np.dot() and multiplication

- np.dot() performs a matrix-matrix or matrix-vector multiplication. This is different from np.multiply() and the * operator (which is equivalent to .* in Matlab/Octave), which performs an element-wise multiplication.

Regularization

- regularization term R to the optimization objective, whose job is to control the complexity of the parameter value, and avoid cases of overfitting:
    a.

$$\hat{\Theta} = \underset{\Theta}{\operatorname{argmin}}\, \mathcal{L}(\Theta) + \lambda R(\Theta)$$

$$= \underset{\Theta}{\operatorname{argmin}}\, \frac{1}{n} \sum_{i=1}^{n} L(f(\boldsymbol{x}_i; \Theta), \boldsymbol{y}_i) + \lambda R(\Theta).$$

- regularization term considers the parameter values, and scores their complexity.

LIMITATIONS OF LINEAR MODELS: THE XOR PROBLEM

- If we transform the points by feeding each of them through the nonlinear function the XOR problem becomes linearly separable.
- In the XOR example the transformed data has the same dimensions as the original one, but often in order to make the data linearly separable one needs to map it to a space with a much higher dimension.

Performance measure

- In regression analysis, you typically use the coefficient of determination, root-mean-square error, mean absolute error, or similar quantities.
- For classification problems, you often apply accuracy, precision, recall, F1 score, and related indicators.
- The cost function used in Logistic Regression is Log Loss.
- The validation set is used for unbiased model evaluation during hyperparameter tuning.
- Given two sequences, like x(input) and y(output)(target variables) here, train_test_split() performs the split and returns four sequences (in this case NumPy arrays) in this order:
    a. x_train: The training part of the first sequence (x)
    b. x_test: The test part of the first sequence (x)
    c. y_train: The training part of the second sequence (y)
    d. y_test: The test part of the second sequence (y)
- We can implement cross-validation with KFold, StratifiedKFold, LeaveOneOut, etc

K-fold validation(Test -Train Split)

- If you have insufficient data, then a suitable alternate model evaluation procedure would be the k-fold cross-validation procedure.
- There is no optimal split percentage.
- The samples of the dataset are shuffled randomly and then split into the training and test sets according to the size you defined
- You must choose a split percentage that meets your project's objectives with considerations that include:
    a. Computational cost in training the model.
    b. Computational cost in evaluating the model.
    c. Training sets representativeness.
    d. Test set representativeness.

Randomness

- (random_state = 101)Randomness is a feature, which allows an algorithm to attempt to avoid overfitting the small training set and generalize to the broader problem.
- The value of random_state isn't important—it can be any non-negative integer.
  Link: https://realpython.com/train-test-split-python-data

Traditional Software vs. Machine Learning

Traditional Software

- Goal: Meet a functional specification
- Quality depends only on code
- Typically pick one software stack w/ fewer libraries and tools
- Limited deployment environments

Machine Learning

- Goal: Optimize metric(e.g., accuracy. Constantly experiment to improve it
- Quality depends on input data and tuning parameters
- Compare + combine many libraries, model
- Diverse deployment environments

Imbalanced dataset

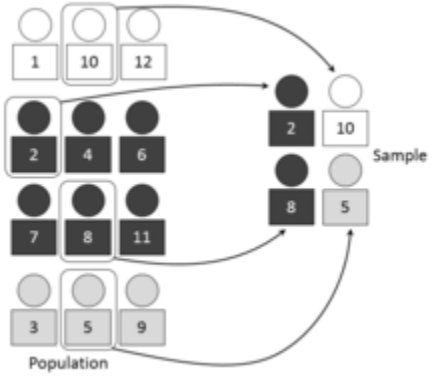
- The two main approaches to randomly resampling an imbalanced dataset are to delete examples from the majority class, called undersampling, and to duplicate examples from the minority class, called oversampling.
- the imbalance in this dataset does provide a good platform on which to demonstrate the merits of anomaly detection for classification problems

Mutual information

Scikit-learn has two mutual information metrics in its feature_selection module: one for real-valued targets (mutual_info_regression) and one for categorical targets (mutual_info_classif)

Stratify

- “stratify” argument to the y component of the original dataset.



- Stratify useful for : Some classification problems do not have a balanced number of examples for each class label. As such, it is desirable to split the dataset into train and test sets in a way that preserves the same proportions of examples in each class as observed in the original dataset.

Underfitting and overfitting

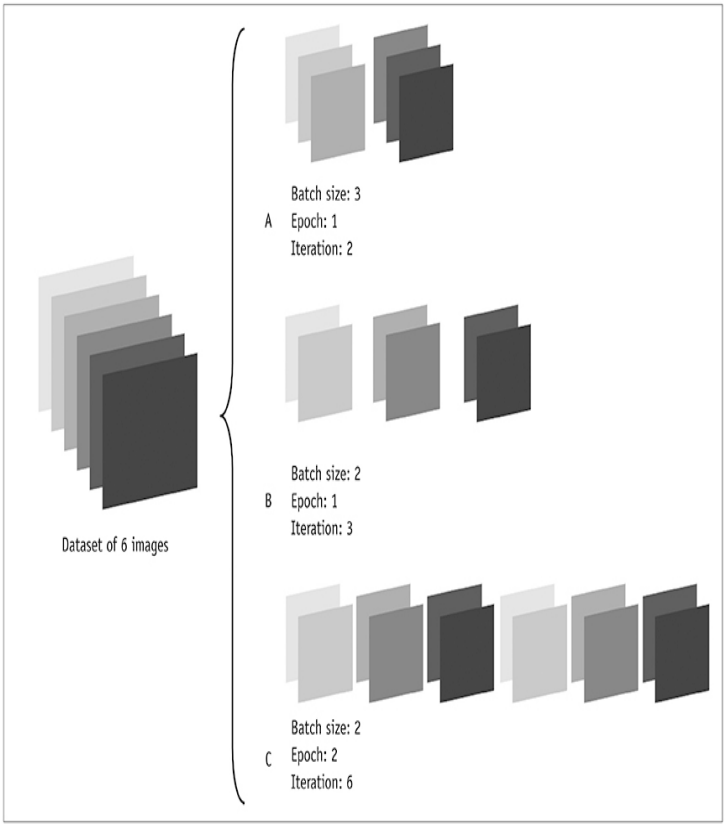
- underfitting occurs because the model is not suitable for the problem you are trying to solve. Usually, this means that the model is less complex than required in order to learn those parameters that can be proven to be predictive.
- Overfitting. This phenomenon occurs when a model performs really well on the data that we used to train it but it fails to generalize well to new, unseen data points.

GloVe (Global Vector)

- GloVe stands for global vectors for word representation.
- It is an unsupervised learning algorithm developed by Stanford for generating word embeddings by aggregating global word-word co-occurrence matrices from a corpus.

EPOCH vs Batch Size vs Iteration

- A sample is a single row of data.
- The batch size is a parameter that defines the number of samples to work.
    a. Batch Gradient Descent. Batch Size = Size of Training Set
    b. Stochastic Gradient Descent. Batch Size = 1
    c. Mini-Batch Gradient Descent. 1 < Batch Size < Size of Training Set
- The number of epochs is a parameter that defines the number times that the learning algorithm will work through the entire training dataset.



https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

Loss In Neural Network

- we get the predicted value (yhat) through the output layer. But prediction is always closer to the actual (y), which we term as errors. So, we define the loss/cost functions to capture the errors and try to optimize it though backpropagation.
- Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1.

1	Loss functions in Regression based problem	a. Mean Square Error Loss
		b. Mean Absolute Error Loss
		c. Huber Loss
2	Loss functions in Binary classification-based problem	a. Binary Cross Entropy Loss
		b. Hinge Loss
3	Loss functions in Multiclass classification-based problem	a. Multiclass Cross Entropy Loss
		b. Spare Multiclass Cross Entropy Loss
		c. Kullback Leibler Divergence Loss

Learning curve

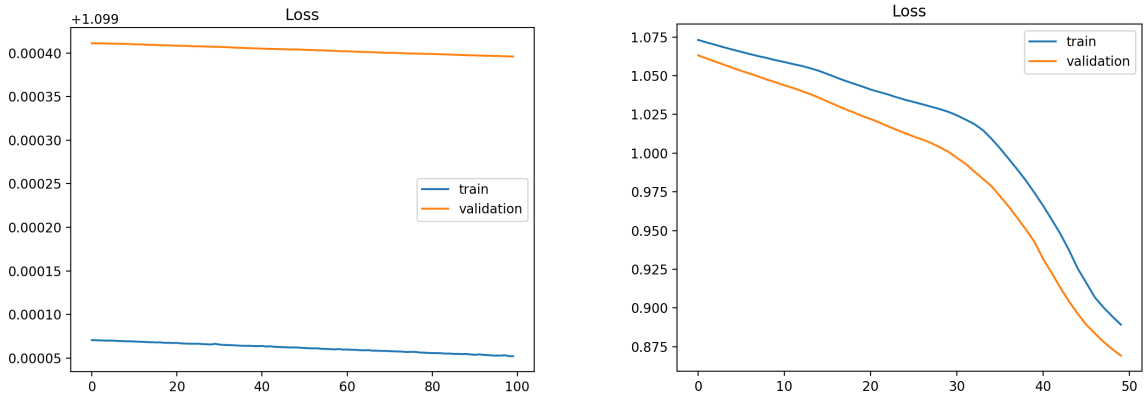
- The machine learning curve(learning curve) is useful for many purposes including comparing different algorithms,[2] choosing model parameters during design,[3] adjusting optimization to improve convergence, and determining the amount of data used for training.

- It is common to create line plots that show epochs along the x-axis as time and the error or skill of the model on the y-axis. These plots are sometimes called learning curves. These plots can help to diagnose whether the model has over learned, under learned, or is suitably fit to the training dataset.

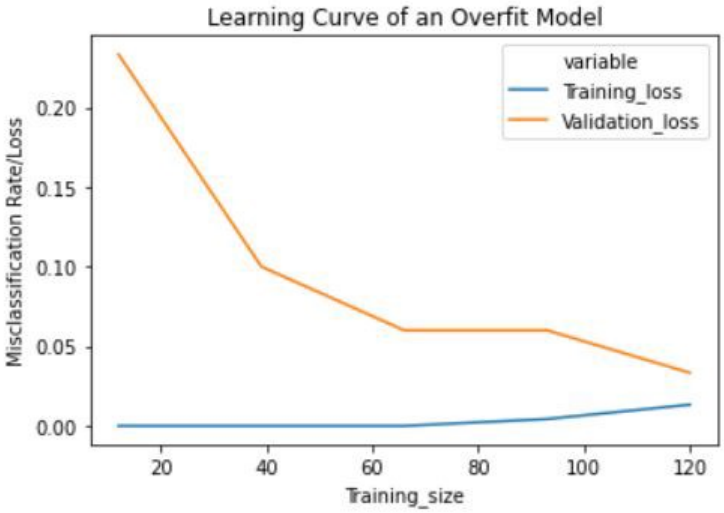
- An underfit model may also be identified by a training loss that is decreasing and continues to decrease at the end of the plot.
- This indicates that the model is capable of further learning and possible further improvements and that the training process was halted prematurely.

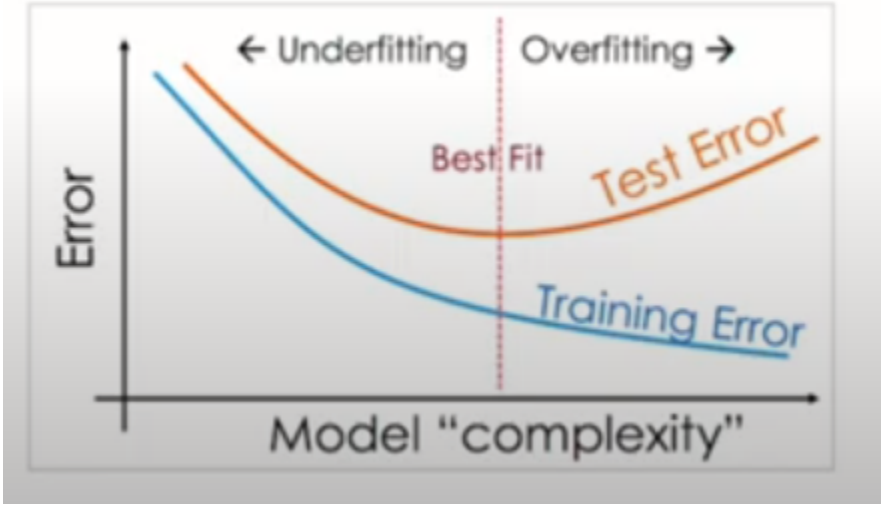
A plot of learning curves shows underfitting if:

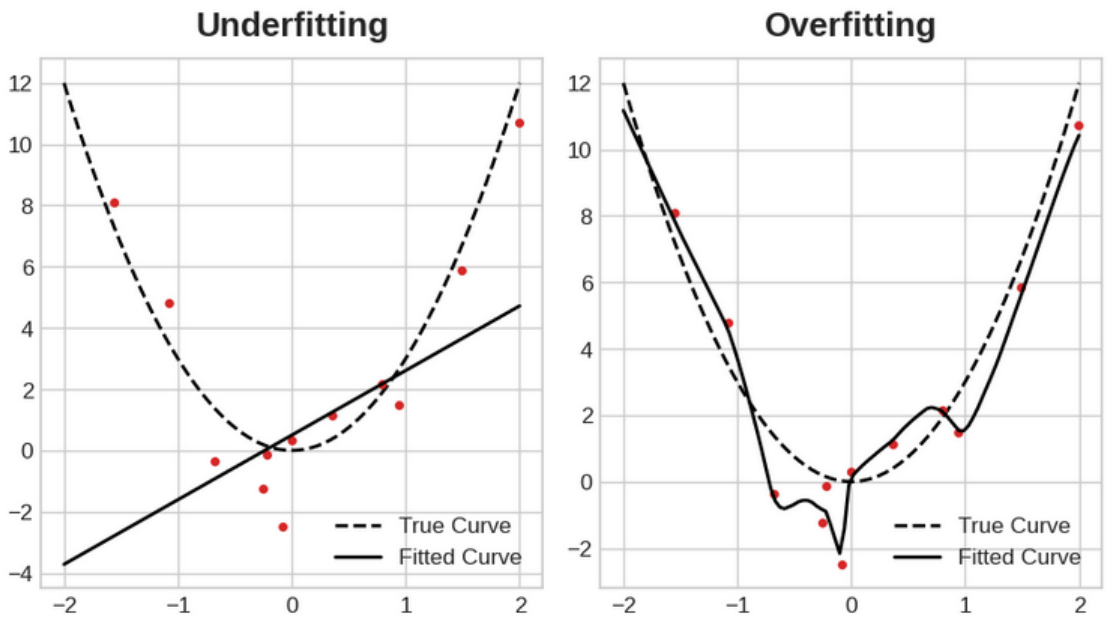
- The training loss remains flat regardless of training.
- The training loss continues to decrease until the end of training.



Underfitting curve above


Learning Curve of an Overfit Model
variable
Training_loss
Validation_loss
Misclassification Rate/Loss
0.20
0.15
0.10
0.05
0.00
20
40
60
80
100
120
Training_size


← Underfitting
Overfitting →
Best Fit
Test Error
Training Error
Error
Model "complexity"



Underfitting and overfitting.

