AI Submission 2

Gargeya Sharma – MSc Artificial Intelligence – 220278025

2. Crystal Clear (Logic)

2.1

1. ∃x Dog(x) ∧ Own(YOU, x)
2. Buy(ROBIN)
3. ∀x ∃y (Rabbit(y) ∧ Own(x, y)) → ∀z ∃w (Chase (z, w) ∧ Rabbit(w)) → Hates(x, z)
4. ∀x Dog(x) → ∃y(Rabbit(y) ∧ Chase(x, y))
5. ∀x Buy(x) → Own(x, y) ∧ ∃y(Rabbit(y) ∨ Grocery_store(y))
6. ∀x ∀y ∃z (Own(y, z) ∧ Hates(x, z)) → ¬Date(x, y)

***Note**: We have 2 literals {ROBIN, YOU}*

2.2

Step 1: Rewrite →

1. Nothing to do
2. Nothing to do
3. ∀x ¬∃y(Rabbit(y) ∧ Own(x,y)) ∨ ∀z ¬∃w(Chase(z,w) ∧ Rabbit(w)) ∨ Hates(x,z)
4. ∀x ¬Dog(x) ∨ ∃y(Rabbit(y) ∧ Chase(x, y))
5. ∀x ¬Buy(x) ∨ ∃y(Own(x, y) ∧ (Rabbit(y) ∨ Grocery_store(y)))
6. ∀x ∀y ¬∃z (Own(y, z) ∧ Hates(x, z)) ∨ ¬Date(x, y)

Step 2: Minimize Negations using logical definitions

1. Nothing to do
2. Nothing to do
3. ∀x ∀y(¬Rabbit(y) ∨ ¬Own(x,y)) ∨ ∀z ∀w(¬Chase(z,w) ∨ ¬Rabbit(w)) ∨ Hates(x,z)
4. Nothing to do
5. Nothing to do
6. ∀x ∀y ∀z (¬Own(y, z) ∨ ¬Hates(x, z)) ∨ ¬Date(x, y)

Step 3: Standardize variables apart

1. ∃x1 Dog(x1) ∧ Own(YOU, x1)
2. Nothing to do
3. ∀x2 ∀y1(¬Rabbit(y1) ∨ ¬Own(x2,y1)) ∨ ∀z1 ∀w1(¬Chase(z1,w1) ∨ ¬Rabbit(w1)) ∨ Hates(x2,z1)
4. ∀x3 ¬Dog(x3) ∨ ∃y2(Rabbit(y2) ∧ Chase(x3, y2))
5. ∀x4 ¬Buy(x4) ∨ ∃y3(Own(x4, y3) ∧ (Rabbit(y3) ∨ Grocery_store(y3)))
6. ∀x5 ∀y4 ∀z2 (¬Own(y4, z2) ∨ ¬Hates(x5, z2)) ∨ ¬Date(x5, y4)

Step 4: Skolemise

1. Dog(D) ∧ Own(YOU, D)
2. Nothing to do
3. Nothing to do
4. ∀x3 ¬Dog(x3) ∨ (Rabbit(R(x3)) ∧ Chase(x3, R(x3)))
5. ∀x4 ¬Buy(x4) ∨ (Own(x4, Q(x4)) ∧ (Rabbit(Q(x4)) ∨ Grocery_store(Q(x4))))
6. Nothing to do

Step 5: Drop universal quantifiers

1. Nothing to do
2. Nothing to do
3. ¬Rabbit(y1) ∨ ¬Own(x2,y1) ∨ ¬Chase(z1,w1) ∨ ¬Rabbit(w1) ∨ Hates(x2,z1)

4. ¬Dog(x3) ∨ (Rabbit(R(x3)) ∧ Chase(x3, R(x3)))
5. ¬Buy(x4) ∨ (Own(x4, Q(x4)) ∧ (Rabbit(Q(x4)) ∨ Grocery_store(Q(x4))))
6. ¬Own(y4, z2) ∨ ¬Hates(x5, z2)) ∨ ¬Date(x5, y4)

