Session 4
Discussion Forum 2
Training and optimizing a Neural Network

Organization of DISCUSION FORUMs

The discussion forum are exercises/challenges that will test practically the information shared in On-line classes. You must write a program and execute it in a notebook format. At the end of the week you must submit this notebook. The submission deadline date will be Saturday before the on-line class.

The notebook must contain the responses to some questions. To respond them, use a cell in the notebook with the answers.

During the week (Monday to Friday) you can post in the forum sharing your questions, your thoughts and your accomplishments to the group. I will try to answer all your posts trying to guide you to solve the exercise and any other question that you may raise.

Learning Objectives

The objective of this Exercise is to optimize the first program you made using neural networks. You will apply and learn all the different options that we have available to optimize the training of an Artificial Neural Network. You must experiment with all the different options and try to obtain a network that generates the best possible result, in accuracy and with a best looking confusion matrix

Files for this Practice

There are some notebook examples that can be obtained from the course github REPO.

```
github.com/castorgit/DL-Course
```

The sample notebook for this practice is:

```
040_MLP_MNIST_KERAS-Baseline.ipynb
040_MLP_CIFAR_KERAS-Baseline.ipynb
```

Background on the MNIST Dataset

The MNIST dataset is a collection of handwritten digits widely used for training and testing machine learning models, particularly in the field of image processing and classification. It consists of 70,000 grayscale images, each representing a single digit from 0 to 9 (see the reference [Den12]).

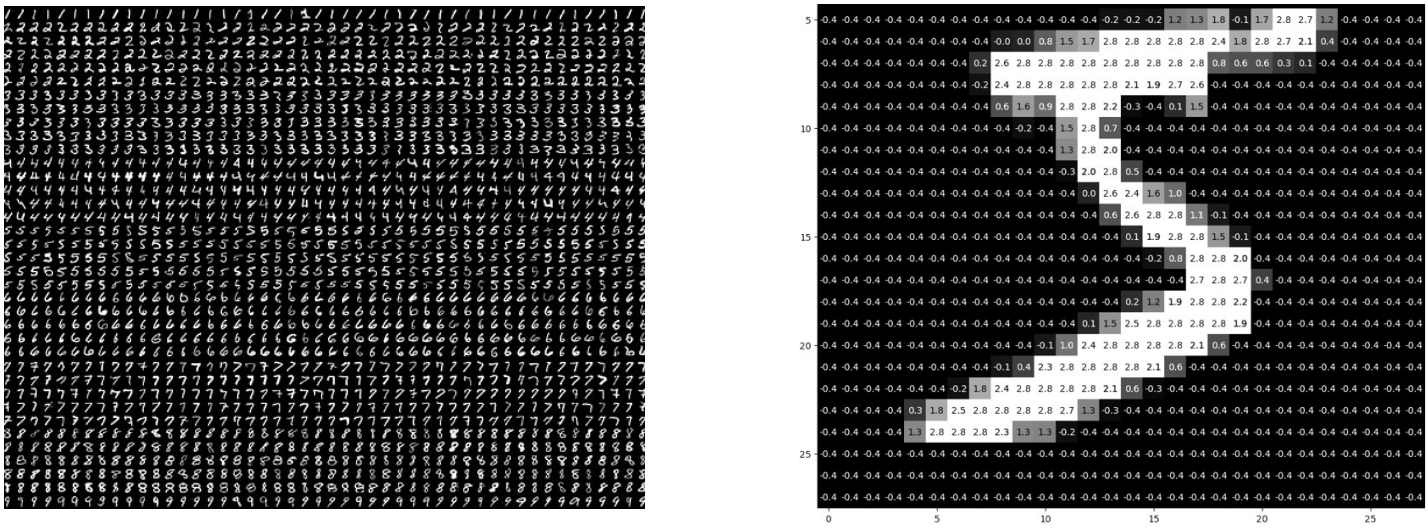
The MNIST dataset serves as a benchmark for evaluating machine learning algorithms, especially for image classification tasks. It has been extensively used to train and test various models, ranging from simple classifiers to complex convolutional neural networks.

The MNIST dataset is a fundamental resource in the machine learning community due to its simplicity, accessibility, and effectiveness in benchmarking algorithms for handwritten digit recognition.

- **Number of Samples:** The dataset contains 60,000 training images and 10,000 test images.
- **Image Dimensions:** Each image is 28 pixels in height and 28 pixels in width, resulting in a total of 784 pixels per image.
- **Classes:** The dataset is balanced, with an equal distribution of digits from 0 to 9, making it suitable for multiclass classification tasks.

