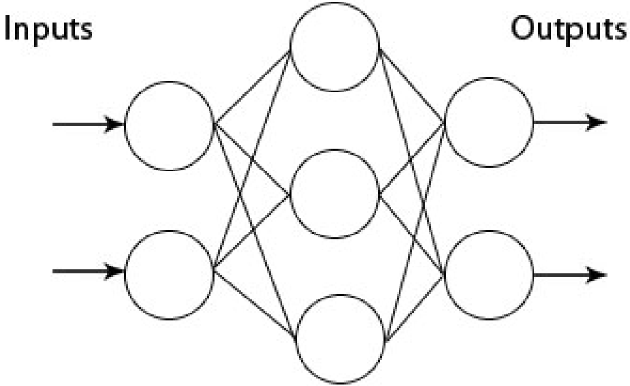
Neural Networks (Part 2)

Matthew Galbraith & Mitchell Corbett

April 7, 2015

Multilayer Perceptron Networks:



Overview

MLP: A type of feed-forward neural network

- Network composed of layers of neurons
- Each unit (neuron) in the a layer is directly connected to every unit in the following layer, starting from the input layer and ending at the output layer.
- The layers between the input and output layer are hidden layers

Multilayer Perceptron Networks

Overview (Continued)

- Whereas a true Perceptron is a binary classifier, Multilayer Perceptron networks are suited for both regression and classification problems.
- Clarification: MLP is not one perceptron, (although it's name makes it sound like that) but, a network composed of layers of perceptrons which are free to take any arbitrary activation function.

Formal definition of a MLP with one hidden layer

A one-hidden-layer MLP can be represented as a function

$$f : R^D \to R^L$$

D: size of input vector x

L: size of the output vector $f(x)$

Continued on chalkboard...

A sample function f mapping 6400 pixel images to 3 classification probabilities

So... What can we model with one-hidden-layer MLPs?

EVERYTHING!

Well, not quite everything.. But when it comes to continuous functions...

Universal Approximation Theorem

Summarized: A feed-forward network with a single hidden layer containing a finite number of neurons (Sound familiar?) can approximate continuous functions on compact subsets of R^n.

This means simple MLP networks such as the one I've been talking about can be used to model a wide variety of problems!

Training the network

High level pseudocode:

1. Initialize the weights

2. For a number of epochs, we:

2.1. Forward-pass: We compute the output of the network

2.1 We compute the loss (MSE) wrt our output

2.2 Using backpropagation and an optimization method we work backwards to update the weights to minimize the error.

Backpropagation

```
initialize network weights
loop:
forEach training example ex
  #forward-pass:
  prediction = network-output(ex)
  actual = correct-output(ex)
  compute loss at output units
  #backwards propagation:
  compute change in weights from hidden to output layer
  compute change in weights from input to hidden layer
  update network weights
if (stopping criterion satisfied):
    return network
```

Update methods

First order optimization algorithms

Stochastic Gradient Descent

Gradient Descent with Momentum

Gradient Descent with Nesterov Momentum

Update methods - SGD

Stochastic Gradient Descent

```
def sgd(loss, all_params, learning_rate):
    all_grads = theano.grad(loss, all_params)
    updates = []
    for param_i, grad_i in zip(all_params, all_grads):
        updates.append((param_i,
        param_i - learning_rate * grad_i))
    return updates
```

Update methods - Momentum

Gradient Descent with Momentum

```
def momentum(loss, all_params, learning_rate, momentum=0.9):
    all_grads = theano.grad(loss, all_params)
    updates = []

    for param_i, grad_i in zip(all_params, all_grads):
        mparam_i = theano.shared(np.zeros(param_i.get_value().shape,
                                          dtype=theano.config.floatX),
                                 broadcastable=param_i.broadcastable)
        v = momentum * mparam_i - learning_rate * grad_i
        updates.append((mparam_i, v))
        updates.append((param_i, param_i + v))

    return updates
```

Update methods - Nesterov Momentum

Accelerated Gradient Descent with Nesterov Momentum

```
# using the alternative formulation of nesterov momentum described at
# https://github.com/lisa-lab/pylearn2/pull/136
# such that the gradient can be evaluated at the current parameters.
def nesterov_momentum(loss, all_params, learning_rate, momentum=0.9):
    all_grads = theano.grad(loss, all_params)
    updates = []

    for param_i, grad_i in zip(all_params, all_grads):
        mparam_i = theano.shared(np.zeros(param_i.get_value().shape,
                                          dtype=theano.config.floatX),
                                 broadcastable=param_i.broadcastable)
        v = momentum * mparam_i - learning_rate * grad_i  # new momemtum
        w = param_i + momentum * v - learning_rate * grad_i  # new param values
        updates.append((mparam_i, v))
        updates.append((param_i, w))

    return updates
```

Moving on

MLP applications

Now that I've introduced MLP Networks..

