How to implement a neural network Intermezzo 2

This page is part of a 5 (+2) parts tutorial on how to implement a simple neural network model. You can find the links to the rest of the tutorial here:

- Part 1: Linear regression (/posts/neural_network_implementation_part01/)
- Intermezzo 1: Logistic classification function (/posts/neural_network_implementation_intermezzo01/)
- Part 2: Logistic regression (classification) (/posts/neural_network_implementation_part02/)
- Part 3: Hidden layer (/posts/neural_network_implementation_part03/)
- Intermezzo 2: Softmax classification function (/posts/neural_network_implementation_intermezzo02/)
- Part 4: Vectorization (/posts/neural_network_implementation_part04/)
- Part 5: Generalization of multiple layers (/posts/neural_network_implementation_part05/)

Softmax classification function

This intermezzo will cover:

- The softmax function (http://peterroelants.github.io/posts/neural_network_implementation_intermezzo02/#Softmax-function)
- Cross-entropy (http://peterroelants.github.io/posts/neural_network_implementation_intermezzo02/#Cross-entropy-cost-function-for-the-softmax-function) cost function

The previous intermezzo described how to do a classification of 2 classes with the help of the logistic function (http://en.wikipedia.org/wiki/Logistic_function) . For multiclass classification there exists an extension of this logistic function called the softmax function (http://en.wikipedia.org/wiki/Softmax_function) which is used in multinomial logistic regression (http://en.wikipedia.org/wiki/Multinomial_logistic_regression) . The following section will explain the softmax function and how to derive it.

In [1]:
```
# Python imports
```

Softmax function

The logistic output function (http://en.wikipedia.org/wiki/Logistic_function) described in the previous intermezzo can only be used for the classification between two target classes *[Math Processing Error]* and *[Math Processing Error]*. This logistic function can be generalized to output a multiclass categorical probability distribution by the softmax function (http://en.wikipedia.org/wiki/Softmax_function) . This softmax function *[Math Processing Error]* takes as input a *[Math Processing Error]*-dimensional vector *[Math Processing Error]* and outputs a *[Math Processing Error]*-dimensional vector *[Math Processing Error]* of real values between *[Math Processing Error]* and *[Math Processing Error]*. This function is a normalized exponential and is defined as:

[Math Processing Error]
The denominator *[Math Processing Error]* acts as a regularizer to make sure that *[Math Processing Error]*. As the output layer of a neural network, the softmax function can be represented graphically as a layer with *[Math Processing Error]* neurons.

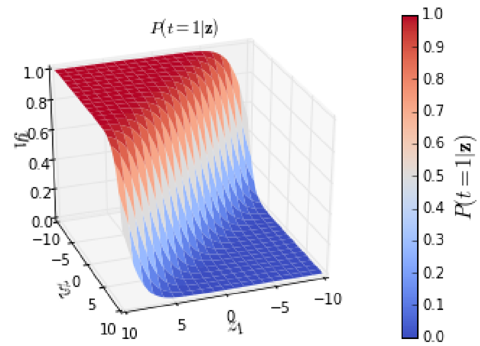
We can write the probabilities that the class is *[Math Processing Error]* for *[Math Processing Error]* given input *[Math Processing Error]* as:

[Math Processing Error]
Where *[Math Processing Error]* is thus the probability that that the class is *[Math Processing Error]* given the input *[Math Processing Error]*.

These probabilities of the output *[Math Processing Error]* for an example system with 2 classes (*[Math Processing Error]*, *[Math Processing Error]*) and input *[Math Processing Error]* is shown in the figure below. The other probability *[Math Processing Error]* will be complementary.

