IE4497 Homework 2

Full Name: Nguyen Tuan Anh
Matric Number: U2120070G

Question 1: Which of the following techniques can be used to reduce model overfitting? Justify your answer. Please write down all correct answers.
(i) Data augmentation
(ii) Dropout
(iii) Batch Normalization
(iv) Using Adam instead of SGD
(v) None of the Above

Answer:

(i) Data augmentation and
(ii) Dropout

To reduce model overfitting, we can use:

- **Data augmentation:** By expanding our dataset artificially, we can mitigate overfitting. For instance, if we are training for an image classification task, we can perform various image transformations to our image dataset (e.g., flipping, rotating, rescaling, shifting).

- **Dropout:** Introducing dropout as a regularization strategy in our network layers helps to randomly deactivate a fraction of the network's neurons during training. Using dropout, we can reduce interdependent learning among units, which may have led to overfitting. However, using dropout may require an increased number of training epochs for the model to achieve convergence.

While both Batch Normalization and the shift from SGD to Adam optimization are valuable techniques, their primary objectives are not to combat overfitting. Batch Normalization aims to normalize the input layer by re-centering and re-scaling, which can lead to more efficient training. Switching to Adam optimization focuses on adapting the learning rate during training, which can lead to faster convergence.

Question 2. (5 Marks) You are using gradient descent to train a neural network. Which of the following is/are true? Justify your answer. (Note: faster here is defined in terms of wall clock time)
(i) It is possible for Stochastic Gradient Descent to converge faster than Batch Gradient Descent
(ii) It is possible for Mini Batch Gradient Descent to converge faster than Stochastic Gradient Descent
(iii) It is possible for Mini Batch Gradient Descent to converge faster than Batch

Gradient Descent
(iv) It is possible for Batch Gradient Descent to converge faster than Stochastic Gradient Descent

Answer:

True answer: (i), (ii), (iii)

(i) Stochastic Gradient Descent updates the parameters for each training example, leading to faster convergence in wall clock time, especially in the early stages of training. This is because it makes frequent updates and doesn't wait to go through the entire dataset. On the other hand, Batch Gradient Descent computes the gradient of the cost function w.r.t. the parameters for the entire training dataset and, thus, updates the parameters only once per epoch, which can be significantly slower, especially with very large datasets.

(ii) Mini Batch Gradient Descent strikes a balance between the computational efficiency of Batch Gradient Descent and the update frequency of SGD. Updating parameters after computing the gradient on a small subset of the training data can reduce the variance in the parameter updates, leading to more stable convergence. It can utilize vectorized operations more efficiently than SGD, making it faster in terms of wall clock time while potentially avoiding some of the erratic behaviour of SGD.

(iii) Mini Batch Gradient Descent can converge faster in wall clock time than Batch Gradient Descent because it updates the parameters more frequently (after each mini-batch) and can make use of vectorized operations more efficiently than pure stochastic updates. It doesn't need to load the entire dataset into memory to make a single update, which can be particularly advantageous with large datasets.

For (iv), given the clarification that "faster" is defined strictly in terms of wall clock time, the statement that "It is possible for Batch Gradient Descent to converge faster than Stochastic Gradient Descent" generally would be considered incorrect in most practical scenarios, this is because in terms of wall clock time:

- SGD is often faster at the beginning of training because it starts making progress and improving the model with every single example.
- Batch Gradient Descent might not be as efficient in terms of wall clock time, especially with large datasets, because each iteration takes much longer to process due to the need to go through the entire dataset before making an update.

In conclusion, considering wall clock time, which refers to the actual elapsed time, Mini-Batch Gradient Descent tends to be the most efficient, as it combines noise reduction benefits from averaging gradients and the speed of not having to process the entire dataset at once. This is followed by SGD, which, despite being noisier, can still be fast, especially when the data is redundant. Batch Gradient Descent is usually the slowest due to the computational load of processing the entire dataset for each update.

Question 3. (5 Marks) Which of the following is/are true about dropout? Justify your answer.
(i) Dropout leads to sparsity in the trained weights
(ii) At test time, dropout is applied with inverted keep probability
(iii) The larger the keep probability of a layer, the stronger the regularization of the weights in that layer
(iv) None of the above

Answer: (iv) None of the above
Explanation:

(i) False. Instead, it helps prevent overfitting by temporarily and randomly removing units (along with their connections) from the network during training, which encourages the network to learn more robust features that are not reliant on any small set of neurons.

(ii) False. At test time, dropout is typically not applied; instead, the activations are scaled down by the keep probability used during training.

(iii) False. The keep probability refers to the proportion of neurons that are kept (i.e., not dropped out) during training. It determines the fraction of neurons that remain active during training. A larger keep probability means fewer neurons are dropped out, which implies less regularization.

Question 4: (5 Marks) Consider a step function f(x) = 1 if x>0; f(x) = 0 otherwise. Is this a good choice for hidden neuron activation functions when you need to train the network using backprop? Justify your answer.

Answer: The step function is **not a good choice** for hidden neuron activation in networks trained via backpropagation due to its non-differentiability at zero, making gradient calculation impossible. Its derivative is zero for all inputs except at the discontinuity, leading to the vanishing gradient problem, where no learning occurs. Moreover, it also lacks of incremental activation, the step function switches abruptly from 0 to 1, which means it does not allow for incremental activation levels

---

Question 5: Which of the following activation functions can lead to vanishing gradients? Justify your answer.
(i) ReLU

(ii) Tanh
(iii) Leaky ReLU
(iv) Sigmoid
(v) None of the above

Answer: (ii) Tanh and (iv) Sigmoid

Vanishing gradients occur when the gradients of the activation functions become very small, effectively preventing the weights from updating during backpropagation. Both of these functions exhibit saturation behavior that can cause gradients to become very small, potentially leading to the vanishing gradient problem in deep neural networks. In details:

(ii) For inputs with large magnitude (either positive or negative), the tanh function saturates, approaching -1 or 1, and its derivative becomes very close to zero. This makes the gradient small, potentially leading to vanishing gradients in deep networks.
(iv) Similar to tanh, the sigmoid function also saturates at its output bounds of 0 and 1 for large-magnitude inputs. Its derivative becomes very small, leading to vanishing gradients.

Question 6: Write and run a Python program to train a 1-N-1 neural network to approximate

$$g(p) = 1 + \sin\left(\frac{6\pi}{4}p\right)$$

. Search for an appropriate N. Show your figures. Submit your source code (Write your answers to this Homework in a PDF file without your source code. Put your PDF and Jupyter Notebook of this question into a zip file. Submit your zip file to NTULearn).

Answer: Please refer to the zip file for question 6