Artificial In CW2 Report

Anatol Satler
ec24713

1.1

The optimal neural network architecture that I have found is the following:

```
Evaluation on validation set complete. Accuracy: 0.5014 (180/359 correct)
choose hyperparameters: population_size = 18, num_generations = 12, num_parents = 5, BATCH_SIZE = 64,
LR = 0.001, EPOCHS = 20

{'neurons': 128, 'activations': 'gelu', 'dropout': 0.2}
{'neurons': 8, 'activations': 'tanh', 'dropout': 0.1}
{'neurons': 16, 'activations': 'gelu', 'dropout': None}
{'neurons': 128, 'activations': 'tanh', 'dropout': 0.1}
```

1.2

For the **selection** method, the population is evaluated by their fitness scores, and the top-performing parents are selected using heapq.nlargest. This method is more efficient than sorting the entire population because it uses a heap data structure to directly retrieve the largest elements, reducing the time complexity from O(n log n) to O(n log k), where k is the number of top performers we wish to select. This ensures that only higher-performing individuals are chosen, while still maintaining some diversity in the gene pool. The **crossover** method uses a single-point strategy. This means that we create a random crossover point which is within the genome lengths. The genomes are randomly selected from two parents. The child genome is constructed by combining the genes from the first parent, up to the crossover point, and the second part, from the crossover point onward. This gives the opportunity for recombination of features from different architectures e.g. activation function from the one parent with the amount of dropout probability. **Mutation** maintains the diversity and explores new architectures to ensure that does not converge prematurely. A random layer is being picked from that random layer 1 of the 3 attributes is being modified e.g. neurons are being replaced from 4 -> 8, activations Sigmoid -> ReLu, dropout None -> 0.1.

1.3

At the 7th iteration, we can see high similarity to my optimal solution architecture. The first and last layers are very high while the mid layers tend to be small. The activation functions do not have the same pattern as the optimal one. From 5 out of 10 models fail to gain above 35% of accuracy. In no. 1and 4, we can we see a pattern of the neurons that the model tends to be better with a high neuron amount in the first 3 layers.

No.	Accuracy	Architecture
1	Evaluation on validation set complete. Accuracy: 0.3928 (141/359 correct)	{'neurons': 128, 'activations': 'elu', 'dropout': None} {'neurons': 32, 'activations': 'sigmoid', 'dropout': None} {'neurons': 64, 'activations': 'gelu', 'dropout': 0.2} {'neurons': 16, 'activations': 'leaky_relu', 'dropout': 0.2}
2	Evaluation on validation set complete. Accuracy: 0.3788 (136/359 correct)	{'neurons': 16, 'activations': 'leaky_relu', 'dropout': None} {'neurons': 64, 'activations': 'sigmoid', 'dropout': 0.35} {'neurons': 4, 'activations': 'leaky_relu', 'dropout': 0.2} {'neurons': 32, 'activations': 'tanh', 'dropout': 0.35}
3	Evaluation on validation set complete. Accuracy: 0.3231 (116/359 correct)	{'neurons': 16, 'activations': 'relu', 'dropout': None} {'neurons': 16, 'activations': 'elu', 'dropout': 0.35} {'neurons': 32, 'activations': 'elu', 'dropout': 0.35} {'neurons': 4, 'activations': 'gelu', 'dropout': 0.1}
4	Evaluation on validation set complete. Accuracy: 0.4178 (150/359 correct)	{'neurons': 64, 'activations': 'sigmoid', 'dropout': 0.2} {'neurons': 32, 'activations': 'elu', 'dropout': 0.1} {'neurons': 32, 'activations': 'leaky_relu', 'dropout': 0.2} {'neurons': 4, 'activations': 'relu', 'dropout': 0.1}
5	Evaluation on validation set complete. Accuracy: 0.2953 (106/359 correct)	{'neurons': 4, 'activations': 'leaky_relu', 'dropout': 0.2} {'neurons': 4, 'activations': 'sigmoid', 'dropout': 0.2} {'neurons': 128, 'activations': 'leaky_relu', 'dropout': 0.2} {'neurons': 64, 'activations': 'gelu', 'dropout': None}