It's applications time!

Facial Keypoint Recognition

Problem:

Kaggle competition

Observations:

7049 samples of 96x96 dimension images

Up to 15 (x, y) pairs of coordinates for each image (Some missing)

Model

We want to model some function

$$f : R^{9216} \rightarrow R^{30}$$

mapping our input image data to 15 pairs of (x, y) coordinates

Python Packages used

```
from pandas.io.parsers import read_csv
import numpy as np
import cPickle as pickle
import os.path
import matplotlib.pyplot as plt
import theano
from lasagne import layers
from lasagne.nonlinearities import sigmoid
from lasagne.updates import nesterov_momentum
from nolearn.lasagne import NeuralNet
from sklearn.utils import shuffle
```

Processing the data

Load function courtesy of Daniel Nouri's tutorial

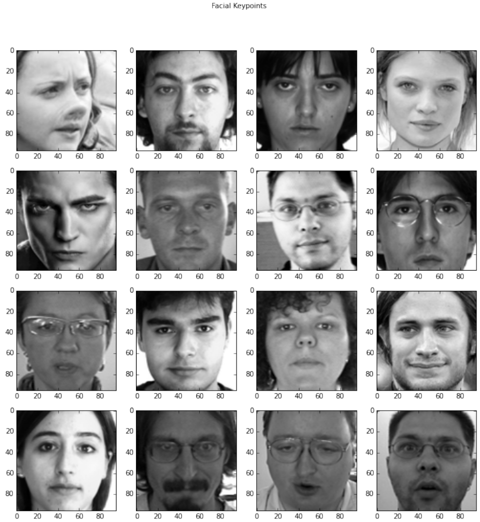
```
def load(test=False, cols=None, drop_missing=True):
    #...
    df = read_csv(os.path.expanduser(fname))  # load pandas dataframe

    df['Image'] = df['Image'].apply(lambda im: np.fromstring(im, sep=' '))
    if cols:  # get a subset of columns
        df = df[list(cols) + ['Image']]
    if (drop_missing):
        df = df.dropna()  # drop all rows that have missing values
    X = np.vstack(df['Image'].values) / 255.  # scale pixel values to [0, 1]
    X = X.astype(np.float32)

    if not test:  # only FTRAIN has any target columns
        y = df[df.columns[:-1]].values
        y = (y - 48) / 48  # scale target coordinates to [-1, 1]
        X, y = shuffle(X, y, random_state=65539)  # shuffle train data
    else:
        y = None
```

Processing the data

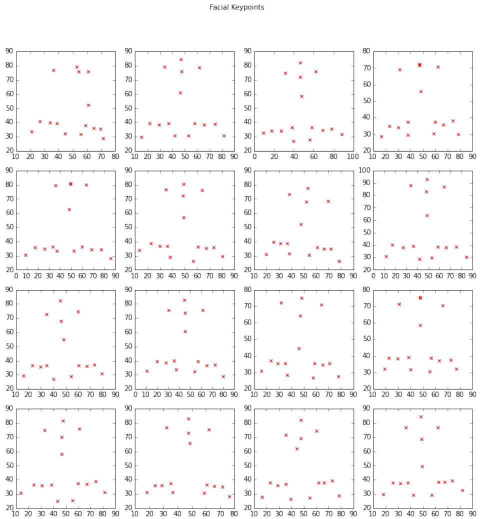
Here's what loads..

```
>>>X, y = load()
>>>print summary(X,y)
X shape: (2140L, 9216L)
y shape: (2140L, 30L)
X: min, max = (0.0,1.0)
y: min, max = (-0.920286595821,0.996020495892)
Using 30.3589161583% of the provided training data
```

