NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

CZ3005 Artificial Intelligence
FS5 Lab #2 Report

Prepared by:
Tey Chin Yi U1920268L

Task 1: Build a three layer feed-forward neural network to solve the monitoring problem of injection molding machines. Implementation is in Pytorch and executable in Google Colab environment. Proportion of training and testing samples is 70:30.

Final variables chosen:

Number of nodes of hidden layers	10
Number of mini batch size	10
Number of epoch	40
(Initial) Learning rate	0.05

Final testing/classification error achieved: 0% testing error

Flow of neural network:

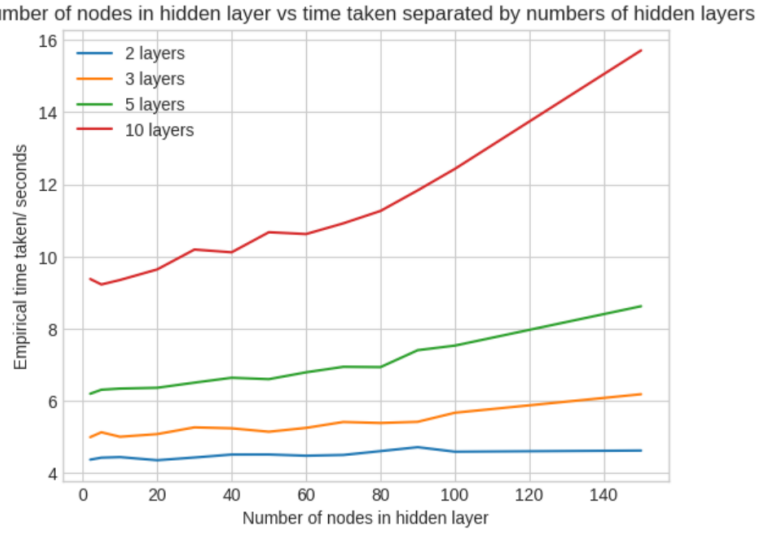
1) Create a network using Pytorch. The “connections” between the layers are defined using the nn.Linear() function. Forward propagation is done with the Rectified Linear Unit (ReLU) activation function.
2) The total cost of the network is evaluated by the cross entropy loss function, defined using nn.CrossEntrophyLoss() function. It is derived from comparison of output’s value and their expected/real value.
3) Backpropagation is then done through the Stochastic Gradient Descent (SGD) method, which makes use of derivatives of the loss function.

```
Training the network:


epoch= 0  time= 0.12800168991088867  loss= 1.0920673203468323  error= 51.75000020861626 percent lr= 0.05
test error  =  38.999999695354035 percent

epoch= 10  time= 1.1650619506835938  loss= 0.1324393234634772  error= 5.050000548362732 percent lr= 0.05
test error  =  1.8888892730077107 percent

epoch= 20  time= 2.1770613193511963  loss= 0.013436883895192295  error= 0.050000011920928955 percent lr= 0.03333333333333333
test error  =  0.11111113760206434 percent

epoch= 30  time= 3.200132369995117  loss= 0.0057667471082822885  error= 0.050000011920928955 percent lr= 0.022222222222222223
test error  =  0.0 percent

epoch= 40  time= 4.2072913646698  loss= 0.0032846526794673993  error= 0.0 percent lr= 0.014814814814814815
test error  =  0.0 percent
```

Task 2: Study the effect of network structure: hidden nodes, hidden layers to the classification performance. Try different network configurations and understand the patterns.

Two variables of concern here: number hidden nodes and hidden layers

An **empirical time analysis** is done on **4 different neural networks** with 2,3,5 and 10 layers respectively. For each neural network (NN), 2, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100,150 nodes in hidden layers are considered. Other variables like epoch, mini batch size and learning rate are kept constant.



An direct observation (from the graph above) is that the empirical time taken for training of NN increases as:

1. Number of nodes in hidden layer increases
2. Number of layers increases

For NN with more hidden layers, time taken increases exponentially with the increase of nodes in hidden layers.

	node	2	5	10	40	70	150
2 Layers test error (%)							
epoch	10	65.7	20.8	22.6	22	15.6	11
	20	38.5	3	1.9	2.4	1.6	2
	30	38.5	0.55	0	0	0	0.4
	40	38.5	0	0.2	0	0	0
	50	38.5	0	0	0	0	0
3 Layers test error (%)							
epoch	10	66.11	52.5	37.6	30.3	15.6	11.7
	20	65.77	4.33	1	4.2	0.3	0.1
	30	65.77	4.33	0	0	0	0
	40	65.77	4.33	0	0	0	0
	50	65.77	4.33	0	0	0	0

	node	2	5	10	40	70	150
5 Layers test error (%)							
epoch	10	72.7	61.4	61.4	65.8	65.8	45.1
	20	72.6	65.8	6.7	4.3	4.2	1.6
	30	72.5	65.8	4.3	0	0	0
	40	65.8	65.8	4.3	0	0	0
	50	65.8	65.8	4.4	0	0	0
10 Layers test error (%)							
epoch	10	65.8	65.8	72.6	65.8	62.4	65.8
	20	65.8	65.8	72.6	65.8	61.7	40
	30	65.8	65.8	72.6	65.8	61.1	41
	40	65.8	65.8	72.6	65.8	60.7	41
	50	65.8	65.8	72.6	65.8	60.7	41.7

Classification error analysis (table of data above) shows the type of classification error yielded for various network configurations.

