TÉCNICO LISBOA

Deep Learning

Homework 1

Authors:

Tiago Lopes Carvalho (106396) tiagoccarvalho@tecnico.pt
Pedro Duarte (116390) pedro.alegre.duarte@tecnico.ulisboa.pt
Tomás Fernandes (116122) tomas.santos.f@tecnico.ulisboa.pt

Group 12

2025/2026 – 1st Semester, P2

Contents

Disclaimer

The work presented herein was mostly developed in strong collaboration amongst the three members in equal measure. Group 1 was split such that Question 1 was developed by Pedro and Tiago, Question 2 was taken on by Tomás, and Question 3 was taken on by Tiago. Group 2: Question 1 was taken on by Tiago, while Questions 2 and 3 were headed by Tomás with support from Pedro and Tiago. Finally, Group 3 was headed by Pedro, with some additional work carried out by Tomás and Tiago on questions 1, 2, 5.

We hereby attest that this work was solely produced by the three members of the group. To further and speed our own learning, Generative AI was ***sparsely*** used in this project, namely for: code integrity checking and proofreading; working out some implementation caveats; help with LaTeX syntax. Nevertheless, it was always used with integrity and adhering to honest principles.

Models used include: OpenAI GPT-5.1, OpenAI GPT-5.2, Claude Opus 4.5, Claude Sonnet 4.5.

1 Question 1

1.1 Perceptron

The best test accuracy was ≈ 0.6110.

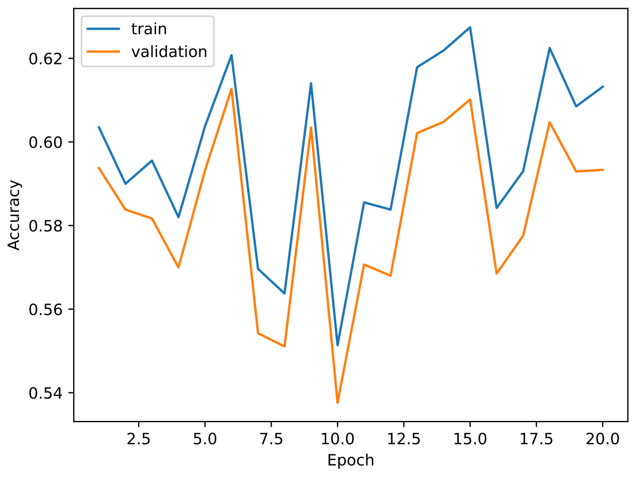


Figure 1: Training and validation accuracies for each epoch

1.2 Logistic Regression

1.2.1 Exercise A

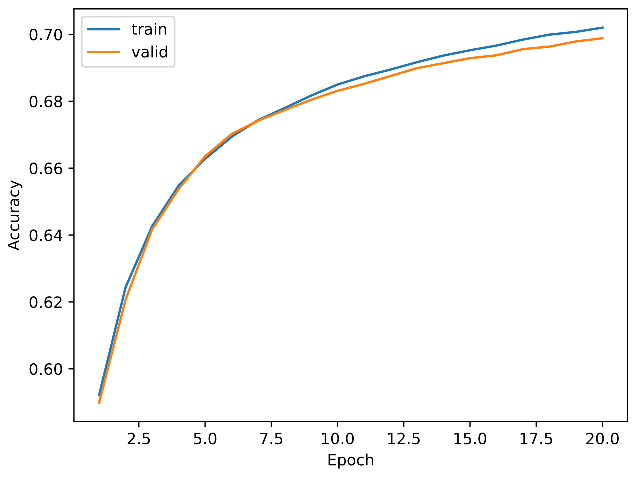


Figure 2: Training and validation accuracies for each epoch

The test accuracy of the best-performing checkpoint was ≈ 0.7022 (≈ 70%).

Note: We used small random initialization of the weight matrix to break symmetry and avoid large outputs.

1.2.2 Exercise B

The 784-dimensional pixel input suggests we should reduce dimensionality. Dimensionality-reduction techniques preserve the essential information while lowering the number of predictors, which improves generalizability and cuts training time and memory. In class we covered *Principal Component Analysis (PCA)*. Other options are *LDA*, *t-SNE* and *autoencoders* (this later was also approached in class). [1]

After deeper research we concluded PCA fits our needs: simple to implement, computationally efficient and widely used. PCA reduces dimensionality while preserving most important information. [2] EMNIST images (28x28 = 784 features/components) contain correlated and noisy pixels (neighboring pixels, blank margins around letters, correlated stroke patterns), so many features are redundant. PCA finds an orthogonal basis (principal components) that captures directions of largest variance, combining redundant variables so that keeping the top-k components retains most signal with far fewer dimensions. This yields faster training, a smaller model (fewer weights) and less memory. Projecting onto top components discards low-variance (often noisy) dimensions, which can improve generalization for a simple linear model. The choice of k can be based on explained-variance ratio (e.g., pick k so $\geq 90\%$ variance is retained), making k measurable. We can test various k to trade off reduction vs. information loss. PCA is unsupervised, so it reduces dimensions without using class labels. [2,3]

We implemented this technique from scratch (see functions `pca_fit` and `pca_transform`)

Thus, after testing several k values, we settled on $k = 90$ components, which retains an explained-variance ratio sum of 0.9315 (≈ 93%). This means we keep most of the original variability while reducing the dimensionality from 784 to 90. The training time dropped significantly, and we maintained almost the same best validation and test metrics values. For example, with $learning\_rate = 0.0001$ and $l2\_penalty = 0.00001$, training time decreased from ≈402ms to ≈165ms, while the best validation accuracy only slightly decreased from 69.92% to 69.63%, and the test accuracy dropped only minimally from 70.27% to 70.00%.

So, concluding, we found PCA a straightforward, effective fit for our task.

1.2.3 Exercise C

We performed a grid search over the following hyperparameters for logistic regression (trained with SGD, batch $size = 1$, 20 epochs, saving the checkpoint with best validation accuracy):

- Learning rates: $\{1e{-}3, 1e{-}4, 1e{-}5\}$.
- ℓ_2 penalties: $\{1e{-}5, 1e{-}4\}$.
- Feature representations: `raw` (784-d flattened pixels) and `pca` (PCA projection, dimensionality reduced - see 1.2.2 for details).

