Mini Project 3

Mehmet Ozan Baykan — 21901660 — EEE4443

December 2023

1 Introduction

In this project, we will implement a Multi-Layer Perceptron (MLP) model with several regularization features to solve the Human Activity Recognition (HAR) problem. The dataset contains samples $x \in \mathbb{R}^{561}$, where every dimension represents a statistic of the time-series data captured from movement sensors. The training data X_{train} contains 7352 samples and test data contains 2947 data. For model selection, we randomly select 10% of the training data as the validation set, and evaluate the models with respect to the performance on the validation set.

2 Methods

2.1 Network Architecture

All models share the same architecture: an MLP with two hidden layers, ReLU activation on the first and second layers, and softmax activation at the output layer. The equations that define the network are as follows:

$$\begin{aligned}
Z^{[i]} &= W^{[i]}A^{[i-1]} + b^{[i]} && i \in \{1,2\} \\
A^{[i]} &= ReLU(Z^{[i]}) && i \in \{1,2\} \\
Z^{[3]} &= W^{[3]}A^{[2]} + b^{[2]} \\
A^{[3]} &= softmax(Z^{[3]})
\end{aligned}$$

where:

- $A^{[0]} = X \in \mathbb{R}^{561xm}$ is the input, m= batch size
- $ReLU(A)_{ij} = max\{A_{ij}, 0\}, A \in \mathbb{R}^{Dxm}$, D = dimension of the layer.
- $softmax(A)_i = \frac{e^{A_i}}{\sum_{k=1}^{N} e^{A_k}}, A \in \mathbb{R}^N$, N $\in \mathbb{N} W^{[i]} \in \mathbb{R}^{D_i x D_{i-1}}$ is the weight matrix of the linear transformation between layer i-1 and layer i.
- $b^{[i]} \in \mathbb{R}^{D_i}$ is the bias vector of the linear transformation between layer i-1 and layer i.

This is implemented by the NN2.forwardprop() function inside the class NN2.

The cost function is the categorical cross-entropy loss:

$$\mathrm{E}(Y, Y_{\text{pred}}) = -\sum_{i=1}^{6} Y_i \ln(Y_{\text{pred}})_i$$

where:

- Y: the ground truth at time step, with shape (batch size, output dimension) $= (6, m)$.
- Y_{pred}: the model's prediction, with shape (batch size, output dimension) $= (6, m)$.
- Y_i: the ground truth for category i
- Y_{pred_i}: the predicted probability for category i.

Notice that the only term that contributes to the cost for each sample is $\log([Y_{\text{pred}}]_i)$ if $Y_i = 1, i \in \{1...6\}$. Hence, only the predicted probability for the true category is penalized, and only if $[Y_{\text{pred}}]_i \neq 1$. Since the output of the softmax is a probability distribution, it makes sense to penalize only the cases where $[Y_{\text{pred}}]_i \neq 1$, since the true distribution is attained if and only if $[Y_{\text{pred}}]_i = 1$.

2.2 Cost Function

The cost function is implemented by the NN2.CCEcost() function of the NN class.

The derivatives of the network parameters with respect to the categorical cross-entropy loss function are given as follows:

$$\begin{aligned}
\frac{\partial E}{\partial Z^{[3]}} &= (A\hat{[}3] - Y) \\
\frac{\partial E}{\partial Z^{[i]}} &= (W^{[i+1]T}\frac{\partial E}{\partial Z^{[i+1]}}) \cdot \mathbb{1}_{\{ReLU(Z^{[i]})>0\}} && i \in \{1, 2\} \\
\frac{\partial E}{\partial W^{[i]}} &= \frac{1}{m} \cdot \frac{\partial E}{\partial Z^{[i]}} A^{[i]T} && i \in \{1, 2, 3\} \\
\frac{\partial E}{\partial b^{[i]}} &= \frac{1}{m} \cdot \sum_{n=1}^{m} (\frac{\partial E}{\partial Z^{[i]}})^{(m)} && i \in \{1, 2, 3\}
\end{aligned}$$

where the notation is the same as in the forward propagation equations except:

- $\mathbb{1}_{\{ReLU(Z^{[i]})>0\}}$ is the characteristic function applied elementwise to the matrix $Z^{[i]}$, such that $[\mathbb{1}_{\{ReLU(Z^{[i]})>0\}}]_{n,m} = \mathbb{1}_{\{ReLU([Z^{[i]}]_{n,m})>0\}}$

The backward propagation is implemented by the NN2.backprop() function.

