Anna Krasilnikova 2562668
Mirza Misbah Mubeen Baig 2571567
Shahzain Mehboob 2571564

Exercise 9.1 – *Dropout*

a) *Given n random variables $X_i, i = 1, \dots, n$ which are identically distributed with positive pairwise correlation ρ and $var(X_i) = \sigma^2$. Show that*

$$var(\frac{1}{n}\sum_{i=1}^{n} X_i = \rho\sigma^2 + \frac{1-\rho}{n}\sigma^2$$

Intuition:
The variance of $X_1 + X_2 + \cdots + X_n$ is the sum of $Cov(X_i, X_j)$ where the summation is over all i,j.
If X_i has variance σ^2 and X_i and X_j have correlation ρ ,
$\Longrightarrow$ the variance of the sum is:

$$n\sigma^2 + n(n-1)\sigma^2\,\rho\; = n^2\,\sigma^2\,\rho + n(1-\rho)\sigma^2$$

and the variance of the average is obtained by dividing by n_2 which leads to the desired formula.
Given that $X_1 + \cdots + X_n$ is a collection of n random variables which are identically distributed but not necessary independent.

$$0 < \rho := \frac{E\left[(X_i - \mu)(X_j - \mu)\right]}{\sigma^2}$$
$$\Longrightarrow E(X_i, X_j) = \rho\sigma^2 + n^2$$

We find that,

$$var\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) = \frac{1}{n^2}\left[E\left[\left(\sum_{i=1}^{n} X_i\right)^2\right] - E\left[\left(\sum_{i=1}^{n} X_i\right)^2\right]\right] = \rho\sigma^2 + \frac{1-\rho}{n}\sigma^2$$

b) *Explain why Bootstrap Aggregating (Bagging) can alleviate the effect of overfitting with the help of the formula above.*

Bootstrap aggregating minimize the variance of predicted values by generating additional training data other than original data. These additional training data help in improving the predictive force of model by minimizing variance and narrowly training the prediction to expected outcome. Hence this reduction of variance helps in alleviating the over-fitting.

c) *Why can dropout be considered as an approximation to Bagging? |Explain in two or three sentences.*

Bagging and dropout do not achieve quite the same even though they both are types of model averaging.

However, dropout can be seen as an approximation to a bagging techniques.

Each iteration can be viewed as different model since we're dropping randomly different units on each layer. This means that the error would be the average of errors from all different models (iterations). Therefore, averaging errors from different models especially if those errors are uncorrelated would reduce the overall errors. In the worst case where errors are perfectly correlated, averaging among all models won't help at all; however, we know that in practice errors have some degree of uncorrelation. As result, it will always improve generalization error.

d) *Do we apply dropout during the inference? Justify your answer.*

Yes, there are cases when it might be useful. I.e. at inference time: in order to get a notion of uncertainty and variability in the prediction of the network model, we could take a given input and run predict on it many times, each with different randomly assigned dropout neurons.

Assume, we run predict 100 times for a single test input. The average of these will approximate what you get with no dropout, the 'expected value' over different weight schemes. And various metrics like the

Anna Krasilnikova 2562668
Mirza Misbah Mubeen Baig 2571567
Shahzain Mehboob 2571564

standard deviation of these results will give you a sense of the error bounds of your estimate (conditioned on assumptions about the validity of the underlying model structure). In this sense, it would be very useful to have to ability to re-activate Dropout settings from training, but specifically during testing or regular inference.

e) *In the lecture, we have been introduced to Dropout, one common way to implement that is inverted Dropout. Explain the idea behind inverted Dropout.*

With normal dropout at test time we have to scale activations by dropout rate since we are not dropping out any of the neurons, so we might need to match expected value at training.

With inverted dropout, scaling is applied at the training time, but inversely. It means, first dropout all activations by dropout factor p, secondly, scale them by inverse dropout factor 1/p.

Inverted dropout has an advantage: we don't have to do anything at the test time, which makes inference faster.

Exercise 9.2 – *Stochastic Gradient Descent*

In practice, we have 3 common variations for Gradient Descent Method, namely Stochastic Gradient Descent (SGD), Batch Gradient Descent and Mini-Batch Gradient Descent. Stochastic Gradient Descent takes only one data point to update in each step. Batch Gradient Descent takes the whole dataset for updating in each step. Mini-Batch Gradient Descent compromises the two methods above and takes a subset of the whole dataset for updating in each step.

a) Is it necessary to use learning rate decay for batch gradient descent based learning to converge? Please give your reason.

Not really, since we are using the whole dataset while computing the gradient by batch gradient descent. Essentially, this means that for each computation th full information of available data to define the optimal point is used, thus the gradient computation depends only on the set of initial weights.

b) Why is it important to use learning rate decay when doing stochastic gradient descent?

Learning rate decay is important, since it would allow us to use smaller steps in order to find the optimal weights, which essentially would increase the probability of converge on the late stage of training.
This is crutial, since in general, stochastic gradient descent uses only one sample, hence dependent on stochastic aspect of the samples. Thus neural network never converge because of the fixed learning rate, but can only descent to the minima.

c) What is the advantage of using a small batch size instead of the full set of examples in the training data?

Given the small stochastic sample, mini batches search for minima step-by step. On the other hand, if we are using full-set approach, there is only one possible stochastic property – weight initialization, since having same initial weights would lead the process to the same output.
Hence, small batch size can possibly find better solution.

Anna Krasilnikova 2562668
Mirza Misbah Mubeen Baig 2571567
Shahzain Mehboob 2571564

Exercise 9.3. – *Vanishing and Exploding Gradient*

In the lecture we've discussed some of the challenges in Neural Network Optimization. One among them is the so-called Exploding/Vanishing gradient problem. To be more specific, this problem happens only during the backward pass in training (very deep) Neural Networks.

a) Assume that you have a 100-layer Feed Forward Neural Network with sigmoid activation function as non-linearities. Explain why the exploding and vanishing gradient problem occurs only during the backward pass but not the forward pass with formulas.

Assuming that we have a NN of 100 non-linearity layers, the weight matrix for every layer would be W, which means in W^{100} in total.

Applying eigendecomposition, it is possible to get following on W^{100}:

$$W^{100} = diag(\lambda)^{100} V^{-1}$$

We know, that during backpropogation, gradients on the input layer are scaled by $diag(\lambda)^{100}$, which is also applicable to the exploding problem.

On the other hand, in forward pass we have limited function (sigmoid), thus there is no problem.

b) Explain how can we avoid the problem of gradient explosion based on what you learn in the lecture.

There are several approaches to fix exploding gradients. One of them – is to re-design the network model. This means that we might be able to redesign the network to have fewer layers. In recurrent neural networks updating across fewer prior time steps during training, called truncated Backpropagation through time, may reduce the exploding gradient problem.

Another way is to use gradient clipping: exploding gradients can still occur in very deep Multilayer Perceptron networks with a large batch size and LSTMs with very long input sequence lengths.

If exploding gradients are still occurring, you can check for and limit the size of gradients during the training of your network. It means, we can use gradient clipping if their norm exceeds a given threshold[1].

Another approach is called weight regularization. There we can check the size of network weights and apply a penalty to the networks loss function for large weight values. Using an L1(absolute weights) or an L2 (squared weights) penalty on the recurrent weights can help with exploding gradients[2].

[1] "Neural Network Methods in Natural Language Processing", 2017
[2] On the difficulty of training recurrent neural networks, 2013