Table 1 reports, for each configuration, the best validation accuracy obtained (over 20 epochs), the corresponding test accuracy (measured at the checkpoint with best validation accuracy), and the execution time for that run, in milliseconds.

#	LR	ℓ_2	Feature	Val acc	Test acc	Duration (ms)
1	0.001	1e−5	raw	0.7201	0.7218	354.25
2	0.001	1e−5	pca	0.7140	0.7158	157.85
3	0.001	1e−4	raw	**0.7206**	**0.7222**	**359.77**
4	0.001	1e−4	pca	0.7151	0.7153	188.03
5	0.0001	1e−5	raw	0.6992	0.7027	401.83
6	0.0001	1e−5	pca	0.6963	0.7000	165.41
7	0.0001	1e−4	raw	0.6985	0.7022	386.79
8	0.0001	1e−4	pca	0.6961	0.6994	180.64
9	1e-05	1e−5	raw	0.6212	0.6236	373.16
10	1e-05	1e−5	pca	0.6205	0.6230	154.29
11	1e-05	1e−4	raw	0.6210	0.6235	348.17
12	1e-05	1e−4	pca	0.6205	0.6230	156.14

Table 1: Grid search results: best validation accuracy (over 20 epochs), test accuracy at the best-validation checkpoint and execution time. The best configuration (by validation accuracy) is highlighted in bold.

Thus, the best validation accuracy (0.7206) was obtained with:

$$\text{learning rate} = 0.001, \quad \ell_2 = 1e{-}4, \quad \text{feature} = \texttt{raw}.$$

The test accuracy measured at that best-validation checkpoint is **0.7222**. The run time for this configuration was approximately 359.77 seconds.

1.3 Multi-Layer Perceptron

The accuracy of the best performing model on the test set was ≈ 0.8803.

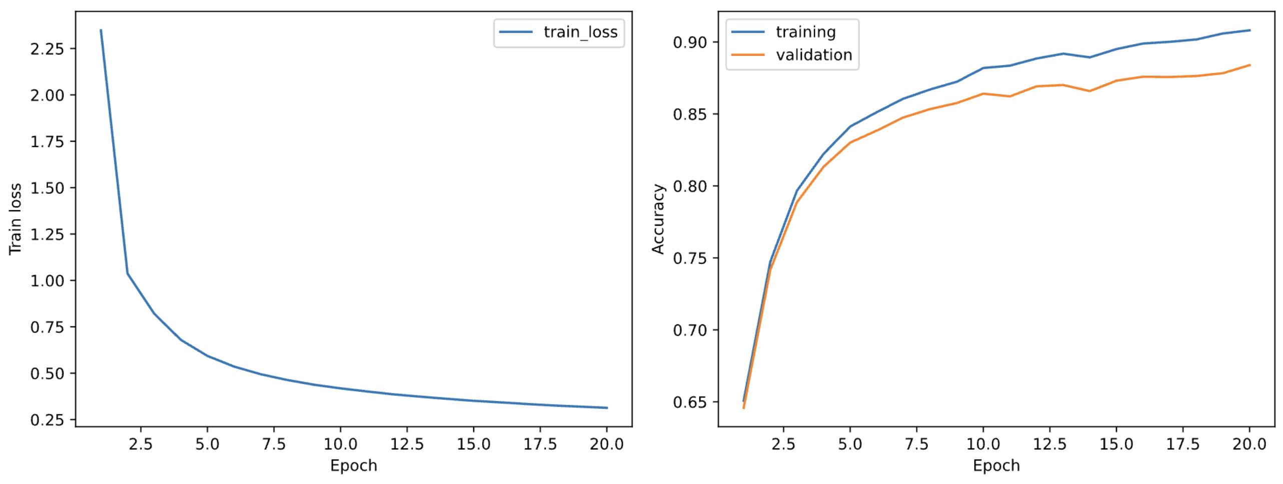
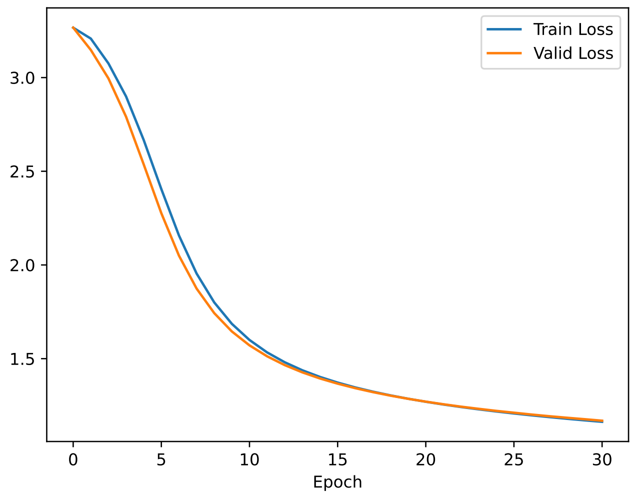


Figure 3: Train loss and train and validation accuracies by epoch number

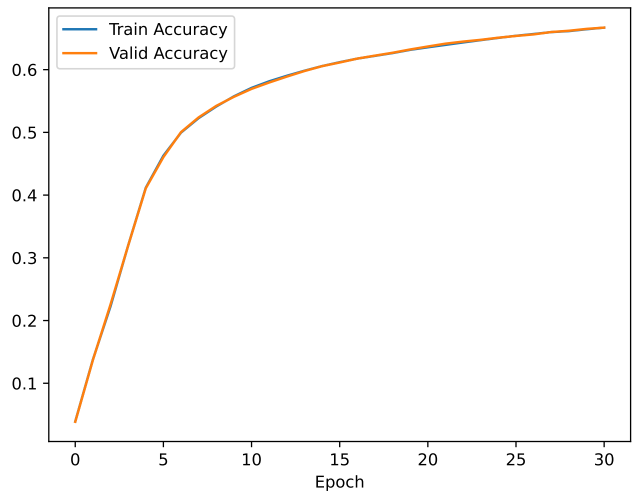
2 Question 2

2.1 Implementation Warmup

We implemented the missing pieces in `hw1_ffn_ex1.py`. The implementation is validated by the training curves: the training loss decreases and validation accuracy increases during training, and the model produces sensible final accuracies. This confirms the forward/backward/update logic and that the model trains correctly.



