Name: Gustav Henning

Assignment 2

In assignment 2 we implemented a 2 layer neural network using relu activation and softmax for classification. The batch normalization layer is started but right now it only performs the task in the assignment (and doesn't learn). We will now proceed to answer the questions posed on page 7 of the assignment. The overfitting is bundled with ii)

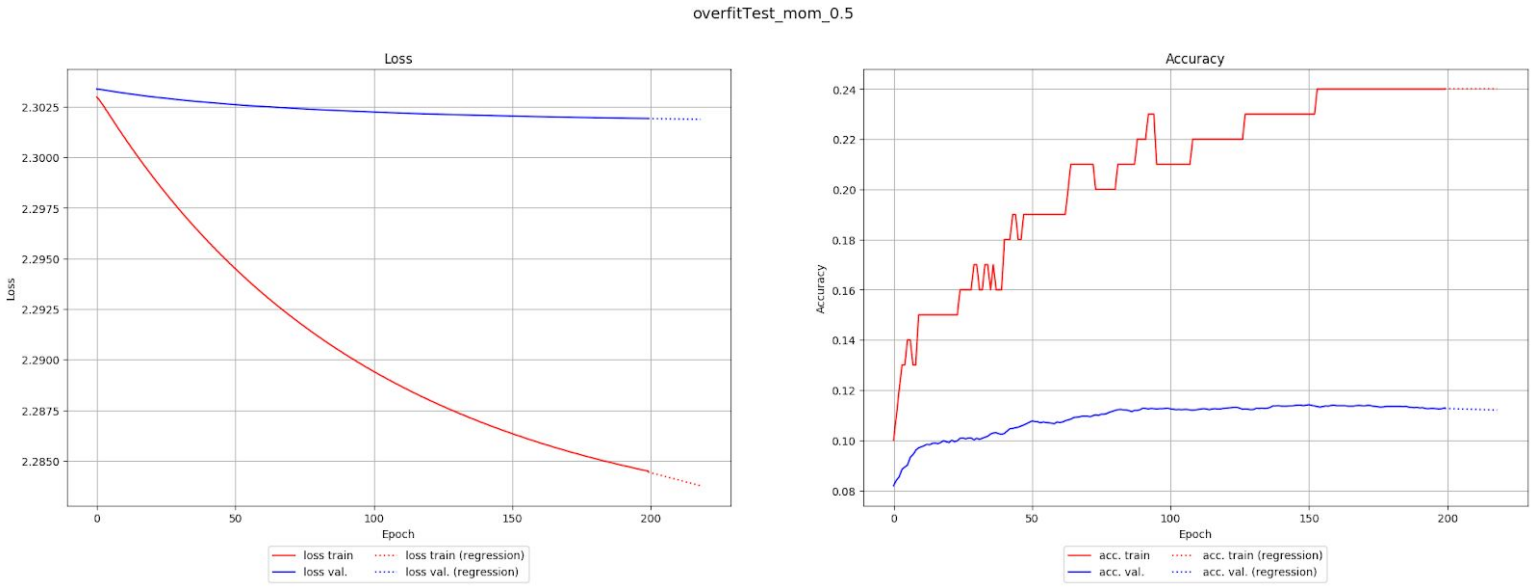
i) Analytic gradient computations.

I chose to implement the *numeric* gradient for comparison, very similar to the first assignment but with obvious modifications to fit the two layers, with the results:

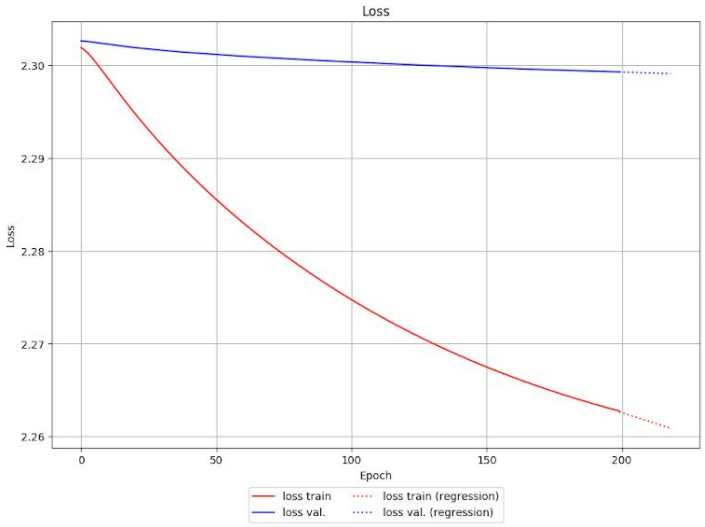
100%|████████████| 50/50 [00:00<00:00, 520.06it/s]
100%|████████████| 50/50 [05:50<00:00, 7.00s/it]
Gradient relative error check:
Relative error W: 1.532820e-02
Relative error B: 1.573210e-08
100%|████████████| 10/10 [00:00<00:00, 436.14it/s]
100%|████████████| 10/10 [00:01<00:00, 9.57it/s]
Gradient relative error check:
Relative error W: 5.072127e-05
Relative error B: 4.476419e-07

