# Assignment 1

CS F425 Deep Learning Assignment

Aditya Chopra
2019A7PS0178H

Omkar Pitale
2019A7PS0083H

Aditya Jhaveri
2018A7PS0209H

October 2021

## 1 Experimental Results

The results are the mean values of the metrics, taken over 5 runs. Experimentatl Conditions:

| Hyperparameter | Value |
|---|---|
| Optimizer | AdamW |
| Parameter Initalization | Normal Xavier Initialization |
| Random Seed | 42 (for all random functions) |
| Epochs | 50 |
| Batch Size | 32 |
| Learning Rate | $3 \cdot 10^{-4}$ |
| Weight Decay | $1 \cdot 10^{-3}$ |

Unless otherwise specified, every layer (except the last) is followed by ReLU activation. The last layer has implict softmax activation

| Index | Architecture | Parameters | Test Acc | Test Prec | Test Rec |
|---|---|---|---|---|---|
| 0 | Basline | 7850 | 0.8394 | 0.838951 | 0.840418 |
| 1 | 128, sigmoid | 101770 | 0.8606 | 0.862840 | 0.861921 |
| 2 | 128, tanh | 101770 | 0.8763 | 0.879838 | 0.877457 |
| 3 | 128, hardtanh | 101770 | 0.8806 | 0.885084 | 0.881755 |
| 4 | 128, mish | 101770 | 0.8777 | 0.883246 | 0.879091 |
| 5 | 128, leakyrelu | 101770 | 0.8777 | 0.883389 | 0.879367 |
| 6 | 128, relu | 101770 | 0.8785 | 0.884023 | 0.880200 |
| 7 | 115, 115 | 104775 | 0.8803 | 0.886437 | 0.880509 |
| 8 | 125, 27 | 101807 | 0.8795 | 0.883983 | 0.881655 |
| 9 | 512 | 407050 | 0.8826 | 0.887148 | 0.884466 |
| 10 | 350, 350 | 401110 | 0.8856 | 0.890028 | 0.886410 |
| 11 | 400, 128 | 366618 | 0.8843 | 0.887692 | 0.884947 |
| 12 | 5 | 3985 | 0.8263 | 0.826856 | 0.826550 |
| 13 | $512...2^i...16$ | 577178 | 0.8829 | 0.887832 | 0.884245 |
| 14 | 256, 64, 128 | 227018 | 0.8831 | 0.886028 | 0.883613 |
| 15 | 512, 256, 64, 256 | 569016 | 0.8843 | 0.888600 | 0.886414 |
| 16 | Convolutional Network[1] | 44374 | 0.8899 | 0.893300 | 0.891262 |

[1] The architecture details are provided in its section

# 2 Comparisons and Result Interpretation

## 2.1 Baseline Model

We compare the performance of all models with respect to the most basic artificial neural network: A Multiclass Logistic Regression Model, i.e. Model 0.

## 2.2 Effect of Loss Function

We tested each model with Cross Entropy Loss and KL-Divergence. In the case of Multi-Class Classification with hard targets (i.e. the target is a single class and not a probability distribution over the classes) KL-Divergence collapses to Cross Entropy Loss. Consequently, the results for each model trained with both the loss functions is absolutely identical (this is due to the constant random seed. If the seed was not fixed, the results would vary due to the stochastic nature of our programs.)

## 2.3 Effect of Activation Functions

### 2.3.1 Models under consideration (in decreasing order of performance)

| Index | Architecture | Parameters | Test Acc | Test Prec | Test Rec |
|---|---|---|---|---|---|
| 3 | 128, hardtanh | 101770 | 0.8806 | 0.885084 | 0.881755 |
| 6 | 128, relu | 101770 | 0.8785 | 0.884023 | 0.880200 |
| 4 | 128, mish | 101770 | 0.8777 | 0.883246 | 0.879091 |
| 5 | 128, leakyrelu | 101770 | 0.8777 | 0.883389 | 0.879367 |
| 2 | 128, tanh | 101770 | 0.8763 | 0.879838 | 0.877457 |
| 1 | 128, sigmoid | 101770 | 0.8606 | 0.862840 | 0.861921 |

The performance of ReLU and hardtanh superseded all others in all test runs, followed by mish and leaky-relu and finally tanh and sigmoid. The reason suggested by most papers is two fold:

1. Vanishing Gradients: tanh and sigmoid have their gradients saturate to 0, when the outputs reach near their respective extrema. Consequently, the models stop learning, increasing the number of epochs required for training. This problem is especially evident in DNNs and Image Classification. ReLU, LeakyReLU and Mish does not suffer with such problems, and as the output grows the gradient balances it out. The problem can be avoided in hardtanh with careful initalization.

2. Sparsity: ReLU produces 0, whenever the inputs are 0. The more such units that exist in a layer the more sparse the resulting representation. Sigmoids on the other hand are always likely to generate some non-zero value resulting in dense representations. Sparse representations seem to be more beneficial than dense representations. This may also be attributed to the reason why it performs better than LeakyReLU and why hardtanh outperforms all others.

Owing to the dominating performance of ReLU, we use the same in all further experiments.

## 2.4 Effect of Number of Parameters

- We keep the architecture constant in our investigation.

