Bahri Batuhan Bilecen
22201372
15.12.2023

EEE543 Mini Project 3 – MLP on Human Activity Recognition

1. Problem Definition

In this work, we train a 3-layer-perceptron (MLP) classifier on a human activity recognition (HAR) dataset via backpropagation. The first 2 layers of the MLP have ReLU activations, and the output activation is SoftMax.

The dataset consists of 6 classes, each class representing a different human activity. We do not utilize time series data taken from the sensors attached to the subjects directly, but rather utilize the statistical values derived from the aforementioned data, such as mean, variance, skewness, etc.

Some ablation studies regarding the hyperparameters of the network, and different training regimes (change of layer sizes, batch sizes, learning rates, momentum and dropout) are also demonstrated with accompanying conclusions in the following sections.

2. Derivations and Implementation Details

This section focuses on the differences between the first mini project and the second one. Mainly, the change to the SoftMax layer, momentum, and dropout will be explained.

The general method for training is given in **Algorithm 1.** Note that validation and test stages are not shown here. Please take a look at the attached .py file.

Notice that the matrix notations are compliant with the course slides and the 1st mini-project.

Algorithm 1. Pseudocode for training the 3-layer perceptron

```
Initialize all extended weights $\bar{\mathbf{W}}^{n}$ for $n = 0{:}2$ and $\Delta\bar{\mathbf{W}}^{prev}_{out\times in+1}$ with zeros
for epoch do
    for sample=$(\mathbf{x}, \mathbf{d})$ do # data and labels
        for $n = 0{:}1$ with input & output sizes $(in\,\&\,out)$ do # Forward pass
            $\bar{\mathbf{x}}_{in+1\times1} \leftarrow \mathbf{x}_{in\times1} \parallel -1$ # extended input
            inputs[n]$\leftarrow \bar{\mathbf{x}}$
            If dropout enabled, randomly mask elements in $\bar{\mathbf{x}}_{in+1\times1}$ with $\mathbf{p}$ probability
            $\mathbf{v}_{out\times1} \leftarrow \bar{\mathbf{W}}^{n}{}_{out\times in+1}\bar{\mathbf{x}}_{in+1\times1}$
            $\mathbf{o}_{out\times1} \leftarrow f(\mathbf{v}_{out\times1})$
            If in validation or test and dropout enabled, scale $\mathbf{o}_{out\times1}$ with $\mathbf{p}$
            Construct the activation derivative matrix $\boldsymbol{\Gamma}^{n\prime}{}_{out\times out}$
            $\mathbf{x}_{in\times1} \leftarrow \mathbf{o}_{out\times1}$
        end for
        $\mathbf{e}_{out\times1} = -\log(\mathbf{o}_{out\times1}[\mathbf{d}])$
        # log likelihood error, output is 1-hot encoded vector, get the ground
        truth label's $(\mathbf{d})$ class response
        for $n = 2{:}0$ with input & output sizes $in\,\&\,out$ do # Backward pass
            $\boldsymbol{\nabla}^{n}_{out\times1} \leftarrow \boldsymbol{\Gamma}^{n\prime}_{out\times out}\,\mathbf{W}^{T(n+1)}_{out\times in}\,\boldsymbol{\nabla}^{(n+1)}_{in\times1}$ where $\mathbf{W}^{T(2)} = I_{out\times out}$, $\boldsymbol{\nabla}^{(2)} = \mathbf{e}_{out\times1}$
            # act as if there is another layer with error gradients and identity
            weights at the output of the network
            $\bar{\mathbf{x}} \leftarrow$inputs[n]
            $\Delta\bar{\mathbf{W}}^{n}_{out\times in+1} \leftarrow \eta\boldsymbol{\nabla}^{n}_{out\times1}\,\bar{\mathbf{x}}^{T}{}_{1\times in+1}$ # calculate step size
            delta_w_list[n]$\leftarrow \Delta\bar{\mathbf{W}}^{n}_{out\times in+1}$
        end for
        for $n = 0{:}2$ with input & output sizes $(in\,\&\,out)$ do # Gradient descent
            if mod(sample_num, batch_size) == 0
```

```
23              $\bar{\mathbf{W}}^{n}_{\text{out}\times\text{in}+1} \leftarrow \bar{\mathbf{W}}^{n}_{\text{out}\times\text{in}+1}$ + avg(delta_w_list[n]) + $\Delta\bar{\mathbf{W}}^{prev}_{\text{out}\times\text{in}+1}$* momentum_const
                $\Delta\bar{\mathbf{W}}^{prev}_{\text{out}\times\text{in}+1} \leftarrow$ avg(delta_w_list[n]) + $\Delta\bar{\mathbf{W}}^{prev}_{\text{out}\times\text{in}+1}$* momentum
24              empty delta_w_list[n]
25            end if
26          end for
27          Track train_error=$\frac{1}{2}\sum\mathbf{e}$ until convergence
28          Stop training if |val_error-train_error|>0.05 and epoch>50
29        end for
30        shuffle training dataset
31      end for
```

Constructing the activation derivative matrix $\boldsymbol{\Gamma}^{n'}_{\text{out}\times\text{out}}$ for ReLU is trivial (**line 11**). Since derivative of ReLU is a step function, and there are no skip-connections between the layers, $\boldsymbol{\Gamma}^{n'}_{\text{out}\times\text{out}}$ for 1st and 2nd layers are $\mathbf{diag}(\frac{\mathbf{1}}{\mathbf{2}}(\mathbf{sgn}(\mathbf{v}_{\mathbf{out}\times\mathbf{1}}) + \mathbf{1}))$, where $\text{sgn}(x) = \begin{cases} 1 \; if \; x \geq 0 \\ -1 \; else \end{cases}$.

