ECS759P: *Artificial Intelligence* Assignment 2, Question 2

- **Student name**: Pranav Narendra Gopalkrishna
- **Student number**: 231052045

Table of contents

Brief summary of the data & task

The data used contains 70000 images (60000 from the training set, 10000 from the testing set) of various clothing items along with labels associating each image to the kind of clothing item it depicts. Each image is greyscale with very low resolution. The task of our neural network will be to classify the images according to their labels.

Question 1: Loss function used

The task we want the convolutional neural network (CNN) to perform is multi-class classification. In practice, we do such multi-class classification by obtaining a probability value for each class (using the softmax function) that indicates the probability that the given observation belongs to the given class. For obtaining the loss function in such a problem, the loss function must:

- Deal with the accuracy of the estimation of probability distributions
- Give substantial gradients despite the small range of the predictions (which range from 0 to 1)
- Consider only the predictions for the observed classes

NOTE: *Loss function is relevant only in training, so note that the following is discussed in the context of training the CNN.*

The **cross-entropy loss** function is one that satisfies the above conditions, and thus, is used widely in multi-class classification problems. Its formula per observation, i.e. per prediction, is as follows:
$H(P, Q) = -\sum_{x \in X} P(x) \log(Q(x))$

Here, we have:

- X: The random variable denoting the label (class)
- x: A possible label (class)
- P: Probability distribution 1

- Q: Probability distribution 2
- $P(x)$: Probability of $X = x$ under P
- $Q(x)$: Probability of $X = x$ under Q
- $H(P, Q)$: The cross-entropy between P and Q
- Logarithm is generally base 2 or base e

In our case in particular, we interpret P and Q as:

- $P(x)$: The observed probability of the label being x
- $Q(x)$: The estimated (softmax) probability of the label being x

In our case, since we know which class the given observation belongs to, $P(x)$ is simply 1 for a particular x and 0 for every other x.

Another advantage of cross entropy:
Since it uses the negative logarithm of the softmax value (i.e. probability), lower valued points will have a steeper gradient, leading to greater updates of weights when predictions are poorer.

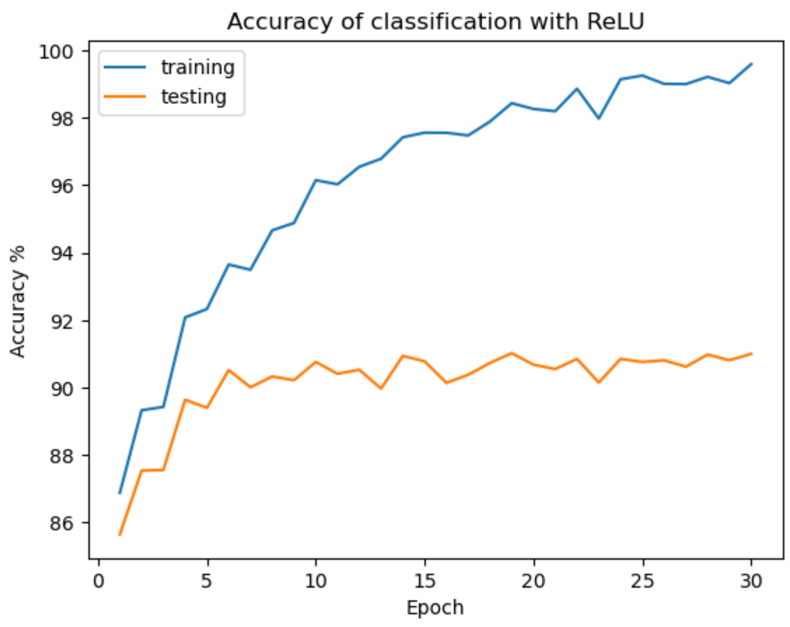
> REFERENCES:
>
> - https://analyticsindiamag.com/a-beginners-guide-to-cross-entropy-in-machine-learning/
> - https://365datascience.com/tutorials/machine-learning-tutorials/cross-entropy-loss/

Question 2: Creating the CNN

Loss & accuracy of the model

- Final train cost: 47.6082
- Final train accuracy: 99.583
- Final test accuracy: 91.0

Plotting the accuracy of classification over epochs...

