Artificial Intelligence

ECS629U-759P

Coursework2

Spyridon Roumpis

181004877

Question 1: Crystal clear!

(a)

1. ∃ x (DOG(x) ∧ OWNS(You,x))
2. BUY(Robin)
3. ∀ x ∀ y (OWNS(x,y) ∧ RABBIT(y) → ∀ z ∀ w (RABBIT(w) ∧ CHASE(z,w) → HATES(x,z)))
4. ∀ x (DOG(x) → ∃ y (RABBIT(y) ∧ CHASE(x,y)))
5. ∀ x (BUY(x) → ∃ y (OWNS(x,y) ∧ (RABBIT(y) ∨ GROCERY(y))))
6. ∀ x ∀ y ∀ z (OWNS(y,z) ∧ HATES(x,z) → ¬ DATE(x,y))

(b)

1. DOG(A) ∧ OWNS(You,A)

2. BUY(Robin)

3. ¬ OWNS(x1,x2) ∨ ¬ RABBIT(x2) ∨ ¬ RABBIT(x3) ∨ ¬ CHASE(x4,x3) ∨ HATES(x1,x2)

4. ¬ DOG(x5) ∨ RABBIT(f1(x5)) ∧ CHASE(x5, f1(x5))

5. ¬ BUY(x6) ∨ OWNS(x6, f2(x6)) ∧ RABBIT(f2(x6)) ∨ GROCERY(f2(x6))

6. ¬ OWNS(x7,x8) ∨ ¬ HATES(x9,x8) ∨ ¬ DATE(x9,x7))

(c) If Robin does not own a grocery store, she will not date you.
FOL: (¬ ∃ x (GROCERY(x) ∧ OWN(Robin,x))) → ¬ DATE(Robin,You)

Negate: ¬ ((¬ ∃ x (GROCERY(x) ∧ OWN(Robin,x))) → ¬ DATE(Robin,You))

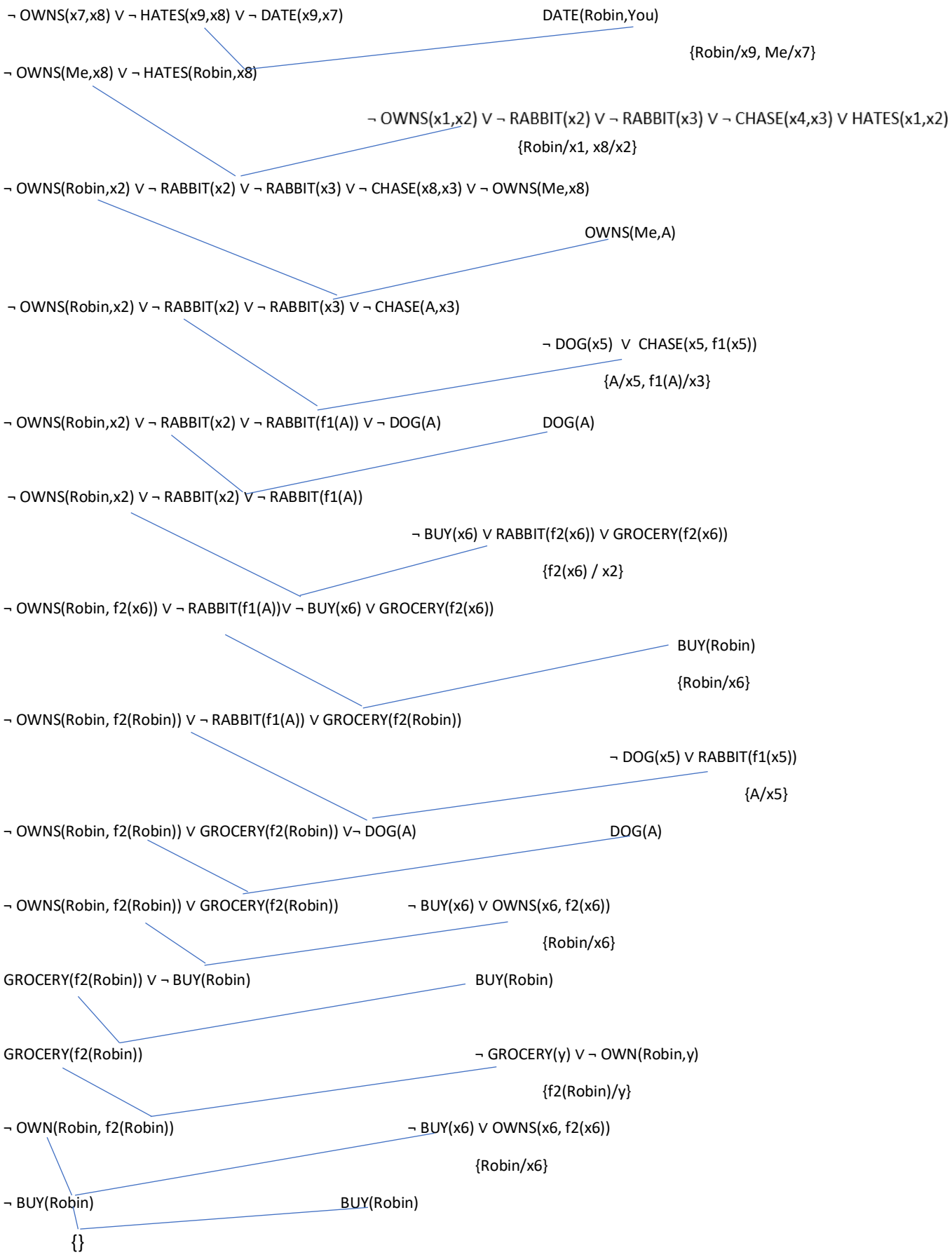
CNF: ¬ ((∀ x ¬ (GROCERY(x) ∧ OWN(Robin,x))) → ¬DATE(Robin,You))

