Deep Learning - Ex1 Report

Submitted by: Or Oxenberg, 312460132
Sahar Baribi, 311232730

Auxiliary functions:
In addition to the functions requested in the assignment, we added some functions to help us write a more comprehensive code.
We divided our code into three files:

1. ex1_functions - holds the required functions and some auxiliary functions that are listed below. All functions that are required to run the network are contained here.
2. ex1_run_network - Thie file holds the functions that are in charge of running the flow - loading the data, and running the 3 different flows required in the assignment (with and without batch normalization, and with dropout).
3. ex1_dropout - holds all the functions we had to change to support the dropout option. Further explanations on these changes are in the dropout section below.

The added functions are the followings:

1. create_batches - The function gets as input the data (X, Y) and the required batch size, and creates randomized batches. The output is two lists of batches - one for X and one for Y.
2. load_data - Reads the data from Keras dataset, transforms the y data into one hot encoded matrices, flattens the X matrices, and normalizes them. Returns the flattened and transformed X matrices and the one-hot encoded y matrices.
3. one_hot_transform - Transforms the data into one hot encoded matrix. The functions get the data it needs to transform as input and return a transposed one-hot encoded matrix.
4. plot_costs - Plots the train and validation costs.
5. run_NN - this function receives the data, and all the information required for the L_layer_model function. It holds a global variable that states whether the network should be run with or without batch normalization and runs the network
6. train_test_split - splits the data by the requested proportion. The input is the data we want to split and the output is

Dropout (Bonus)
The functions that weren't changed are imported from the ex1_functions file.
We added two global variables:

- Mode - a dictionary with a boolean value that states whether we are training or predicting at the moment.
- use_batch_norm - a boolean value that states if we are using batch normalization.
- dropout_rate - An array that holds the dropout rates for each layer. This is also a global variable.
- predict - A boolean value that states if we are training or predicting the model.

Moreover, we changed the following functions:

1. L_model_forward - In each layer we had a dropout mask that is multiplied in A when we are training the model. When predicting, we multiply the A in each layer with the

dropout rate. We also add the masks to the cache we hold so we will have the current mask of each batch for the backward propagation.

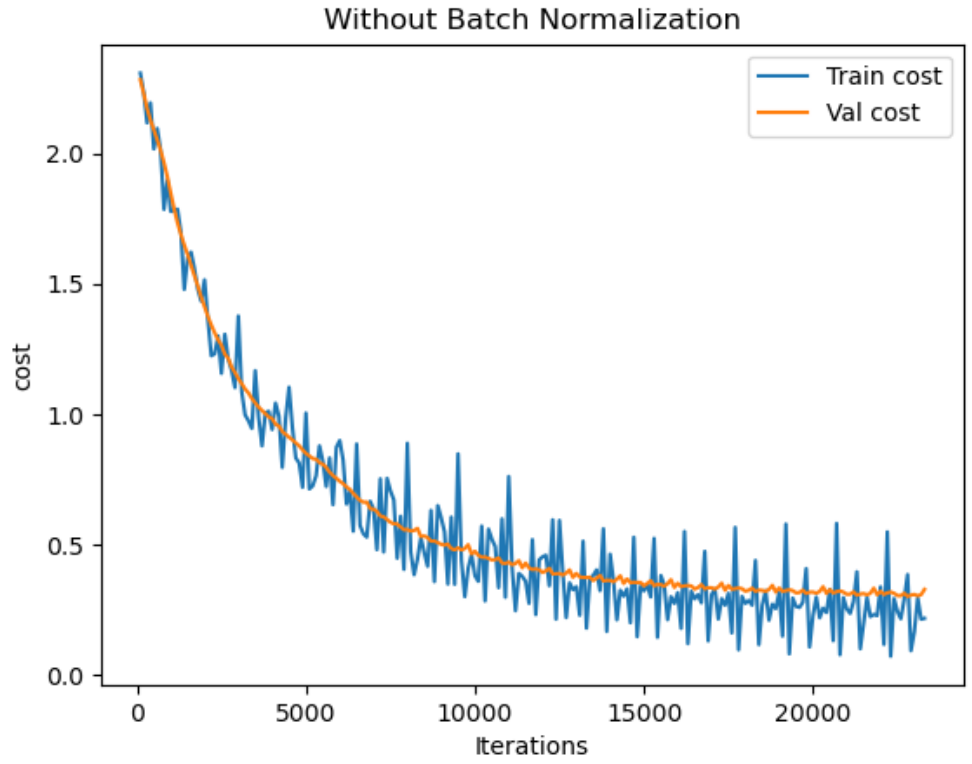
2. L_model_backward - We added the dropout masks for each layer with ReLU activation, and sent it to the linear_activation_backward function.
3. linear_activation_backward - We multiply dZ by the dropout mask of the current layer.
4. Predict - In the predict function we only change the predict flag. This is for the L_model_forward to know if it should multiply in the dropout mask or the dropout rate.