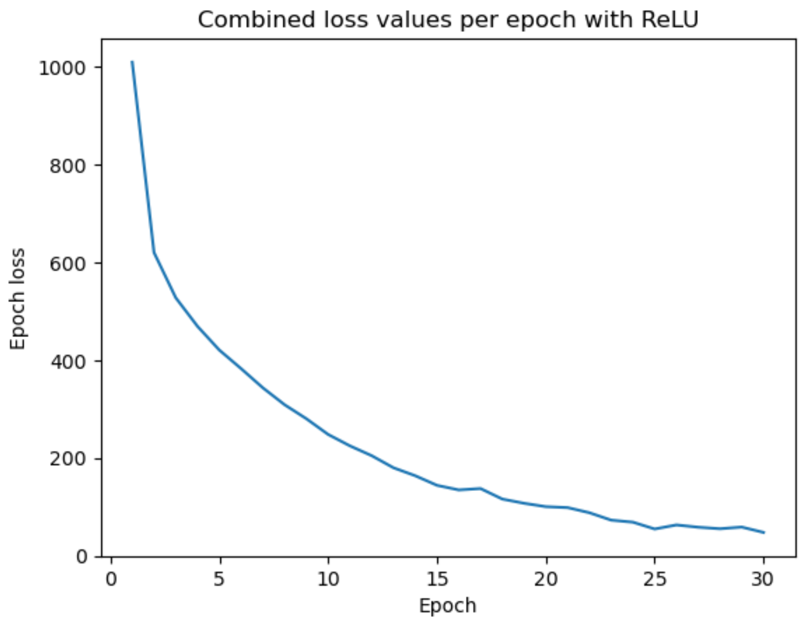


COMMENTS:

Note that the performance, i.e. accuracy over epochs has random spikes at certain points, but we shall ignore these and focus on the overall trend. We observe that for the speed of performance change (i.e. rate of change in accuracy) rises rapidly for the initial few epochs; the rate of this change reduces for the later epochs. More specifically, we observe that the training accuracy keeps rising (overall) until epoch 30, but its rate of increase decreases gradually. On the other hand, the testing accuracy rises rapidly only until around epoch 6 or 7, after which it stays roughly constant until epoch 30.

The substantial divergence (around 8.583% difference) between training and testing accuracy indicates overfitting, wherein the model fine-tunes itself with respect to the training data; in such a case, the general accuracy of the model is not improved and may even decrease.

Plotting the loss values per epoch...

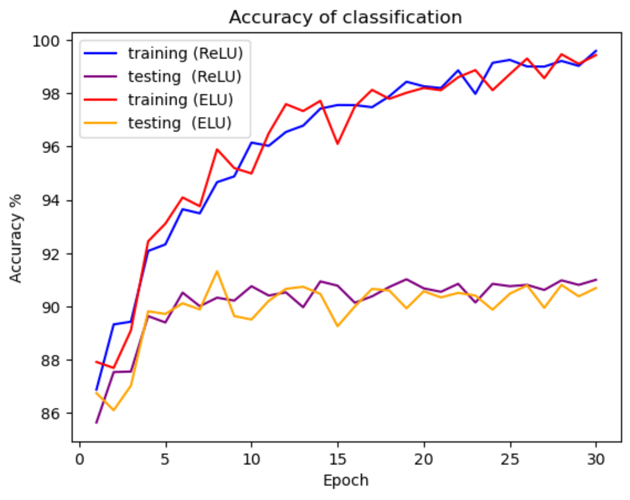
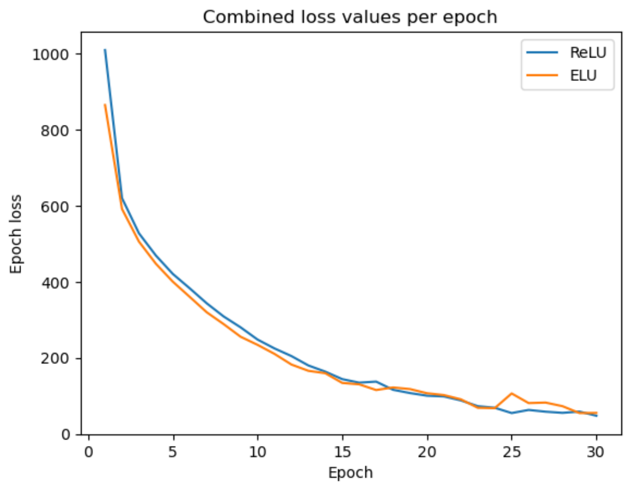


COMMENTS:
The above curve of loss values is for training loss only. We observe that for the of loss values per epoch falls rapidly for the initial few epochs; the rate of this change reduces for the later epochs. This coincides with the way the previously discussed training accuracy changes over epochs.