Exploring the data



Exploring the data



Exploring the data

Creating NNets in Python

Here's the general workflow I followed

```
#Initialize the network
net = NeuralNet( #layers = ...
                 #more params = ...
                 #max_epochs = n
               )
X, y = load() #Load data
net.fit(X,y) #Train network
#analysis can go here...
X = load(test=false)[0] #Load test data
y_pred = net.predict(X) #Predict
#code to make Kaggle submission goes here...
```

Using Early Stopping to reduce overfitting

```
class EarlyStopping(object):
    def __init__(self, patience=100):
        self.patience = patience
        self.best_valid = np.inf
        self.best_valid_epoch = 0
        self.best_weights = None

    def __call__(self, nn, train_history):
        current_valid = train_history[-1]['valid_loss']
        current_epoch = train_history[-1]['epoch']
        if current_valid < self.best_valid:
            self.best_valid = current_valid
            self.best_valid_epoch = current_epoch
            self.best_weights = [w.get_value() for w in nn.get_all_params()]
        elif self.best_valid_epoch + self.patience < current_epoch:
            print("Early stopping.")
            print("Best valid loss was {:.6f} at epoch {}.".format(
                self.best_valid, self.best_valid_epoch))
            nn.load_weights_from(self.best_weights)
            raise StopIteration()
```

Model 1

```
#Neural Network # 1  - A MLP with 1 hidden layer
net1 = NeuralNet(
    layers=[  # three layers: one hidden layer
        ('input', layers.InputLayer),
        ('hidden', layers.DenseLayer),
        ('output', layers.DenseLayer),
        ],
    # layer parameters:
    input_shape=(None, 9216),  # 96x96 input pixels
    hidden_num_units=100,  # number of units in hidden layer
    hidden_nonlinearity=sigmoid,
    output_nonlinearity=None,  # output layer uses identity function
    output_num_units=30,  # 30 target values
    # optimization params
    update=nesterov_momentum,
    update_learning_rate=0.01,
    update_momentum=0.9,
    regression=True,
    max_epochs=1000,
    on_epoch_finished=[
        EarlyStopping(patience=50)
        ],
    verbose=1 #0 to not print anything
    )
```

Using gridsearchCV to determine appropriate number of hidden units

```
#paramter grid for gridsearch cv
param_grid = {
'more_params': [{'hidden_num_units': 100}, {'hidden_num_units':150}...,
{'hidden_num_units': 300}]
}

#find net with best params , and then refit with all data on best net
gs = GridSearchCV(net1, param_grid, cv=2, refit=True, verbose=4)
X,y=load() # Load our data
gs.fit(X,y) #Fit our gridsearchCV object
with open('net1_gridsearch.pickle', 'wb') as f:
    # we serialize the gridsearch model, as it also has the fitted
    #neural network with best hidden unit number:
    pickle.dump(gs, f, -1)
    #if you just want to save the best estimator - ie - the neural network
    #itself - without the gridsearch info:
    #pickle.dump(gs.best_estimator_, f, -1)
```

Analyzing results of the gridsearchCV

```
>>>#Load the serialized gridsearchCV object which contains our network.
>>>net1_gridsearch = pickle.load(open('./net1_gridsearch.pickle', 'rb'))
>>>#This just shows the params the gridsearch considered when finding the best m
>>>net1_gridsearch.param_grid
{'more_params': [{'hidden_num_units': 100},
  {'hidden_num_units': 150},
  {'hidden_num_units': 200},
  {'hidden_num_units': 250},
  {'hidden_num_units': 300}]}
>>>#Here are the best parameters:
>>>print net1_gridsearch.best_params_
{'more_params': {'hidden_num_units': 100}}
```