(a) Train and validation losses



(b) Train and validation accuracies

Figure 4: Expected behavior for the accuracies and losses

2.2 Towards an Infinite-Width One-Layer FFN

2.2.1 Exercise A

We ran the full grid (5 widths $\times$ 4 learning rates $\times$ 2 dropout settings $\times$ 2 L2 settings, 80 runs), using `Adam` as the optimizer and `relu` as the activation function. The remaining chosen hyperparameters are presented in the tables below.

Table 2: Validation accuracy for all configurations (widths 16 and 32).

LR	Dropout	L2	Val Acc
Width = 16			
0.01	0.0	0.0	0.7203
0.01	0.0	0.0001	0.7478
0.01	0.3	0.0	0.6650
0.01	0.3	0.0001	0.6834
0.001	**0.0**	**0.0**	**0.7844**
0.001	0.0	0.0001	0.7788
0.001	0.3	0.0	0.7272
0.001	0.3	0.0001	0.7256
0.0001	0.0	0.0	0.7160
0.0001	0.0	0.0001	0.7158
0.0001	0.3	0.0	0.6760
0.0001	0.3	0.0001	0.6758
1e-05	0.0	0.0	0.5995
1e-05	0.0	0.0001	0.5994
1e-05	0.3	0.0	0.5855
1e-05	0.3	0.0001	0.5853
Width = 32			
0.01	0.0	0.0	0.7786
0.01	0.0	0.0001	0.7816
0.01	0.3	0.0	0.7357
0.01	0.3	0.0001	0.7322
0.001	0.0	0.0	0.8364
0.001	**0.0**	**0.0001**	**0.8452**
0.001	0.3	0.0	0.8092
0.001	0.3	0.0001	0.8104
0.0001	0.0	0.0	0.7907
0.0001	0.0	0.0001	0.7902
0.0001	0.3	0.0	0.7698
0.0001	0.3	0.0001	0.7686
1e-05	0.0	0.0	0.6618
1e-05	0.0	0.0001	0.6619
1e-05	0.3	0.0	0.6456
1e-05	0.3	0.0001	0.6453

Table 3: Validation accuracy for all configurations (widths 64 and 128).

LR	Dropout	L2	Val Acc
Width = 64			
0.01	0.0	0.0	0.8267
0.01	0.0	0.0001	0.8304
0.01	0.3	0.0	0.7959
0.01	0.3	0.0001	0.7899
0.001	0.0	0.0	0.8779
0.001	**0.0**	**0.0001**	**0.8799**
0.001	0.3	0.0	0.8620
0.001	0.3	0.0001	0.8593
0.0001	0.0	0.0	0.8462
0.0001	0.0	0.0001	0.8456
0.0001	0.3	0.0	0.8338
0.0001	0.3	0.0001	0.8332
1e-05	0.0	0.0	0.6986
1e-05	0.0	0.0001	0.6981
1e-05	0.3	0.0	0.6919
1e-05	0.3	0.0001	0.6914
Width = 128			
0.01	0.0	0.0	0.8429
0.01	0.0	0.0001	0.8349
0.01	0.3	0.0	0.8186
0.01	0.3	0.0001	0.8120
0.001	0.0	0.0	0.8942
0.001	**0.0**	**0.0001**	**0.8954**
0.001	0.3	0.0	0.8909
0.001	0.3	0.0001	0.8913
0.0001	0.0	0.0	0.8813
0.0001	0.0	0.0001	0.8812
0.0001	0.3	0.0	0.8764
0.0001	0.3	0.0001	0.8750
1e-05	0.0	0.0	0.7304
1e-05	0.0	0.0001	0.7304
1e-05	0.3	0.0	0.7273
1e-05	0.3	0.0001	0.7268

Table 4: Validation accuracy for all configurations (width 256).

LR	Dropout	L2	Val Acc
Width = 256			
0.01	0.0	0.0	0.8438
0.01	0.0	0.0001	0.8340
0.01	0.3	0.0	0.8323
0.01	0.3	0.0001	0.8115
0.001	0.0	0.0	0.9012
0.001	**0.0**	**0.0001**	**0.9059**
0.001	0.3	0.0	0.9053
0.001	0.3	0.0001	0.9055
0.0001	0.0	0.0	0.8977
0.0001	0.0	0.0001	0.8963
0.0001	0.3	0.0	0.8963
0.0001	0.3	0.0001	0.8953
1e-05	0.0	0.0	0.7684
1e-05	0.0	0.0001	0.7679
1e-05	0.3	0.0	0.7665
1e-05	0.3	0.0001	0.7652

Validation accuracy improves as hidden width increases (from ≈ 0.7844 at 16 units to ≈ 0.9059 at 256). Improvements are substantial up to moderate widths and show diminishing returns at large widths (the gain from 128 to 256 is smaller than from 16 to 32 or 32 to 64). All best configurations used $lr = 0.001$, zero dropout and a small L2, indicating that moderate regularization plus that learning rate worked best.

2.2.2 Exercise B

The model with highest validation accuracy was:

$$\text{hidden\_size} = 256, \quad \text{learning rate} = 0.001, \quad \text{dropout} = 0.0, \quad \ell_2 = 1\text{e}{-4}$$

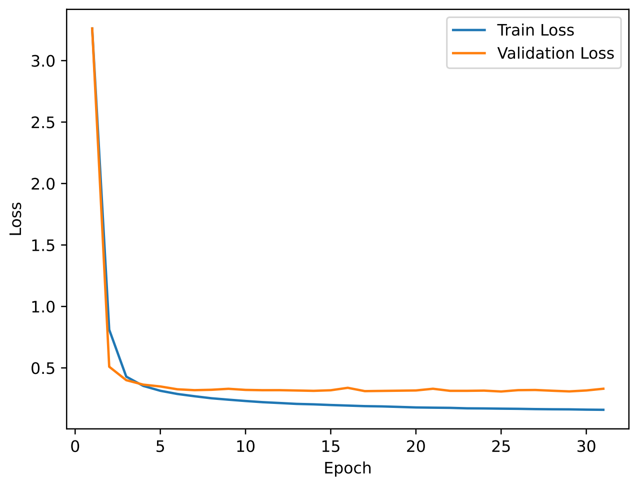
The validation accuracy was ≈ 0.9059, with the corresponding test accuracy ≈ 0.9030.

