TDT4171 Artificial Intelligence Methods

Exercise 5

April 8th, 2015

- **Delivery deadline: Wednesday April 22** by 23:59.
- Required reading for this assignment: Chapter 18.7 – it is a good idea to have a basic understanding of how the backpropagation algorithm works before beginning this exercise.
- Deliver your solution on *It's Learning*
- Students may work alone or in groups of two.
- Homeworks can be answered in English or in Norwegian.
- This homework counts for 3% of the final grade.
- The homework is graded on a pass/fail basis. A pass grade will only be given when a decent attempt has been made to solve **all** parts of the exercise.
- Cribbing from other students (koking) is not accepted, and if detected will lead to the assignment being failed.

Exercise: Learning to Rank with a Neural Network

An important application area of Artificial Intelligence is algorithms for ranking. "Learning to Rank" is a machine-learning approach to ranking that is used by several of the largest online search engines, such as Yahoo and Bing.

The idea is that for any query supplied by the user, we want our model to automatically give us a ranked list of the most relevant documents. To avoid having to do an evaluation of each document on the Internet, some steps are first taken to retrieve the most relevant documents (these steps are not discussed here, to limit the scope of this exercise). Then, only those documents are ranked using the learned model.

Learning to Rank algorithms operate on data generated by human rankers. A set of queries and documents have been evaluated by humans, and the model tries to learn which features separate a highly ranked document from one with a lower rank. In other words, this is a *supervised* learning algorithm, which uses the human experts' ranking as the target.

The input to the algorithm consists of a number of queries along with sites ranked for their match with the given query. Each line in the dataset has the following contents:

1. A single integer, ranking how well this document matches the given query. In our case, we use the values 0, 1 and 2, signifying "bad", "medium" and "good" matches, respectively.
2. A query id. This is important, because we are not trying to learn that "site x is generally better than site y", but that "for query q, site x is better than site y".
3. A number of features known to be relevant for ranking. In our dataset, we have 46 features, giving information about the site, the query and the combination of the two (e.g. the number of matching words). For a list of all 46 features, see [2].

Interestingly, we don't have to know what the features we base the ranking on are. It's up to our model to learn which features are important for ranking, and learn how to use them. The supplied data is part of the popular LETOR 4.0 dataset for Learning to Rank.

RankNet

RankNet [1] uses the backpropagation algorithm (see Figure 18.24 on page 734 in the course textbook) to train a Neural Network how to rank. A variant of this algorithm is used by Microsofts' search engine Bing for ranking search results.

RankNet is one of several ranking algorithm that trains the ranking model by actually teaching it to *compare pairs* of documents. If the model can learn which document is the best one of any pair, it can form a full ranking by comparing pairs of documents for the given query.

Your task is to implement the RankNet algorithm and train it on the supplied training data. Backpropagation generally learns a mapping from inputs to outputs by observing many examples of inputs and their desired target output. The RankNet algorithm follows the setup of the backpropagation algorithm, but with the modification that instead of training it on *single instances of data*, we train it on *pairs of instances*. We supply you with Python code you can use as a basis for your implementation. If you are using a different language, feel free to use this as pseudocode to start your implementation. You may also implement the algorithm yourself based on the pseudocode for backpropagation learning in the textbook.

To implement RankNet, some extra steps are required, in addition to the regular backpropagation algorithm. The following steps are required to prepare the data for training:

- Read the dataset, partition it into one part for each query (the supplied code helps you do this).
- For each query, extract all *pairs* of rated sites. Pairs with the *same rating* can be skipped. We only train on *differently ranked* sites for the *same*

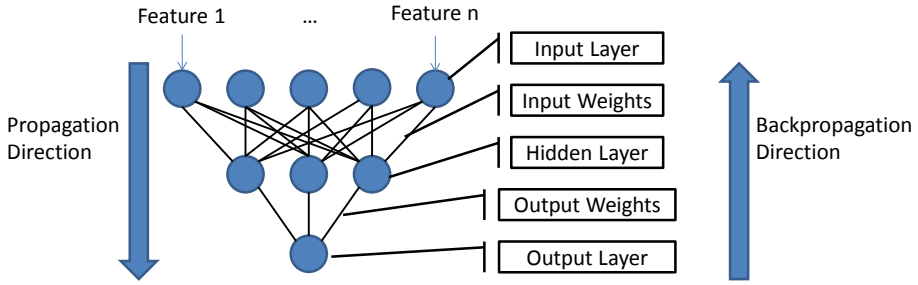

Figure 1: A smaller version of the ANN we provide in this exercise. The actual network has more input nodes and hidden nodes. The equations below calculate deltas for the hidden and output layers, and use these deltas to update both input and output weights.

query.

- Feed these pairs into the neural network. The highest rated site should always be passed through first.

For the backpropagation algorithm, instead of backpropagating for each input, we backpropagate for each *pair* of inputs. Denoting the pair as (a, b), where a is the highest rated site, we perform the following steps:

- Propagate a through the network (IMPORTANT: a is always the highest rated site of the pair)
- Propagate b through the network
- Do the backpropagation step (Equations 1-6 below) based on the state of the network during propagation of a and b.

