**Neural net final notes**

**Learning algorithm:** A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P.

If its performance at tasks in T, as masured by P, improves with experience E

IN easy language: in any task if the performance improves with experience, then we would say that the computer program is learning from it’s own experience. Hence learning algorithms.

Experience(E) -> Algorithm -> Task (T) -> performance (P)

Task (T): task (T) is **what we want the algorithm to learn to “DO”**. In ML the task is to process an **example** (composed by a set of features measured from a phenomenon). Denoted by

 where, vector x= [x1, x2,….xn] features.



**Typical tasks of a learning/classification algorithm: face recognition, object recognition, image classification etc.**

**Classification algorithm:** a classification algorithm produces a function



The notation can be read as :

We started the course talking about basic machine learning concepts. You should know the following about this topic:

Function f takes vectors (x) of size n as input and maps them to one of k categories.

That is when y= f(x)

The model/function/learning algorithm assigns a the input defined by x to a category identified by a numeric code y.

**\*\*\* in classification tasks, accuracy is a common performance measure. Like how many was correctly, and how many were incorrectly classified.**

**\*\* accuracy is the proportion of correctly classified examples**

**Some tasks might require other measures such as recall precision, F-1 score etc.**

**Bottom line: The choice of performance measure is nontrivial (not obvious or easy to prove) and may require some thought and experimentation.**

**Typical tasks:**

**classification with missing inputs** => same as classification but the input vector may be missing some features.

**Regression:** A regression learning algorithm **predicts** a numerical value given an input.



Function f takes vectors (x) of size n as input and outputs a real number.

Example:

1. predicts the size of insurance claims given the features of a customer.
2. Predict the price of a stock given previous prices, news etc.
3. Weather, crypto currency.

**Experience E:**

Usually available to the learning algorithm as **Dataset (collection of examples/datapoints).**

**Datasets can be roughly classified into supervised and unsupervised.**

**For unsupervised learning:**

The dataset is unlabeled. The task is to find out the probability distribution that generated the examples or cluster the example.

**For supervised learning:**

The dataset is labeled. The tasks include classification.

**Supervised vs unsupervised**

* There is no formal definition. Example: Given the sepal length, sepal width and petal length from iris dataset predict the petal width.

**Reinforcement learning**

1. The learning algorithm learns from observing an observing an environment.
2. The task performed by the algorithm affects the environment. Example: video game playing.

**Linear Regression**

**Task: map the input vector to a scalar real number.**

From regression problem the learning algorithm provides a function



Which maps vectors x of n size to a real number.

IN linear regression, we assume that the mapping between input vectors are real numbers can be described by a linear function

Shape

Description automatically generated with medium confidence

**Where   is the vector of parameters and b is called bias. The function is called “model”.**

**The ML algorithm will learn to predict the value of y given x :**  ****

**Y ^ means the model is predicted and our best estimation. And the learning algorithm learns the parameter w and b.**

**Experience in Linear regression:**

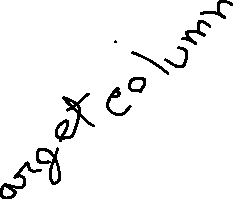
**The experience for linear regression is given by a matrix. Rows are examples/ datapoints and columns are features. One column is used as the target to be predicted.**

**Dataset**

**Shape

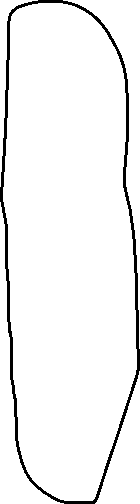
Description automatically generated with medium confidence**

**Training dataset**



**Shape

Description automatically generated with medium confidence**



**Testing dataset**

**Shape

Description automatically generated with medium confidence**

**How well linear regression is working/performance measure:**

**MSE(mean squared error) of the algos predictions is calculated as follows:**

Shape

Description automatically generated with medium confidence

Alternatively,

**Shape

Description automatically generated with medium confidence**

Another way: norm notation

**Shape

Description automatically generated with medium confidence**

**Lower MSE = Better performance**.

**However, the performance we are most interested in is the one that measures the performance under the test data( the data that was not used to train the learning algo). Which also allow us to measure the generalization ability of the algorithm.**

****

**The training algorithm**

1. The algo should be able to **improve it’s prediction** as more examples are presented to it.
2. We measure the performance of the algo by how much it **is able to reduce the mean squared error.**
3. Thus, we **aim to find** a learning algo that **finds the parameters (when the gradient (i.e. derivative) of the MSE is set to 0) that minimize** the **mean squared error**.





**Examples: one feature, one parameter, 10 training datapoints**

Chart, scatter chart

Description automatically generated

Here at w1 when it’s 1.5 there seem to be a straight line and the MSE value is also the lowest at this point which means the derivative is also 0.

1. Explain capacity, overfitting, and underfitting​



**Generalization**: creating a algo with a good performance measure when presented with previously unseen examples. The algo will then generalize the experience from the training to the future sample.

**Objective**: how to translate the good performance to a test data.

**Generalization error vs training error:**

The training error is computed using the **output obtained** by the algorithm when presented with **training samples**.

The generalization error is computed using **the output obtained** by the algorithm when presented with **previously unseen samples**.

Generalization error:

Since we cannot know which samples specifically the training algorithm will see in the future how can we calculate the generalization error?

We assume that **both the training samples and the testing samples are generated by the same data-generating process.**

**formally we assume that data-generalization process produces samples that are independently drawn from each other and identically distributed(iid assumption). We denote this data generating distribution by P-data.**

The main goal of each machine learning model is **to generalize well**. Here **generalization** defines the ability of an ML model to provide a suitable output by adapting the given set of unknown input. It means after providing training on the dataset, it can produce reliable and accurate output. Hence, the underfitting and overfitting are the two terms that need to be checked for the performance of the model and whether the model is generalizing well or not.

Before understanding the overfitting and underfitting, let's understand some basic term that will help to understand this topic well:

* **Signal:** It refers to the true underlying pattern of the data that helps the machine learning model to learn from the data.
* **Noise:** Noise is unnecessary and irrelevant data that reduces the performance of the model.
* **Bias:** Bias is a prediction error that is introduced in the model due to oversimplifying the machine learning algorithms. Or it is the difference between the predicted values and the actual values.
* **Variance:** If the machine learning model performs well with the training dataset, but does not perform well with the test dataset, then variance occurs.

1.1 You should know what is capacity and what parameters of a deep neural network can be modified to increase or decrease the capacity.

1.2 You should know what is overfitting, what causes it, and how it affects the performance of the model.

1.3 You should know what is underfitting, what causes it, and how it affects the performance of the model.

**Underfitting: During the generalization procedure, if our learning algorithm is too simple to learn more complex distributions, then it is an underfitting model.**

Underfitting occurs when our machine learning model is not able to capture the underlying trend of the data. To avoid the overfitting in the model, the fed of training data can be stopped at an early stage, due to which the model may not learn enough from the training data. As a result, it may fail to find the best fit of the dominant trend in the data.

In the case of underfitting, the model is not able to learn enough from the training data, and hence it reduces the accuracy and produces unreliable predictions.

An underfitted model has **high bias and low variance**.

**Overfitting: During the generalization procedure, if our learning algorithm ends-up learning the specific realization of the distribution ( too good at understanding complex distributions) instead of the distribution itself. Then Overfitting model.**

Overfitting occurs when our [machine learning](https://www.javatpoint.com/machine-learning) model tries to cover all the data points or more than the required data points present in the given dataset. Because of this, the model starts caching noise and inaccurate values present in the dataset, and all these factors reduce the efficiency and accuracy of the model. The overfitted model has **low bias** and **high variance.**

**Model Capacity: Given by the number and complexity of functions it can use to learn the data-generating distribution. (i.e: how many formulas or how many specific models our model can take)**

**increasing the number of formulas available to our model = increasing capacity = more likely to overfit**

**limit/reduce the number of formulas available to our model = Less capacity = more likely to underfit.**

**Model capacity**

**Example: linear regression can choose from any linear function.**

**We can increase the capacity of linear regression by allowing it to choose from linear and quadratic functions:**

****

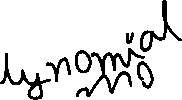


**or from high degree polynomials.**

****

**Diagram, shape, rectangle

Description automatically generated**



Diagram

Description automatically generated

How to determine the best model:

Chart

Description automatically generated

Here at left side we can see the error and the right side we can see that the capacity and there is zero where there is no model or no function. so at the beginning we can see that the generalization gap and the error when we do not have any model or function is highest well that time there was no generalization error. since the capacity increases our error will go down but our generalization gap or generalization error will also start going up. before our generalization gap and error increases we need to find the optimal spot where our model will be best fit and not an underfit or not an overfitting zone, that is our optimal capacity.