However, for the 3rd layer which has SoftMax activation, all neuron outputs are dependent on each other. Hence, $\boldsymbol{\Gamma}^{n'}_{\text{out}\times\text{out}}$ should not be a purely diagonal matrix. We will discuss the structure of $\boldsymbol{\Gamma}^{n'}_{\text{out}\times\text{out}}$ for SoftMax in **Section 2.1.**

Early stopping (**line 28**) is determined empirically.

2.1. Classification with SoftMax

Since we are working on a classification problem, a SoftMax activation at the output is appropriate.

$\text{softmax}(\mathbf{z}) = \frac{e^{z_i}}{\Sigma e^{z_i}}$ function essentially turns the output layer activations into a probability distribution, providing better classification results in most scenarios. Since normalization is obtained by all output neurons, the resulting values are dependent on each other, making the activation derivatives ($\boldsymbol{\Gamma}^{n'}_{\text{out}\times\text{out}}$ in **Algorithm 1**) different from the ReLU case.

Consider the output layer input is the vector $\mathbf{x}$**.** Taking the derivative of the output $\text{softmax}(\mathbf{x})$ with respect to input $\mathbf{x}$ should be investigated in two parts, since now cross-terms will not be zero anymore. Taking the i-th output for j-th input is given in **Equation 1 and 2**.

$$\frac{\partial \text{softmax}(x)_i}{\partial x_j} = \text{softmax}(x)_i\left(1 - \text{softmax}(x)_j\right), \qquad i = j, \qquad (1)$$

$$\frac{\partial \text{softmax}(x)_i}{\partial x_j} = -\text{softmax}(x)_i\left(\text{softmax}(x)_j\right), \qquad i \neq j\,, \qquad (2)$$

Writing these in a matrix form is now straightforward. Diagonal elements of $\boldsymbol{\Gamma}^{n'}_{\text{out}\times\text{out}}$ will come from **Equation 1,** and off-diagonal elements will come from **Equation 2.**

2.2. Dropout

Dropout is "dropping/masking out" neurons with random probability while training. This avoids certain neurons overpowering others and hence enables other neurons to have a chance of learning better weights. It is also known as a model ensemble, since at the end we would take the average response of each masked sub-network.

There are two main methods of dropout: regular dropout and inverted dropout. **Regular dropout** masks the inputs with probability **p** in training. In inference, the activations are also scaled with **p**. In **inverted dropout**, to not modify the inference procedure, weights are masked with probability **p** and scaled with **1/p** in training.

I used **regular dropout** in my implementation, but in theory, they are identical.

2.3. Momentum

Momentum enables keeping track of the gradient updates so that they would not have high variations, hence avoiding divergence issues. A normal gradient descent update is done like in **Equation 3.** Adding momentum changes the step size and takes the previous weight update into account (**Equation 4),** scaled with a momentum constant**.**

$$\bar{\mathbf{W}}^{n}{}_{\mathrm{out\times in+1}} \leftarrow \bar{\mathbf{W}}^{n}{}_{\mathrm{out\times in+1}} + \Delta\bar{\mathbf{W}}^{n}_{\mathrm{out\times in+1}}, \qquad (3)$$

$$\bar{\mathbf{W}}^{n}{}_{\mathrm{out\times in+1}} \leftarrow \bar{\mathbf{W}}^{n}{}_{\mathrm{out\times in+1}} + \Delta\bar{\mathbf{W}}^{n}_{\mathrm{out\times in+1}} + \Delta\bar{\mathbf{W}}^{\boldsymbol{prev}}_{\mathrm{out\times in+1}} * \mathrm{const},$$

$$\Delta\bar{\mathbf{W}}^{\boldsymbol{prev}}_{\mathrm{out\times in+1}} \leftarrow \Delta\bar{\mathbf{W}}^{n}_{\mathrm{out\times in+1}} + \Delta\bar{\mathbf{W}}^{\boldsymbol{prev}}_{\mathrm{out\times in+1}} * \mathrm{const} \quad (4)$$

2.4. Implementation Details

The implementation details are briefly given below:

- The implementation is made from scratch; hence the matrix notations are compliant with the course content. I utilized `torch` library to migrate the data to GPU and perform the training sessions faster with `CUDA`, but I did not utilize any of its automatic gradient functionalities. In other words, using `torch` is identical to using `numpy` but with GPU in this scenario; and in fact, most of the matrix functions are the same in `torch` & `numpy` in terms of valid input arguments and operation behavior.

3. Results and Discussion

This section includes all results and conclusions.

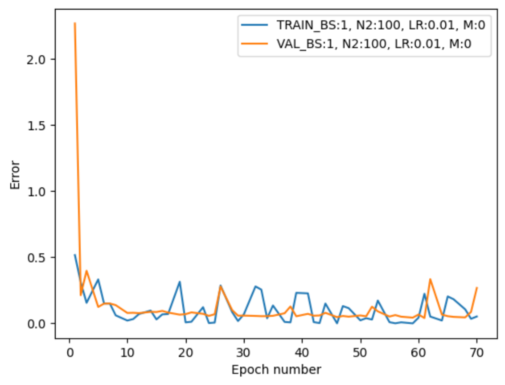
BS, **N2**, **LR**, and **Mom** represent the batch size, 2nd hidden layer size, learning rate, and momentum constant, respectively.

