Multilayer Perceptron Neural Network Object 1.1

By z33z & Mokhtar M. Khorshid

This is a quick FAQ that should make up for the lack of adequate documentation, at least until we write one. Using this object might seem a bit complicated for the first time, but once you grasp the core concepts you should be able to find your way.

If after reading this you still have tons of questions or suggestions, please post them on the boards or e-mail me at MMK@Thunder-Power.com (if you do not get a reply on the boards).

Enjoy!

What does the concept of a "neural network" mean?

In our context, a neural network is a simplified model of how brains function as a collection of neurons.

What is an MLP network?

MLP is one specific neural network model that learns by experience. Think of this as a baby brain that is created without knowing anything. As time goes, you teach it things one by one and gradually it will start making relations between what it is learning so that it may apply this knowledge to new things.

What is the difference between traditional AI and neural networks?

Classical AI is concerned with mechanisms and algorithms that produce interesting results for specific problems. Most search and path finding problems are efficiently solvable by classical AI.

Neural networks on the other hand work differently. Instead of building a solution for a specific problem instance, general models are created and used in different scenarios. For example, the same neural network model that is used to classify whether a patient is susceptible to a certain disease can be used to anticipate where a projectile will fall if it was fired with a specific velocity.

How do we use it? (Overview)

Firstly, you create a network by specifying its size and properties. By size I mean the number of neurons in each layer as well as the number of layers. In particular, you will have to specify how many neurons are in the input and output layers as this will define the size of the input and output vectors you can supply. This will result in a brand new network that knows absolutely nothing and is pretty much useless.

Your newly created network may be given an input vector (collection/array) of numbers (each typically between 0 and 1) and it will respond by producing an output vector which is also a bunch of numbers. There are two ways to deal with the network: you can either take its output (and use it somehow) or you can teach (train) it what the correct output should have been for this particular input.

How exactly do we use it?

After you drop the object into the frame specify its properties and make sure you supply the correct number of neurons for the Input & Output layers. There are various other properties you can play with, but they are application dependant and are used to tweak the network.

Next thing is to create a training set. The way the object learns is that you keep adding patterns (pairs of inputs and outputs) to a set and then tell it to learn from this set. For a simple application you will probably only need one set.

After you've created your set, use "Set Pattern Input" and "Set Pattern Target" actions to create a pattern. Each of these will require an index (which is where in the vector to place the value you're supplying) that is 0-based and must be smaller than the number of neurons in Input or Output layer.

Now that you have created your pattern you can add it to the set using "Add Pattern to Set" action or you can directly use it for getting a result from the network by using "Test Pattern". When you use the "Test Pattern" action you will need to supply a result ID which is used to store the results until you use them by using the "Get Result" expression.

Adding patterns to the set does not cause any learning to take place; you will need to explicitly tell the object when to learn by using either "Train by Pattern" or "Train by Epoch" and specifying how many times it should reuse the set in training.

How long would training a network take?

This depends largely on the problem and how it is represented. Sometimes a few milliseconds every while would be sufficient, other times days would be necessary to train the network to do something.

If training is expected to take ages, how can the network be used in a game or a real-time application?

If you are anticipating long training time, then it makes sense to pre-train your network offline and include a trained network in your final product. You can also leave room for learning during runtime by using small sets of patterns.

What are the characteristics of good input/output for an MLP?

a. Values should be bounded (normalize them to be between 0 and 1)

b. Small changes in input parameters should result in small changes in output parameters, i.e. the change should be proportional (but not necessarily linear). For example, if you are using a bunch of dots as your input for a numeric handwriting recognition network, then you should not have a single output neuron representing the written digit since 3 and 8, for instance, are pretty close in shape, and yet the difference the network sees will be larger than that between 7 and 8.

c. Input should be relevant. Ideally the network should be able to detect useless input and ignore it, but if it is missing vital data to make the decision then it will be pretty difficult to get accurate results.

How accurate is an MLP?

For most complicated problems an MLP will almost never approach 100% accuracy. You should compare MLPs with how you would think about the problem not how a calculator would. They can make good high level decisions for you based on what they learn, but do not rely on MLPs for accuracy.

How can I improve the MLP accuracy?

a. The most important factor is the quality of your inputs/outputs model. No amount of tweaking can help a network if the problem is modeled poorly.

b. Setting the network properties correctly may improve the accuracy of your network. Note that you may need more training for a one setup compared to another one.

c. Train it with all sufficiently varied inputs and train it enough. Do most of your training before you ship your product.

What are hidden layers?

Hidden layers are the core of the MLP network; they allow it to solve more complicated problems. Typically you should use a single hidden layer and experiment with the number of neurons until you get the best results. Do **not** attempt to add more hidden layers unless you know what you are doing.

Are all networks expected to learn in the same way?

No. In fact the same settings for two networks trained on the same sets for the same amount may result in totally different behaviors. Just as people with similar brain structures vary in intelligence. So whenever you get an “intelligent” network, make sure not to lose it.

Will adding more neurons make the network smarter?

Not always, you should always aim at finding the right number of neurons. If the network has too little neurons then it wouldn’t have enough memory to remember what it is learning. If it has too many, it will tend to memorize patterns rather than generalize what it already knows to reach solutions.

Does more training result in more accurate networks?

Once again: not always. Once you get good results from your network save it and stop training it or at least reduce the amount of training. Networks (especially small ones) can suffer from overtraining if you train them too much.