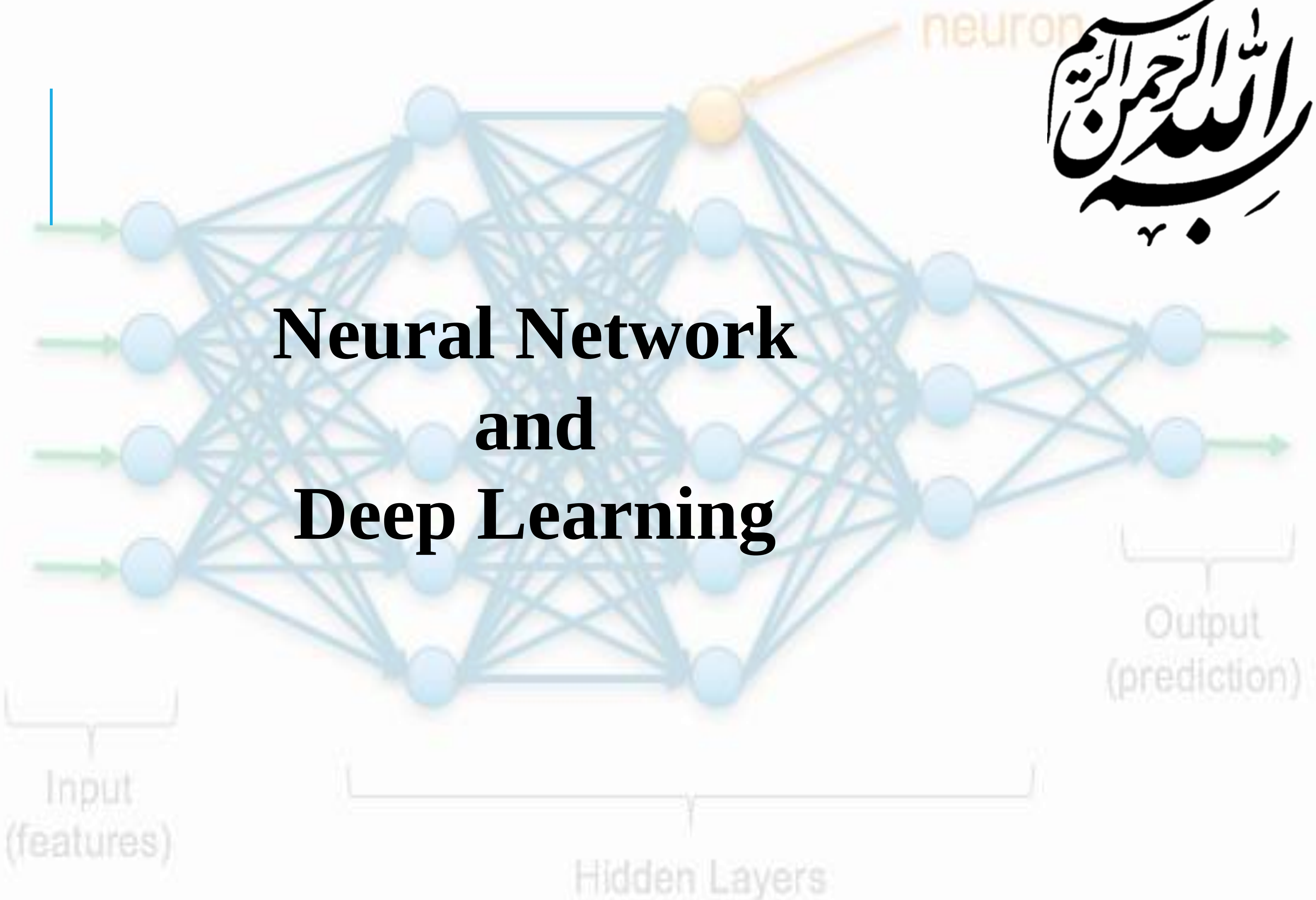
بسم الله الرحمن الرحیم



Neural Network and Deep Learning

Lecture3:

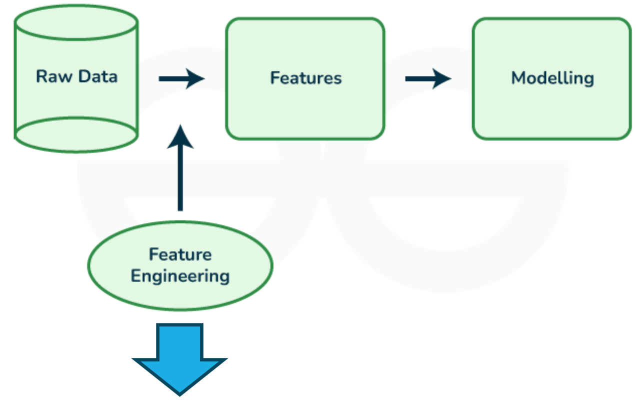
Artificial neural network life cycle

OUTLINE

- Artificial neural network life cycle
- Data
- Dataset
- Features
- Train the model
- Test the model

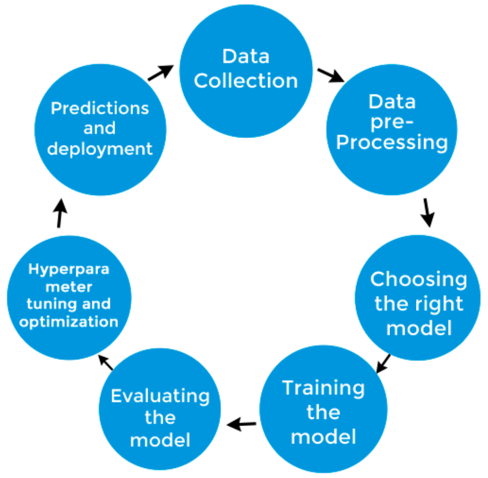
ARTIFICIAL NEURAL NETWORK LIFE CYCLE

- Gathering Data
- Data preparation
- Feature extraction
- Train the model
- Test the model



Feature Engineering in ML

- Feature selection
- Feature transformation
- Feature extraction



DATA

Data definition:

- Factual information (such as measurements or statistics) used as a basis for reasoning, discussion, or calculation.
- Information in digital form that can be transmitted or processed.
- In common usage, data (/ˈdeɪtə/, also US: /ˈdætə/) is a collection of discrete or continuous values that convey information, describing the quantity, quality, fact, statistics, other basic units of meaning, or simply sequences of symbols that may be further interpreted formally.

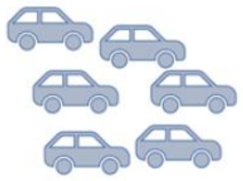
Different Types of data types

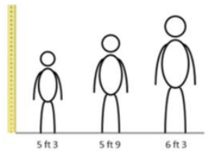
- The Data type is broadly classified into
  1. Quantitative
  2. Qualitative

DATA

1. **Quantitative data type:**

- This type of data type consists of numerical values. Anything which is measured by numbers.
- E.g., Profit, quantity sold, height, weight, temperature, etc.
- This is again of two types
- **A.) Discrete data type:**
  - ✓The numeric data which have discrete values or whole numbers. This type of variable value if expressed in decimal format will have no proper meaning. Their values can be counted.
  - ✓ E.g.: – No. of cars you have, no. of marbles in containers, students in a class, etc.
- **B.) Continuous data type:**
  - ✓The numerical measures which can take the value within a certain range. This type of variable value if expressed in decimal format has true meaning. Their values can not be counted but measured. The value can be infinite
  - ✓E.g.: – height, weight, time, area, distance, measurement of rainfall, etc.


No. of Cars



Height

Time

Fig: Continuous Data Types

DATA

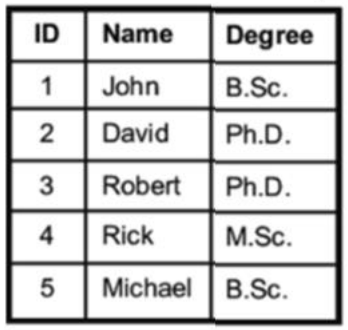
2. Qualitative data type:

- These are the data types that cannot be expressed in numbers. This describes categories or groups and is hence known as the categorical data type.
- This can be divided into:

a. Structured Data:

This type of data is either number or words. This can take numerical values but mathematical operations cannot be performed on it. This type of data is expressed in tabular format.

E.g.) Sunny=1, cloudy=2, windy=3 or binary form data like 0 or1, Good or bad, etc.



ID	Name	Degree
1	John	B.Sc.
2	David	Ph.D.
3	Robert	Ph.D.
4	Rick	M.Sc.
5	Michael	B.Sc.

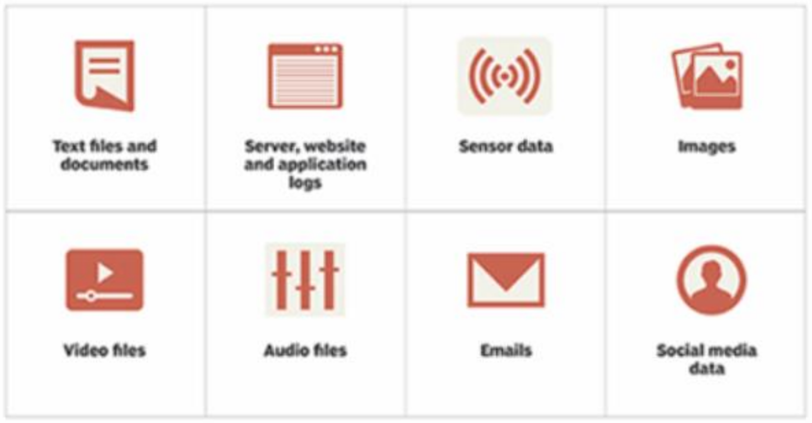
DATA

2. **Qualitative data type:**

- These are the data types that cannot be expressed in numbers. This describes categories or groups and is hence known as the categorical data type.
- This can be divided into:

   b. **Unstructured data:**

   This type of data does not have the proper format and therefore known as unstructured data. This comprises textual data, sounds, images, videos, etc.



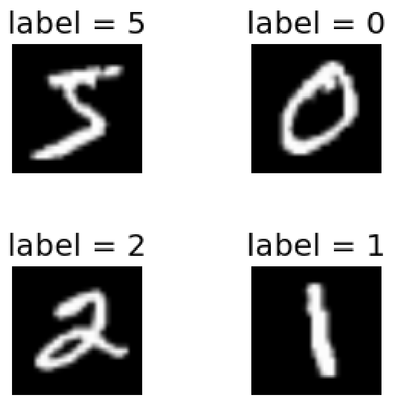
DATA

- **Dataset**: A dataset is a set of data that is employed to teach deep learning models. The scale, intricacy, or scope of these data sets may differ greatly depending on what exactly you're attempting to achieve in terms of deep learning tasks or problems at hand.

- **Examples of deep learning datasets**
  - MNIST
  - ImageNet
  - Free Spoken Digit Dataset
  - Yelp Open Dataset
  - Machine Translation of Various Languages
  - WordNet
  - …

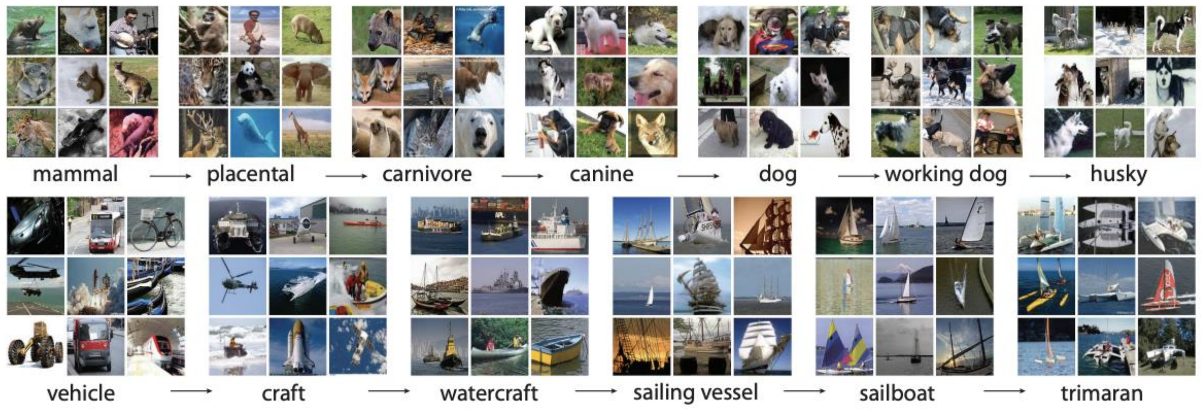
DATASET

- The **MNIST** dataset is a classic benchmark dataset in machine learning and deep learning.
  - It consists of 60,000 grayscale images of handwritten digits (0 through 9), with each image being a 28×28 pixel square.
  - MNIST is widely used for image classification tasks.
  - The dataset is relatively small and easily accessible, making it a popular choice for educational purposes and experimentation with neural network architectures.