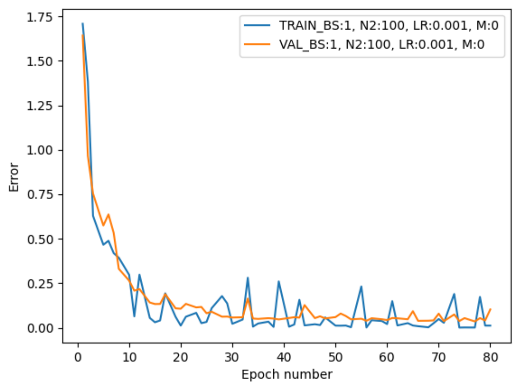
3.1. Training with different hyperparameters (no dropout, Part b)

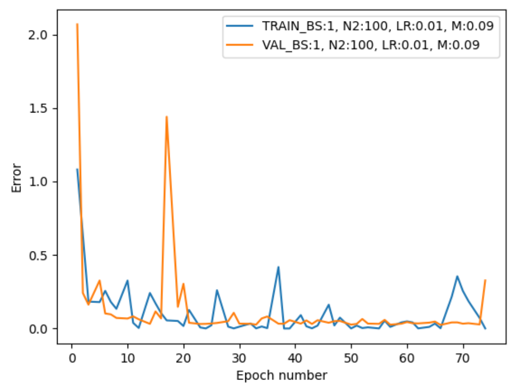
Table 1. Top-k accuracy scores for the models trained in Part b&c. The best model is given in bold. Note that Top-6 score is always 1 since there are 6 classes.

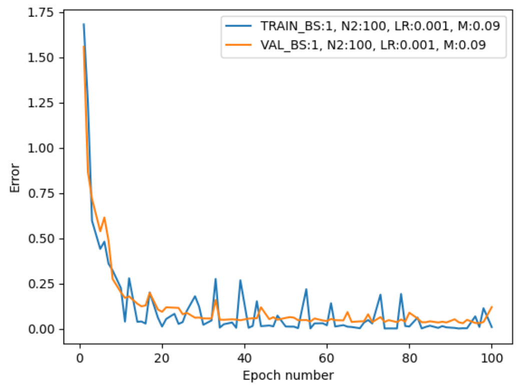
BS	N2	LR	Mom	Epoch	Dropout	Top-1	Top-2	Top-3	Top-4	Top-5
1	100	0.01	0	53	0	0.944	0.995	0.998	0.999	0.999
1	100	0.001	0	61	0	0.959	0.997	0.999	1	1
1	100	0.01	0.09	56	0	0.955	0.997	0.998	0.998	1
1	**100**	**0.001**	**0.09**	**76**	**0**	**0.966**	**0.996**	**0.998**	**1**	**1**
1	200	0.01	0	50	0	0.95	0.994	0.998	0.999	1
1	200	0.001	0	66	0	0.957	0.996	0.998	1	1
1	200	0.01	0.09	59	0	0.931	0.994	0.999	0.999	1
1	200	0.001	0.09	53	0	0.949	0.997	0.998	1	1
50	100	0.01	0	50	0	0.946	0.997	0.999	1	1
50	100	0.001	0	56	0	0.955	0.996	0.999	1	1

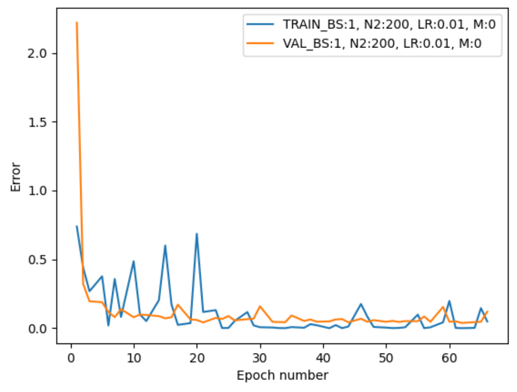
50	100	0.01	0.09	50	0	0.951	0.997	0.999	1	1
50	100	0.001	0.09	50	0	0.959	0.997	0.999	1	1
50	200	0.01	0	50	0	0.910	0.994	1	1	1
50	200	0.001	0	50	0	0.956	0.996	0.998	1	1
50	200	0.01	0.09	50	0	0.887	0.993	0.999	1	1
50	200	0.001	0.09	72	0	0.947	0.996	1	1	1
1	**100**	**0.001**	**0.09**	**100**	**0.5**	**0.942**	**0.997**	**1**	**1**	**1**