Figure 5 shows both training and validation losses, as well as training and validation accuracies over epochs.

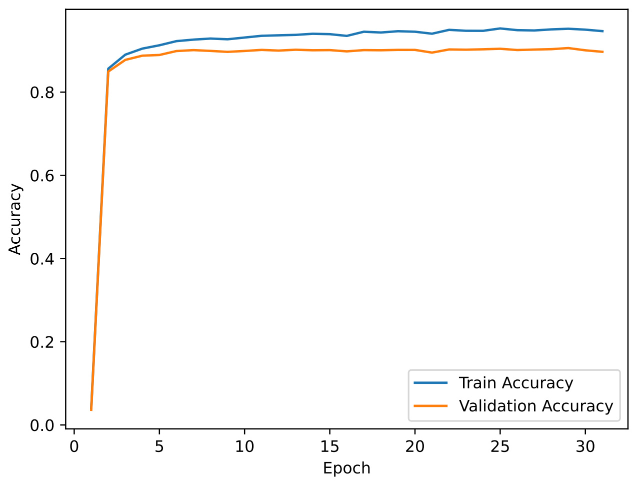
The plots show a steady decrease in training loss and a rapid increase in validation accuracy during early epochs with a plateau afterwards. The model converges within the 30 epochs (validation accuracy plateaus roughly in the middle/late epochs).

The final train accuracy at the best validation checkpoint is ≈ 0.9523, so there is a modest train-validation gap (≈ 0.05), indicating mild overfitting but overall good generalization, where the test accuracy (0.9030) is essentially in line with validation accuracy, showing the model generalizes well.

However, the validation loss starts increasing again in the later epochs, while the training loss continues to decrease. This divergence suggests that, if training were extended further, the model would begin to overfit after reaching its minimum validation loss.



(a) Train and validation losses



(b) Train and validation accuracies

Figure 5: Best overall model losses and accuracies

Optimization behaved stably (loss decreases smoothly), the model reached high training and validation performance, and no catastrophic overfitting (for this number of epochs) or failure to converge was observed.

2.2.3 Exercise C

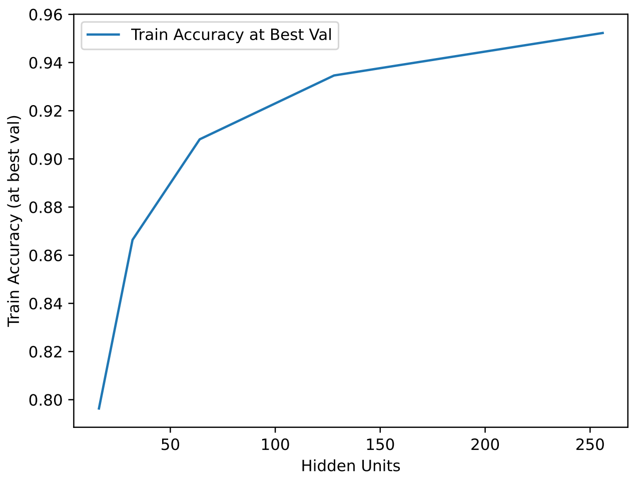


Figure 6: Final training accuracy as a function of hidden-layer width

Training accuracy increases monotonically with width, meaning that wider single hidden-layer networks fit the training data better. The gains diminish as width grows (the curve flattens), suggesting we get rapidly improving capacity at small widths and diminishing marginal improvement as width gets large (tending to converge towards maximum accuracy).

The Universal Approximation Theorem states that a single hidden layer with sufficiently many units can approximate arbitrarily well any continuous function, so in the infinite-width limit we would expect the network to have the capacity to completely interpolate the training set (i.e., approach perfect training accuracy) *assuming optimization can find a solution.* In practice, however, reaching perfect interpolation may be limited by optimization dynamics, regularization, finite precision or the finite amount of training data, so we observe high but not perfect training accuracy (≈ 0.95 at 256 units) and signs of diminishing returns. With an infinite hidden layer and ideal optimization, the expected training accuracy would approach 100% (interpolation), but that does not guarantee better generalization without appropriate regularization or other kind of optimizations.

2.3 Effect of Depth in Vanilla FFNs

2.3.1 Exercise A

Table of the highest validation accuracy reached during training for each depth:

Depth L	**Best Val Acc**
1	0.8452
3	0.8683
5	0.8660
7	0.8411
9	0.8037

Table 5: Best validation accuracy achieved for each network depth.

Validation accuracy improves when going from 1 to 3 layers (peak at $L = 3$), stays comparable at 5, and then drops for deeper models (7 and 9). This suggests a small amount of added depth helps, but beyond a moderate depth the model's validation performance degrades (likely due to optimization difficulty with an increase of parameters and/or increased overfitting capacity without extra regularization).

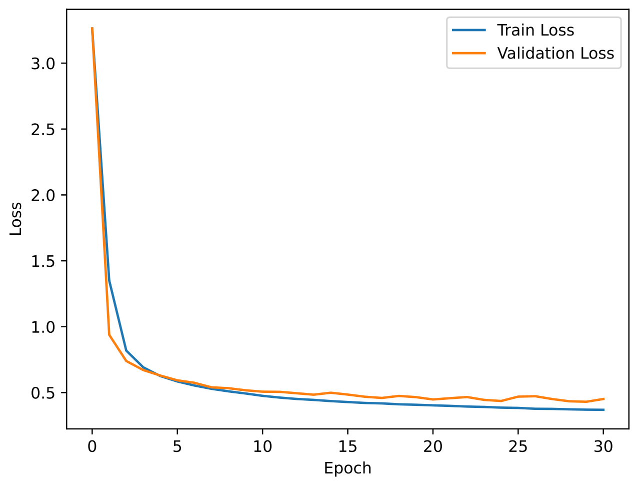
2.3.2 Exercise B

The selected configuration is $L = 3$ (highest validation accuracy) with the corresponding test accuracy ≈ 0.864567.