1. Explain regularization​

2.1 You should know what is regularization, what is the purpose of regularization, and give on example of regularization.

Regularization is a technique used to reduce the errors by fitting the function appropriately on the given training set and avoid overfitting. ( controlling the capacity of the model to be limited by assigning preferences to the functions by forcing the learning algo to choose small parameters.)

**Formally: Regularization is any modification we make to a learning algo to reduce the generalization error but not the training error.**

To do so we can change the performance metric to include a term related to the size of the parameters.



The commonly used regularization techniques are : 

1. L1 regularization
2. L2 regularization
3. Dropout regularization

Example for the 9th degree polynomial:

Diagram

Description automatically generated

We can set lambda to very large number and that how we can control the capacity.

1. Explain hyperparameters and validation sets​
   1. You should know what is a hyperparameter?

Hyperparameters are model parameters that are not learned during the training. This are the things that we need to choose before the ml algo is start running and learn anything.

Example: In regression, the hyper parameters is the degree of the polynomials that are allowed.

The number of layers and number of nurons in each of the layers are the hyper parameters.

How to choose hyper parameters: validation sets

* 1. You should know what is a validation set and how it differs from testing and training sets.

A validation set is a subset of the training dataset, which allows us to change hyperparameters outside the training algorithm that finds the parameters.

\*\*\* k-fold cross validation is a typical way of dividing training dataset.

The model must be assessed regularly to be trained, which is exactly what the validation set is for. We may determine [**how accurate a model is**](https://deepchecks.com/glossary/machine-learning-model-accuracy/) by computing the loss it produces on the validation set at each given point. This is what training is all about.

What is a validation dataset? In simple terms:

* **A validation dataset is a collection of instances used to fine-tune a classifier’s hyperparameters**

The number of hidden units in each layer is one good analogy of a hyperparameter for machine learning neural networks. It should have the same probability distribution as the training dataset, as should the testing dataset. When a classification variable must be updated, a validation dataset in machine learning, including the test and training datasets, is required to avoid overfitting.

If the most appropriate classifier for the problem is sought, the training dataset is used to train the various candidate classifiers, the [**data validation in machine learning**](https://deepchecks.com/how-to-validate-your-ml-model-before-deploying-it-into-production/) is used to compare their performances and choose which one to employ, and the test dataset is used to acquire performance characteristics like as F-measure, sensitivity, accuracy or specificity.

The validation dataset is a hybrid: it is training data that is used for testing, but it is not included in either the low-level training or the final testing. Early stopping is a technique in which the candidate models are iterations of the same network, and training stops when the error on the validation set develops, choosing the previous model – the one with the least error.

1. Explain maximum likelihood estimators​

Point estimator: when we train a learning algorithm, we are actually estimating its parameters. finding the best parameter is called point estimation. it is called points estimation because we are finding one estimate rather than multiple ones.

why do we need point estimation: to formally analyze underfitting and overfitting. we first need to define point estimation.

Point estimators:

Points estimator are the functions that find the point estimate of parameter  

Let,  be the set of m, i.i.d. data points. then the point estimator is any function that finds an estimate of the parameters. i.e.

Shape

Description automatically generated with medium confidence

Function estimation:

**parameter estimation is actually function estimation**. since the function is defined by its parameters, we can use points parameters estimation as function estimation.

**Estimator bias**

**the bias of an estimator is a measure of how much the estimate will deviate from the true value of the parameter we are estimating. formally the bias is given by-**

**Shape

Description automatically generated with medium confidence**

**where the expectation is taken over the data point. ideally our estimators are unbiased, they have bias equal to 0.**

Estimator variance

the variance is a measure of how much our estimate will change when we resample the data.

for example:

For two datasets taken from the same data generating process, how different our parameter estimation is likely to be for the two of them?

Another way to see it is, how much will the estimate deviate from its expected value? we denote the variance with

Shape

Description automatically generated with medium confidence

**Bias versus variance**

**ideally we would like our estimators to have low bias and low variance. one common way of achieving this is by using cross validation (i.e., train the model using multiple subsets of the training data set.**

**another common way is to minimize the mean squared error.**

Shape

Description automatically generated with medium confidence

Example: bias versus variance and its relation on over or underfitting when using MSE

Diagram

Description automatically generated with medium confidence

Here when our generalization error starts to increase our performance starts to go down which is why the moment the generalization error will start going up again we should stop and then we will have our optimal capacity. note that when we have the generalization error at the lowest our variance and capacity are also at a stable point capacity will go down at the same time the variance will go up we want to keep the variance low but

When we will decrease the bias the variance will increase.

* 1. You should know what is a maximum likelihood estimator.

[Maximum Likelihood Estimation](https://analyticsindiamag.com/a-comprehensive-guide-to-maximum-likelihood-estimation-and-bayesian-estimation/) (MLE) is a probabilistic based approach to determine values for the parameters of the model. Parameters could be defined as blueprints for the model because based on that the algorithm works. MLE is a widely used technique in machine learning, [time series](https://analyticsindiamag.com/general-overview-of-time-series-data-analysis/), panel data and discrete data. **The motive of MLE is to maximize the likelihood of values for the parameter to get the desired outcomes.**

The likelihood function measures the extent to which the data provide support for different values of the parameter. It indicates how likely it is that a particular population will produce a sample. For example, if we compare the likelihood function at two-parameter points and find that for the first parameter the likelihood is greater than the other it could be interpreted as the first parameter being a more plausible value for the learner than the second parameter.

* 1. You should know why we use macimum likelihood etimators (i.e., what is the advantage)

The maximization of the likelihood estimation is the main objective of the MLE. **Example**: Consider there is a binary classification problem in which we need to classify the data into two categories either 0 or 1 based on a feature called “salary”.

So MLE will calculate the possibility for each data point in salary and then by using that possibility, it will calculate the likelihood of those data points to classify them as either  0 or 1. It will repeat this process of likelihood until the learner line is best fitted. This process is known as the maximization of likelihood.

MLE is the base of a lot of supervised learning models, one of which is [Logistic regression](https://analyticsindiamag.com/beginners-guide-to-logistic-regression-in-python/). Logistic regression maximum likelihood technique to classify the data. Let’s see how Logistic regression uses MLE. Specific MLE procedures have the advantage that they can exploit the properties of the estimation problem to deliver better efficiency and numerical stability. These methods can often calculate explicit confidence intervals. The parameter “solver” of the logistic regression is used for selecting different solving strategies for classification for better MLE formulation.

Maximum likelihood estimator

* A principle to design good estimators
* Consider the set of samples  

drawn from the true data generating process distribution



Let,  

be a function that maps the input X to a real number estimating the true probability



The maximum likelihood estimator for   is

Shape

Description automatically generated with medium confidence

To improve the computational stability, we take the logarithm:



Alternatively, we can divide by M and express this formula as:

****

**Intuition:**

the maximum likelihood estimator can be interpreted as**: finding a set of parameters that maximize the difference** between the probability distribution learned by the model and the probability distribution of the training data.



Since the first term is constant [ logp^ data(x)] we can **just minimize the second term** which is equivalent the maximum likelihood estimator: which **means P model equals be data.**

**Here, KL is the “Kullback leibler” divergence- That measures the difference between 2 distributions which is usually Called the “cross entropy”.**

**Conditional log-likelihood:**

**in supervised learning algorithms we are not interested in learning the probability distribution of the data-generating process. instead we are interested in estimating the conditional probability**

****

the conditional maximum likelihood estimator is given by

****

For example : linear regression

it can be shown that **minimizing the mean squared error is equivalent = to maximizing the conditional log likelihood.**

**The main reasons to use maximum likelihood estimators:**

1. **the maximum likelihood estimator can be shown to be the best estimator asymptotically, as the number of examples approaches Infinity.**
2. **the true distribution must lie within the model family. there is a unique set of parameters that describe the original data**
3. Why deep learning?

5.1 You should know what are the limitations of traditional machine learning

The curse of dimensionality

The total number of possible values of x is much larger than the total number of examples. How do we predict the value of X when there are no examples within its cell? ML does a bad job here.