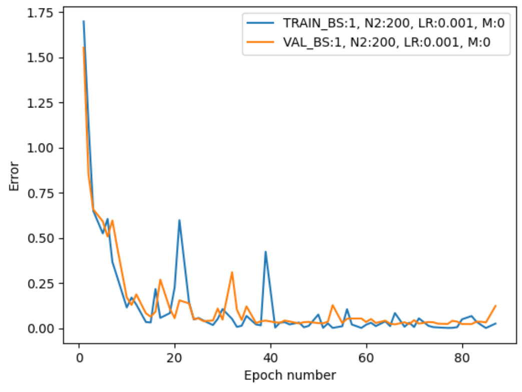


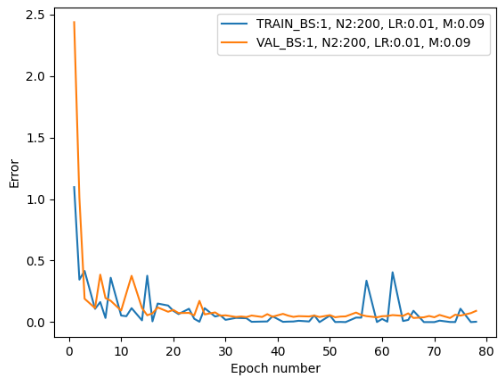


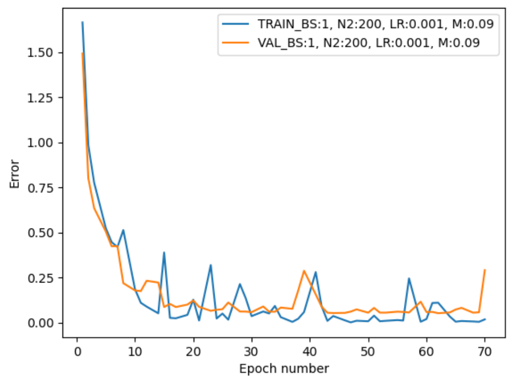


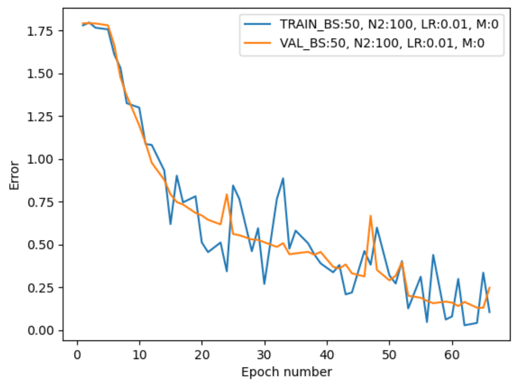


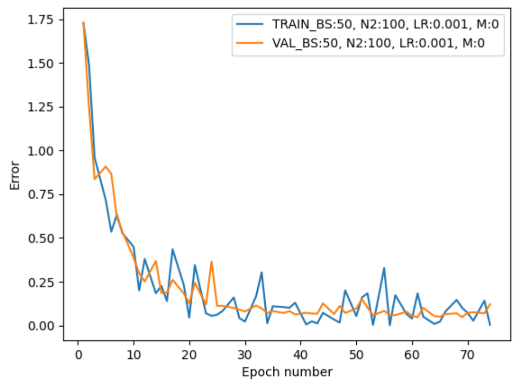


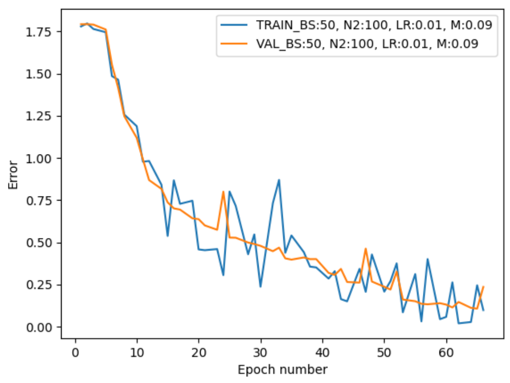


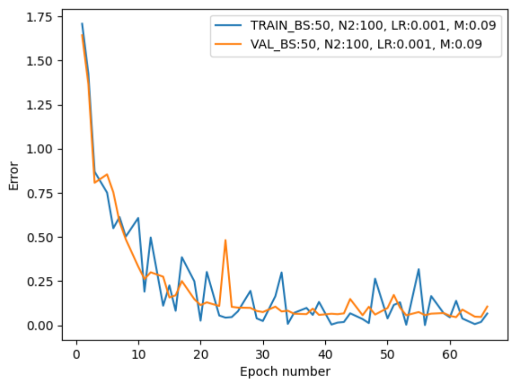

TRAIN_BS:1, N2:200, LR:0.01, M:0.09
VAL_BS:1, N2:200, LR:0.01, M:0.09
Error
Epoch number


TRAIN_BS:1, N2:200, LR:0.001, M:0.09
VAL_BS:1, N2:200, LR:0.001, M:0.09
Error
Epoch number


TRAIN_BS:50, N2:100, LR:0.01, M:0
VAL_BS:50, N2:100, LR:0.01, M:0
Error
Epoch number


TRAIN_BS:50, N2:100, LR:0.001, M:0
VAL_BS:50, N2:100, LR:0.001, M:0
Error
Epoch number


TRAIN_BS:50, N2:100, LR:0.01, M:0.09
VAL_BS:50, N2:100, LR:0.01, M:0.09
Error
Epoch number


TRAIN_BS:50, N2:100, LR:0.001, M:0.09
VAL_BS:50, N2:100, LR:0.001, M:0.09
Error
Epoch number