Question 3: Comparing with ELU activation function

	Activation	Final train accuracy	Final test accuracy
Model M1	ELU	99.583333	91.00
Model M2	ReLU	99.426667	90.69

Some extra visualisation...

Accuracy	Per-epoch loss
	

COMMENTS:
Comparing the activation functions "ReLU" (rectified linear unit) and "ELU" (exponential linear unit), we observe only a negligible difference in both the final training and testing accuracies, indicating that these activation functions are similar in performance for multi-class classification. The formulations of these activation functions are as follows:

ReLU:
$$f(x > 0) = x, f(x \leq 0) = 0$$

ELU:
$$g(x > 0) = x, g(x \leq 0) = \alpha e^x - 1$$

Here, α is a constant. Also note that x in both the above cases denotes the input to the neuron, which is in fact the weighted sum of some subset of outputs from the previous layer.

ReLU and ELU are similar, with the key difference that ELU produces negative outputs for negative inputs while having a positive gradient. It is a strong alternative to ReLU, since one of ReLU's key limitations is the case where large weight updates make inputs to certain neurons always negative, regardless of the initial input to the network. In such a case, these neurons will stop responding to variations in error or input because the gradient for this neuron becomes zero; this is called the dying ReLU problem.

In our case, the similar performance of ReLU and ELU suggests that the issue of dying ReLU does not arise, at least for the learning rate used (0.1).