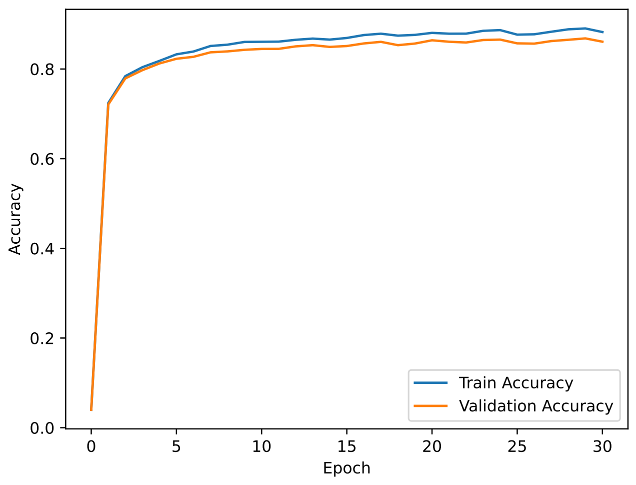
Figure 7 shows both training and validation losses, as well as training and validation accuracies over epochs.

The training loss steadily decreases and validation accuracy rises early, with rapid improvement in the first 5 epochs. Both curves flatten toward the later epochs, indicating the model appears to converge within 30 epochs (training accuracy at best validation is ≈ 0.8905, and final-epoch training accuracy is ≈ 0.8824 - see results in `results_depth_grid_search.csv`).

Regarding overfitting, there is a similar behavior to 2.2, since also here there is no strong overfitting signal, as validation accuracy (0.8683) is close to test accuracy (0.8646) and not far below training accuracy (0.8905), indicating good generalization for this configuration. The small gap between training and validation metrics suggests the model has learned generalizable patterns rather than memorizing the training data. However, the validation loss curve exhibits oscillation between epochs 20 and 30, with a mild tendency to increase in the very final epochs.



(a) Train and validation losses



(b) Train and validation accuracies

Figure 7: Best model (L=3) losses and accuracies

This pattern could indicate that with extended training beyond 30 epochs, overfitting might emerge as the training curve continues converging toward minimum error while validation performance plateaus or degrades. Nevertheless, at the current stopping point, generalization remains strong.

Comparing with the best single-layer network (256 hidden units) from the previous exercise, this deeper model is superseded by approximately 0.05 in test accuracy, suggesting that for this task and dataset, increased width may be more beneficial than increased depth in vanilla FFN architectures (without considering specific optimizations techniques applied to each architecture).

2.3.3 Exercise C

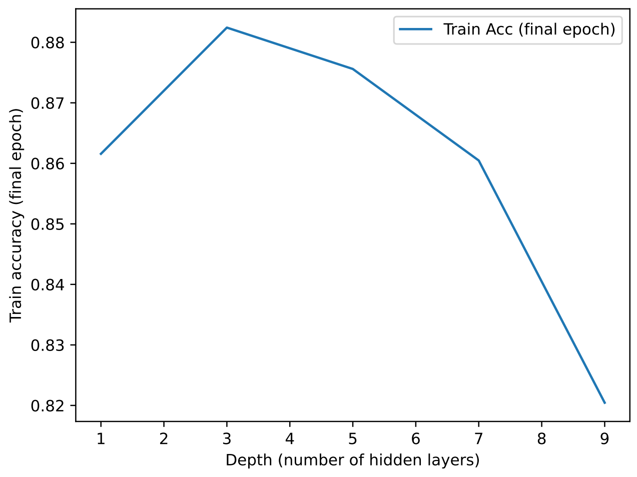


Figure 8: Final training accuracy as a function of depth

The plot of final epoch training accuracy as a function of depth reveals a non-monotonic relationship. Training accuracy does not increase consistently with depth. Instead, it exhibits an inverted-U pattern, achieving maximum training accuracy at moderate depth ($L = 3$), not at the deepest configuration, with performance deteriorating significantly beyond 5 layers.

While deeper networks theoretically possess greater representational capacity, the results demonstrate that intermediate depth (3 layers) provides the optimal balance for interpolating the training data in this setting. This suggests that raw depth alone does not guarantee improved learning capacity in practice.

The declining performance at greater depths ($L = 7$, $L = 9$) reveals optimization difficulties characteristic of deep vanilla FFNs without modern architectural innovations:

- **Vanishing gradients:** Without normalization techniques (batch normalization, layer normalization), gradients diminish as they backpropagate through many layers, severely hampering the learning of early layers.
- **Degraded gradient flow:** Deeper networks create longer pathways for gradient propagation, making optimization with standard optimizers (e.g., SGD) increasingly challenging.
- **Complex loss landscape:** Additional layers introduce more parameters and create a more convoluted optimization landscape that is difficult to navigate effectively without appropriate architectural designs.

Why deeper architectures fail to achieve higher interpolation: In this vanilla FFN setting, the optimization challenges fundamentally outweigh the theoretical capacity benefits of depth. The network cannot effectively utilize the additional layers due to gradient-related issues, resulting in worse training accuracy despite increased model complexity. This demonstrates that depth without appropriate architectural support (as learned in theoretical classes: skip connections, normalization layers, careful initialization) can be counterproductive. Modern deep learning architectures such as ResNets specifically address these limitations through skip connections, for example, enabling effective training of networks with hundreds of layers. The results highlight that architectural design is as critical as model capacity for successful deep learning.

Note: ResNets might be out of scope for this Homework, but we have just learned in this week classes, and it directly tries to solve the problem presented above. Therefore, we decided to reference it.

3 Question 3

3.1 Problem 1

$$\boxed{\operatorname{relu}(z) := \arg\min_{y \geq 0} \|y - z\|^2}$$

Proof.
Note that the objective function is separable across coordinates:

$$\|y - z\|^2 = \sum_{i=1}^{d} (y_i - z_i)^2.$$

The constraint $y \geq 0$ is also coordinate-wise, so we can minimize each term independently:

$$\min_{y_i \geq 0} (y_i - z_i)^2.$$

Without the constraint, the minimizer would be $y_i = z_i$. Note that:

- If $z_i \geq 0$, then z_i is feasible and remains the minimizer.
- If $z_i < 0$, then z_i is not feasible, so the minimum occurs at the boundary $y_i = 0$.