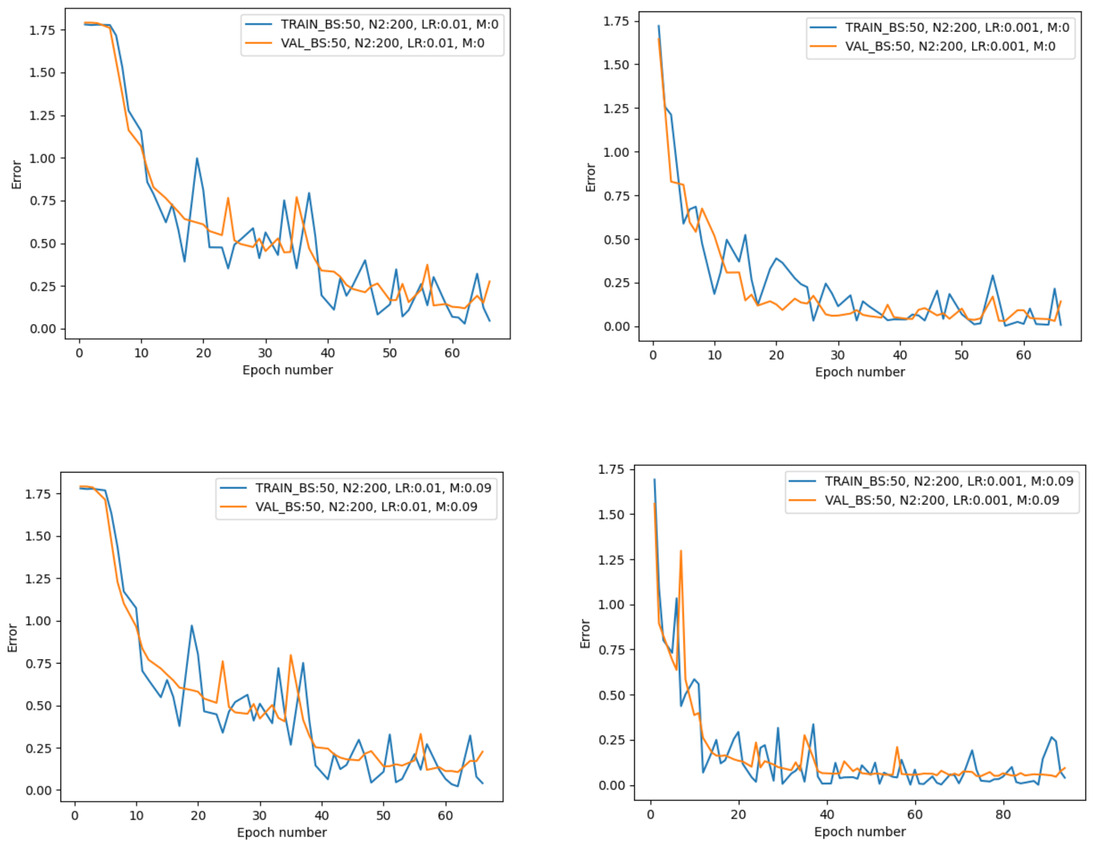


Figure 1. Training loss versus validation loss for all trained models in Part (b).

- Notice the large learning rate (0.01) combined with large number of parameters (N2=200) does not yield satisfactory results and slows down convergence compared to other hyperparameter combinations.

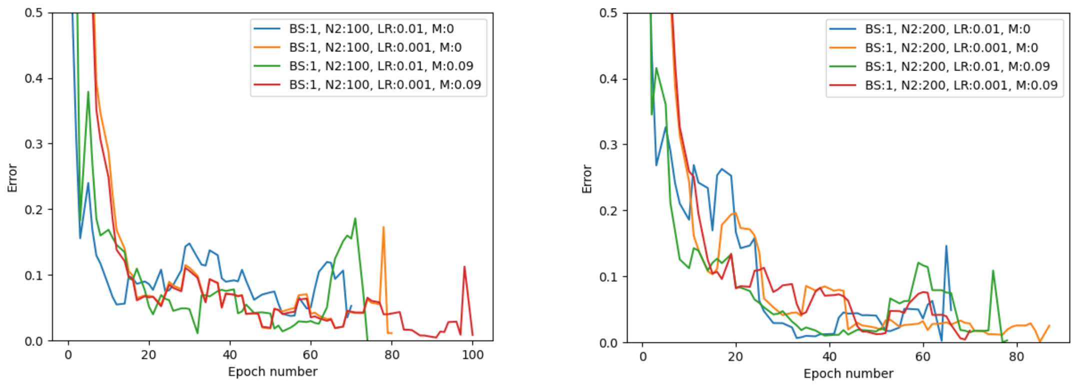


Figure 2. Comparing the models on N2 (100 on left, 200 on right) for BS=1.