DATASET

- The **ImageNet** dataset is one of the largest and most widely used datasets for training deep convolutional neural networks (CNNs).
  - It consists of over 14 million images across 22,000 of categories, covering a wide range of objects, animals, scenes, and more.
  - ImageNet is commonly used for image classification, object detection, and object localization tasks.
- **ImageNet Large-Scale Visual Recognition Challenge (ILSVRC):** 1000 images in each of 1000 categories 1.2 million training images

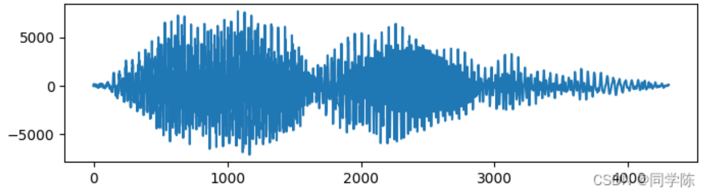

mammal → placental → carnivore → canine → dog → working dog → husky
vehicle → craft → watercraft → sailing vessel → sailboat → trimaran

DATASET

➢ Free Spoken Digit Dataset

Another entry in this list inspired by the MNIST dataset! This one was created to solve the task of identifying spoken digits in audio samples. It's an open dataset so the hope is that it will keep growing as people keep contributing more samples. Currently, it contains the following characteristics:

- 3 speakers
- 1,500 recordings (50 of each digit per speaker)
- English pronunciations



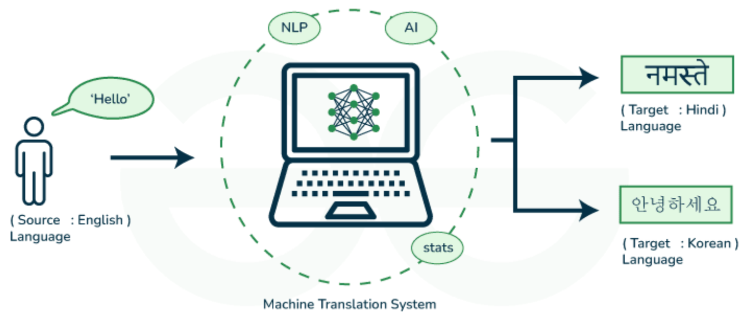
CSDN @同学陈

DATASET

- **Machine Translation of Various Languages**

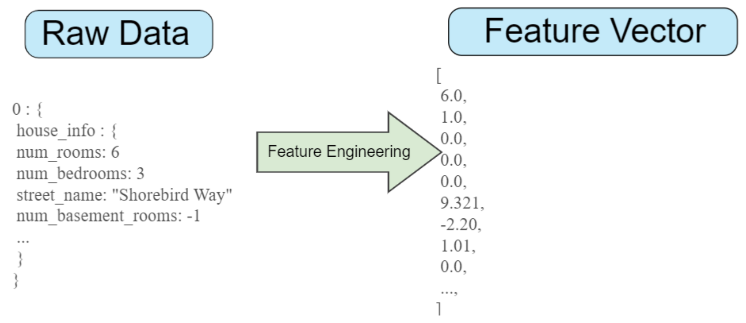
This dataset consists of training data for four European languages. The task here is to improve the current translation methods. You can participate in any of the following language pairs:

- English-Chinese and Chinese-English
- English-Estonian and Estonian-English
- English-German and German-English
- English-Russian and Russian-English
- English-Turkish and Turkish-English
- ⋮
- Size: ~15 GB
- Number of Records: ~30,000,000 sentences and their translations



FEATURES

Definition: In machine learning, a feature is data that's used as the input for ML models to make predictions. Raw data is rarely in a format that is consumable by an ML model, so it needs to be transformed into features. This process is called feature engineering.

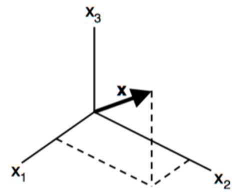


```
0 : {
 house_info : {
 num_rooms: 6
 num_bedrooms: 3
 street_name: "Shorebird Way"
 num_basement_rooms: -1
 ...
 }
}
```

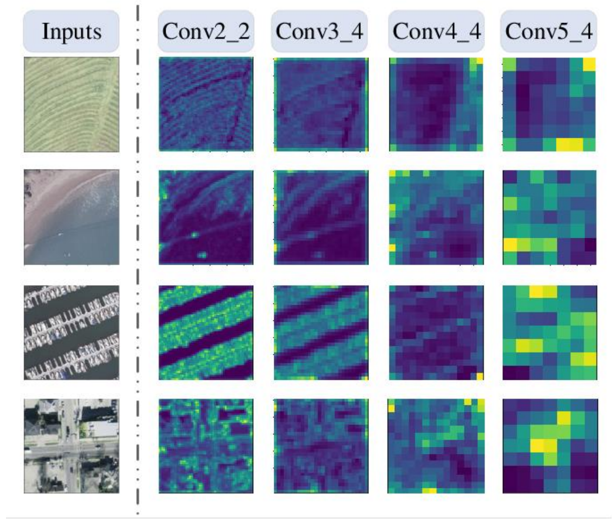
```
[
 6.0,
 1.0,
 0.0,
 0.0,
 0.0,
 9.321,
 -2.20,
 1.01,
 0.0,
 ...,
]
```

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \\ x_d \end{bmatrix}$$

Feature vector



Feature space (3D)



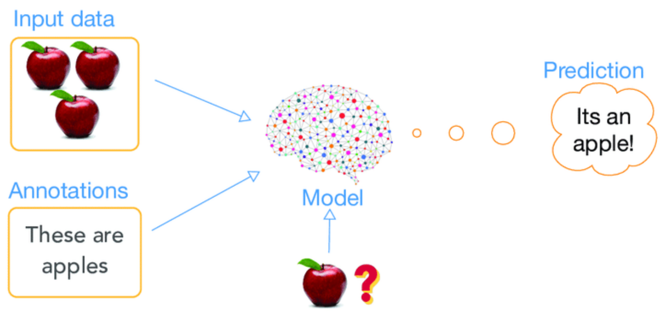
Feature map

TRAIN THE MODEL

- **Types of Machine Learning**
  - There are several types of machine learning, each with special characteristics and applications. Some of the main types of machine learning algorithms are as follows:
    1. Supervised Machine Learning
    2. Unsupervised Machine Learning
    3. Semi-Supervised Machine Learning
    4. Reinforcement Learning

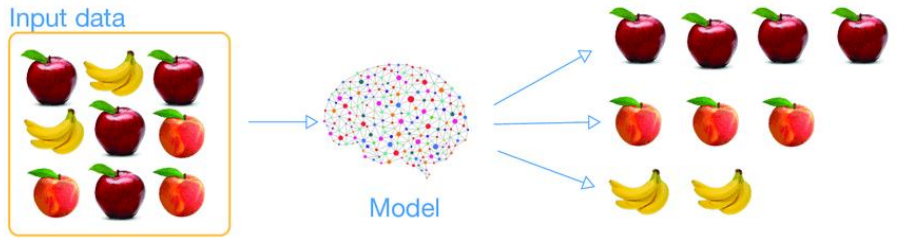
TRAIN THE MODEL

1. **Supervised Machine Learning**
   - Supervised learning is defined as when a model gets trained on a "Labelled Dataset".
   - Labelled datasets have both input and output parameters.
   - In Supervised Learning algorithms learn to map points between inputs and correct outputs.



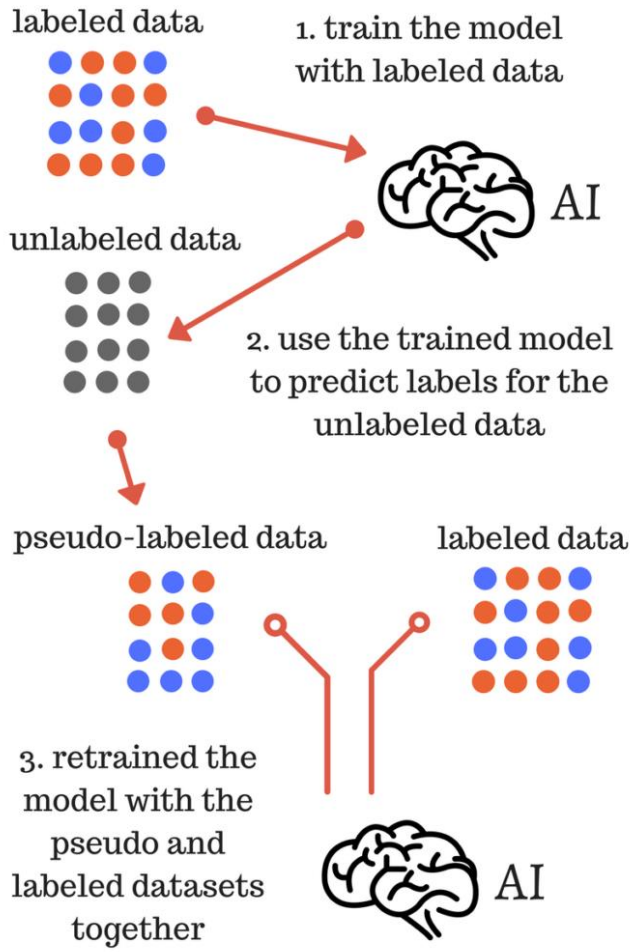
TRAIN THE MODEL

2. **Unsupervised Machine Learning**
   - Unsupervised learning is a type of machine learning technique in which an algorithm discovers patterns and relationships using unlabeled data.
   - Unlike supervised learning, unsupervised learning doesn't involve providing the algorithm with labeled target outputs.
   - The primary goal of Unsupervised learning is often to discover hidden patterns, similarities, or clusters within the data, which can then be used for various purposes, such as data exploration, visualization, dimensionality reduction, and more.



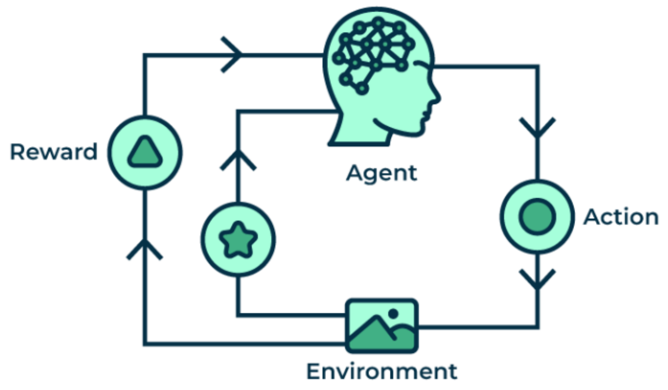
TRAIN THE MODEL

3. **Semi-Supervised Learning**
   - Semi-Supervised learning is a machine learning algorithm that works between the supervised and unsupervised learning so it uses both labelled and unlabelled data.
   - It's particularly useful when obtaining labeled data is costly, time-consuming, or resource-intensive.
   - We use these techniques when we are dealing with data that is a little bit labeled and the rest large portion of it is unlabeled.
   - We can use the unsupervised techniques to predict labels and then feed these labels to supervised techniques.
   - This technique is mostly applicable in the case of image data sets where usually all images are not labeled.


labeled data
1. train the model with labeled data
AI
unlabeled data
2. use the trained model to predict labels for the unlabeled data
pseudo-labeled data
labeled data
3. retrained the model with the pseudo and labeled datasets together
AI

TRAIN THE MODEL

4. **Reinforcement Machine Learning**
   - Reinforcement machine learning algorithm is a learning method that interacts with the environment by producing actions and discovering errors.
   - Trial, error, and delay are the most relevant characteristics of reinforcement learning.
   - In this technique, the model keeps on increasing its performance using Reward Feedback to learn the behavior or pattern.