2.3 Weight Updates

The weight updates are done with momentum. The momentum method is based on the idea that using a decaying average of previous updates guides the optimization process more robustly to minima. By using the momentum parameter β, at each batch, the update is added to the momentum term by using a moving average. So at every batch, we first update the momentum terms $V_{W^{[i]}} \in \mathbb{R}^{D_i x D_{i-1}}$, $V_{b^{[i]}} \in \mathbb{R}^{D_i}$

$$V_{W^{[i]}} \leftarrow \beta V_{W^{[i]}} + (1-\beta)\frac{\partial E}{\partial W^{[i]}}$$
$$V_{b^{[i]}} \leftarrow \beta V_{b^{[i]}} + (1-\beta)\frac{\partial E}{\partial b^{[i]}}$$

Then the parameter updates are as follows:

$$W^{[i]} \leftarrow W^{[i]} - \eta V_W^{[i]}$$
$$b^{[i]} \leftarrow b^{[i]} - \eta V_b^{[i]}$$

Of course, with $\beta = 0$, this is just the standard gradient-descent update rule. This is implemented by introducing conditional branching inside the network functions, in order only to reduce the computational load in the case $\beta = 0$.

2.4 Weight Initialization

For weight initialization we follow:

$$W \in \mathbb{R}^{NxM} \sim \mathcal{N}(0, \sqrt{\frac{2}{N+M}})$$
$$b \in \mathbb{R}^{N} = 0$$

which is a modified version of Xavier initialization.

2.5 Early Stopping

Early stopping is a regularization technique used during model training, whereby the training process is terminated if a given metric does not improve over some fixed number of steps s. For this, we randomly select 10% of the training set as the validation set, and terminate the learning process if the validation loss has not improved over the last s steps. This is implemented by the EarlyStopping() class and using appropriate conditional statements in the NN2.train() function.

2.6 Dropout

Dropout is a regularization technique whereby, given a binomial probability $p \in [0, 1]$, every unit in some given layer is excluded during training with probability p. This corresponds to setting any element $A_i^{[t]}$ of the vector representing the activation $A^{[t]} \in \mathbb{R}^D$. Since the choice of which units to drop out should be independent for every sample, we implement this technique by using the NN2.dropout() function with appropriate conditional statements in NN2.train(). Of course, the dropout should not be done at testing, since it only helps to regularize the estimation of the weights. Also, because we train the whole network to function together, randomly dropping out units harms the interpretability of the test results

3 Results

3.1 Part A

As suggested in the exercise, we first train 16 models, each for a combination of the following hyperparameters:

- $N_1, N_2 \in (300, 200), (300, 100)$ the dimensions of the first and second hidden layers.
- $\eta \in 0.01, 0.001$, the learning rate
- $\beta \in 0, 0.99$, the momentum rate
- $m \in 1, 50$, the batch size

As suggested, all models are trained with early stopping, with the chosen parameter of = 10. In this part, none of the models employ dropout, so all models are trained as fully connected networks. The metrics are given in Figure 1 and 2.

First, an important observation to make is the effects of learning rate, momentum rate and batch size on the loss curves. Notice that, in general, as learning rate is increased, the curve becomes steeper; as momentum rate is increased, the curve becomes less steep; and as batch size is increased, the curve becomes less steep.

Regarding the learning rate, higher learning rates causes each update to be greater, hence, the training becomes more vulnerable to changes in batch samples, hence more chaotic, but overall the network progresses to local minima faster. Luckily, in none of the networks, the high learning rate causes the network to jump out of local optima, as was the case with the networks in Mini Project 2. With respect to the accuracies, we observe that, expectedly, the accuracies follow the inverse trend in the loss, hence, higher learning rates in our models produce faster learning behavior.