Therefore,

$$y_i^* = \begin{cases} z_i, & \text{if } z_i \geq 0, \\ 0, & \text{if } z_i < 0 \end{cases} = \max(z_i, 0).$$

Hence,

$$\boxed{\operatorname{relu}(z) = \max(z, 0)}$$

3.2 Problem 2

Let $z \in \mathbb{R}^d$. Recall

$$\operatorname{relu}(t) = \max(t, 0), \ \operatorname{relumax}_b(z)_i := \frac{\operatorname{relu}(z_i - \max(z) + b)}{\sum_{j=1}^d \operatorname{relu}(z_j - \max(z) + b)},$$

and the sparsemax form

$$\operatorname{sparsemax}(z)_i = \operatorname{relu}(z_i - \tau),$$

where τ is chosen so that the entries sum to 1.

3.2.1 (a) Invariance to adding a constant

Softmax. Adding c to every coordinate yields

$$\operatorname{softmax}(z_i + c) = \frac{e^{z_i + c}}{\sum_j e^{z_j + c}} = \frac{e^{z_i} e^c}{(\sum_j e^{z_j}) e^c} = \frac{e^{z_i}}{\sum_j e^{z_j}},$$

so softmax is unchanged.

Sparsemax. Sparsemax has the form $\max(z_i - \tau, 0)$ with τ satisfying $\sum_i \max(z_i - \tau, 0) = 1$. Replacing z by $z + c$ shifts both z_i and τ by c, giving

$$\max((z_i + c) - (\tau + c), 0) = \max(z_i - \tau, 0),$$

so sparsemax is unchanged.

Relumax. Since $\max(z + c) = \max(z) + c$,

$$z_i + c - \max(z + c) + b = z_i - \max(z) + b,$$

and both numerator and denominator remain the same. Thus $\operatorname{relumax}_b(z + c) = \operatorname{relumax}_b(z)$.

3.2.2 (b) Zero-temperature limit

Let entries of z be distinct and let $i^* = \arg\max_i z_i$. Consider the scaled vector z/T as $T \to 0^+$.

Softmax. For $i \neq i^*$,

$$\frac{e^{z_{i^*}/T}}{e^{z_i/T}} = e^{(z_{i^*}-z_i)/T} \to +\infty.$$

Since

$$\begin{aligned}\frac{\partial}{\partial z_i}\text{softmax}(z/T)_i &= \frac{\partial}{\partial z_i}\frac{e^{z_i/T}}{\sum_j e^{z_j/T}}\\ &= \frac{\partial}{\partial z_i}\frac{1}{\frac{e^{z_1/T}+\ldots+e^{z_j/T}}{e^{z_i/T}}}\\ &= \begin{cases}1, & \text{if } i = i^*\\ 0, & \text{if } i \neq i^*\end{cases},\end{aligned}$$

softmax concentrates on i^*.

Sparsemax. Scaling by $1/T$ amplifies the gap between the maximum and all other coordinates, causing the simplex projection to place all mass on i^*.

As proven previously, $\text{sparsemax}(z + c\mathbf{1}) = \text{sparsemax}(z)$ for any constant c, because the shift applied to z is also applied to the threshold τ.

Let the entries of z be distinct and let $i^* = \arg\max_i z_i$. By translation invariance, we can write:

$$\text{sparsemax}(z) = \text{sparsemax}(z - \max(z)\mathbf{1}) = \max(z - \max(z) - \tau, 0),$$

where τ is the threshold for the shifted vector.

Now consider the scaled vector z/T as $T \to 0^+$. For $i \neq i^*$, we have $z_i < z_{i^*}$, so:

$$\frac{z_i - z_{i^*}}{T} \to -\infty \quad \text{as } T \to 0^+.$$

Therefore:

$$\text{sparsemax}(z/T)_i = \max\left(\frac{z_i - z_{i^*}}{T} - \tau(T), 0\right) \to 0 \quad \text{for } i \neq i^*.$$

For $i = i^*$, we have:

$$\text{sparsemax}(z/T)_{i^*} = \max(-\tau(T), 0).$$

Since the output must sum to 1 and all other entries vanish, we have $\max(-\tau(T), 0) = 1$, implying $\tau(T) \to -1$.

Thus, $\lim_{T\to 0^+} \text{sparsemax}(z/T) = \mathbf{e}_{i^*}$, the one-hot vector at i^*.

Relumax.

$$\mathrm{relu}\left(\frac{z_i - \max z}{T} + b\right) = \mathrm{relu}\left(\frac{z_i - z_{i^*}}{T} + b\right).$$

For $i = i^*$ the argument is $b > 0$. For $i \neq i^*$ the argument tends to $-\infty$, giving 0.

$$\mathrm{relumax}_b(z/T)_i = \frac{\mathrm{relu}(\frac{z_i - \max z}{T} + b)}{\sum_j \mathrm{relu}(\frac{z_j - \max z}{T} + b)} \to \begin{cases} 0, & \text{if } i \neq i^* \\ 1, & \text{if } i = i^* \end{cases},$$

$$\text{as } T \to 0^+.$$

Thus the output becomes a one-hot vector at i^*.