Example Images

Figure 1 shows examples of handwritten digits from the MNIST dataset.



(a) MNIST numbers

(b) A single digit representation

Figure 1: The MNIST dataset

1 Background on CIFAR-10 Dataset

The CIFAR-10 and CIFAR-100 datasets are labeled subsets of the 80 million tiny images dataset. CIFAR-10 and CIFAR-100 were created by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton (see reference [Kri09]

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

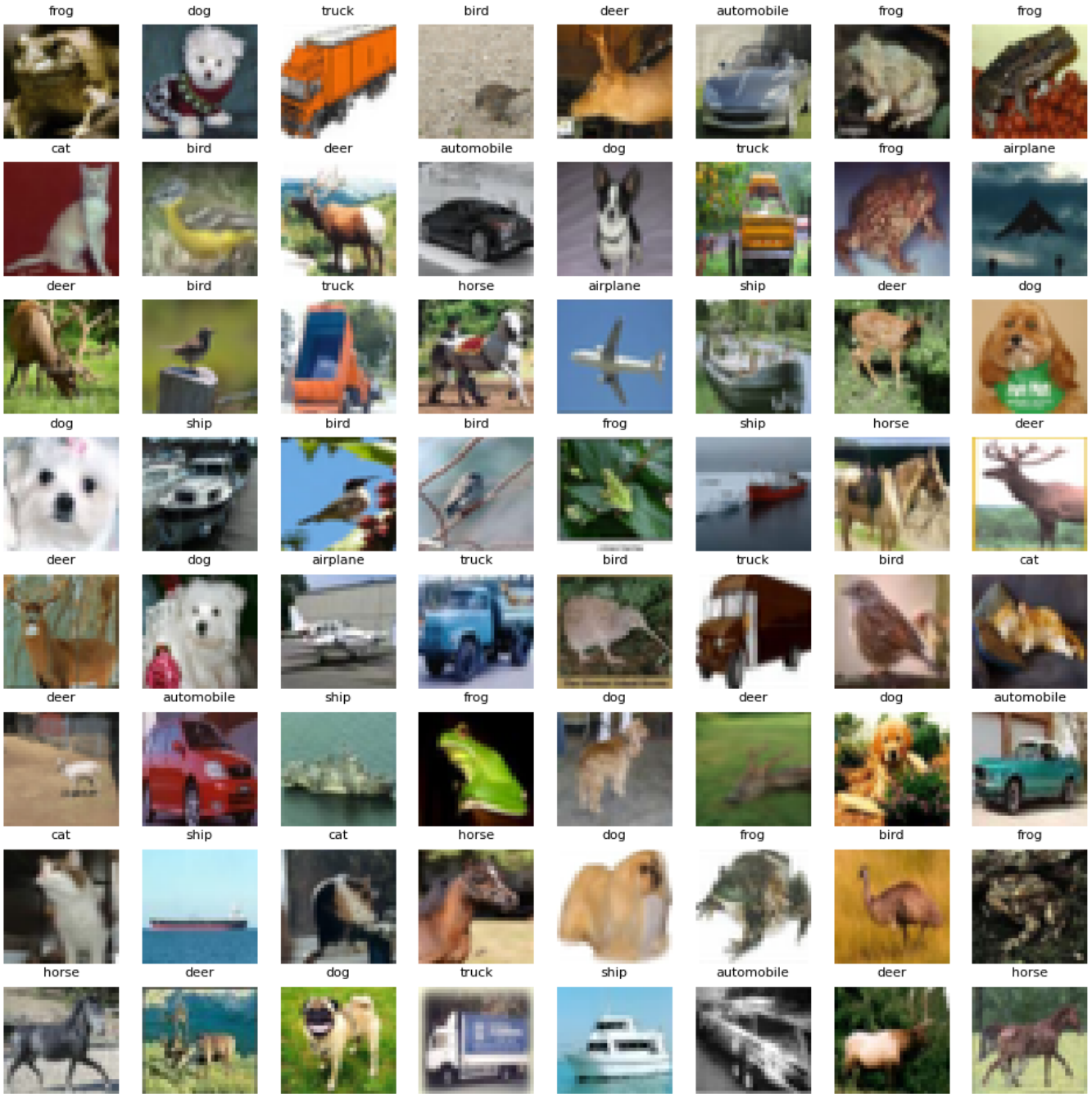


Figure 2: Example of images on CIFAR 10 Dataset

The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

The images are very small, that's the reason when printed they are blurry, as the number of pixels is very low. [1]