In [2]:
```
# Define the softmax function
def softmax(z):
    return np.exp(z) / np.sum(np.exp(z))
```

In [3]:
```
# Plot the softmax output for 2 dimensions for both classes
```



Derivative of the softmax function

To use the softmax function in neural networks, we need to compute its derivative. If we define *[Math Processing Error]* so that *[Math Processing Error]*, then this derivative *[Math Processing Error]* of the output *[Math Processing Error]* of the softmax function with respect to its input *[Math Processing Error]* can be calculated as:

[Math Processing Error]
Note that if *[Math Processing Error]* this derivative is similar to the derivative of the logistic function.

Cross-entropy cost function for the softmax function

To derive the cost function for the softmax function we start out from the likelihood function (http://en.wikipedia.org/wiki/Likelihood_function) that a given set of parameters *[Math Processing Error]* of the model can result in prediction of the correct class of each input sample, as in the derivation for the logistic cost function. The maximization of this likelihood can be written as:

[Math Processing Error]
The likelihood *[Math Processing Error]* can be rewritten as the joint probability (http://en.wikipedia.org/wiki/Joint_probability_distribution) of generating *[Math Processing Error]* and *[Math Processing Error]* given the parameters *[Math Processing Error]*: *[Math Processing Error]*. Which can be written as a conditional distribution:

[Math Processing Error]
Since we are not interested in the probability of *[Math Processing Error]* we can reduce this to: *[Math Processing Error]*. Which can be written as *[Math Processing Error]* for fixed *[Math Processing Error]*. Since each *[Math Processing Error]* is dependent on the full *[Math Processing Error]*, and only 1 class can be activated in the *[Math Processing Error]* we can write

[Math Processing Error]
As was noted during the derivation of the cost function of the logistic function, maximizing this likelihood can also be done by minimizing the negative log-likelihood:

[Math Processing Error]
Which is the cross-entropy error function *[Math Processing Error]*. Note that for a 2 class system output *[Math Processing Error]* and this results in the same error function as for logistic regression: *[Math Processing Error]*.

The cross-entropy error function over a batch of multiple samples of size *[Math Processing Error]* can be calculated as:

[Math Processing Error]
Where *[Math Processing Error]* is 1 if and only if sample *[Math Processing Error]* belongs to class *[Math Processing Error]*, and *[Math Processing Error]* is the output probability that sample *[Math Processing Error]* belongs to class *[Math Processing Error]*.

Derivative of the cross-entropy cost function for the softmax function

The derivative *[Math Processing Error]* of the cost function with respect to the softmax input *[Math Processing Error]* can be calculated as:

[Math Processing Error]
Note that we already derived *[Math Processing Error]* for *[Math Processing Error]* and *[Math Processing Error]* above.

The result that *[Math Processing Error]* for all *[Math Processing Error]* is the same as the derivative of the cross-entropy for the logistic function which had only one output node.

This post at peterroelants.github.io (http://peterroelants.github.io/posts/neural_network_implementation_intermezzo02/) is generated from an IPython notebook file. Link to the full IPython notebook file (https://github.com/peterroelants/peterroelants.github.io/blob/master/notebooks/neural_net_implementation/neural_network_implementation_intermezzo02.ipynb) (https://twitter.com/intent/tweet?source=http://peterroelants.github.io/posts/neural_network_implementation_intermezzo02/&text=How to implement a neural network Intermezzo 2%20http://peterroelants.github.io/posts/neural_network_implementation_intermezzo02/%20via%20@PeterRoelants) (https://plus.google.com/share?url=http://peterroelants.github.io/posts/neural_network_implementation_intermezzo02/) (http://www.reddit.com/submit?url=http://peterroelants.github.io/posts/neural_network_implementation_intermezzo02/&title=How to implement a neural network Intermezzo 2) (http://www.linkedin.com/shareArticle?mini=true&url=http://peterroelants.github.io/posts/neural_network_implementation_intermezzo02/&title=How to implement a neural network Intermezzo 2&source=http://peterroelants.github.io/posts/neural_network_implementation_intermezzo02/) (mailto:?subject=How to implement a neural network Intermezzo 2&body=http://peterroelants.github.io/posts/neural_network_implementation_intermezzo02/)