Step 6: Convert to CNFs

1. Nothing to do
2. Nothing to do
3. Nothing to do
4. **(**¬Dog(x3) ∨ Rabbit(R(x3))**)** ∧ **(**¬Dog(x3) ∨ Chase(x3, R(x3))**)**
5. **(**¬Buy(x4) ∨ Own(x4, Q(x4))**)** ∧ **(**¬Buy(x4) ∨ Rabbit(Q(x4)) ∨ Grocery_store(Q(x4))**)**
6. Nothing to do

Final Set:

1. Dog(D) ∧ Own(YOU, D)
2. Buy(ROBIN)
3. ¬Rabbit(y1) ∨ ¬Own(x2,y1) ∨ ¬Chase(z1,w1) ∨ ¬Rabbit(w1) ∨ Hates(x2,z1)
4. **(**¬Dog(x3) ∨ Rabbit(R(x3))**)** ∧ **(**¬Dog(x3) ∨ Chase(x3, R(x3))**)**
5. **(**¬Buy(x4) ∨ Own(x4, Q(x4))**)** ∧ **(**¬Buy(x4) ∨ Rabbit(Q(x4)) ∨ Grocery_store(Q(x4))**)**
6. ¬Own(y4, z2) ∨ ¬Hates(x5, z2)) ∨ ¬Date(x5, y4)

2.3

Conclusion: ¬∃x (Grocery_store(x) ∧ Own(ROBIN, x)) → ¬Date(ROBIN, YOU)

Negative Conclusion: ¬∃x (Grocery_store(x) ∧ Own(ROBIN, x)) ∧ Date(ROBIN, YOU)

From now we will use the negative conclusion to get NULL set during resolution.

Step 1: Rewrite →

- Nothing to do

Step 2: Minimize Negations using logical definitions

- ∀x (¬Grocery_store(x) ∨ ¬Own(ROBIN, x)) ∧ Date(ROBIN, YOU)

Step 3: Standardize variables apart

- Nothing to do

Step 4: Skolemise

- Nothing to do

Step 5: Dropping Universal Quantifiers

- (¬Grocery_store(x) ∨ ¬Own(ROBIN, x)) ∧ Date(ROBIN, YOU)

Step 6: Convert to CNFs

- Nothing to do

Conclusion:

(¬Grocery_store(x) ∨ ¬Own(ROBIN, x)) ∧ Date(ROBIN, YOU)

2.4

Final Set after applying step-7 and 8 of :

1. Dog(D)
2. Own(YOU, D1)
3. Buy(ROBIN)
4. ¬Rabbit(y1) ∨ ¬Own(x2,y1) ∨ ¬Chase(z1,w1) ∨ ¬Rabbit(w1) ∨ Hates(x2,z1)
5. ¬Dog(x3) ∨ Rabbit(R(x3))
6. ¬Dog(x6) ∨ Chase(x6, R(x6))
7. ¬Buy(x4) ∨ Own(x4, Q(x4))
8. ¬Buy(x7) ∨ Rabbit(Q(x7)) ∨ Grocery_store(Q(x7))

9. ¬Own(y4, z2) ∨ ¬Hates(x5, z2)) ∨ ¬Date(x5, y4)
10. ¬Grocery_store(x) ∨ ¬Own(ROBIN, x)
11. Date(ROBIN, YOU)

Resolution:

From ③ & ⑦
~~Buys(ROBIN)~~ ~~¬Buys(ROBIN)~~ ∨ Own(ROBIN, Q(x4)) {ROBIN/x4}

with ⑨ Own(ROBIN, Q(x4)) ¬Own(YOU, z2) ∨ ¬Hates(x5, z2) ∨ ¬Date(x5, YOU) {YOU/y4}

with ② Own(ROBIN, Q(x4)) ∨ ~~¬Own(YOU, z2)~~ ∨ ¬Hates(x5, z2) ∨ ¬Date(x5, YOU) ~~Own(YOU, z2)~~ {z2/D1}

with ④ Own(ROBIN, Q(x4)) ∨ ~~¬Hates(x5, z2)~~ ∨ ¬Date(x5, YOU) {x5/x2, z2/z1}
~~Hates(x5, z2)~~ ∨ ¬Rabbit(y1) ∨ ¬Rabbit(w1) ∨ ¬Own(x5, y1) ∨ ¬Chase(z2, w1)