2 Developing a Deep Learning Classifier

To develop a deep learning image classification algorithm we need a set of labeled images, in this case MNIST which contains thousands of digital digit images. (A good book to use as an introductory book to deep learning is this one [Tra19], but to follow the examples in pytorch I recomend this one [SAV20], which has the examples in torch)

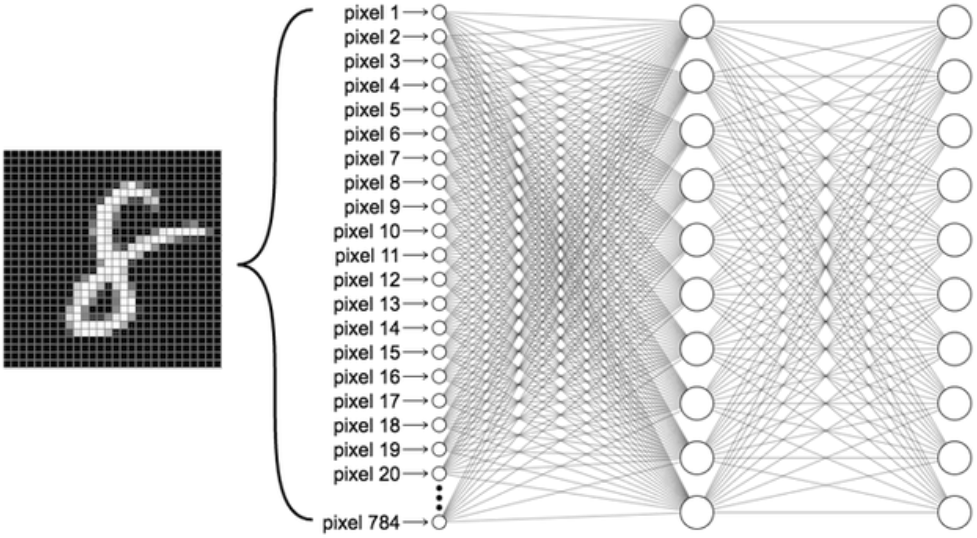


Figure 3: An MLP network for MNIST with hidden layer of 10 neurons

The first step consists of Preprocessing the data by scaling pixel values and splitting it into training, validation, and test sets.

Why do we scale the pixels in the images?. There is a common understanding that by scaling the pixels (value between 0 and 1) we create a much better representation for the network training while not losing information. There are several ways to normalize information that will be used to train an artificial neural network like [1,0] scaling or standarization. Standarization ensures that all variables have mean 0, and σ - standard deviation - of 1 (and thus also variance of 1).

[1] **CIFAR10 as a benchmark tool** Geoffrey Hinton's suggested to move away from using CIFAR-10 as a benchmark dataset in machine learning research and this was mentioned during an interview with him, which was released in late 2021. This conversation was part of broader discussions in the AI community concerning the limitations of commonly used benchmark datasets. Hinton emphasized the need for new challenges that more accurately reflect real-world complexities and encourage the development of robust, generalizable AI models. This suggestion aligns with ongoing debates about the effectiveness of current benchmarks and their role in shaping the research priorities of the machine learning community.

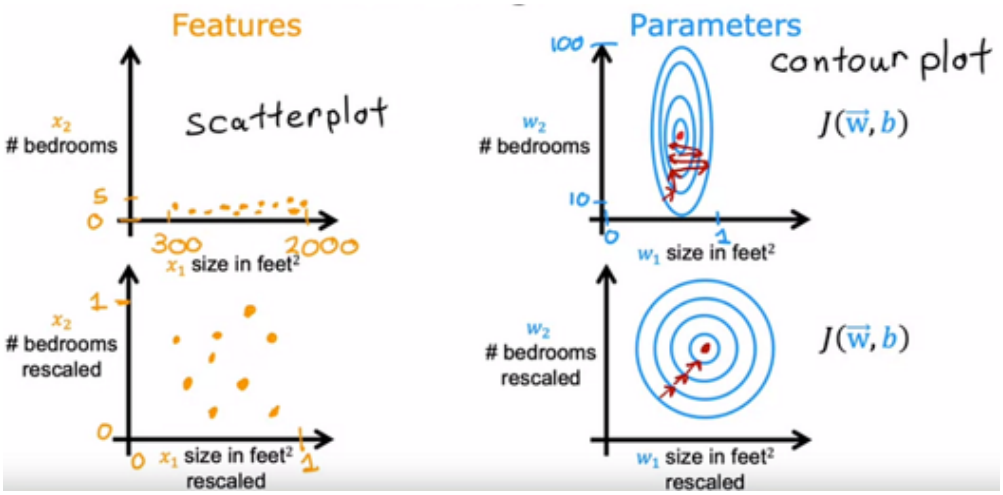


Figure 4: A comparison of Gradient Descent with/without scaling (Andrew Ng ML Course in Coursera)

$$x_i^{'} = \frac{(x_i - \bar{x})}{\sigma} \tag{1}$$

Scaling generates a more stable training as the gradients in the Gradient Descent have a better calculation (without outliers and a better distribution of points)- You can see an image representation in 4.

