

# Programming Assignment #1

## Task 0

This task was simply to just generate training and test data using Gaussian distribution and an identity matrix. Process is explained in the code.

## Task 1

This task was to build the fully connected layer model consisting of inputlayer, 2 fully connected layers to an output layer. Where ReLU was the only activation function. The weights to these layers was also initialized with Gaussian distribution.

## Task 2

In this task I just had to implement a simple forward pass using the test data where the input data was calculated with the weights and biases of each neuron in the layers. The accuracy was calculated by dividing the amount of correct samples with total samples multiplied by 100 to get the percentage.

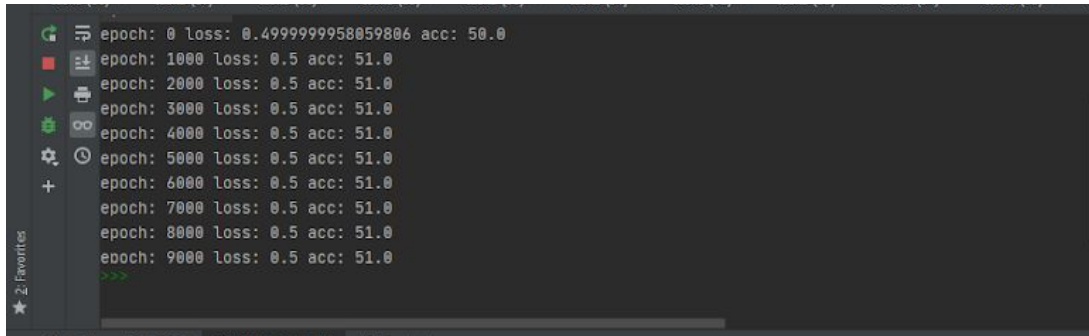
```
task3 (84) × task3 (85) × task3 (86) × task3 (87) × task3 (88) × task3 (89) × task3 (90) × task3 (91) × task3 (92) × task3 (93) ×
iteration: 1 acc: 51.0
iteration: 2 acc: 51.0
iteration: 3 acc: 51.0
iteration: 4 acc: 51.0
iteration: 5 acc: 51.0
iteration: 6 acc: 51.0
iteration: 7 acc: 51.0
iteration: 8 acc: 51.0
iteration: 9 acc: 51.0
iteration: 10 acc: 51.0
```

### Task 3

In this last task it got more complicated. This was the last step for a neural network to be working correctly, and I had to implement a backward pass. The backward pass was implemented, and weights were updated through the Stochastic Gradient Descent method, but there were some problems using the ReLU as both activations during the backwards propagation making almost all the new weights close to 0. Then using the MSE as the loss function and getting the mean from one hot encoded labels only returns the average which came out as 0,5 everytime. By using SGD I didn't see any progress in my NN, as the loss and accuracy stayed the same. So I implemented my own optimizer which generated random weights and just kept the best ones with lowest loss. And I saw progress in my NN. The code is also commented, so you would have a good overview of my thoughts. I also found out that a learning rate of 0.003 worked best for my NN.

There are not a lot of documentation of using MSE online with one-hot-encoded labels, and most people suggest using softmax, since it is easier to work with one-hot-encoding.

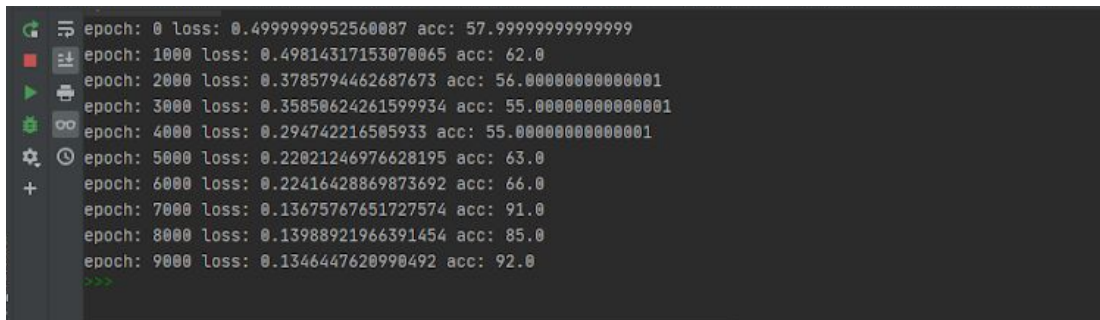
## With SGD

A terminal window showing the output of a training process using Stochastic Gradient Descent (SGD). The output displays metrics for epochs 0 through 9000. The loss remains constant at 0.4999999958059806, and the accuracy remains constant at 50.0. The terminal has a dark background with green and white text. On the left side, there is a sidebar with icons for file explorer, search, and other IDE features.

```
epoch: 0 loss: 0.4999999958059806 acc: 50.0
epoch: 1000 loss: 0.5 acc: 51.0
epoch: 2000 loss: 0.5 acc: 51.0
epoch: 3000 loss: 0.5 acc: 51.0
epoch: 4000 loss: 0.5 acc: 51.0
epoch: 5000 loss: 0.5 acc: 51.0
epoch: 6000 loss: 0.5 acc: 51.0
epoch: 7000 loss: 0.5 acc: 51.0
epoch: 8000 loss: 0.5 acc: 51.0
epoch: 9000 loss: 0.5 acc: 51.0
>>>
```

Accuracy and loss stays the same

## With my own made optimizer

A terminal window showing the output of a training process using a custom-made optimizer. The output displays metrics for epochs 0 through 9000. Both the loss and accuracy improve significantly over time. The loss decreases from 0.4999999952560087 to 0.1346447620990492, and the accuracy increases from 57.99999999999999 to 92.0. The terminal has a dark background with green and white text. On the left side, there is a sidebar with icons for file explorer, search, and other IDE features.

```
epoch: 0 loss: 0.4999999952560087 acc: 57.99999999999999
epoch: 1000 loss: 0.49814317153070065 acc: 62.0
epoch: 2000 loss: 0.3785794462687673 acc: 56.00000000000001
epoch: 3000 loss: 0.35850624261599934 acc: 55.00000000000001
epoch: 4000 loss: 0.294742216505933 acc: 55.00000000000001
epoch: 5000 loss: 0.22021246976628195 acc: 63.0
epoch: 6000 loss: 0.22416428869873692 acc: 66.0
epoch: 7000 loss: 0.13675767651727574 acc: 91.0
epoch: 8000 loss: 0.13988921966391454 acc: 85.0
epoch: 9000 loss: 0.1346447620990492 acc: 92.0
>>>
```

Both accuracy and loss gets better.