6	Evaluation on validation set complete. Accuracy: 0.2953 (106/359 correct)	{'neurons': 32, 'activations': 'sigmoid', 'dropout': 0.2} {'neurons': 16, 'activations': 'elu', 'dropout': 0.2} {'neurons': 8, 'activations': 'leaky_relu', 'dropout': 0.1} {'neurons': 16, 'activations': 'gelu', 'dropout': 0.35}
7	Evaluation on validation set complete. Accuracy: 0.4763 (171/359 correct)	{'neurons': 128, 'activations': 'relu', 'dropout': 0.1} {'neurons': 32, 'activations': 'elu', 'dropout': 0.1} {'neurons': 8, 'activations': 'tanh', 'dropout': 0.35} {'neurons': 128, 'activations': 'elu', 'dropout': None}
8	Evaluation on validation set complete. Accuracy: 0.3565 (128/359 correct)	{'neurons': 4, 'activations': 'gelu', 'dropout': 0.35} {'neurons': 16, 'activations': 'elu', 'dropout': None} {'neurons': 64, 'activations': 'elu', 'dropout': 0.1} {'neurons': 32, 'activations': 'leaky_relu', 'dropout': 0.1}
9	Evaluation on validation set complete. Accuracy: 0.2702 (97/359 correct)	{'neurons': 16, 'activations': 'tanh', 'dropout': 0.1} {'neurons': 16, 'activations': 'elu', 'dropout': None} {'neurons': 32, 'activations': 'relu', 'dropout': None} {'neurons': 8, 'activations': 'sigmoid', 'dropout': 0.1}
10	Evaluation on validation set complete. Accuracy: 0.3008 (108/359 correct)	{'neurons': 8, 'activations': 'elu', 'dropout': 0.2} {'neurons': 64, 'activations': 'gelu', 'dropout': 0.1} {'neurons': 4, 'activations': 'relu', 'dropout': 0.35} {'neurons': 4, 'activations': 'tanh', 'dropout': 0.1}

1.4.

I wanted to observe if the number of generations increase the accuracy also always increases or not. I was assuming because of the iterations the model is increasing and therefore has more “time” to optimize. I am assuming the more generation; we will add the more stable it becomes around the +0.30 area or maybe even higher.
The amount of population was 18 and parents = 5.

No.	Accuracy	Architecture	Generation
1	Evaluation on validation set complete. Accuracy: 0.2730 (98/359 correct)	{'neurons': 16, 'activations': 'tanh', 'dropout': 0.35} {'neurons': 8, 'activations': 'tanh', 'dropout': 0.2} {'neurons': 32, 'activations': 'gelu', 'dropout': 0.2} {'neurons': 16, 'activations': 'relu', 'dropout': 0.35}	4
2	Evaluation on validation set complete. Accuracy: 0.3621 (130/359 correct)	{'neurons': 128, 'activations': 'sigmoid', 'dropout': 0.1} {'neurons': 128, 'activations': 'relu', 'dropout': 0.2} {'neurons': 16, 'activations': 'gelu', 'dropout': 0.1} {'neurons': 4, 'activations': 'relu', 'dropout': 0.1}	6
3	Evaluation on validation set complete. Accuracy: 0.2006 (72/359 correct)	{'neurons': 4, 'activations': 'sigmoid', 'dropout': 0.35} {'neurons': 4, 'activations': 'sigmoid', 'dropout': 0.35} {'neurons': 128, 'activations': 'tanh', 'dropout': None} {'neurons': 32, 'activations': 'relu', 'dropout': None}	8
4	Evaluation on validation set complete. Accuracy: 0.3593 (129/359 correct)	{'neurons': 64, 'activations': 'relu', 'dropout': 0.1} {'neurons': 4, 'activations': 'gelu', 'dropout': 0.35} {'neurons': 32, 'activations': 'elu', 'dropout': 0.35} {'neurons': 8, 'activations': 'elu', 'dropout': 0.2}	10
5	Evaluation on validation set complete. Accuracy: 0.3120 (112/359 correct)	{'neurons': 16, 'activations': 'tanh', 'dropout': 0.35} {'neurons': 16, 'activations': 'relu', 'dropout': 0.1} {'neurons': 4, 'activations': 'tanh', 'dropout': 0.2} {'neurons': 4, 'activations': 'sigmoid', 'dropout': 0.2}	12

