Advanced MLP

- **Advanced techniques for training neural networks**
    - **Weight Initialization**
    - **Nonlinearity (Activation function)**
    - **Optimizers**
    - **Batch Normalization**
    - **Dropout (Regularization)**
    - **Model Ensemble**

In [1]:

```
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from keras.datasets import mnist
from keras.models import Sequential
from keras.utils.np_utils import to_categorical
```

```
Using TensorFlow backend.
C:\Users\Vihan\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:516: Future
Warning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future versio
n of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\Vihan\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:517: Future
Warning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future versio
n of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\Vihan\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:518: Future
Warning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future versio
n of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\Vihan\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:519: Future
Warning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future versio
n of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\Vihan\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:520: Future
Warning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future versio
n of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\Vihan\anaconda3\lib\site-packages\tensorflow\python\framework\dtypes.py:525: Future
Warning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future versio
n of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
C:\Users\Vihan\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:541:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\Vihan\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:542:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\Vihan\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:543:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\Vihan\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:544:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\Vihan\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:545:
```

```
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\Vihan\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:550:
FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

Load Dataset

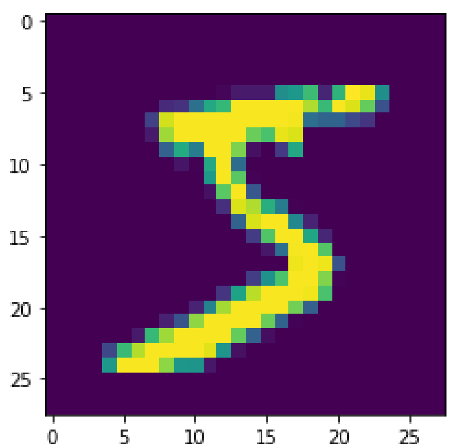
- **MNIST dataset**
- **source: http://yann.lecun.com/exdb/mnist/**

In [2]:

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

In [3]:

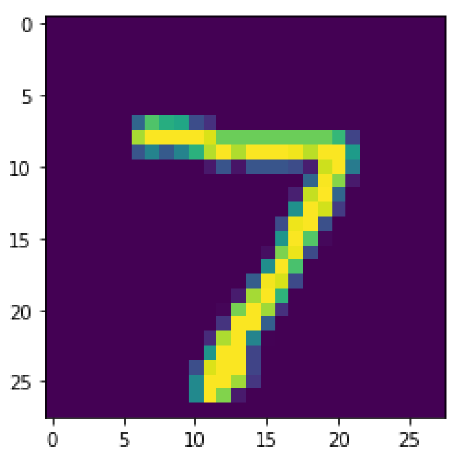
```
plt.imshow(X_train[0])     # show first number in the dataset
plt.show()
print('Label: ', y_train[0])
```




```
Label:  5
```

In [4]:

```
plt.imshow(X_test[0])     # show first number in the dataset
plt.show()
print('Label: ', y_test[0])
```




```
Label:  7
```

In [5]:

```
# reshaping X data: (n, 28, 28) => (n, 784)
X_train = X_train.reshape((X_train.shape[0], -1))
X_test = X_test.reshape((X_test.shape[0], -1))
```

In [6]:

```
# use only 33% of training data to expedite the training process
X_train, _ , y_train, _ = train_test_split(X_train, y_train, test_size = 0.67, random_state
= 7)
```

In [7]:

```
# converting y data into categorical (one-hot encoding)
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

In [8]:

```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(19800, 784) (10000, 784) (19800, 10) (10000, 10)
```

Basic MLP model

- **Naive MLP model without any alterations**

In [9]:

```
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import optimizers
```

In [10]:

```
model = Sequential()
```

In [11]:

```
model.add(Dense(50, input_shape = (784, )))
model.add(Activation('sigmoid'))
model.add(Dense(50))
model.add(Activation('sigmoid'))
model.add(Dense(50))
model.add(Activation('sigmoid'))
model.add(Dense(50))
model.add(Activation('sigmoid'))
model.add(Dense(10))
model.add(Activation('softmax'))
```

In [12]:

```
sgd = optimizers.SGD(lr = 0.001)
model.compile(optimizer = sgd, loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

In [13]:

```
history = model.fit(X_train, y_train, batch_size = 256, validation_split = 0.3, epochs = 10
0, verbose = 0)
```

