ECS759P Artificial Intelligence: Coursework 2

Crystal clear

Note: I've referred to Lab8 session on the FOL solutions to formulate and arrive at my solutions for the question below.

2.1 First order Logic.

1. You have a dog.
   - ∃x Dog(x) ∧ Own (YOU, x)

2. The person you are looking for buys carrots by the bushel.
   - BuyCarrots (ROBIN)

3. Anyone who owns a rabbit hate anything that chases any rabbit.
   - ∀x (∃y(Own(x,y)∧Rabbit(y)) → (∀z (∃w (Rabbit(w)∧Chase(z,w)) → Hate(x,z))))

4. Every dog chases some rabbit.
   - ∀x Dog(x) → ∃y (Rabbit(y) ∧ Chase(x,y))

5. Anyone who buys carrots by the bushel owns either a rabbit or a grocery store.
   - ∀x (BuyCarrots(x) → (∃y (Own (x, y) ∧ Rabbit(y) ∨ GroceryStore(y))

6. Someone who hates something owned by another person will not date that person.
   - ∀x ∀y (∃z (Owns (y, z) ∧ Hates (x, z)) → ¬Date (x, y))

2.2 Conjunctive Normal Forms

a) Remove implications.
   7. ∀x(¬∃y(Own(x,y) ∧ Rabbit(y))) ∨ (∀z(¬∃w(Rabbit(w) ∧ Chase(z,w)) ∨ Hate(x,z))) [from 3.]

   8. ∀x ¬Dog(x)∨∃y(Rabbit(y) ∧ Chase(x,y)) [from 4.]

   9. ∀x ¬(BuyCarrots(x) ∨ (∃y (Own (x, y) ∧ Rabbit(y) ∨ GroceryStore(y))) [from 5.]

   10. ∀x ∀y (¬∃z(Own(y,z) ∧ Hate(x,z))) ∨ ¬Date(x,y)) [from 6.]

b) Minimise negations.
   11. ∀x∀y(¬Own(x,y) ∨ ¬ Rabbit(y)) ∨ (∀z∀w (¬Rabbit(w) ∨ ¬Chase(z,w)) ∨ Hate(x,z)) [from 7.]

   12. ∀x ∀y ∃z(¬Own(y,z) ∨ ¬Hate(x,z)) ∨ ¬Date(x,y)) [from 10.]

c) Standardise variables.
   13. ∀x1∀y1(¬Own (x1, y1) ∨ ¬ Rabbit(y1)) ∨ (∀z1∀w1 (¬Rabbit(w1) ∨ ¬Chase (z1, w1)) ∨ Hate(x1, z1)) [from 11.]

   14. ∀x2 ¬Dog(x2) ∨ ∃y2(Rabbit(y2) ∧ Chase (x2, y2)) [from 8.]

15. ∀x3 ¬ (BuyCarrots(x3) ∨ (∃y3 (Own (x3, y3) ∧ Rabbit(y3) ∨ GroceryStore(y3)))
[from 8.]

16. ∀x4 ∀y4 ∃z2(¬Own (y4, z2) ∨ ¬Hate (x4, z2)) ∨ ¬Date(x4,y4)) [from 12.]

d) Skolemise existentials.
17. Dog(D) ∧ Own (YOU, D) [from 1.]

18. ∀x2 ¬Dog(x2) ∨ (Rabbit(A(x2)) ∧ Chase (x2, A(x2))) [from 14.]
Note: the existential on y2 was in the scope of the universal on x, hence the need for introduction of the function A(x2).

19. ∀x3 ¬ (BuyCarrots(x3) ∨ (Own (x3, B(x3)) ∧ Rabbit(B(x3)) ∨ GroceryStore(y3))
[from 8.]
Note: the existential on y3 was in the scope of the universal on x, hence the need for introduction of the function B(x3).

e) Drop universals.
20. □ ¬ Own (x1, y1) ∨ ¬ Rabbit(y1) ∨ □ ¬ Rabbit(w1) ∨ ¬ Chase (z1, w1) ∨ Hate (x1, z1))
[from 13.]
21. □ ¬Own (y4, z2) ∨ ¬Hate (x4, z2) ∨ ¬Date (x4, y4)) [from 16.]

22. □ ¬Dog(x2) ∨ (Rabbit(A(x2)) ∧ Chase (x2, A(x2))) [from 18.]