1. **Deep network** (A lot of layers but not enough nodes in the hidden layers)
   (Example: 10 layers testing error; bottom right)
   a. As the number of layers increases, the classification error will increase if the number of nodes in the hidden layer does not increase proportionally.
   b. A quick research explained that adding more layers can help extract more features. But we can do that upto a certain extent. There is a limit. After that, instead of extracting features, we tend to 'overfit' the data. Overfitting can lead to errors in some or the other form like false positives.
2. **Shallow network** (Few layers with few nodes)
   (Example: 2 layers testing error when node = 2)
   a. When there are not enough hidden layers and nodes in the hidden layers, features cannot be abstracted properly. A sufficient number of nodes in the hidden layers are needed for the NN to learn the features of the inputs.
3. **Wide network** (A lot of nodes in the hidden layers)
   (Example: 2, 3 & 5 layers testing error when node = 40/70/150)
   a. Generally when the number of nodes is sufficient, an optimal testing error can be achieved (0% in the case study)

Conclusion: In our case study, **wide, shallow networks** are very good at **memorization**, but not so good at generalization. **Deep networks** help in learning features at various levels of abstraction and are much **better at generalising** because they learn all the immediate features between the raw inputs and high level classification. Hence, a sufficiently wide NN with a shallow network will be good enough.

<u>Task 3: Study the effect of learning rates. This includes possible adaptive learning rates where the value increases or decreases as the increase of epochs.</u>

Empirical time analysis (graph above) shows that time taken is almost constant within each NN. Reason being the training of models used through the same number of epoch and mini batch sizes and number of nodes in the hidden layers.

	LR	0.05	0.2	0.4	0.6	0.8	0.95
				Layer 3 test error (%)			
epoch	10	45.2	12.9	65.8	65.8	65.8	65.8
	20	1.4	65.8	65.7	65.8	65.8	65.8
	30	0	65.8	65.8	65.8	65.8	65.8
	40	0	65.8	65.8	65.8	65.8	65.8
	50	0	65.8	65.8	65.8	65.8	65.8

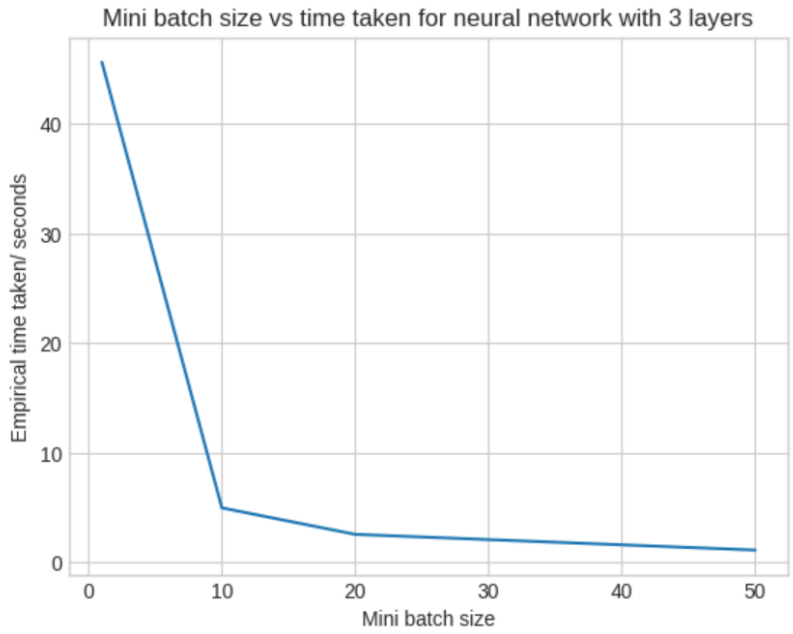
Looking at the **classification error data** (table above), a **learning rate that is too large** will not be suitable as the SGD method will not be able to locate the local minima during learning updates from back propagation process.

	LR	0.01	0.02	0.03	0.05	0.07	0.09
				Layer 3 test error (%)			
epoch	10	60.5	65.1	51.1	60.1	51.4	60.9
	20	12	9.3	5	0.4	2.6	2.6
	30	6.2	2.8	0.1	0	0	0
	40	4.8	0.2	0.1	0	0	0
	50	2.66	0.1	0.1	0	0	0

Hence we scope into **smaller learning rates < 0.05** to evaluate the effects of learning rate on classification error. It can be observed that learning rates 0.01, 0.02 and 0.03 are too slow to update the learnings and thus require additional epochs to train the NN, which can be unnecessary.

Conclusion: learning rate selection is important as too large a value will cause the model to miss the local minima during the learning process while too small a value will require more epochs, causing the training of NN to be more costly. **Hence, values greater than 0.04 and smaller than 1.0 can be used to get the optimal results efficiently,** considering possible adaptive learning rates where the value increases or decreases as the increase of epochs.

Task 4. Study the effect of mini-batch size. You can set mini-batch size to be 1 (stochastic gradient descent), N (batch gradient descent) or any other size.



Keeping all other variables like hidden layers, number of nodes in the hidden layers, learning rates and epoch constant, **empirical time** is recorded against the magnitude of mini-batch size.

Direct observation: The empirical time taken decreases drastically as the mini batch sizes increase.

	LR	1	10	20	50
		Layer 3 test error (%)			
epoch	10	12.7	71.9	50.4	65.1
	20	0	0.7	5.4	11.3
	30	0	0	2.2	7.8
	40	0	0	0	5.6
	50	0	0	0	5

Classification error analysis shows that as mini batch size increases, more epoch is needed to derive an optimal solution.
Conclusion: A mini batch size of 10/20 will be appropriate in this case study in order to reduce time taken as well as epoch needed to iterate through.

(End of lab report)