This trade-off indicates that there can be two problems that occur when training a model: not enough signal or too much noise. **Underfitting** the training set is when the loss is not as low as it could be because the model hasn't learned enough *signal*. **Overfitting** the training set is when the loss is not as low as it could be because the model learned too much *noise*.

Capacity of neural network

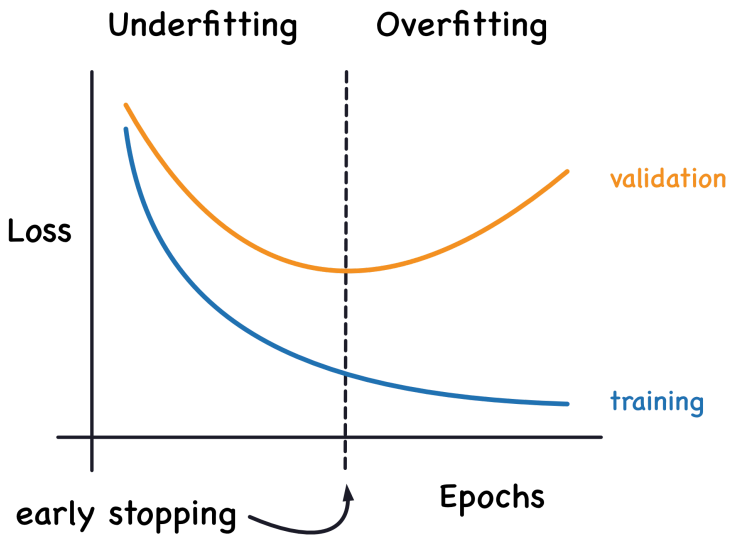
A model's **capacity** refers to the size and complexity of the patterns it is able to learn. For neural networks, this will largely be determined by how many neurons it has and how they are connected together.

If it appears that your network is underfitting the data, you should try increasing its capacity.

You can increase the capacity of a network either by making it wider (more units to existing layers) or by making it deeper (adding more layers)

Early Stopping

We mentioned that when a model is too eagerly learning noise, the validation loss may start to increase during training. To prevent this, we can simply stop the training whenever it seems the validation loss isn't decreasing anymore. Interrupting the training this way is called **early stopping**.



Once we detect that the validation loss is starting to rise again, we can reset the weights back to where the minimum occured. This ensures that the model won't continue to learn noise and overfit the data.

A **callback** is just a function you want run every so often while the network trains. The early Keras has a variety of useful callbacks pre-defined,stopping callback will run after every epoch

Classes

`class BackupAndRestore`: Callback to back up and restore the training state.

`class BaseLogger`: Callback that accumulates epoch averages of metrics.

`class CSVLogger`: Callback that streams epoch results to a CSV file.

`class Callback`: Abstract base class used to build new callbacks.

`class CallbackList`: Container abstracting a list of callbacks.

`class EarlyStopping`: Stop training when a monitored metric has stopped improving.

`class History`: Callback that records events into a `History` object.

`class LambdaCallback`: Callback for creating simple, custom callbacks on-the-fly.

`class LearningRateScheduler`: Learning rate scheduler.

`class ModelCheckpoint`: Callback to save the Keras model or model weights at some frequency.

`class ProgbarLogger`: Callback that prints metrics to stdout.

`class ReduceLROnPlateau`: Reduce learning rate when a metric has stopped improving.

`class RemoteMonitor`: Callback used to stream events to a server.

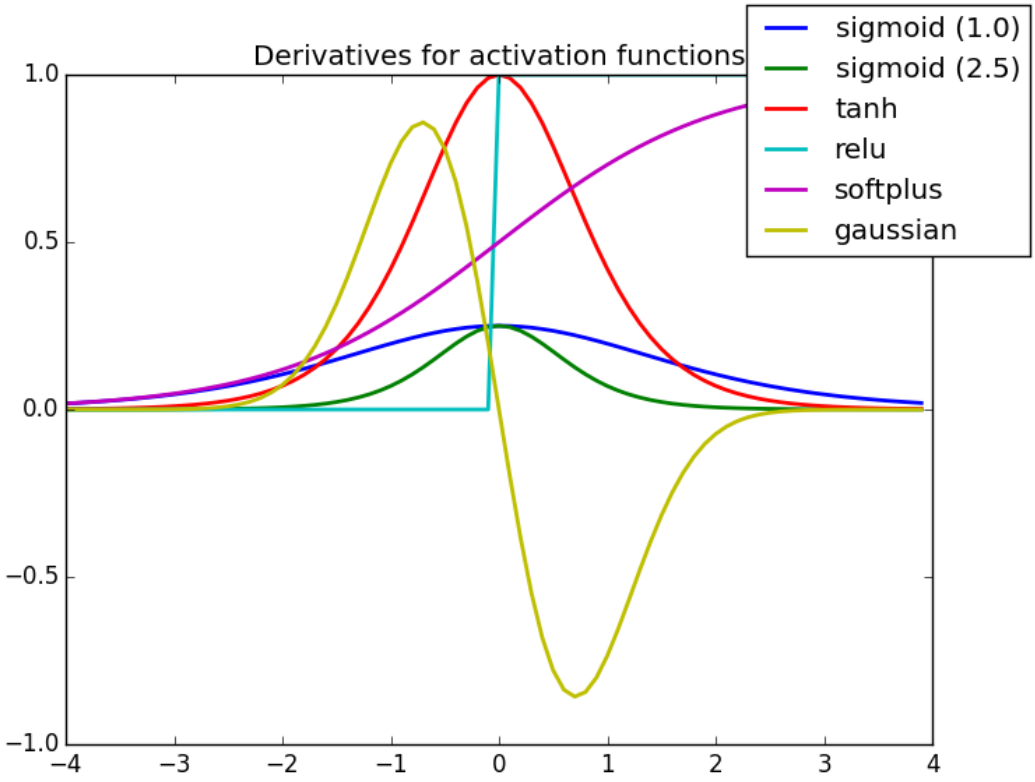
`class TensorBoard`: Enable visualizations for TensorBoard.

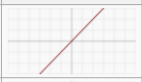
`class TerminateOnNaN`: Callback that terminates training when a NaN loss is encountered.

Why is the activation function required?

- An Activation Function decides whether a neuron should be activated or not. This means that it will decide whether the neuron's input to the network is important or not in the process of prediction using simpler mathematical operations.
- The purpose of the activation function is to introduce non-linearity into the output of a neuron.
- The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.
- The main terminologies needed to understand for nonlinear functions are:

- Derivative or Differential: Change in y-axis w.r.t. change in x-axis.It is also known as slope.
- Monotonic function: A function which is either entirely non-increasing or non-decreasing.
- The Nonlinear Activation Functions are mainly divided on the basis of their range or curves.
- Both tanh and logistic sigmoid activation functions are used in feed-forward nets.



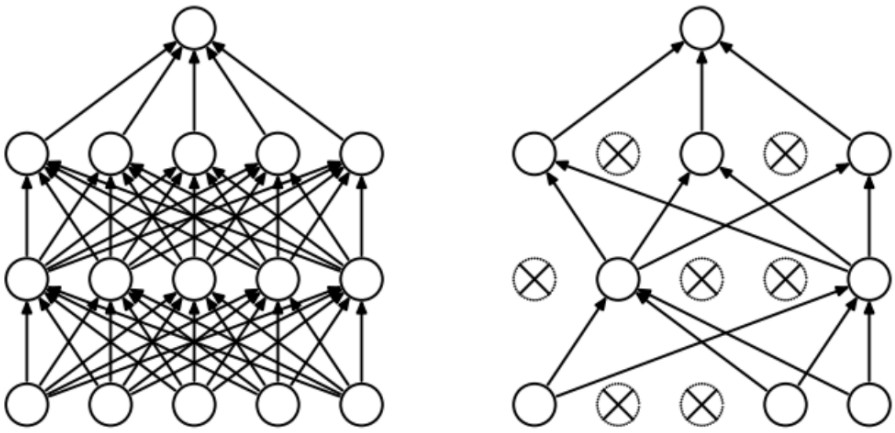
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Softmax

- Softmax function is used in various multiclass classification machine learning algorithms such as multinomial logistic regression.
- You can think of softmax as a normalizing function used when your algorithm needs to classify two or more classes.
- Softmax function is considered as an activation function in neural networks and algorithms such as multinomial logistic regression.
- Some of these algorithms are following:
    a. Neural networks
    b. Multinomial logistic regression (Softmax regression)
    c. Bayes naive classifier
    d. Multi-class linear discriminant analysis
    e. In artificial neural networks, the softmax function is used in the final / last layer.

Dropout(**Grid Search** hyperparameter)

- By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections
- During training, some number of layer outputs are randomly ignored or “dropped out.”
- It can be used with most types of layers, such as dense fully connected layers, convolutional layers, and recurrent layers such as the long short-term memory network layer.
- A good value for dropout in a hidden layer is between 0.5 and 0.8. Input layers use a larger dropout rate, such as 0.8.
- dropout is more effective on those problems where there is a limited amount of training data and the model is likely to overfit the training data.
- Another effective technique for preventing neural networks from overfitting the training data is dropout training.
- The dropout method is designed to prevent the network from learning to rely on specific weights.
-



(a) Standard Neural Net (b) After applying dropout.

dead neuron

- The drawback of ReLU is that they cannot learn on examples for which their activation is zero. It usually happens if you initialize the entire neural network with zero and place ReLU on the hidden layers.

Learning Rate

Too large learning rates will prevent the network from converging on an effective solution.
Too small learning rates will take a very long time to converge.
As a rule of thumb, one should experiment with a range of initial learning rates in range Œ0; 1

e.g., 0:001 , 0:01 , 0:1 , 1 . Monitor the network's loss over time, and decrease the learning rate once the loss stops improving on a held-out development set. Learning rate scheduling decreases the rate as a function of the number of observed minibatches.

Why is Relu so popular?

- ReLU is considered as one of the biggest breakthroughs in deep learning because ReLU makes it possible to train a very deep neural network.
- ReLU prevents the **vanishing gradient descent problem.**
- ReLU is easy to optimize because it is so simple, computationally cheap, and similar to the linear activation function, but in fact, ReLU is a nonlinear activation function that allows complex patterns in the data to be learned.

Dying RELu

- The "Dying ReLU" refers to a neuron which outputs 0 for your data in the training set.
- This happens because sum of weight * inputs in a neuron (also called activation) becomes <= for all input patterns. This causes ReLU to output 0.

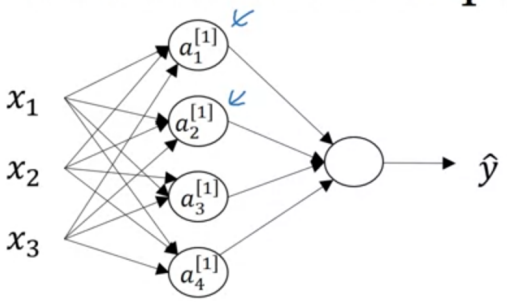
leaky RELU

- Instead of pushing the negative value to 0, Leaky ReLU allows some lake at the negative region by multiplying x with a constant of 0.01
- By doing this, even if the neuron has a big negative weight and bias, it's still possible to backpropagate the gradient through the layer.
- By doing this, even if the neuron has a big negative weight and bias, it's still possible to backpropagate the gradient through the layer

shape of weights in each matrix

- As a thumb rule, weight matrix has following dimensions :
- The number of rows must equal the number of neurons in the previous layer. (in this case the previous layer is the input layer). So 3
- The number of columns must match the number of neurons in the next layer.

Neural Network Representation



$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]},\ a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]},\ a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]},\ a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]},\ a_4^{[1]} = \sigma(z_4^{[1]})$$

- Dense : most of the data-points are non zero
- Sparse : most of data-points are zero

Scalar Vector and Tensor

Scalar is basically any number, like 1, 5, 23, or 42. But we are not limited to integers.Also, real numbers are scalars, like 23.5,
Scalar => 1 or 2.3 or 0.001 or 0 or 12345

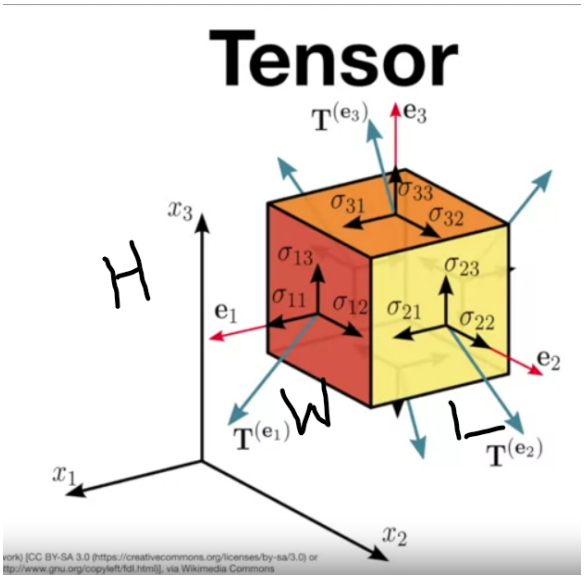
Vector can easily be confused with tuple. So this guy here can be either a vector or a tuple.But this is a tuple. Can you see the difference? So in a vector, each element has to have the same type, whereas in the tuple, types can be mixed.
Vector = > (1,2,34, 5 ,50)

Tuple => (1.1, 23, 0.001, 1, a)

A Matrix is the big brother of a vector. It's basically a list of equal-sized vectors. Notice that the number of rows and columns can be different, but each element has to have the same type. A Matrix has n columns and m rows, therefore we call these matrices m- by- n matrices.

Matrix =[[1,2,3], [4,5,6], [7,8,9]]

The coolest guy is called the Tensor. Here, you can see a 3D tensor which is basically nothing else like a matrix, but in three dimensions. Tensors can be quite handy in image processing, for example. So you can have one dimension for the height, one for the width, and one for colors, alpha channel, and focus information, for example. So a tensor is more general term for special cases. So, for example, a zero-dimensional tensor is a scalar, a one-dimensional tensor is a vector, a two-dimensional tensor is a matrix.



CNN

Problem with Feedforward Neural Network

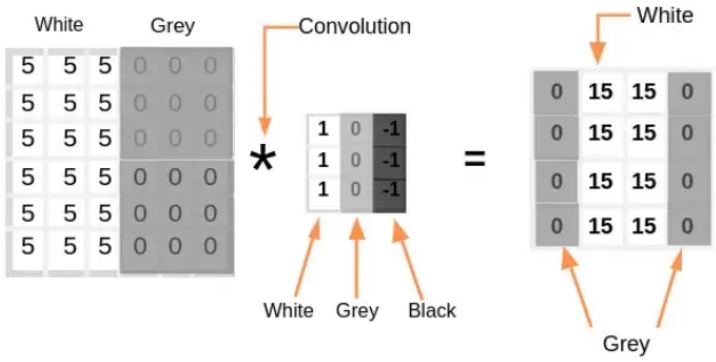
Suppose you are working with MNIST dataset, you know each image in MNIST is 28 x 28 x 1(black & white image contains only 1 channel). Total number of neurons in input layer will 28 x 28 = 784, this can be manageable. What if the size of image is 1000 x 1000 which means you need 10^6 neurons in input layer. Oh! This seems a huge number of neurons are required for operation. It is computationally ineffective right

Convolutional Neural Network or CNN. In simple word what CNN does is, it extract the feature of image and convert it into lower dimension without loosing its characteristic.