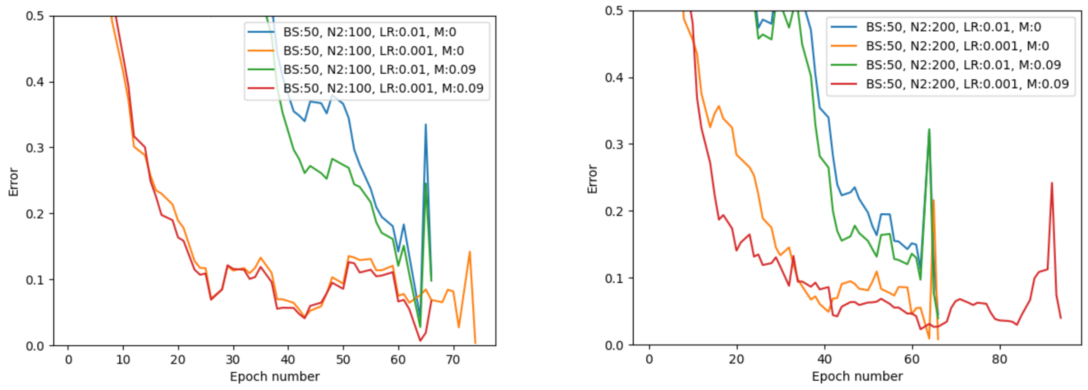
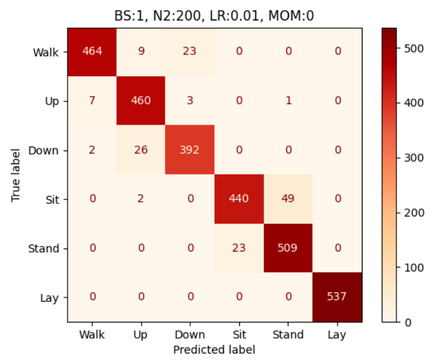
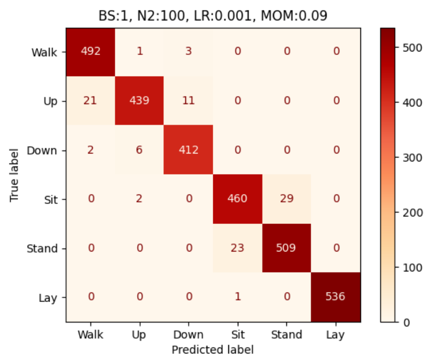
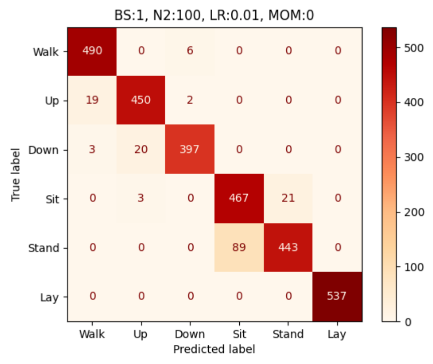


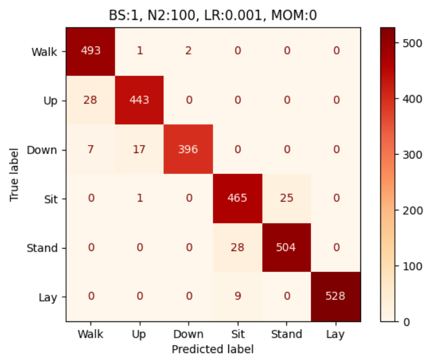
Figure 3. Comparing the models on N2 (100 on left, 200 on right) for BS=50. The error range is limited to [0,0.5] for easy visualization.

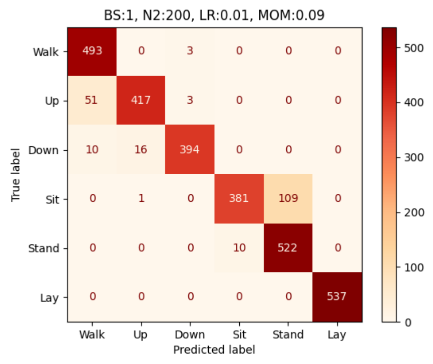
- **Figure 2** and **3** reveal that momentum is useful in cases where the learning rate is relatively large (LR=0.01) in combination with small batch size (BS=1). Since momentum accounts for running averaging the weight updates with some coefficient, it acts similar to increasing the batch size (B=50). Furthermore, even with increasing batch size (B=50), still we can see the positive effect of including momentum.
- Working with large (0.01) learning rates when trainable parameter number is also large can cause convergence issues (notice the large gap in **Figure 3** caused by the difference in learning rate).

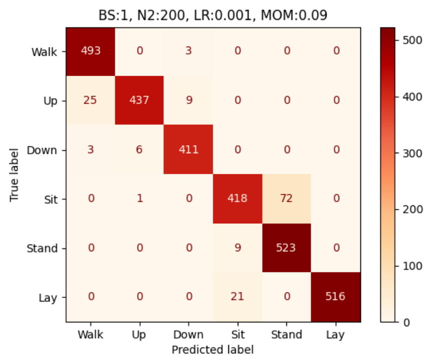

BS:1, N2:200, LR:0.01, MOM:0
True label
Predicted label
Walk Up Down Sit Stand Lay
Walk 464 9 23 0 0 0
Up 7 460 3 0 1 0
Down 2 26 392 0 0 0
Sit 0 2 0 440 49 0
Stand 0 0 0 23 509 0
Lay 0 0 0 0 0 537
0 100 200 300 400 500

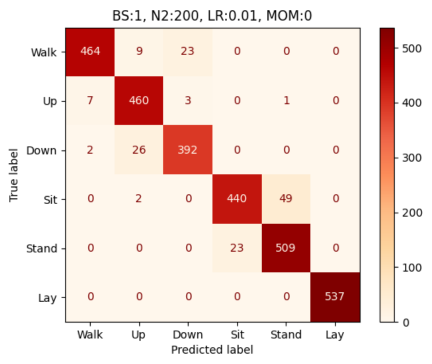

BS:1, N2:100, LR:0.001, MOM:0.09
True label
Predicted label
Walk Up Down Sit Stand Lay
Walk 492 1 3 0 0 0
Up 21 439 11 0 0 0
Down 2 6 412 0 0 0
Sit 0 2 0 460 29 0
Stand 0 0 0 23 509 0
Lay 0 0 0 1 0 536
0 100 200 300 400 500