```
WARNING:tensorflow:From C:\Users\Vihan\anaconda3\lib\site-packages\keras\backend\tensorflow_
backend.py:422: The name tf.global_variables is deprecated. Please use tf.compat.v1.global_v
ariables instead.
```

In [14]:

```
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.legend(['training', 'validation'], loc = 'upper left')
plt.show()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-14-4a9188539d19> in <module>
----> 1 plt.plot(history.history['acc'])
      2 plt.plot(history.history['val_acc'])
      3 plt.legend(['training', 'validation'], loc = 'upper left')
      4 plt.show()

KeyError: 'acc'
```

Training and validation accuracy seems to improve after around 60 epochs

In [ ]:

```
results = model.evaluate(X_test, y_test)
```

In [ ]:

```
print('Test accuracy: ', results[1])
```

1. Weight Initialization

- **Changing weight initialization scheme can significantly improve training of the model by preventing vanishing gradient problem up to some degree**
- **He normal or Xavier normal initialization schemes are SOTA at the moment**
- **Doc: https://keras.io/initializers/**

In [15]:

```
# from now on, create a function to generate (return) models
def mlp_model():
    model = Sequential()

    model.add(Dense(50, input_shape = (784, ), kernel_initializer='he_normal'))      # use h
e_normal initializer
    model.add(Activation('sigmoid'))
    model.add(Dense(50, kernel_initializer='he_normal'))                            # use h
e_normal initializer
    model.add(Activation('sigmoid'))
    model.add(Dense(50, kernel_initializer='he_normal'))                            # use h
e_normal initializer
    model.add(Activation('sigmoid'))
    model.add(Dense(50, kernel_initializer='he_normal'))                            # use h
e_normal initializer
    model.add(Activation('sigmoid'))
    model.add(Dense(10, kernel_initializer='he_normal'))                            # use h
e_normal initializer
    model.add(Activation('softmax'))

    sgd = optimizers.SGD(lr = 0.001)
    model.compile(optimizer = sgd, loss = 'categorical_crossentropy', metrics = ['accuracy'
])

    return model
```

In [ ]:

```
model = mlp_model()
history = model.fit(X_train, y_train, validation_split = 0.3, epochs = 100, verbose = 0)
```

In [ ]:

```
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.legend(['training', 'validation'], loc = 'upper left')
plt.show()
```

Training and validation accuracy seems to improve after around 60 epochs

In [ ]:

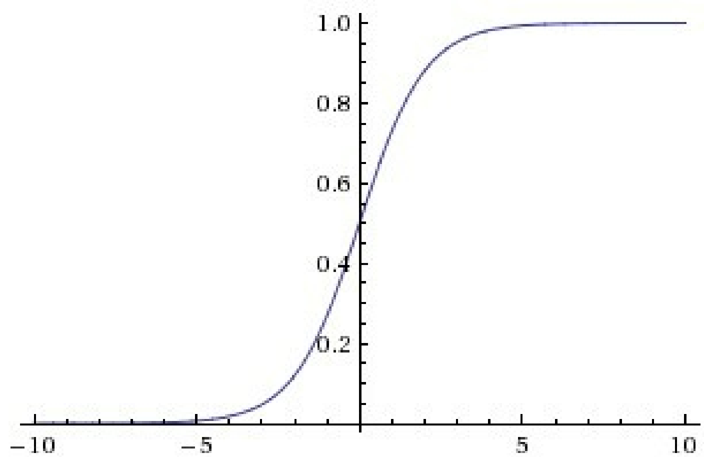
```
results = model.evaluate(X_test, y_test)
```

In [ ]:

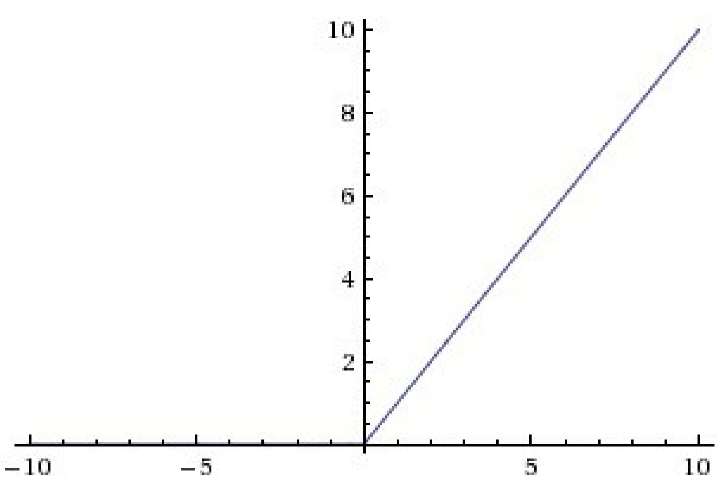
```
print('Test accuracy: ', results[1])
```

2. Nonlinearity (Activation function)

- **Sigmoid functions suffer from gradient vanishing problem, making training slower**
- **There are many choices apart from sigmoid and tanh; try many of them!**
  - **'relu' (rectified linear unit) is one of the most popular ones**
  - **'selu' (scaled exponential linear unit) is one of the most recent ones**
- **Doc: https://keras.io/activations/**
- **TODO: Explore and compare ReLU and SeLU**



****Sigmoid Activation Function****



Relu Activation Function

In [ ]:

```
def mlp_model():
    model = Sequential()

    model.add(Dense(50, input_shape = (784, )))
    model.add(Activation('relu'))     # use relu
    model.add(Dense(50))
    model.add(Activation('relu'))     # use relu
    model.add(Dense(50))
    model.add(Activation('relu'))     # use relu
    model.add(Dense(50))
    model.add(Activation('relu'))     # use relu
    model.add(Dense(10))
    model.add(Activation('softmax'))

    sgd = optimizers.SGD(lr = 0.001)
    model.compile(optimizer = sgd, loss = 'categorical_crossentropy', metrics = ['accuracy'
])

    return model
```

In [ ]:

```
model = mlp_model()
history = model.fit(X_train, y_train, validation_split = 0.3, epochs = 100, verbose = 0)
```

In [ ]:

```
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.legend(['training', 'validation'], loc = 'upper left')
plt.show()
```

Training and validation accuracy improve instantaneously, but reach a plateau after around 30 epochs

In [ ]:

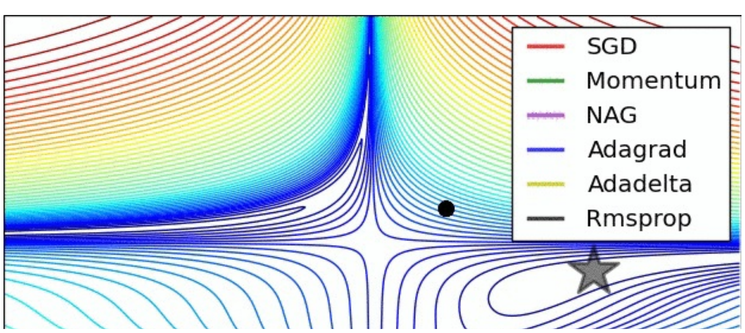
```
results = model.evaluate(X_test, y_test)
```

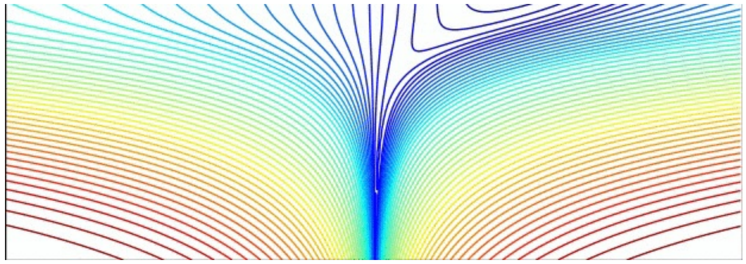
In [ ]:

```
print('Test accuracy: ', results[1])
```

3. Optimizers

- **Many variants of SGD are proposed and employed nowadays**
- **One of the most popular ones are Adam (Adaptive Moment Estimation)**
- **Doc: https://keras.io/optimizers/**





Relative convergence speed of different optimizers

In [67]:

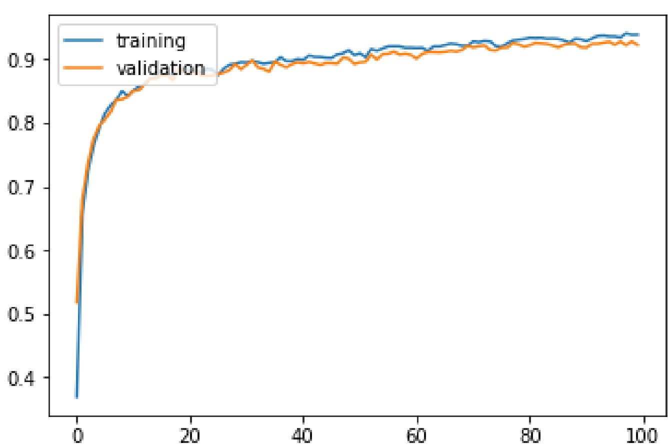
```
def mlp_model():
    model = Sequential()

