MultiLayerPerceptron

Node: This page is always under construction.

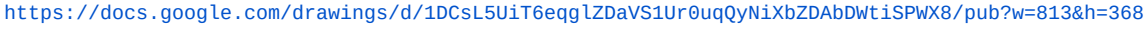
What is Multilayer Perceptron?

A multilayer perceptron (MLP) is a kind of Too feed forward artificial neural network, which is a mathematical model inspired by the biological neural network. The multilayer perceptron can be used for various machine learning tasks such as classification and regression.

The basic component of a multilayer perceptron is the neuron. In a multilayer perceptron, the neurons are aligned in layers and in any two adjacent layers the neurons are connected in pairs with weighted edges. A practical multilayer perceptron consists of at least three layers of neurons, including one input layer, one or more hidden layers, and one output layer.

The size of input layer and output layer determines what kind of data a MLP can accept. Specifically, the number of neurons in the input layer determines the dimensions of the input feature, the number of neurons in the output layer determines the dimension of the output labels. Typically, the two-class classification and regression problem requires the size of output layer to be one, while the multi-class problem requires the size of output layer equals to the number of classes. As for hidden layer, the number of neurons is a design issue. If the neurons are too few, the model will not be able to learn complex decision boundaries. On the contrary, too many neurons will decrease the generalization of the model.

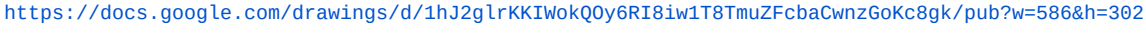
Here is an example MLP with 1 input layer, 1 hidden layer and 1 output layer:



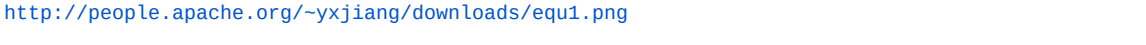
How Multilayer Perceptron works?

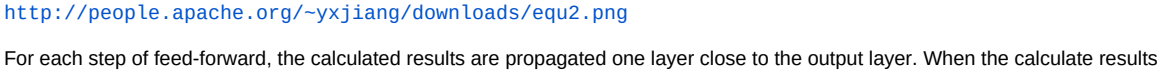
In general, people use the (already prepared) MLP by feeding the input features to the input layer and get the result from the output layer. The results are calculated in a feed-forward approach, from the input layer to the output layer.

One step of feed-forward is illustrated in the below figure.



For each layer except the input layer, the value of the current neuron is calculated by taking the linear combination of the values output by the neurons of the previous layer, where the weight determines the contribution of a neuron in the previous layer to current neuron (as equation (1) shown). Obtaining the linear combination result z, a non-linear squashing function is used to constrain the output into a restricted range (as equation (2) shown). Typically, sigmoid function or tanh function are used.





For each step of feed-forward, the calculated results are propagated one layer close to the output layer. When the calculate results are propagated to the output layer, the procedure of feed-forward finishes and the neurons of the output layer contain the final results. More details about the feed-forward calculation can be seen at UFLDL tutorial

How Multilayer Perceptron is trained in Hama?

In general, the training data is stored in HDFS and is distributed in multiple machines. In Hama, the current implementation (0.6.2 and later) allows to train the MLP in parallel. Two kinds of components are involved in the training procedure: the ***master task*** and the ***groom task***. The master task is in charge of merging the model updating information and sending model updating information to all the groom tasks. The groom tasks is in charge of calculate the weight updates according to the training data.

The training procedure is iterative and each iteration consists of two phases: *update weights* and *merge update*. In the *update weights* phase, each *groom task* would first update the local model according to the received message from the *master task*. Then they would compute the weight updates locally with assigned data partitions and finally send the updated weights to the *master task*. In the *merge update* phase, the *master task* would update the model according to the messages received from the *groom tasks*. Then it would distribute the updated model to all *groom tasks*. The two phases will repeat alternatively until the termination condition is met (reach a specified number of iterations).