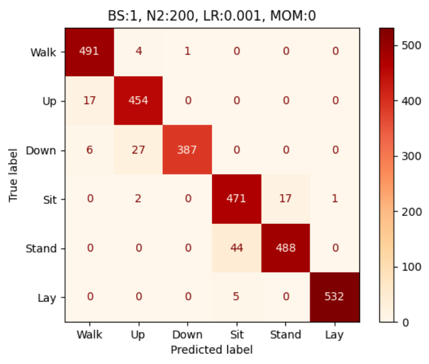

BS:1, N2:100, LR:0.01, MOM:0
True label
Predicted label
Walk Up Down Sit Stand Lay
Walk 490 0 6 0 0 0
Up 19 450 2 0 0 0
Down 3 20 397 0 0 0
Sit 0 3 0 467 21 0
Stand 0 0 0 89 443 0
Lay 0 0 0 0 0 537
0 100 200 300 400 500


BS:1, N2:100, LR:0.001, MOM:0
True label
Predicted label
Walk Up Down Sit Stand Lay
Walk 493 1 2 0 0 0
Up 28 443 0 0 0 0
Down 7 17 396 0 0 0
Sit 0 1 0 465 25 0
Stand 0 0 0 28 504 0
Lay 0 0 0 9 0 528
0 100 200 300 400 500


BS:1, N2:200, LR:0.01, MOM:0.09
True label
Predicted label
Walk Up Down Sit Stand Lay
Walk 493 0 3 0 0 0
Up 51 417 3 0 0 0
Down 10 16 394 0 0 0
Sit 0 1 0 381 109 0
Stand 0 0 0 10 522 0
Lay 0 0 0 0 0 537
0 100 200 300 400 500


BS:1, N2:200, LR:0.001, MOM:0.09
True label
Predicted label
Walk Up Down Sit Stand Lay
Walk 493 0 3 0 0 0
Up 25 437 9 0 0 0
Down 3 6 411 0 0 0
Sit 0 1 0 418 72 0
Stand 0 0 0 9 523 0
Lay 0 0 0 21 0 516
0 100 200 300 400 500


BS:1, N2:200, LR:0.01, MOM:0
True label
Predicted label
Walk Up Down Sit Stand Lay
Walk 464 9 23 0 0 0
Up 7 460 3 0 1 0
Down 2 26 392 0 0 0
Sit 0 2 0 440 49 0
Stand 0 0 0 23 509 0
Lay 0 0 0 0 0 537
0 100 200 300 400 500


BS:1, N2:200, LR:0.001, MOM:0
True label
Predicted label
Walk Up Down Sit Stand Lay
Walk 491 4 1 0 0 0
Up 17 454 0 0 0 0
Down 6 27 387 0 0 0
Sit 0 2 0 471 17 1
Stand 0 0 0 44 488 0
Lay 0 0 0 5 0 532
0 100 200 300 400 500