RGB CHANNEL IN IMAGE

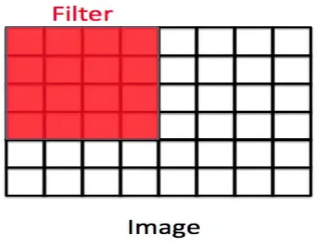
So the intensity of the red channel at each point with width and height can be represented into a matrix, the same goes for the blue and green channels, so we end up having three matrices, and when these are combined they form a tensor.
You need to reshape it into a single column. Suppose you have image of dimension 28 x 28 =784, you need to convert it into 784 x 1 before feeding into input.

Convolution Operation

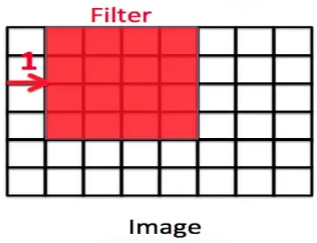


Convolution operation

Stride in CNN

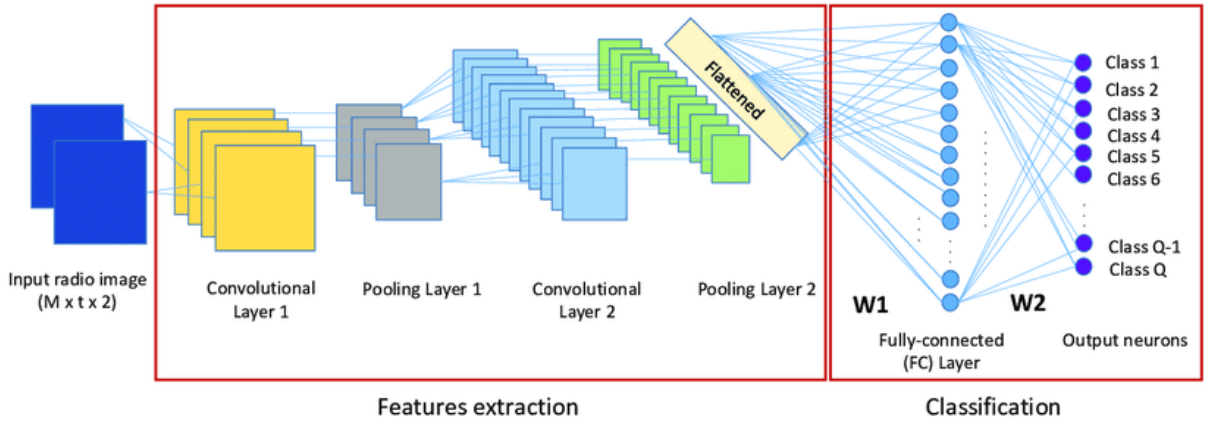


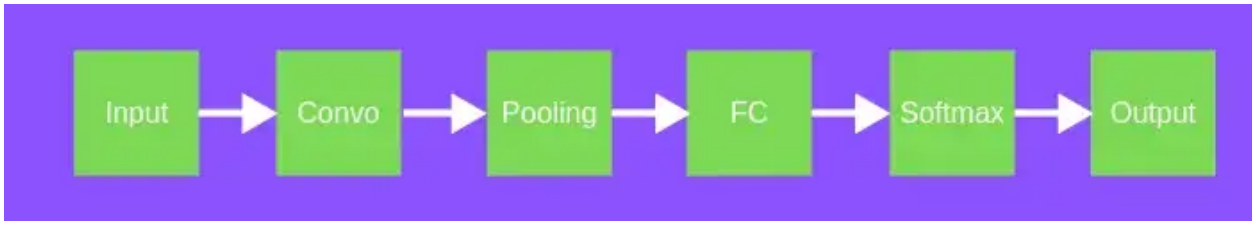
If stride = 1, the filter will move one pixel.



Stride is a component of convolutional neural networks, or neural networks tuned for the compression of images and video data. Stride is a parameter of the neural network's filter that modifies the amount of movement over the image

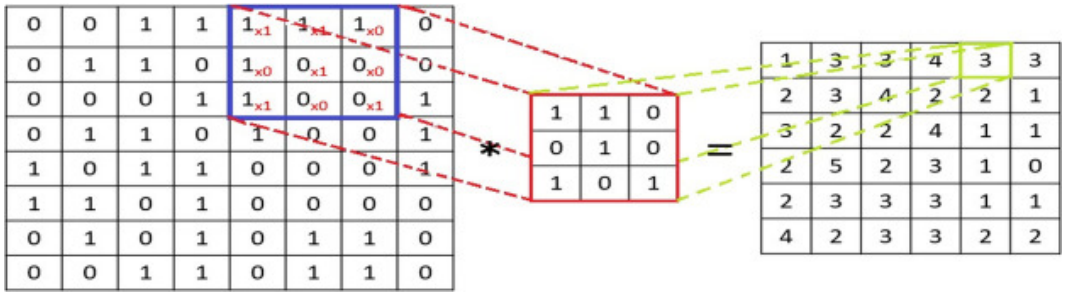
Layers In CNN





Convolution Layer

- Convo layer is sometimes called feature extractor layer because features of the image are get extracted within this layer
- In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size MxM. Result of the operation is single integer of the output volume.
- The result is a feature map, which is a filtered version of the input image that highlights the features detected by the filters.



- Convo layer also contains ReLU activation to make all negative value to zero.
- ReLU, or rectified linear unit, is a non-linear activation function that is commonly used in convolutional layers of CNNs because it helps to introduce non-linearity into the model.
- The function maps any input value less than zero to zero and any input value greater than or equal to zero to the same value.

- This non-linearity helps the model to learn more complex and abstract representations of the input data, which in turn improves its ability to classify or classify and locate objects in images. Additionally, ReLU also helps to alleviate the problem of vanishing gradients, which can occur when training deep neural networks with other activation functions such as sigmoid or tanh.

Pooling Layer

Use of pooling layer:
If we apply FC(Fully Connected layer) after Convo layer without applying pooling or max pooling, then it will be computationally expensive and we don't want it

Pooling layer is used to reduce the spatial volume of input image after convolution. It is used between two convolution layer.

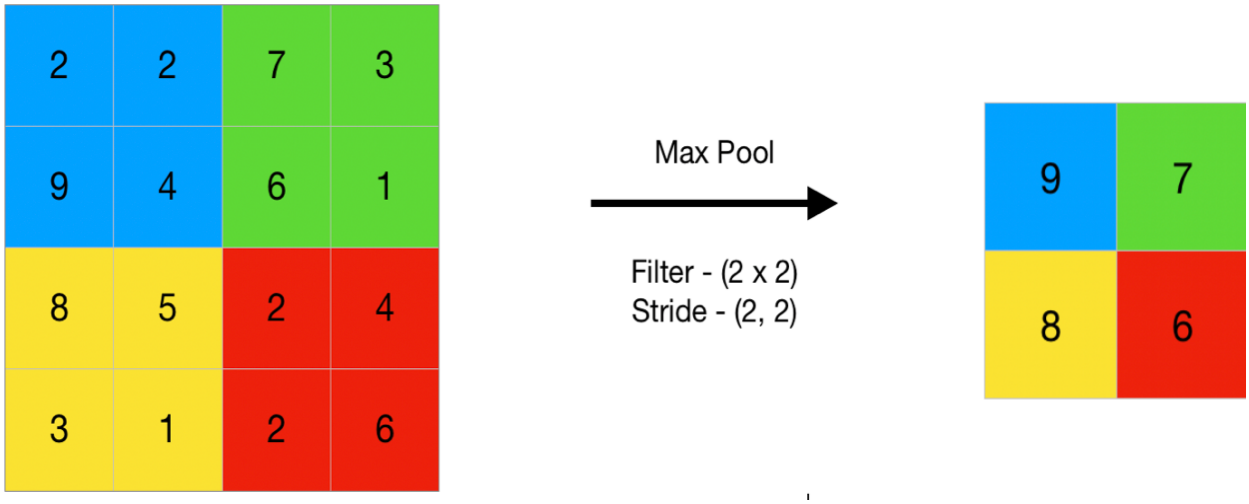
There are two main types of pooling layers commonly used in CNNs:

1. max pooling
2. average pooling.

There are also some variations of pooling layer such as L2-norm pooling, which is a type of pooling that computes the L2-norm within a rectangular window in the feature map and outputs it as the new value for the corresponding position in the pooled feature map.

Max Pooling Layer

- A MaxPool2D layer is much like a Conv2D layer, except that it uses a simple maximum function instead of a kernel, with the pool_size parameter analogous to kernel_size.
- A MaxPool2D layer doesn't have any trainable weights like a convolutional layer does in its kernel.
- Max pooling takes a patch of activations in the original feature map and replaces them with the maximum activation in that patch.



Advantages of Max-Pooling:

- One of the main use cases of max-pooling is to increase the invariance of the model to small translations or deformations of the input image.
- This helps the model to be more robust to small changes in the position or shape of an object in the image.
- Max-pooling also helps to reduce the computational cost of the model by reducing the number of parameters and computations required in the next layers.
- It also helps to control overfitting by reducing the number of parameters that need to be learned.
- Another use case is for object detection, max-pooling is used to reduce the spatial resolution of the feature map and increase the field of view of the convolutional filters

Fully Connected Layer(FC) Layer

- Fully connected layer involves weights, biases, and neurons. It connects neurons in one layer to neurons in another layer. It is used to classify images between different category by training.
- In fully connected layers, the neuron applies a linear transformation to the input vector through a weights matrix. A non-linear transformation is then applied to the product through a non-linear activation function *f*.

GAN

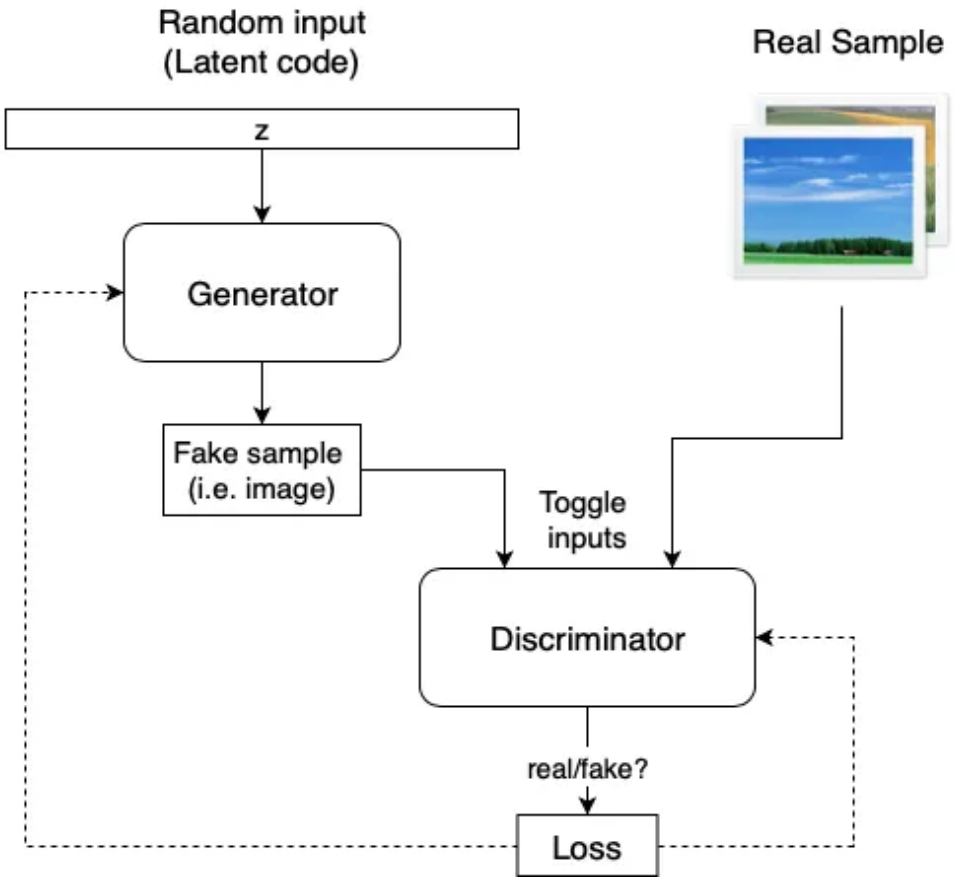
The basic components of every GAN are two neural networks

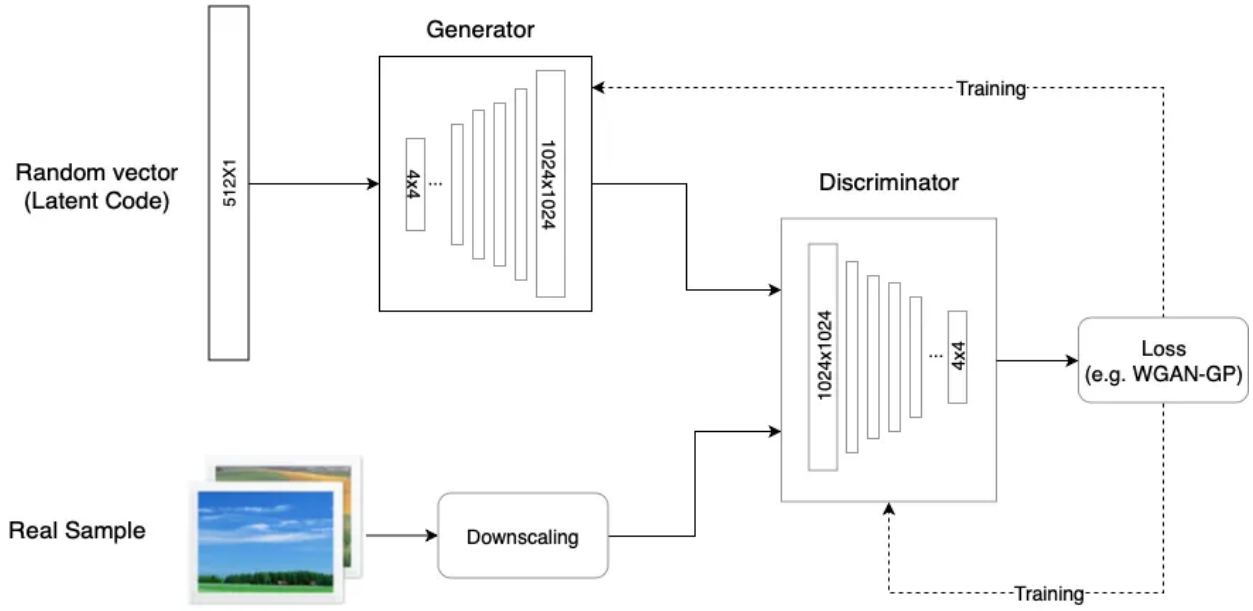
1. a generator : that synthesizes new samples from scratch,
2. discriminator : that takes samples from both the training data and the generator's output and predicts if they are "real" or "fake".

The generator input is a random vector (noise) and therefore its initial output is also noise.

Over time, as it receives feedback from the discriminator, it learns to synthesize more "realistic" images.

The discriminator also improves over time by comparing generated samples with real samples, making it harder for the generator to deceive it.





The generator model attempts to fool the discriminator and trains on more data to produce plausible results.

This architecture is adversarial because the generator and discriminator work against each other

one model tries to mimic reality while the other tries to identify fakes. These two components train simultaneously, improving their capabilities over time. They can learn to identify and reproduce complex training data such as image, audio, and video.