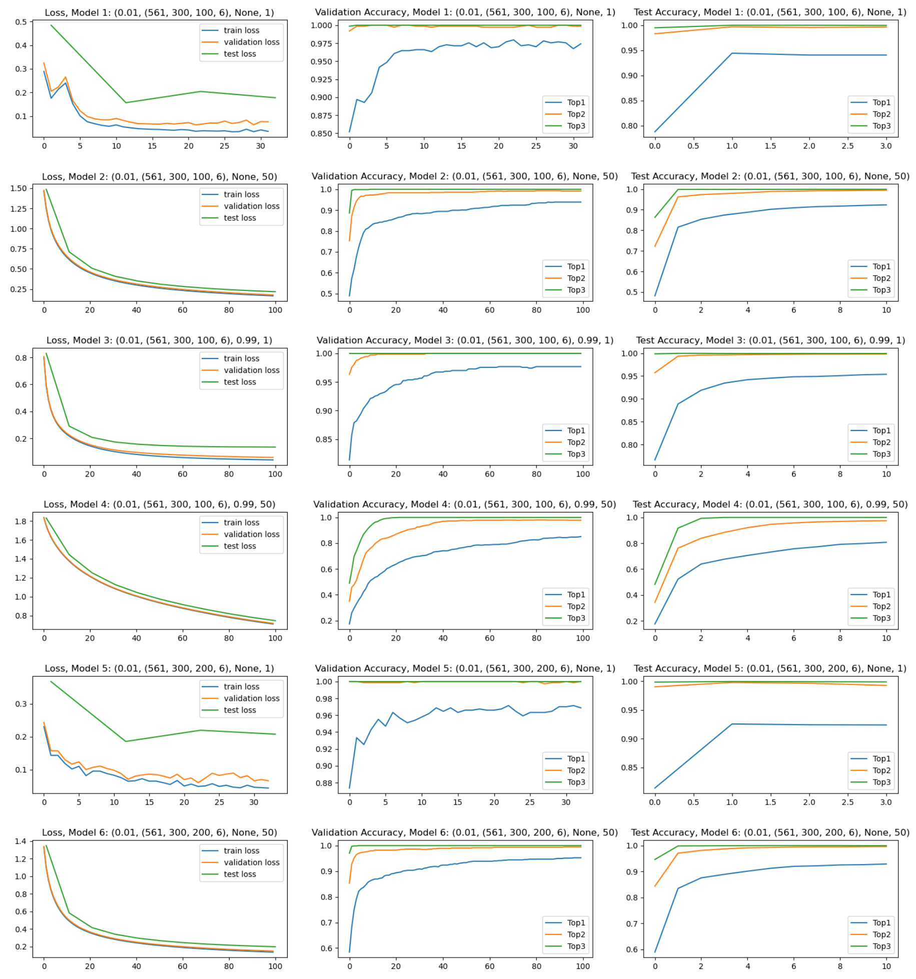

Figure 1: Metrics for models with the given parameter configurations $[\eta, [D_x, N_1, N_2, D_y], \beta, batchsize]$. The first column is the loss, where test loss is calculated only at every 10 epochs, the second column is the validation accuracies, and the third column is test accuracies.

