Issues with MLPs

Dr. Fayyaz Minhas

Parameter Selection

- A MLP has a large number of parameters
  - Number of Neurons in Each Layer
  - Number of Layers
  - Activation Function for each neuron: ReLU, logsig…
  - Layer Connectivity: Dense, Dropout…
  - Objective function
    - Loss Function: MSE, Entropy, Hinge loss, …
    - Regularization: L1, L2…
  - Optimization Method
    - SGD, ADAM, RMSProp, LM …
  - Parameters for the Optimization method
    - Weight initialization
    - Momentum, weight decay, etc.

Implementation

- Keras! https://keras.io/

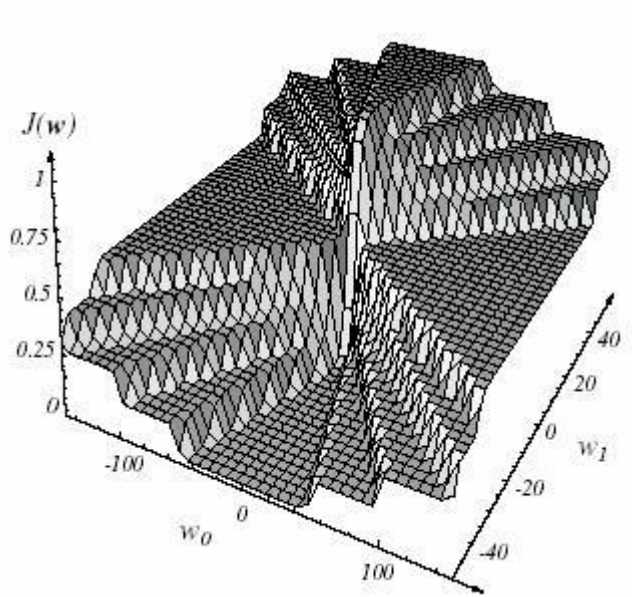
```
1  from keras.models import Sequential
2  from keras.layers import Dense
3  import numpy
4  seed = 7
5  numpy.random.seed(seed)
6  # Load the dataset
7  dataset = numpy.loadtxt("pima-indians-diabetes.csv", delimiter=",")
8  X = dataset[:,0:8]
9  Y = dataset[:,8]
10 # Define and Compile
11 model = Sequential()  # The network is not recurrent and has a sequence of layers
12 model.add(Dense(12, input_dim=8, init='uniform', activation='relu'))
13 model.add(Dense(8, init='uniform', activation='relu'))
14 model.add(Dense(1, init='uniform', activation='sigmoid'))
15 model.compile(loss='binary_crossentropy' , optimizer='adam', metrics=['accuracy'])
16 # Fit the model
17 model.fit(X, Y, nb_epoch=150, batch_size=10)
18 # Evaluate the model
19 scores = model.evaluate(X, Y)
20 print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
# Number of layers,
neurons, activations &
weight init.
```

```
# Loss function and optimization
```

Issues with Neural Networks with non-linear activations

- Unlike an SVM, which has a single global optimum due to its convex loss function, the error surface of a neural network is not as smooth
- This complicates the optimization
- A number of "tricks" are used to make the neural network learn



How to improve MLP?

- Don't let the network stop learning prematurely!
  - For example: Don't let the neurons saturate!
    - If the input or the gradient goes to zero, the learning stops!
- How to achieve?
  - Weight initialization
    - Use Nguyen-Widrow or more sophisticated weight initialization methods
    - Start with small random weights
    - Large weights will cause saturation
    - Implicit regularization!