A picture containing clipart

Description automatically generated

One dimensional 2 dimensional 3d

We can see some gap here a series of examples in one-dimensional, in 2 dimensional add 3 dimensional, if these gaps are filled with another examples/data in the future we probably won’t be able to reduce the generalization error. This is where ML has limitation. But deep learning has solution.

In machine learning we make our assumption that if our model doesn't change, the output of our function will not change as well. In deep learning we break that assumption.

\*\*\* local constancy and smoothness regularization: associated to the curse of dimensionality machine learning algorithms force their models to change slowly within a neighborhood. this limit their generalization ability when there are not enough samples in every region.

5.2 You should know how deep learning addresses those limitations.

Deep learning gives us lower generalization error. Also, machine learning algorithms have trouble recognizing manifolds ( a manifold is a set of points connected in a lower dimensional space that is embedded in a higher dimensional space)). Deep learning can do it easily.

**Deep learning versus machine learning**

**AL <= ML <= Representation learning( not covered in syllabus) <= Deep learning**

**Diagram

Description automatically generated**

**Diagram

Description automatically generated**

Deep learning need much less resources than machine learning needs to create some classification.

**Convolutional Neural Networks**

After covering machine learning basics, we started talking about deep feed forward networks in general. We will skip that part in this exam. Hence, the next topic is CNNs. ​

1. Describe a CNN​

A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

Biology behind cnn: The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

* 1. You should know what is the difference between a traditional deep feedforward network and a convolutional network.

The main difference between classic feedforward neural networks and deep neural networks is the number of hidden layers. If a network has two hidden layers that's probably already a deep neural network.

Classic feedforward networks: feedforward networks are neural networks where the input was directly toward the output.

recurrent networks: recurrent networks have feedback layers where the input may flow in cycles.

A feed-forward network connects every pixel with each node in the following layer, ignoring any spatial information present in the image. By contrast, a convolutional architecture looks at local regions of the image. A ConvNet is able to **successfully capture the Spatial and Temporal dependencies** in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and the reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

Convolutional neural network is better than a feed-forward network since **CNN has features parameter sharing and dimensionality reduction**. Because of parameter sharing in CNN, the number of parameters is reduced thus the computations also decreased. The main intuition is the learning from one part of the image is also useful in another part of the image. Because of the dimensionality reduction in CNN, the computational power needed is reduced.

For math please check the “ 02 -deep feed forward networks slide”

A deep feedforward network is a feedforward network with multiple hidden layers. deep feedforward networks can be interpreted as nested function where each function represents a layer  

Example: learning XOR. Design A neural network that can act as an xor gate

  X1 x2 y

0 0 0

1 0 1

0 1 1

1 1 0

Training dataset : (experience)



We are only interested in minimizing the training error we're not interested in the generalization error in this example. the main purpose is to learn the math notation and how that relates back to the network diagram.

Learning Xor: math notation

the neural network contains two functions chained together

1. Hidden layer function (nonlinear function)



1. second layer(linear function)

[essentially linear regression applied to the output of the first function]



Learning Xor: hidden layer function

1. what type of function should  A black rectangle with a black background

   Description automatically generated with low confidenceBe?

if we choose linear the whole network becomes linear and can be represented as a single linear function.

1. we instead choose an activation function of an affine transformation





learning XOR: Activation function:

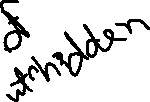
Chart, line chart

Description automatically generated

If the number is positive it gonna fire, if the number is negative it will remain 0.

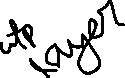
The activation function is applied elementwise





A common choice is the rectified linear unit ( ReLU)





The whole network can be defined by





Since this is a simple problem we can manually brute force the parameters:

A picture containing text, clock

Description automatically generated  A picture containing text, clock

Description automatically generated  Diagram

Description automatically generated  

We can calculate the outputs for all possible inputs at once:

Inputs ,

Table

Description automatically generated

Learning XOR: batch output computation

we can start by multiplying the inputs with the parameter matrix, here the parameter matrix is W





Text, logo

Description automatically generated A picture containing text, clock

Description automatically generated  A picture containing text, clock

Description automatically generated

A picture containing text, clock

Description automatically generated

At c parameter



A picture containing text, clock, watch, gauge

Description automatically generated   **= C= [0 -1]**

Add c- transpose to each row of XW.

XW + C^T =  A picture containing text, clock

Description automatically generated

Now put the [ XW + C ] into “ max” activation function. We get :

A picture containing text, clock

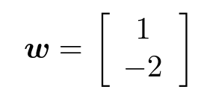
Description automatically generated [ Remember from activation function graph: If the number is positive it gonna fire, if the number is negative it will remain 0.]

Space transformation:

Chart, box and whisker chart

Description automatically generated

Now multiplying W-transpose with our output form activation function “max”

= [1 -2]

[1 -2]\*  A picture containing text, clock

Description automatically generated =  A picture containing text, clock

Description automatically generated

1. Describe the convolution operation​
   1. You should be able to describe the convolution operation

An image is nothing but a matrix of pixel values, right? So why not just flatten the image (e.g. 3x3 image matrix into a 9x1 vector) and feed it to a Multi-Level Perceptron for classification purposes? Uh.. not really.

In cases of extremely basic binary images, the method might show an average precision score while performing prediction of classes but would have little to no accuracy when it comes to complex images having pixel dependencies throughout.

A ConvNet is able to **successfully capture the Spatial and Temporal dependencies** in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and the reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

Convolutional neural network

specialized to process data with grid-like structure. such as images( 2D), time-series data (1D)

**the main difference with other feedforward networks: uses a convolution instead of a matrix multiplication in at least one of its layer**. Pooling layers are also common.

**Deconvolution operation**

Intuitively, the convolution operation takes as input a target function and a filter function and outputs a function that describes how the filter affects the target function.

2.2 You should be able to compute the convolution operation given a small example

**Convolution in deep learning**

Fully defined for matrices, both the target and filter are matrices. Intuitively, the filter matrix acts as a magnifying glass that is applied to a small section of the input matrix.

A picture containing text

Description automatically generated

* **Padding.** Padding expands the input matrix by adding fake pixels to the borders of the matrix. This is done because convolution reduces the size of the matrix. For example, a 5x5 matrix turns into a 3x3 matrix when a filter goes over it.
* **Striding.** It often happens that when working with a convolutional layer, you need to get an output that is smaller than the input. One way to achieve this is to use a pooling layer. Another way to achieve this is to use striding. The idea behind stride is to skip some areas when the kernel slides over: for example, skipping every 2 or 3 pixels. It reduces spatial resolution and makes the network more computationally efficient.

Padding and striding can help process images more accurately.

A picture containing text, electronics

Description automatically generated

3 row x 4 colum \* 2x2 = 2 row x 3 column

Formally, convolution for deep neural network is defined by



equivalently,



I is the input matrix. K is the kernel or filter .

1. Describe the advantage of using the convolution operation​.

3.1 You should know why we use the convolution operation

* **Sparse interaction\*\*\*: Colonel is smaller than the input, reduces number of competition, meaningful features of the image can be recognized by only analyzing a few pixels at a time.**
* **Parameter sharing\*\*\*: Colonel values are applied at every position of the input, reduces the amount of parameters we need to store.**

**Diagram

Description automatically generated**

* **Equivariant representation\*\*\*: convolution reduces the receptive field. Similar to parameter sharing**

**To sum up, convolutional neural networks are an awesome tool for computer vision and similar areas because of their ability to recognize features in raw data.**

**They can recognize the connections between different pixels in training data and use this information to engineer features on their own, building up from low-level (edges, circles) to high-level (faces, hands, cars).**

**The problem is the features can become rather incomprehensible to humans. Moreover, a wild pixel in the image can sometimes lead to new surprising results.**

### Computational efficiency

CNN, due to the procedure of convolution, are much more computationally efficient than regular neural networks. CNN uses parameter sharing and dimensionality reduction, which makes models easy and quick to deploy. They can be optimised to run on any device, even on smartphones.

