FTDL — Take Home Exam 3
Fundamentals and Tools of Deep Learning, 2022–2023
Solutions

Subham Shome
subham.shome@estudiante.uam.es
IPCV 2022-24
Escuela Politécnica Superior — Universidad Autónoma de Madrid

May 22, 2023

1. **Argue what family of deep neural networks would not be adequate to solve a classification problem involving 10000 available patterns with 20-coordinate input vectors and two target classes in which all patterns have a non-overlapping dimension, i.e., at least one of the input coordinates has a distinct minimum value for each class in the training set.**

Ans: The classification problem described involves 10000 patterns with 20-coordinate input vectors and two target classes. Each pattern has a non-overlapping dimension, meaning that at least one of the input coordinates has a distinct minimum value for each class in the training set. To correctly classify these patterns, the neural network needs to learn which input coordinates are most informative for the classification task.

A shallow neural network with only one hidden layer may not be adequate to solve this problem. Shallow neural networks are defined by a single hidden layer of neurons, and they have a limited capacity to learn complex representations of the input data. The output of a shallow neural network with one hidden layer can be represented by the following equation:

$$\mathbf{y} = \sigma(\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2) \tag{1}$$

where $\mathbf{x}$ is the input vector, $\mathbf{W}_1$ and $\mathbf{W}_2$ are weight matrices, $\mathbf{b}_1$ and $\mathbf{b}_2$ are bias vectors, and σ is the activation function. However, since shallow neural networks have a limited number of parameters to learn from, they are more likely to underfit the training data and fail to generalize to new data.

On the other hand, deep neural networks with multiple hidden layers are more suitable for this type of classification problem. Deep neural networks can learn to identify which input coordinates are most informative for the classification task and build hierarchical representations of the input data, leading to better performance in the classification task. The output of a deep neural network with L hidden layers can be represented by the following equation:

$$\mathbf{y} = \sigma(\mathbf{W}_L \sigma(\mathbf{W}_{L-1} \dots \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_{L-1}) + \cdots + \mathbf{b}_2) \tag{2}$$

where $\mathbf{W}_i$ and $\mathbf{b}_i$ are the weight matrix and bias vector for the i-th layer, respectively. By increasing the number of hidden layers, deep neural networks can learn more complex representations of the input data and improve their performance in the classification task.

2.

(a) Calculate the error in the last layer of a feed-forward neural network with a cross-entropy cost function and a bipolar sigmoid transfer function f.

(b) Express it only in terms of f and the target values in the most simplified way so that is has minimal computational cost in a raw implementation.

(c) Discuss how this error contributes to the vanishing gradient problem in the first layers and how it could be solved.

Ans:

(a) The cross-entropy cost function is defined as:

$$C = -\frac{1}{N} \sum_{n=1}^{N} \sum_{i=1}^{m} t_i^{(n)} \log(y_i^{(n)}) + (1 - t_i^{(n)}) \log(1 - y_i^{(n)}) \tag{3}$$

Taking the derivative of the cost function with respect to the output of the last layer gives:

$$\frac{\partial C}{\partial y_i^{(n)}} = -\frac{t_i^{(n)}}{y_i^{(n)}} + \frac{1 - t_i^{(n)}}{1 - y_i^{(n)}} \tag{4}$$

Using the bipolar sigmoid function as the activation function, we have:

$$y_i^{(n)} = \frac{1}{2}(f(z_i^{(n)}) + 1) \tag{5}$$

where $z_i^{(n)}$ is the input to the activation function for neuron i in training example n. Substituting this expression into the derivative of the cost function with respect to $y_i^{(n)}$, we get:

$$\frac{\partial C}{\partial z_i^{(n)}} = \frac{1}{2}(y_i^{(n)} - t_i^{(n)}) \tag{6}$$

Finally, the error in the last layer can be obtained by applying the chain rule:

$$\delta_L = \frac{\partial C}{\partial z_i^{(n)}} \frac{\partial z_i^{(n)}}{\partial w_{ij}^L} = \frac{1}{2}(y_i^{(n)} - t_i^{(n)}) f'(z_i^{(n)}) y_j^{(n-1)} \tag{7}$$

where w_{ij}^L is the weight connecting neuron j in the $(L-1)$th layer to neuron i in the Lth layer.