A GAN uses this basic workflow:

1. The generator ingests an input containing random numbers.
2. The generator processes the input to produce an image.
3. The discriminator ingests the image generated by the generator and additional, real images.
4. The discriminator compares the entire image set and attempts to determine which images are real or fake.
5. The discriminator returns a prediction for each image, using a number between 0 and 1 to express the probability of authenticity. A score of 0 indicates a fake image, while 1 indicates a real image.

This workflow creates a continuous feedback loop. The discriminator determines the ground truth (empirical truth) for image inputs, and the generator feeds the discriminator new and improved generated images.

ProGAN:

This technique first creates the foundation of the image by learning the base features which appear even in a low-resolution image, and learns more and more details over time as the resolution increases. Training the low-resolution images is not only easier and faster, it also helps in training the higher levels, and as a result, total training is also faster.

Limitations of ProGAN:

ProGAN generates high-quality images but, as in most models, its ability to control specific features of the generated image is very limited. In other words, the features are entangled and therefore attempting to tweak the input, even a bit, usually affects multiple features at the same time.

Optimizers

- **GD**

- **SGD (Stochastic Gradient Descent)**
  SGD used in linear regression (y - y^)2 for each separate datapoint

- **Mini Batch SGD**
  Useful for load balancing of a large dataset by selecting K data points.

- **Adaptive Gradient Algorithm**
  (AdaGrad) that maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g. natural language and computer vision problems).

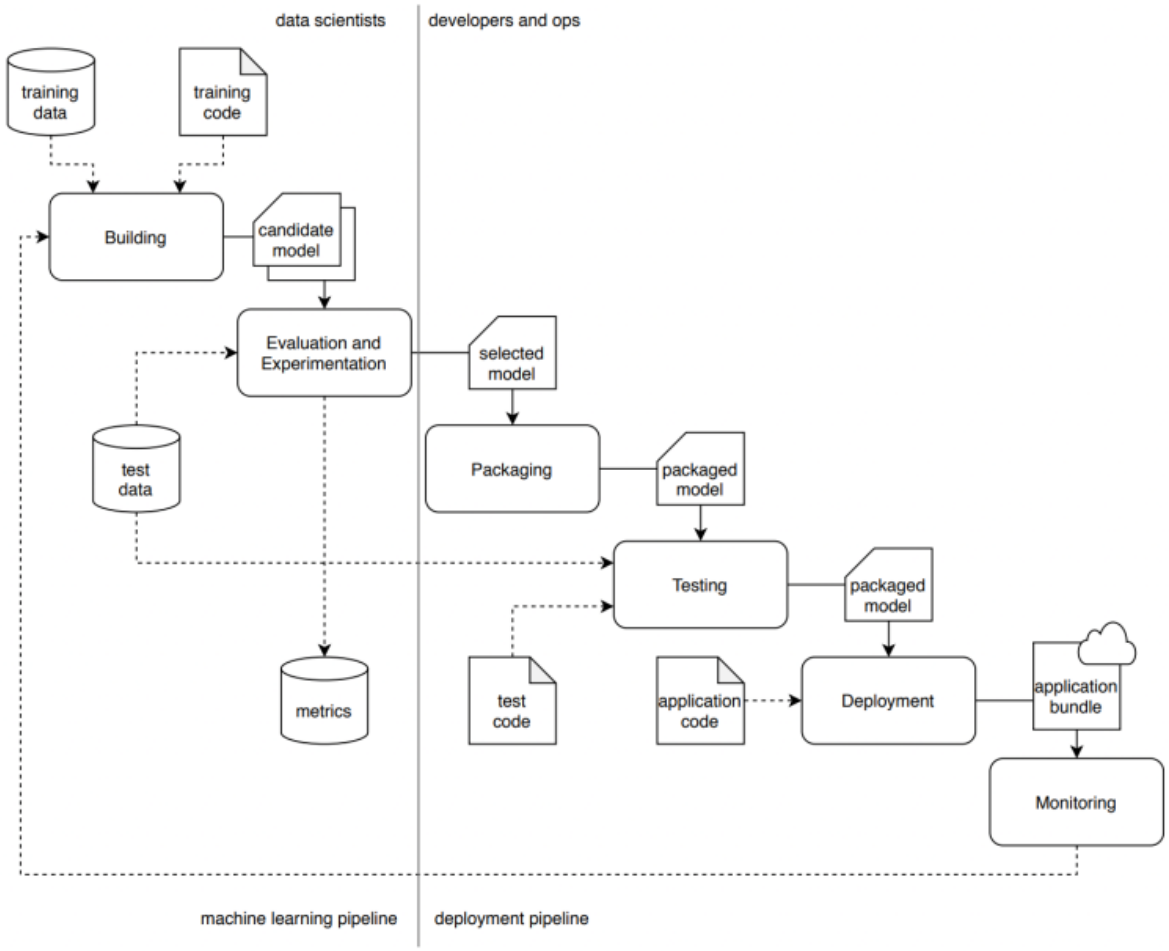
- **Root Mean Square Propagation (By Default optimizer in model.compile( ))**
  (RMSProp) that also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing)

- **Adam**
  The Adam optimization algorithm is an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications in computer vision and natural language processing.

  For multivariate analysis, we can use the pairplot() method of the seaborn module. We can also use it for the multiple pairwise bivariate distributions in a dataset.

Model Deployment

Model deployment (release) is a process that enables you to integrate machine learning models into production to make decisions on real-world data.
Once deployed, the model further needs to be monitored to check whether the whole process of data ingestion, feature engineering, training, testing etc. are aligned properly so that no human intervention is required and the whole process is automatic.
There are various strategies one can use before deploying the ML model and

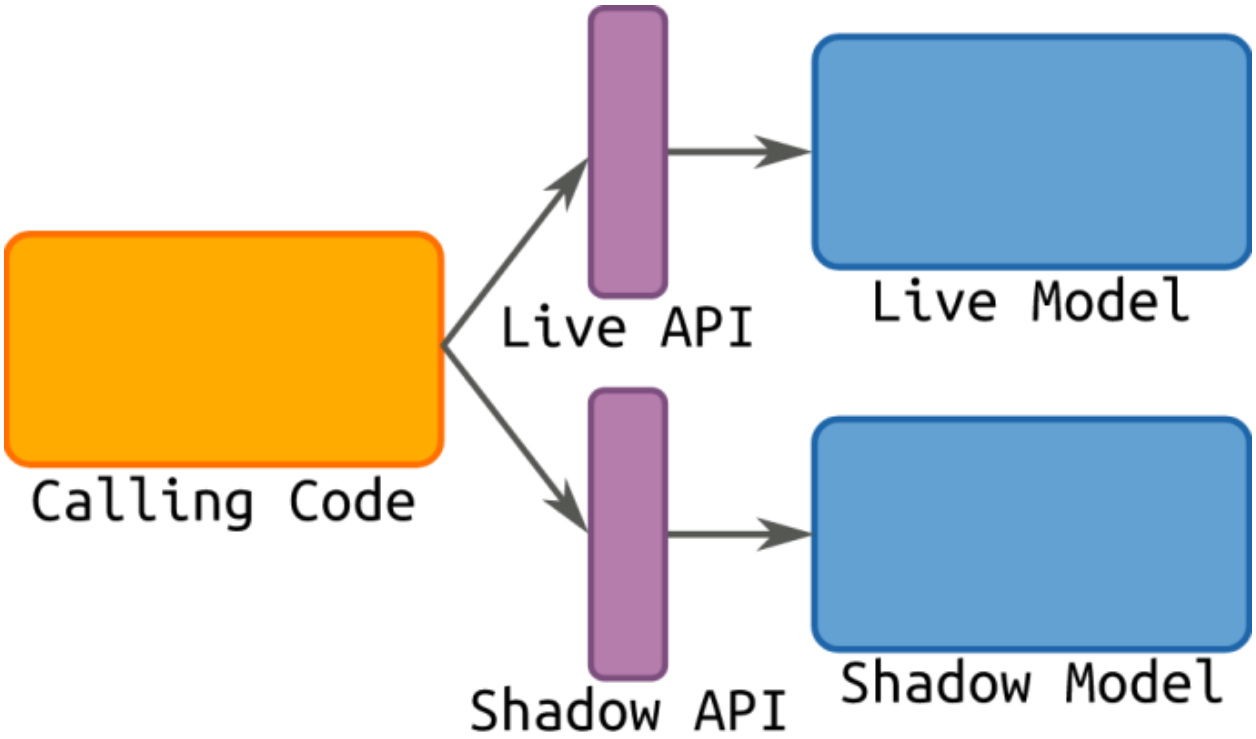


Model deployment strategies

1. Shadow deployment strategy

In shadow deployment or shadow mode, the new model is deployed with new features alongside the live model. The new deployed model in this case is known as a shadow model. The shadow model handles all the requests just like the live model except it is not released to the public.

This strategy allows us to evaluate the shadow model better by testing it on real-world data while not interrupting the services offered by the live model.



Q : What is Champion Challenger and How does It Enable Choosing the Right Decision?

When to use it?

- If you want to compare multiple models with each other then shadow testing is great, although tedious.
- Shadow testing will allow you to evaluate the pipeline, latency while yielding results as well the load-bearing capacity.

2. A/B testing model deployment strategy

A/B testing is a data-based strategy method. It is used to evaluate two models namely A and B, to assess which one performs better in a controlled environment. It is primarily used in e-commerce websites and social media platforms.

Methodology

In A/B the two models are set up parallelly with different features. The aim is to increase the **conversion rate** of a given model. In order to do that data scientist sets up a hypothesis. A hypothesis is an assumption based on an abstract intuition of the data. This assumption is proposed through an experiment, if the assumption passes the test it is accepted as fact and the model is accepted, otherwise, it's rejected.

Advantages:

- It is simple.
- Yields quick results and helps in the elimination of the low performing model.

Disadvantages:

- Models can be unreliable if the complexity is increased. One should use A/B testing in the case of simple hypothesis testing.

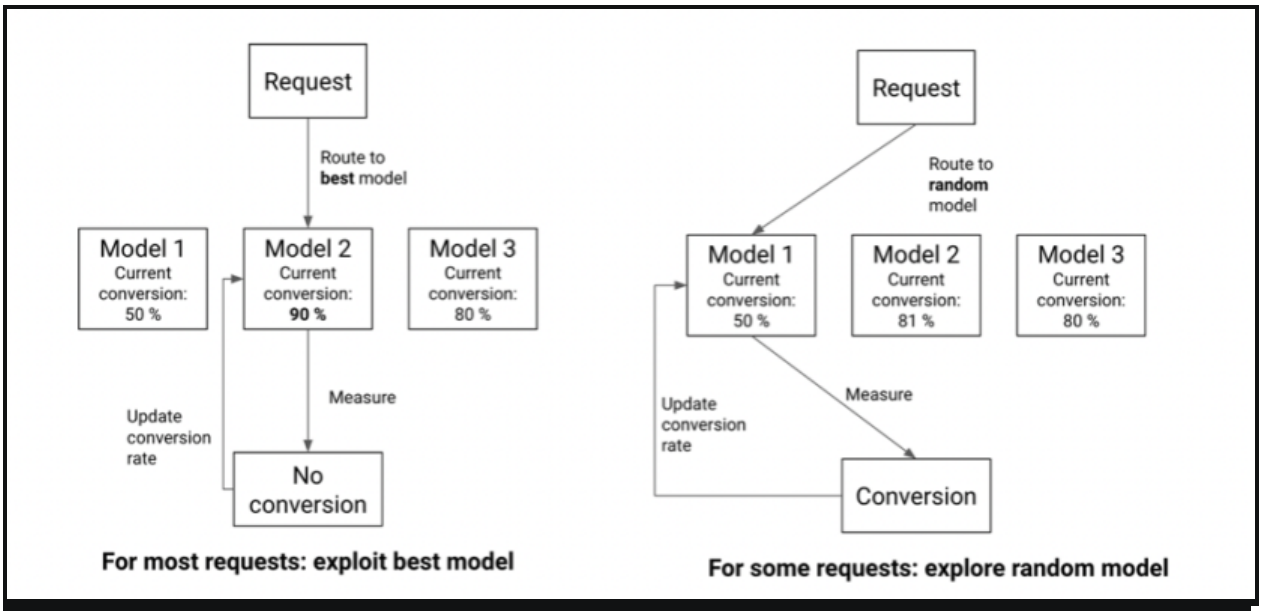
When to use it?

As mentioned earlier, A/B testing is predominantly used for e-commerce, social media platforms, and online streaming platforms. In such a setting and if you have two models you can use A/B to evaluate and choose which one to deploy globally.

3. Multi Armed Bandit

Multi-Armed Bandit or MAB is an advanced version of A/B testing. It is also inspired by reinforcement learning, and the idea is to explore and exploit the environment that maximizes the reward function.

MAB leverages machine learning to explore and exploit the data received to optimize the key performance index (KPI). The advantage of using this technique is that the user traffic is diverted according to the KPI of two or more models.



Advantages:

- With exploring and exploiting the MAB offers adaptive testing.
- Resources are not wasted like in A/B testing.
- Faster and efficient way of testing.

Disadvantages:

- It is expensive because exploiting takes a lot of computing power which can be economically expensive.

When to use it?

MAB is very helpful for scenarios where the conversion rate is all you care about and where the time to make a decision is small. For example, optimizing offers or discounts on a product for a limited period.

4. Blue-green deployment strategy

Blue-green deployment strategies involve two production environments instead of just models. The blue environment consists of the live model whereas the green environment consists of the new version of the model

The green environment is set as a staging environment i.e. an exact replica of a live environment but with new features. Let us briefly understand the methodology.

Methodology

In Blue-green deployment, the two identical environments consist of the same database, containers, virtual machines, same configuration et cetera. Keep in mind that setting up an environment can be expensive so usually, some components like a database are shared between the two.

The Blue environment which contains the original model is live and keeps servicing requests while the Green environment acts as a staging environment for a new version of the model. It is subjected to deployment and final stages of testing against the real data to ensure that it performs well and is ready to deploy to production. Once the testing is successfully completed ensuring that all the bugs and issues are rectified the new model is made live.

Once this model is made live, the traffic is diverted from the blue environment to the green environment.

When to use it?

In case your application cannot afford downtime then one should use the Blue-Green deployment strategy.

Comparison: which model release strategy to use?

Deployment or testing pattern	Zero downtime	Real production traffic testing	Releasing to users based on conditions	Rollback duration	Impact on hardware and cloud costs
Recreate Version 1 is terminated, and Version 2 is rolled out.	x	x	x	Fast but disruptive because of downtime	No extra setup required
Rolling update Version 2 is gradually rolled out and replaces Version 1.	✓	x	x	Slow	Can require extra setup for surge upgrades
Blue/green Version 2 is released alongside Version 1; the traffic is switched to Version 2 after it is tested.	✓	x	x	Instant	Need to maintain blue and green environments simultaneously
Canary Version 2 is released to a subset of users, followed by a full rollout.	✓	✓	x	Fast	No extra setup required
A/B Version 2 is released, under specific conditions, to a subset of users.	✓	✓	✓	Fast	No extra setup required
Shadow Version 2 receives real-world traffic without impacting user requests.	✓	✓	x	Does not apply	Need to maintain parallel environments in order to capture and replay user requests

Key takeaways

Deployment strategies often help data scientists to figure out how their model is performing in a given situation. A good strategy depends upon the type of product and users it aims to target. To sum it up, here are the points one should keep in mind:

- If you want the model to be tested in real-world data then a shadow evaluation strategy or something similar to it must be considered. Unlike the other strategies where the sample of users are used, the shadow evaluation strategy uses live and real user requests.
- Check the complexity of the task, if the model requires simple or minor tweaks then A/B testing is the way to go.
- If there is time constraint and ideas are more, then one should opt for multiarm bandits since it gives you the best results in such a situation.