### High accuracy

The current state-of-the-art NNs in image classification are not convolutional nets, for example, in image transformers. However, CNNs have now been dominating for a very long time in most cases and tasks regarding image and video recognition and similar tasks. They usually show [higher accuracy](https://www.researchgate.net/post/CNN-classification-v-s-Artificial-NN-classification-why-there-is-a-significant-difference-in-accuracy) than non-convolutional NNs, especially when there is a lot of data involved

Regular artificial neural networks do not scale very well. For example, in CIFAR, a dataset that is commonly used for training computer vision models, the images are only of size 32x32 px and have 3 color channels. That means that a single fully-connected neuron in a first hidden layer of this neural network would have 32x32x3 = 3072 weights. It is still manageable. But now imagine a bigger image, for example, 300x300x3. It would have 270,000 weights (training of which demands so much computational power)!

A huge neural network like that demands a lot of resources but even then remains prone to overfitting because the large number of parameters enable it to just memorize the dataset.

CNNs use parameter sharing. All neurons in a particular feature map share weights which makes the whole system less computationally intense.

For real-life tasks, convolution is usually performed in 3D. The majority of images have 3 dimensions: height, width and depth, where depth corresponds to color channels (RGB). So the convolutional filter needs to be 3-dimensional as well.

Example: There are multiple filters in a convolutional layer and each of them generates a filter map. Therefore, the output of a layer will be a set of filter maps, stacked on top of each other.

For example, padding and passing a 30x30x3 matrix through 10 filters will result in a set of 10 30x30x1 matrices. After we stack these maps on top of each other, we will get a 30x30x10 matrix.

This is the output of our convolutional layer.

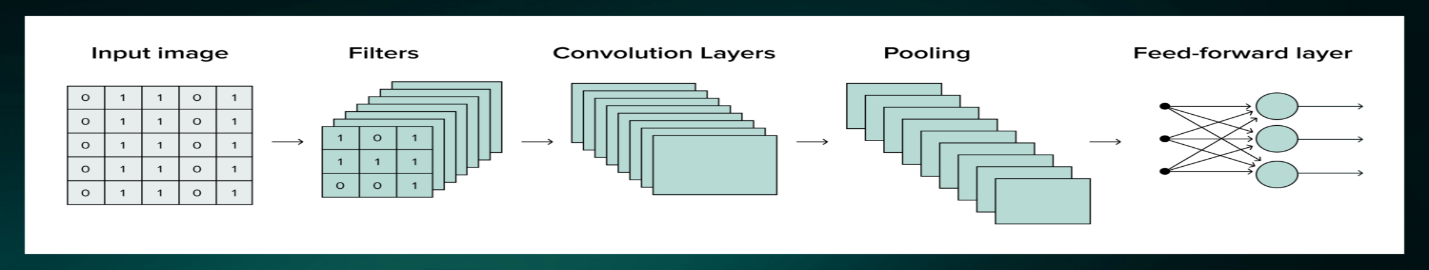
The process can be repeated: CNNs usually have more than one convolutional layer.

### layers of CNN

i)A convolutional layer is responsible for recognizing features in pixels.

ii)A pooling layer is responsible for making these features more abstract.

iii)A fully-connected layer is responsible for using the acquired features for prediction.



* 1. You should knwo what type of data is best suited for convolution operations and why.

specialized to process data with grid-like structure. such as images( 2D), time-series data (1D). They can recognize the connections between different pixels in training data and use this information to engineer features on their own, building up from low-level (edges, circles) to high-level (faces, hands, cars).

1. Pooling

We’ve already described how convolution layers work [above](https://serokell.io/blog/introduction-to-convolutional-neural-networks#how-does-a-cnn-work%3F). They are at the center of CNNs, enabling them to autonomously recognize features in the images.

But going through the convolution process generates a large amount of data, which makes it hard to train the neural network. To compress the data, we need to go through pooling.

4.1 You should be able to describe what is pooling and compute the value in a small example

The convolution layer is formed by three stages :

Diagram

Description automatically generated

1. convolution stage: affine transform is done in this stage, the convolution stage performs the convolution operation in parallel.
2. detector stage: applies the activation function
3. pooling stage: modifies the output of the activation function.

The pooling function “summarizes” the output from the activation function in a small neighborhood of the input.more formally it replaces the output of a layer at a certain location with a summary statistics of the nearby outputs.

Table

Description automatically generated

Notice that the 4X4 matrix has four different sections inside. from each section we take the highest value and create a 2X2 Max pool matrix. bye

4.2 You should know what is the kernel/filter

A filter **acts as a single template or pattern, which, when convolved across the input, finds similarities between the stored template & different locations/regions in the input image.**

In Convolutional neural network, the kernel is nothing but **a filter that is used to extract the features from the images**. The kernel is a matrix that moves over the input data, performs the dot product with the sub-region of input data, and gets the output as the matrix of dot products.

Kernel vs Filter  
  
The dimensions of the kernel matrix is how the convolution gets it's name. For example, in 2D convolutions, the kernel matrix is a 2D matrix. **A filter however is a concatenation of multiple kernels, each kernel assigned to a particular channel of the input**.

* 1. You should be able to explain why we use pooling after a convolutional layer
* Pooling is used to make the output of the convolutional layer approximately invariant to small changes in the input. This helps us detect image features independently of where they are located in the image. for example, if we are classifying images of dogs, we only need to know if a dog is present. we do not care where the dog is exactly in the image.

To learn about “combining multiple convolutional layers”, downsampling and ResNet, check the slides “ 04- pooling” { downsampling and Resnet are examples of convolutional neural network.

1. Stochastic gradient descent

**5.1 You should be able to explain the intuition behind stochastic gradient descent.**

When we cannot directly optimize our performance measure and we need to minimize a loss function, we will use stochastic gradient dexscent. This is one of the mo

st commonly used algorithm for deep learning(many variants, same principle). it uses the gradient of the network at the input (would you sculpted using backpropagation). and this algorithm is iterative.

Chart

Description automatically generated with low confidence

Timeline

Description automatically generated

Text, letter

Description automatically generated

Small beauty batches/ mini batches improves our testing error and larger batches provide a more accurate estimate of the gradient however we still want mini bed size since it improves the testing error. We want less iteration. More iteration means more computationally expensive. Which is why we need big batches.

The loss function depends on the gradient value. If the gradient is big our change will be big as well. ( change depends on learning rate)

Since this algorithm is iterative each time we use different samples Once we are done with a total set of samples we call it an epoch.

Sample of mini batch of m examples is important,

**5.2. You should be able to explain what is a batch and a mini-batch**

The batch means how many samples we want to look at to update the parameters. we could take all the samples at the same time and then update the parameters.

If we use larger batches it will be more accurate but it also becomes computationally invisible too. which is why we want mini batches.

**5.3 You should be able to explain how batches are used in stochastic gradient descent**

mini batches also improve our testing error although the gradient is more accurate with the larger batches the generalization of the model may not be as good as if we use the small batch. however if we go very small batches then our training time increases a lot.

Because if our end value of sample of mini batches, m = 1, then We would be updating the parameters for each sample in our data set. if we have millions of points in our datasets then this is one million or several million iteration. and calculating these gradients is computationally expensive. so we want this batch to be relatively large so that we don't have to do so many iteration .

Larger batches provide a more accurate estimate of the gradient. small batches offer a regularization effect (i.e. improve testing error). the small bed size introduces noise into the computation of the gradient expected value. however a small batch size may drastically increase the training time.

When all samples in the training data set have been included in one mini batch, we have completed an epoch. the number of epoch is an important parameter. after a certain number of epoch the testing error stops decreasing.

1 epoch means that your stochastic gradient descent algorithm has gone through all the training samples in your data set. and usually we go through multiple epochs. Because every time the mini batches will sample different sets from the training data set. the update to the parameters will be different every time so we can keep going and iterating not only through the mini batches but also through

the data set and that can increase our performance and decrease our training error. Which is why we should increase the epochs.

Early Stopping

Chart

Description automatically generated

.

**Recurrent Neural Networks**

Finally, we covered RNNs.

1. Motivation for RNNs

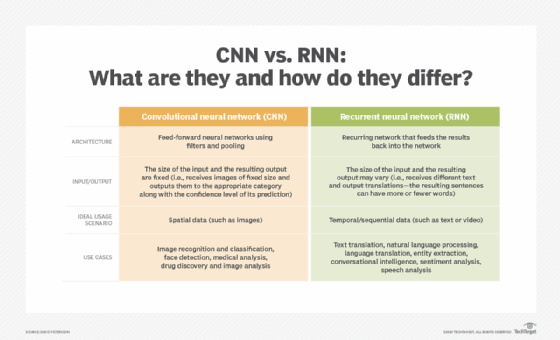
RNN are specialized for processing sequential data. such as time series, sentences, blockchain blocks etc. RNNs shares parameters between multiple neurons.