TRAIN THE MODEL

- **Learning with a Teacher (Supervised learning)**
  - Indeed, the desired response represents the "optimum" action to be performed by the neural network. The network parameters are adjusted under the combined influence of the training vector and the error signal. The error signal is defined as the difference between the desired response and the actual response of the network.

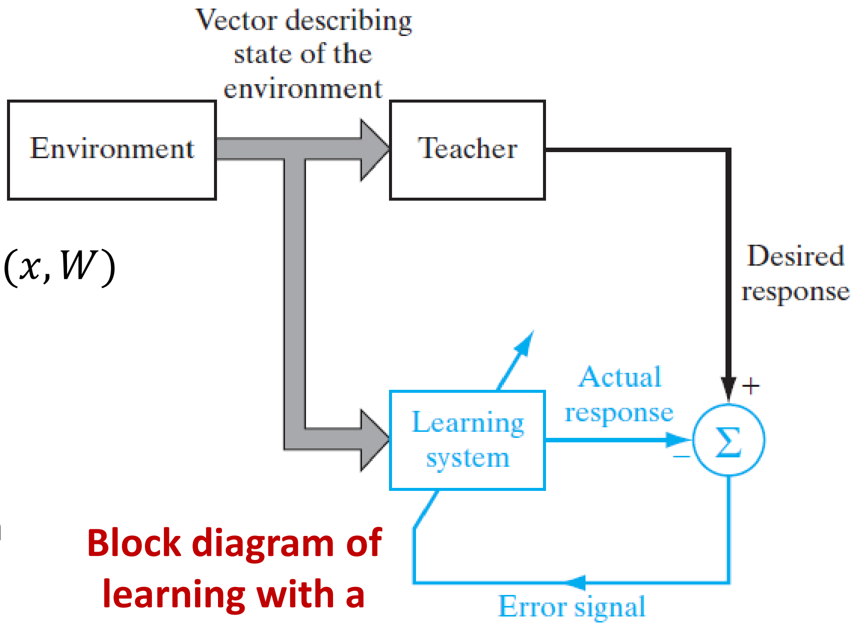
  - **Calculate the error signal:**

  Loss function or cost function, $L(y, \hat{y})$

$$y: label\ of\ data, \hat{y}: predicted\ by\ network, \hat{y} = f(x, W)$$

$$L(W) = \frac{1}{2}\sum_{i=1}^{N}(y_i - f(x_i, W))^2$$

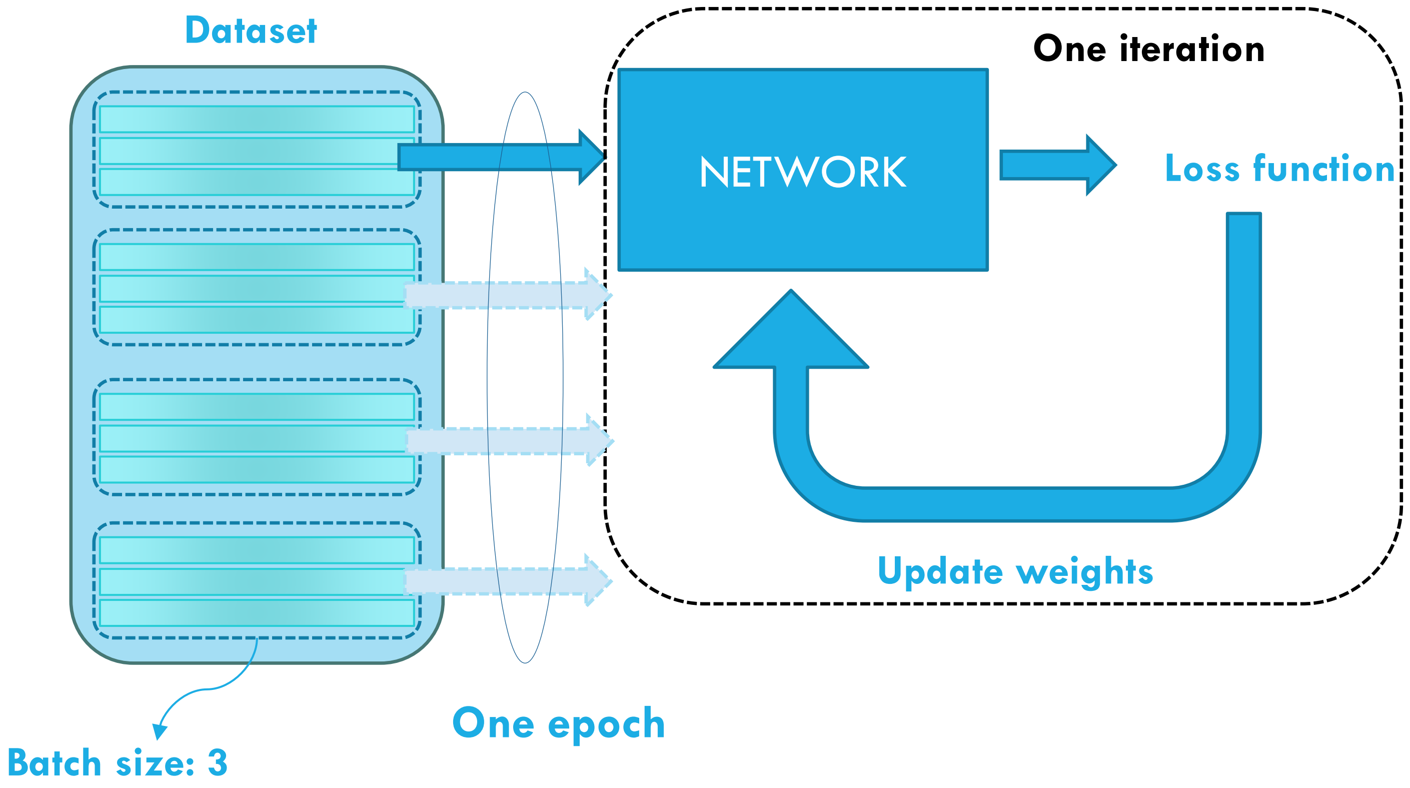
Minimizing this cost function with respect to w yields a formula for the ordinary least-squares estimator.



Block diagram of learning with a teacher

TRAIN THE MODEL

- **Mini-batch Learning**



Data randomly shuffled for the next epoch.

TRAIN THE MODEL

- **Main elements in learning stage:**
  - Loss functions
  - Optimizers
  - Model fitting
  - Regularization techniques
  - Batch size

TRAIN THE MODEL

➢ **Loss functions or cost function or objective function**

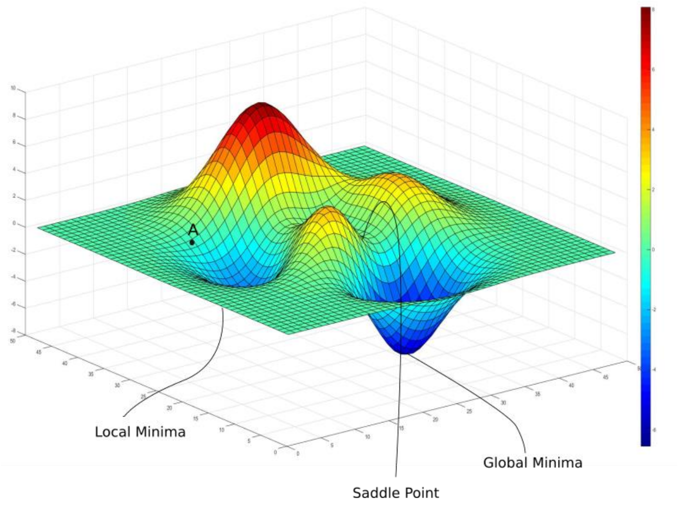
The purpose of loss functions is to compute the quantity that a model should seek to minimize during training.

- **Available losses**

  Probabilistic losses
  - binary_crossentropy
  - categorical_crossentropy
  - sparse_categorical_crossentropy

  Regression losses
  - mean_squared_error
  - mean_absolute_error
  - mean_absolute_percentage_error
  - mean_squared_logarithmic_error
  - cosine_similarity
  - Huber

  …



A visualization of the loss function of a neural network.

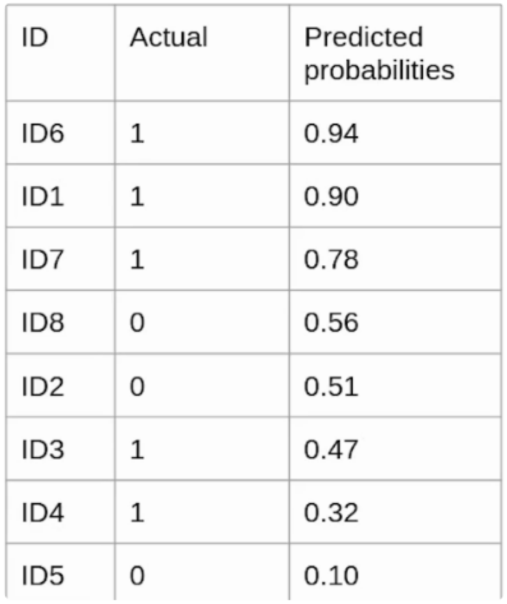
TRAIN THE MODEL

- **Loss functions**
  - **Binary Cross-Entropy Loss:**
    - Binary Cross Entropy is a loss function used in machine learning and deep learning to measure the difference between predicted binary outcomes and actual binary labels.
    - In binary classification, the activation function used is the sigmoid activation function. It constrains the output to a number between 0 and 1.

      ID: It represents a unique instance.

      Actual: It is the class the object originally belongs to.

      Predicted_probabilities: This is output given by the model that tells, the probability object belongs to class 1.



ID	Actual	Predicted probabilities
ID6	1	0.94
ID1	1	0.90
ID7	1	0.78
ID8	0	0.56
ID2	0	0.51
ID3	1	0.47
ID4	1	0.32
ID5	0	0.10

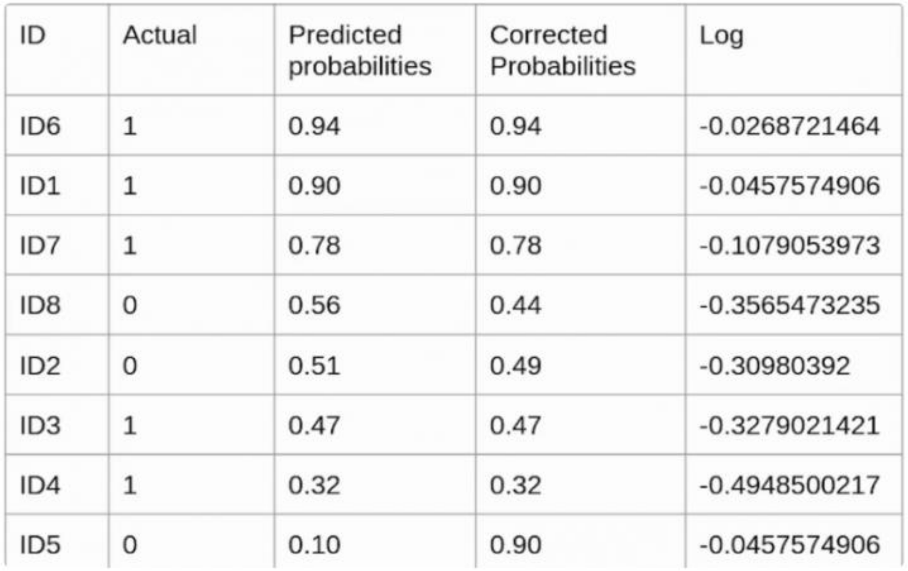
TRAIN THE MODEL

➢**Loss functions**

- **Binary Cross-Entropy Loss:**

Corrected Probabilities: It is the probability that a particular observation belongs to its original class. As shown in the above image, ID6 originally belongs to class 1 hence its predicted probability and corrected probability is the same i.e 0.94.