- If your model is complex and needs proper monitoring before deploying then Blue-green strategy will help you analyse and monitor your model.
- If you want no downtime and you are okay to expose your model to the public then opt for Canary deployment.
- The rolling deployment must be used when you want to gradually deploy the new version of the model.
-

Versioning and Scaling Management Libraries

MLFlow

MLflow is an open source platform for managing the end-to-end machine learning lifecycle. It tackles four primary functions:

MLflow is an open source platform to manage machine learning life-cycles. The platform offers four distinct components, which can be used either in stand-alone mode or together.

MetaFlow

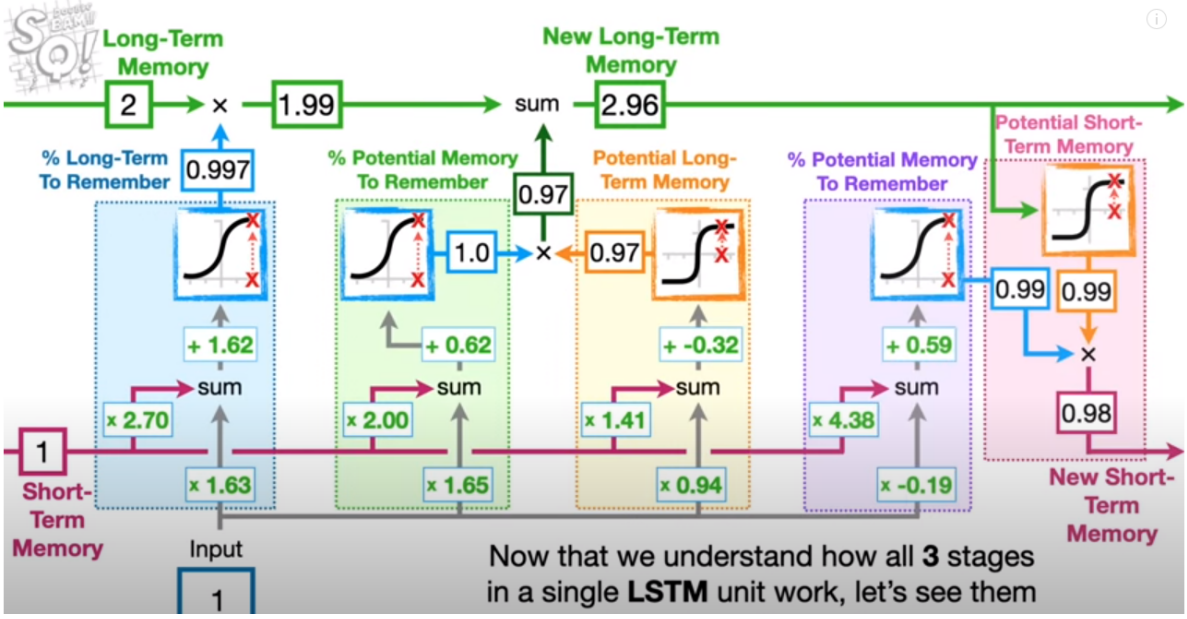
Metaflow is a human-friendly Python library that makes it straightforward to develop, deploy, and operate various kinds of data-intensive applications, in particular those involving data science and ML.

Metaflow is a python library, originally developed at Netflix that helps building and managing data science projects.

Metaflow provides a robust and user-friendly foundation for a wide spectrum of data-intensive applications, including most data science and ML use cases
Metaflow takes care of the low-level infrastructure: data, compute, orchestration, and versioning.
Metaflow provides a unified API to the whole infrastructure stack that is required to execute data science projects from prototype to production.

comparison checklist documentation for MLFlow vs. Metaflow vs. DVC.:
https://github.com/hzdr/mlops_comparison/blob/master/Content/Comparison_table.pdf

LSTM



	Simple Neural Network	Convolutional Neural Network (CNN)	Long Short-Term Memory Network (LSTM)
Intuition	Input data fed forward to dense layer, only capable to learn simple patterns	Ability to extract spatial features from input data using self-learning filters	Capable of learning & remembering patterns across time
Structure (specific to this project)	Embedding layer >> Flatten >> Dense layer >> Output	Embedding layer >> Convolution + Pooling >> Dense layer >> Output	Embedding layer >> LSTM >> Dense layer >> Output
Speciality	Can handle incomplete knowledge, high fault tolerance	Accuracy in recognizing images	Memory and self-learning
Data Type	Fed on tabular and text data	Relies on image data	Trained with sequence data
Applications	Simple problem solving such as predictive analysis, prone to Overfitting	Image/Video Analytics, Speech Recognition and understanding natural language processing	Machine Translation, Language Modeling and Multilingual Language Processing

Deep Learning Libraries

TensorFlow

TensorFlow was developed by Google and is one of the older Deep Learning libraries, ported across many languages since it was first released to the public in 2015. It is very versatile and capable of much more than Deep Learning but as a result it often takes a lot more lines of code to write Deep Learning operations in TensorFlow than in other libraries. It offers (almost) seamless integration with GPU accelerators and Google's own TPU (Tensor Processing Unit) chips that are built specially for machine learning.

PyTorch

PyTorch was developed by Facebook in 2016 and is a popular choice for Deep Learning applications. It was developed for Python from the start and feels a lot more "pythonic" than TensorFlow. Like TensorFlow it was designed to do more than just Deep Learning and offers some very low level interfaces. PyTorch Lightning offers a higher level interface to PyTorch to set up experiments. Like TensorFlow it's also very easy to integrate PyTorch with a GPU. In many benchmarks it outperforms the other libraries.

Keras

Keras is designed to be easy to use and usually requires fewer lines of code than other libraries. We have chosen it for this workshop for that reason. Keras can actually work on top of TensorFlow (and several other libraries), hiding away the complexities of TensorFlow while still allowing you to make use of their features.

The performance of Keras is sometimes not as good as other libraries and if you are going to move on to create very large networks using very large datasets then you might want to consider one of the other libraries. But for many applications the performance difference will not be enough to worry about and the time you'll save with simpler code will exceed what you'll save by having the code run a little faster.

Types of Layers and their usefulness

What are Dense layers and when are they useful?
Dense layers are used when association can exist among any feature to any other feature in a data point. Since between two layers of size n1 and n2, there can n1∗n2 connections and these are referred to as Dense.

Where do Conv layers come in and when are they useful?

Coming to the conv layers, these are important when nearby associations among the features matter, example object detection. Neighborhoods matter to classify or detect. It is very less likely that the pixels at the opposite corners(very far away) are somehow helpful in these use cases. Filters do this job of getting associations among neighborhoods. This answer is great at understanding the difference between 1D and 2D convolutions
https://stackoverflow.com/questions/42883547/intuitive-understanding-of-1d-2d-and-3d-convolutions-in-convolutional-neural-n

Dropout and Flatten

Dropout is a way of cutting too much association among features by dropping the weights (edges) at a probability. The original paper from Hinton et.al is a quick and great read to grasp it. Reducing associations can be applied among any layers which stops weight updation for the edge. The other key difference here is it has no weights associated with it. It is just there dropping things.

Flatten layers are used when you have a multidimensional output and you want to make it linear to pass it onto a Dense layer. If you are familiar with numpy, it is equivalent to numpy.ravel. An output from flatten layers is passed to an MLP for classification or regression tasks you want to achieve. No weighting are associated with these too. It is just flattening the hell out.

Transfer Learning

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.

Transfer learning is a machine learning technique where a model trained on one task is re-purposed on a second related task.

Transfer learning only works in deep learning if the model features learned from the first task are general.

reusing or transferring information from previously learned tasks for the learning of new tasks has the potential to significantly improve the sample efficiency.

Transfer learning has also been applied to cancer subtype discovery, building utilization, general, game playing, text classification, digit recognition, medical imaging and spam filtering.

The very last classification layer (on "top", as most diagrams of machine learning models go from bottom to top) is not very useful. Instead, you will follow the common practice to depend on the very last layer before the flatten operation. This layer is called the "bottleneck layer". The bottleneck layer features retain more generality as compared to the final/top layer.

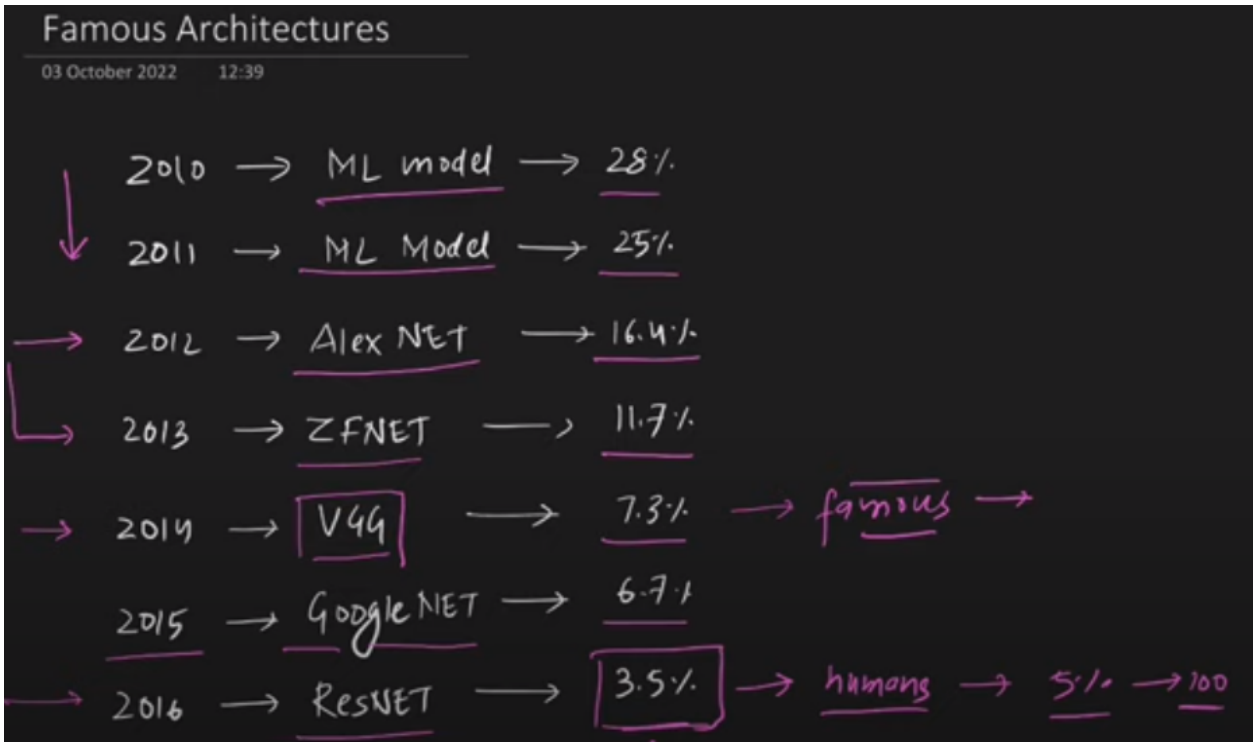
Many models contain tf.keras.layers.BatchNormalization layers. This layer is a special case and precautions should be taken in the context of fine-tuning, as shown later in this tutorial.

tf.keras.layers.Dense layer to convert these features into a single prediction per image. Flattening is converting the data into a 1-dimensional array for inputting it to the next layer.

If you are wondering why the validation metrics are clearly better than the training metrics, the main factor is because layers like tf.keras.layers.BatchNormalization and tf.keras.layers.Dropout affect accuracy during training. They are turned off when calculating validation loss.

How to Use Transfer Learning?

1. Develop Model Approach [1. **Select Source Task**. 2. **Develop Source Model** 3. **Fit Model** 4. **Tune Model**.]
2. Pre-trained Model Approach(most common) [1.**Select Source Model** 2. **Reuse Model** 3. **Tune Model**]



Available models

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27.0	6.7
NASNetLarge	343	82.5%	96.0%	88.9M	533	344.5	20.0

EfficientNetB0	29	77.1%	93.3%	5.3M	132	46.0	4.9
EfficientNetB1	31	79.1%	94.4%	7.9M	186	60.2	5.6
EfficientNetB2	36	80.1%	94.9%	9.2M	186	80.8	6.5
EfficientNetB3	48	81.6%	95.7%	12.3M	210	140.0	8.8
EfficientNetB4	75	82.9%	96.4%	19.5M	258	308.3	15.1
EfficientNetB5	118	83.6%	96.7%	30.6M	312	579.2	25.3
EfficientNetB6	166	84.0%	96.8%	43.3M	360	958.1	40.4
EfficientNetB7	256	84.3%	97.0%	66.7M	438	1578.9	61.6
EfficientNetV2B0	29	78.7%	94.3%	7.2M	-	-	-
EfficientNetV2B1	34	79.8%	95.0%	8.2M	-	-	-
EfficientNetV2B2	42	80.5%	95.1%	10.2M	-	-	-
EfficientNetV2B3	59	82.0%	95.8%	14.5M	-	-	-
EfficientNetV2S	88	83.9%	96.7%	21.6M	-	-	-
EfficientNetV2M	220	85.3%	97.4%	54.4M	-	-	-
EfficientNetV2L	479	85.7%	97.5%	119.0M	-	-	-
ConvNeXtTiny	109.42	81.3%	-	28.6M	-	-	-
ConvNeXtSmall	192.29	82.3%	-	50.2M	-	-	-
ConvNeXtBase	338.58	85.3%	-	88.5M	-	-	-
ConvNeXtLarge	755.07	86.3%	-	197.7M	-	-	-
ConvNeXtXLarge	1310	86.7%	-	350.1M	-	-	-

Transfer Learning with Image Data

- Oxford VGG Model
- Google Inception Model
- Microsoft ResNet Model

Transfer Learning with Language Data

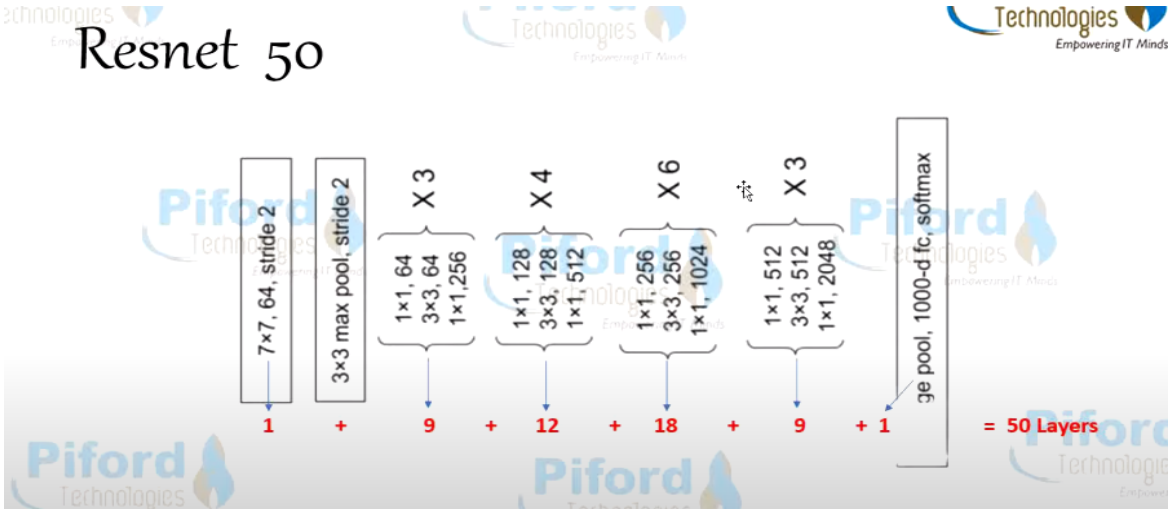
- Google's word2vec Model
- Stanford's GloVe Model

