Effect on the model by varying all the model parameters

Activation Function

When we replace the Relu Function with Sigmoid and Tanh, the model suffered from the vanishing gradient problem. During backpropagation, the gradients become very small due to successive multiplications of Jacobians, so the layers are trained slowly. The problem is compounded when using regularization.

The problem is of prediction of sales, the activation function best suited for this problem is ReLU as it also prevents gradients from saturating in the network, and thus mitigates the risk of vanishing gradients. And ReLU has much lower run time than Sigmoid and Tanh.

Loss Function

Here we used Mean squared error (MSE) which gives the average squared difference between the predicted value and the true value. As the problem is of predicting the sales, MSE is well suited here.

In this problem, we cannot use Binary Cross Entropy and Cross Entropy as both the entropies quantifies the difference between two probability distribution which is best suited for the classification problem.

Optimizer

When we use AdaGrad, it performed the larger updates for infrequent parameters and smaller updates for frequent parameters. It is well suited when we have sparse data as in large scale neural networks**.**

When we use Nadam, the learning process is accelerated by summing up the exponential decay of the moving averages for the previous and current gradient.

When we replaced Adam with AdaGrad, it penalizes the learning rate too harshly for parameters which are frequently updated and gives more learning rate to sparse parameters.

And also in Adagrad , there was a problem of vanishing gradient due to the learning rate being divided by the sum of the gradients.

And when we use NAG, it does nothing but it wants to take the momentum step first and calculate the gradient on the top of it. When we use Adadelta, it sums the exponentially weighted averages.

The best suited optimizer for the problem is Adam. It implements the exponential moving average of the gradients to scale the learning rate instead of a simple average as in Adagrad.

It keeps an exponentially decaying average of past gradients. Adam is computationally efficient and has very little memory requirement. By using Adam, we can also store eda of Momentum as well.

Epochs

We started with 20 to see if the model training shows decreasing loss and any improvement in accuracy. But, there is no minimal success with 20 epochs, and then keep on increasing the epoch and get some minimal success when epochs reach 100.

Batch Size

In this problem, larger batch sizes result in faster progress in training, but don't always converge as fast. Smaller batch sizes train slower, but converges faster.

The model improved with more epochs of training, to a point. It started to plateau in accuracy as they converge. I tried something like 50 and plot number of epochs vs accuracy. It levels out at around 100.

Number of neurons and layers

Input layer will need to have as many nodes as there are inputs. I have used correlation and dimensionality reduction techniques to first identify the relevant inputs.

If we have too few hidden units, we will get high training error and high generalization error due to underfitting and high statistical bias. If we have too many hidden units, we may get low training error but still have high generalization error due to overfitting and high variance.

To calculate the number of hidden nodes we use a general rule of:
(Number of inputs + outputs) * (2/3)"

We started by using a single layer at the starting and a few more nodes as compared to the input, but this does not give us high accuracy after some desired epochs then added more layers. The model gives high accuracy with two layers.

Dropout layers

We added Dropout after every layer, except the Input layer. A common value is a probability of 0.5 for retaining the output of each node in a hidden layer and a value close to 1.0, for retaining inputs from the visible layer.

So I set the Dropout rate to 0.5. Dropout rate > 0.5 is counter-productive. If a rate of 0.5 is regularizing too many nodes, then we can increase the size of the layer instead of reducing the Dropout rate to less than 0.5 to prevent the model from overfitting

I prefer to not set any Dropout on the Input layer. But if we required to do that, then we can set the Dropout rate < 0.2.