$$\Delta w_{jk} = \alpha \delta_k z_j$$

$$\delta_k = (t_k - y_k) f'(y\_in_k)$$

$$\Delta v_{ij} = \alpha \delta_j x_i$$

$$\delta_j = \delta\_in_j \, f'(z\_in_j)$$

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk}$$

	RANDOM	NGUYEN-WIDROW
Binary XOR	2,891	1,935

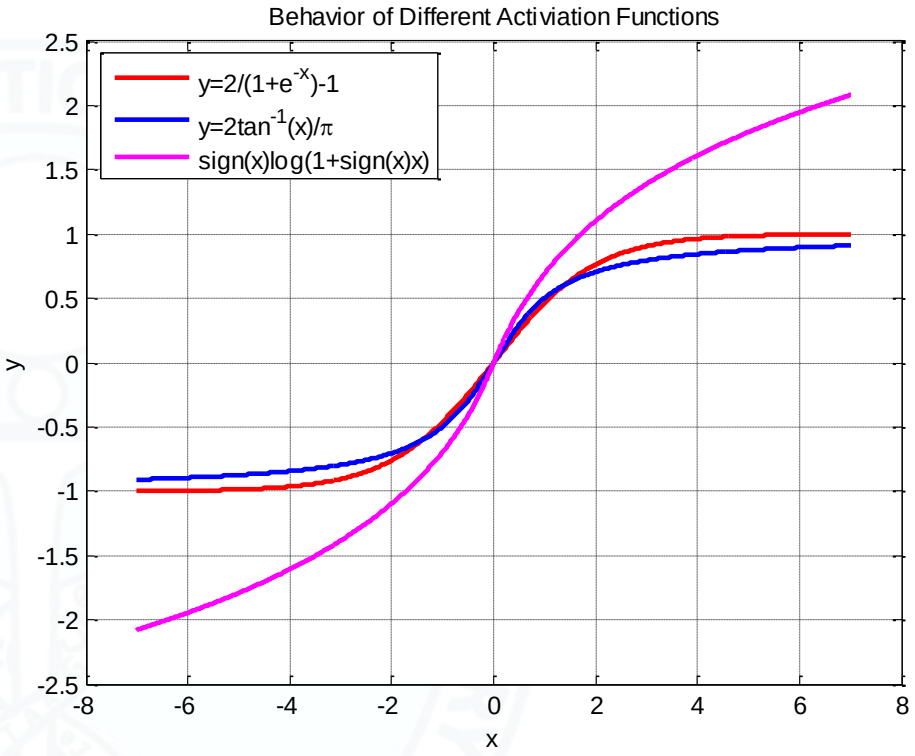
How to improve MLP?

- Changes in Data Representation
  - Bipolar inputs/targets are better than binary
    - Zeros in inputs can cause stalls
  - Using clipped bipolar outputs instead of bipolar ones
    - Sigmoidal activation functions will produce a 1.0 or 0.0 only in the asymptote

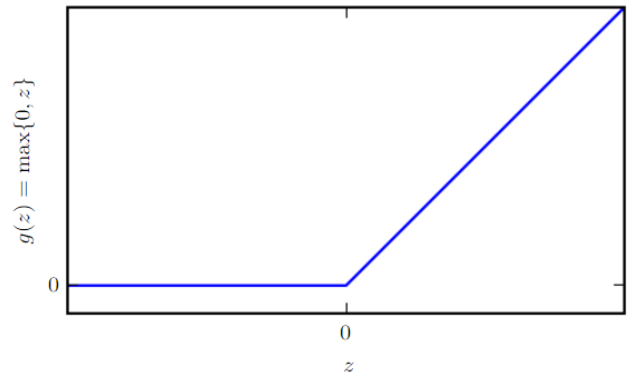
	RANDOM	NGUYEN-WIDROW
Binary XOR	2,891	1,935
Bipolar XOR	387	224
Modified bipolar XOR (targets = +0.8 and −0.8)	264	127

How to improve MLP?

- Use slowly saturating or non-saturating nonlinear activation functions
  - Examples: ReLU, Log Activation



$$f(x) = \begin{cases} \log(1 + x) & \text{for } x > 0 \\ -\log(1 - x) & \text{for } x < 0. \end{cases}$$



How to improve MLP?

- Effect of log activation

PROBLEM	LOGARITHMIC	BIPOLAR SIGMOID
standard bipolar XOR	144 epochs	387 epochs
modified bipolar XOR (targets of +0.8 or −0.8)	77 epochs	264 epochs