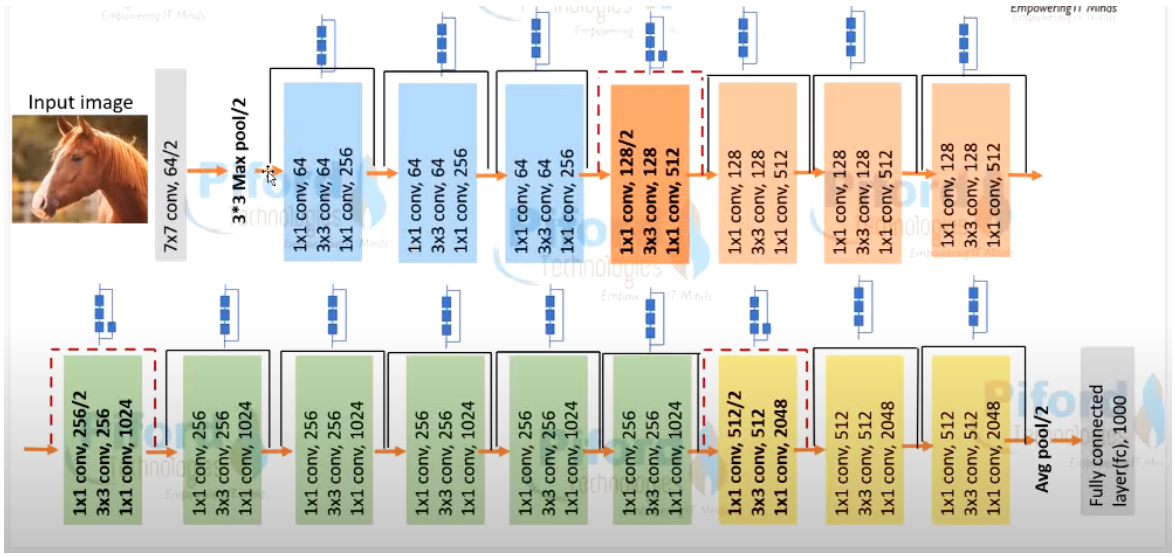
RESNET

Keras Applications include the following ResNet implementations and provide ResNet V1 and ResNet V2 with 50, 101, or 152 layers:

- ResNet50
- ResNet101
- ResNet152
- ResNet50V2
- ResNet101V2
- ResNet152V2

The primary difference between ResNetV2 and the original (V1) is that V2 uses batch normalization before each weight layer.





FIRST LAYER:

Input Image: 300 X 300 X 3 (RGB)

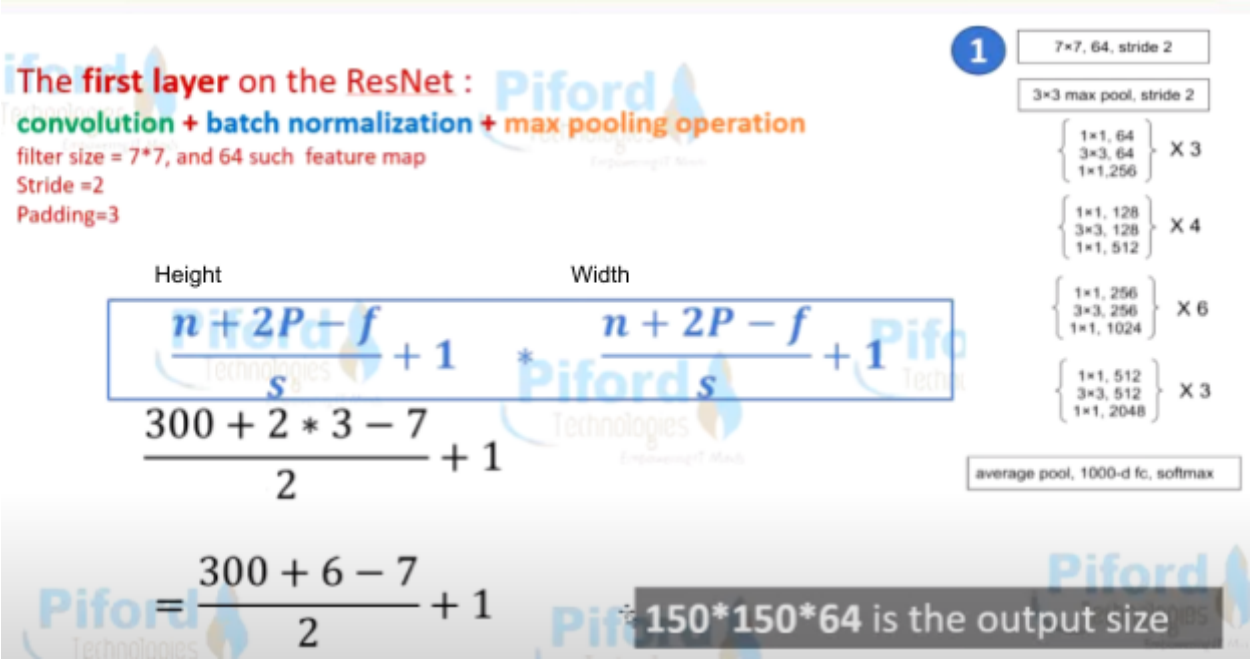
n = size of image(will be differ height width)

P = Padding

f = filter size

S = Stride

We can reduce size of image by stride



The advantage of adding this type of skip connection is that if any layer hurt the performance of architecture then it will be skipped by regularization. So, this results in training a very deep neural network without the problems caused by vanishing/exploding gradients.

VGG

VGG is a convolutional neural network architecture developed by researchers at the **Visual Geometry Group (VGG)** at the University of Oxford. It was introduced in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" published in 2014.

What is data drift?

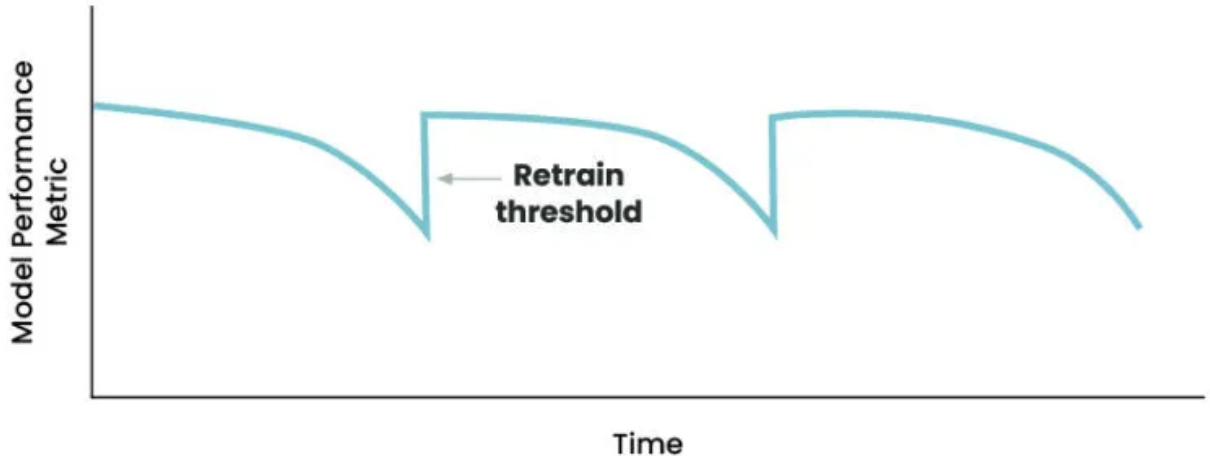
- "Drift" is a term used in machine learning to describe how the performance of a machine learning model in production slowly gets worse over time.
- This can happen for a number of reasons, such as changes in the distribution of the input data over time or the relationship between the input (x) and the desired target (y) changing.
- where data is often dynamic and always changing.
- Machine learning models are trained with historical data, but once they are used in the real world, they may become outdated and lose their accuracy over time due to a phenomenon called drift. Drift is the change over time in the statistical properties of the data that was used to train a machine learning model.
- drift detection is a key step in the ML lifecycle and hence it should not be an afterthought it should be **part of your plan to deploy** the model in production, it should be **automated**, careful thought must be given to **identify the drift methodology**, **thresholds** to apply and **actions** to be taken when a drift is detected.

Types of Drift

Let's explore the two different types of drift to consider:

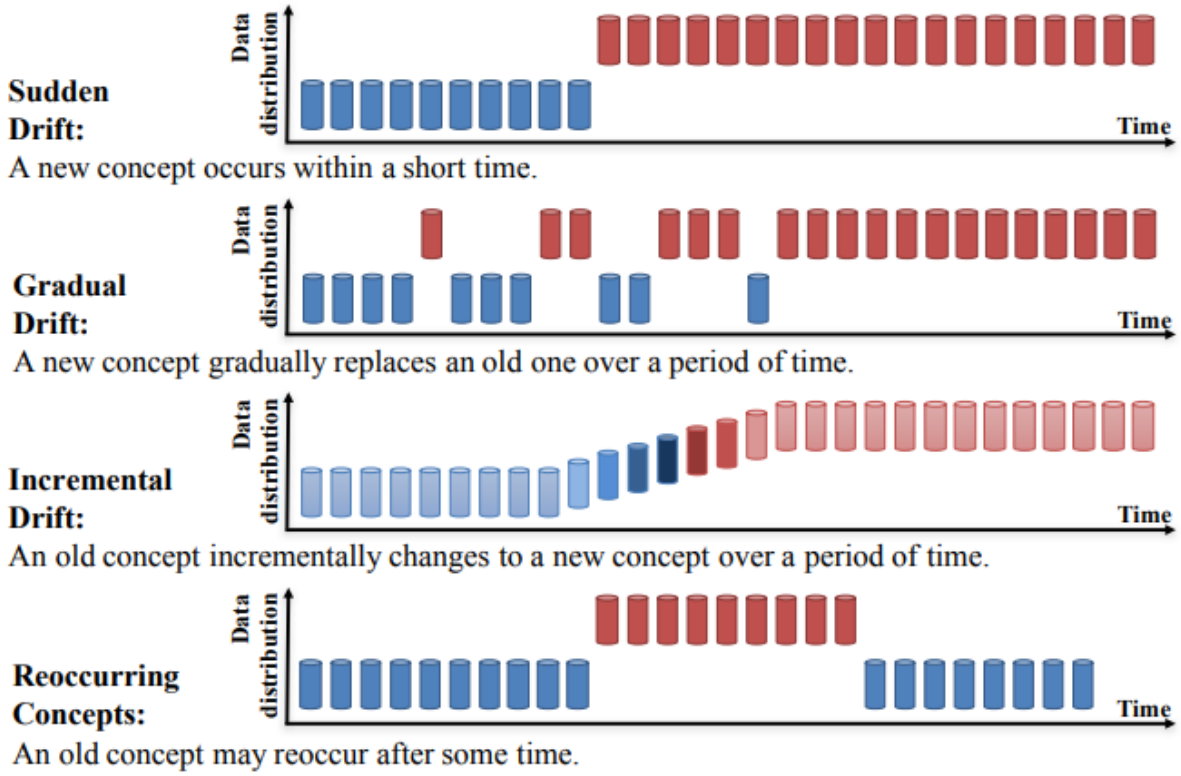
1. Concept Drift

Concept drift, also known as model drift, occurs when the task that the model was designed to perform changes over time. For example, imagine that a machine learning model was trained to detect spam emails based on the content of the email. If the types of spam emails that people receive change significantly, the model may no longer be able to accurately detect spam.



Concept Drift can be further divided into four categories (Learning under Concept Drift: A Review, Jie Lu et al.):

- Sudden Drift
- Gradual Drift
- Incremental Drift
- Recurring Concepts



2. Data Drift

Data drift, also known as covariate shift, occurs when the distribution of the input data changes over time. For example, consider a machine learning model that was trained to predict the likelihood of a customer purchasing a product based on their age and income. If the distribution of ages and incomes of the customers changes significantly over time, the model may no longer be able to predict the likelihood of a purchase accurately.

It is important to be aware of both concept drift and data drift and take steps to prevent or mitigate their effects.

Algorithms for Detecting Data Drift

Kolmogorov-Smirnov (K-S) test

The Kolmogorov-Smirnov (K-S) test is a nonparametric statistical test that is used to determine whether two sets of data come from the same distribution. It is often used to test whether a sample of data comes from a specific population or to compare two samples to determine if they come from the same population.

The null hypothesis in this test is that the distributions are the same. If this hypothesis is rejected, it suggests that there is a drift in the model.

The K-S test is a useful tool for comparing datasets and determining whether they come from the same distribution.

Population Stability Index

The Population Stability Index (PSI) is a statistical measure that is used to compare the distribution of a categorical variable in two different datasets.

The Population Stability Index (PSI) is a tool used to measure how much the distribution of a variable has changed between two samples or over time. It is commonly used to monitor changes in the characteristics of a population and to identify potential problems with the performance of a machine learning model.

The PSI was originally developed to monitor changes in the distribution of a score in risk scorecards, but it is now used to examine distributional shifts for all model-related attributes, including both dependent and independent variables.

A high PSI value indicates that there is a significant difference between the distributions of the variable in the two datasets, which may suggest that there is a drift in the model.

If the distribution of a variable has changed significantly, or if several variables have changed to some extent, it may be necessary to recalibrate or rebuild the model to improve its performance.

Data Drift Detection Framework

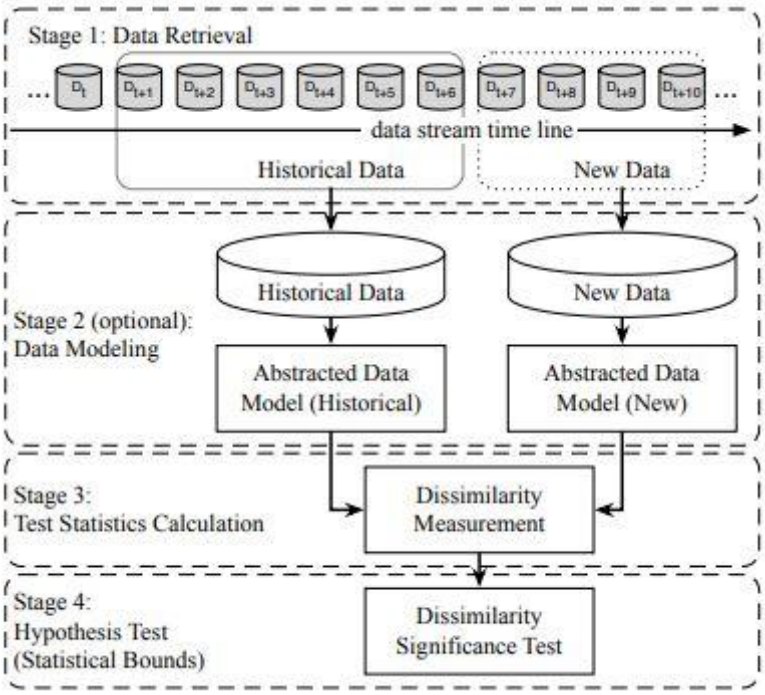


Image source: https://arxiv.org/pdf/2004.05785.pdf

Stage 1 (Data Retrieval) is used to retrieve data from data streams in chunks since a single data point cannot carry enough information to infer the overall distribution.