Summary Report

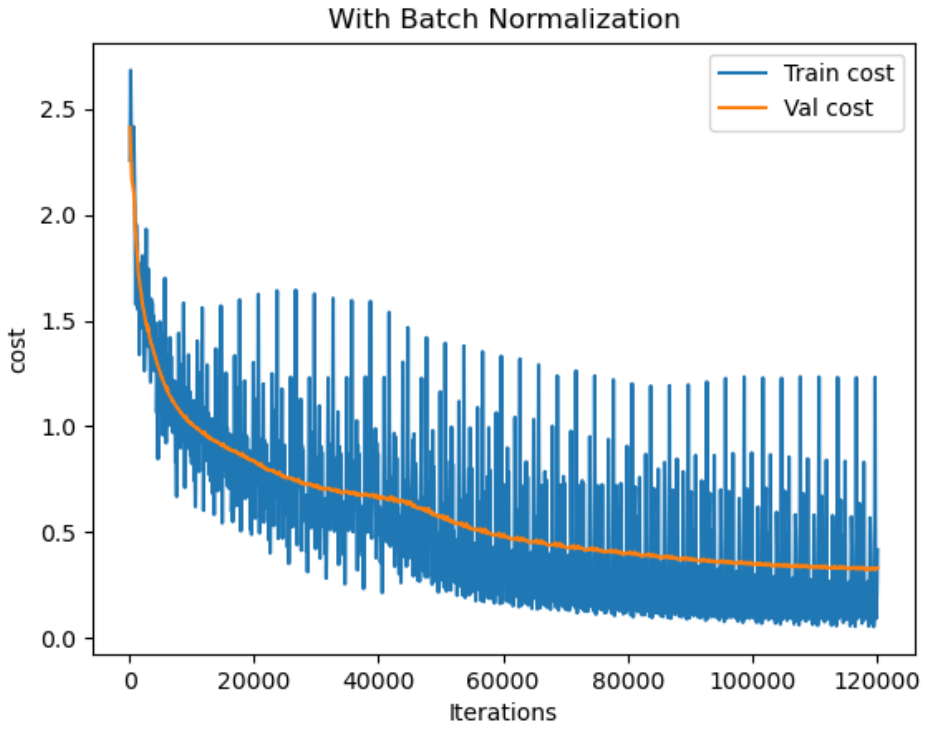
		Train	Validation	Test
Accuracy	With Batch Normalization	0.925	0.921	0.9174
	Without Batch Normalization	0.927	0.917	0.9224
	Dropout (without batchnorm)	0.9279	0.921	0.9216
Batch Size	With Batch Normalization	16		
	Without Batch Normalization	32		
	Dropout (without batchnorm)	32		
Epochs	With Batch Normalization	40		
	Without Batch Normalization	15		
	Dropout (without batchnorm)	12		
Iterations	With Batch Normalization	120100		
	Without Batch Normalization	23,300		
	Dropout (without batchnorm)	19,100		