    model.add(Dense(50, input_shape = (784, )))
    model.add(Activation('sigmoid'))
    model.add(Dense(50))
    model.add(Activation('sigmoid'))
    model.add(Dense(50))
    model.add(Activation('sigmoid'))
    model.add(Dense(50))
    model.add(Activation('sigmoid'))
    model.add(Dense(10))
    model.add(Activation('softmax'))

    adam = optimizers.Adam(lr = 0.001)                          # use Adam optimizer
    model.compile(optimizer = adam, loss = 'categorical_crossentropy', metrics = ['accuracy
'])

    return model
```

In [ ]:

```
model = mlp_model()
history = model.fit(X_train, y_train, validation_split = 0.3, epochs = 100, verbose = 0)
```

In [69]:

```
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.legend(['training', 'validation'], loc = 'upper left')
plt.show()
```



Training and validation accuracy improve instantaneously, but reach plateau after around 50 epochs

In [70]:

In [70]:

```
results = model.evaluate(X_test, y_test)
```

```
9472/10000 [===========================>..] - ETA: 0s
```

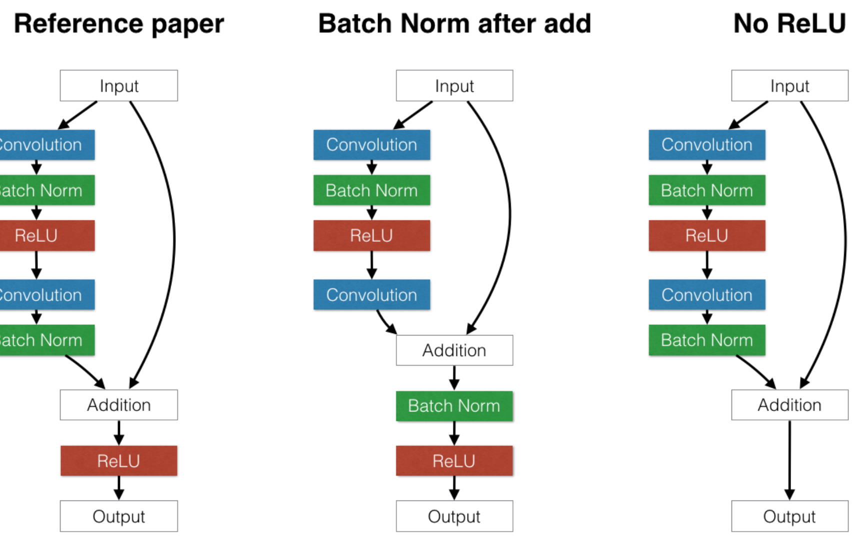
In [71]:

```
print('Test accuracy: ', results[1])
```

```
Test accuracy:  0.9248
```

4. Batch Normalization

- **Batch Normalization, one of the methods to prevent the "internal covariance shift" problem, has proven to be highly effective**
- **Normalize each mini-batch before nonlinearity**
- **Doc: https://keras.io/optimizers/**



Batch normalization layer is usually inserted after dense/convolution and before nonlinearity

In [72]:

```
from keras.layers import BatchNormalization
```

In [73]:

```
def mlp_model():
    model = Sequential()