Propagating the signal is handled by the supplied code. You will implement the backpropagation step. It may be helpful to refer to Figure 1 while going through the rest of the exercise text, to get an understanding of which updates should happen where in the network. Below follow equations for doing the backpropagation. Notice that they are quite similar to the equations in Figure 18.24, the main difference being that they operate on *pairs* of inputs. We also make the assumption here and in the supplied code that we have a *single output*, making some of the calculation simpler.

Equations for updating delta in the output layer:

$$P_{ab} \leftarrow \frac{1}{1 + e^{-(o_a - o_b)}} \tag{1}$$

$$\Delta[o]_a \leftarrow g'(o_a) \times (1 - P_{ab}) \tag{2}$$

$$\Delta[o]_b \leftarrow g'(o_b) \times (1 - P_{ab}) \tag{3}$$

Explanations for the equations: o is the single output node. P_{ab} can be interpreted as the probability our network gives to the statement "a is better ranked than b". The probability we *want* is 1. So, $(1 - P_{ab})$ is the error we have to correct. o_a and o_b are the outputs the network gives for pattern a and pattern b. $g'(x)$ is the derivative of the transfer function (supplied in the code). The result, $\Delta[o]_a$ and $\Delta[o]_b$ are the *deltas* we use to update the output weights of the network.

Equations for updating the deltas in the hidden layer (for all hidden nodes h):

$$\Delta[h]_a \leftarrow g'(out_{ha}) w_{h,o}(\Delta[o]_a - \Delta[o]_b) \tag{4}$$

$$\Delta[h]_b \leftarrow g'(out_{hb}) w_{h,o}(\Delta[o]_a - \Delta[o]_b) \tag{5}$$

out_{ha} and out_{hb} are the outputs from node h for input pattern a and b, respectively. $w_{h,o}$ is the weight from hidden node h to the single output node o. Notice that, compared to the standard backpropagation algorithm, we here propagate the *difference* of deltas from the output to the hidden nodes. The result, $\Delta[h]_a$ and $\Delta[h]_b$ are the *deltas* we use to update the input weights of the network.

Equation for updating the weights of the network (for all weights, $w_{i,j}$, in the network):

$$w_{i,j} \leftarrow w_{i,j} + \alpha \times (\Delta[j]_a \times out_{ia} - \Delta[j]_b \times out_{ib}) \tag{6}$$

Weight $w_{i,j}$ sends the output from node i to node j. This is similar to the regular weight update, but once again we are interested in optimizing the *difference* between outputs, so we use differences instead of single values. α is the learning rate of the network. $\Delta[j]_a$ and $\Delta[j]_b$ are the already calculated deltas of node j, for input pattern a and b. out_{ia} and out_{ib} are the outputs of node i for inputs a and b. This update is calculated for *all weights* in the network.

Note: In the above equations, the two patterns we compare are called a and b. In the supplied code, we implement this simply by keeping track of *current* and *previous* levels. The previous level of a variable then corresponds to pattern a, and the current one to pattern b. Also note that in the code, what we call the nodes' "activation" level is the same as the nodes' "output" in the equations above.

If successful, after training with this algorithm, the network has learned which types of features to prefer. When we feed it with test input, it will output a single value indicating how high a ranking the given features are indicating, with higher output values corresponding to a higher ranking. To test the learned ranking function, simply feed the network *pairs* of sites for a given query, and check which site of the pair gives the highest output. If the network agrees with the expert-supplied ratings on which site is better, we count this pair as a success, otherwise as a failure.

The tasks

The tasks in the exercise are:

- Implement RankNet using the description above. The supplied Python code has a lot of the elements you need.
- Train RankNet on the supplied training data for at least 20 epochs, and at least 5 different runs with different random initial network weights. An epoch is a single round of learning on each pair in the training set.
- Test the network: Present a graph showing how performance on the training data and testing data changes through the epochs. You can measure performance as the percentage of site pairs for all queries in the set that your model can *order correctly*. If your implementation is correct, performance should increase gradually, going towards at least 75% correctly ordered pairs on the training data. Like you did during training, do not present *equally rated pairs* to the network during testing. The graph you generate should average results over the 5 (or more) runs you did in the previous step.

Make a report of your findings including the folloing:

- A short description on what you needed to implement to get a working ranking algorithm. You do not have to describe the steps already implemented in the hand-out code – only your extensions to it.
- Briefly discuss what the test and training accuracy graphs tell you. Are they as expected?
- What do you think would happen to those graphs if you ran the network for many more epochs?
- (Optional) Did you have any specific difficulties in implementing the ranking algorithm? If so, please describe them here.

For this assignment, you are allowed to work in groups of two students (you may work alone as well). If you team up, however, you need to briefly describe the contributions of each team member in solving the task. If you are cooperating with someone, you should still only deliver the report once for the whole group. Mark your report with the names of the participants, and also use It's Learning to let the TAs know that the report is a collaborative effort.

References

[1] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. *Proceedings of the 22nd international conference on Machine learning ICML 05*, pages:89–96, 2005.

[2] Tao Qin and Tie-Yan Liu. Introducing LETOR 4.0 Datasets. Technical report, Microsoft Research Asia, 2013.