23. □ ¬ BuyCarrots (x3) ∨ (Own (x3, B(x3)) ∧ (Rabbit(B(x3) ∨ GroceryStore(B(x3)))
[from 19.]

f) Convert to CNF.
24. (¬Dog(x2) ∨ Rabbit(A(x2))) ∧ (¬Dog(x2) ∨ Chase (x2, A(x2))) [from 22.]

25. (¬BuyCarrots(x3) ∨ (Own (x3, B(x3)) ∧ (¬BuyCarrots(x3) ∨ Rabbit(B(x3)) ∨ GroceryStore(B(x3)) [from 23.]

g) Final set
26. Dog(D) ∧ Own (You,D)
27. BuyCarrots(ROBIN)
28. ¬ Own (x1, y1) ∨ ¬ Rabbit(y1) ∨ ¬ Rabbit(w1) ∨ ¬ Chase (z1, w1) ∨ Hate (x1, z1)
29. (¬Dog(x2) ∨ Rabbit(A(x2))) ∧ (¬Dog(x2) ∨ Chase (x2, A(x2)))
30. (¬BuyCarrots(x3) ∨ (Own (x3, B(x3)) ∧ (¬BuyCarrots(x3) ∨ Rabbit(B(x3)) ∨ GroceryStore(B(x3))
31. ¬Own(y4, z2) ∨ ¬Hate(x4,z2) ∨ ¬Date(x4,y4)

2.3 Goal state

• If the person you are looking for does not own a grocery store, she will not date you.

- ¬(∃y(Own(ROBIN, y) ∧ GroceryStore(y))→ ¬Date(ROBIN,YOU)
- Negation:
  - ¬(¬(∃y(Own(ROBIN, y) ∧ GroceryStore(y))→ ¬Date(ROBIN,YOU))
  - ¬(¬(∃y(Own(ROBIN, y) ∧ GroceryStore(y))→ ¬Date(ROBIN,YOU)
  - ¬(¬(¬(∃y(Own(ROBIN, y) ∧ GroceryStore(y))) ∨ ¬Date(ROBIN,YOU)
  - ¬(∃y(Own(ROBIN, y) ∧ GroceryStore(y)) ∨ ¬Date(ROBIN,YOU))
  - ∃y(¬Own(ROBIN, y) ∨ ¬GroceryStore(y)) ∧ Date(ROBIN,YOU)

    - ¬Own(ROBIN, y) ∨ ¬GroceryStore(y) ∧ Date(ROBIN,YOU)
- Final set
    - ¬Own(ROBIN, y) ∨ ¬GroceryStore(y) ∧ Date(ROBIN,YOU)

2.3 Resolution Proof

Final set

1. Dog(D)
2. Own (YOU, D)
3. BuyCarrots(ROBIN)
4. ¬ Own (x1, y1) ∨ ¬ Rabbit(y1) ∨ ¬ Rabbit(w1) ∨ ¬ Chase (z1, w1) ∨ Hate (x1, z1)
5. ¬Dog(x2) ∨ Rabbit(A(x2))
6. ¬Dog(x2) ∨ Chase (x2, A(x2))
7. ¬BuyCarrots(x3) ∨ Own (x3, B(x3))
8. ¬BuyCarrots(x3) ∨ Rabbit(B(x3) ∨ GroceryStore(B(x3))
9. ¬Own(y4, z2) ∨ ¬Hate(x4,z2) ∨ ¬Date(x4,y4)
10. ¬Own(ROBIN, y) ∨ ¬GroceryStore(y)
11. Date(ROBIN,YOU)