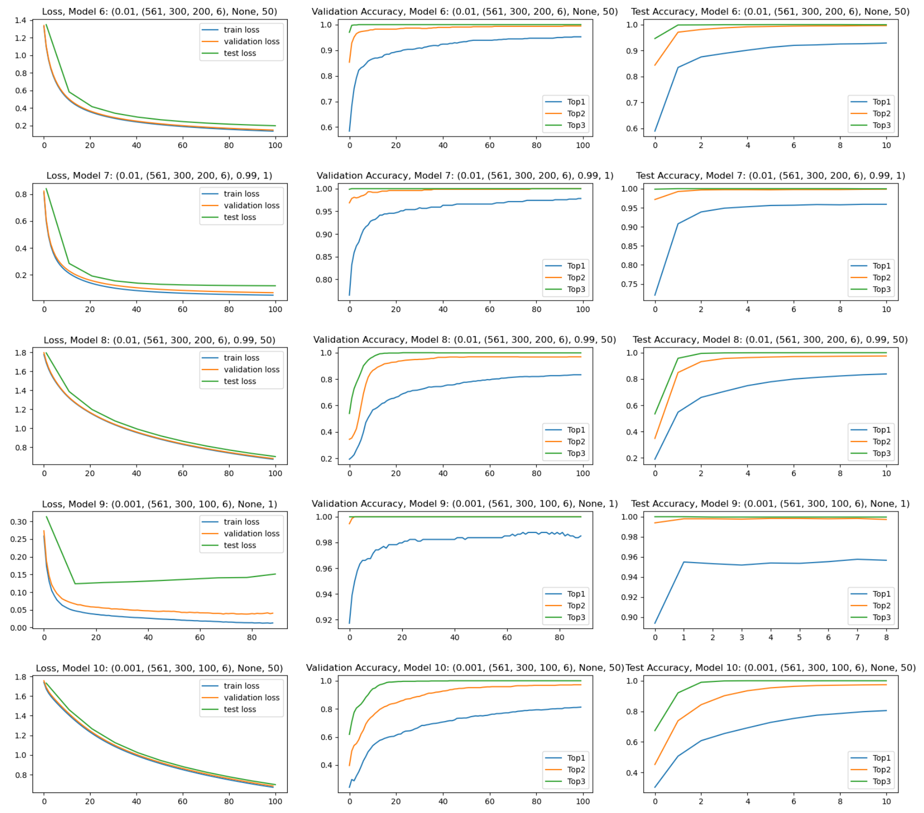

Figure 2: Metrics for models with the given parameter configurations $[\eta, [D_x, N_1, N_2, D_y], \beta, batchsize]$. The first column is the loss, where test loss is calculated only at every 10 epochs, the second column is the validation accuracies, and the third column is test accuracies.

