Deep Learning

Intro to neural networks using Theano and deep learning examples
Deep Learning

• Logistic Regression
• Neural Network
• Autoencoders Example
• Recursive Neural Network Example
Logistic Regression

- probabilistic, linear classifier
- parametrized by a weight matrix $W$ and a bias vector $b$

$$P(Y = i | x, W, b) = \text{softmax}_i (Wx + b)$$

- $\text{softmax}_i (Wx + b) = \frac{e^{W_i x + b_i}}{\sum_j e^{W_j x + b_j}}$

- SoftMax a generalisation of the logistic function

$$y_{\text{pred}} = \text{argmax}_i P(Y = i | x, W, b)$$
Logistic Regression - Theano Code

• \( x = \text{T.fmatrix('x')} \)

• \( y = \text{T.lvector('y')} \)

• \( b = \text{theano.shared(numpy.zeros((10,)), name='b')} \)

• \( W = \text{theano.shared(numpy.zeros((784, 10)), name='W')} \)

• \( p_y_{\text{given}_x} = \text{T.nnet.softmax(T.dot(x, W) + b)} \)

• \( \text{get}_p_y_{\text{given}_x} = \text{theano.function(inputs=[x], outputs=p_y_{\text{given}_x})} \)

• \( \text{print 'Probability that x is of class %i is %f' % (i, get_p_y_{\text{given}_x}(x_value)[i])} \)

• \( y_{\text{pred}} = \text{T.argmax(p_y_{\text{given}_x}, axis=1)} \)

• \( \text{classify} = \text{theano.function(inputs=[x], outputs=y_{\text{pred}})} \)
Logistic Regression - Loss Function

• use the negative log-likelihood as the loss

\[ \mathcal{L}(\theta = \{W, b\}, \mathcal{D}) = \sum_{i=0}^{|\mathcal{D}|} \log(P(Y = y^{(i)}|x^{(i)}, W, b)) \]

\[ \ell(\theta = \{W, b\}, \mathcal{D}) = -\mathcal{L}(\theta = \{W, b\}, \mathcal{D}) \]

• loss = -T.mean(T.log(p_y_given_x)[T.arange(y.shape[0]), y])

• T.log(p_y_given_x) is a matrix of all classes probability for all the batches

• [T.arange(y.shape[0]), y] selects the correct class for each batch
Logistic Regression - Learning W and B

- Use gradient decent to maximize the probability of our dataset on W and B

  - \( \text{cost} = \text{classifier.negative_log_likelihood}(y) \)
  - \( \text{g}_W = \text{T.grad}(\text{cost}, \text{classifier.W}) \)
  - \( \text{g}_b = \text{T.grad}(\text{cost}, \text{classifier.b}) \)
  - \( \text{updates} = [(\text{classifier.W}, \text{classifier.W} - \text{learning_rate} \times \text{g}_W),  \)
    - \( (\text{classifier.b}, \text{classifier.b} - \text{learning_rate} \times \text{g}_b)] \)
Logistic Regression

- More info and all the code could be found here: http://deeplearning.net/tutorial/logreg.html#logreg
Neural Networks
Neural Network

• Output:

\[ f(x) = G(b^{(2)} + W^{(2)}(s(b^{(1)} + W^{(1)}x))) \]

• Where s is:

\[ \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \]

• or

\[ \text{sigmoid}(a) = \frac{1}{1 + e^{-a}} \]
Neural Network - L₁ and L₂ Regularization

• L₁ and L₂ regularization involve adding an extra term to the loss function

• \( L_1 = T.\text{sum}(\text{abs}(\text{param})) \)

• \( L_{2\text{_sqr}} = T.\text{sum}(\text{param} \times 2) \)

• \( \text{loss} = \text{NLL} + \lambda_1 \times L_1 + \lambda_2 \times L_{2\text{_sqr}} \)

• The regularization keeps our network parameters low

• \( \lambda_1 \) and \( \lambda_2 \) are hyper parameters with values ranging between 1 to 2
Neural Network - Back Propagation

• **Phase 1: Propagation:**

1. Forward propagation of a training pattern's input through the neural network in order to generate the propagation's output activations.

2. Backward propagation of the propagation's output activations through the neural network using the training pattern target in order to generate the deltas of all output and hidden neurons.
Neural Network - Back Propagation

• **Phase 2: Weight update:** For each weight-synapse

1. Multiply its output delta and input activation to get the gradient of the weight.

2. Subtract a ratio (percentage) of the gradient from the weight.
Neural Network - Back Propagation

The Error Function

\[ E = \frac{1}{2}(t - y)^2, \]

- \( E \) is the squared error
- \( t \) is the target output
- \( y \) is the actual output of the output neuron

The Actual Output Function

\[ y = \sum_{i=1}^{n} w_i x_i \]

- \( n \) is the number of input units to the neuron
- \( w_i \) is the \( i \)th weight
- \( x_i \) is the \( i \)th input value to the neuron
Neural Network - Back Propagation

\[ \frac{\partial E}{\partial w_i} = \frac{dE}{dy} \frac{dy}{d\text{net}} \frac{\partial \text{net}}{\partial w_i} \]

\[ \frac{\partial E}{\partial w_i} \] = How the error changes when the weights are changed.

\[ \frac{dE}{dy} \] = How the error changes when the output is changed.

\[ \frac{dy}{d\text{net}} \] = How the output changes when the weighted sum changes.

\[ \frac{\partial \text{net}}{\partial w_i} \] = How the weighted sum changes as the weights change.
Neural Network - Back Propagation

\[ \Delta w_i = \alpha (t - y) x_i \]
Autoencoders

- An input layer. For example, in a face recognition task, the neurons in the input layer could map to pixels in the photograph.
- A number of considerably smaller hidden layers, which will form the encoding.
- An output layer, where each neuron has the same meaning as in the input layer.
Sentiment Analysis

- Uses the sentence syntax to learn phrases representation
- Learns from a tagged dataset of parsed phrases and sentiment
- Input is two phrases
- Output is the phrases and their sentiment
- Hidden layer learns the representation of the phrase
Sentiment Analysis

sentiment

analysis

never

worked
Sentiment Analysis

sentiment

analysis

never

worked

better
Sentiment Analysis

• Online Demo: 
  http://nlp.stanford.edu:8080/sentiment/rntnDemo.html
Questions?