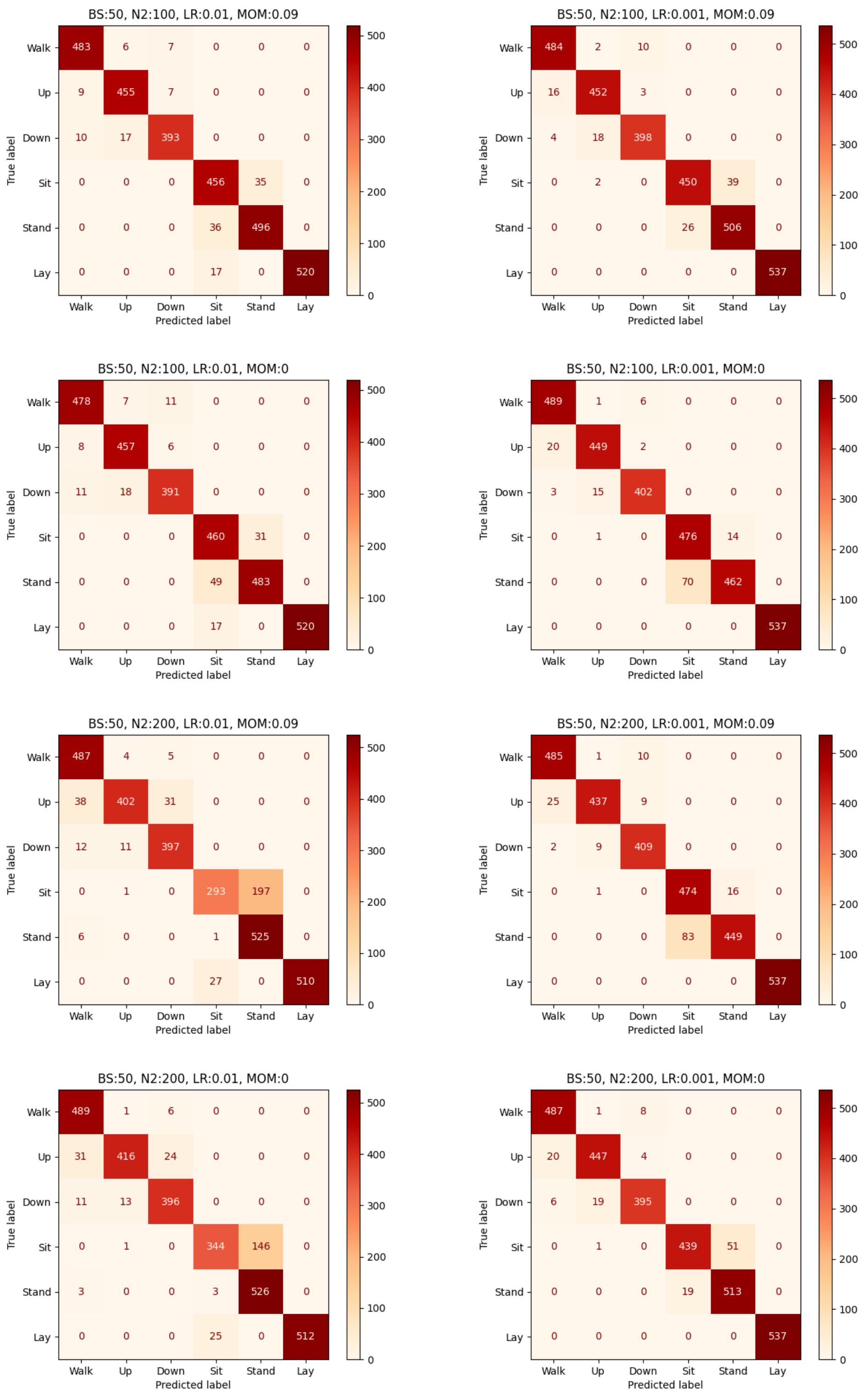


Figure 4. Confusion matrices for all trained 8 models in Part (b), including all dataset. The ideal case is to get a diagonal matrix.

- Generally, standing and sitting actions are mixed up in the classification procedure, which intuitively makes sense. There are also some noticable cases where upstairs and walking actions are also incorrectly classified. The results can be quantitatively seen in **Table 1** as well, as they are compliant with the confusion matrices.

3.2. Adding dropout to the best model in 3.1. (Part c)

The last row of **Table 1** includes the model with dropout, with probability 0.5. Notice that the Top-1 score has decreased compared to the original model; however, other k-scores are increased, proving the fact that the generalization capacity of the model has improved overall. Comparing the confusion matrices of two models in **Figure 6** also supports the observation.

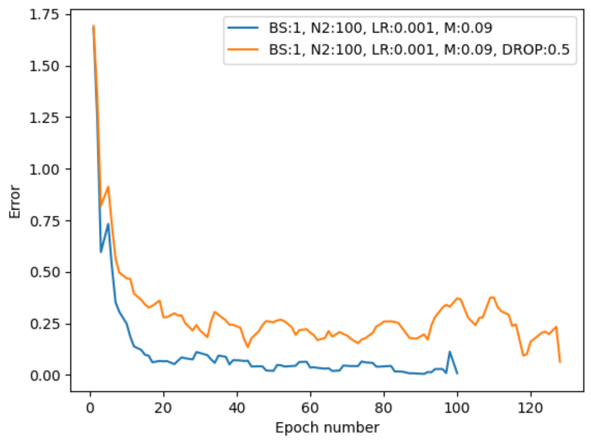
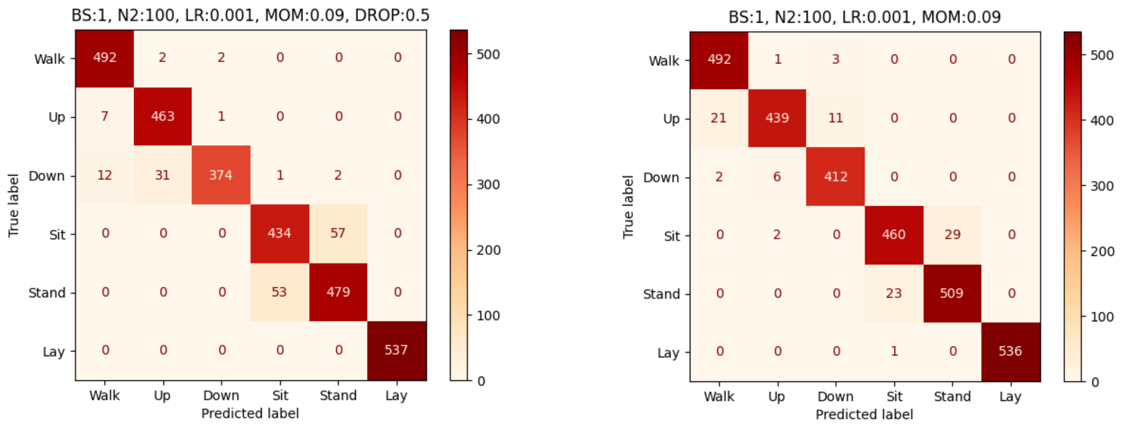


Figure 5. Training losses for the best model in Part (b), and dropout applied version in Part (c).



True \ Predicted (DROP:0.5)	Walk	Up	Down	Sit	Stand	Lay
Walk	492	2	2	0	0	0
Up	7	463	1	0	0	0
Down	12	31	374	1	2	0
Sit	0	0	0	434	57	0
Stand	0	0	0	53	479	0
Lay	0	0	0	0	0	537

True \ Predicted	Walk	Up	Down	Sit	Stand	Lay
Walk	492	1	3	0	0	0
Up	21	439	11	0	0	0
Down	2	6	412	0	0	0
Sit	0	2	0	460	29	0
Stand	0	0	0	23	509	0
Lay	0	0	0	1	0	536

Figure 6. Comparison of the confusion matrices for the best model in Part (b) (right), and the best model with dropout in Part (c) (left), including all dataset. Notice that the Up prediction has improved, but other predictions have worsened.

The usage of dropout may depend on the use case. If we have the flexibility to sacrifice from top-k scores where k is small, then dropout may work for enabling generalizability. If not, then avoiding dropout (especially for small datasets and small networks, like in this study) may be more beneficial.