NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
TDT4171 – ARTIFICIAL INTELLIGENCE METHODS

Assignment 5

Per Magnus Veierland
permve@stud.ntnu.no

April 30, 2016

COST FUNCTION

To understand Equations 1-3 in the assignment; the cross entropy cost function from Equation 3 in [1] was considered:

$$C_{ij} = -\overline{P}_{ij} \cdot o_{ij} + \ln(1 + e^{o_{ij}}) \tag{1}$$

Where $\overline{P}_{ij}$ is the target probability of $f(\boldsymbol{x}_i) > f(\boldsymbol{x}_j)$ and $o_{ij} = f(\boldsymbol{x}_i) - f(\boldsymbol{x}_j)$. With the assignment notation this becomes:

$$C_{ab} = -\overline{P}_{ab} \cdot o_{ab} + \ln(1 + e^{o_{ab}}) = -1 \cdot o_{ab} + \ln(1 + e^{o_{ab}}) = -o_{ab} + \ln(1 + e^{o_{ab}}) \tag{2}$$

Given that the target probability of the network output given input pattern a being greater than the network output given input pattern b is $\overline{P}_{ab} = 1$ and that $o_{ab} = o_a - o_b$.

Differentiating the cost function with respect to the difference of the outputs gives the derivative of the cost function:

$$\begin{aligned}
\frac{\partial C_{ab}}{\partial o_{ab}} &= \frac{\partial(-o_{ab} + \ln(1 + e^{o_{ab}}))}{\partial o_{ab}} = \frac{\partial(-o_{ab})}{\partial o_{ab}} + \frac{\partial \ln(1 + e^{o_{ab}})}{\partial o_{ab}} \\
&= -1 + \frac{\partial \ln(1 + e^{o_{ab}})}{\partial(1 + e^{o_{ab}})} \cdot \frac{\partial(1 + e^{o_{ab}})}{o_{ab}} = -1 + \frac{1}{1 + e^{o_{ab}}} \cdot \left(\frac{\partial 1}{\partial o_{ab}} + \frac{\partial e^{o_{ab}}}{\partial o_{ab}}\right) \\
&= -1 + \frac{1}{1 + e^{o_{ab}}} \cdot \left(0 + e^{o_{ab}}\right) = -1 + \frac{e^{o_{ab}}}{1 + e^{o_{ab}}} \\
&= -1 + \frac{1}{1 + e^{-o_{ab}}} = -(1 - P_{ab})
\end{aligned} \tag{3}$$

When updating network parameters the negative of the cost derivative is used, since *gradient descent* moves against the cost gradient in order to minimize cost.

The negative of the derivative, $1 - P_{ab}$, which is used in Equations 2 and 3 of the assignment, can be rewritten as $\sigma(o_b - o_a)$, where σ is the logistic function.

IMPLEMENTATION

The implementation mostly follows the comment suggestions. A few deviations are a) training examples are shuffled for every epoch to improve convergence, b) $\sigma(x)(1-\sigma(x))$ is used as the transfer function derivative, c) $\sigma(o_b - o_a)$ is used as the negative cost function derivative.

OPTIMIZATIONS

The following optimizations were implemented to improve program runtime ($\approx$ 2X):

1. `NN.weightsInput` was changed to be `numHidden` × `numInputs`, using the hidden node as its first index. This results in the propagate method accessing weights in proper row-major order.

2. Caching was added internally to `countMisorderedPairs` such that network outputs are reused instead of reevaluated for the same data instance. This cache is recreated for every call to `countMisorderedPairs`, and provides a significant speedup since the same data instance can be part of several pairs.

3. All deep copies in `NN` were replaced with reuse of the existing "previous" lists.

RESULTS

Figure 1 shows the mean training- and test classification rates for 25 learning runs of 50 epochs each. The plot shows that the mean training classification rate surpasses the requirement of 75% after 13 epochs, peaking at 75.85% after 29 epochs. The peak mean test classification rate is 71% after 23 epochs.

The plotted classification rates follow the expectation that the training classification rate converges and increasingly indicates overfitting. With overfitting, the test classification rate shows an indication of gradual decay. Further training would likely result in the test classification rate decaying further. Ideally the training classification rate would stay the same.

Some stability issues were experienced when training. During some runs, training classification accuracy gradually dropped below 50-60% without recovering after training for many epochs (>50). This behavior was also seen by other students in other implementations. One possible explanation may be that it is caused by poor weight initialization.

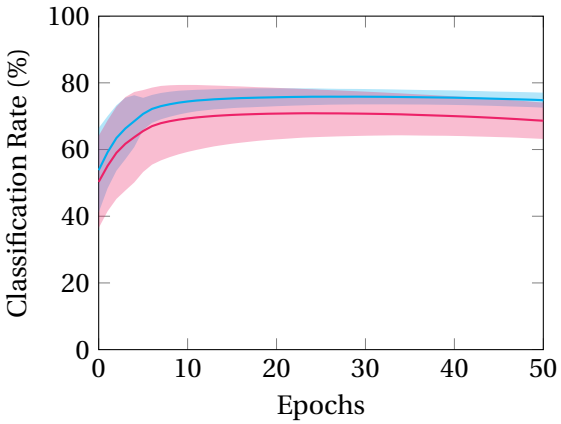

Figure 1: Average classification rate for 25 training runs of 50 epochs each, with the training data set in blue and the test data set in red. The shaded areas indicate standard deviation.

REFERENCES

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