How to use Multilayer Perceptron in Hama?

MLP can be used for both regression and classification. For both tasks, we need first initialize the MLP model by specifying the parameters.

Train the model

For training, the following things need to be specified:

- The ***model topology***: including the number of neurons (besides the bias neuron) in each layer; whether current layer is the final layer; the type of squashing function.
- The ***learning rate***: Specify how aggressive the model learning the training instances. A large value can accelerate the learning process but decrease the chance of model convergence. Recommend in range (0, 0.5].
- The ***momemtum weight***: Similar to learning rate, a large momemtum weight can accelerate the learning process but decrease the chance of model convergence. Recommend in range (0, 0.5].
- The ***regularization weight***: A large value can decrease the variance of the model but increase the bias at the same time. As this parameter is sensitive, it's better to set it as a very small value, say, 0.001.

The following is the sample code regarding how to train a model.

```
SmallLayeredNeuralNetwork ann = new SmallLayeredNeuralNetwork();

ann.setLearningRate(0.1); // set the learning rate
ann.setMomemtumWeight(0.1); // set the momemtum weight

// initialize the topology of the model, a three-layer model is created in this example
ann.addLayer(featureDimension, false, FunctionFactory.createDoubleFunction("Sigmoid"));
ann.addLayer(featureDimension, false, FunctionFactory.createDoubleFunction("Sigmoid"));
ann.addLayer(labelDimension, true, FunctionFactory.createDoubleFunction("Sigmoid"));

// set the cost function to evaluate the error
ann.setCostFunction(FunctionFactory.createDoubleDoubleFunction("CrossEntropy"));
String trainedModelPath = ...;
ann.setModelPath(trainedModelPath); // set the path to store the trained model

// add training parameters
Map<String, String> trainingParameters = new HashMap<String, String>();
trainingParameters.put("tasks", "5"); // the number of concurrent tasks
trainingParameters.put("training.max.iterations", "" + iteration); // the number of maximum iterations
trainingParameters.put("training.batch.size", "300");  // the number of training instances read per update
ann.train(new Path(trainingDataPath), trainingParameters);
```

The parameters related to training are listed as follows: (All these parameters are optional)

Parameter	Description
training.max. iterations	The maximum number of iterations (a.k.a. epoch) for training.
training.batch.size	As the mini-batch update is leveraged for training, this parameter specify how many training instances are used in one batch.
convergence.check. interval	If this parameters is set, then the model will be checked every time when the iteration is a multiple of this parameter. If the convergence condition is satisfied, the training will terminate immediately.
tasks	The number of concurrent tasks.

Use the trained model

Once the model is trained and stored, it can be reused later.

```
String modelPath = ...;  // the location of the existing model

DoubleVector features = ...; // the features of an instance
SmallLayeredNeuralNetwork ann = new SmallLayeredNeuralNetwork(modelPath);
DoubleVector labels = ann.getOutput(instance);  // the label evaluated by the model
```

Two class learning problem

In machine learning, two class learning is a kind of supervised learning problem. Given the instances, the goal of the classifier is to classify them into two classes.

Example: XOR problem

To be added...

Multi class learning problem

In machine learning, multiclass or multinomial classification is the problem of classifying instances into more than two classes.

Example:

To be added...

Regression problem

From the machine learning perspective, regression problem can be considered as the classification problem where the class label is a continuous value.

Example: Predict the sunspot activity

To be added...

Advanced Topics

To be added...

Parameter setting

To be added...

Reference

[1] Tom Mitchell. Machine Learning. [McGraw] Hill, 1997.

[2] Stanford Unsupervised Feature Learning and Deep Learning tutorial. http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial.

[3] Jiawei Han and Micheline Kamber. Data Mining Concept and Technology. The Morgan Kaufmann Series in Data Management Systems. 2011.

[4] Christopher M. Bishop. Neural Networks and Pattern Recognition. Oxford University Press. 1995.