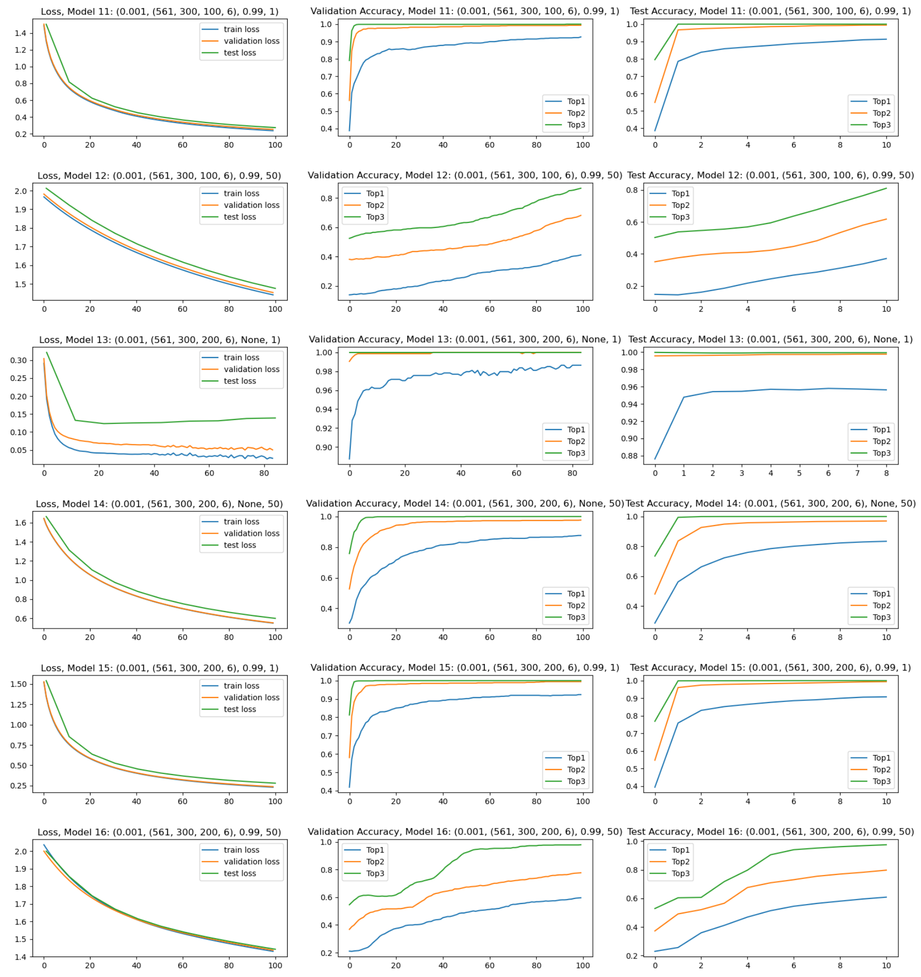

Figure 3: Metrics for models with the given parameter configurations $[\eta, [D_x, N_1, N_2, D_y], \beta, batchsize]$. The first column is the loss, where test loss is calculated only at every 10 epochs, the second column is the validation accuracies, and the third column is test accuracies.

Regarding the momentum rate, we can safely claim that momentum, as expected, dilates and smooths the loss curve. Due to the time-averaging, the loss curve stably approaches the asymptotic limit reached in some of the models, e.g 3 and 7, but is so slow that it fails to reach the asymptotic limit, hence the minima in the parameter space, over 100 epochs. Hence, we note that while using momentum, care should be taken to adjust other parameters controlling the rate of optimization to ensure that stability and duration of the training are appropriately traded off. These effects are reflected in the accuracy results in the same manner as with the learning rate.

Regarding the batch size, although the individual updates in the training loop might be more chaotic, as the loss is evaluated after every epoch, the loss curves look smoother and less steep. However, the final accuracies of the online-trained models are higher overall, and especially in models with low learning rate and momentum, mini-batch learning significantly slows down the training process, e.g Model 12. However, online-learning itself does not guarantee good convergence, as we observe in Model 6, where although the loss curve displays asymptotic behavior, the Top1 accuracy is bounded at 0.6, implying that the model most likely has got stuck in bad local optima.

Regarding N_2, the size of the second hidden layer, we observe that it does not have any significant effect on the results, as models 3, 7, 9, 13 perform the best with respect o test accuracy, and some of them have $N_2 = 100$ whereas some has $N_2 = 200$.

In Figure 4 are the confusion matrices of the model predictions on the test set.

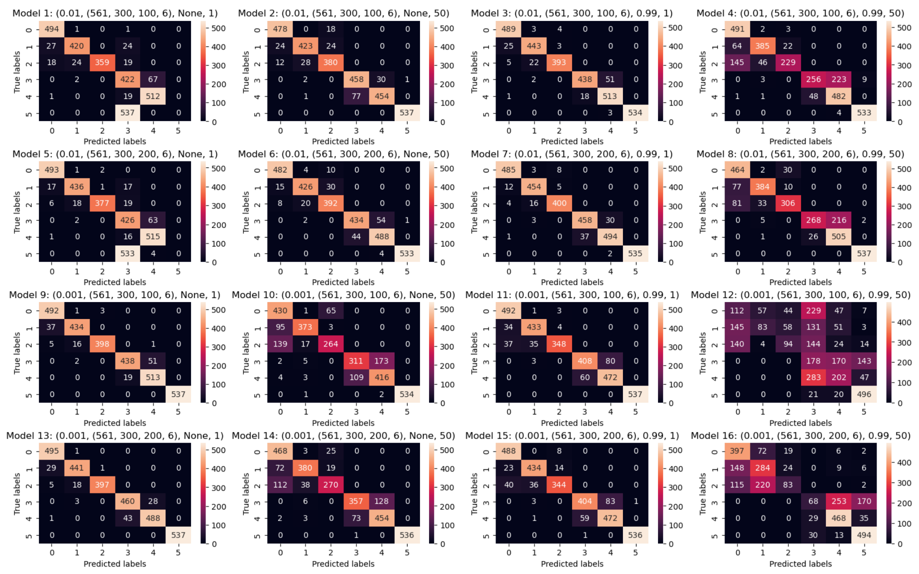

Figure 4: Confusion matrices of the models in part(a) on the test set.

