Instituto Superior Técnico
IST-2024 - Deep Learning

Report - Deep Learning - HW1

Students:
BRUNO Elie
GAUGES Maira

Professor:
FIGUEIREDO Mario

December 11, 2024

Contents

1 Question I

2 Question II

2.1 Logistic Regression

2.1.1 Tuning Learning Rate

For this experiment, logistic regression was trained with a batch size of 32, L2 regularization set to 0.01, and a learning rate from the range $\{10^{-5}, 10^{-3}, 0.1\}$. The training was conducted for 100 epochs.

Table 1: Logistic Regression: Tuning Learning Rate

BS	**LR**	**L2**	**Val Acc**	**Test Acc**
32	10^{-5}	0.01	0.4694	0.4623
32	10^{-3}	0.01	0.5264	0.5247
32	0.1	0.01	0.3889	0.3807

The best performance was achieved with a learning rate of 10^{-3}, resulting in the highest validation and test accuracy (table 1). The training and validation losses for different learning rates are shown in fig. 1, and the validation accuracies are presented in fig. 2.

2.2 Feedforward Neural Network (MLP)

2.2.1 Default Hyperparameters vs. Batch Size = 512

For this experiment, the default hyperparameters were used, and the batch size was varied between 64 (default) and 512.

Table 2: Feedforward Neural Network: Default Hyperparameters vs. Batch Size = 512

BS	**HS**	**L**	**Drop**	**Val Acc**	**Test Acc**
64	200	2	0.3	0.6068	0.6057
512	200	2	0.3	0.5028	0.5200

The results (table 2) demonstrate that a smaller batch size (64) yields better performance. The corresponding training and validation losses are presented in fig. 3, and the validation accuracies are in fig. 4.

2.2.2 Effect of Dropout

The effect of different dropout values $\{0.01, 0.25, 0.5\}$ was explored.

Table 3: Feedforward Neural Network: Effect of Dropout

Model	**Drop**	**HS**	**L**	**Val Acc**	**Test Acc**
MLP	0.01	200	2	0.5741	0.5713
MLP	0.25	200	2	0.6054	0.6000
MLP	0.5	200	2	0.6026	0.5963

The results (table 3) indicate that dropout values around 0.25–0.3 achieve the best balance between regularization and performance. Training and validation losses for different dropout values are shown in fig. 5, and the validation accuracies are presented in fig. 6.

2.2.3 Effect of Momentum

Momentum values $\{0.0, 0.9\}$ were tested with a batch size of 1024.

Table 4: Feedforward Neural Network: Effect of Momentum

Model	**Mom**	**BS**	**Val Acc**	**Test Acc**
MLP	0.0	1024	0.4701	0.4883
MLP	0.9	1024	0.5933	0.6033

The results (table 4) demonstrate that using momentum significantly improves validation and test accuracy. Training and validation losses for different momentum values are shown in fig. 7, and the validation accuracies are presented in fig. 8.

2.3 Discussion

The results show:

- The optimal learning rate for logistic regression was 10^{-3} (table 1).
- Increasing the batch size reduced training time but negatively impacted accuracy (table 2).
- Dropout values around 0.25 to 0.3 balanced regularization and model performance (table 3).
- Momentum significantly improved accuracy for large batch sizes (table 4).

3 Question III

3.1 Part a

We have a single-hidden-layer neural network with activation $g(z) = z(1-z)$ and parameters $\Theta = (W, b, v, v_0)$. The hidden layer representation is:

$$h = g(Wx + b).$$

For each hidden unit h_k:

$$h_k = g(W_k x + b_k) = (W_k x + b_k)[1 - (W_k x + b_k)].$$

Define $z_k := W_k x + b_k$. Then:

$$h_k = z_k - z_k^2.$$

Expanding z_k^2:

$$z_k^2 = (W_k x + b_k)^2 = b_k^2 + 2b_k(W_k x) + (W_k x)^2.$$

Thus:

$$h_k = (W_k x + b_k) - \left[b_k^2 + 2b_k(W_k x) + (W_k x)^2\right].$$

Rearranging terms gives:

$$h_k = (b_k - b_k^2) + (1 - 2b_k)(W_k x) - (W_k x)^2.$$

Note that $(W_k x)$ is linear in x, and $(W_k x)^2$ expands to quadratic terms in the components of x. Hence h_k is a quadratic polynomial in the entries of x.