2.1

The Cross-Entropy is the most appropriate loss function because of the probabilistic interpretation and effective gradient behaviour that guides the optimization process effectively.
Variable description:
N = number of samples
C = number of classes
y_i = true class label
p head at class j for the i-th = predicted probability for classes
s for sample i, class j = raw class scores **before** softmax
Formula:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{i,j}\log\left(\hat{p}_{\mathrm{i,j}}\right)$$

Predicted Probability for class j is:

$$\hat{p}_{\mathrm{i,j}} = \frac{e^{s_{i,j}}}{\sum_{k=1}^{C} e^{s_{i,k}}}$$

This penalises predictions with low probabilities to the true class

The raw scores are being passes through the softmax function before computing the loss.
Formula:

$$\frac{\partial \mathrm{L}}{\partial s_{\mathrm{i,j}}} = \hat{p}_{i,j} - y_{i,j}$$

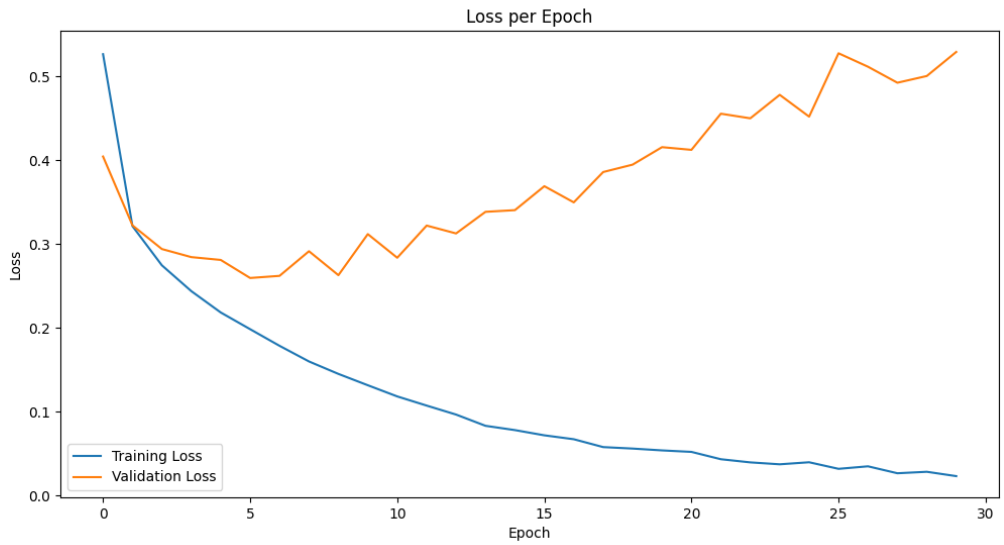
For all y that are true classes $y_{i,j} = 1$, the gradient is $\hat{p}_{i,j} - 1$. It encourages the lower loss in the predicting state a high probability for the true class. For the opposite, for all classes that are $y_{i,j} = 1$, the gradient is just $\hat{p}_{i,j}$. It penalises the model for assigning high probability to incorrect classes.

Interpretation of Mathematical properties:

To ensure that the probabilities for all classes sum to one for each sample, we use softmax function which transforms raw logits into a probability distribution. Cross-entropy loss quantifies the distance between the predicted probability distribution and the true class distribution. The gradient of the loss with respect to the logits is directly related to the error in the predicted probability. It is calculated as the difference between the predicted probability and the true label. The gradient is small when the predicted probability for the correct class is near 1, leading to minor adjustments. For the predicted probability that shifts significantly from 1, the gradient becomes larger, leads the model to make more substantial corrections. This encourages the model to assign higher probabilities to the correct class while minimizing the probabilities for incorrect ones. Therefore, the gradient naturally scales with the confidence of the model, by that I mean, high-confidence or correct predictions lead to smaller updates, while incorrect prediction result in larger adjustments.