Resolution

12. ¬Own(YOU, z2) ∨ ¬Hate(ROBIN,z2) ~~∨ ¬Date(ROBIN, YOU)~~ {ROBIN/x4, YOU/y4} [from. 9, 11]
13. ~~¬Own(YOU, D) ∨~~ ¬Hate(ROBIN, D) {D/z2} [from 2, 12]
14. ¬ Own (ROBIN, y1) ∨ ¬ Rabbit(y1) ∨ ¬ Rabbit(w1) ∨ ¬ Chase (D, w1) ~~∨ Hate (ROBIN, D)~~ {ROBIN/x1, D/z1} [from 4, 13]
15. ~~¬BuyCarrots(ROBIN) ∨~~ Own(ROBIN, B(ROBIN)) {ROBIN/x3} [from 3, 7]
16. ~~¬ Own (ROBIN, B(ROBIN)) ∨~~ ¬ Rabbit(B(ROBIN)) ∨ ¬ Rabbit(w1) ∨ ¬ Chase (D, w1) {B(ROBIN)/y1} [from 14, 15]
17. ~~¬Dog(D) ∨~~ Chase (D, A(D)) {D/x2} [from 1, 6]
18. ¬ Rabbit(B(ROBIN)) ∨ ¬ Rabbit(A(D)) ~~∨ ¬ Chase (D, A(D))~~ {A(D)/w1} [from 16, 17]
19. ~~¬Dog(D) ∨~~ Rabbit(A(D)) {D/x2} [from 1, 5]
20. ¬ Rabbit(B(ROBIN)) ~~∨ ¬ Rabbit(A(D)~~ {} [from 18, 19]
21. ~~¬BuyCarrots(ROBIN) ∨~~ Rabbit(B(ROBIN) ∨ GroceryStore(B(ROBIN)) {ROBIN/x3} [from 3, 8]
22. ~~Rabbit(B(ROBIN) ∨~~ GroceryStore(B(ROBIN)) {} [from 20, 21]
23. ¬Own (ROBIN, (B(ROBIN)) ~~∨ ¬GroceryStore((B(ROBIN))~~ {(B(ROBIN)/y} [from 10, 22]
24. Φ {} [from 15, 23]

Since the resolution led to an empty clause, which indicates a contradiction, it means the conclusion is valid, and Madame Irma's prediction is true. Therefore, you should go see Robin and declare your love to her.

Lost in closet.

3.1 Choosing a loss function.

The choice of a loss function depends on the nature of the task you are trying to solve. For the Fashion MNIST dataset, which is a dataset of grayscale images of 10 different clothing categories, it is a classification problem. Thus, for this task it's best to use the CrossEntropyLoss function, which combines the SoftMax activation and the cross-entropy loss in a single step. This is appropriate for multi-class classification problems, such as the one posed by the Fashion MNIST dataset.

SoftMax Activation: Before computing the loss, the raw output from the neural network (logits) is passed through the SoftMax function. SoftMax normalizes the output values into probabilities, ensuring that they sum to 1. This is done for each example in the batch and for each class independently.

Cross-Entropy Loss: After the SoftMax activation, the cross-entropy loss is calculated. For each example, the cross-entropy loss measures the difference between the predicted class probabilities and the true class labels. The formula for the cross-entropy loss for a single example is:

$$\text{CrossEntropyLoss} = -\sum_i \text{target}_i \cdot \log(\text{softmax}(x)_i)$$

where target_i is 1 if the true class is i and 0 otherwise, and $\text{SoftMax}(x)_i$ is the predicted probability of class i after the SoftMax activation.

Final Loss: The cross-entropy losses for all examples in the batch are averaged to obtain the final loss that is used to update the model's parameters during backpropagation.

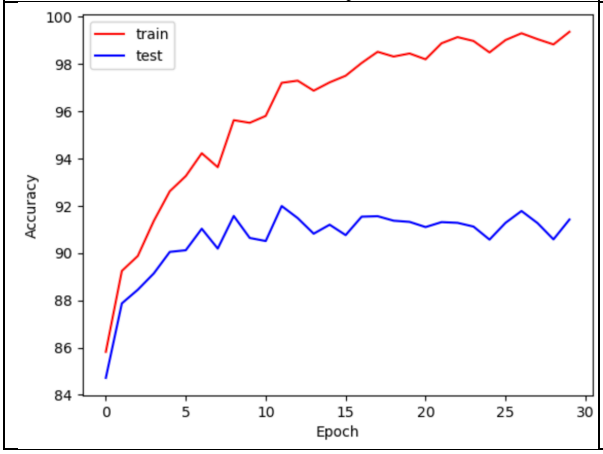
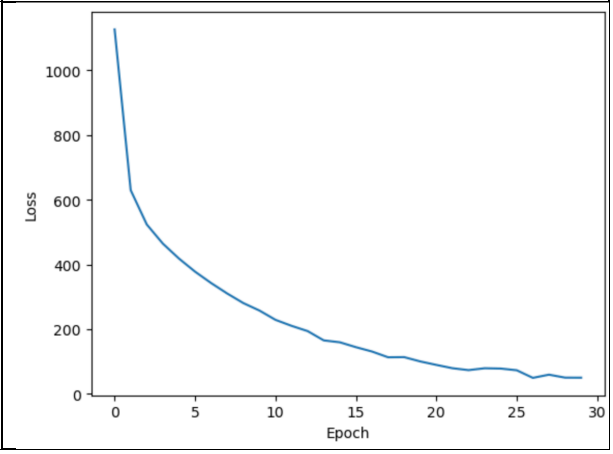
3.2 Model architecture

```
FashionCNN(
  (conv1): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1))
  (activation1): ReLU()
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1))
  (activation2): ReLU()
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=1024, out_features=1024, bias=True)
  (activation3): ReLU()
  (fc2): Linear(in_features=1024, out_features=256, bias=True)
  (activation4): ReLU()
  (fc3): Linear(in_features=256, out_features=10, bias=True)
)
```

Note: I've referred to Lab7 session on the CNNs to arrive at my solutions for the questions below.