    model.add(Dense(50, input_shape = (784, )))
    model.add(BatchNormalization())                    # Add Batchnorm layer before Activat
ion
    model.add(Activation('sigmoid'))
    model.add(Dense(50))
    model.add(BatchNormalization())                    # Add Batchnorm layer before Activat
ion
    model.add(Activation('sigmoid'))
    model.add(Dense(50))
    model.add(BatchNormalization())                    # Add Batchnorm layer before Activat
ion
    model.add(Activation('sigmoid'))
    model.add(Dense(50))
    model.add(BatchNormalization())                    # Add Batchnorm layer before Activat
ion
```

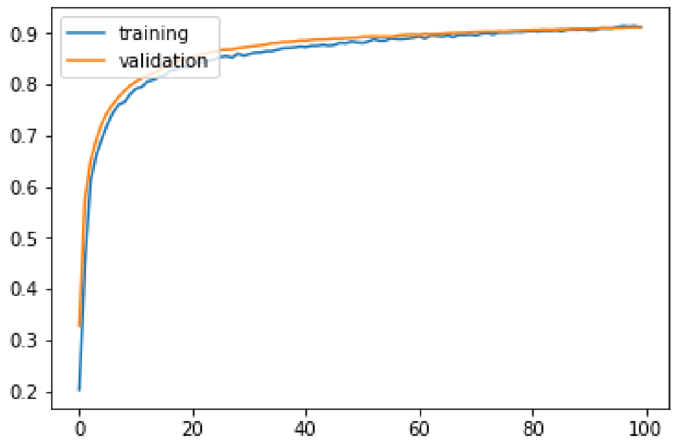
```
    model.add(Activation('sigmoid'))
    model.add(Dense(10))
    model.add(Activation('softmax'))

    sgd = optimizers.SGD(lr = 0.001)
    model.compile(optimizer = sgd, loss = 'categorical_crossentropy', metrics = ['accuracy'
])

    return model
```

In [ ]:

```
model = mlp_model()
history = model.fit(X_train, y_train, validation_split = 0.3, epochs = 100, verbose = 0)
```

In [75]:

```
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.legend(['training', 'validation'], loc = 'upper left')
plt.show()
```



Training and validation accuracy improve consistently, but reach plateau after around 60 epochs

In [76]:

```
results = model.evaluate(X_test, y_test)
```

```
 9504/10000 [===========================>..] - ETA: 0s
```

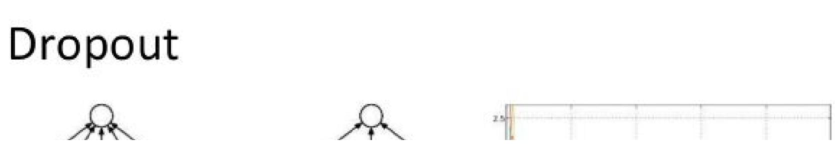
In [77]:

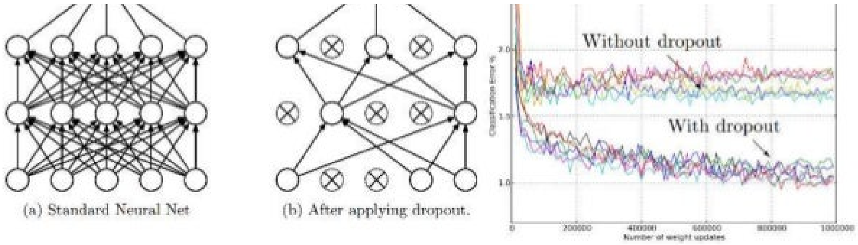
```
print('Test accuracy: ', results[1])
```

```
Test accuracy:  0.9154
```

5. Dropout (Regularization)

- **Dropout is one of powerful ways to prevent overfitting**
- **The idea is simple. It is disconnecting some (randomly selected) neurons in each layer**
- **The probability of each neuron to be disconnected, namely 'Dropout rate', has to be designated**
- **Doc: https://keras.io/layers/core/#dropout**





Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SVM on Fisher Vectors of Dense SIFT and Color Statistics	-	-	27.3
Avg of classifiers over FVs of SIFT, LBP, GIST and CSIFT	-	-	26.2
Conv Net + dropout (Krizhevsky et al., 2012)	40.7	18.2	-
Avg of 5 Conv Nets + dropout (Krizhevsky et al., 2012)	38.1	16.4	16.4

Table 6: Results on the ILSVRC-2012 validation/test set.

Dropout: A simple way to prevent neural networks from overfitting [Srivastava JMLR 2014]

In [78]:

```
from keras.layers import Dropout
```

In [79]:

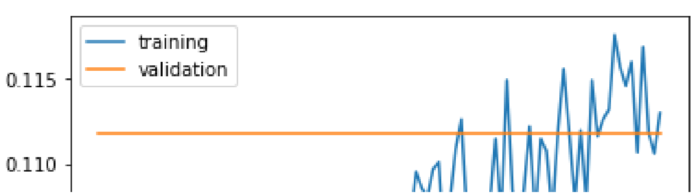
```
def mlp_model():
    model = Sequential()

    model.add(Dense(50, input_shape = (784, )))
    model.add(Activation('sigmoid'))
    model.add(Dropout(0.2))                        # Dropout layer after Activation
    model.add(Dense(50))
    model.add(Activation('sigmoid'))
    model.add(Dropout(0.2))                        # Dropout layer after Activation
    model.add(Dense(50))
    model.add(Activation('sigmoid'))
    model.add(Dropout(0.2))                        # Dropout layer after Activation
    model.add(Dense(50))
    model.add(Activation('sigmoid'))
    model.add(Dropout(0.2))                         # Dropout layer after Activation
    model.add(Dense(10))
    model.add(Activation('softmax'))

    sgd = optimizers.SGD(lr = 0.001)
    model.compile(optimizer = sgd, loss = 'categorical_crossentropy', metrics = ['accuracy'
])

    return model
```

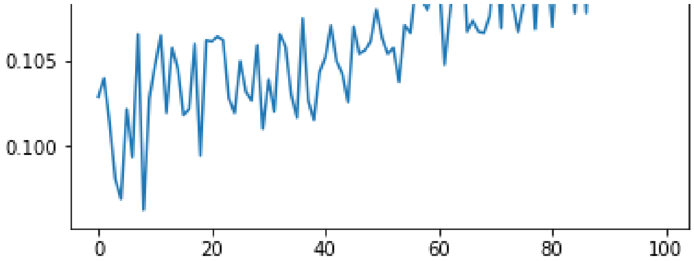
In [ ]:

```
model = mlp_model()
history = model.fit(X_train, y_train, validation_split = 0.3, epochs = 100, verbose = 0)
```

In [81]:

```
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.legend(['training', 'validation'], loc = 'upper left')
plt.show()
```





Validation results does not improve since it did not show signs of overfitting, yet.
Hence, the key takeaway message is that apply dropout when you see a signal of overfitting.

In [82]:

```
results = model.evaluate(X_test, y_test)
```

```
 9952/10000 [============================>.] - ETA: 0s
```

In [83]:

```
print('Test accuracy: ', results[1])
```

```
Test accuracy:  0.1135
```

6. Model Ensemble

- **Model ensemble is a reliable and promising way to boost performance of the model**
- **Usually create 8 to 10 independent networks and merge their results**
- **Here, we resort to scikit-learn API, VotingClassifier**
- **Doc: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html**

In [12]:

```
import numpy as np

from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import accuracy_score
```

In [13]:

```
y_train = np.argmax(y_train, axis = 1)
y_test = np.argmax(y_test, axis = 1)
```

In [14]:

```
def mlp_model():
    model = Sequential()

    model.add(Dense(50, input_shape = (784, )))
    model.add(Activation('sigmoid'))
    model.add(Dense(50))
    model.add(Activation('sigmoid'))
    model.add(Dense(50))
    model.add(Activation('sigmoid'))
    model.add(Dense(50))
    model.add(Activation('sigmoid'))
    model.add(Dense(10))
    model.add(Activation('softmax'))

    sgd = optimizers.SGD(lr = 0.001)
    model.compile(optimizer = sgd, loss = 'categorical_crossentropy', metrics = ['accuracy'
])

    return model
```

In [15]:

```
model1 = KerasClassifier(build_fn = mlp_model, epochs = 100, verbose = 0)
model2 = KerasClassifier(build_fn = mlp_model, epochs = 100, verbose = 0)
model3 = KerasClassifier(build_fn = mlp_model, epochs = 100, verbose = 0)
```

In [104]:

```
ensemble_clf = VotingClassifier(estimators = [('model1', model1), ('model2', model2), ('mod
el3', model3)], voting = 'soft')
```

In [ ]:

```
ensemble_clf.fit(X_train, y_train)
```

In [106]:

```
y_pred = ensemble_clf.predict(X_test)
```

```
 9088/10000 [==========================>...] - ETA: 0s
```

In [109]:

```
print('Test accuracy:', accuracy_score(y_pred, y_test))
```

```
Test accuracy: 0.3045
```

Slight boost in the test accuracy from the outset (0.2144 => 0.3045)

Summary

Model	Naive Model	He normal	Relu	Adam	Batchnorm	Dropout	Ensemble
Test Accuracy	0.2144	0.4105	0.9208	0.9248	0.9154	0.1135	0.3045

It turns out that most methods improve the model training & test performance. Why don't try them out altogether?