On the other hand, the observation ID8 is from class 0. In this case, the predicted probability i.e the chances that ID8 belongs to class 1 is 0.56 whereas, the corrected probability means the chances that ID8 belongs to class 0 is (1-predicted_probability) is 0.44. In the same way, corrected probabilities of all the instances will be calculated.



ID	Actual	Predicted probabilities	Corrected Probabilities	Log
ID6	1	0.94	0.94	-0.0268721464
ID1	1	0.90	0.90	-0.0457574906
ID7	1	0.78	0.78	-0.1079053973
ID8	0	0.56	0.44	-0.3565473235
ID2	0	0.51	0.49	-0.30980392
ID3	1	0.47	0.47	-0.3279021421
ID4	1	0.32	0.32	-0.4948500217
ID5	0	0.10	0.90	-0.0457574906

$$-\frac{1}{N}\sum_{i=1}^{N}\log(\mathrm{p_i})$$

The value of the negative average of corrected probabilities we calculate comes to be **0.214** which is our Log loss or **Binary cross-entropy** for this particular example.

TRAIN THE MODEL

- **Loss functions**
  - **Binary Cross-Entropy Loss:**

Further, instead of calculating corrected probabilities, we can calculate the Log loss using the formula given below.

$$\log loss = \frac{1}{N}\sum_{i=1}^{N} -(y_i * \log(p_i) + (1 - y_i) * \log(1 - p_i))$$

Here, pi is the probability of class 1, and (1-pi) is the probability of class 0.

When the observation belongs to class 1 the first part of the formula becomes active and the second part vanishes and vice versa in the case observation’s actual class are 0. This is how we calculate the Binary cross-entropy.

Binary cross-entropy measures the dissimilarity between predicted probabilities for binary classification. It is well-suited for models with sigmoid activation in the output layer.

TRAIN THE MODEL

- **Loss functions**
  - **Multiclass classification**
    - Problems involving the prediction of more than one class use different loss functions. In this section we'll look at a couple:
  - **Categorical Crossentropy**
    - The CategoricalCrossentropy also computes the cross-entropy loss between the true classes and predicted classes. The labels are given in an one_hot format.
    - **Description**: Categorical cross-entropy quantifies the dissimilarity between predicted class probabilities and true class distributions. It is ideal for models handling multiclass classification

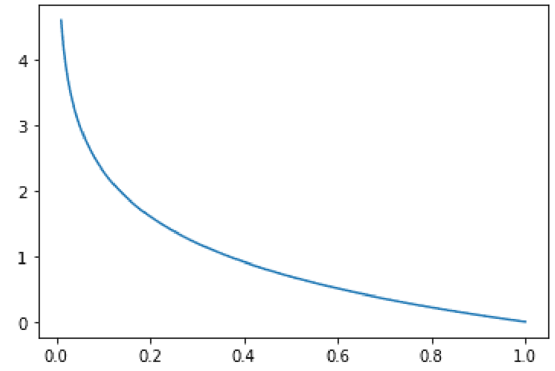
TRAIN THE MODEL

- **Loss functions**
  - **Categorical Cross-Entropy**

$$y_{true} = y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}, \; p_{pred} = \hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \end{bmatrix}, \; CE = -\sum_{i=1}^{i=N} y_i \,.\log(\hat{y}_i)$$

$$CE = -[y_1.\log(\hat{y}_1) + y_2.\log(\hat{y}_2) + y_3.\log(\hat{y}_3)]$$

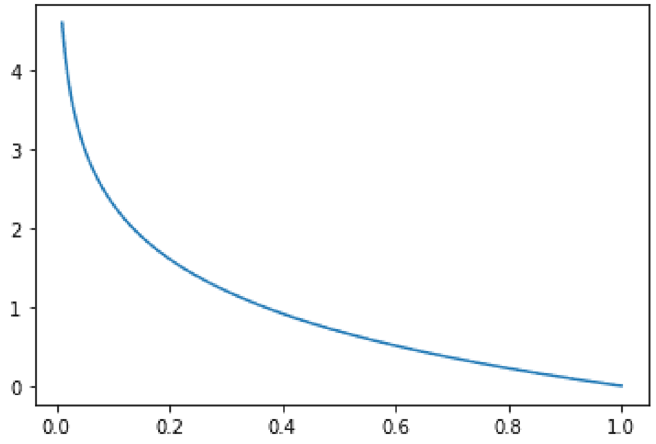
Cross-entropy metrics have a negative sign because log(x) tends to negative infinity as x tends to zero. We want a higher loss when the probability approaches 0 and a lower loss when the probability approaches 1.



Categorical cross entropy loss function, where x is the predicted probability of the ground truth class

TRAIN THE MODEL

- **Loss functions**
  - **Categorical Cross-Entropy**



Categorical cross entropy loss function, where x is the predicted probability of the ground truth class

Notice that the loss is exactly 0 if the probability of the ground truth class is 1 as desired. Also, as the probability of the ground truth class tends to 0, the loss tends to positive infinity as well, hence substantially penalizing bad predictions.

TRAIN THE MODEL

- **Loss functions**
  - **Sparse Categorical Cross-Entropy Loss:**

    The other way of implementing the categorical cross entropy loss in some frameworks is using a label-encoded representation for the class, where the class is represented by a single **non-negative integer indicating the ground truth class** instead.

  - **Categorical Cross-Entropy Loss vs. Sparse Categorical Cross-Entropy Loss:**
    - Both, categorical cross entropy and sparse categorical cross entropy have the same loss function. The only difference is the format in which you mention Yi (i,e true labels).
    - If your Yi's are one-hot encoded, use categorical_crossentropy. Examples (for a 3-class classification): [1,0,0] , [0,1,0], [0,0,1]
    - But if your Yi's are integers, use sparse_categorical_crossentropy. Examples for above 3-class classification problem: [1] , [2], [3]
    - The usage entirely depends on how you load your dataset. One advantage of using sparse categorical cross entropy is it saves time in memory as well as computation because it simply uses a single integer for a class, rather than a whole vector.

TRAIN THE MODEL

- **Loss functions**
  - **Regression**

In regression problems, you have to calculate the differences between the predicted values and the true values but as always there are many ways to do it.

1. **Mean Squared Error (MSE) Loss:** The MeanSquaredError class can be used to compute the mean square of errors between the predictions and the true values.

- Equation: $\frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$
- Use Case: Regression problems.
- Description: MSE measures the average squared difference between predicted values and true target values. It penalizes large errors heavily and is sensitive to outliers.

TRAIN THE MODEL

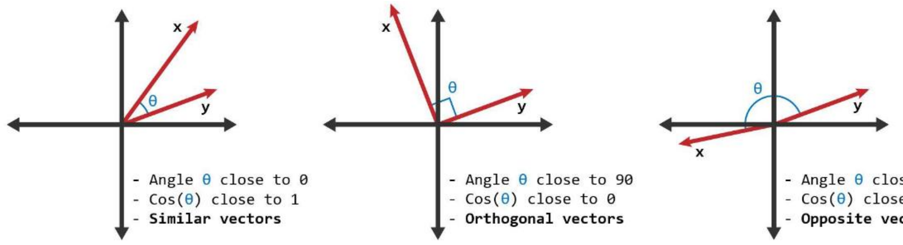
- **Loss functions**

2. **Mean Absolute Error (MAE) Loss:**
   - Equation: $MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$
   - Use Case: Regression problems.
   - Description: MAE calculates the mean absolute difference between predicted values and true target values. It is less sensitive to outliers compared to MSE

- **Mean Squared Error (MSE) vs. Mean Absolute Error (MAE) Loss:**
  - This gives the output 5.0 as expected since $\frac{1}{2}[(2-1)^2 + (3-0)^2] = \frac{1}{2}(10) = 5.$
  - . Notice that the second example with a predicted value of 3 and actual value of 0 contributes 90% of the error under the mean squared error vs. 75% under the mean absolute error.

TRAIN THE MODEL

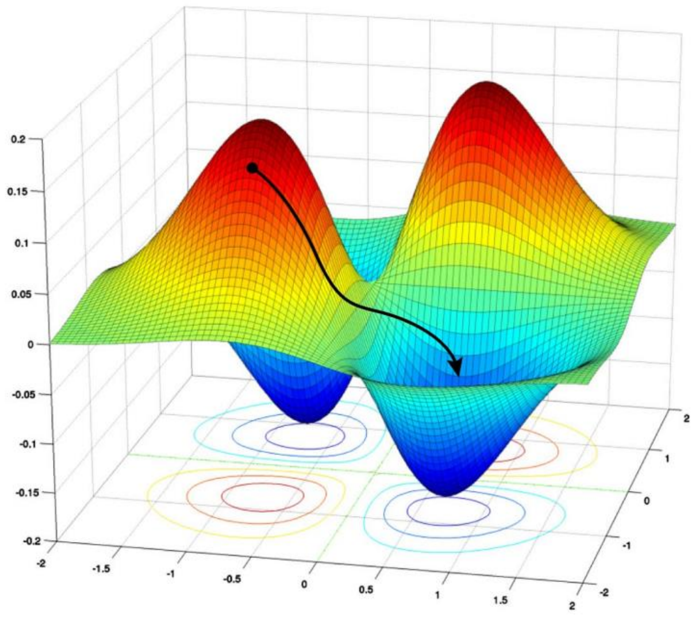
- **Loss functions**
  - **Cosine Similarity Loss**
    - Cosine similarity is a metric that determines how two vectors (words, sentences, features) are similar to each other. Basically, it is an angle between two vectors.
    - If your interest is in computing the cosine similarity between the true and predicted values, you'd use the CosineSimilarity class.
    - The cosine of two non-zero vectors can be derived by using the Euclidean dot product formula: $\mathbf{A.B} = \|\boldsymbol{A}\|\|\boldsymbol{B}\|\boldsymbol{cos\theta}$
    - The result is a number between -1 and 1 . 0 indicates orthogonality while values close to 1 show that there is great similarity.
    - $\cos(\theta) = \frac{\mathbf{A.B}}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}$
    - Where $A_i$and B_i are the ith components of vectors **A** and **B**, respectively.



TRAIN THE MODEL

➢ **Optimizers**

- we can use an optimization algorithm in attempt to minimize the loss. In optimization, a **loss function** is often referred to as the **objective function** of the optimization problem.