3.2.8 Experiment 1

Epochs	30
Learning rate	0.1
Optimizer	SGD
Training Accuracy	99.36%
Test Accuracy	91.42%
Loss	49

Accuracy	***Loss***
	

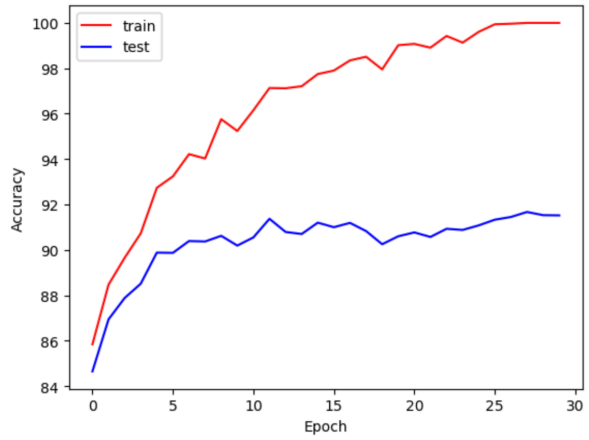
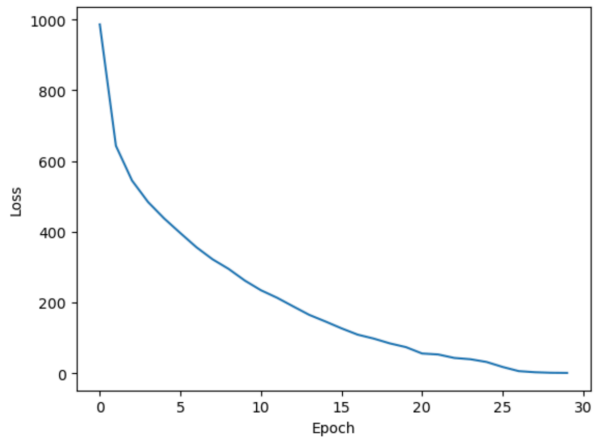
b) On the speed of performance changes across epochs, we see an early raise in accuracy at 88% which then raises further until 96% beyond this point the change in accuracy is quite minimal and saturates for later epochs.

c) On the speed of decrease of loss, is analogous the change in accuracy and the drop in loss is gradual until epoch 15 and then tapers of for further epochs as the training is close to completion.

3.3 Choosing activation functions.

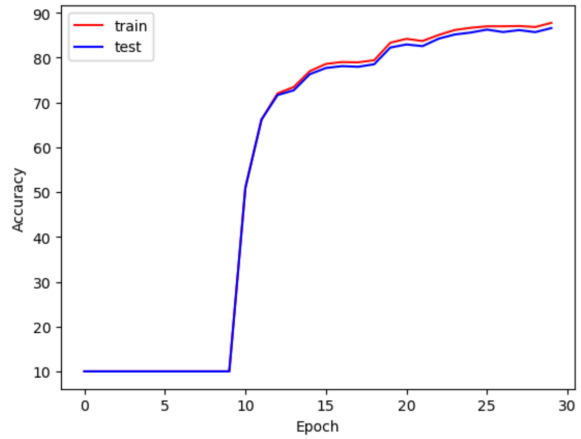
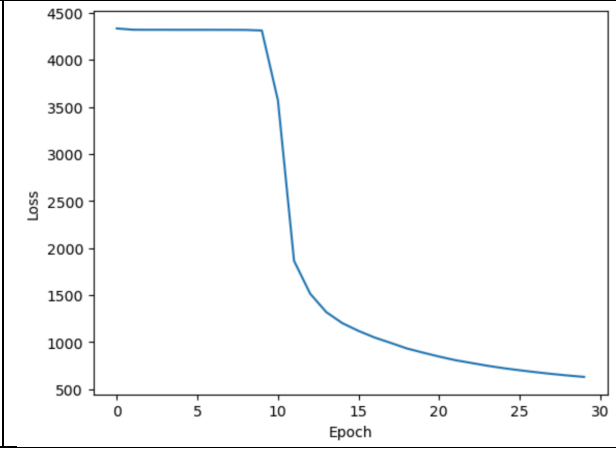
Tanh