Simplifying this expression using the derivative of the bipolar sigmoid function, we obtain:

$$\delta_L = y_i^{(n)}(1 - y_i^{(n)})(y_i^{(n)} - t_i^{(n)}) \tag{8}$$

(b) Expressing the error only in terms of f and the target values, we can simplify the equation as follows:

$$\delta_L = f(z_L) - t \tag{9}$$

where z_L is the input to the last layer, given by:

$$z_L = \sum_{j=1}^{m_{L-1}} w_{j,L} y_j^{(L-1)} + b_L \tag{10}$$

Here, m_{L-1} is the number of neurons in the previous layer, $w_{j,L}$ is the weight connecting the j^{th} neuron in the previous layer to the L^{th} neuron in the current layer, and b_L is the bias of the L^{th} neuron in the current layer.

(c) The error in the last layer is used to calculate the gradients in the previous layers of the network during backpropagation. However, as the gradients are propagated back through the layers, they can become very small, particularly in networks with many layers. This can make it difficult to train the network effectively, as the updates to the weights become very small and the learning process slows down.

To mitigate the vanishing gradient problem, various techniques have been proposed, such as using activation functions that have a more pronounced derivative, initializing the weights of the network carefully, and using techniques such as batch normalization and residual connections. These techniques can help to ensure that the gradients do not become too small, allowing the network to be trained more effectively.

3.

(a) Specify a parallelization strategy that could be implemented for a *N*-neuron neural network if no parallelization libraries, TPUs or GPUs are available for such implementation but a multicore system is ready with a core number larger than the number of neurons in the network.

(b) Specify another parallelization strategy if just the opposite situation occurs: TPUs, GPUs and parallelization libraries are available but number of processors is considerable less than *N*. Mention speed bottleneck sources in both cases.

Ans:

(a) In situations where there is no access to parallelization libraries, TPU, or GPU, data parallelization can be used to implement an N-neuron neural network. This involves splitting the entire dataset into smaller batches, which are then processed by multiple processor cores. Each core is loaded with a complete replica of the model and runs its batch of training examples through the model. After processing, the processor cores communicate with each other to share the results for backpropagation and weight updates.

However, the downside of this approach is the speed issue arising from communication between the processors. If a large number of processors are used, they will spend more time communicating with each other, and slower processors may cause the faster ones to wait. As a result, the overall training time can increase significantly.

(b) In cases where TPUs, GPUs, and parallelization libraries are available, but the number of processors is significantly less than N, model parallelization can be used. Here, the neural network is divided and distributed among different processor cores of GPUs or TPUs, rather than the training data. All the cores process the same training data. This method works well when the model has a high number of parameters and multiple branches. During the training process, when a part of the model is trained, its output is synchronized with the next layer in another processor.

However, the bottleneck in terms of speed in this approach is the time taken to transfer data between the GPUs/TPUs. Therefore, the overall training time can increase significantly due to this communication delay.

4.

(a) **What is the gradient of the cost function in hinge loss?**

(b) **How hinge loss will perform as a cost function for a deep feed-forward neural network?**

(c) **Will there be saturation problems?**

Ans:

(a) Hinge loss is a convex function and can be optimized using common convex optimization algorithms in machine learning. Although it is not differentiable, it has a subgradient with respect to model parameters w of a linear support vector machine (SVM) with a score function $y = \mathbf{w} \cdot \mathbf{x}$ that is given by:

$$\frac{\partial l}{\partial w_i} = \begin{cases} -t \cdot x_i & \text{if } t \cdot y < 1 \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

(b) Hinge loss can perform well as a cost function for a deep feedforward neural network, especially in classification problems where the classes are not well separated. It can handle noisy or overlapping data, and encourages a margin between the decision boundary and the training examples.