After the data preprocessing phase we start with the Neural Network by defining the model architecture suitable for our images, now matrixes or data structure, in our case the input layer will be 784 (or 28x28) and the output layer will be 10 (as we are recognizing 10 clases or digits). Each architecture will have a specific set of layers like convolutional, pooling, and fully connected layers, along with activation functions such as ReLU.

After defining the network we must train it (calculate the weights) by using the training dataset. For this reason we must initialize the model parameters and choose a loss function, commonly cross-entropy for classification tasks. Select an optimizer like Adam or SGD to update the model weights during training.

Train the model by passing the training data through the network, computing the loss, and updating the weights iteratively. Use the validation set to tune hyperparameters and avoid overfitting, implementing techniques like dropout or data augmentation if necessary.

After training, evaluate the model performance on the test set to gauge its accuracy and generalization capability. Visualize results using confusion matrices and ROC curves. If performance is unsatisfactory, consider refining the model architecture, adding more data, or employing transfer learning.

A Multilayer perceptron for the MNIST will have the structure of figure 3

3 Hyperparameter setting in an MLP

You have two exercises, the CIFAR-10 and the MNIST. The MNIST is fairly easy, after you've tested, then try the CIFAR10, CIFAR10 is a challenge for an MLP you'll obtain a result in the 50's%, (don't

worry, next week we will apply a convolutional network and you'll observe a big jump in accuracy). Today try to improve the accuracy little by little. By modifying the following parameters [2]:

- Layers
- Activation functions
- epochs and validation split
- Callbacks - early termination
- loss function and optimizer
- Apply regularization strategies (to CIFAR)

4 Layers

Network architecture consists of the size and amount of layers of the network. Using Keras you just 'pile up' the layers making sure there is an input and an output. If you use the net.summary() method you'll see a graphic structure of the network. You can also use the utility function plot_model to visualize the structure of the network.

```
from keras.utils import plot_model
hidden_size = 16

inputs = Input(shape= (784,))
x = Dense(hidden_size)(inputs)
x = ReLU()(x)
x = Dense(10)(x)
output = Softmax()(x)

model = Model(inputs=inputs, outputs=output)
plot_model(model, to_file='./data/model_plot.png', show_shapes=True,
    show_layer_names=True)
```

HINT: The plot_model utility may be a bit cumbersome to install. Summary is simpler and more effective.

The total number of parameters is the number of weights or trainable elements in your network. In this example you can calculate by multiplying the shapes of the layers like this $784 * 16 + 16(RELU) + 16 * 10 + 10(Softmax) = 12.730$. Think for a second that the large language models are as large as 1 trillion parameters!.

[2] In deep learning, a parameter refers to a component of the model that is learned from the training data. These are the aspects of the model that are automatically adjusted through the learning process as the model tries to minimize error and improve its accuracy on given tasks. Typically are the weights. We are using Parameter to name the different options that our neural network have for training and inferencing, but we must bear in mind that usually parameter refers to the number of weights and points directly to the complexity of the ANN

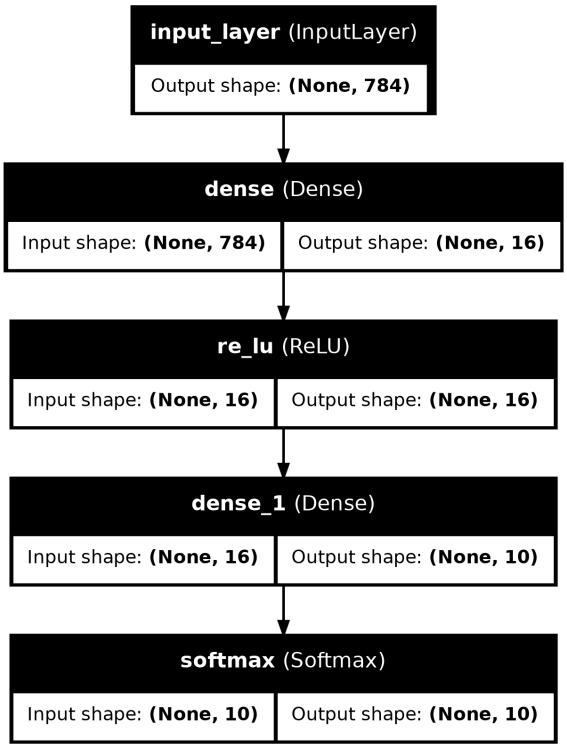


Figure 5: Structure of Network from Assignment 1 using plot_model

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 784)	0
dense (Dense)	(None, 16)	12,560
re_lu (ReLU)	(None, 16)	0
dense_1 (Dense)	(None, 10)	170
softmax (Softmax)	(None, 10)	0