Epochs	30
Learning rate	0.1
Optimizer	SGD
Training Accuracy	96.54%
Test Accuracy	91.12%
Loss	49

Training accuracy	***Training loss***
	

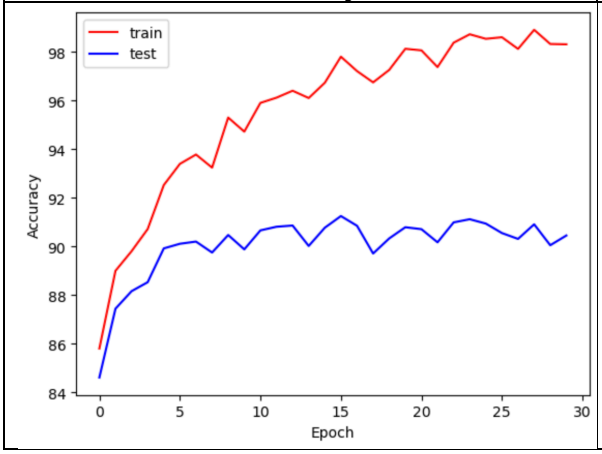
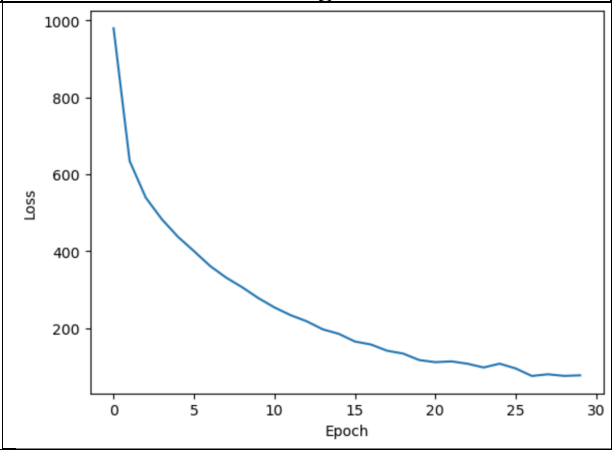
Sigmoid

Epochs	30
Learning rate	0.1
Optimizer	CrossEntropyLoss
Training Accuracy	87.77%
Test Accuracy	86.61%
Loss	631.72

Accuracy	***Loss***
	

ELU

Epochs	30
Learning rate	0.1
Optimizer	CrossEntropyLoss
Training Accuracy	98.31%
Test Accuracy	90.45
Loss	77.48

Accuracy	***Training loss***
	

The choice of activation function often depends on the specific characteristics of your data and the nature of your problem. It's a good practice to experiment with different activation functions and architectures to find what works best for a given use case.

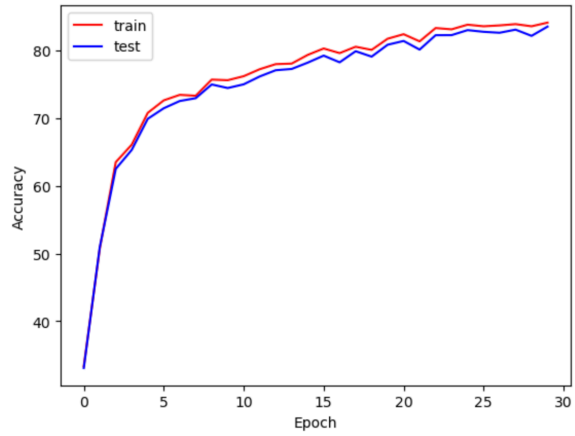
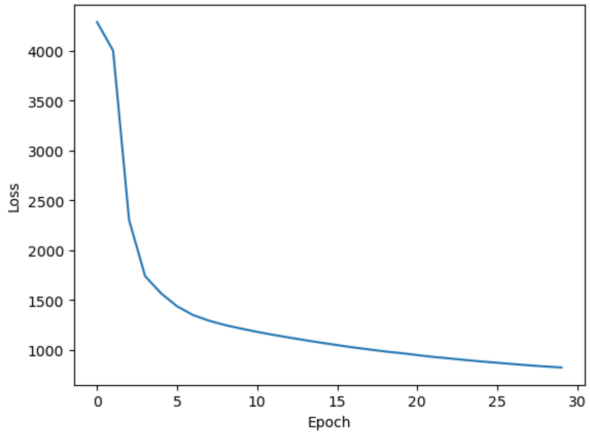
Tanh activation performs the best when compared with ELU and Sigmoid. The tanh function squashes input values to the range of [-1, 1], which helps mitigate the vanishing gradient problem better than the sigmoid function. It is zero-cantered, making it sometimes easier to optimize because the negative and positive regions are balanced.