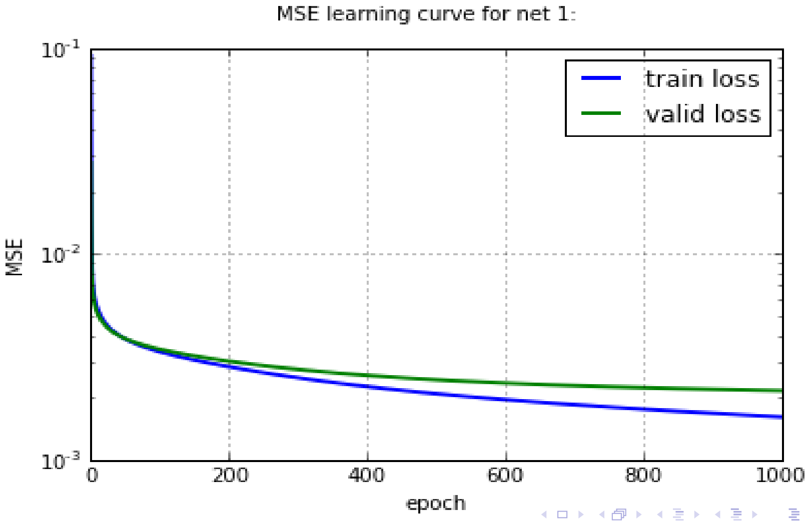
Grid Search CV results

```
>>>print "The best score (Lowest MSE) was: " + str(net1_gridsearch.best_score_)
>>>print "This translates to a RMSE (Kaggle's evaluation) score of: " +
str(np.sqrt(net1_gridsearch.best_score_) * 48.)
The best score (Lowest MSE) was: 0.0025870305397
This translates to a RMSE (Kaggle evaluation) score of: 2.44141728581
```

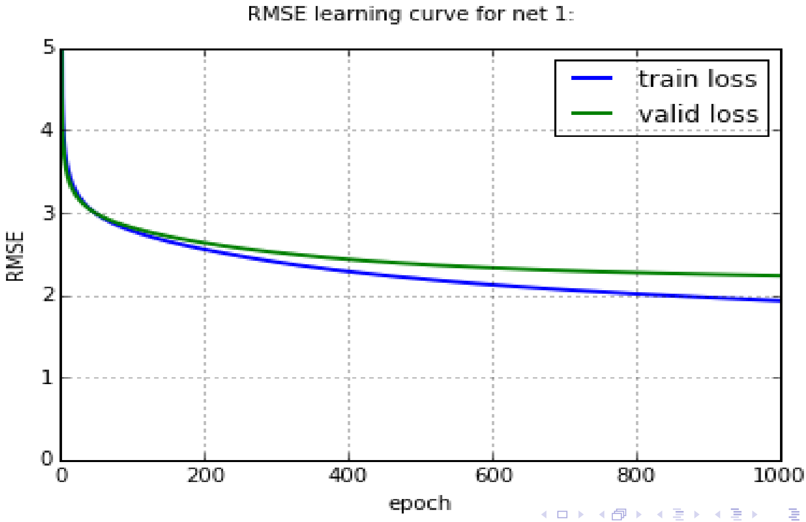
Analyzing the first Neural Network model

```
#Let's load our neural net from the gridserach object!
>>>net1 = net1_gridsearch.best_estimator_
>>>net1
NeuralNet(X_tensor_type=<function matrix at 0x...>,
     batch_iterator_test=<nolearn.lasagne.BatchIterator object at 0x...>,
     batch_iterator_train=<nolearn.lasagne.BatchIterator object at 0x...>,
     eval_size=0.2,
     hidden_nonlinearity=<theano.tensor.elemwise.Elemwise object at 0x...>,
     hidden_num_units=100, input_shape=(None, 9216),
     layers=[('input', <class 'lasagne.layers.input.InputLayer'>),
     ('hidden', <class 'lasagne.layers.dense.DenseLayer'>),
     ('output', <class 'lasagne.layers.dense.DenseLayer'>)],

     loss=<function mse at 0x...>, max_epochs=1000,
     more_params={'hidden_num_units': 100},
     on_epoch_finished=[<__main__.EarlyStopping object at 0x...>],
     on_training_finished=(), output_nonlinearity=None,
     output_num_units=30, regression=True,
     update=<function nesterov_momentum at 0x...>,
     update_learning_rate=0.01, update_momentum=0.9,
     use_label_encoder=False, verbose=1,
     y_tensor_type=TensorType(float32, matrix))
```