(c) There can be saturation problems in hinge loss, where the gradient becomes very small or zero for examples that are correctly classified but close to the margin. This can cause slow convergence or even prevent convergence in some cases. To address this, techniques like weight decay, early stopping, or using other loss functions that do not suffer from saturation can be employed.

5. **Reasonably propose and justify your design for:**

(a) **A network type and architecture**

(b) **Activation functions**

(c) **Initialization strategy**

(d) **Learning Rule**

(e) **A subdivision of training and test sets**

(f) **optimization options**

(g) **An associated method to validate the learning in a classification problem that uses 100,000,000 patterns, each of which has 64,000 components (well-characterized images) in the input vectors and 4 components in the target output.**

(h) Indicate how would you deal the problem of several highly unbalanced classes in the training.

(i) What would change in your selection of the network if the information corresponds to time series instead of images?

Ans:

(a) For the problem of image classification, a suitable network architecture would be a Convolutional Neural Network (CNN) with max-pooling layers. Kernels of typical sizes such as 3x3 and 5x5 can be used for the convolutional layers followed by a max-pooling of 2x2. To get the final output, fully connected hidden layers can be added. In the initial model, three convolutional layers can be used, but the number of convolutional layers and hidden layers can be adjusted based on the results. Batch normalization and dropout layers can be added to the network if the model becomes overfitted.

(b) As for the activation functions, rectified linear units (ReLU) can be used for the convolutional layers and the hidden layers. This activation function is commonly used in CNNs and has shown to be effective in reducing training time and preventing the vanishing gradient problem.

(c) For the initialization strategy, the weights can be initialized using the He initialization method, which is known to perform well with ReLU activation function.

(d) As for the learning rule, the Adam optimizer can be used. This optimizer is efficient and has shown to work well in a wide range of deep learning tasks. Stochastic Gradient Descent (SGD) is another good option.

(e) For the subdivision of training and test sets, a common approach is to use 80% of the data for training and 20% for testing. However, in the case of 100,000,000 patterns, a smaller training set may be sufficient. Stratified sampling can also be used to ensure that each class is represented equally in the training and test sets.

(f) The model will be trained initially with an Adam optimizer using a default learning rate of 0.001, as it is known to work well in most cases. Later, the learning rate can be modified or the optimizer function can be changed to SGD or RMSprop, if necessary, to achieve better model accuracy. Moreover, hyperparameter tuning can be performed using techniques such as grid search or random search to find the optimal hyperparameters.

(g) To validate the learning in a classification problem that uses 100,000,000 patterns, each of which has 64,000 components in the input vectors and 4 components in the target output, cross-validation can be used as a suitable method to evaluate the performance of the model and ensure that it generalizes well to new data. The validation accuracy and validation loss on the stratified validation set can be calculated after every epoch to validate the accuracy of the model.

(h) To deal with the problem of highly unbalanced classes in the training, techniques such as oversampling the minority class, undersampling the majority class, or using class weights can be used. The class weight can be computed by using the `compute_class_weight()` function and then used during the training process to balance out the classes which have a lesser weight by using the inverse ratio. In other words, a higher weight is assigned to classes with a lower number of representatives and a lower weight will be assigned to classes with a higher number of representatives.

(i) If the information corresponds to time series instead of images, a suitable network architecture would be a Recurrent Neural Network (RNN) or a Long Short-Term Memory (LSTM) network. These types of networks are designed to process sequential data and have shown to be effective in various applications such as speech recognition and natural language processing. The activation function and initialization strategy can be similar to those used in the image classification problem. However, the learning rule and optimization options may need to be adjusted based on the specifics of the problem.

Website References

- Wikipedia
- StackExchange
- TowardsDataScience
- GoogleResearch
- MachineLearningMastery