Anna Krasilnikova 2562668
Mirza Misbah Mubeen Baig 2571567
Shahzain Mehboob 2571564

Exercise 10.1 – *Dropout*

Familiarize yourself with the SGD with momentum from the lecture slides (Chapter 8 slide 15) and DL book and understand how it works.

a) *It is known that the cost function of NNs usually has many saddle points (cf. DL book chapter 8.2.3). How does SGD with momentum help to alleviate the problem of getting stuck at these saddle points when compared to vanilla SGD (Chapter 8 slide 13). Describe a situation in which vanilla SGD will get stuck at one saddle point, while SGD with momentum won't?*

There exists a challenge in minimizing highly non-convex error functions common for neural networks, that is to avoid getting trapped in their numerous suboptimal minima. This issue might arise from saddle points (where one dimension slopes up and another slopes down). These saddle points are usually surrounded by a plateau of same error, which brings up the situation when SGD is not able to escape, since the gradient is too close to zero in all dimensions.
Overall, the issue of getting stuck while using vanilla SGD arises when we are training deep networks with complex error surfaces. Using SGD with momentum can help to overcome this problem. We use the momentum hyperparameter to determine what fraction of the previous velocity to retain in the new update and add this memory of past gradients to the current gradient.

Exercise 10.2 – *Parameter Initialization*

a) We normally initialize the parameters in our neural networks using a Gaussian distribution with mean zero and a small variance. Why shouldn't we initialize the parameters with a large variance?

Initialization of parameters has to be done so that we can mitigate the chances of exploding or vanishing gradients. This means, that the weights have to be set neither too much bigger than 1, nor too much less than 1. This would mean, that the gradients don't vanish or explode too quikly. They help to avoid slow convergence, also ensuring that we do not keep oscillating off the minima. Oerall, the main thing to take care about, thus, is to minimize the variance of the parameters.

b) Can we initialize the parameters with the same constant (suppose that the constant is small enough s.t. you won't have the problem in a))? Why/Why not?

We should not do that. For example there are at least two reasons not to set the initial weights to the constant close to the zero:
First, neural networks tend to get stuck in local minima, so it's a good idea to give them many different starting values. You can't do that if they all start at zero.
Second, if the neurons start with the same weights, then all the neurons will follow the same gradient, and will always end up doing the same thing as one another.

Exercise 10.3. – *Batch Normalization*

In previous assignments/projects, we normally dealt with data whose features have roughly the same scaling, e.g. in a grayscale image each feature (grayscale of a pixel) is between 0 and 255. Now, let's consider data whose features have very different scaling. For an extreme case: consider the situation in which each data point consists of two features: weight (x_1) and height(x_2). However, the unit for weight is kg, while the unit for height is mm (1 mm = 10^{-3} m).

Obviously the scales of these two features differ much. Consider a very simple loss function $L(w_1, w_2, b) = \left(y - (x_1 w_1 + x_2 w_2 + b)\right)^2$, the (2D) contour line will have a very elongated ellipse shape (the condition number of the Hessian is very large), which will harm the convergence speed of gradient descent algorithm..

Anna Krasilnikova 2562668
Mirza Misbah Mubeen Baig 2571567
Shahzain Mehboob 2571564

a) *To deal with the problem above, we can easily standardize the input data by subtracting the mean and dividing by the standard deviation. However, in the case that we use a deep neural network, it could still happen that the features in some hidden layers have very different scaling, then the same problem might occur again. Therefore, one can standardize the features in all hidden layers in each iteration. That is, standardize the features before passing it to the next layer. This idea is called batch normalization. Now, consider that in practice we train on mini-batches instead of the entire dataset, we cannot access the mean and sd of the whole dataset. What can we do in this case? Does your solution require a large batch size?*

If we don't have mean & standard deviation of the whole data, we can update our parameters by adding (minus sign) the gradient computed on a single instance of the dataset. Since it's based on one random data point, it's very noisy and may go off in a direction far from the batch gradient. However, the noisiness is exactly what you want in non-convex optimization, because it helps you escape from saddle points or local minima. it has a dis advantage that it is inefficient and we need to loop over the entire dataset many times to find a good solution but choosing a good minibatch size can influence the learning process indirectly, since a larger mini-batch will tend to have a smaller variance than a smaller mini-batch. The minibatch methodology injects enough noise to each gradient update, while achieving a relative speedy convergence.

b) *Actually, the idea of batch normalization introduced above is only half the story of the true batch normalization method. Let x denote the value of a particular neuron, let $\hat{x}$ denote the value of that neuron after standardization. We will transform $\hat{x}$ to $\gamma\hat{x} + \beta$ before passing it to the next layer, where γ and β here are learnable parameters. Why do we need these two additional parameters?*

γ and β are learned parameters on the level of each mini-batch. In particular, for a fixed mini-batch they can take any value. It seems like this makes shifting by the mean and scaling by the standard deviation pointless but these parameters are necessary because otherwise the normalized outputs would be mostly close to 0, which would hinder the network ability to fully utilize nonlinear transformations (consider an example of sigmoid function, which is close to identity transformation in 0 proximity).