Total params: 12,730 (49.73 KB)

Trainable params: 12,730 (49.73 KB)

Non-trainable params: 0 (0.00 B)

Figure 6: Structure of Network from Assignment 1 using summary() method observe that you obtain the number of parameters an important point as it tells us the complexity of the training

4.1 LAYERS: What should be done in the Assignment

You must try alternative network structures. Try 3/4 layers and try larger layers (64, 128, 256) See if your accuracy improves.

5 Activation Functions

We know that each layer must have associated an activation function. This is a function that introduces non-linearity in the model, allowing it to learn and represent complex patterns and relationships in the data. Without it it would be a linear model with very limited representation properties.

The most common activation functions are:

- **Sigmoid** : Sigmoid, useful for probabilities
- **Tanh** : Centered around 0
- **RelU** : Computationally efficient, very effective
- **Leaky ReLU:** : helps with some vanishing gradient issues
- **Softmax:** Final Layer for multi-class classification

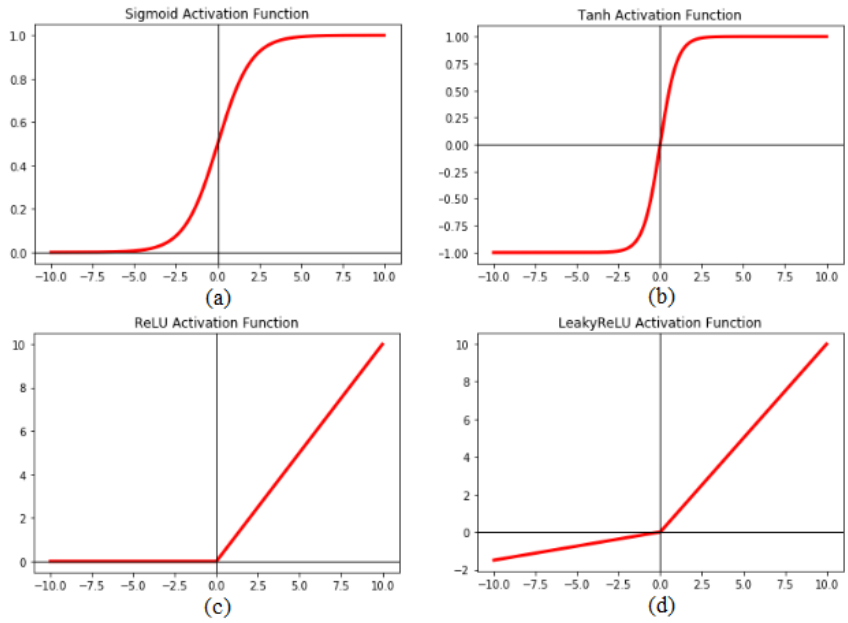


Figure 7: Two dimensional representation of several activation functions

5.1 ACTIVATIONS: What should be done in the Assignment

Try **ReLU**, **Tanh** and **Leaky ReLU**. Use only **Softmax** for the final layer.

Softmax cannot be changed because there are several properties to be taked into accoun, and for multi-class classification, we must use Softmax.

6 Forward, Backward Pass and Training definition

The neural networks have two differentiated processes. In Tensorflow or Pytorch you can program in detail each one of them independently, however, using Keras both processes are done in a simpler way.

The forward pass is the process of passing input data through the network to produce an output.

The backward pass (also called backpropagation) is the process of calculating gradients for the loss function with respect to the network parameters. These gradients are then used to update the weights and biases.

When we are training the network we use both. Forward pass calculates the accuracy (or target function on each round or epoch) and the backward pass modifies the weights.

When the neural network is trained we can use a forward pass to *infere* the result of the input. This can be done with the predict() method.

To do a forward pass for inference we use the predict() method

```
model.predict(X_test)
```

Remember that the X_test shape must be aligned with the shape of the Input Layer

We activate the training of the network with the fit() methodd

```
model.fit(x_train, y_train, epochs=100, batch_size=100, validation_split=0.2)
```

We activate the training of the network with the fit() method.