Stage 2 (Data Modeling) is used to extract the key features, that is, the features of the data that most impact a system if they drift.

Stage 3 (Test Statistics Calculation) is to measure the drift and calculate test statistics for the hypothesis test.

Stage 4 (Hypothesis Test) is used to evaluate the statistical significance of the change observed in Stage 3 or the p-value.

Using specialized drift detection techniques such as Adaptive Windowing (ADWIN):

The Adaptive Windowing (ADWIN) algorithm uses a sliding window approach to detect concept drift. Window size is fixed and ADWIN slides the fixed window for detecting any change on the newly arriving data. When two sub-windows show distinct means in the new observations the older sub-window is dropped.

A user-defined threshold is set to trigger a warning that drift is detected. If the absolute difference between the two means derived from two sub-windows exceeds the pre-defined threshold, an alarm is generated. This method is applicable for univariate data.

```
from skmultiflow.drift_detection import ADWIN

adwin = ADWIN()

for col in df_numerical.columns:

  data_stream=[]

  a = np.array(df_salary_low[col])

  b = np.array(df_salary_high[col])

  data_stream = np.concatenate((a,b))

# Adding stream elements to ADWIN and verifying if drift
occurred

  for i in range(len(data_stream)):

    adwin.add_element(data_stream[i])
```

```
    if adwin.detected_change():

        print('Change detected in data: ' +
str(data_stream[i]) + ' - at index: ' + str(i) +'for column:'
+ col)
```

```
Change detected in data: 2 - at index: 111for column:Tenure
Change detected in data: 5 - at index: 143for column:Tenure
Change detected in data: 5 - at index: 239for column:Tenure
Change detected in data: 3 - at index: 303for column:Tenure
Change detected in data: 7 - at index: 335for column:Tenure
Change detected in data: 164870.81 - at index: 31for column:Balance
Change detected in data: 117412.19 - at index: 63for column:Balance
Change detected in data: 0.0 - at index: 95for column:Balance
Change detected in data: 157780.93 - at index: 127for column:Balance
Change detected in data: 8816.37 - at index: 175for column:EstimatedSalary
Change detected in data: 2085.32 - at index: 207for column:EstimatedSalary
Change detected in data: 5078.9 - at index: 271for column:EstimatedSalary
Change detected in data: 9241.52 - at index: 303for column:EstimatedSalary
Change detected in data: 9087.81 - at index: 463for column:EstimatedSalary
Change detected in data: 2850.01 - at index: 495for column:EstimatedSalary
Change detected in data: 38131.77 - at index: 623for column:EstimatedSalary
Change detected in data: 100127.71 - at index: 975for column:EstimatedSalary
Change detected in data: 15766.1 - at index: 1135for column:EstimatedSalary
```

Drift Detection Implementation in Python

In this section, we will use **Evidently** to detect drift. Evidently is an open-source Python library made for data scientists and engineers who work with machine learning. It helps them test, evaluate, and keep track of how well their models work from validation to production.

The statistical distance metrics that are available are **Wasserstein distance** (numerical features) and **Euclidian distance** (categorical features). As you may notice the drift is above the threshold and the percentage drift for each of the top features is shown at the top.

What is machine learning model drift?

Machine learning model drift is when a model's performance on new data is different from how it performed on the training data it was built on. This can happen for a variety

of reasons, including changes in the distribution of data over time, the addition of new data that doesn't fit the original model's assumptions, or the model's own inability to adapt to changing conditions.

Why is model drift a problem?

Model drift can significantly impact the performance and accuracy of a machine learning model. As the model's predictions become less reliable, it can lead to incorrect decisions or actions that can have negative consequences. For example, in a healthcare setting, model drift could lead to incorrect diagnoses or treatment recommendations, while in a finance setting, it could result in poor investment decisions.

How do you detect model drift?

There are several ways to determine if a model is drifting, such as statistical tests, drift detection algorithms, and looking at how well the model is doing. Some of these methods are made to find drift in real-time, while others are better for testing at set times or in groups. It's important to choose the right technique for the specific application and data environment.

How do you prevent model drift?

Preventing model drift requires a combination of careful model selection, regular monitoring and testing, and proactive intervention. This may involve using algorithms that are more robust to drift, regularly retraining models on new data, or implementing strategies to actively address drift when it is detected. It's also important to have a clear understanding of the factors that can cause drift so that you can take steps to prevent them.

How does data distribution affect model drift?

The data distribution can significantly affect the performance of a machine learning model. If the distribution of data changes over time, it can lead to model drift, as the model may no longer be able to accurately predict new data that doesn't match its

original assumptions. This can happen in a variety of ways, such as through natural variations in the data, the addition of new data sources, or changes in the underlying processes or systems that generate the data.

Is model drift reversible?

In some cases, model drift can be reversible by retraining the model on new data or adjusting its parameters. However, this is not always possible, especially if the data distribution has changed significantly or the model has become overly complex or specialized. In these situations, it may be necessary to start over with a new model.

Is it possible to completely eliminate model drift?

Completely eliminating model drift is difficult, if not impossible. Even the most robust and well-designed machine learning models can be affected by changes in the data or the underlying processes that generate it. The best approach is to manage and mitigate the impact of model drift through regular monitoring, testing, and intervention.

How does model drift impact model performance?

Model drift can have a significant impact on the performance of a machine learning model. As the model's predictions become less accurate, it can lead to reduced performance on important metrics such as accuracy, precision, recall, and overall model effectiveness. In some cases, model drift can even cause a model to fail completely, resulting in incorrect or unreliable predictions.

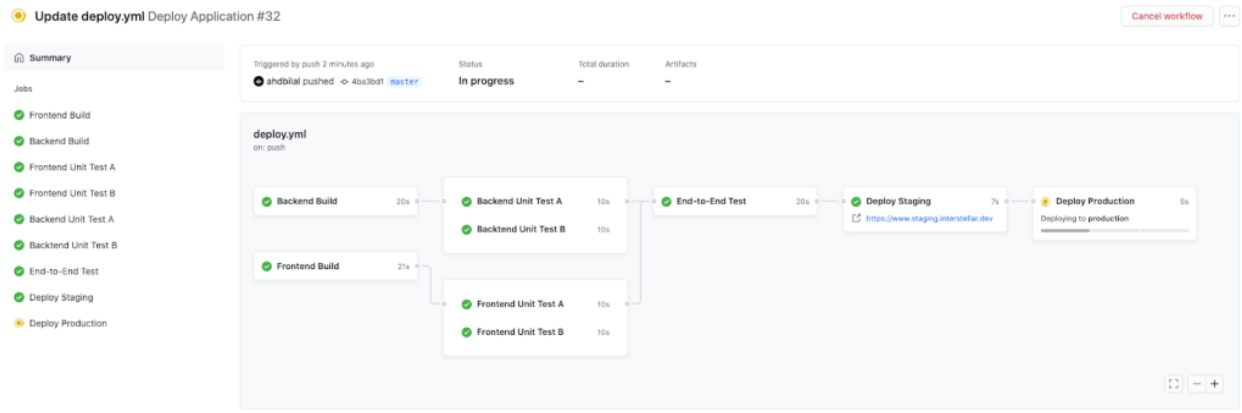
How does model drift affect model accuracy?

Model drift can have a negative impact on the accuracy of a machine learning model. As the model's predictions become less accurate, it can lead to incorrect decisions or actions, which can have negative consequences in real-world applications. For example, in a healthcare setting, model drift could lead to incorrect diagnoses or treatment recommendations, while in a finance setting, it could result in poor investment decisions. It's important to regularly monitor and test for model drift in order to maintain the accuracy of the model.

How do you use CI/CD in your workflow?

The **CICD (Continuous Integration and Continuous Deployment)** practice assures that the life cycle of a software application adheres to predetermined stages for writing code, building, testing, and deploying applications.

In other words, the CICD pipeline is a well-structured framework for developing a full-stack software project. The CICD approach provides continuous integration and delivery, leading to continuous deployment of a software application when a new update is released by the organization.



Explain the semantic search project?

Semantic search is the task of retrieving documents from a collection of documents (also known as a 'corpus') in response to a query asked in natural language.

Haystack's modular setup and the availability of high-quality pre-trained language models, you'll be able to set up your own semantic search system

Semantic search is a **data searching technique** in which a search query aims to not only find keywords but to **determine the intent**

and contextual meaning of the words a person is using for a search.
The idea behind semantic search is to embed all entries in your corpus, which can be sentences, paragraphs, or documents, into a vector space.

models tuned for cosine-similarity will prefer the retrieval of short documents, while models tuned for dot-product will prefer the retrieval of longer documents

Q1. What sort of embeddings will work ?
Q2. How to store documents and their huge embeddings if using BERT?

Solution to Q2

At this point, we have an understanding of our data and accordingly, we have selected the embeddings model. Now next we need to understand how to encode data and what other information we need to store with encodings will be helpful in retrieving search results.

For storage, we have options like :

(a) ***ElasticSearch****: It can be a very good option if we have a lot of meta-information storage and we want to run some cross-cluster search. Having said that it can cost you a lot and maintenance can*

cost you another expert if shipping things to production and scale.

(b) ***FAISS****: (Facebook AI Similarity Search) is a library that allows developers to quickly search for embeddings of multimedia documents that are similar to each other. It solves the limitations of traditional query search engines that are optimized for hash-based searches and provides more scalable similarity search functions.*

(c) ***Annoy****: a C++ library with Python bindings to search for points in space that are close to a given query point. It also creates large read-only file-based data structures that are mapped into memory so that many processes may share the same data.*

Asymmetric Search and trying to retrieve long passages so for our case, a dot-product model will suit well. And for this experiment, i am choosing ***"sentence-transformers/msmarco-distilbert-base-dot-prod-v3"*** *model which performs great in Semantic Textual Similarity (Asymmetric) tasks and it's quite faster than BERT as it is considerably smaller.*

Q3. What if we have long documents like blogs and small pieces of content like product descriptions? How will the approach change?

Q4. How model fine-tuning can give me good results ?
We could have easily fine-tuned a sentence-transformer model on our dataset given if we had ***query & relevant passages information. But you would not have this data if building something from ground zero.***

Can we devise some unsupervised approach to fine-tune our model on our dataset?

1. Synthetic Query Generation

We use synthetic query generation to achieve our goal. We start with the passage from our document collection and create for these possible queries users might ask / might search for.

BEIR: A Heterogenous Benchmark for Zero-shot Evaluation of Information Retrieval Models presented a method to learn (or adapt) model for asymmetric semantic search without requiring training data.

Which loss function is suitable depends on the available training data and on the target task.

Keyword Search Vs Semantic Search

At first, search engines were lexical: the search engine looked for literal matches of the query words, without understanding of the query's meaning and only returning links that contained the exact query.By using regular keyword search, a document either contains the given word or not, and there is no middle ground

On the other hand, "Semantic Search" can simplify query building, becuase it is supported by automated natural language processing programs i.e. using Latent Semantic Indexing - a concept that search engines use to discover how a keyword and content work together to mean the same thing.

LSI adds an important step to the document indexing process. LSI examines a collection of documents to see which documents contain some of those same words. LSI considers documents that have many words in common to be semantically close, and ones with less words in common to be less close.

In brief, LSI does not require an exact match to return useful results. Where a plain keyword search will fail if there is no exact match, LSI will often return relevant documents that don't contain the keyword at all.

What is docker?

Docker is a containerization tool used for spinning up isolated, reproducible application environments.

The main problem with VM is that an "extra OS" on top of the host operating system adds gigabytes of space to the project. Most of the time your server will host several VMs that will take up even more space.

Another significant drawback of VM is a slow boot as, in addition, it runs identical processes that use CPU resources.
Docker eliminates all the above by simply sharing the OS kernel across all the containers running as separate processes of the host OS.

Why do we need Docker?

The short list of benefits includes:

- Faster development process
- Handy application encapsulation
- The same behavior on local machine / dev / staging / production servers
- Easy and clear monitoring
- Easy to scale

How you deployed flask API in GKR

1. Containerize the Flask Application: First, you need to containerize your Flask application. Create a Dockerfile that defines the container image for your Flask API. This Dockerfile should include the necessary dependencies, configurations, and instructions to run your Flask application.
2. Build the Docker Image: Build the Docker image by executing the appropriate Docker commands. For example, you can use the `docker build` command to build the image based on your Dockerfile. Make sure to tag the image with a suitable name and version.
3. Push the Docker Image to a Container Registry: Choose a container registry, such as Google Container Registry (GCR), and push the Docker image to the registry. This step allows your Kubernetes cluster to access the container image when deploying the application.
4. Create a Kubernetes Deployment: Define a Kubernetes Deployment configuration file that specifies how to deploy your Flask API as a Kubernetes Deployment. This file should include details such as the container image, port mappings, environment variables, and resource requirements.
5. Create a Kubernetes Service: Create a Kubernetes Service configuration file to expose your Flask API internally within the Kubernetes cluster. This file defines the service type (e.g., ClusterIP, NodePort, LoadBalancer), port mappings, and selectors to match your Flask API deployment.
6. Apply the Kubernetes Configurations: Use the `kubectl apply` command to apply the Kubernetes Deployment and Service configurations to your GKE cluster. This will create the necessary resources to run and expose your Flask API.
7. Verify the Deployment: Use `kubectl` commands to check the status of your deployment, pods, and services. Ensure that your Flask API pods are running and the services are accessible within the cluster.
8. Expose the API Externally: If you want to expose your Flask API externally, you can update the Service configuration to use a LoadBalancer type or set up an Ingress controller to route external traffic to your API service.
9. Monitor and Scale: Set up monitoring and logging solutions to track the performance and health of your Flask API in the Kubernetes environment. Configure scaling rules based on resource utilization and traffic patterns to automatically adjust the number of Flask API replicas.

What is KAFKA

What is the difference between MLOps and DevOps?

	DevOps	MLOps
Team skills	Software development skills such as version control and object orientated programming whilst working with delivery methodologies such as Agile.	Skills from roles such as data scientists and data engineers are also needed which include modelling, data-wrangling and experimentation.
Development	Known target to develop towards with a fairly linear development curve to completion.	Development is experimental in nature where differing modelling techniques, features and algorithms are used. Progress can be deemed slow until there is a breakthrough.
Testing	Typical tests include unit and integration tests.	You will also, at the very least, need data validation, trained model quality validation and model validation tests.
Deployment	The process typically starts with a build and then releases the software to staged environments (CI/CD)	In addition to the build and release, the deployment workflow must account for the continuous training of models from new data, based on sometimes complex conditions (such as data/concept drift)
Production	Solutions are monitored in production for bugs or degradation of service due to changing conditions such as an increase in traffic	Models have ties to the data in which it was created from, and as future data changes, predictions skew. In addition to the typical metrics to monitor, you must monitor data profiles and raise alerts if these drift to unacceptable levels.