with ⑥ Own(ROBIN, Q(x4)) ∨ ¬Date(x5, YOU) ∨ ¬Rabbit(y1) ∨ ¬Rabbit(w1) ∨ ¬Own(x5, y1) ∨ ~~¬Chase(z2, w1)~~ {z2/x6, w1/R(x6)}
¬Dog(z2) ∨ ~~Chase(z2, w1)~~

with ⑤ Own(ROBIN, Q(x4)) ∨ ¬Date(x5, YOU) ∨ ~~¬Rabbit(R(x3))~~ ∨ ~~¬Rabbit(R(x3))~~ ∨ ¬Own(x5, y1) ∨ ¬Dog(z2) {R(x3)/y1, R(x3)/w1, z2/x3}
¬Dog(z2) ∨ ~~Rabbit(R(x3))~~

with ① Own(ROBIN, Q(x4)) ∨ ¬Date(x5, YOU) ∨ ¬Own(x5, y1) ∨ ~~¬Dog(z2)~~ ∨ ~~¬Dog(z2)~~ {z2/D}
~~Dog(z2)~~

with ⑪ ~~Own(ROBIN, y1)~~ ∨ ~~¬Date(ROBIN, YOU)~~ ∨ ~~¬Own(ROBIN, y1)~~ {ROBIN/x5, y1/Q(x4)}
~~Date(ROBIN, YOU)~~

Hence proved: { }

3. Lost in the closet (Classification)

3.1

Cross Entropy loss would be the most appropriate loss function we can use here. Main reason would be the number of classes we are dealing with classification. Here, there are 10 distinct classes so sigmoid function that we have been using until now for binary classification is insufficient. Cross Entropy can deal with multi-class classification.

Formula: $-\sum_{i=1}^{n} y_i \log(p_i)$

n: number of classes **yi** = Ground truth class (c) of the current instance, **p** = predicted probability for instance to lie in class ‘c’, **log** = natural log

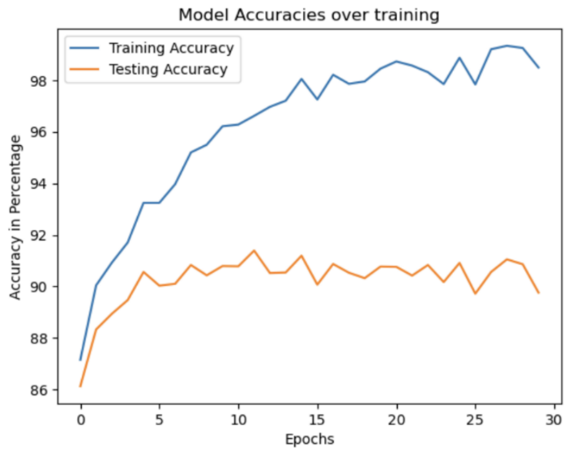
Cross entropy works by using the target class label and multiplying it by the natural log of probability of predicting that class for that instance. So, it can work for any many classes as we want. That’s why for this use case of 10 classes, we are using cross entropy loss.

3.2

Note: In this model I have used ReLU as my activation function.

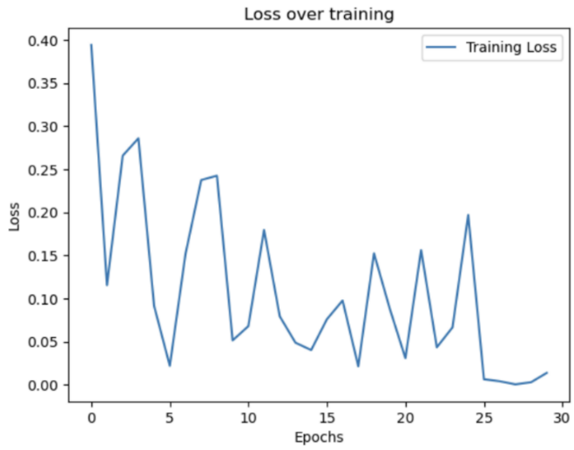
a.) Train Accuracy: 98.4883% Test Accuracy: 89.76%

b.) Plot of the accuracy on the training and test sets per each epoch



Because we are starting from a random point in the loss plane, chances of improvements in the earlier epochs are higher so the speed of performance is higher in earlier epochs. As we reach to an area where it needs to be refined and gradient is relatively not that high, the rate of increase of accuracy decreases in finding the optimal local minima. In our case the shape of the training accuracy plot looks like a curve while after 4 epochs, testing accuracy is pretty much stuck between ~90%. The increase in training accuracy but not in testing accuracy after this point is due to overfitting of the model on the training dataset as the model performance is not improving further on unseen (testing) dataset.