Define the feature map

$$\phi : \mathbb{R}^D \to \mathbb{R}^{\frac{(D+1)(D+2)}{2}},$$

where

$$\phi(x) = [\,1, x_1, x_2, \ldots, x_D, x_1^2, x_1 x_2, \ldots, x_D^2\,]^T.$$

Since h_k is a quadratic polynomial in x, we can write:

$$h_k = A_{\Theta,k}\phi(x),$$

for some row vector $A_{\Theta,k}$ that depends on W_k and b_k. Stacking all $k = 1, \ldots, K$:

$$h = A_\Theta \phi(x),$$

where $A_\Theta \in \mathbb{R}^{K \times \frac{(D+1)(D+2)}{2}}$.

$$A_\Theta = \begin{pmatrix} b_1 - b_1^2 & (1-2b_1)W_{11} & \cdots & (1-2b_1)W_{1D} & -W_{11}^2 & \cdots & -W_{1D}^2 & -2W_{11}W_{12} & \cdots & -2W_{1D-1}W_{1D} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ b_K - b_K^2 & (1-2b_K)W_{K1} & \cdots & (1-2b_K)W_{KD} & -W_{K1}^2 & \cdots & -W_{KD}^2 & -2W_{K1}W_{K2} & \cdots & -2W_{KD-1}W_{KD} \end{pmatrix}.$$

3.2 Part b

The output is given by:

$$\hat{y} = v^T h + v_0.$$

Substitute $h = A_\Theta \phi(x)$:

$$\hat{y} = v^T(A_\Theta \phi(x)) + v_0.$$

Define a new parameter vector

$$c_\Theta := \begin{bmatrix} v_0 \\ A_\Theta^T v \end{bmatrix} \in \mathbb{R}^{\frac{(D+1)(D+2)}{2}}.$$

Then:

$$\hat{y}(x; c_\Theta) = c_\Theta^T \phi(x).$$

This shows that $\hat{y}$ is a linear function of $\phi(x)$, parameterized by c_Θ, and thus $\hat{y}$ can be seen as a linear model in the transformed feature space.

3.3 Part c

Statement: Assume $K \geq D$. Show that for any real vector

$$c \in \mathbb{R}^{\frac{(D+1)(D+2)}{2}}$$

and any $\epsilon > 0$, there is a choice of the original parameters $\Theta = (W, b, v, v_0)$ such that $\|c_\Theta - c\| < \epsilon$. In other words, we can approximate any c-parameterization arbitrarily closely with some Θ-parameterization.

Proof/Construction:
Recall from the previous parts that:

$$\hat{y}(x; \Theta) = v_0 + v^T h = v_0 + v^T(A_\Theta \phi(x)) = c_\Theta^T \phi(x),$$

where

$$c_\Theta := \begin{bmatrix} v_0 \\ A_\Theta^T v \end{bmatrix} \in \mathbb{R}^{\frac{(D+1)(D+2)}{2}}.$$

We are given any vector

$$c \in \mathbb{R}^{\frac{(D+1)(D+2)}{2}}.$$

Our goal is to find parameters $\Theta = (W, b, v, v_0)$ that produce c_Θ arbitrarily close to c.