### 2.4.1 Models under Consideration (in increasing order of Parameters)

| Index | Architecture | Parameters | Test Acc | Test Prec | Test Rec |
|---|---|---|---|---|---|
| 12 | 5 | 3985 | 0.8263 | 0.826856 | 0.826550 |
| 0 | Basline | 7850 | 0.8394 | 0.838951 | 0.840418 |
| 6 | 128, relu | 101770 | 0.8785 | 0.884023 | 0.880200 |
| 9 | 512 | 407050 | 0.8826 | 0.887148 | 0.884466 |

- In general, models with more parameters have better results which is clearly visible from the experiments.
- The terrible performance of Model 12 maybe due to the fact that the comparatively much larger input space is converted to a tiny encoding space, which is not enough to represent the complex features in inputs and hence the extremely poor performance.
- As the number of parameters increases, more complex features can be represented and better the performance of the model. A smaller parameter space limits the kind of secondary representations that can be generated.

- However, as we will note later, very large parameters spaces might also not desired due to diminishing returns. The model requires much more training to achieve similar levels of accuracy, even more so to improve upon others (Eg: Model 14)

## 2.5 Effect of Addition of Layers

- We keep total number of parameters as close as possible in all models considered simultaneously

### 2.5.1 Models under Consideration (in decreasing order of performance)

| Index | Architecture | Parameters | Test Acc | Test Prec | Test Rec |
|---|---|---|---|---|---|
| 7 | 115, 115 | 104775 | 0.8803 | 0.886437 | 0.880509 |
| 8 | 125, 27 | 101807 | 0.8795 | 0.883983 | 0.881655 |
| 6 | 128, relu | 101770 | 0.8785 | 0.884023 | 0.880200 |

| Index | Architecture | Parameters | Test Acc | Test Prec | Test Rec |
|---|---|---|---|---|---|
| 10 | 350, 350 | 401110 | 0.8856 | 0.890028 | 0.886410 |
| 11 | 400, 128 | 366618 | 0.8843 | 0.887692 | 0.884947 |
| 9 | 512 | 407050 | 0.8826 | 0.887148 | 0.884466 |

| Index | Architecture | Parameters | Test Acc | Test Prec | Test Rec |
|---|---|---|---|---|---|
| 13 | 512...$2^i$...16 | 577178 | 0.8829 | 0.887832 | 0.884245 |
| 15 | 512, 256, 64, 256 | 569016 | 0.8843 | 0.888600 | 0.886414 |

- In general, adding more layers helps the model learn better and more complex representations. At every subsequent layer, a more complex representation can be created using the representations of the previous layer, allowing for better generalization.
- Hierarchial Representation is a strong reason why DNNs give such impressive results.
- In the first two sets of models, the parameter space size remains relatively constant and yet the performance of deeper netword has visible improvements over shallow networks. Even with lower number of parameters, deeper models are able to generalise better.
- In the third model set, even though the parameter space size is relatively constant, adding layers does not improve performance. This is because, although adding more layers allows for learning more complex representations, learning such complex representations is a time consuming affair and requires many more epochs to show the improve in performance. Other reasons for better performance of model 15 is discussed in the next section.

## 2.6 Encoder type models - Effect of Structure

- Addition of Layers without consdiering parameter space size ### Models under Consideration (in decreasing order of performance)

| Index | Architecture | Parameters | Test Acc | Test Prec | Test Rec |
|---|---|---|---|---|---|
| 15 | 512, 256, 64, 256 | 569016 | 0.8843 | 0.888600 | 0.886414 |
| 14 | 256, 64, 128 | 227018 | 0.8831 | 0.886028 | 0.883613 |
| 13 | 512...$2^i$...16 | 577178 | 0.8829 | 0.887832 | 0.884245 |
| 12 | 5 | 3985 | 0.8263 | 0.826856 | 0.826550 |

- For the model 12, as discussed previously the encoding space is too small to form any complex representations, and information is lost leading to poor performance.
- We make use of hierarchial representations to squeeze out as much data as possible. At each level, we expect the representations to become more complex. To achieve this goal, we must structure the layers with careful consideration as well.
- Even with double the parameter space size, and double the number of layers, performance of model 13 is equal, if not slightly worse after 50 epochs. The assumption, that progressively smaller hidden sizes might help performance is not generally true. Autoencoder type architectures, usually perform better as in the experiments.
- Model 14 and 15 uses an **Autoencoder** type architecture and gives the best performance overall. This can be attributed to the fact that the model is forced to learn a compressive encoding of the input and recreate the feature space. This allows it to abstract away unnecesary details, and utilise the important features for better performance.

## 2.7 CNNs

### 2.7.1 Architecture:

1. 2D Convolution: 6 filters and 5x5 kernel
2. ReLU
3. Max Pool: 2x2 Kernel
4. 2D Convolution: 16 filters and 10x10 kernel
5. ReLU
6. Max Pool: 2x2 Kernel
7. Flattening
8. Fully Connected Layers with hidden sizes: [128, 64, 10]

| Index | Architecture | Parameters | Test Acc | Test Prec | Test Rec |
|---|---|---|---|---|---|
| 16 | Convolutional Network | 44374 | 0.8899 | 0.893300 | 0.891262 |

### 2.7.2 Explanation

- The CNN uses several times less number of parameters than the 2nd best model, yet performs consistently better than any other model considered.
- A CNN makes use of several filters and the convolution operation to extract features from the inputs with increasing sophistication at each layer (such as edges and contours on lower layers, to more complex shapes and patterns).
- This is particularly well suited to Computer Vision tasks due to kind of features desired and extracted.