The approach to follow in the training is defined in the model, and in Keras this is done using the compile() method.

```
model.compile(optimizer='SGD',
              loss='MeanSquaredError',
              metrics=['accuracy'])
```

The compile method allows for a great deal of training approaches, but basically requires a Loss function, an Optimizer function, the set of data to be used for training (how this data will be split in training and validation), the number of epochs to be trained or a callback (if we will use early stopping or any adaptive learning rate), the learning rate, and many more. In a way this is the definition core of the training strategy of our network.

We must understand each one of the elements of the method fit() because is the one that defines how the neural network will be trained influencing the final result.

6.1 FIT: What should be done in the assignment

- Increase and decrease the epochs.
- Change the validation split to 0.3 or even 0.4. Do you see any changes?

7 epochs and callbacks

The number of epochs defines for how long the network is trained. Neural Networks have an issue with overfitting. If we train a neural network long enough it will learn the training dataset perfectly but will perform poor with the test data. Basically, it loses the capability to generalize (see 8

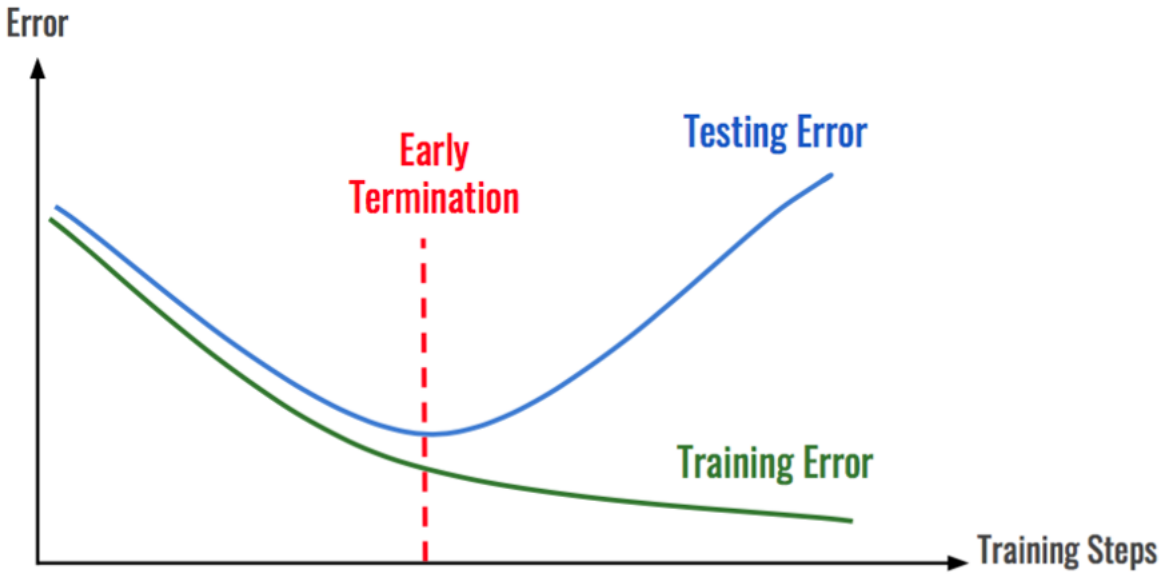


Figure 8: When does a network start overfitting

We can program (using early stopping, which is a feature in the Keras Callbacks) the early termination point. We will not do it at this point, but we need to try to understand, just looking at the accuracy and loss plots where the overfitting point sits in our problem.

7.1 What should be done in the Assignment

In the assignment just have a look at your plots to decide if the number of epochs chosen is good enough to avoid overfitting. You don't need to change anything

8 Setting up the Loss Function and the Optimizer

8.1 The Loss Function

As you've seen in class the Loss Function is critical. It ties up the target metric (usually accuracy) with the weights in the neural network. The loss quantifies the difference between the predicted output $\hat{y}$ and the true output y, This value is the loss. The loss function is the compass guiding the training process. Choosing the right one is crucial as it directly impacts the network's ability to learn and perform well on the given task. For most tasks, starting with the standard loss function for your problem type (e.g., cross-entropy for classification) is a solid choice.

The loss function must allow to generate a backward pass and to calculate the gradients to allow the modification of the weights. The loss function must have several properties, must be differentiable, Convex (or Quasi-convex), sensible to errors, and efficient (some other properties apply as well).