Key Idea: Since $h = A_\Theta \phi(x)$ involves a matrix A_Θ that depends on W and b, and c_Θ involves both A_Θ and v, v_0, we want to "reverse engineer" a set of W, b, v, v_0 from a given c. To do this, consider the following steps:

1. **Dimension Setup:** We know $h \in \mathbb{R}^K$, where $K \geq D$. The matrix A_Θ maps $\phi(x) \in \mathbb{R}^{\frac{(D+1)(D+2)}{2}}$ into $\mathbb{R}^K$. Because K is at least D, we have a lot of degrees of freedom to choose A_Θ and hence W, b.
2. **Orthogonal Decomposition:** If we consider the linear map A_Θ, it can be viewed as a $K \times \frac{(D+1)(D+2)}{2}$ matrix. For large enough K, we can construct A_Θ so that its rows form a set of vectors spanning a sufficiently large subspace of $\mathbb{R}^{\frac{(D+1)(D+2)}{2}}$. Since any matrix is ϵ-close to a non-singular submatrix (when K is large), we can perturb W and b slightly to ensure A_Θ has the desired properties.
3. **Choosing A_Θ and v:** Given a target c, decompose c as

$$c = \begin{bmatrix} c_0 \\ c_{\text{rest}} \end{bmatrix},$$

where $c_0 \in \mathbb{R}$ corresponds to what will become v_0, and $c_{\text{rest}} \in \mathbb{R}^{\frac{(D+1)(D+2)}{2}-1}$ corresponds to $A_\Theta^T v$.
If we fix $v_0 = c_0$, we then need $A_\Theta^T v = c_{\text{rest}}$. Since A_Θ is $K \times \frac{(D+1)(D+2)}{2}$, we want to choose A_Θ so that for some $v \in \mathbb{R}^K$:

$$A_\Theta^T v = c_{\text{rest}}.$$

This is a linear system in v. If A_Θ is constructed (by an appropriate choice of W and b) so that it has full column rank or at least the ability to approximate any given vector c_{rest}, then we can solve for v.
4. **Approximation Procedure:** - Start with the desired c. - Set $v_0 = c_0$ directly. - Use an ϵ-approximation argument: for any $\epsilon > 0$, since $K \geq D$, we can select W and b such that A_Θ forms a basis or near-basis for the space $\mathbb{R}^{\frac{(D+1)(D+2)}{2}-1}$. By a small perturbation (using the fact any matrix is ϵ-close to a nonsingular matrix), ensure that A_Θ is well-conditioned. Then, solve $A_\Theta^T v = c_{\text{rest}}$ for v. If needed, slightly adjust W and b to reduce any approximation error.

As we can make these adjustments arbitrarily fine (due to continuity and the polynomial nature of h in terms of

$W, b)$, we can achieve $\|c_\Theta - c\| < \epsilon$. Thus, any c can be approximated to arbitrary precision by an appropriate choice of the original parameters Θ.

Conclusion: We have shown a constructive procedure to obtain Θ from any given c-vector to within ϵ-precision. This proves that the parameterization by c is essentially no different from the original Θ parameterization in terms of expressiveness, given that $K \geq D$.

3.4 Part d

Statement: Suppose we are given training data

$$\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N}$$

with $N > \frac{(D+1)(D+2)}{2}$. Define $X \in \mathbb{R}^{N \times \frac{(D+1)(D+2)}{2}}$ as the matrix whose rows are the feature vectors $\phi(x_n)^T$. Assume X has full column rank

$$\operatorname{rank}(X) = \frac{(D+1)(D+2)}{2}.$$

We consider the squared loss:

$$L(c_\Theta; \mathcal{D}) = \frac{1}{2}\sum_{n=1}^{N}\left(\hat{y}(x_n; c_\Theta) - y_n\right)^2.$$

We know that $\hat{y}(x_n; c_\Theta) = c_\Theta^T \phi(x_n)$, and thus:

$$L(c_\Theta; \mathcal{D}) = \frac{1}{2}\|Xc_\Theta - y\|_2^2,$$

where $y = [y_1, y_2, \dots, y_N]^T$.

Solution: Since this is a standard least-squares problem in the parameters c_Θ, we know the closed-form solution is given by:

$$\hat{c}_\Theta = (X^T X)^{-1} X^T y.$$

Here, $X^T X$ is invertible due to the full rank assumption on X.