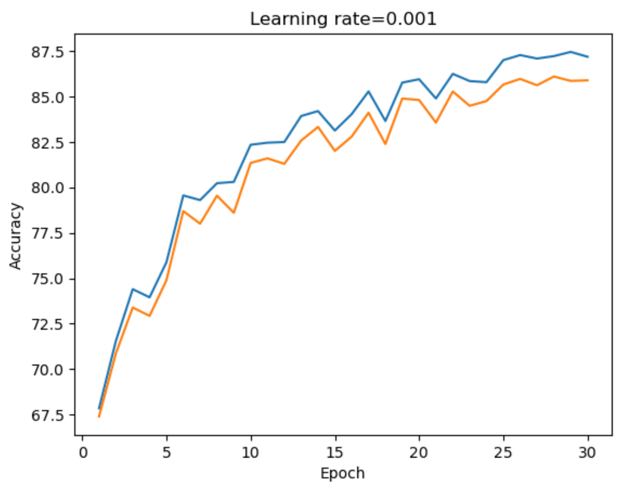
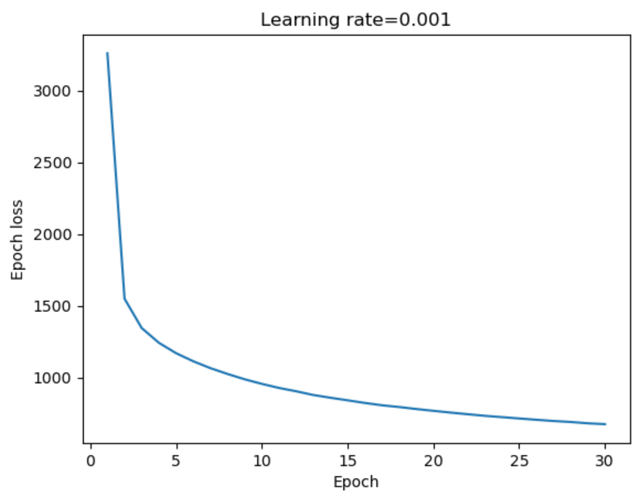
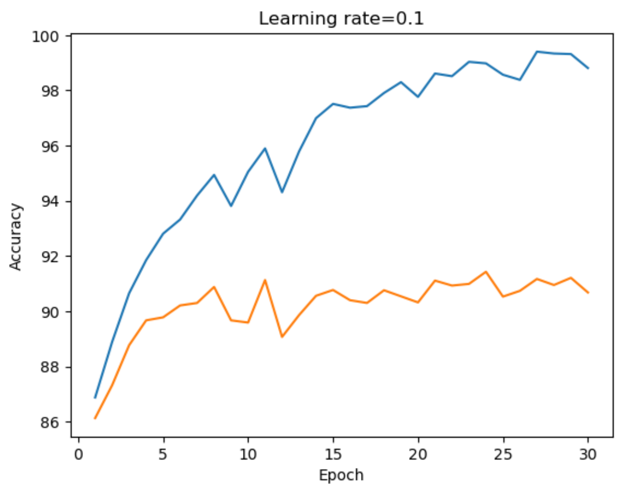
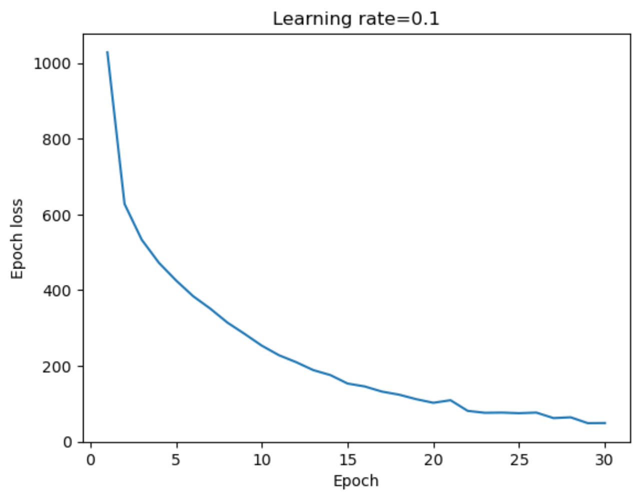
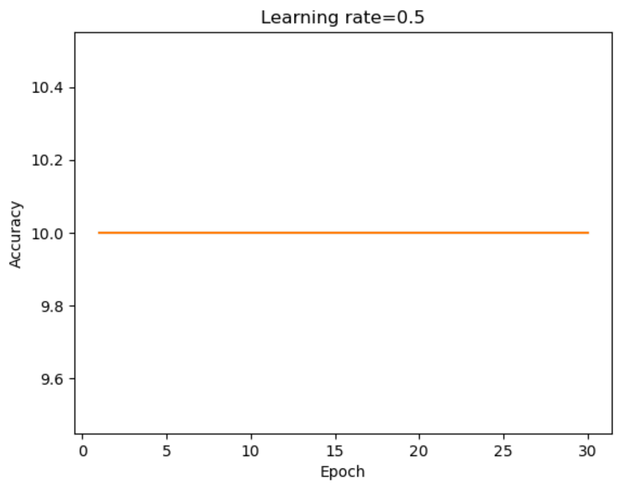
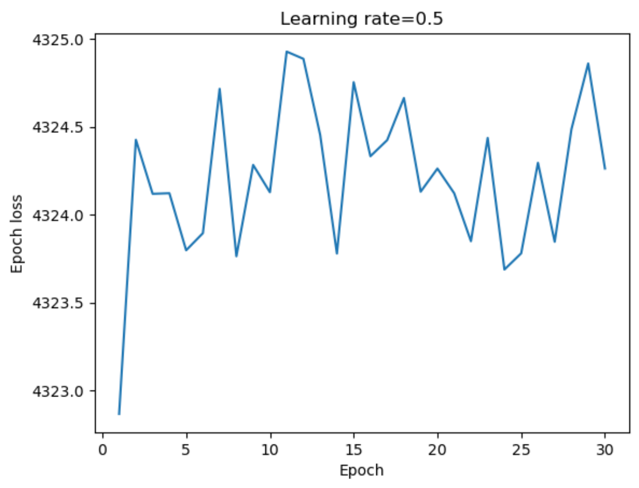
> REFERENCES:
>
> - https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html
> - https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