The most important Loss Functions used commonly with ANN are:

- **MSE** : Mean Squared Error

- **MAE** : Mean Absolute Error
- **Huber** : Combines properties of MSE and MAE, highly effective
- **Categorical Cross-Entropy** : used in Classification Tasks
- **BCE:** Binary Cross-Entropy : used in Binary Classification

The Formula of Categorical Cross Entropy is:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(\hat{y}_{ij}) \tag{2}$$

The Mathematical description formula of MAE is

$$L = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{3}$$

The Mathematical description formula of MSE is

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{4}$$

The Mathematical description formula of the Huber loss is

$$\begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2 & \text{if } |y_i - \hat{y}_i| \leq \delta, \\ \delta|y_i - \hat{y}_i| - \frac{1}{2}\delta^2 & \text{if } |y_i - \hat{y}_i| > \delta. \end{cases} \tag{5}$$

δ is a transition hyperparameter that controls transition between MSE and MAE

8.1.1 LOSS: What should be done in the Assignment

You don't need to do anything. The Loss function for a multi-class classifier is Categorical Cross Entropy, you don't have many more alternatives for this problems. There is only one that may apply which is Sparse Categorical Cross-Entropy but is very similar to the one used.

- Categorical Cross-Entropy: it is used for multi-class classifiers
- Binary Cross-Entropy: Used for binary classifications

8.2 The Optimizer

An optimizer is an algorithm used during the training of a neural network to adjust the model's weights, aiming to minimize the loss function. The optimizer plays a critical role in ensuring that the model learns effectively and converges toward an optimal or near-optimal solution

The Optimizer uses the gradients to adjust the weights, improving the network result in the search to find the optimum (which is the highest accuracy or the smaller loss). The modification of the weights is made on the direction of the gradient ∇ and using the step size defined by the learning rate or step size η.

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla L(\theta_t) \tag{6}$$

There are some optimizers that are commonly used in Artificial Neural Networks.

- **SGD** : Stochastic Gradient Descent
- **RMSprop** : Scales learning rats using an exponentially decaying average of past gradients
- **Adam** : Combines momentum and adaptive learning rates, making it one of the most popular optimizers

```
from tensorflow.keras.optimizers import SGD

model.compile(optimizer=SGD(learning_rate=0.001),
              loss=CCE,
              metrics=['accuracy'])
history = model.fit(x_train, y_train, epochs=100, batch_size=100, validation_split=0.2)
```

8.2.1 OPTIMIZER: What should be done in the Assignment

For optimizers try SGD and Adam. There are others you can try like RMSPROP

HINT: The best loss function will probably be Adam, as it is very effective and in most cases is used as default. SGD, in general is not as effective.

```
from tensorflow.keras.losses import CategoricalCrossentropy as CCE

model.compile(optimizer=SGD(learning_rate=0.001),
              loss=CCE,
              metrics=['accuracy'])
history = model.fit(x_train, y_train, epochs=100, batch_size=100, validation_split=0.2)
```

9 Regularization strategies

Regularization strategies are techniques used to prevent overfitting in neural networks by improving training, in a way is like training for a running race by wearing weights, or by running intervals, in this case we force the training to improve generalization. The most common regularization methods:

1. **L1 and L2 Regularization**: Add penalties to the loss function based on the magnitude of model weights:

$$\text{L1: } \lambda \sum_i |w_i|, \quad \text{L2: } \lambda \sum_i w_i^2,$$

   where λ is the regularization strength.

2. **Dropout**: Randomly sets a fraction of neurons to zero during training to reduce reliance on specific neurons and improve generalization.

3. **Data Augmentation**: Increases training data by creating variations, such as flipping, rotating, or scaling images, to improve robustness.

4. **Batch Normalization**: Normalizes layer inputs to reduce internal covariate shifts, stabilizing and regularizing training.
5. **Weight Constraints**: Constrains weights to lie within a specified range, avoiding overly large weights.
6. **Noise Injection**: Adds noise to inputs or weights during training to make the model more robust to small perturbations.

9.1 REGULARIZATION: What should be done in the Assignment

This problem is quite simple to use regularization. You can skip this part.

if you want you can add some dropout layers to your network and see if it improves. As it is very simple network you may don't see an improvement, but keep it. In the next exercise (images) you'll need to add dropouts and Batch Normalizations.

10 Automatic Hyperparameter optimization

Hyperparameter setting is a complex task that is made in Deep Learning to tune the architectures and training. The bigger the architecture the more complex is the task. Can you imagine the complexity to train a network like GPT-4 that has almost 1 Trillion Parameters?