CNF: ¬ ((∀ x ¬ ¬ (GROCERY(x) ∧ OWN(Robin,x))) ∨ ¬DATE(Robin,You))

CNF : (¬ GROCERY(y) ∨ ¬ OWN(Robin,y)) ∧ DATE(Robin,You)

(d) Split into disjunctive clauses :

1. DOG(A)
2. OWNS(Me,A)
3. BUY(Robin)
4. ¬ OWNS(x1,x2) ∨ ¬ RABBIT(x2) ∨ ¬ RABBIT(x3) ∨ ¬ CHASE(x4,x3) ∨ HATES(x1,x2)
5. ¬ DOG(x5) ∨ RABBIT(f1(x5))
6. ¬ DOG(x5) ∨ CHASE(x5, f1(x5))
7. ¬ BUY(x6) ∨ OWNS(x6, f2(x6))
8. ¬ BUY(x6) ∨ RABBIT(f2(x6)) ∨ GROCERY(f2(x6))
9. ¬ OWNS(x7,x8) ∨ ¬ HATES(x9,x8) ∨ ¬ DATE(x9,x7))
10. ¬ GROCERY(y) ∨ ¬ OWN(Robin,y)
11. DATE(Robin,Me)


¬ OWNS(x7,x8) V ¬ HATES(x9,x8) V ¬ DATE(x9,x7)
DATE(Robin,You)
{Robin/x9, Me/x7}
¬ OWNS(Me,x8) V ¬ HATES(Robin,x8)
¬ OWNS(x1,x2) V ¬ RABBIT(x2) V ¬ RABBIT(x3) V ¬ CHASE(x4,x3) V HATES(x1,x2)
{Robin/x1, x8/x2}
¬ OWNS(Robin,x2) V ¬ RABBIT(x2) V ¬ RABBIT(x3) V ¬ CHASE(x8,x3) V ¬ OWNS(Me,x8)
OWNS(Me,A)
¬ OWNS(Robin,x2) V ¬ RABBIT(x2) V ¬ RABBIT(x3) V ¬ CHASE(A,x3)
¬ DOG(x5) V CHASE(x5, f1(x5))
{A/x5, f1(A)/x3}
¬ OWNS(Robin,x2) V ¬ RABBIT(x2) V ¬ RABBIT(f1(A)) V ¬ DOG(A)
DOG(A)
¬ OWNS(Robin,x2) V ¬ RABBIT(x2) V ¬ RABBIT(f1(A))
¬ BUY(x6) V RABBIT(f2(x6)) V GROCERY(f2(x6))
{f2(x6) / x2}
¬ OWNS(Robin, f2(x6)) V ¬ RABBIT(f1(A))V ¬ BUY(x6) V GROCERY(f2(x6))
BUY(Robin)
{Robin/x6}
¬ OWNS(Robin, f2(Robin)) V ¬ RABBIT(f1(A)) V GROCERY(f2(Robin))
¬ DOG(x5) V RABBIT(f1(x5))
{A/x5}
¬ OWNS(Robin, f2(Robin)) V GROCERY(f2(Robin)) V¬ DOG(A)
DOG(A)
¬ OWNS(Robin, f2(Robin)) V GROCERY(f2(Robin))
¬ BUY(x6) V OWNS(x6, f2(x6))
{Robin/x6}
GROCERY(f2(Robin)) V ¬ BUY(Robin)
BUY(Robin)
GROCERY(f2(Robin))
¬ GROCERY(y) V ¬ OWN(Robin,y)
{f2(Robin)/y}
¬ OWN(Robin, f2(Robin))
¬ BUY(x6) V OWNS(x6, f2(x6))
{Robin/x6}
¬ BUY(Robin)
BUY(Robin)
{}

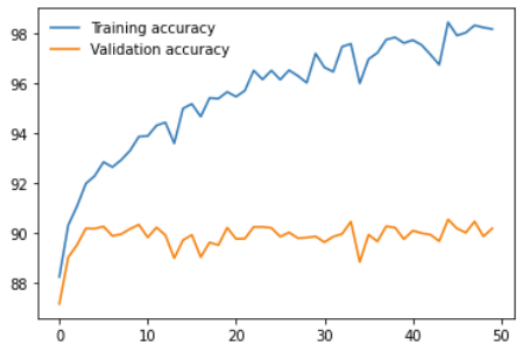
Question 2: Lost in the closet

(a) The choice of an error function, conventionally called a loss function, is used to estimate the loss of the model so that the weights can be updated to reduce the loss on the next evaluation. Our case is a multi-class classification problem with 10 different classes. The options for the loss function are Multi-Class Cross-Entropy Loss, Sparse Multiclass Cross-Entropy Loss(classification problems with a large number of labels is the one-hot encoding process.), Kullback Leibler Divergence Loss(is more commonly used when using models that learn to approximate a more complex function than simply multi-class classification).

Cross-Entropy is the default and preferred loss function under the inference framework of maximum likelihood to use for multi-class classification problems and it's the one used to for this task. a score that summarizes the average difference between the actual and predicted probability distributions will be calculated from it. The score is minimized and a perfect cross-entropy value is 0.

(b)