3.2.3 (c) Existence of b such that relumax$_b(z)$ = sparsemax(z)

Given:

$$\mathrm{relumax}_b(z)_i := \frac{\mathrm{relu}(z_i - \max(z) + b)}{\sum_j \mathrm{relu}(z_j - \max(z) + b)}$$

$$\mathrm{sparsemax}(z)_i = \mathrm{relu}(z_i - \tau)$$

where τ is chosen so that $\sum_i \mathrm{sparsemax}(z)_i = 1$.
Proof.
By the normalization property of sparsemax:

$$\sum_i \mathrm{relu}(z_i - \tau) = 1$$

Therefore, we can write:

$$\mathrm{sparsemax}(z)_i = \frac{\mathrm{relu}(z_i - \tau)}{1} = \frac{\mathrm{relu}(z_i - \tau)}{\sum_i \mathrm{relu}(z_i - \tau)}$$

Matching the forms.
We want to find b such that:

$$\frac{\mathrm{relu}(z_i - \max(z) + b)}{\sum_j \mathrm{relu}(z_j - \max(z) + b)} = \frac{\mathrm{relu}(z_i - \tau)}{\sum_j \mathrm{relu}(z_j - \tau)}$$

For the numerators and denominators to match, we need:

$$z_i - \max(z) + b = z_i - \tau$$

Solving for b:

$$-\max(z) + b = -\tau \quad \Leftrightarrow \quad b = \max(z) - \tau$$

Conclusion.
For any $z \in \mathbb{R}^d$, choosing $b = \max(z) - \tau$ (where τ is the sparsemax threshold) ensures that:

$$\mathrm{relumax}_b(z) = \mathrm{sparsemax}(z).$$

3.3 Problem 3

3.3.1 Softmax

Function

$$\text{softmax}(z)_2 = \frac{e^{z_2}}{e^{z_1}+e^{z_2}} = \frac{e^t}{e^0+e^t} = \frac{e^t}{1+e^t}$$

This can be rewritten as:

$$\boxed{\text{softmax}(z)_2 = \sigma(t)}$$

where σ is the sigmoid function.

Derivative

$$\frac{d}{dt}\text{softmax}(z)_2 = \frac{d}{dt}\frac{e^t}{1+e^t}$$

Using the quotient rule:

$$= \frac{e^t(1+e^t) - e^t \cdot e^t}{(1+e^t)^2} = \frac{e^t}{(1+e^t)^2}$$

This can be written as:

$$\boxed{\frac{d}{dt}\text{softmax}(z)_2 = \frac{e^t}{(1+e^t)^2} = \sigma(t)(1-\sigma(t))}$$

3.3.2 Sparsemax

Function For $K = 2$ and $z = [0, t]$, sparsemax projects onto the 1-simplex.
The sparsemax formula is:

$$\text{sparsemax}(z)_i = \max(z_i - \tau, 0)$$

where τ is chosen so that $\sum_i \text{sparsemax}(z)_i = 1$.

Case 1: Only z_1 in support If $0 - \tau = 1$ and $t - \tau \leq 0$:

$$\tau = -1$$

This requires $t - \tau \leq 0 \Rightarrow t \leq -1$.
So for $t \leq -1$:

$$\text{sparsemax}(z)_2 = \max(t-(-1), 0) = \max(t+1, 0) = 0$$

Case 2: Both coordinates in support If both $z_1 - \tau > 0$ and $z_2 - \tau > 0$:

$$(0 - \tau) + (t - \tau) = 1$$

$$t - 2\tau = 1 \quad \Rightarrow \quad \tau = \frac{t-1}{2}$$

This requires:

- $0 - \tau > 0 \Rightarrow \tau < 0 \Rightarrow t < 1$
- $t - \tau > 0 \Rightarrow t > \tau = \frac{t-1}{2} \Rightarrow t + 1 > 0 \Rightarrow t > -1$

So for $-1 < t < 1$:

$$\text{sparsemax}(z)_2 = t - \frac{t-1}{2} = \frac{t+1}{2}$$

Case 3: Only z_2 in support If $t - \tau = 1$ and $0 - \tau \leq 0$:

$$\tau = t - 1$$

This requires $\tau \geq 0 \Rightarrow t \geq 1$.
So for $t \geq 1$:

$$\text{sparsemax}(z)_2 = 1$$

Combined:

$$\boxed{\text{sparsemax}(z)_2 = \begin{cases} 0 & \text{if } t \leq -1 \\ \dfrac{t+1}{2} & \text{if } -1 < t < 1 \\ 1 & \text{if } t \geq 1 \end{cases}}$$

Derivative

$$\boxed{\frac{d}{dt}\text{sparsemax}(z)_2 = \begin{cases} 0 & \text{if } t < -1 \\ \dfrac{1}{2} & \text{if } -1 < t < 1 \\ 0 & \text{if } t > 1 \end{cases}}$$

Note: At $t = \pm 1$, the derivative is not defined (there are corners).

3.3.3 Relumax

Function

$$\text{relumax}_b(z)_2 = \frac{\text{relu}(z_2 - \max(z) + b)}{\sum_{j=1}^{2} \text{relu}(z_j - \max(z) + b)}$$

With $z = [0, t]$:

Case 1: $t \geq 0$ (so $\max(z) = t$)

$$\text{relumax}_b(z)_2 = \frac{\text{relu}(t - t + b)}{\text{relu}(0 - t + b) + \text{relu}(t - t + b)} = \frac{\text{relu}(b)}{\text{relu}(b - t) + \text{relu}(b)}$$

- **Subcase 1a:** $b > t$ (both terms positive) $\Rightarrow \dfrac{b}{(b-t)+b} = \dfrac{b}{2b-t}$
- **Subcase 1b:** $0 < b \leq t$ (only second term positive) $\Rightarrow \dfrac{b}{0+b} = 1$

Case 2: $t < 0$ (so $\max(z) = 0$)

$$\text{relumax}_b(z)_2 = \frac{\text{relu}(t + b)}{\text{relu}(b) + \text{relu}(t + b)}$$

- **Subcase 2a:** $t + b > 0 \Rightarrow \dfrac{t+b}{b+(t+b)} = \dfrac{t+b}{2b+t}$
- **Subcase 2b:** $t + b \leq 0 \Rightarrow 0$

Combined:

$$\boxed{\text{relumax}_b(z)_2 = \begin{cases} 0 & \text{if } t \leq -b \\ \dfrac{t+b}{2b+t} & \text{if } -b < t < 0 \\ \dfrac{b}{2b-t} & \text{if } 0 \leq t < b \\ 1 & \text{if } t \geq b \end{cases}}$$

Derivative **Region 1:** $t < -b$

$$\frac{d}{dt}(1) = 0$$

Region 2: $-b < t < 0$

$$\frac{d}{dt}\frac{t+b}{2b+t} = \frac{(2b+t)-(t+b)}{(2b+t)^2} = \frac{b}{(2b+t)^2}$$