How do you create Infrastructure in MLOps?

1. Cloud Platform Selection: Choose a cloud platform (e.g., AWS, Google Cloud, Microsoft Azure) that suits your requirements and offers robust ML services and infrastructure capabilities. Consider factors like cost, scalability, security, and availability of ML-specific services.
2. Compute Resources: Provision the necessary compute resources to train and deploy ML models. This can include virtual machines, containers, or serverless functions. Ensure that the resources are adequately sized to handle your workload and can scale as needed.
3. Data Storage: Set up a data storage system to store your training and inference data. Consider options like object storage (e.g., Amazon S3, Google Cloud Storage) or file storage (e.g., Amazon EFS, Azure Files). Design your data storage to be scalable, durable, and secure.
4. Data Versioning and Management: Implement a data versioning system to track and manage changes to your datasets. Use tools like Git LFS (Large File Storage) or dedicated data versioning platforms like DVC (Data Version Control) to maintain a history of your data and enable reproducibility.
5. Model Versioning and Management: Establish a model versioning system to track and manage different versions of your ML models. This can be done using a version control system like Git or dedicated ML model management tools like MLflow or Kubeflow. Ensure that you can easily track model changes, compare performance, and reproduce previous model versions.
6. Experiment Tracking: Implement a system to track and monitor experiments during the model development process. Tools like MLflow, Neptune, or TensorBoard can help you log metrics, visualize results, and compare different experiments to make informed decisions.
7. Automated Deployment: Set up automated deployment pipelines to streamline the process of deploying ML models into production. This can involve using infrastructure-as-code tools like Terraform or CloudFormation to define and manage your deployment infrastructure. Automate the deployment process to ensure consistency, repeatability, and reduce manual errors.
8. Monitoring and Logging: Implement monitoring and logging mechanisms to track the performance of your ML models in production. Use tools like Prometheus, Grafana, or cloud-native monitoring solutions to collect and analyze metrics, detect anomalies, and troubleshoot issues. Ensure that you have proper logging in place for debugging and auditing purposes.

9. Security and Access Control: Implement appropriate security measures to protect your ML infrastructure and data. Apply access controls and permissions to restrict access to sensitive resources. Encrypt data in transit and at rest. Regularly apply security patches and updates to your infrastructure components.
10. Scaling and Autoscaling: Design your infrastructure to handle varying workloads and scale as needed. Utilize features like autoscaling groups or Kubernetes clusters to automatically adjust resource capacity based on demand. Ensure your infrastructure can handle increased traffic during peak periods without sacrificing performance.
11. Continuous Integration/Continuous Deployment (CI/CD): Integrate your infrastructure with CI/CD pipelines to automate the building, testing, and deployment of ML models. This ensures consistent and reliable deployments across different environments.
12. Documentation and Collaboration: Document your infrastructure setup, configurations, and dependencies to ensure smooth collaboration among team members. Share best practices and guidelines for maintaining and updating the infrastructure. Foster a culture of knowledge sharing and documentation.

How can you create CI/CD pipelines for Machine Learning?

1. Version Control: Use a version control system (such as Git) to manage your ML project's code, data, and model files. Ensure that all relevant files are tracked and accessible.
2. Automated Testing: Create automated tests to evaluate the performance and accuracy of your ML models. This can include unit tests, integration tests, and validation against ground truth datasets. Test scripts can be written in frameworks like PyTest or TensorFlow's testing utilities.
3. Containerization: Package your ML models and dependencies into containers, such as Docker images. This enables consistent and reproducible deployments across different environments. Define a Dockerfile that specifies the necessary dependencies and configurations for running your ML model.
4. Build Automation: Set up a build system (e.g., Jenkins, GitLab CI, or Travis CI) to automate the build process. This involves defining build scripts or configuration files that specify how to build the ML model and create the required artifacts (e.g., Docker images).
5. Continuous Integration: Configure your CI system to trigger builds whenever changes are pushed to the version control repository. This ensures that your codebase is continuously integrated, and any issues are identified early in the development cycle.
6. Testing Automation: Integrate your automated tests into the CI pipeline. Run the tests after the build step to verify that the ML model behaves as expected and meets the defined criteria for quality and performance.
7. Continuous Deployment: Once the build and tests pass, deploy the ML model to the desired target environment (e.g., staging, production). This can involve deploying the Docker image to a container orchestration platform like Kubernetes or deploying the model to a cloud service like AWS SageMaker.
8. Monitoring and Rollbacks: Implement monitoring and logging mechanisms to track the performance of your ML model in production. Set up alerting systems to detect anomalies. If any issues arise, be prepared to roll back to a previous version or take appropriate corrective actions.
9. Iterative Improvements: Continuously refine and update your ML model, tests, and deployment pipeline based on user feedback, performance metrics, and new requirements. Use the feedback loop to drive iterative improvements.

AMAZON SAGEMAKER:
Amazon SageMaker is a fully managed service provided by Amazon Web Services (AWS) that is designed to simplify the process of building, training, and deploying machine learning models. It is an integrated development environment for machine learning, which makes it easier for data scientists and developers, especially those who are new to the field of machine learning, to build and deploy their models at scale.

Here's a breakdown of what Amazon SageMaker offers:

1. **Data Preparation**: SageMaker allows you to easily access and preprocess your data from various sources, such as Amazon S3, databases, or other AWS services. You can explore and visualize your data to gain insights and ensure it is suitable for training.

2. **Built-in Algorithms and Frameworks**: SageMaker provides a wide range of built-in algorithms and supports popular machine learning frameworks like TensorFlow and PyTorch. These algorithms and frameworks cover tasks such as regression, classification, clustering, and more.

3. **Custom Algorithms**: In addition to the built-in algorithms, SageMaker allows you to bring your custom algorithms using containerization, giving you the flexibility to use any language or framework of your choice.

4. **Model Training**: SageMaker automates the process of training your machine learning models on large datasets. It automatically scales the training environment to handle the data size, so you don't need to worry about infrastructure management.

5. **Hyperparameter Optimization**: SageMaker offers automatic hyperparameter tuning, which helps you find the best combination of hyperparameters for your models, leading to better performance.

6. **Model Deployment**: Once you've trained your model, SageMaker makes it easy to deploy it to a scalable and secure infrastructure. It can deploy models in real-time for inference or as batch jobs for batch processing.

7. **Monitoring and Management**: SageMaker provides tools to monitor the performance of your deployed models and helps you manage and update them as needed.

8. **Cost Optimization**: SageMaker offers pay-as-you-go pricing, which means you only pay for the resources you use during model training and inference. This can be cost-effective for small-scale projects and allows you to scale up when needed.

9. **Security and Compliance**: AWS takes care of the security and compliance aspects, ensuring that your data and models are protected with various security measures.

As someone new to machine learning and data science, SageMaker can be a valuable tool to accelerate your learning and development process. It abstracts away much of the infrastructure and complex setup involved in machine learning, allowing you to focus more on experimenting with different models, algorithms, and ideas.

To get started with SageMaker, you can follow the provided tutorials and examples in the AWS documentation. The service also offers Jupyter notebooks integrated into the platform, which is a popular environment for data exploration, data visualization, and running code related to machine learning.

Remember that machine learning is a vast field, and learning is an iterative process. Don't hesitate to experiment, try different approaches, and learn from the experiences as you work with SageMaker and dive deeper into the world of machine learning and data science.

Sure, here are short descriptions of each technology followed by their inter-connections and pros and cons:

1. KDEA (Kubernetes Dynamic External Admission): KDEA is a Kubernetes feature that allows cluster administrators to define custom admission controllers outside the Kubernetes cluster. It enables integration of external validation and mutation policies into the admission process of Kubernetes.

2. EKS (Amazon Elastic Kubernetes Service): EKS is a managed Kubernetes service provided by AWS. It simplifies the deployment, management, and scaling of containerized applications using Kubernetes.

3. Kops: Kops is a command-line tool for provisioning and managing Kubernetes clusters on AWS. It allows you to create, update, and delete Kubernetes clusters in a simple and declarative way.

4. SQS (Amazon Simple Queue Service): SQS is a fully managed message queuing service by AWS. It enables decoupling of microservices by providing reliable, scalable, and distributed message queues.

5. Redis Queue (RedisQ): RedisQ is a lightweight message queuing system built on top of Redis. It provides a simple way to manage work queues and distribute tasks to multiple workers.

6. ZeroMQ: ZeroMQ is an open-source messaging library that enables building distributed applications and microservices by providing socket-like messaging patterns.

7. RabbitMQ: RabbitMQ is a popular open-source message broker that implements the Advanced Message Queuing Protocol (AMQP). It facilitates communication between different services in a scalable and robust way.

8. AWS Route 53: AWS Route 53 is a scalable and highly available domain name system (DNS) web service. It provides domain registration, DNS routing, health checking, and traffic management for AWS resources.

9. Iceberg (AWS): Iceberg is an object storage format for large-scale analytics workloads on AWS. It offers better performance and optimizations for reading and querying data in data lakes.

10. Amazon ECS (Elastic Container Service): Amazon ECS is a container orchestration service provided by AWS. It allows you to run and scale containerized applications using Docker containers.

11. Redshift: Amazon Redshift is a fully managed data warehousing service by AWS. It enables data analysts and data engineers to run complex analytics queries on large datasets efficiently.

12. Snowflake Schema in Data Warehouse: Snowflake schema is a type of database schema used in data warehousing. It organizes data into multiple normalized dimensions, resulting in better query performance and reduced data redundancy.

13. Fivetran: Fivetran is a cloud-based data integration platform that automates data pipelines between various sources and destinations, making it easy to sync data into a data warehouse.

14. BigQuery: BigQuery is a fully managed, serverless data warehouse by Google Cloud Platform. It enables super-fast SQL-like queries on large datasets and supports real-time data analysis.

Inter-connections and Pros/Cons:

1. EKS and Kops: Both are related to Kubernetes cluster management on AWS. EKS is a managed service, while Kops is a tool for creating and managing Kubernetes clusters. EKS offers better ease of use but might have less flexibility compared to Kops.

2. SQS, Redis Queue, ZeroMQ, RabbitMQ: These are all messaging systems designed to decouple and manage communication between microservices. SQS is fully managed, while Redis Queue, ZeroMQ, and RabbitMQ offer more flexibility but require more operational overhead.

3. AWS Route 53 and Iceberg: These are both AWS services. AWS Route 53 manages DNS and traffic routing, while Iceberg is a storage format optimized for data lakes. They serve different purposes and don't have a direct inter-connection.

4. ECS AWS and Redshift: ECS is a container orchestration service, and Redshift is a data warehousing service on AWS. They can be used together to run containerized ETL jobs and data processing before loading data into Redshift.

5. Redshift and Snowflake Schema: Redshift is a data warehouse, and the Snowflake schema is a data modeling technique. Redshift can work with various schema designs, including the Snowflake schema, to optimize query performance.

6. Fivetran and BigQuery: Fivetran is a data integration platform, and BigQuery is a data warehouse. Fivetran can sync data from various sources into BigQuery, making it easier to analyze and query the data.

Pros and Cons:

1. EKS:
   Pros: Managed Kubernetes, Easy to scale, Integration with AWS services.
   Cons: Higher cost compared to self-hosted solutions, Limited control over underlying infrastructure.

2. SQS:
   Pros: Fully managed, Highly reliable and scalable, Supports message retention.
   Cons: Some limitations on message size and processing time, No support for message prioritization.

3. Redis Queue:
   Pros: Lightweight, High throughput, Supports various data structures, Real-time message processing.
   Cons: Not fully managed, Might require more operational effort, Data persistence and durability concerns.

4. AWS Route 53:
   Pros: Highly available and scalable, Global coverage, Integration with other AWS services.
   Cons: Cost can increase with traffic, Limited support for advanced routing policies.

5. Redshift:
   Pros: Fast query performance, Fully managed, Integration with BI tools, Columnar storage.
   Cons: Can be expensive for large datasets, Limited support for real-time data.

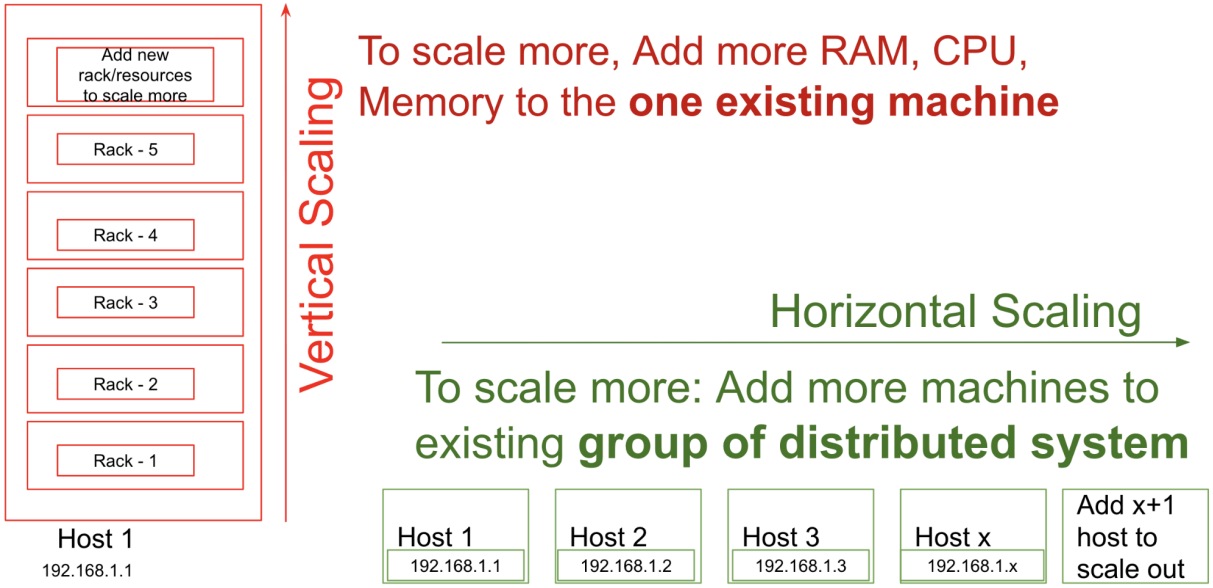
6. BigQuery:
   Pros: Serverless and fully managed, Super-fast query performance, Scalable and flexible pricing.
   Cons: Costs can escalate with data volume, Limited integration with non-Google services.

7. Fivetran:
   Pros: Easy setup and maintenance, Automated data integration, Support for various data sources.
   Cons: Cost might increase with data volume, Limited control over the integration process.

These technologies serve various purposes in modern data infrastructure and can be combined to create robust and efficient data pipelines and analytics solutions. The choice of tools will depend on specific requirements, budget constraints, and operational preferences.

Scaling

The primary difference between horizontal scaling and vertical scaling is that horizontal scaling involves adding more machines or nodes to a system, while vertical scaling involves adding more power (CPU, RAM, storage, etc.) to an existing machine.



What Is Scalability?

Scalability is the ability to expand the system's existing configuration to handle the increasing load. Scaling can be done by adding hardware or upgrading the current system configuration. There are two major ways to accomplish scaling – vertical scaling and horizontal scaling

https://www.redswitches.com/blog/horizontal-vs-vertical-scaling/