the human genome. Machine learning is so pervasive today that you probably use it dozens of times a day without knowing it.

```
>>> Building Machine Learning Systems with Python
>>> Second Edition
```

```
ValueError: The truth value of an array with more than one element is ambiguous. Use a.any() or a.all()
```

```
ypeError: data type not understood

from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
```

HAMLET, PRINCE OF DENMARK by William Shakespeare

PERSONS REPRESENTED.

Claudius, King of Denmark.
Hamlet, Son to the former, and Nephew to the present King.
Polonius, Lord Chamberlain.
Horatio, Friend to Hamlet.
Laertes, Son to Polonius.
Voltimand, Courtier.
Cornelius, Courtier.
Rosencrantz, Courtier.
Guildenstern, Courtier.
Osric, Courtier.
A Gentleman, Courtier.
A Priest.
Marcellus, Officer.
Bernardo, Officer.
Francisco, a Soldier
Reynaldo, Servant to Polonius.
Players.
Two Clowns, Grave-diggers.
Fortinbras, Prince of Norway.
A Captain.
English Ambassadors.
Ghost of Hamlet's Father.

Gertrude, Queen of Denmark, and Mother of Hamlet.
Ophelia, Daughter to Polonius.

Lords, Ladies, Officers, Soldiers, Sailors, Messengers, and other
Attendants.

SCENE. Elsinore.