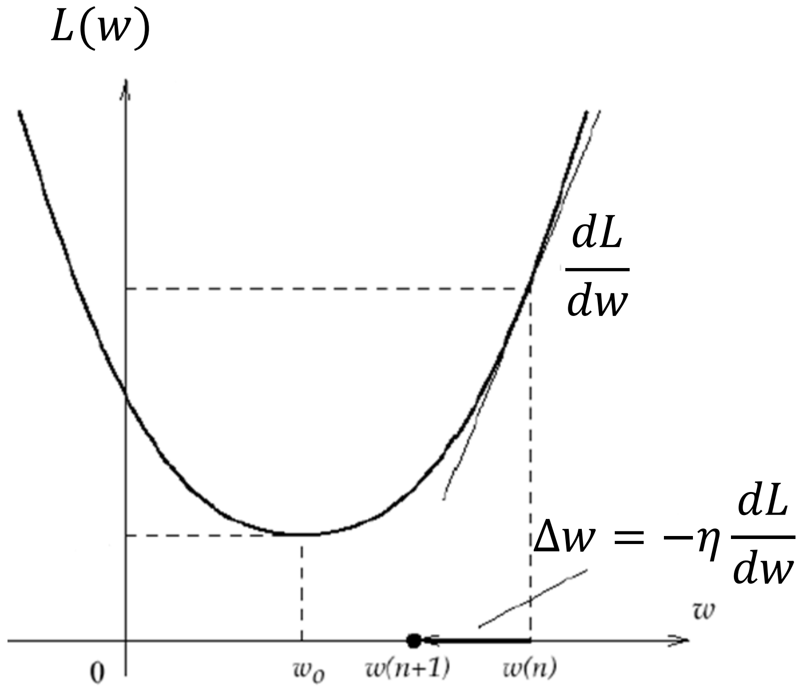
TRAIN THE MODEL

- **Optimizers**
  - we can use an optimization algorithm in attempt to minimize the loss. In optimization, a loss function is often referred to as the *objective function* of the optimization problem.
  - **Gradient Descent:**

$$\boldsymbol{g} = \nabla L(\boldsymbol{w}), \qquad \boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \eta \boldsymbol{g}(n)$$

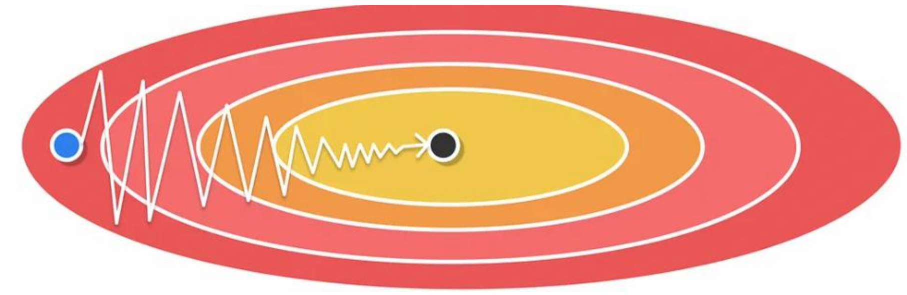
where η is a positive constant called the stepsize, or learning-rate, parameter, and g(n) is the gradient vector evaluated at the point w(n). In going from iteration n to n+1, the algorithm applies the correction:

$$\Delta \boldsymbol{w}(n) = \boldsymbol{w}(n+1) - \boldsymbol{w}(n) = -\eta \boldsymbol{g}(n)$$



TRAIN THE MODEL

- **Optimizers**
  - we can use an optimization algorithm in attempt to minimize the loss. In optimization, a loss function is often referred to as the objective function of the optimization problem.
  - **Different types of optimizers**
    - **Stochastic Gradient Descent (SGD)**
      - Equation: $w_{t+1} = w_t - \alpha . \frac{\partial L}{\partial w_t}$
      - Very popular optimization strategy
      - Sometimes slow



TRAIN THE MODEL

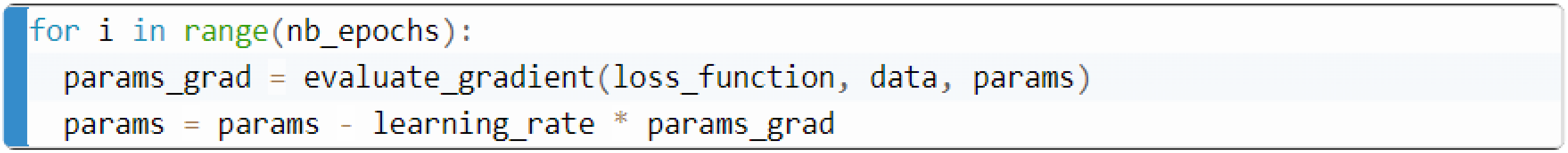
- **Gradient descent** is the preferred way to optimize neural networks and many other machine learning algorithms.
- **Gradient descent variants**
  - Batch gradient descent
  - Stochastic gradient descent
  - Mini-batch gradient descent

TRAIN THE MODEL

- **Batch gradient descent**

Vanilla gradient descent, aka batch gradient descent, computes the gradient of the cost function w.r.t. to the parameters θ for the entire training dataset:

- **Disadvantages:**
  - For the whole dataset performs just one update
  - Very slow
  - It doesn't allow us to update our model online



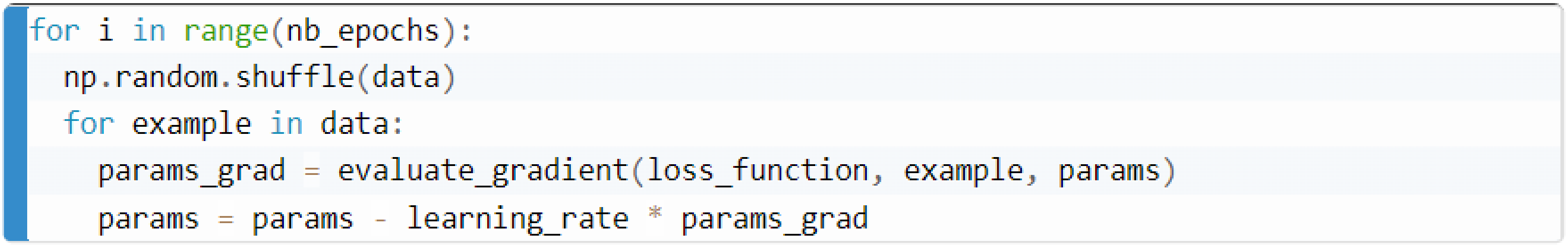
```
for i in range(nb_epochs):
  params_grad = evaluate_gradient(loss_function, data, params)
  params = params - learning_rate * params_grad
```

TRAIN THE MODEL

- **Stochastic gradient descent**

Stochastic gradient descent (SGD) in contrast performs a parameter update for each training example.

- **Advantages**:
  - Much faster
  - Online learning
- **Disadvantage**:
  - Jumping to local minima

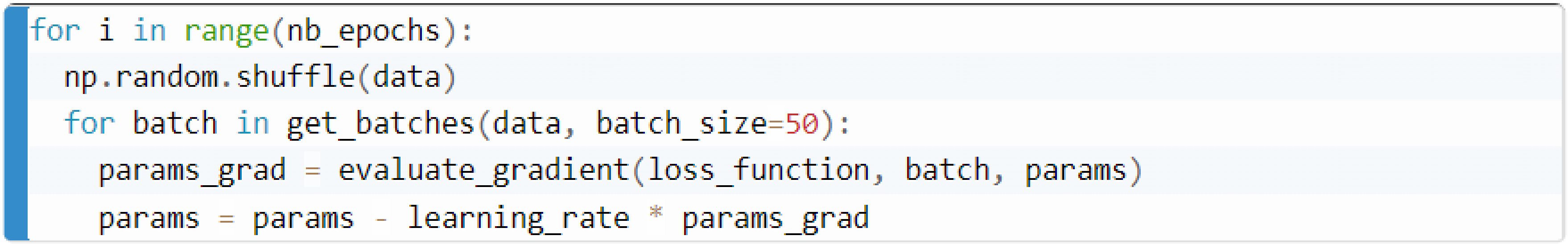


```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for example in data:
    params_grad = evaluate_gradient(loss_function, example, params)
    params = params - learning_rate * params_grad
```

TRAIN THE MODEL

➢ **Mini-batch gradient descent**

Mini-batch gradient descent finally takes the best of both worlds and performs an update for every mini-batch of n training examples:

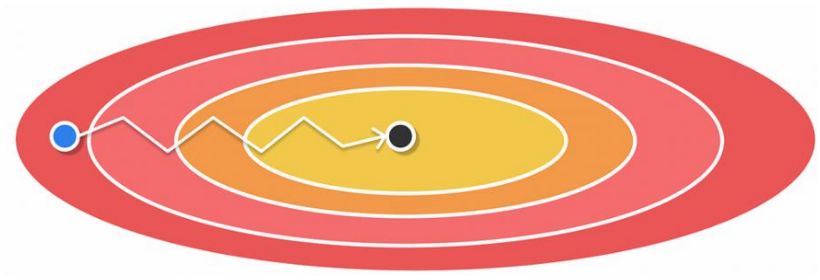


```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for batch in get_batches(data, batch_size=50):
    params_grad = evaluate_gradient(loss_function, batch, params)
    params = params - learning_rate * params_grad
```

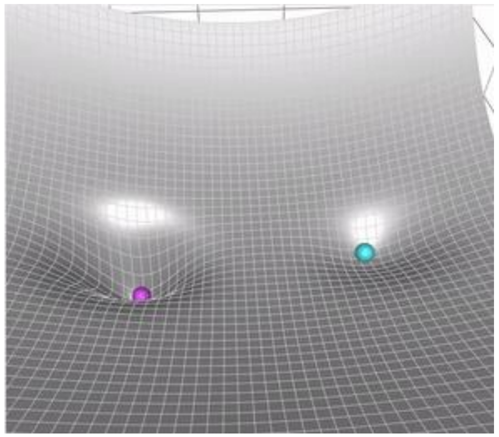
TRAIN THE MODEL

- **Optimizers**
  - **Different types of optimizers**
    - **SGD + Momentum**
      - Equation: $v_{t+1} = \rho v_t + \frac{\partial L}{\partial w_t}, \qquad w_{t+1} = w_t - \alpha v_{t+1}$
      - The method of momentum is designed to accelerate learning

You may see SGD+Momentum formulated different ways, but they are equivalent give same sequence of w



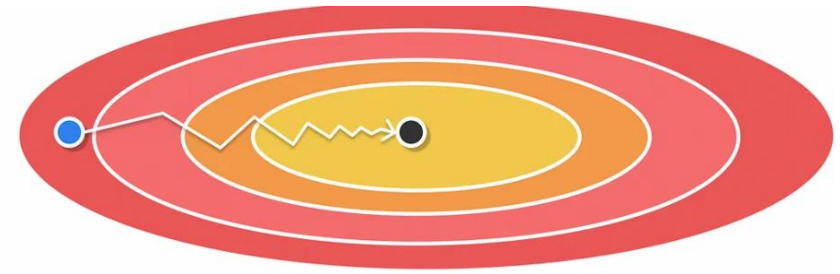
-Rho gives "friction"; typically rho=0.9 or 0.99



SGD
SGD+Momentum

TRAIN THE MODEL

- **Optimizers**
  - **Different types of optimizers**
    - **AdaGrad (Adaptive Gradient Algorithm)**
      - Equation: $v_{t+1} = v_t + (\frac{\partial L}{\partial w_t})^2, \; w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_{t+1}} + \varepsilon} \times \frac{\partial L}{\partial w_t}$
      - The greatest **advantage** of AdaGrad is that there is no longer a need to manually adjust the learning rate as it adapts itself during training.
      - Nevertheless, there is a **negative side of AdaGrad**: the learning rate constantly decays with the increase of iterations (the learning rate is always divided by a positive cumulative number). Therefore, the algorithm tends to converge slowly during the last iterations where it becomes very low.