Model 1 - Learning Curve

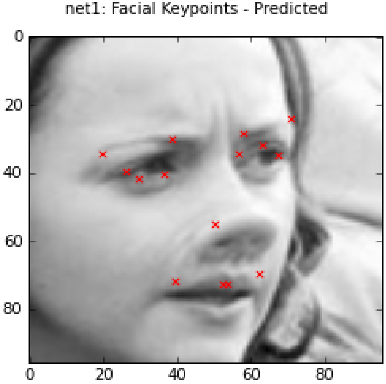


Model 1 - Learning Curve



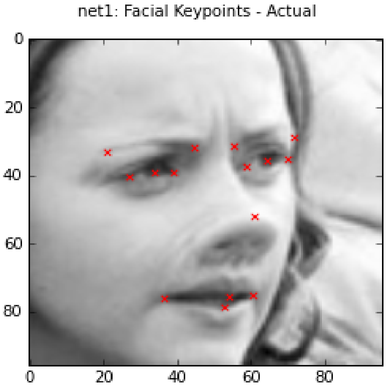
Plotting a sample prediction

Taking a sample face (from the training set), here are predicted coordinates:



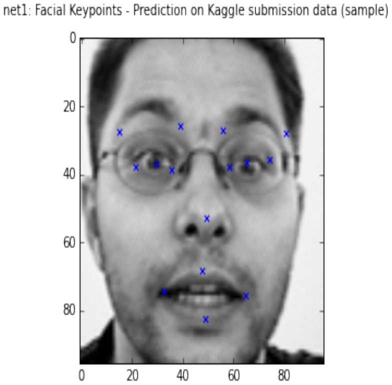
Plotting a sample prediction –pt.2

Taking a sample face (from the training set), here are the actual coordinates:



Kaggle submission –pt.1

Here's a plot of predicted keypoints for a Kaggle test sample:



Kaggle submission –pt.2

Submitting to kaggle!

Not as simple as predicting and writing to csv...

Submission only wants specific coordinates from specific images in the test set

Need to filter our predictions by the coordinates Kaggle wants

Kaggle submission –pt.3

Submission made!

30	↓5	ashok k harnal	3.55854	22	Sun, 22 Mar 2015 12:49:35 (-2.1d)
31	new	rockers	3.58801	1	Thu, 02 Apr 2015 09:03:43
32	↓6	Jia Chen	3.58884	9	Sat, 14 Mar 2015 03:40:25 (-0.2h)
33	↓6	dscrimager	3.60199	1	Fri, 20 Feb 2015 04:16:55
34	↓6	kainster	3.62655	4	Sat, 14 Mar 2015 02:44:21 (-0.8h)
35	**↑16**	**mpg317**	**3.66788**	**7**	**Mon, 06 Apr 2015 03:37:16**
36	↓7	Suresh Kumar	3.75814	4	Fri, 13 Feb 2015 13:47:47 (-26.2h)
37	↓7	Elixir	3.77399	2	Fri, 13 Mar 2015 09:08:22 (-0.2h)
38	↓7	Manohar Swamynathan	3.80685	1	Thu, 12 Feb 2015 13:24:59

Your Best Entry ↑
You improved on your best score by 0.49130.

You just moved up 25 positions on the leaderboard. Tweet this!

Model 2

Specialist Networks

- Idea: Train multiple networks on subsets of the 15 facial keypoints!
- Allows us to use more training data
- Result: 6 Specialized networks

Dynamically updating learning rate and momentum coefficients

```
#http://danielnouri.org/notes/2014/12/17/using-convolutional-neural-nets-to-dete
class AdjustVariable(object):
    def __init__(self, name, start=0.03, stop=0.001):
        self.name = name
        self.start, self.stop = start, stop
        self.ls = None

    def __call__(self, nn, train_history):
        if self.ls is None:
            self.ls = np.linspace(self.start, self.stop, nn.max_epochs)

        epoch = train_history[-1]['epoch']
        new_value = float32(self.ls[epoch - 1])
        getattr(nn, self.name).set_value(new_value)
```