Example: in this sentences “I went to Nepal in 2009” and “ in 2019, I went to Nepal", an RNN shares parameters between the input for the first and last words. This allows them to identify the year of travel regardless of its position in the sentence.

* 1. You should be able to explain when RNNs are better suited than CNNs

The main difference between a CNN and an RNN is the **ability to process temporal information** — data that comes in sequences, such as a sentence. Recurrent neural networks are designed for this very purpose, while convolutional neural networks are incapable of effectively interpreting temporal information.

CNNs are commonly used in solving problems related to spatial data, such as images.**RNNs are better suited to analyzing temporal, sequential data, such as text or videos**. A CNN has a different architecture from an RNN. CNNs are "feed-forward neural networks" that use filters and pooling layers, whereas RNNs feed results back into the network



Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

**RNN's with recursion between hidden layers and previous outputs are less powerful but easier to train the training data provides the ideal values for the outputs O. so we can train each time step in parallel.**

**Type 3**

Diagram

Description automatically generated

In some cases, we are only interested in the last output of the RNN.

Bi-directional RNNS

Diagram

Description automatically generated

Text, letter

Description automatically generated

1. **Describe RNNs**
   1. **You should be able to describe a computational graph**
2. A **recurrent neural network** (**RNN**) is a class of [artificial neural networks](https://en.wikipedia.org/wiki/Artificial_neural_network) where connections between nodes can create a cycle, allowing output from some nodes to affect subsequent input to the same nodes. This allows it to exhibit temporal dynamic behavior. Derived from [feedforward neural networks](https://en.wikipedia.org/wiki/Feedforward_neural_networks), RNNs can use their internal state (memory) to process variable length sequences of inputs.[[1]](https://en.wikipedia.org/wiki/Recurrent_neural_network#cite_note-1)[[2]](https://en.wikipedia.org/wiki/Recurrent_neural_network#cite_note-2)[[3]](https://en.wikipedia.org/wiki/Recurrent_neural_network#cite_note-3) This makes them applicable to tasks such as unsegmented, connected [handwriting recognition](https://en.wikipedia.org/wiki/Handwriting_recognition)[[4]](https://en.wikipedia.org/wiki/Recurrent_neural_network#cite_note-4) or [speech recognition](https://en.wikipedia.org/wiki/Speech_recognition).[[5]](https://en.wikipedia.org/wiki/Recurrent_neural_network#cite_note-sak2014-5)[[6]](https://en.wikipedia.org/wiki/Recurrent_neural_network#cite_note-liwu2015-6) Recurrent neural networks are theoretically [Turing complete](https://en.wikipedia.org/wiki/Turing_complete) and can run arbitrary programs to process arbitrary sequences of inputs.[[7]](https://en.wikipedia.org/wiki/Recurrent_neural_network#cite_note-7)
3. The term "recurrent neural network" is used to refer to the class of networks with an [infinite impulse response](https://en.wikipedia.org/wiki/Infinite_impulse_response), whereas "[convolutional neural network](https://en.wikipedia.org/wiki/Convolutional_neural_network)" refers to the class of [finite impulse](https://en.wikipedia.org/wiki/Finite_impulse_response) response. Both classes of networks exhibit temporal [dynamic behavior](https://en.wikipedia.org/wiki/Dynamic_system).[[8]](https://en.wikipedia.org/wiki/Recurrent_neural_network#cite_note-8) A finite impulse recurrent network is a [directed acyclic graph](https://en.wikipedia.org/wiki/Directed_acyclic_graph) that can be unrolled and replaced with a strictly feedforward neural network, while an infinite impulse recurrent network is a [directed cyclic graph](https://en.wikipedia.org/wiki/Directed_cyclic_graph) that can not be unrolled.
4. Both finite impulse and infinite impulse recurrent networks can have additional stored states, and the storage can be under direct control by the neural network. The storage can also be replaced by another network or graph if that incorporates time delays or has feedback loops. Such controlled states are referred to as gated state or gated memory, and are part of [long short-term memory](https://en.wikipedia.org/wiki/Long_short-term_memory) networks (LSTMs) and [gated recurrent units](https://en.wikipedia.org/wiki/Gated_recurrent_unit). This is also called Feedback Neural Network (FNN).

**2.2 You should be able to describe how different RNN architectures can be used for different tasks as well as their advantages and disadvantages.**

**Different RNN architechtures: Gated RNN, Deep RNNs, LSTM**

#### Advantages:

* RNN can process inputs of any length.
* An RNN model is modeled to remember each information throughout the time which is very helpful in any time series predictor.
* Even if the input size is larger, the model size does not increase.
* The weights can be shared across the time steps.
* RNN can use their internal memory for processing the arbitrary series of inputs which is not the case with feedforward neural networks.

#### Disadvantages:

* Due to its recurrent nature, the computation is slow.
* Training of RNN models can be difficult. Requires the computational gradient, forward and back propagation.
* If we are using relu or tanh as activation functions, it becomes very difficult to process sequences that are very long. Loss function
* Prone to problems such as exploding and gradient vanishing. Bi-directional, encoder decoder, deep rnn, lstd (probably) but not the gated RNN .

Prone to exploding and gradient vanishing : In an RNN, the same weight matrix is being multiplied with inputs and previous outputs, and hence the gradients explode and vanish. Also, the path for gradient flow is very long from the last iteration of the RNN to the first. This means the amount of contextual information the RNN can maintain in memory before gradients explode or vanish, is small. Hence the memory of vanilla RNNs is low, and the size of the reference window (the number of words before the current word, from which RNNs can draw contextual information ) is small.

1. Describe a encoder-decoder architecture and what are its advantages.

**Encoder-decoder sequence-to-sequence architectures**:

**The previous RNNS we covered, deals with variable length inputs but fixed sized outputs.**

**encoder decoder are an ENS allows us to have variable length inputs and outputs. this is useful for language translation, speech recognition, question answering etc.**

**In this case, the encoder takes the input sequence and outputs a fixed= length vector, call the context. the decoder takes the context and outputs a variable length sequence.**

The standard approach to handling this sort of data is to design an *encoder-decoder* architecture ([Fig. 10.6.1](https://d2l.ai/chapter_recurrent-modern/encoder-decoder.html#fig-encoder-decoder)) consisting of two major components: an *encoder* that takes a variable-length sequence as input, and a *decoder* that acts as a conditional language model, taking in the encoded input and the leftwards context of the target sequence and predicting the subsequent token in the target sequence.

Input => encoder => state => decoder => output

Diagram

Description automatically generated

The Encoder-Decoder architecture with recurrent neural networks has become an effective and standard approach for both neural machine translation (NMT) and sequence-to-sequence (seq2seq) prediction in general.

The key benefits of the approach are the ability to train a single end-to-end model directly on source and target sentences and the ability to handle variable length input and output sequences of text.

For a specific problem: vocabulary problem

An Encoder-Decoder architecture was developed where an input sequence was read in entirety and encoded to a fixed-length internal representation.

A decoder network then used this internal representation to output words until the end of sequence token was reached. LSTM networks were used for both the encoder and decoder.

*The idea is to use one LSTM to read the input sequence, one timestep at a time, to obtain large fixed-dimensional vector representation, and then to use another LSTM to extract the output sequence from that vector*

The final model was an ensemble of 5 deep learning models. A left-to-right beam search was used during the inference of the translations.