ELU ranging from (-∞, ∞), is designed to mitigate the vanishing gradient problem and allows the model to learn representations with a more informative gradient. It has a non-zero mean output, which helps with optimization. ELU tends to perform well in deeper networks.

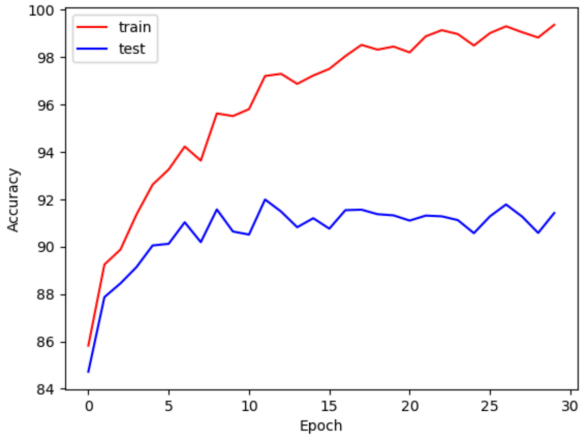
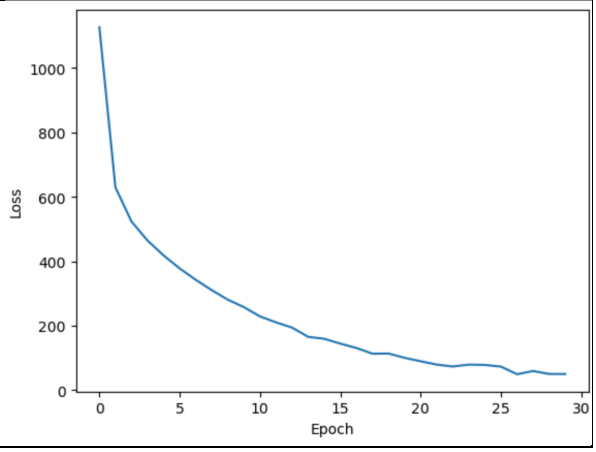
The sigmoid function is commonly used in binary classification problems at the output layer, where the goal is to produce probabilities between 0 and 1. It suffers from the vanishing gradient problem, especially during backpropagation in deep networks which explains the noticeable lower performance in the above comparisons.

3.3 Choosing learning rates.

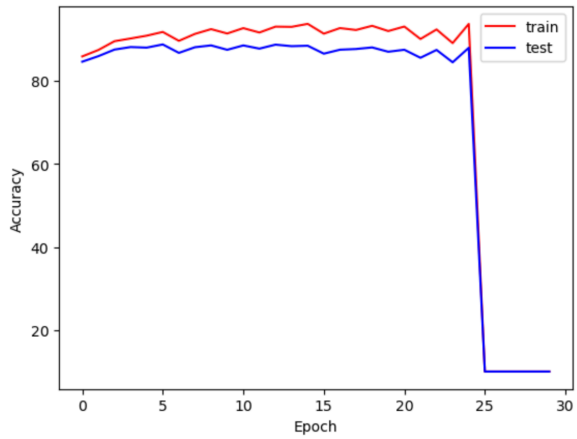
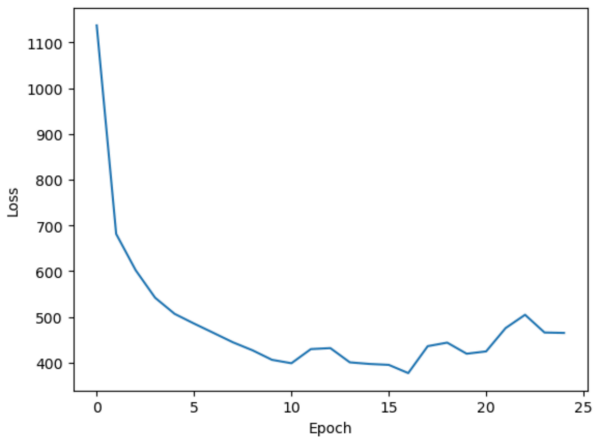
Epochs	30
Learning rate	0.001
Optimizer	CrossEntropyLoss
Training Accuracy	84.12%
Test Accuracy	83.54%
Loss	653.34