The basis for our specialized neural networks

```
net = NeuralNet(
    layers=[  # three layers: one hidden layer
        ('input', layers.InputLayer),
        ('hidden', layers.DenseLayer),
        ('output', layers.DenseLayer),
        ],
    # layer parameters:
    input_shape=(None, 9216),  # 96x96 input pixels per batch
    hidden_num_units=100,  # number of units in hidden layer
    hidden_nonlinearity=sigmoid,
    output_nonlinearity=None,  # output layer uses identity function
    output_num_units=30,  # 30 target values for original, but we're changi
    # optimization params
    update=nesterov_momentum,
    update_learning_rate=theano.shared(float32(0.01)),
    update_momentum=theano.shared(float32(0.9)),
    regression=True,
    max_epochs=500,
    on_epoch_finished=[AdjustVariable('update_learning_rate', start=0.01, s
        AdjustVariable('update_momentum', start=0.9, stop=0.999),
        EarlyStopping(patience=200)
        ],
    verbose=1
    )
```

the function to fit our specialized networks

```
    for setting in SPECIALIST_SETTINGS:
        cols = setting['columns']
        X, y = load(cols=cols)

        model = clone(net)
        model.output_num_units = y.shape[1]
        # set number of epochs relative to number of training examples:
        model.max_epochs = int(1e7 / y.shape[0])
        if 'kwargs' in setting:
            # an option 'kwargs' in the settings list may be used to
            # set any other parameter of the net:
            vars(model).update(setting['kwargs'])

        print("Training model for columns {} for up to {} epochs".format(
            cols, model.max_epochs))
        model.fit(X, y)
        specialists[cols] = model

    with open('net-specialists.pickle', 'wb') as f:
        # we persist a dictionary with all models:
        pickle.dump(specialists, f, -1)
```

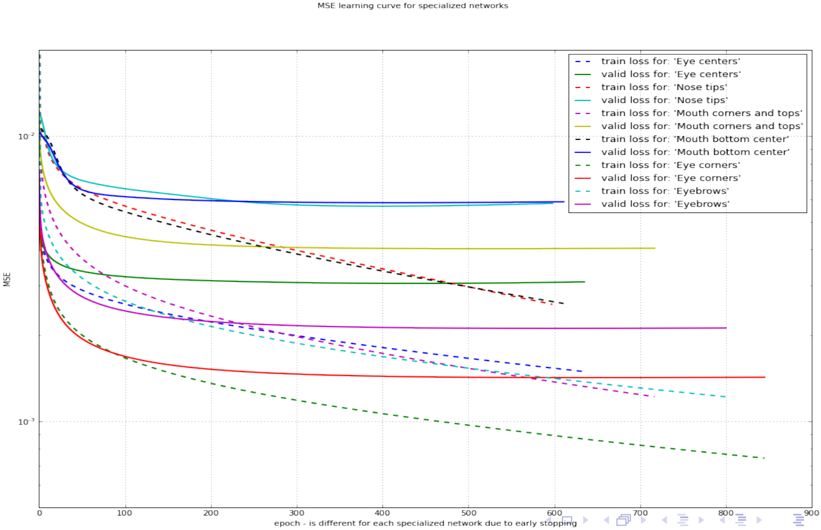
Analysis of the results - 1

```
#Load our 6 networks
net2_specialists = pickle.load(open('./net-specialists.pickle', 'rb'))
net2 = [x for x in net2_specialists.iteritems()]
#load networks into list
net2_nets = [x[1] for x in net2]
```