*Each time the proposed model generates a word in a translation, it (soft-)searches for a set of positions in a source sentence where the most relevant information is concentrated. The model then predicts a target word based on the context vectors associated with these source positions and all the previous generated target words.*

**Deep RNNs**

Diagram

Description automatically generated

**Gated RNNS**

A screenshot of a computer

Description automatically generated with medium confidence

**The gated ordinance add leaky units which add parts in the network with derivatives that neither vanish or explode the main idea is to add connection weights**

**Long short-Term Memory ( LSTM)**

Diagram

Description automatically generated

**Long short-term memory** (**LSTM**)[[1]](https://en.wikipedia.org/wiki/Long_short-term_memory#cite_note-lstm1997-1) is an [artificial neural network](https://en.wikipedia.org/wiki/Artificial_neural_network) used in the fields of [artificial intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence) and [deep learning](https://en.wikipedia.org/wiki/Deep_learning). Unlike standard [feedforward neural networks](https://en.wikipedia.org/wiki/Feedforward_neural_network), LSTM has feedback connections. Such a [recurrent neural network](https://en.wikipedia.org/wiki/Recurrent_neural_network) (RNN) can process not only single data points (such as images), but also entire sequences of data (such as speech or video). For example, LSTM is applicable to tasks such as unsegmented, connected [handwriting recognition](https://en.wikipedia.org/wiki/Handwriting_recognition),[[2]](https://en.wikipedia.org/wiki/Long_short-term_memory#cite_note-graves2009-2) [speech recognition](https://en.wikipedia.org/wiki/Speech_recognition),[[3]](https://en.wikipedia.org/wiki/Long_short-term_memory#cite_note-sak2014-3)[[4]](https://en.wikipedia.org/wiki/Long_short-term_memory#cite_note-liwu2015-4) [machine translation](https://en.wikipedia.org/wiki/Machine_translation),[[5]](https://en.wikipedia.org/wiki/Long_short-term_memory#cite_note-GoogleTranslate-5)[[6]](https://en.wikipedia.org/wiki/Long_short-term_memory#cite_note-FacebookTranslate-6) robot control,[[7]](https://en.wikipedia.org/wiki/Long_short-term_memory#cite_note-mayer2006-7)[[8]](https://en.wikipedia.org/wiki/Long_short-term_memory#cite_note-OpenAIhand-8) video games,[[9]](https://en.wikipedia.org/wiki/Long_short-term_memory#cite_note-OpenAIfive-9)[[10]](https://en.wikipedia.org/wiki/Long_short-term_memory#cite_note-alphastar-10) and healthcare.[[11]](https://en.wikipedia.org/wiki/Long_short-term_memory#cite_note-decade2022-11)

The name of LSTM refers to the analogy that a standard RNN has both "long-term memory" and "short-term memory". The connection weights and biases in the network change once per episode of training, analogous to how physiological changes in synaptic strengths store long-term memories; the activation patterns in the network change once per time-step, analogous to how the moment-to-moment change in electric firing patterns in the brain store short-term memories.[[12]](https://en.wikipedia.org/wiki/Long_short-term_memory#cite_note-12) The LSTM architecture aims to provide a short-term memory for RNN that can last thousands of timesteps, thus "**long** short-term memory".[[1]](https://en.wikipedia.org/wiki/Long_short-term_memory#cite_note-lstm1997-1)

LSTM addresses the issues of the RNN. It does this by maintaining a cell state, which is the state at any given time. The main advantage of using a cell state is that the cell state can be maintained and updated with minimal computation and processing, and the cell state creates a gradient highway for gradients to backpropagate, hence avoiding the vanishing and exploding gradient problem.

**Labs**

1. You should be able to apply the concepts from CNN and ML basics to train a CNN and/or RNN  
   model given a certain data set.
2. Given the results of training and testing, you should be able to change the model and training parameters to improve its performance.
3. You should be able to explain why certain changes to the model and training parameters improve or degrade performance.

# Convolutional Neural Networks Labs

The objective of this lab is to give you hands-on experience designing a convolutional neural network (CNN). These instructions will provide you with a basic CNN architecture, and provide you with default training settings. This sample model has a mediocre performance. Your job is to use your knowledge about model capacity, under/over fitting, and deep network architectures to improve the performance, i.e., reduce the testing error.

## Code test drive

Before we start tinkering with the code, let's take a look at the various files, understand what they do, and make sure they work.

### The data-exploration.py file

This file downloads the CIFAR10 dataset and saves a sample of 25 images. The CIFAR10 dataset is commonly used to measure the performance of a CNN. As with any data science project, we start by doing some data exploration. In this case, we are just going to take a look at a few images.

To run the file,

1. Open the data-exploration.py file.
2. Press Ctrl+Shift+D to open the debug icon and then click Run and debug.
3. Alternatively, you can press Ctrl+F5 to directly run the file.

To open the sample images,

1. Take a look at line 44.
2. Open the file with the name specified in line 44.

### The build-cnn-model.py file

This file builds a CNN model and saves it to a file. This is an important file as all the design choices about the architecture of the model are coded here.

Let's run the file and see the model that it creates.

1. Run the file by pressing Ctrl+F5
2. Open the file with the name specified in line 70

#### Questions

1. Describe the architecture of the CNN. Provide details about the layers' size, type, and number. Also provide details about the activation functions used.

Answer: We can observe that the CNN is formed by 6 layers.

**Layer 1**: This is the input layer. It doesn't apply any processing but it tells our model the shape of the inputs. In this case, we have an input with shape None, 32, 32, 3. This means the images are 32 by 32 pixels and have three channels one for each primary color. None allows the model to take any number of images at a time. Recall that we can process multiple inputs simultaneously. We call multiple inputs a batch.

**Layer 2**: This is the convolutional layer. It takes an input of size None, 32, 32, 3, and outputs a tensor of size None, 30, 30, 32. The input is no longer an image that makes sense to the human eye.

**Layer 3**: This is the maxpooling layer. Recall that convolutional layers are usually paired with a pooling layer. In this case, we are using maxpooling. The layer reduces the size of the tensor to None, 15, 15, 32.

**Layer 4**: Next, we add a flatten layer. So far, we have been dealing with tensors (i.e., three-dimensional matrices). Since we are interested in classifying the images, we eventually want to have a vector of neurons of size 10, which is the number of classes. However, instead of jumping to a vector of size 10 we add an additional layer. This increases the model's capacity. The output of this layer is None, 7200. This layer doens't change the values, it simply "rolls out" the input. To see this we can multiply the dimensions of the input: 15x15x32=7200.

**Layer 5**: This is a dense layer. The dense layer is a traditional feedforward layer where all inputs are connected to all outputs. We set the size of the output to 64.

**Layer 6**: This layer does the final prediction. It outputs 10 probabilities. Each probability corresponds to one class. Since we are using the SparseCategoricalCrossEntropy, TensorFlow takes a softmax operation to find the largest probability, and the compares that to the label.

The convolution and dense layers are both using the rectilinear activation function.

Here's the architecture figure you should be seeing

1. Build a two-column table. The first row should have the lines of code that define the layers. The second column should have a screen shot of the corresponding layer in the architecture figure.

Answer: Layer 1 and Layer 2 are defined by model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3))) Layer 3 is defined by model.add(tf.keras.layers.MaxPooling2D((2, 2))) Layer 4 is defined by model.add(tf.keras.layers.Flatten()) Layer 5 is defined by model.add(tf.keras.layers.Dense(64, activation='relu')) Layer 6 is defined by model.add(tf.keras.layers.Dense(10))

* 1. Draw the input and output tensors and label its sizes for each layer. You should be able to this based on the answer to the first question.

### The training.py file

Now that we have defined the architecture of the model, we are ready to train it. The training algorithm and parameter choices are coded in this file. The file also runs trains the model and then saves it to a file.

Lets run the file to train the model.

1. Press Ctrl+F5

#### Questions

1. Which variables in the code are used as the training dataset by the function fit?

Answer: train\_images, train\_labels

1. Which variables in the code are used as the testing dataset by the function fit?

Answer: test\_images, test\_labels 3. Is accuracy a good performance measure? Why?

Answer: Accuracy is a good performance measure as long as there is a roughly equal number of \*training and testing sample for each category. The accuracy is calculates as correctly predicated sample divided by the total number of predictions. However, if one of the categories has many more samples, then even if we have a poor model that always predicts the dominant category, we would still have high accuracy.

1. Is categorical cross-entropy the appropriate loss function? Why? Yes. The reason is that we have multiple categories.

### The analysis.py file

Now that we have a trained model, we can observe the training error, testing error, and accuracy.

Run the file to generate the plots

1. Press Ctrl+F5
2. Open the accuracy plot (file name specified in line 22)
3. Open the loss plot (file name in line 35)

#### Questions

1. **What is difference between validation and testing in this code?**

**Answer. The difference between validation and testing is that the data that we are using for testing is not used for training, and vice versa.**

1. **What is the difference between training/testing loss and training/testing accuracy?**

**Answer. The training loss and accuracy are calcualted when the model predicts the label for samples in the training dataset. The testing accuracy and loss are calculated when the model predicts labels in the testing data set.**

1. **We are using the categorical cross-entropy (CCE) to measure the loss. Would a larger CCE or smaller CCE result in a lower testing error?**

**Answer. Our testing error is equal to the CCE itself. So, lower CCE means lower testing error.**

1. **What is the accuracy of the model?**

**Answer. After one epoch, the accuracy is about 0.55. You can see this value in the cnn-training-history.csv file.**

1. **Has the training/testing accuracy converged in the accuracy plot?**

**Answer. We cannot tell because we only had one epoch.**

1. **Has the training/testing loss converged in the loss plot?**

**Answer. We cannot tell because we only had one epoch.**

1. Did the difference between the training and testing loss/accuracy decreased or  
   increased across epochs? If it increased and decreased in the plot specify in which epochs it increased/decreased?