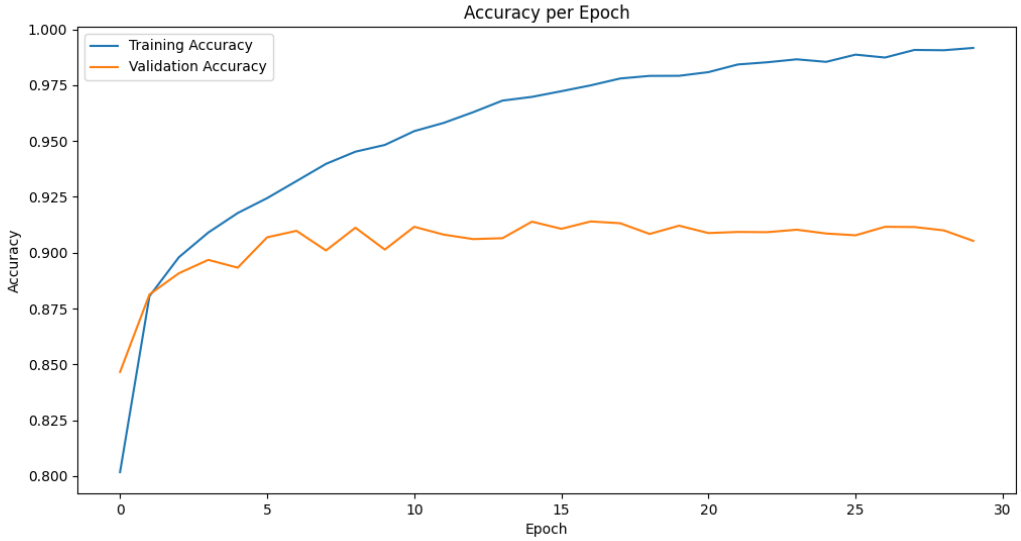
2.2

Final Training Accuracy: 0.9917
Final Validation Accuracy: 0.9053



Throughout the training process, the training loss and validation loss display distinct behaviors, reflecting the model's ability to fit the training data and generalize to unseen data. Initially, the training loss decreases steadily, indicating that the model is learning and improving its ability to fit the training data. By around 15 epochs, the training loss stabilizes near 0.1, signaling that the model has largely converged and no longer makes significant improvements in fitting the training set.

In contrast, the validation loss remains relatively constant at around 0.3 until approximately 10 epochs. During this phase, the model struggles to generalize to the validation set, suggesting that while it performs well on the training data, it has difficulty applying this learning to unseen examples. After 10 epochs, the validation loss begins to increase, signaling the onset of overfitting. This increase in validation loss suggests that the model is fitting the training data more closely, but this results in poorer performance on the validation set. As a result, the model's ability to generalize diminishes, leading to higher loss on the validation data.



The accuracy metrics for both training and validation show how well the model performs in terms of correct predictions during training. The training accuracy improves rapidly during the early epochs as the model learns basic features and patterns in the data. By around 15 epochs, the training accuracy reaches approximately 0.975 and continues to edge closer to 1.0, signaling that the model is performing excellently on the training set. However, the rate of improvement slows as the model approaches its optimal state, with further increases becoming harder to achieve.
On the other hand, the validation accuracy follows a more complex pattern. Between 5 to 10 epochs, validation accuracy fluctuates as the model tries to improve, but it eventually becomes stuck around 0.9, failing to make significant gains beyond this point. Despite some attempts to improve, the models performance on the validation set remains still, suggesting that the model is not generalizing well beyond the training data. This stagnation in validation accuracy, combined with the increasing validation loss, indicates potential overfitting.

3.1

I left out the precondition being clean out, because we are assuming the things are already "clean".
I will be notating each action, precondition and effects of the associated action with numbers e.g. 1) etc.

1. **Take whole Apple:** The preconditions must meet that 1) the apple isn't already in the hand and is available to grab, 2) apple is not peeled, 3) the apple isn't cut either. 4) The apple is now not available to be picked up and 5) is currently in our hand.