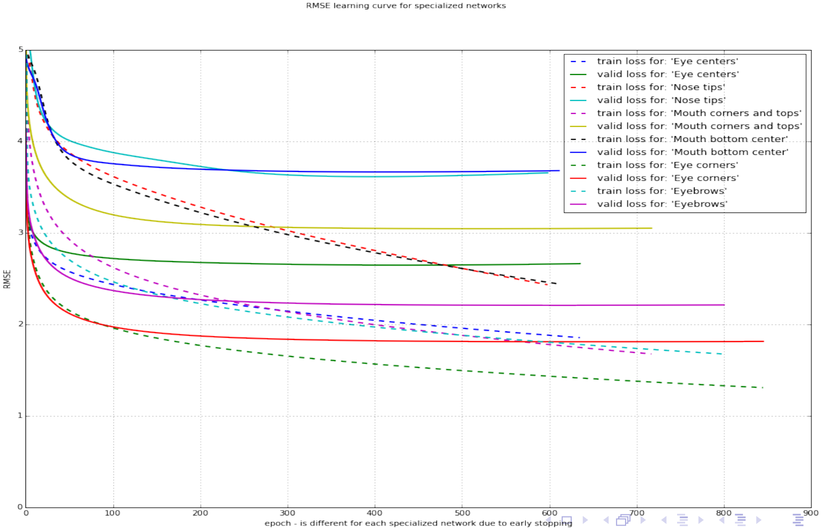
Analysis of the results - 2

```
>>>for x,net in enumerate(net2):
>>>....print summary(x, net[0])
Specialized network #1 is trained on the keypoints:
('left_eye_center_x', ...,
'right_eye_center_y')

Specialized network #2 is trained on the keypoints:
('nose_tip_x', 'nose_tip_y')

Specialized network #3 is trained on the keypoints:
('mouth_left_corner_x',...,
'mouth_right_corner_y', ..., 'mouth_center_top_lip_y')

Specialized network #4 is trained on the keypoints:
('mouth_center_bottom_lip_x', 'mouth_center_bottom_lip_y')

Specialized network #5 is trained on the keypoints:
('left_eye_inner_corner_x', ..., 'right_eye_outer_corner_y')

Specialized network #6 is trained on the keypoints:
('left_eyebrow_inner_end_x', ..., 'right_eyebrow_outer_end_y')
```

Analysis of the results - MSE Learning Curve



Analysis of the results - RMSE Learning Curve



Analysis of the results:

```
labels|train score|valid score
'Eye centers'|0.00174424204346|0.00304428217138
'Nose tips'|0.00344257359455|0.005985269282
'Mouth corners and tops'|0.00159532899415|0.00399073439391
'Mouth bottom center'|0.00331330332695|0.00583614665805
'Eye corners'|0.000931936667851|0.00145246363777
'Eyebrows'|0.00142408664489|0.00218300545633
```

Submitting to kaggle

```
preds = []
For each specialized network, net:
    Load X data for cols corresponding to net preds.append([net.predict(X)])
    #rearrange indices to follow same order as kaggle data
    preds[i] = preds[i][:, 0, 1, 4, 5, 2, 3, 6, 7]
    #stack predictions into one large array
    preds = np.hstack(preds)
    #create submission the same way as submission 1
```

Submitting to kaggle: results

Submission made!

18	↓4	Carlos Mattoso	3.03034	5	Mon, 16 Mar 2015 14:26:15 (-0.3h)
19	↓4	Florian Muellerklein	3.04421	3	Fri, 13 Mar 2015 11:14:20
20	**↑31**	**mpg317**	**3.07987**	**9**	**Mon, 06 Apr 2015 04:20:15**

Your Best Entry ↑
You improved on your best score by 0.58801.

You just moved up 15 positions on the leaderboard. Tweet this!

21	↓5	II UWr	3.15571	5	Mon, 16 Feb 2015 18:28:53
22	↓5	Steven Durand	3.16545	4	Sun, 22 Mar 2015 15:10:52 (-6.7d)
23	↓5	nitinag	3.21959	4	Tue, 17 Mar 2015 06:02:40 (-3d)
24	↓2	0xBBBB	3.26921	3	Tue, 31 Mar 2015 04:23:06

Conclusion

Neural Networks

Complex! A whole course could be devoted to neural network theory

Can be applied to many different problems

References

Useful code snippets:
http://danielnouri.org/notes/2014/12/17/using-convolutional-neural-nets-to-detect-facial-keypoints-tutorial/

1-hidden-layer MLP Mathematical definition - http://deeplearning.net/tutorial/mlp.html Universal Approximation Theorem - http://deeplearning.cs.cmu.edu/notes/Sonia$_H$*ornik.pdf*

Theano - http://deeplearning.net/software/theano/

Lasagne - http://lasagne.readthedocs.org/en/latest/

Nolearn - https://pythonhosted.org/nolearn/

references...

Nesterov Momentum -
http://www.cs.toronto.edu/ fritz/absps/momentum.pdf

Gradient Descent w/ Momentum -
http://brahms.cpmc.columbia.edu/publications/momentum.pdf

Early Stopping -
http://papers.nips.cc/paper/1895-overfitting-in-neural-nets-backpropagation-conjugate-gradient-and-early-stopping.pdf

Initial Weights -
http://stats.stackexchange.com/questions/47590/what-are-good-initial-weights-in-a-neural-network