Answer. We cannot tell because we only had one epoch.

1. **Give the potential reasons that can explain the low accuracy of the model (anything below 90% is considered low)**

**Answer. Low accuracy can be due to low model capacity, over or under fitting, lack of enough data, too few epochs, incorrect learning rate, etc.**

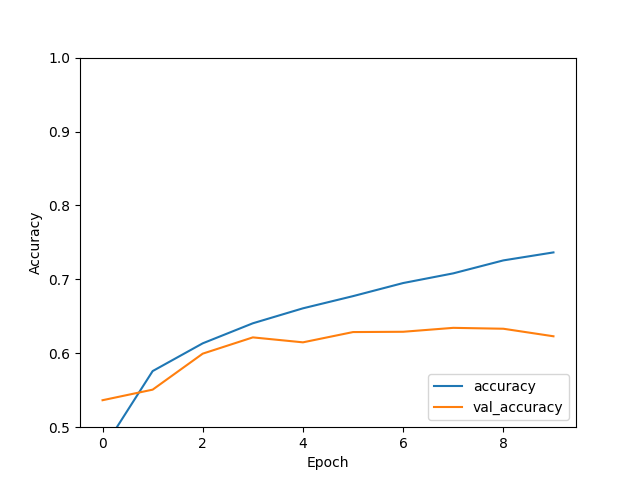
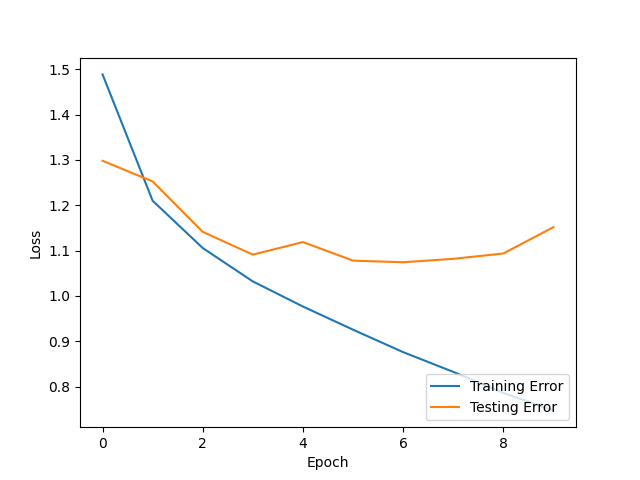
## Updating the code to improve performance.

Now that you have a trained model, lets see if we can improve it. We can change the training parameters and the model itself.

### Updating the Training Parameters

1. Based on your answers to Q.5 in the analysis.py file [section](https://github.com/deep-learning-prof/cnn-lab/#the-analysispy-file), make changes to the training parameters to improve the testing accuracy and loss.
2. Describe your changes.

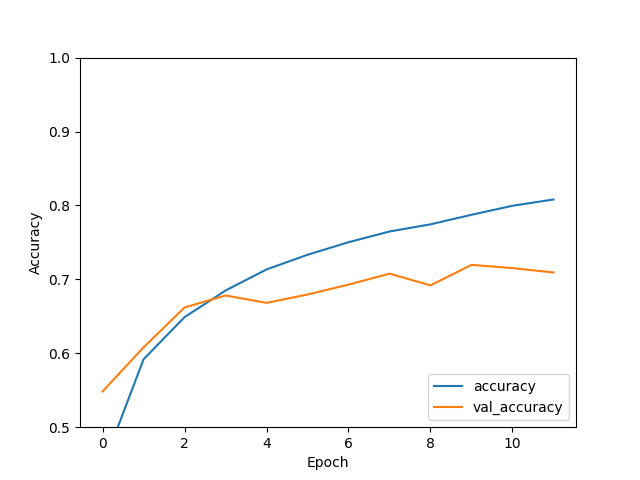
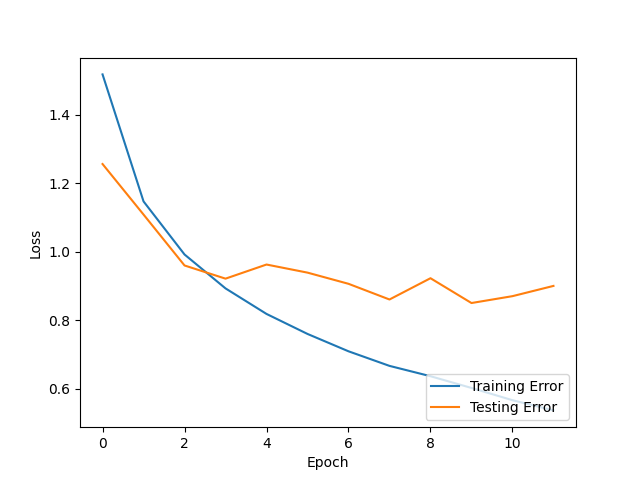
**Answer. There are multiple possible changes to improve the accuracy. The first change that I made was to increase the epohcs parameter in the model.compile function from 1 to 10. After, changing the epochs to a larger value, we can observe the accuracy and loss plots and give a better answer to the questions above. These are the plots I obtained after running 10 epochs.**

[](https://github.com/deep-learning-prof/cnn-lab/blob/main/accuracy-10.png) [](https://github.com/deep-learning-prof/cnn-lab/blob/main/loss-10.png)

**From these plots, we can observe that the testing error starts to increase around 8 epochs. If we choose, to stay with this model, we would stop the training at 8 epochs. We also see that the accuracy maxes out at around 60%.**

Lets see if we can do better. Next, I added a convolutional layer to my network by adding the following lines of code after the original convolutional layer. model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu')) model.add(tf.keras.layers.MaxPooling2D((2, 2)))

I also increased the epochs from 10 to 12 in case the larger model needs more training.

These are the plots I obtained with the new model. [](https://github.com/deep-learning-prof/cnn-lab/blob/main/accuracy-12.png) [](https://github.com/deep-learning-prof/cnn-lab/blob/main/loss-12.png)

**We see that the we can obtained an accuracy of about 70%, which is a 10% increase from the previous model.**

**We can continue extending the network with more layers to improve the testing accuracy.**

1. Run the training.py and analysis.py files with your new parameters.
2. Did the testing performance improve? Explain why or why not.

Answer **The model performance improved with the model that had an extra convolutional layer. The reason is that the capacity of the model, i.e., the total number of functions as well as their complexity from which it can choose to model the training data, is larger. This allows the model to find a more appropriate function.**

### CNN Model updates

1. Based on your answers to Q.5 in the analysis.py file [section](https://github.com/deep-learning-prof/cnn-lab/#the-analysispy-file), make changes to the CNN architecture.
2. Describe your changes.
3. Run the training.py and analysis.py files with your new parameters.
4. Did the testing performance improve? Explain why or why not.

# Recurrent Neural Networks Labs

The objective of this lab is to give you hands-on experience designing a recurrent neural network (RNN). These instructions will provide you with a basic RNN architecture for text classification, and provide you with default training settings. This sample model has a mediocre performance. Your job is to use your knowledge about model capacity, under/over fitting, and deep network architectures to improve the performance, i.e., reduce the testing error.

## Code test drive

Before we start tinkering with the code, let's take a look at the various files, understand what they do, and make sure they work.

### The encoder-embedding-exploration.py file

Unlike image classification problems where we can directly use our data to train our models, **text classification problems require us to first pre-process the data**. The main reason is that deep learning models are unable to take the characters in the text as input. To solve this issue, we first look at all the words that appear in our training data set and form a list of unique words. We call this list the vocabulary. We then assign a unique number to each word in the vocabulary. We call this number the tokens. **The idea of the encoder is to substitute the words in the sample with their respective tokens within our RNN.**

You may be tempted to stop at the encoder. However, our vocabulary has many words that have identical meaning (i.e., synonyms) or similar meanings (i.e., near synonyms.). Since we are doing text classification, **we want our RNN to be able to learn the meaning of the words rather than the words themselves. To this end, we use a technique called embedding. Embedding replaces the tokens with vectors that represent the meaning of the words. Words with similar meaning will have similar vectors.**

To run the file,

1. Open the encoder-embedding-exploration.py file.
2. Read the comments and code
3. Press Ctrl+Shift+D to open the debug icon and then click Run and debug.
4. Alternatively, you can press Ctrl+F5 to directly run the file.