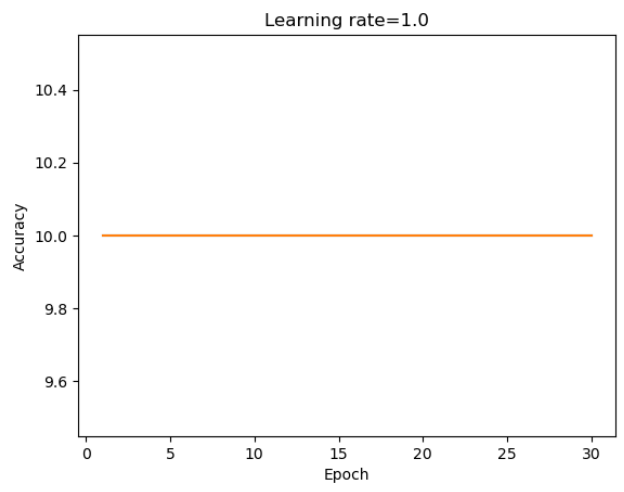
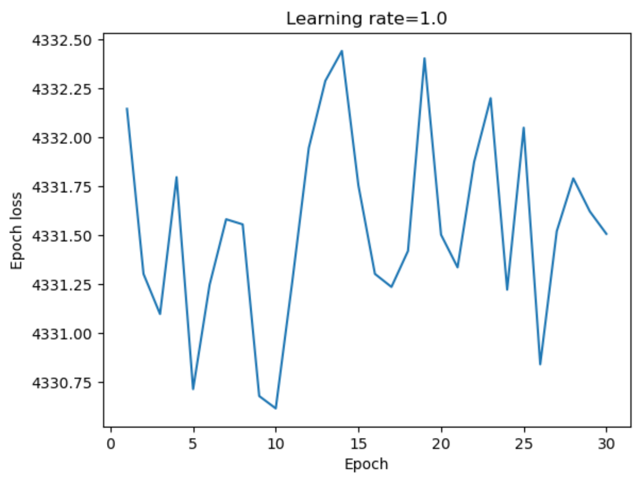
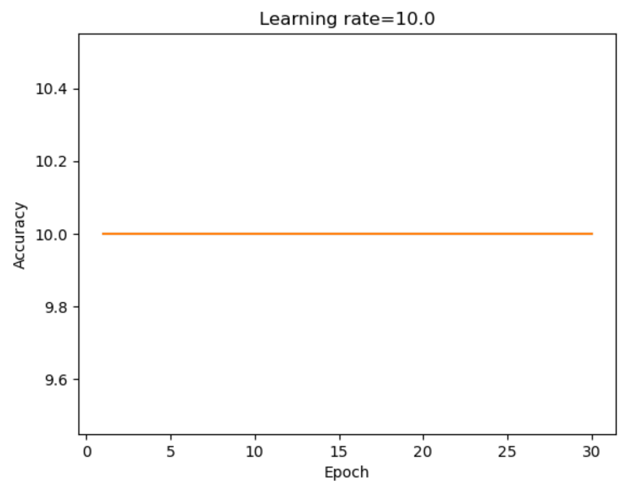
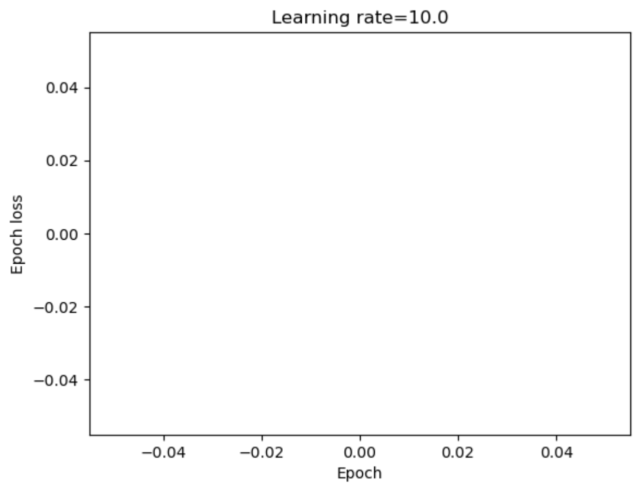
Question 4: Comparing different learning rates

	Learning rate	Final cost	Final train accuracy	Final test accuracy
0	0.001	673.850306	87.176667	85.88

	Learning rate	Final cost	Final train accuracy	Final test accuracy
1	0.100	49.237395	98.810000	90.68
2	0.500	4324.262487	10.000000	10.00
3	1.000	4331.506782	10.000000	10.00
4	10.000	NaN	10.000000	10.00

Observing speed and stability of convergence in accuracy...

Accuracy	**Per-epoch loss**
	
	
	

Accuracy	**Per-epoch loss**
	
	

COMMENTS:

We shall use the abbreviation LR for "learning rate."

We observe the following with respect to training and testing accuracy:

- Highest for LR 0.1 (above 95% for training & above 90% for testing)
- High for LR 0.001 (around 85% for both training & testing)
- Very low (10%) for all attempted learning rates above 0.1

We observe the following with respect to the final per-epoch loss:

- Lowest for LR 0.1
- Significantly higher (by one order of magnitude) for LR 0.001
- Much higher (by two orders of magnitude) for learning rates 0.5 & 1.0
- `NaN` for LR 10.0

The above indicates that the training cost (per-epoch loss value) converges for both LR's 0.01 and 0.1, but converges significantly faster for learning rate 0.1. The `NaN` (not-a-number) value for learning rate 10.0 can be interpreted as a cost so high that it exceeds our computational limits. This is due to the massive divergence or explosion in the per-epoch loss values, caused by the large LR used.