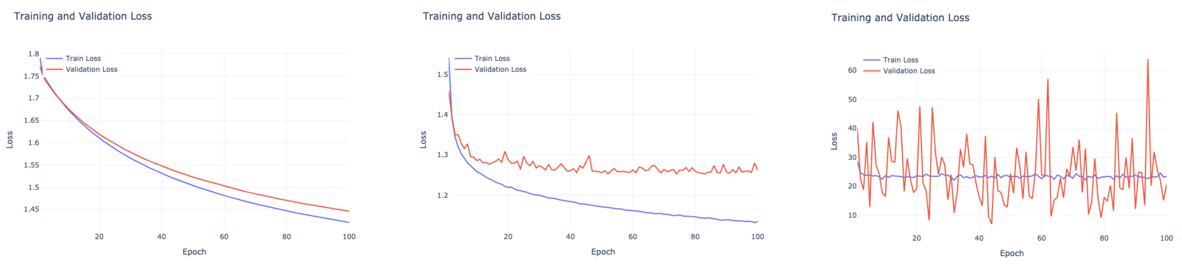
Comment on Complexity: Feedforward neural networks with nonlinear activations typically lead to nonconvex optimization problems without closed-form solutions. The usual process involves gradient-based numerical methods to find a local minimum (not guaranteed to be global).

What makes our problem special is that we used the particular activation $g(z) = z(1-z)$ and derived a quadratic feature map $\phi(x)$ so that the model becomes a linear model in the transformed feature space. This linearization via a polynomial feature map turns the problem into a straightforward linear least-squares problem, thus yielding a closed-form solution.

In contrast, other common activation functions (like sigmoid, tanh, ReLU) do not admit such a reparameterization that turns the problem into a simple linear regression. Therefore, in general, global minimization of feedforward neural networks is intractable, but here the structure of $g(z) = z(1-z)$ allows for a polynomial (specifically quadratic) feature expansion that leads to a convex optimization problem with a known closed-form solution.

4 Appendix: Graphs and Visualizations

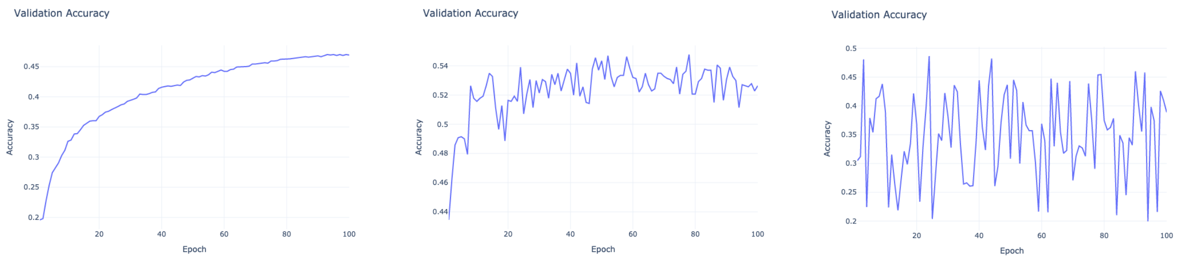
4.1 Logistic Regression: Training and Validation Loss



(a) LR = 10^{-5} (b) LR = 10^{-3} (c) LR = 0.1

Figure 1: Logistic Regression: Training and Validation Loss for different Learning Rates.