<matplotlib.legend.Legend at 0x7f701bf5c0f0>




```
Epoch 1: loss: 915.8150492608547, train accuracy: 88.24166666666666, valid accuracy:87.16
Epoch 2: loss: 586.512155842036, train accuracy: 90.305, valid accuracy:89.02
Epoch 3: loss: 501.1690686196089, train accuracy: 91.06833333333333, valid accuracy:89.52
Epoch 4: loss: 446.2080144546926, train accuracy: 91.97166666666666, valid accuracy:90.18
Epoch 5: loss: 398.9168307669461, train accuracy: 92.28, valid accuracy:90.17
Epoch 6: loss: 358.4783060327172, train accuracy: 92.835, valid accuracy:90.26
Epoch 7: loss: 322.32699505705386, train accuracy: 92.63166666666666, valid accuracy:89.88
Epoch 8: loss: 288.76605114061385, train accuracy: 92.91833333333334, valid accuracy:89.95
Epoch 9: loss: 255.95856711361557, train accuracy: 93.29166666666667, valid accuracy:90.16
Epoch 10: loss: 235.38369527924806, train accuracy: 93.85166666666667, valid accuracy:90.33
...
Epoch 40: loss: 70.18416380994336, train accuracy: 97.59166666666667, valid accuracy:89.75
Epoch 41: loss: 63.37097886790934, train accuracy: 97.71166666666667, valid accuracy:90.09
Epoch 42: loss: 54.501125169825514, train accuracy: 97.51666666666667, valid accuracy:89.99
Epoch 43: loss: 50.99667830328618, train accuracy: 97.13166666666666, valid accuracy:89.93
Epoch 44: loss: 48.302877149134304, train accuracy: 96.72666666666667, valid accuracy:89.67
Epoch 45: loss: 66.78821341346554, train accuracy: 98.425, valid accuracy:90.54
Epoch 46: loss: 47.087433964532664, train accuracy: 97.89333333333333, valid accuracy:90.18
Epoch 47: loss: 79.28076845115316, train accuracy: 98.00833333333334, valid accuracy:90.01
Epoch 48: loss: 63.84675672900052, train accuracy: 98.30333333333333, valid accuracy:90.46
Epoch 49: loss: 49.24497045820425, train accuracy: 98.21333333333334, valid accuracy:89.86
Epoch 50: loss: 74.63194276377378, train accuracy: 98.15166666666667, valid accuracy:90.18
```

As the number of epochs increases, more number of times the weight are changed in the neural network, and the model is trained better, although there is always the chance of overfitting, after a certain point of using more and more epochs.

It can be seen that the loss is kept decreasing until a certain point (around 30 epochs) and then is a bit unstable. This could be due to the model overfitting on the training data that don't generalize to the test data. Although overall the loss was decreased dramatically, it started with 915.81 and ended at 74.63.

(c)

- Tanh

```
Epoch 50: loss: 0.6417028110452065, train accuracy: 100.0, valid accuracy:90.98
```

- Sigmoid

```
Epoch 50: loss: 293.25328133814037, train accuracy: 93.70333333333333, valid accuracy:90.67
```

- ELU

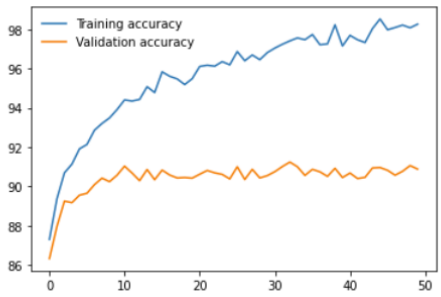
```
Epoch 50: loss: 25.264321280082566, train accuracy: 99.605, valid accuracy:90.62
```

Giving a low value to the learning rates means that the training will be more reliable, but optimization will take a lot of time because steps towards the minimum of the loss function are tiny. On the other hand, having a high learning rate will make the training diverge or may not converge. Weight changes can be huge that the optimizer overshoots the minimum and makes the loss worse.

This is happening also in our case, for learning rates 0.001 and 0.1 we receive some really good results, for 0.001 the model starts with a bigger loss and lower accuracy and keeps improving, needs more time to train and gives slightly less good results than 0.1, while for 0.1 the starting point is better and keeps improving to a certain point. For higher learning rates, 0.5, we don't get as good results especially after epochs=20, the accuracy drops dramatically and overall the loss is not decreasing as much as it did with the previous learning rates. For 1 and 10, the gradient descent of the error function is not converging. The learning rate is too high so it oversteps a local minimum and therefore never hits convergence criteria. They give quite enormous loss which is not improved over the 50 epochs.

(d)

```
<matplotlib.legend.Legend at 0x7fd2159e4128>
```




```
Epoch 50: loss: 101.53255659376373, train accuracy: 98.26833333333333, valid accuracy:90.88
```

Dropout helps to reduce overfitting and generalization error. It can be seen that adding a dropout layer with rate 0.3, the valid accuracy is slightly better and in general the model works smoother. Also, it tries to prevent overfitting.