From the graphs for the per-epoch loss values for each learning rate, we observe the following:

- LR's 0.001 & 0.1 have a stable convergence
- LR's 0.5 & 1.0 have unstable changes in the loss values & accuracy with no overall improvement

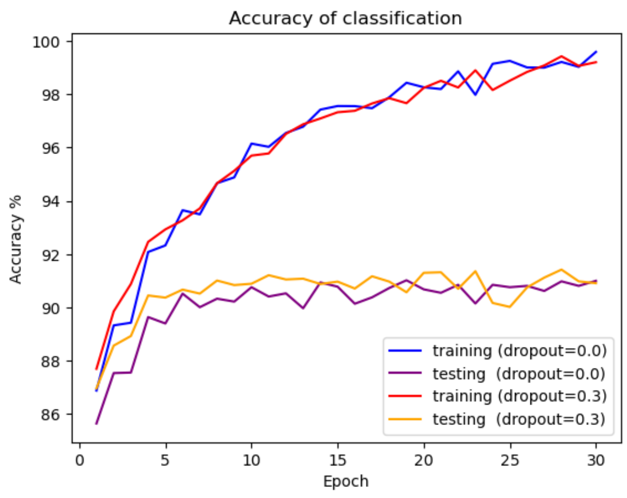
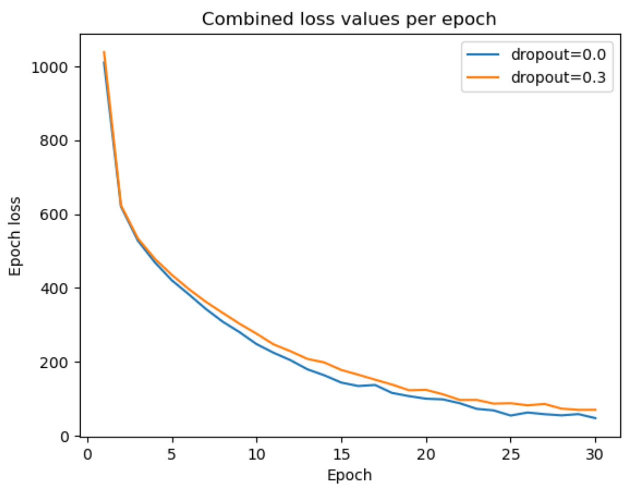
- LR 10.0 has an empty plot due to `NaN` value

Trade-off between speed and stability...
Only LR's 0.001 & 0.1 show a stable convergence in loss values, translating to a stable improvement in performance over multiple epochs. However, from the plots of the per-epoch loss values, we observe that the graph for LR 0.001 has less random fluctuations than the LR 0.1, indicating that the convergence with a lower LR leads to greater stability in performance change. This highlights the speed vs. stability trade-off; up to a point, higher LR's may show faster convergence of cost overall but with greater amounts of random fluctuations in the cost, whereas lower LR's show slower convergence but a more stable rate of change over epochs. In line with these observations, we see in the accuracy plots that LR 0.1 shows a greater overall increase in both training and testing accuracy, but we also see larger random fluctuations.

Question 5: Testing dropout

	Dropout	Final cost	Final train accuracy	Final test accuracy
Model M1	0.0	47.608175	99.583333	91.00
Model M3	0.3	70.234269	99.196667	90.91

Some extra visualisation...

Accuracy	**Per-epoch loss**
	

COMMENTS:
The training and testing accuracy show negligible difference when comparing the model that uses dropout with rate 0.3 to the model that does not use dropout (i.e. that uses dropout with rate 0.0), indicating that the performance was not greatly affected (positively or negatively) by using dropout with rate 0.3. The graphs of the accuracy and per-epoch loss value does not indicate a significant difference in performance either. The difference in the final costs do not indicate a clear difference in performance, especially since the difference is proportionally quite small (proportional to the maximum and average cost).

The relevance of dropout...
Dropout is a computationally cheap method of avoiding overfitting. Dropout forcibly deactivates certain neurons from the selected hidden layer, which reduces the rate at which the neurons may fine-tune their weights to the noise present in the training data.

> REFERENCE: https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

In our case...
In our case, we observe that overfitting is not reduced, i.e. training accuracy still significantly exceeds and increases at a greater rate than testing accuracy, indicating that using dropout with rate 0.3 shows no clear reduction in overfitting.