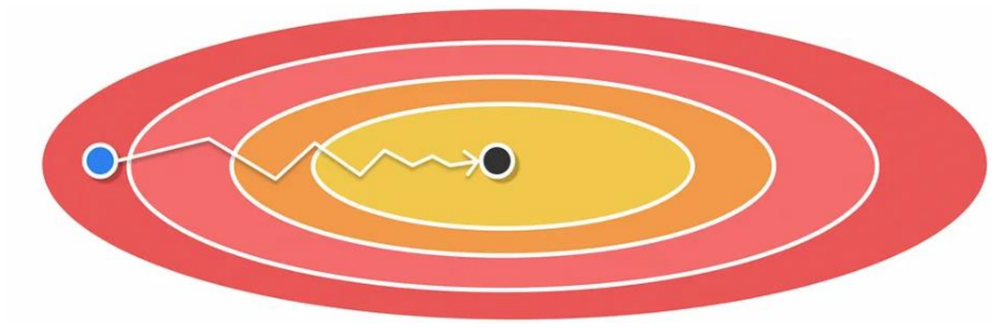
TRAIN THE MODEL

- **Optimizers**
  - **Different types of optimizers**
    - **RMSProp (Root Mean Square Propagation)**
      - Equation: $v_{t+1} = \rho v_t + (1-\rho) \times \left(\frac{\partial L}{\partial w_t}\right)^2, \quad w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_{t+1}} + \varepsilon} \times \frac{\partial L}{\partial w_t}$
      - RMSProp was elaborated as an improvement over AdaGrad which tackles the issue of learning rate decay.
      - Similarly to AdaGrad, RMSProp uses a pair of equations for which the weight update is absolutely the same.
      - RMSProp generally converges faster than AdaGrad
      - Compared to AdaGrad, the learning rate in RMSProp does not always decay with the increase of iterations making it possible to adapt better in particular situations.
      - “adaptive learning rates”
      - Progress along “steep” directions is damped.
      - progress along “flat” directions is accelerated.

TRAIN THE MODEL

- **Optimizers**
  - **Different types of optimizers**
    - RMSProp (Root Mean Square Propagation)

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



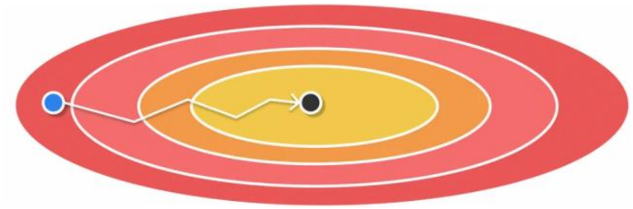
TRAIN THE MODEL

- **Optimizers**
  - **Different types of optimizers**
    - **Adam (Adaptive Moment Estimation)**
      - Equation:

$$v_{t+1} = \beta_1 v_t + (1-\beta_1)\frac{\partial L}{\partial w_t} \longrightarrow \hat{v}_{t+1} = \frac{v_{t+1}}{1-\beta_1{}^t}$$

$$s_{t+1} = \beta_2 s_t + (1-\beta_2)\frac{\partial L}{\partial w_t}^2 \longrightarrow \hat{s}_{t+1} = \frac{s_{t+1}}{1-\beta_2{}^t}$$

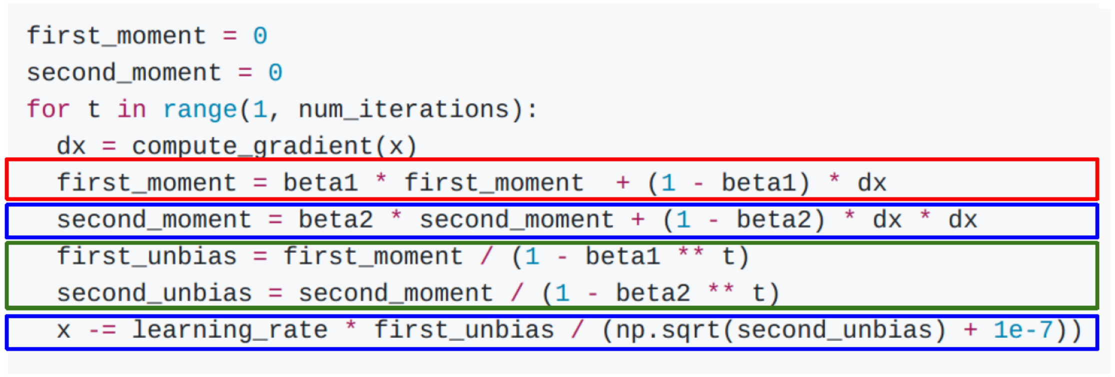
$$w_{t+1} = w_t - \frac{\alpha \hat{v}_{t+1}}{\sqrt{\hat{s}_{t+1}} + \varepsilon}$$



      - According to the Adam paper, good default values for hyperparameters are $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 1e\text{-}8$, and learning_rate = 1e-3 or 5e-4 is a great starting point for many models!
      - For the moment, Adam is the most famous optimization algorithm in deep learning. At a high level, Adam combines Momentum and RMSProp algorithms.