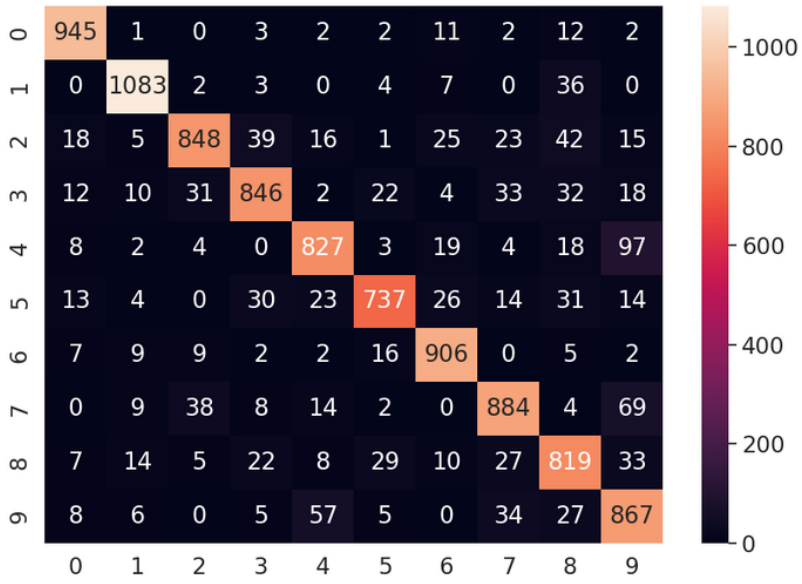


Figure 9: Example of a confussion matrix for the MNIST exercise

To perform Hyperparameter optimization there are some automatized tools that help the activity without too much manual intervention, in any case, we will require large quantities of computing power for this task. Bear in mind that the best combination is an empirical activity some combinations of hyperparameters

are unexpected or dependent on the data used for the problem. You can find more information on this subject in [Agr20]

You can check a version of hyperparameter setting based on the use of the **SCIKIT-SKLEARN** packages using a feature called grid-search hyperparameters.

```
04_MNIST_MLP_Keras_Hyperparameter_setting.ipynb
```

You don't need to use it, but have a look and you'll see how systematic hyperparameter analysis is performed.

11 Activities for the Assignment

Use the MNIST example as a warm-up and try to improve its accuracy. However, don't submit it in the assignment, the one you have to submit is the CIFAR notebook.

You have the Baseline file for the CIFAR using an MLP. [3]

```
040_MLP_CIFAR_KERAS-Baseline.ipynb
```

Perform as many experiments as you want combining the hyperparameters described in the document. Try to get as close as you can to 55% accuracy.

Analyze the confusion matrix 9 to evaluate where are the issues in the classifier.

HINT: The maximum accuracy you can obtain with this problem is close to 55% try as many combinations as you can to get the best accuracy, The accuracy differences between different combinations of activations, optimizers can be quite small, there is not a silver bullet or a combination that is much better.

Create a table like the one described in 1 with the summary of all your experiments.

Exp	Activation	Optimizer	Loss	Hidden Layers	Learning rate	Accuracy	Epochs
E1	ReLU	Adam	Cat-Cross Entropy	2 x 128	0.001	52.5%	50
E2	eLU	SGD	Cat-Cross Entropy	3 x 64	0.01	49.8%	10
E3	Tanh	RMSprop	Cat-Cross Entropy	4 x 64	0.0001	55.4%	100
E4	ReLu	Adam	Cat-Cross Entropy	1 x 256	0.001	54.2%	10
E5	LeakyReLU	Adam	Cat-Cross Entropy	5 x 16	0.001	51.7%	50

Table 1: Table showing the experimental results of the assignment

12 Questions to Answer

1. Which strategy did you follow to set the parameters to optimize your network?

[3] The CIFAR dataset is a difficult dataset for MLP, this is the explanation for the low accuracy results. In the next exercise you will see how using a CNN you will obtain accuracy close to 90% with this same dataset but using CNN

2. What is the best result obtained?
3. What is your learning experience from the hyperparameter setting exercise?

13 Deliverables and submission

Submit the following files with content

- A Notebook with the MLP CIFAR 10 program with the answers to the questions
- A table with the experiments like the one in 1

References

[Kri09] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: (2009), pp. 32–33. URL: `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

[Den12] Li Deng. "The MNIST database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[Tra19] Andrew Trask. *Grokking Deep Learning*. 1st. USA: Manning Publications Co., 2019. ISBN: 1617293709.

[Agr20] Tanay Agrawal. *Hyperparameter Optimization in Machine Learning: Make Your Machine Learning and Deep Learning Models More Efficient*. 1st ed. Apress, 2020. ISBN: 9781484265796.

[SAV20] Eli Stevens, Luca Antiga, and Thomas Viehmann. *Deep Learning with Pytorch*. Manning, 2020. ISBN: 9781633438859.