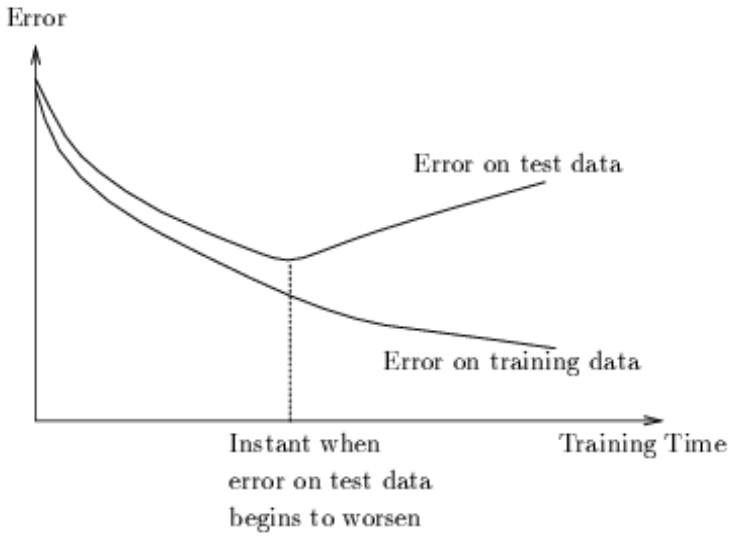
Improvements in Optimization

- Use stochastic gradient updates with mini-batches
  - Easy parallelization
- Change learning rate adaptively
- Use momentum ($0 \leq \mu \leq 1$) based update
  - but too much momentum may cause you to overshoot the local minima

$$\Delta w_{jk}(t+1) = \alpha \delta_k z_j + \mu \Delta w_{jk}(t),$$

Improving MLP

- Data Augmentation
  - Create artificial examples
    - Addition of noise
    - Translation of images or other transforms
- Drop-Off
- Batch Normalization
- Use Early Stopping
  - Keep track of generalization error and stop if the generalization error does not improve enough even when the error on training data is going down



Doing all this in Keras

- Layers

```
model = Sequential()
model.add(Dense(32, input_shape=(500,)))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='rmsprop',
loss='categorical_crossentropy', metrics=['accuracy'])
```

Useful attributes of Model

`model.layers:` is a flattened list of the layers comprising the model graph.
`model.inputs:` is the list of input tensors
`model.outputs:` is the list of output tensors.

Doing all this in Keras

- Activations

```
from keras.layers import Activation, Dense
model.add(Dense(64))
model.add(Activation('tanh'))

model.add(Dense(64, activation='tanh'))
```

- Available Activation
  - Softmax
  - Elu
  - Softplus
  - Softsign
  - Relu
  - Tanh
  - Sigmoid
  - Hard Sigmoid
  - Linear

Doing all this in Keras

- Losses

```
model.compile(loss='mean_squared_error', optimizer='sgd')

from keras import losses model.compile(loss=losses.mean_squared_error, optimizer='sgd')
```

- Available
  - Mean Squared Error
  - Mean Absolute Error
  - Mean Absolute Percentage Error
  - Mean Squared Logarithmic Error
  - Squared Hinge
  - Hinge
  - Categorical Cross Entropy
  - Sparse categorical crossentropy
  - Binary Crossentropy
  - Kullback Leibler Divergence
  - Posison
  - Cosine Proximity

Doing all this in Keras

- Metrics
  - Used to evaluate model performance

```
from keras import metrics
model.compile(loss='mean_squared_error',
              optimizer='sgd',
              metrics=[metrics.mae, metrics.categorical_accuracy])
```

- Available
  - Binary Accuracy
  - Categorical Accuracy
  - Sparse Categorical Accuracy
  - Top K Categorical Accuracy
  - Custom

Doing all this in Keras

- Optimizers

```
from keras import optimizers
model = Sequential()
model.add(Dense(64, init='uniform', input_shape=(10,)) model.add(Activation('tanh'))
model.add(Activation('softmax'))
sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='mean_squared_error', optimizer=sgd)
```

- Available
  - SGD
  - RMSprop
  - Adagrad
  - AdaDelta
  - Adam
  - Adamax
  - Nadam

Doing all this in Keras

- Initializers

```
model.add(Dense(64,
        kernel_initializer='random_uniform',
        bias_initializer='zeros'))
```

Doing all this in Keras

- Regularization
  - L1 and L2
- Drop-Out
- Batch Normalization

Doing all this in Keras

- Data Augmentation
  - Noise Layer
  - ImageDataGenerator

Class Exercise!

- Requires Keras based computers
- Solve the XOR using a single hidden layer BPNN with sigmoid activations
  - See what is the effect of different parameters on the convergence characteristics of the neural network