TRAIN THE MODEL

- **Optimizers**
  - **Different types of optimizers**
    - **Adam (Adaptive Moment Estimation)**
      - Bias correction for the fact that first and second moment estimates start at zero
      - Adam with beta1 = 0.9, beta2 = 0.999, and learning_rate = 1e-3 or 5e-4 is a great starting point for many models!



```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

TRAIN THE MODEL

- **Optimizers**
  - Local minima and Saddle points

- **Why Saddle Points are a Challenge:**

1. **Gradient Misleading:**

   The zero gradient at a saddle point can mislead optimization algorithms. While the algorithm thinks it has found a critical point, it may not be moving toward the actual minimum.

2. **Slow Convergence:**

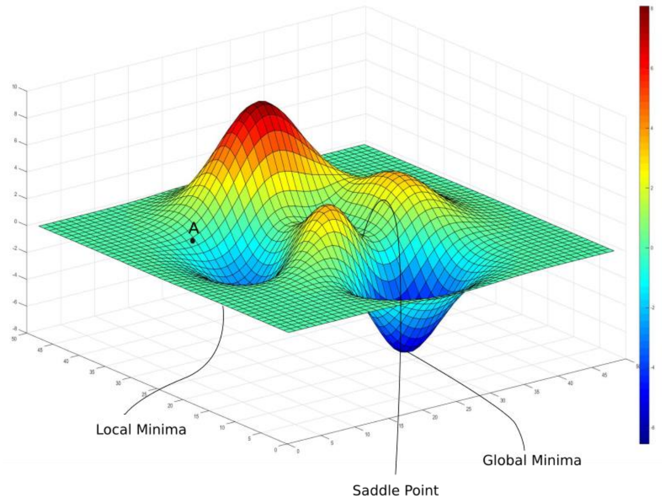
   Optimization algorithms can converge very slowly around saddle points, as the small gradient values make it difficult for the algorithm to escape the region.

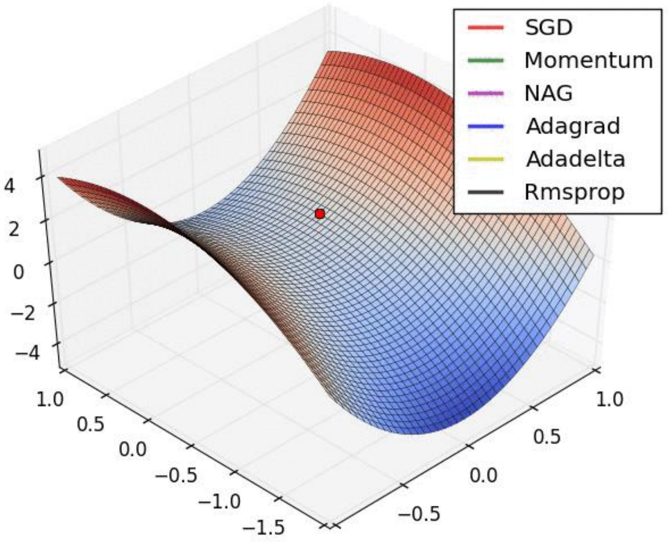
3. **Dimensionality Matters:**

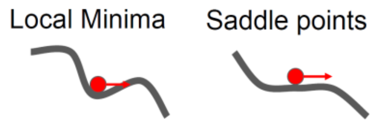
   The likelihood of encountering saddle points increases with the dimensionality of the parameter space. In high-dimensional spaces, the probability of finding points with zero gradient is higher.

4. **Saddle Points vs. Minima:**

   Distinguishing between saddle points and minima is challenging, and optimization algorithms may mistake a saddle point for a minimum.



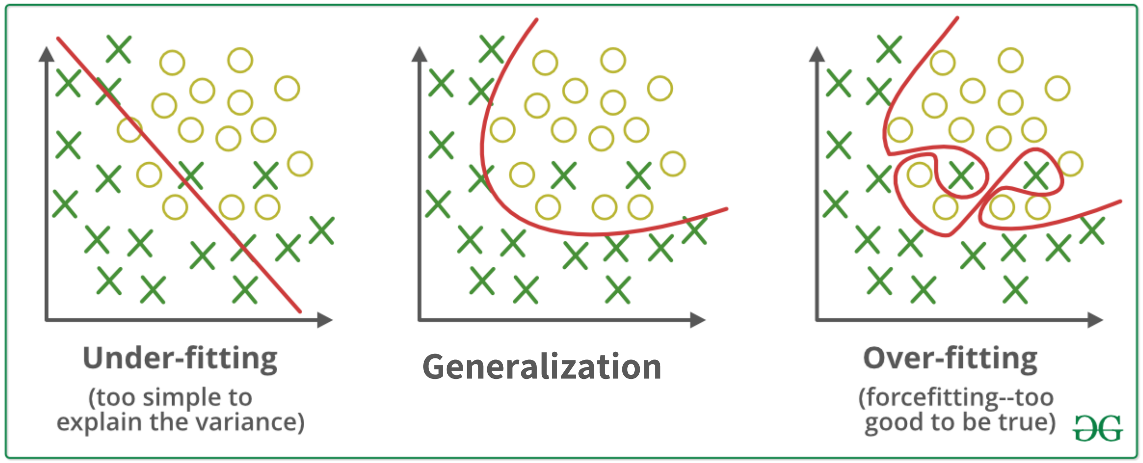




TRAIN THE MODEL

- **Model Fit: Underfitting, overfitting, and generalization**

  A good classifier will learn a decision boundary (the red line in the illustration below) that correctly classifies most of the training data and generalizes to novel data.


Under-fitting
(too simple to explain the variance)
Generalization
Over-fitting
(forcefitting--too good to be true)
ƏG

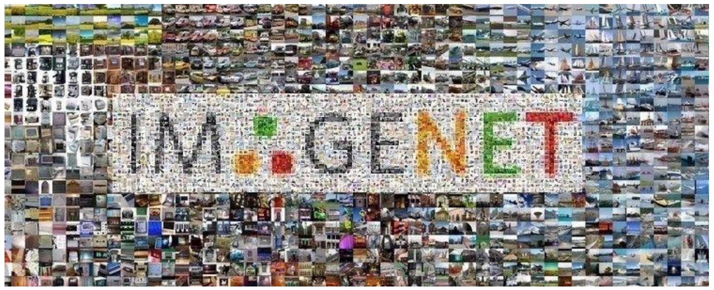
TRAIN THE MODEL

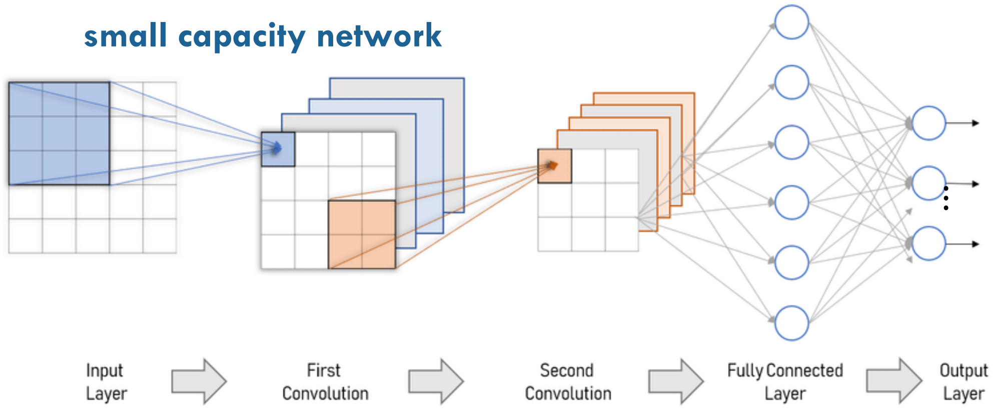
- **Generalization**
  - A neural network’s ability to generalize to novel data depends on two factors:
    1. **The network’s capacity**: By capacity, we mean the scope of functions it can learn. Usually, the <u>more parameters</u> a network uses, the <u>more functions it can learn</u>.
    2. **The complexity of information in the training data**: For example, images taken from the front-facing camera of a car, filled with cars, trucks, bicycles, scooters, pedestrians, and buildings of various shapes and colors, are more complex than images of the sky that contain only white clouds against a blue background.

    ❖When a model lacks sufficient capacity or the training data set doesn’t adequately represent the range of real-world variation, it can lead to problems known as underfitting and overfitting.

TRAIN THE MODEL

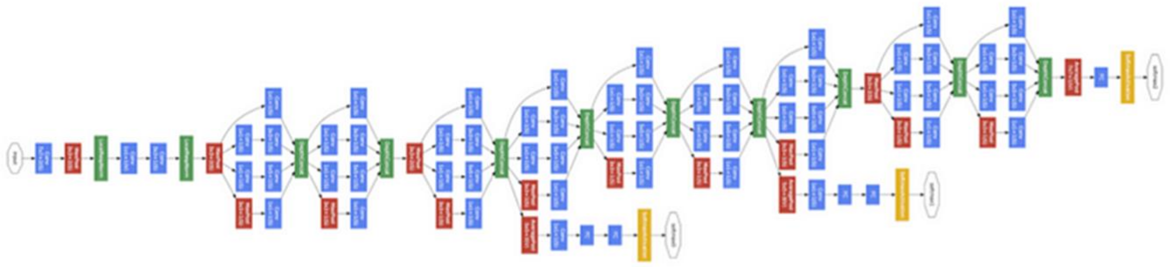
- **Underfitting**
  - Case 1: The network has a small capacity and the complexity of information in the training data set is high.
  - Consider training a two-layer convolutional neural network to classify photos in the ImageNet data set, which contains 1,000 classes. Such a network is too simple to represent the data's salient features.
  - In this case, the model will learn a simple decision boundary that doesn't correspond to the structure of the training data. Consequently, it will misclassify many data points.
  - Such a model is said to **underfit** the training data set.





TRAIN THE MODEL

- **Overfitting**
  - Case 2: The network has a large capacity, but the complexity of information in the training data set is low.
  - You might try training a 100-layer convolutional neural network to detect, in images of the sky, clouds (label 1) or no clouds (label 0). This network is more capable than the task requires.
  - The network's capacity is sufficient to learn the mapping between the training images and their labels. But the network learns salient features that are particular to these particular images. It doesn't learn inherent features such as the colors of the sky. Given new data, it tends not to find an accurate mapping. In other words, it doesn't generalize to novel data.
  - This model is said to **overfit** the training set.



TRAIN THE MODEL

- **Generalization**
  - How to assess a model's ability to **generalize**
  - How can you determine when **regularization** is in order?

    You can estimate a model's ability to generalize by splitting training data into three subsets: training, validation and test.

    - If a model performs well on the training set but not the validation or test sets, it is **overfitting**.
    - If it turns in mediocre performance on all three sets, it is **underfitting**.
    - If it is trained on the training set and tuned on the validation set, and it still performs well on the test set, the model is able to **generalize** successfully.

TRAIN THE MODEL

- **Generalization**
  - To fix **underfitting**, deepen your neural network by adding more layers.
  - To fix **overfitting**, reduce the model's capacity by removing layers and thus reducing the number of parameters.
  - The best way to help a model generalize is to gather a larger data set. But this is not always possible. If you don't have access to more data, you can use regularization methods.
  - The aim to close the performance gap between the test set and the validation and training sets while keeping training performance as high as possible. Regularization will help you do so.

TRAIN THE MODEL

➢ **Regularization techniques**

When a neural network **overfits** on the training dataset, it learns an overly complex representation that models the training dataset too well. As a result, it performs exceptionally well on the training dataset but generalizes poorly to unseen test data.

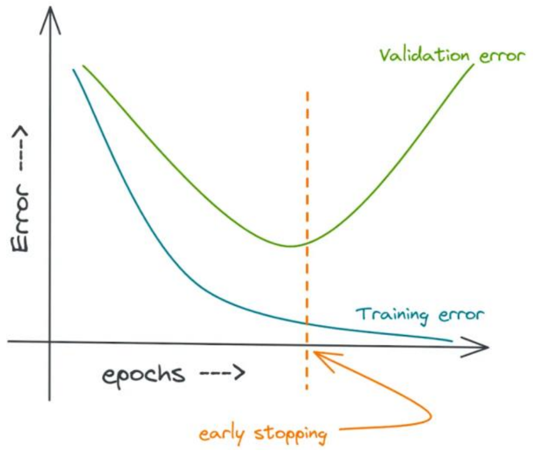
Regularization techniques help improve a neural network's generalization ability by reducing overfitting. They do this by minimizing needless complexity and exposing the network to more diverse data.

- **Common regularization techniques:**
  - Early stopping
  - Data augmentation
  - Dropout
  - Addition of noise
  - L1 and L2 regularization

TRAIN THE MODEL

➢Regularization techniques

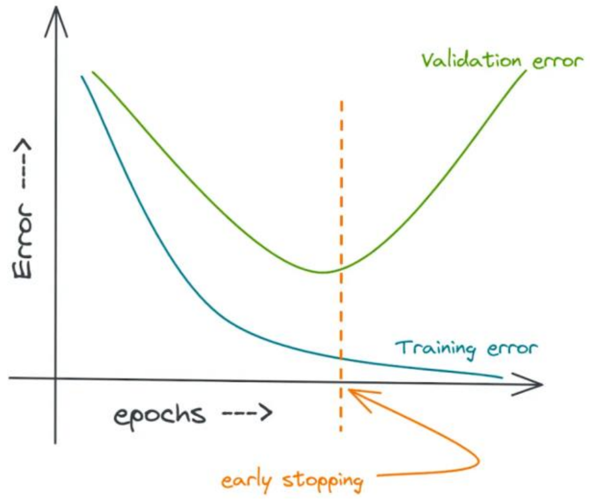
- **Early Stopping:** Early stopping is one of the simplest and most intuitive regularization techniques. It involves stopping the training of the neural network at an earlier epoch; hence the name early stopping.
  - ❖**How and when do we stop?** As you train the neural network over many epochs, the training error decreases. If the training error becomes too low and reaches arbitrarily close to zero, then the network is sure to overfit on the training dataset. Such a neural network is a high variance model that performs badly on test data that it has never seen before despite its near-perfect performance on the training samples. Therefore, heuristically, if we can prevent the training loss from becoming arbitrarily low, the model is less likely to overfit on the training dataset, and will generalize better.

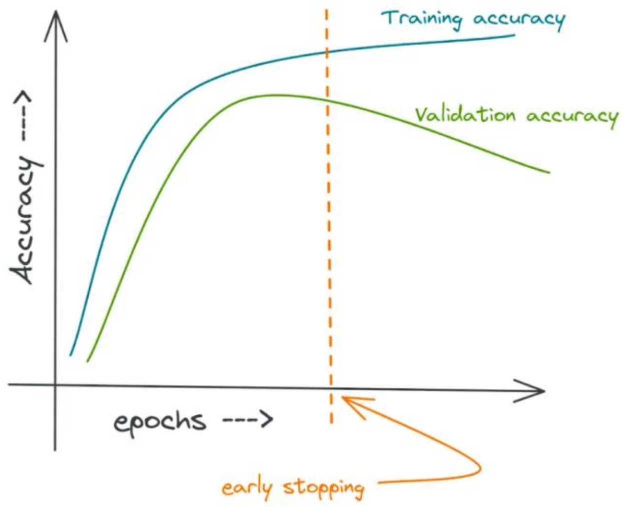


TRAIN THE MODEL

- **Regularization techniques**
  - **Early Stopping**
    - **How do we do it in practice?** You can monitor one of the following:

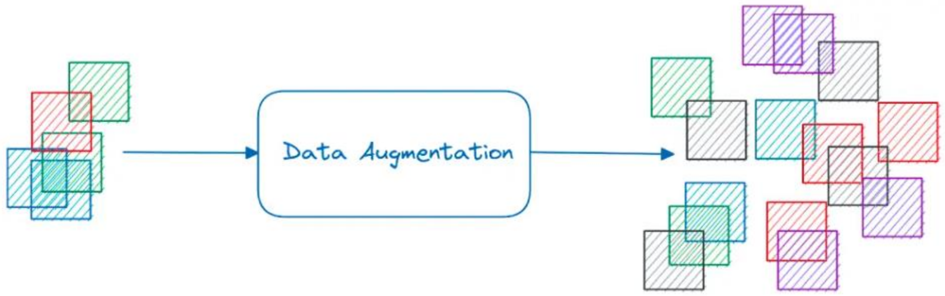
      The change in metrics such as validation error and validation accuracy: A simple approach is to monitor metrics such as validation error and validation accuracy as the neural network training proceeds, and use them to decide when to stop.





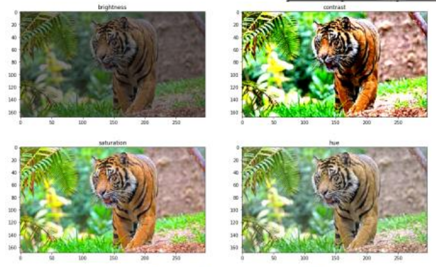
TRAIN THE MODEL

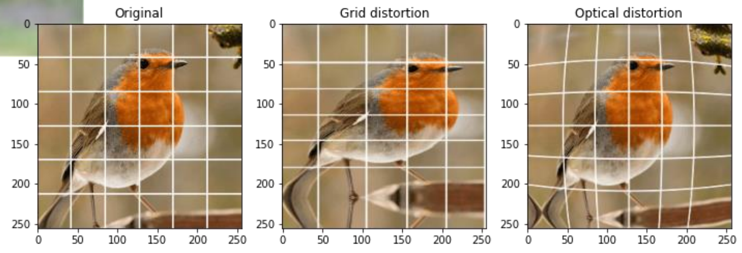
- **Regularization techniques**
  - **Data Augmentation**
    - Data augmentation is a regularization technique that helps a neural network generalize better by exposing it to a more diverse set of training examples.