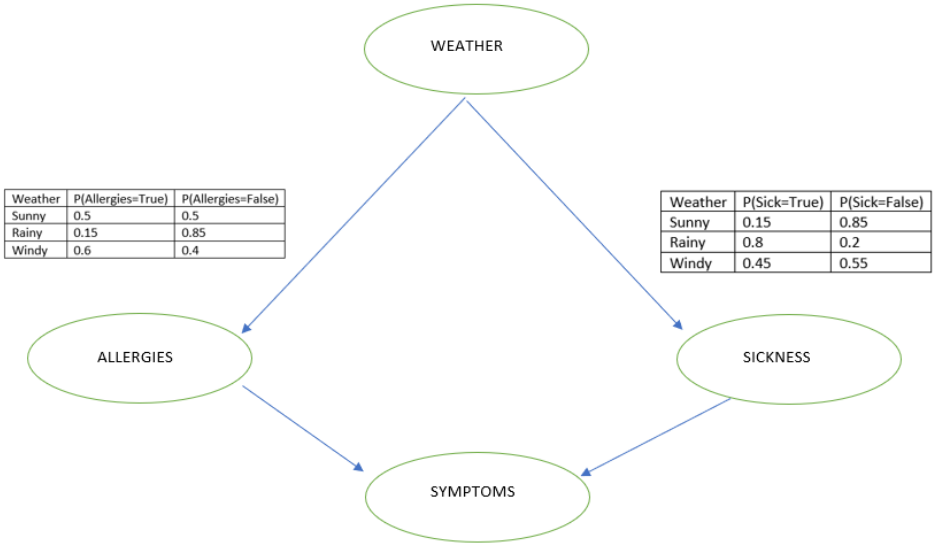
When using a high dropout rate the model cannot be trained, it gives a high loss which is not decreasing and the accuracy is really bad.

On the other hand, using a small dropout rate reduces the loss function and gives a quite satisfying accuracy almost the same as the original, I think that's mainly because our model is more complex, and giving a really small rate does not make any difference.

Question 3: All about the weather

(a),(b)

Weather	P(W=True)	P(W=False)
Sunny	0.6	0.4
Rainy	0.25	0.75
Windy	0.15	0.85



Weather	P(Allergies=True)	P(Allergies=False)
Sunny	0.5	0.5
Rainy	0.15	0.85
Windy	0.6	0.4

Weather	P(Sick=True)	P(Sick=False)
Sunny	0.15	0.85
Rainy	0.8	0.2
Windy	0.45	0.55

Aller	*Sick*	$\mathbb{P}(S = headache)$	$\mathbb{P}(S = runny)$	$\mathbb{P}(S = stom)$
true	true	0.6	0.3	0.1
true	false	0.75	0.2	0.05
false	true	0.1	0.5	0.4
false	false	0.15	0.05	0.8

(c)

P(S = headache, Aller = f|W = sunny) =

= P(headache|~Aller, Sick) *P(~Aller|Sunny) *P(Sick|Sunny) + P(headache|~Aller, ~Sick) *

*P(~Aller|Sunny) *P(~Sick|Sunny) =

= 0.1*0.5*0.15 + 0.15*0.5*0.85

= 0.07125

(d)

P(S = stom|W = rainy) = P(S = stom,W = rainy)/ P(Rainy)

P(S = stom,W = rainy) = P(S = stom|Aller,Sick)* P(Aller |Rainy)* P(Sick |Rainy)* P(Rainy) +

+ P(S = stom|~Aller,~Sick)* P(~Aller |Rainy)* P(~Sick |Rainy)* P(Rainy) +

+ P(S = stom|~Aller,Sick)* P(~Aller |Rainy)* P(Sick |Rainy)* P(Rainy) +

+ + P(S = stom|Aller,~Sick)* P(Aller |Rainy)* P(~Sick |Rainy)* P(Rainy)

= 0.1*0.15*0.8*0.25 + 0.8*0.85*0.2*0.25 + 0.4*0.85*0.8*0.25+ 0.05*0.15*0.2*0.25

=0.105375

P(S = stom|W = rainy) = 0.037/0.25 = 0.4215

References

https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/

https://towardsdatascience.com/build-a-fashion-mnist-cnn-pytorch-style-efb297e22582