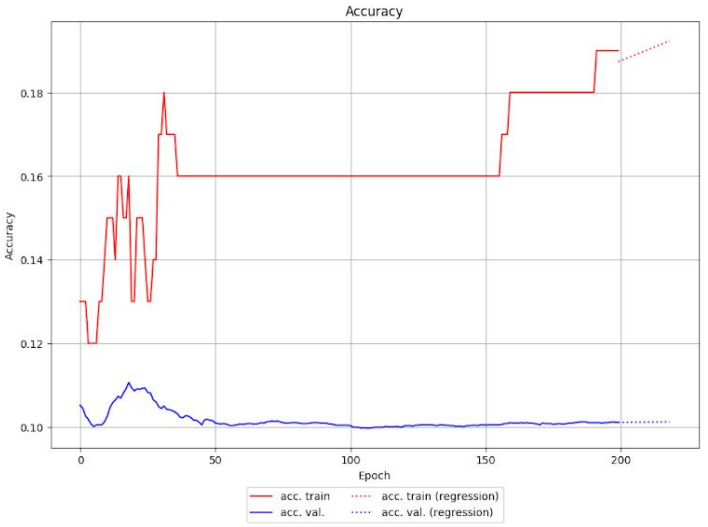
My evidence is that the benchmark for "good settings" for choosing the best network was at least 44%. My fine grained search yielded 44,54 %, which lead me to believe the gradient was correct. Also the relative error is within expected range, especially the second linear layer.

ii) Momentum comparison (0.5, 0.75, 0.8, 0.9) & **overfitting test**:

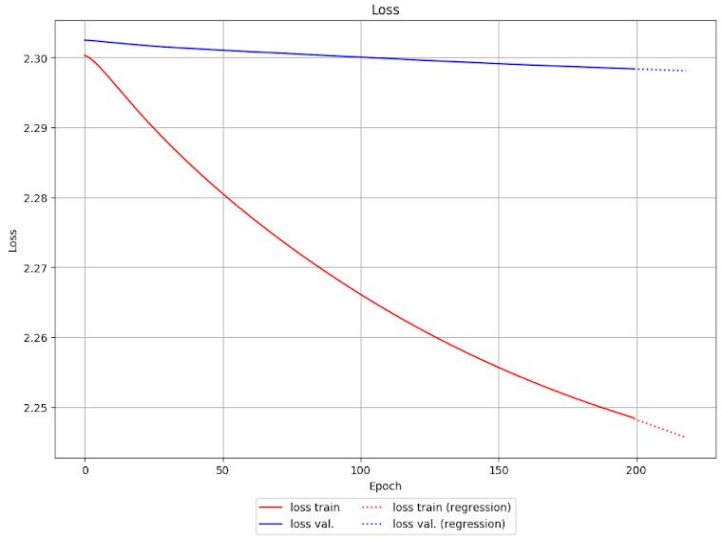


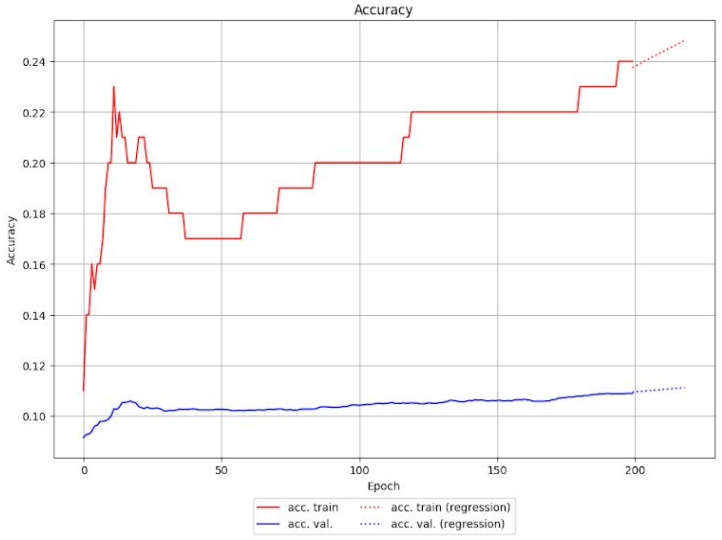
overfitTest_mom_0.75



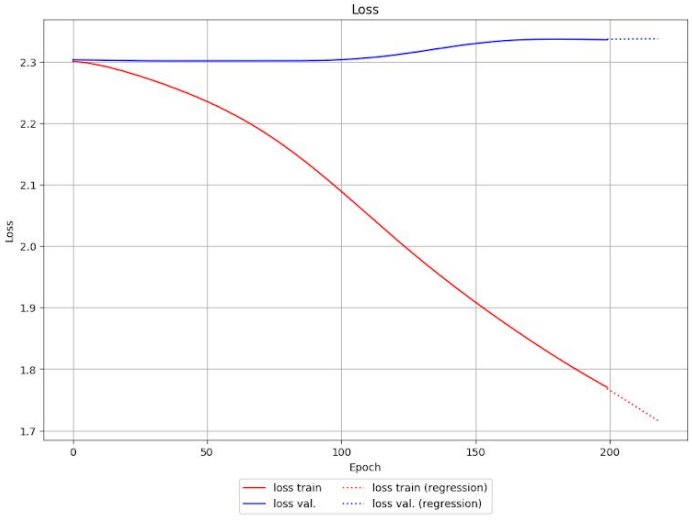


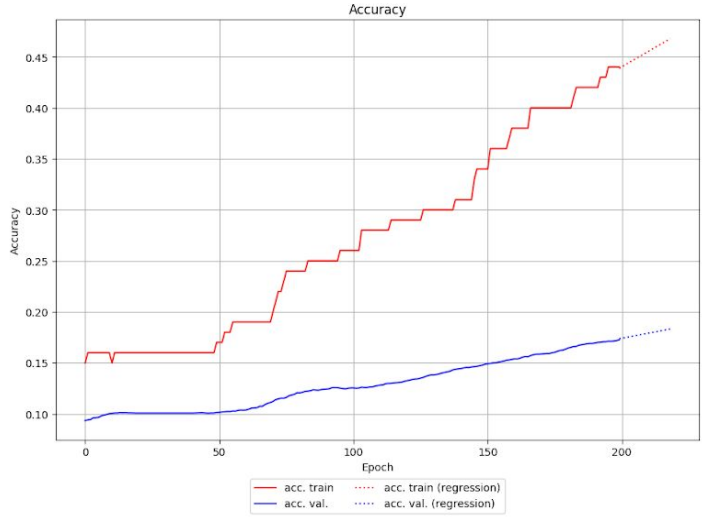
overfitTest_mom_0.8





overfitTest_mom_0.95





Clearly the momentum helps the loss dive but as we can clearly observe the validation accuracy acutely heads in another direction which is a sign of overfitting, sometimes even increasing!

iii) Ranges for coarse search:

reg_range = [0.001, 0.005, 0.01, 0.05, 0.1, 0.25, 0.5] (**lambda**)

l_rate_range = list(reversed(reg_range)) (**learning rate**)

	c_val	**decay rate**	**epochs**	\
2	[1.9824672858176633, 1.817986951157831, 1.7525...	0.995	5	
8	[1.9127212929317412, 1.8513098209679715, 1.839...	0.995	5	
9	[2.061155916833181, 1.900547904280982, 1.84082...	0.995	5	

	lam	last_a_train	last_a_val	last_c_train	last_c_val	**learning rate**	\
2	0.001	0.4713	0.4145	1.530527	1.701396	0.097525	
8	0.005	0.4837	0.4130	1.673354	1.839656	0.243812	
9	0.005	0.4577	0.4035	1.681367	1.808726	0.097525	

	momentum
2	0.8
8	0.8
9	0.8

iv) Ranges for fine search:

reg_range = np.linspace(0.005, 0.0005, num=7) (lambda)

l_rate_range = np.linspace(0.35, 0.1, num=4) (learning rate)

epochs: 10

Top 3 hyperparameters(including some data about the training):

	c_val	**decay rate**	**epochs**	\
27	[1.9810197976596047, 1.8101819529453542, 1.733...	0.995	10	
23	[2.000398733838501, 1.8449787839650191, 1.7696...	0.995	10	
22	[1.9024509468631228, 1.789164522534303, 1.7746...	0.995	10	

	lambda	last_a_train	last_a_val	last_c_train	last_c_val	\
27	0.00050	0.5743	0.4454	1.268968	1.647201	
23	0.00125	0.5610	0.4375	1.357466	1.704438	
22	0.00125	0.5895	0.4360	1.321052	1.782815	

	learning rate	**momentum**
27	0.095111	0.8
23	0.095111	0.8
22	0.174370	0.8

v) Interestingly enough, the "best" network (27 from fine searching) seems to perform worse on this extended batch. 43% on the test set as described by the output:

100%|██████████| 30/30 [09:17<00:00, 18.57s/it]
bestTest Test Accuracy: 0.43
Final train a/c, val a/c: 0.76/0.88, 0.43/2.06