c.) Plot the train loss per epoch



As mentioned above, the logic stays the same with the decrease in rate of decrease of loss over increasing epochs. But another thing to notice here are the spikes in the loss plot after couple of epochs. The spikes are most probably the result of a rough loss plain. Reducing the learning rate can smooth the process but most importantly, matplotlib projects plot between the highest and lowest value in a range. So, the range of loss decrement during the training will determine how smooth our plot will look and the scale of the loss. We will encounter the graphs with different learning rate in the questions below, you can take reference of this change of ranges on y-axis from the comparisons there.

3.3

Activation Functions:

Tanh:	'Training Accuracy': 100.0%	'Testing Accuracy': 91.56%
Sigmoid:	'Training Accuracy': 90.8417%	'Testing Accuracy': 89.12%
ELU:	'Training Accuracy': 98.4133%	'Testing Accuracy': 90.26%

We can see that Tanh activation function performed the best in training accuracy and testing accuracy values in comparison to Sigmoid and ELU. But the gap between the training and testing accuracy is also the maximum with Tanh, which means that the highest level of overfitting is occurring with it. Model with sigmoid activation function is giving us the most stable model in terms of performance gap between seen(training) and unseen(testing) dataset. But with this, Training accuracy of sigmoid is also the lowest among all the candidates. Model using ELU is in between both other activation functions. The performance gap in training and testing accuracies is like that of Tanh so, it closer to how Tanh is responding on the data. They all have different mathematical functioning so difference in influence towards the model is expected.

3.4

Learning rates:

0.001 : {Loss: 0.3083, Training Accuracy: 88.0417, Testing Accuracy: 86.74}

0.1 : {Loss: 0.0115, Training Accuracy: 99.16, Testing Accuracy: 91.07}

0.5 : {Loss: nan, Training Accuracy: 10.0, Testing Accuracy: 10.0}

1 : {Loss: nan, Training Accuracy: 10.0, Testing Accuracy: 10.0}

10 : {Loss: nan, Training Accuracy: 10.0, Testing Accuracy: 10.0}

***NOTE**: For Visualization, refer to Notebook of the solution*

Yes, there is a tradeoff between speed and stability of convergence with respect to learning rate. Learning rate decides the step size for the optimization process during backpropagation to tune the gradients. If the step size is large, jump towards reducing the loss by updating the current parameters values will be large as well and vice versa. A loss plane depending on the problem varies in complexity hence, too large of a step throws you all around the loss plane with no sense of consistent direction. Alternatively, too small of a step is just time and computational exhaustive approach. Choosing the best learning rate for each problem/model is best decided with experimentation and hyper parameters tuning. Getting nan values in higher learning rates is due to a process called exploding gradients/vanishing gradients where your selected large learning rate updates the parameters values as either infinitely large or infinitely small respectively for a computer to store it as data types. Training process is a highly iterative process so, a large learning rate might not look that large of a number, but its iterative effect is what causes this problem.

3.5

Model with dropout layer with p = 0.3 after the second dense layer

Train Accuracy: 98.4583%

Test Accuracy: 90.04%

So, compared to the same model architecture performance but without dropout layer is pretty much the same as this one with dropout layer.

First point that explains this; is the value of dropout that we have used, its 0.3 (30%), which is a decent number but not too great in our dense network to show significant impact. There is still a lot of scope for overfitting left with thousands of other trainable parameters, hence shown in both their performances.

Second reason, I can infer is the use of only single dropout layer. If we are using more dropout layers than it would be harder for model to find a way around regularization and eventually decrease the degree of overfitting and reduce the gap between training accuracy and test accuracy.