Accuracy	***Loss***
	

Epochs	30
Learning rate	0.1
Optimizer	CrossEntropyLoss
Training Accuracy	99.36%
Test Accuracy	91.42%
Loss	49.67

Accuracy	***Loss***
	

Epochs	30
Learning rate	0.5
Optimizer	CrossEntropyLoss
Training Accuracy	10.0% *
Test Accuracy	10.0% +
Loss	NaN

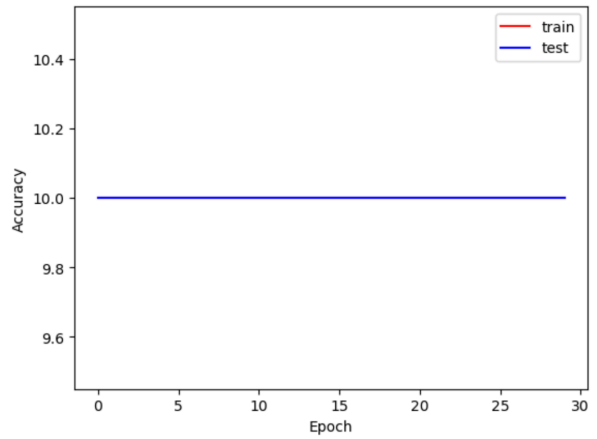
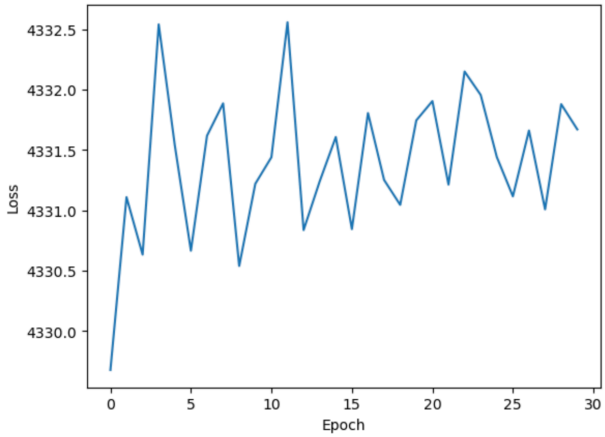
Accuracy	***Loss***
	

A high learning rate can cause the optimization algorithm to overshoot the minimum of the loss function. When the model's weights become very large, computations involving these weights (e.g., exponentiation) can lead to overflow, resulting in NaN values. High learning rates can lead to large gradients during backpropagation. Large gradients may cause weight updates that are so substantial that they result in overflow or NaN values in the weight parameters. Some loss functions involve operations that are sensitive to very large numbers, especially when the output values are exponential or logarithmic. For instance, taking the logarithm of a very large number can result in infinity or NaN. High learning rates exacerbate the issues of vanishing or exploding gradients. Exploding gradients can lead to NaN values as the weight updates become extremely large.

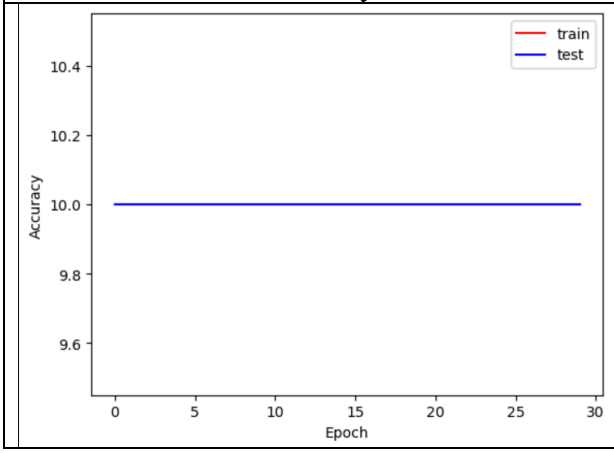
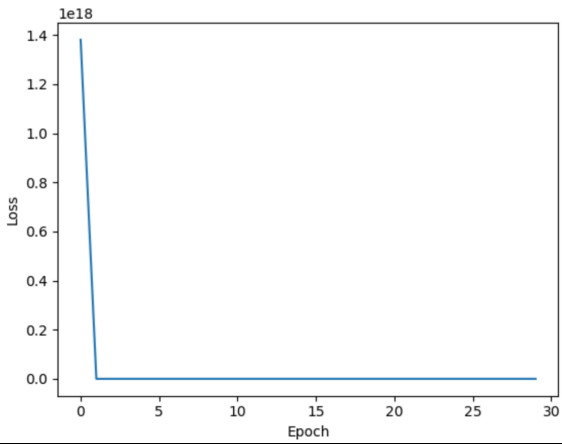
To mitigate NaN issues caused by a high learning rate, consider the following solutions:

1. Gradually reduce the learning rate to a more moderate value. This can prevent overshooting the minimum and allow for more stable convergence.
2. Implement gradient clipping to limit the size of gradients during training. This prevents large updates that can lead to instability.
3. Ensure that the model's weights are initialized properly. Weight initialization techniques like Xavier/Glorot or He initialization can help stabilize training.
4. Consider using optimization algorithms with built-in mechanisms for handling large learning rates, such as adaptive learning rate methods (e.g., Adam, RMSprop, or Adagrad).

Epochs	30
Learning rate	1
Optimizer	CrossEntropyLoss
Training Accuracy	10.00%
Test Accuracy	10.00%
Loss	4331.67

Accuracy	***Loss***
	

Epochs	30
Learning rate	10
Optimizer	CrossEntropyLoss
Training Accuracy	10.00%
Valid Accuracy	10.00%
Loss	4635.32

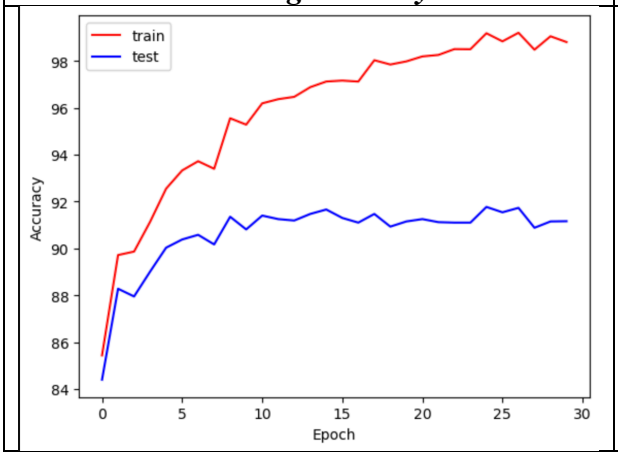
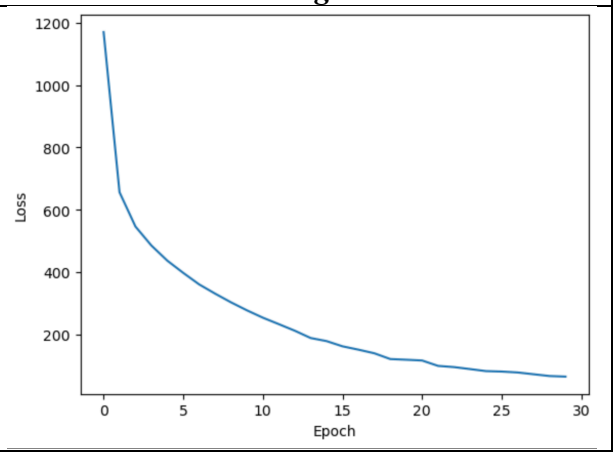
Accuracy	***Loss***
	

3.3 Impact of dropouts.

Model architecture

```
FashionCNN(
  (conv1): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1))
  (activation1): ReLU()
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1))
  (activation2): ReLU()
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=1024, out_features=1024, bias=True)
  (activation3): ReLU()
  (fc2): Linear(in_features=1024, out_features=256, bias=True)
  (activation4): ReLU()
  (dropout): Dropout(p=0.3, inplace=False)
  (fc3): Linear(in_features=256, out_features=10, bias=True)
)
```

Epochs	30
Learning rate	0.1
Optimizer	CrossEntropyLoss
Train Accuracy	98.80%
Test Accuracy	91.16%
Loss	64.84

Training accuracy	*Training loss*
	

Dropout is a regularization technique used in neural networks to improve generalization and prevent overfitting. During training, dropout randomly sets a fraction of the neurons in a layer to zero, which helps prevent co-adaptation (refers to a phenomenon where neurons in a neural network learn to work together and become highly correlated) of hidden units.

1. One of the primary benefits of dropout is its ability to reduce overfitting. By randomly dropping out neurons during training, dropout prevents the model from relying too heavily on specific neurons, making it more robust and less likely to memorize the training data.
2. With dropout, different subsets of neurons are dropped out during each training iteration. This encourages the network to learn more independent and useful features, as different combinations of neurons are forced to carry the load.