4.2 Logistic Regression: Validation Accuracy

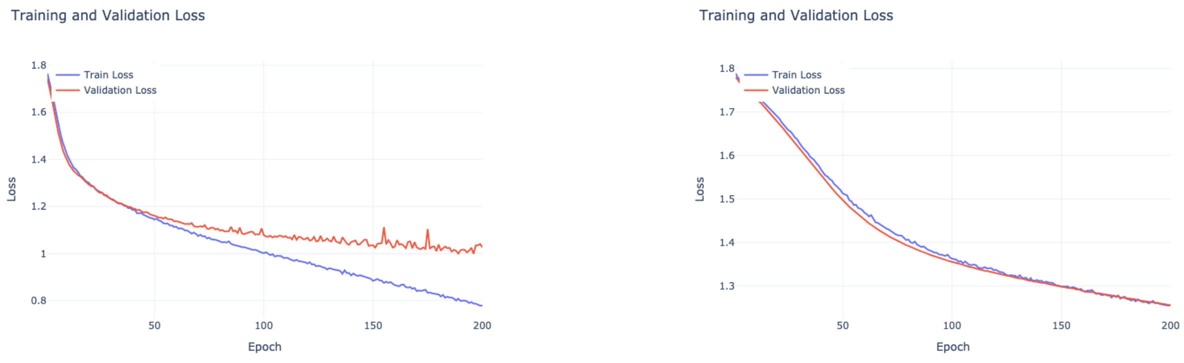


(a) LR = 10^{-5} (b) LR = 10^{-3} (c) LR = 0.1

Figure 2: Logistic Regression: Validation Accuracy for different Learning Rates.

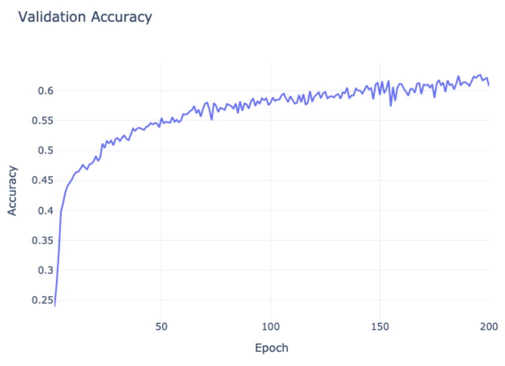
4.2.1 Default Hyperparameters vs. Batch Size = 512

The effect of varying the batch size between the default (64) and a larger value (512) was explored.

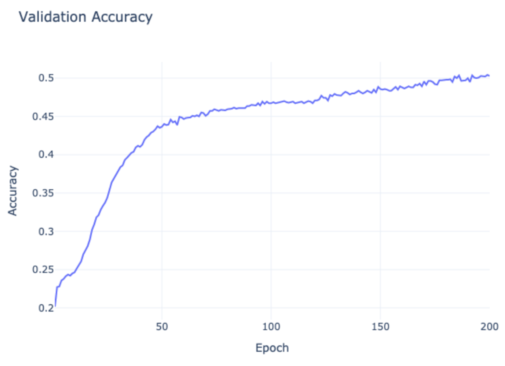


(a) Batch Size = 64 (b) Batch Size = 512

Figure 3: MLP: Training and Validation Loss for different Batch Sizes.



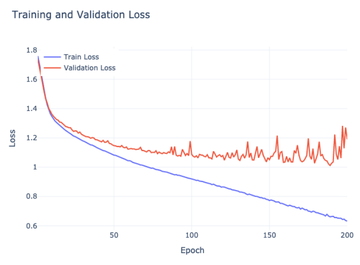
(a) Batch Size = 64



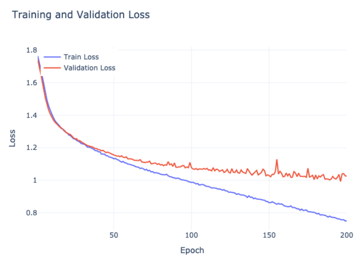
(b) Batch Size = 512

Figure 4: MLP: Validation Accuracy for different Batch Sizes.

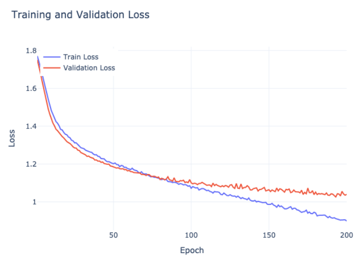
4.3 MLP: Training and Validation Loss (Effect of Dropout)



(a) Dropout = 0.01



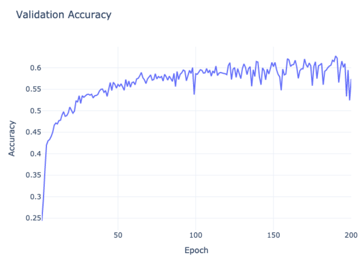
(b) Dropout = 0.25



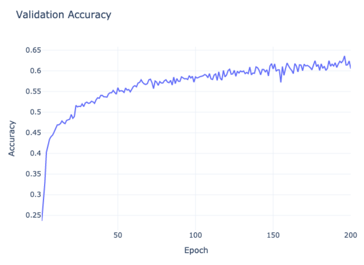
(c) Dropout = 0.5

Figure 5: MLP: Training and Validation Loss for different Dropout values.