Observe that in most of the confusion matrices, the results are expected given the metrics. However, there are some curious cases.

Model 1 seems to map Class 6 (laying) to Class 4 (standing), which is an error we unfortunately cannot give further insight than to say that the network may have got stuck in local optima or might have overfitted to the variation in the Class 6 data, hence is now unable to generalize its learning on Class 6 and ends up predicting the only other static and non-standing action, Class 4, since the data is represented in statics on the date rather than physical patterns. Note that Model 5 also displays a similar behavior.

Model 12, as expected, failed to converge to a local optimum, displaying a very scattered confusion matrix.

Similarly, Model 16 has also failed to converge to any optima, but we can still see that it began to distinguish between static and dynamic actions, as the predictions have concentrated over the top left and bottom right corners.

3.2 Part B

In this part, we select the best-performing hyperparameter configuration out of the 16 configurations in part A, and train it using dropout on the second layer.

To select the best model, we considered the models with the highest test accuracy and the lowest difference between train and test accuracy, therefore we selected Model 7. Moreover, just out of curiosity, we also trained two other models. The parameters of the models in this section are as follows:

- Model 1: $[\eta, [D_x, N_1, N_2, D_y], \beta, batchsize, p] = [0.01, [561, 300, 200, 6], 0.99, 1, 0.5]$.
- Model 2: $[\eta, [D_x, N_1, N_2, D_y], \beta, batchsize, p] = [0.01, [561, 300, 200, 6], 0, 50, 0.5]$.
- Model 3: $[\eta, [D_x, N_1, N_2, D_y], \beta, batchsize, p] = [0.01, [561, 300, 200, 6], 0.99, 50, 0.5]$.

The training and test metrics are given in Figure 5.

As expected, we observe that the first configuration - identical to Model 7 except for dropout - is the best performing. Again, we see that the combination of momentum with batch learning has produced a very slow learning process, displayed in the loss and accuracy plots of Model 3. As a consequence, this model has not been able to converge to any optima during 100 epochs. In contrast, we see that without momentum and with batch learning, Model 2 seems to converge better and faster. Moreover, expectedly we see that Model 1 in this section performs best, but does not significantly outperform the non-dropout counterpart.