Region 3: $0 < t < b$

$$\frac{d}{dt}\frac{b}{2b-t} = \frac{0 \cdot (2b-t) - b \cdot (-1)}{(2b-t)^2} = \frac{b}{(2b-t)^2}$$

Region 4: $b < t$

$$\frac{d}{dt}(1) = 0$$

$$\boxed{\frac{d}{dt}\text{relumax}_b(z)_2 = \begin{cases} 0 & \text{if } |t| > b \\ \dfrac{b}{(2b-|t|)^2} & \text{if } |t| < b \end{cases}}$$

3.4 Problem 4

Let $z \in \mathbb{R}^K$ have distinct entries and fix $b > 0$. Define

$$k \;=\; \arg\max_{\ell} z_\ell, \qquad a_i \;=\; z_i - z_k + b, \qquad r_i \;=\; \mathrm{relu}(a_i), \qquad S \;=\; \sum_{\ell=1}^{K} r_\ell.$$

The relumax function is

$$f_i(z) = \mathrm{relumax}_b(z)_i = \frac{r_i}{S}.$$

Derivation: First note that for all i, j,

$$\frac{\partial a_i}{\partial z_j} = \frac{\partial (z_i - z_k + b)}{\partial z_j} = \delta_{ij} - \delta_{jk},$$

where δ_{ij} is the Kronecker delta ($\delta_{ij} = 1$ if $i = j$, else 0).
Using the derivative of relu (and defining the indicator $I_i = \mathbf{1}_{\{a_i > 0\}}$),

$$\frac{\partial r_i}{\partial z_j} = I_i \left(\delta_{ij} - \delta_{jk}\right).$$

Since $S = \sum_\ell r_\ell$,

$$\begin{aligned}
\frac{\partial S}{\partial z_j} &= \sum_{\ell=1}^{K} \frac{\partial r_\ell}{\partial z_j} = \sum_{\ell=1}^{K} I_\ell (\delta_{\ell j} - \delta_{jk}) \\
&= \sum_{\ell=1}^{K} I_\ell \delta_{\ell j} - \sum_{\ell=1}^{K} I_\ell \delta_{jk} \\
&= I_j - \delta_{jk} \sum_{\ell=1}^{K} I_\ell.
\end{aligned}$$

Defining $|\mathcal{A}| = \sum_{\ell=1}^{K} I_\ell$ as the number of active entries, we obtain

$$\frac{\partial S}{\partial z_j} = I_j - \delta_{jk} |\mathcal{A}|.$$

Applying the quotient rule to $f_i = r_i / S$,

$$\frac{\partial f_i}{\partial z_j} = \frac{(\partial r_i / \partial z_j)\, S - r_i\, (\partial S / \partial z_j)}{S^2}.$$

Substituting the expressions above yields the compact Jacobian formula:

$$\boxed{\frac{\partial\ \mathrm{relumax}_b(z)_i}{\partial z_j} = \frac{I_i (\delta_{ij} - \delta_{jk})}{S} - \frac{r_i}{S^2} \left(I_j - \delta_{jk} |\mathcal{A}|\right)}$$

with $I_i = \mathbf{1}_{\{z_i - z_k + b > 0\}}$, $r_i = \mathrm{relu}(z_i - z_k + b)$, $|\mathcal{A}|$ as the number of active entries, and $S = \sum_\ell r_\ell$.

3.5 Problem 5

3.5.1 Part 1: Why Cross-Entropy with Sparsemax/Relumax is Problematic

For softmax, the output is always strictly positive:

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}} > 0, \quad \forall i$$

Therefore, $L(z) = -\log \text{softmax}(z)_i$ is always well-defined and finite. Both sparsemax and relumax produce sparse outputs that can be exactly zero:

$$\text{sparsemax}(z)_i = \max(z_i - \tau, 0)$$

$$\text{relumax}_b(z)_i = \frac{\text{relu}(z_i - \max(z) + b)}{\sum_j \text{relu}(z_j - \max(z) + b)}$$

Key Problem: If the target class i falls outside the support (i.e., $\text{relumax}_b(z)_i = 0$ or $\text{sparsemax}(z)_i = 0$), then:

$$L(z) = -\log(0) = +\infty$$

This causes several issues:

- **Training instability**: Infinite loss values break gradient descent
- **Non-informative gradients**: When the output is exactly zero, the gradient may not provide useful direction
- **Numerical issues**: Even values very close to zero cause extremely large losses

3.5.2 Part 2: Computing the gradient of the loss $L(z) = -\log \frac{\textbf{relumax}(z)_i+\epsilon}{1+K\epsilon}$

Derivation: To compute the gradient of the loss, we have

$$\begin{aligned}\frac{\partial L}{\partial z_i} &= \frac{\partial}{\partial z_i}\left[-\log \frac{\text{relumax}(z)_i + \epsilon}{1 + K\epsilon}\right] \\ &= \frac{\partial}{\partial z_i}\left[-\log(\text{relumax}(z)_i + \epsilon)\right] \\ &= \frac{1}{\text{relumax}(z)_i + \epsilon}\left[\frac{\partial}{\partial z_i}(\text{relumax}(z)_i + \epsilon)\right]\end{aligned}$$

Let g_j, for the j^{th} line, be

$$g_j = \frac{\partial\, \text{relumax}_b(z)_i}{\partial z_j}$$

Combined:

$$\boxed{\frac{\partial L}{\partial z_j} = \frac{1}{\text{relumax}(z)_i + \epsilon} g_j}$$

References

[1] IBM Cloud Education. What is dimensionality reduction? *IBM Think Topics*, 2025. Available online: https://www.ibm.com/think/topics/dimensionality-reduction. Accessed: Dec. 2025.

[2] IBM Cloud Education. What is principal component analysis? *IBM Think Topics*, 2025. Available online: https://www.ibm.com/think/topics/principal-component-analysis. Accessed: Dec. 2025.

[3] Jonathon Shlens. A tutorial on principal component analysis. *Educational*, 51, 04 2014. Available online: https://arxiv.org/pdf/1404.1100.