You should see a few words from the vocabulary as well as an example of how a movie review was tokenized by the encoder.

### The build-rnn.py file

This file builds an RNN model and saves it to a file. This is an important file as all the design choices about the architecture of the model and its training are coded here.

Let's run the file and see the model that it creates.

1. Run the file by pressing Ctrl+F5
2. Open the image file with the name specified in line 69

#### Questions

1. **Describe the architecture of the RNN. Provide details about the layers' size, type, and number. Also provide details about the activation functions used.**

**Answer. The model is formed by 6 layers,**

**Layer 1. This is the input layer. The size of the input is (None,). None indicates that the number of samples to be processed in batch is variable (i.e, the batch size). The empty field next to none indicates that the size of the inputs (i.e., the reviews) are also variable.**

**Layer 2. The TextVectoriztion layer comes next. The input and ouputs of this layer are also variable since it only replaces words with tokens.**

**Layer 3. The Embedding layer also has a variable input. The output yields a variable number of vectors of size 64.**

**Layer 4. The bidrectional LSTM (long short-term memory) layer outputs a vector of size 128.**

**Layer 5. This dense layer outputs a vector of size 64.**

**Layer 6. This dense layer finally condenses the model into a scalar output.**

1. Build a two-column table. The first row should have the lines of code that define the layers. The second column should have a screen shot of the corresponding layer in the architecture figure.

Layer 1 this layer is atuanatically created by TensorFlow. Layer 2 this layer is given by encoder = tf.keras.layers.TextVectorization(max\_tokens=VOCAB\_SIZE) Layer 3 this layer is defined by model.add(tf.keras.layers.Embedding(input\_dim=len(encoder.get\_vocabulary()), output\_dim=64, mask\_zero=True)) Layer 4 is defined by model.add(tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64))) Layer 5 is defined by model.add(tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64))) Layer 6 is defibed by model.add(tf.keras.layers.Dense(1))

1. Draw the input and output tensors and label its sizes for each layer.

**Answer** You should be able to draw them based on the previous answers.

### The training-rnn.py file

Now that we have defined the architecture of the model, we are ready to train it. The training algorithm and parameter choices are coded in this file. The file also runs trains the model and then saves it to a file.

Lets run the file to train the model.

1. Press Ctrl+F5

#### Questions

1. Which variables in the code are used as the training dataset by the function fit?

**Answer**. train\_dataset

1. **Which variables in the code are used as the testing dataset by the function fit? Answer. test\_dataset**
2. **Is accuracy a good performance measure? Why? Answer Since we are dealing with a binary classification problem, i.e., the review is \*either positive or negative, and we have a balanced dataset, accuracy is a good measure.**
3. Is binary cross-entropy the appropriate loss function? Why not use categorical cross-entropy? We use binary cross-entropy because our labels are either 1 or 0.

### The analysis.py file

Now that we have a trained model, we can observe the training error, testing error, and accuracy.

Run the file to generate the plots

1. Press Ctrl+F5
2. Open the accuracy plot (file name specified in line 22)
3. Open the loss plot (file name in line 35)

#### Questions

1. **We are using the binary cross-entropy (CCE) to measure the loss. Would a larger CCE or smaller CCE result in a lower testing error?**

**Answer. A larger CCE results in a larger testing error.**

1. **What is the accuracy of the model?**

**Answer The accuracy indicates the percentage of correctly classified texts.**

1. **Has the training/testing accuracy converged in the accuracy plot?**

**Answer We cannot tell since we only trained for one epoch.**

1. **Has the training/testing loss converged in the loss plot?**

**Answer We cannot tell since we only trained for one epoch.**

1. **Did the difference between the training and testing loss/accuracy decreased or  
   increased across epochs? If it increased and decreased in the plot specify in which epochs it increased/decreased?**

**Answer We cannot tell since we only trained for one epoch.**

1. Give the potential reasons that can explain the low accuracy of the model (anything below 90% is considered low)

**Answer** We have 0.52 accuracy. This is low. Probably because we only trained for one epoch.

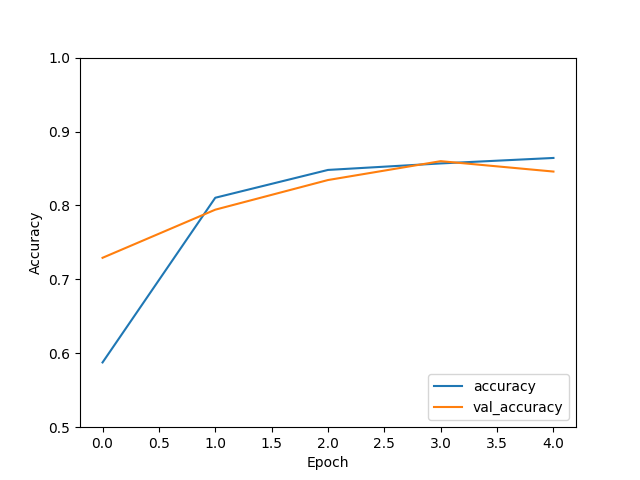
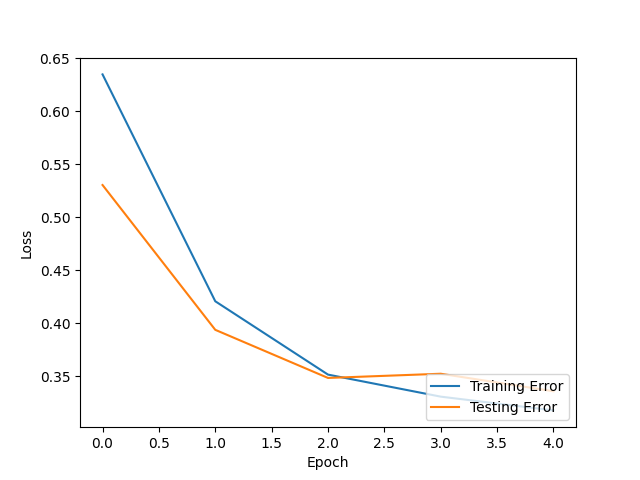
## Updating the code to improve performance.

Now that you have a trained model, lets see if we can improve it. We can change the training parameters and the model itself.

### Updating the Training Parameters

1. Based on your answers to Q.5 in the analysis.py file [section](https://github.com/deep-learning-prof/rnn-lab#the-analysispy-file), make changes to the training parameters to improve the testing accuracy and loss.
2. Describe your changes.

**Answer** Lets start with training the model with more epochs. One epoch is not enough to evaluate if the model is under or overfitting. I changed the number of epochs from 1 to 5 and obtained the following plots.

[](https://github.com/deep-learning-prof/rnn-lab/blob/main/accuracy-5.png) [](https://github.com/deep-learning-prof/rnn-lab/blob/main/loss-5.png)

We see that the testing accuracy starts to decrease at the last epoch. This suggests that the model is starting to overfit. However, we would need to run additional epochs to ensure this is not just a temporary decrease and future epochs can improve the accuracy. Recall that the loss function is non-linear and perhaps we landed on a local minimum. Similarly, the loss function starts to increase during the last two epochs.

Besides increasing the number of epochs, we can use additional convolutional layers after the bidirectional layer. We can also add an additional bidirectional or regular LSTM layer. The first dense layer can be replaced by a convolutional layer.

1. Run the training.py and analysis.py files with your new parameters.
2. Did the testing performance improve? Explain why or why not.

**Answer** The testing performance increased after increasing the number of epochs. We can achieve an accuracy lose to 90 percent, which is good.

### RNN Model updates

1. Based on your answers to Q.5 in the analysis.py file [section](https://github.com/deep-learning-prof/rnn-lab#the-analysispy-file), make changes to the RNN architecture. E.g., add or remove layers, change the activation function, etc.
2. Describe your changes.
3. Run the training.py and analysis.py files with your new parameters.
4. Did the testing performance improve? Explain why or why not. **See answers above**