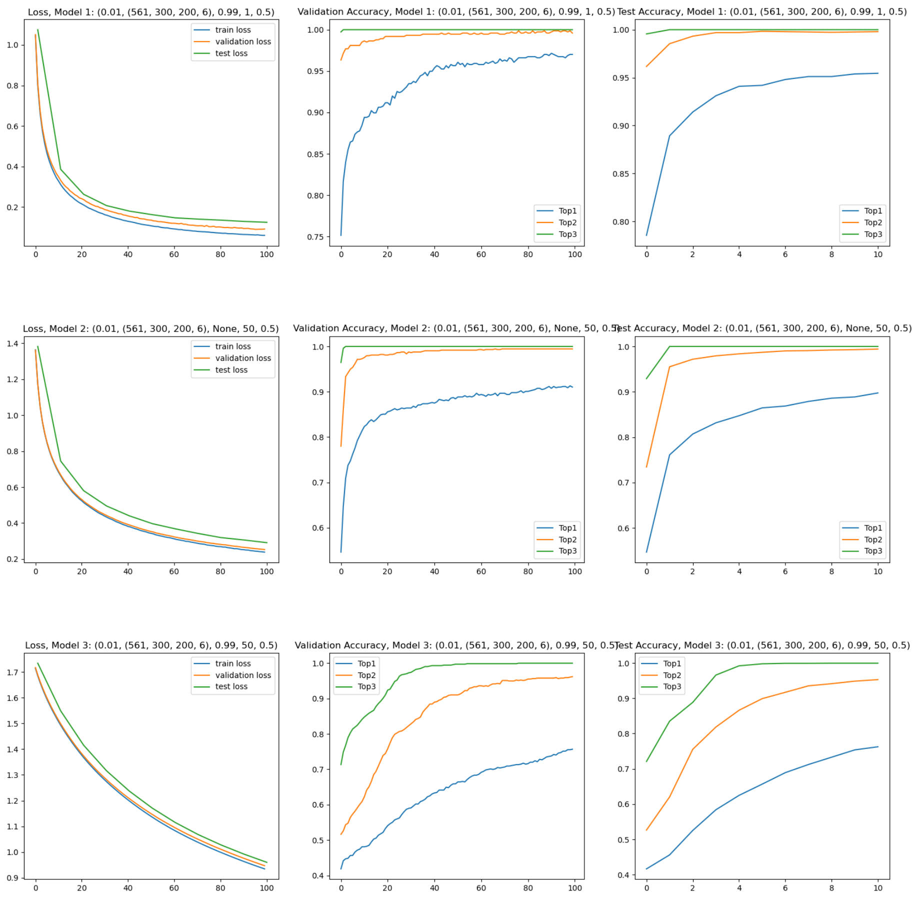

Figure 5: Metrics for models with the given parameter configurations $[\eta, [D_x, N_1, N_2, D_y], \beta, batchsize, p]$. The first column is the loss, where test loss is calculated only at every 10 epochs, the second column is the validation accuracies, and the third column is test accuracies.

It is important here to note that, the model architecture is evidently powerful enough to accomplish the given task with $\sim 95\%$, and does not display any significant tendency to overfit the training data. Hence, we do not see any significant regularizing effect from the dropout, as the model did not have any significant problem with overfitting.

The confusion matrices of the predictions of the models on the test set are given in Figure 6 below:

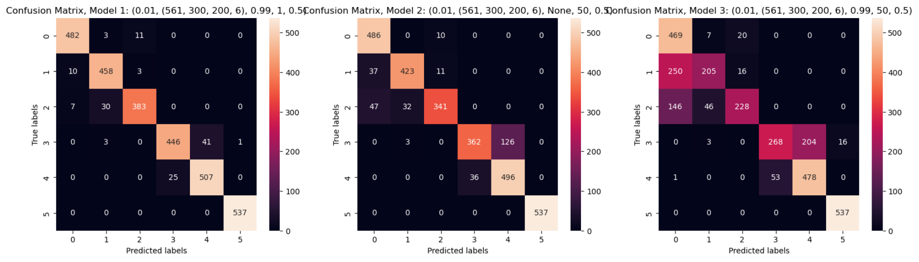

Figure 6: Confusion matrices of the models in part(b) on the test set.

Observe that Model 1 fits the data perfectly, except for some errors between Class 4 (sitting) and Class 5 (standing). Moreover, Model 3 has a scattered confusion matrix that is consistent with its poor learning behavior. In Model 2, we see that the problem in Model 1 is amplified, and the model confuses Class 4 (sitting) for Class 5 (standing) asymmetrically and more frequently.

4 Discussion

First, it is important to observe that overall, this 4-layer MLP architecture is a good match for solving the classification problem at hand. Notice that the given data contains a broad characterization of the statistics of the movement signals and that MLP architectures are known to be powerful tools for approximating probability distributions. Hence, it is plausible to say that the statistical interpretation of the model and the data may provide support in explaining the success of the architecture. In contrast to the time-series data and Recurrent Neural Network (RNN) model used in the previous Mini Project, this problem formulation and model pair perform significantly better and display more stable learning behavior. In this context, we might conjecture that the statistical transformation of the input data has helped represent the measurements in a more information-rich way. It might be that the statistical representation helps to identify global behavior over the measurement period.

In relation to this, we observe that, with appropriate hyperparameters, as in the case with Model 7 in part A, the model does not suffer significantly from overfitting when equipped with momentum and early stopping. Observe that the generalization of the model configuration of Model 7 does not significantly benefit from using dropout.

Interestingly, we see that the increased complexity endowed by increasing the second layer size to 200 does not make a significant difference in model performance, as in the case of Model 3. It is important to note that, although we here chose our model based on the generalization performance, it might also be prudent to select a model with second layer size 100 by favoring a simpler model and faster training.