Costs:

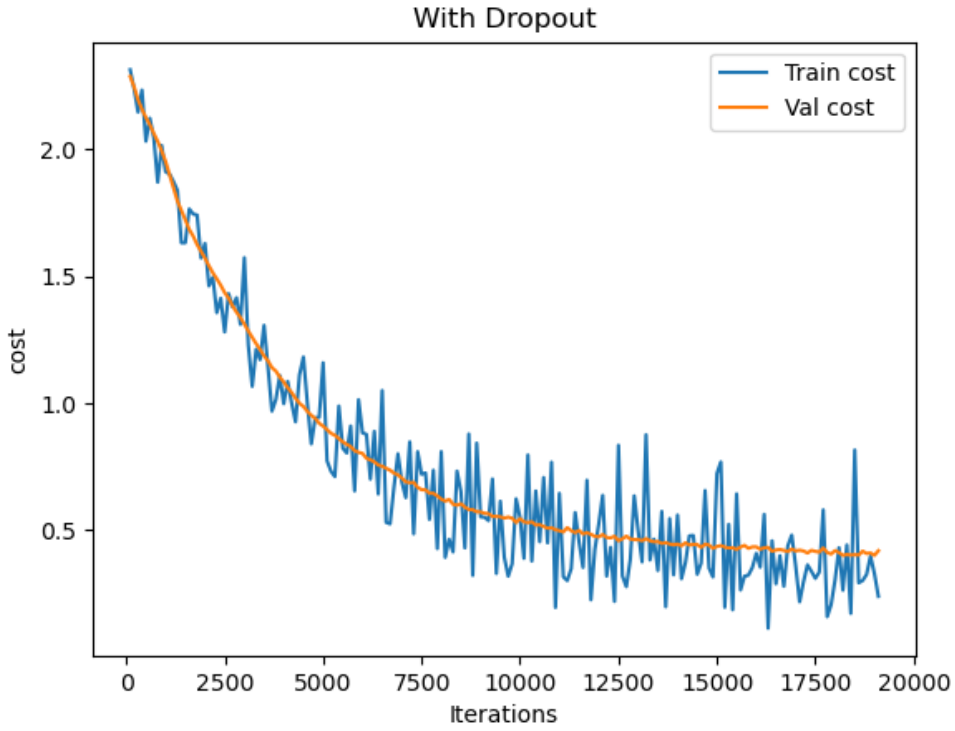
Without Batch Normalization



With Batch Normalization



With dropout



Analysis:

With vs Without BatchNormalization

When comparing the network with and without the batch normalization the main difference we see is in the number of epochs it takes the network to converge. Without batch normalization, we get a different distribution of values on the weights and the gradients are moving towards the minimization target, based on the current batch distribution.

When running the network with batch normalization, we adjust the distribution based on the mean and variance of the current batch. What makes the inputs of each layer and the gradients absolute value similar in each mini-batches. The graph shows that the changes in the train costs are volatile, and it takes the network a lot more epochs to converge in comparison to the network without batch normalization. We don't compare iterations since the batch size in the batch normalization network is smaller.

We will mention that the Batch normalization in this assignment is "crippled" - we don't use the gamma and beta parameters (as required in the assignment), which can impact the performance of the network.

With vs Without Dropout

Comparing the network with dropout to the network without batch normalization the performance is almost the same. For both networks, we use a batch size of 32, and the epochs required are almost the same (12 for dropout and 15 without batch normalization). We did notice that the time it takes the network without the dropout to converge is much faster. We expected the dropout to work faster than the network without the dropout, but we assume the time difference is due to our implementation of the dropout. We create a randomized mask for each layer and each batch and multiply it with the input to each layer. The behavior of the costs is different for each network. Without batch normalization, we see a frequent behavior since the network stays the same, and while applying dropout, the network changes all the time so we don't see the same behavior.