2. **Peel Apple:** For peeling we need a knife. Therefore, 1) we take the knife where we need to assure that 2) the knife must be available. We repeat that with 3) the apple, to take it into our other hand. Now we just checking if 4) the apple hasn't been already peeled.5) Now, we can peel the apple, so it is peeled.

3. **Cut Apple:** To cut the apple we need a knife. Therefore, 1) we take a knife in our hand. 2) It needs to available, as well 3) the apple must be available and 4) picked up into our other hand. Now we need to 5) peel the apple, before 6) cutting. 7) The apple is now cut into pieces.

4. **Put down Knife:** The knife must be in 1) our hand already to be put down. After the knife was put down, 2) it is available again and 3) it is not in our hand anymore.

5. **Add chopped apple to container:** For chopping the apple we firstly need to perform every action from **3.** (To cut the apple we need a knife. Therefore, 1) we take a knife in our hand. 2) It needs to available, as well

3) the apple must be available and 4) picked up into our other hand. Now we need to 5) peel the apple, before 6) cutting. 7) The apple is now cut into pieces.). Our apple is chopped now. Firstly, we put the knife away, repeating the actions of **4.** (The knife must be in 1) our hand already to be put down. After the knife was put down, 2) it is available again and 3) it is not in our hand anymore.). We can now add the pieces into the container if these preconditions are met: 1) we have our apple in our hand and 2) the container is empty. After these conditions are met, we put the pieces into the container. The container 3) becomes not empty, 4) our hand is now free again and 5) container we chose has the apple pieces stored.

3.2

1. Take:
   - Available(?x)
2. Put_down:
   - in_hand(?x)
3. Peel:
   - is_clean(?x) /\ ¬is _peeled(?x) /\ in_hand(?x) in_hand(?y)
4. Cut:
   - is_peeled(?x) /\ ¬is_cut(?x) /\ in_hand(?x) /\ in_hand(?y)
5. Add_to:
   - in_hand(?x) /\ empty(?y)

3.3

FOL
(KnifeAvailable(k)∧AppleAvailable(a)∧Unpeeled(a)) → Peeled(a)
CNF
(¬KnifeAvailable(k) ∨ ¬AppleAvailable(a) ∨ ¬Unpeeled(a) ∨ Peeled(a))

FOL
(KnifeAvailable(k) ∧ AppleAvailable(a) ∧ Peeled(a)) → Cut(a)
CNF
(¬KnifeAvailable(k) ∨ ¬AppleAvailable(a) ∨ ¬Peeled(a) ∨ Cut(a))

3.4

FOL Definitions:

1. **Clean the apple:**
   - Available(a): The apple is available.
2. **Peel the apple:**
   - Peeled(a): The apple a is peeled.
3. **Cut the apple:**
   - Cut(a): The apple a is cut.
   - Peeled(a): The apple a is peeled.
4. **Place in container:**
   - InCointer(a,c): The apple a is placed in the container c.
   - Cut(a): the apple a is cut
   - Available(c): The container c is available.
5. **Serve the salad:**
   - Served(a): The apple a is served

Transition of pre-conditions to effects:

Initial conditions are G(Available(a) /\ Available(c)), the apple a and container c are available

1. G(Available(a))
2. G(Available(a) -> X(Peeled(a)))
3. G(Available(a) -> X(Cut(a)))
4. G(Cut(a) -> X(InContainer(a,c)))
5. G(InContainer(a,c) -> X(Serverd(a)))

Logical Sequence expression:

$$G\,(\text{Available}(a) \wedge \text{Available}(c)) \wedge$$
$$(\text{Available}(a) \rightarrow X\,(\text{Peeled}(a))) \wedge$$
$$(\text{Peeled}(a) \rightarrow X\,(\text{Cut}(a))) \wedge$$
$$(\text{Cut}(a) \rightarrow X\,(\text{InContainer}(a, c))) \wedge$$
$$(\text{InContainer}(a, c) \rightarrow X(\text{Served}(a)))$$