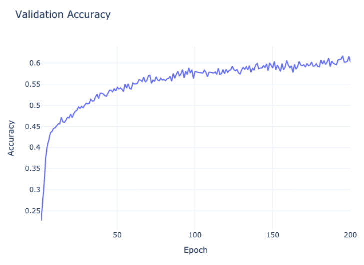
4.4 MLP: Validation Accuracy (Effect of Dropout)



(a) Dropout = 0.01



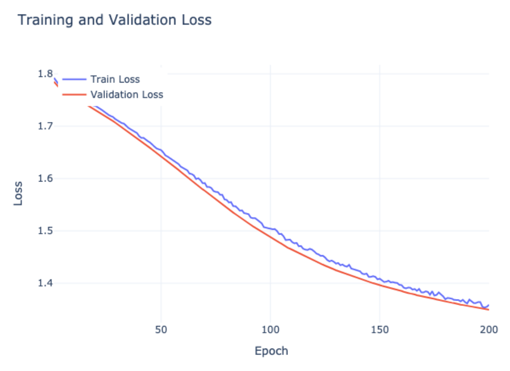
(b) Dropout = 0.25



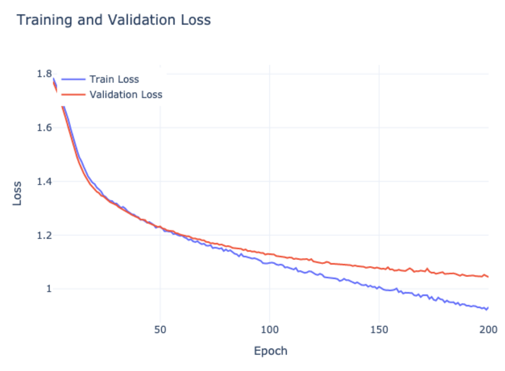
(c) Dropout = 0.5

Figure 6: MLP: Validation Accuracy for different Dropout values.

4.5 MLP: Effect of Momentum

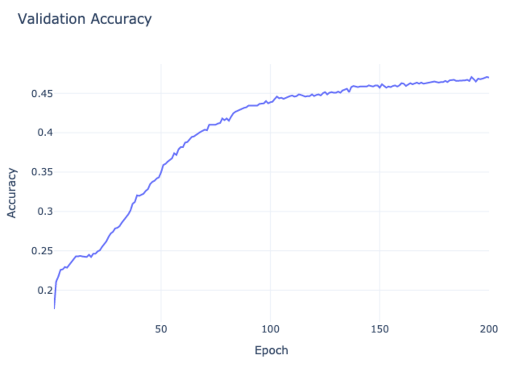


(a) Momentum = 0.0

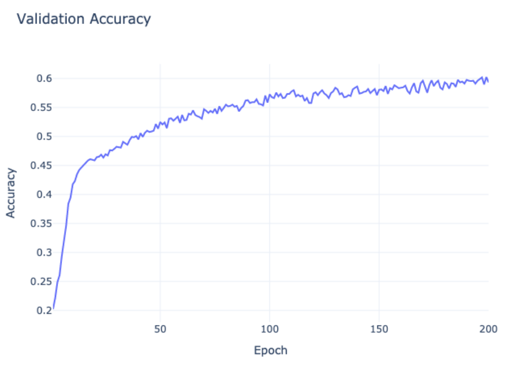


(b) Momentum = 0.9

Figure 7: MLP: Training and Validation Loss for different Momentum values.



(a) Momentum = 0.0



(b) Momentum = 0.9

Figure 8: MLP: Validation Accuracy for different Momentum values.