    - As deep neural networks require a large training dataset, data augmentation is also helpful when we have insufficient data to train a neural network.
    - Let's take the example of image data augmentation. Suppose we have a dataset with **N** training examples across **C** classes. We can apply certain transformations to these **N** images to construct a larger dataset.

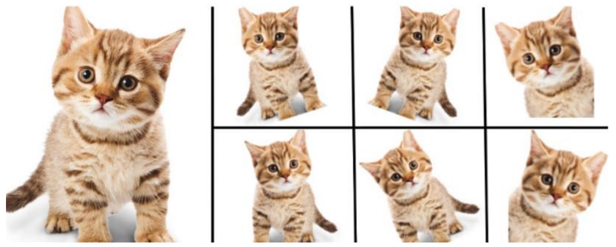


TRAIN THE MODEL

- **Regularization techniques**
  - **Data Augmentation**
    - **What is a valid transformation?** Any operation that does not alter the original label is a valid transformation.
    - **In summary**: we can apply any **label-invariant transformation** to perform data augmentation. The following are some examples:
      - Color space transformations such as change of pixel intensities
      - Rotation and mirroring
      - Noise injection, distortion, and blurring

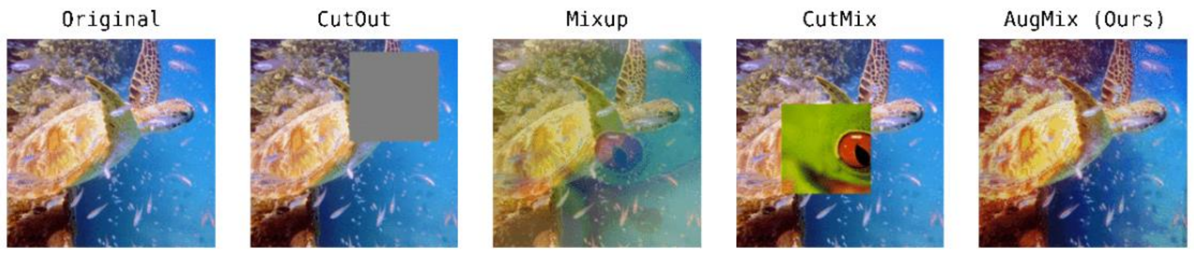






TRAIN THE MODEL

- **Regularization techniques**
  - **Data Augmentation-**Newer Approaches to Image Data Augmentation
    - **Mixup** generates a weighted combination of random image pairs from the training data.
    - **Cutout** involves the random removal of portions of an input image during training.
    - **CutMix** replaces the removed sections with parts of another image.
    - **AugMix** is a regularization technique that makes a neural network robust to distribution change. Unlike mixup that uses images from two different classes, AugMix performs a series of transformations on the same image, and then uses a composition of these transformed images to get the resultant image.

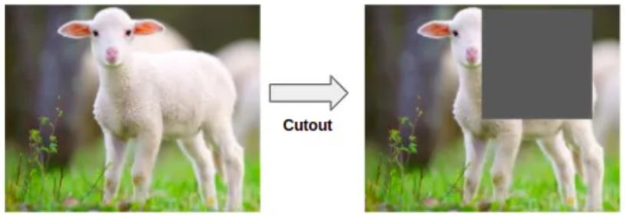


TRAIN THE MODEL

- **Regularization techniques**
  - **Data Augmentation-**Newer Approaches to Image Data Augmentation
    - **Cutout**

    Cutout is an augmentation technique that randomly covers a region of an input image with a square.

    - **Advantages**
      1. Helps in training models to recognize partial or occuluded objects.
      2. Allows the model to consider more of the image context such as minor features rather than relying heavily on major features before making a decision.



TRAIN THE MODEL

- **Regularization techniques**
  - **Data Augmentation-**Newer Approaches to Image Data Augmentation
    - **Cutout**
      - **Limitations**
        1. It can completely remove important features from an image.
        2. It may not work well for images with complex backgrounds, and it may not be effective for small-sized patches or small datasets.
        3. One limitation of removing a square region from an image and filling it with black, grey, or Gaussian noise pixels is that it significantly reduces the proportion of informative pixels used in the training process, which can be problematic for CNNs that require large amounts of data to learn effectively.
      - **Hyperparameter**

        Size and number of patches to be cut out from the image.

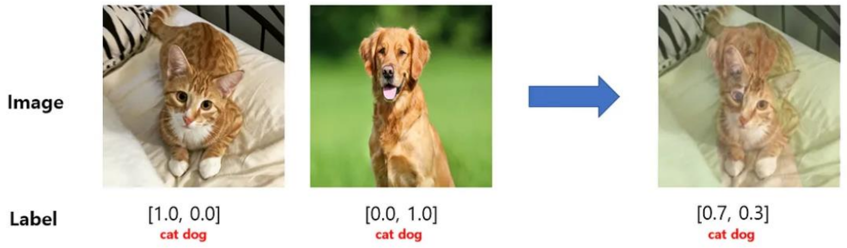
TRAIN THE MODEL

- **Regularization techniques**
  - **Data Augmentation-**Newer Approaches to Image Data Augmentation
    - **Mixup Augmentation**

      Mixup generates a weighted combination of random image pairs from the training data. Given two images and their ground truth labels: (xi,yi),(xj,yj), a synthetic training example (x,y) is generated as:

$$\boxed{\begin{aligned}\hat{x} &= \lambda x_i + (1-\lambda)x_j,\\ \hat{y} &= \lambda y_i + (1-\lambda)y_j,\end{aligned}}$$

where $\lambda \in [0,1]$ is a random number



❖**Note**: the lambda values are values with the [0, 1] range.

TRAIN THE MODEL

- **Regularization techniques**
  - **Data Augmentation-**Newer Approaches to Image Data Augmentation
    - **Mixup Augmentation**

Advantages

1. Neural networks are prone to memorizing corrupt labels. Mixup relaxes this by combining different features with one another (same happens for the labels too) so that a network does not get overconfident about the relationship between the features and their labels.
2. It makes decision boundaries transit linearly from class to class, providing a smoother estimate of uncertainty.

Limitations

1. Only inter-class mixup. For intra class mixup, interpolating only between inputs with equal label did not lead to the performance gains of mixup.
2. The examples are not real representation of class. (unnatural may confuse the model)
3. Label smoothing and mixup usually do not work well together because label smoothing already modifies the hard labels by some factor.

TRAIN THE MODEL

- **Regularization techniques**
  - **Data Augmentation-**Newer Approaches to Image Data Augmentation
    - **Cutmix**

With Cutmix, a square region of an input image is replaced with a patch of similar dimensions from another image, which leads to a more natural-looking output. The ground truth labels of the resulting images are mixed proportionally to the number of pixels from each image, creating a linearly interpolated label that reflects the contribution of both original images.

$$\tilde{x} = M x_i + (1 - M) x_j$$

$$\tilde{y} = \lambda y_i + (1 - \lambda) y_j$$

where M is a binary mask(often square) indicating the Cutout and the fill-in regions from the two randomly drawn images. Just like mixup, λ is drawn from λ∈ [0, 1].

TRAIN THE MODEL

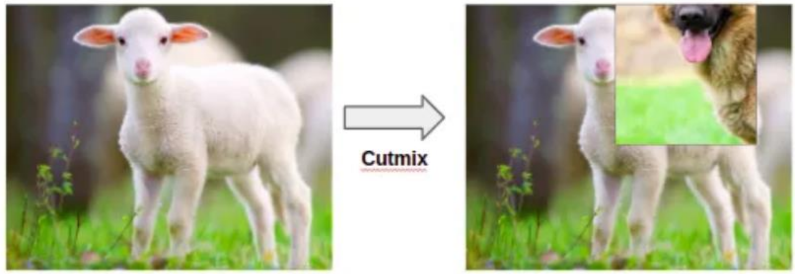
- **Regularization techniques**
  - **Data Augmentation-**Newer Approaches to Image Data Augmentation
    - **Cutmix**

      After the images are randomly selected, bounding box coordinates are sampled such that B = (rx,ry,rw,rh) indicates the Cutout and fill-in regions in both the images. The bounding box sampling is given by,

$$r_x \sim U(0,\ W),\ r_w = W\sqrt{1-\lambda}$$

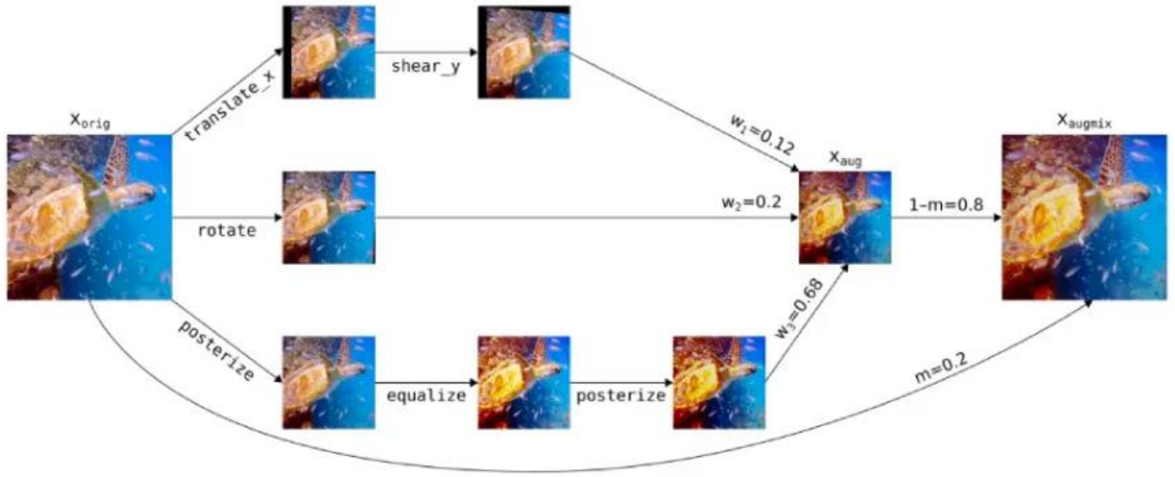
$$r_y \sim U(0,\ H),\ r_h = H\sqrt{1-\lambda}$$

where rx,ry are randomly drawn from a uniform distribution with upper bound as shown.



TRAIN THE MODEL

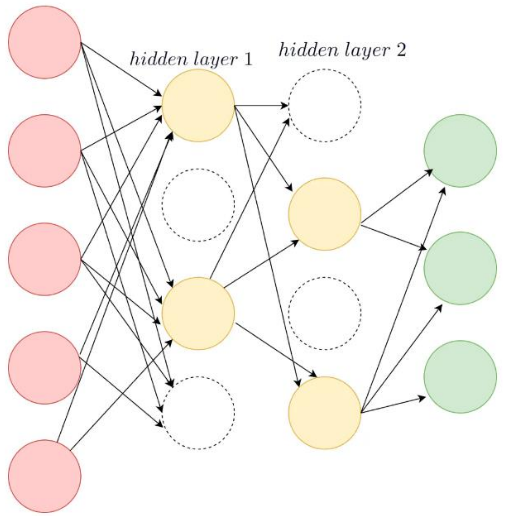
- **Regularization techniques**
  - **Data Augmentation-**Newer Approaches to Image Data Augmentation
    - **AugMix** is a regularization technique that makes a neural network robust to distribution change. Unlike mixup that uses images from two different classes, AugMix performs a series of transformations on the same image, and then uses a composition of these transformed images to get the resultant image.



TRAIN THE MODEL

- **Regularization techniques**
  - **Dropout**

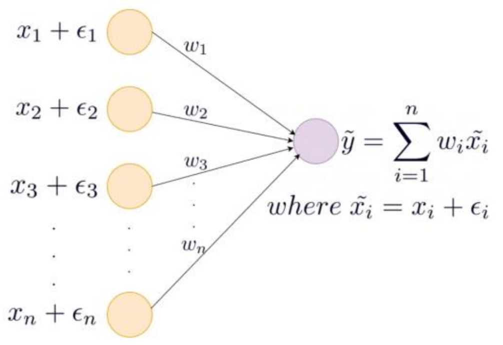
Dropout helps us emulate model ensembles by dropping out neurons during the training process with a fixed probability **p**, ensuring that a different neural network architecture is used for different batches during training. We weight a neuron's output at test time by the same probability **p**.



TRAIN THE MODEL

- **Regularization techniques**
  - **Addition of Noise**

    Another regularization approach is the addition of noise. You can add noise to the input, the output labels, or the gradients of the neural network.



Adding Noise to Inputs

Class Label	1	2	3	...	k	...	N
With Label Smoothing	$\frac{1-\epsilon}{N-1}$	$\frac{1-\epsilon}{N-1}$	$\frac{1-\epsilon}{N-1}$	...	ϵ	...	$\frac{1-\epsilon}{N-1}$

Adding Noise to Output Labels

$$\mathbf{g}_t = \mathbf{g}_t + \mathcal{N}(0, {\sigma_t}^2)$$

Adding Noise to Gradients

TRAIN THE MODEL

- **Regularization techniques**
  - **L1 and L2 Regularization**

In general, Lp norms (for p>=1) penalize larger weights. They force the norm of the weight vector to stay sufficiently small. The Lp norm of a vector **x** in n-dimensional space is given by:

$$Lp(\mathbf{x}) = ||\mathbf{x}||_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p} \qquad L_1(\mathbf{x}) = ||\mathbf{x}||_1 = \sum_{i=1}^{n} |x_i| \qquad L_2(\mathbf{x}) = ||\mathbf{x}||_2 = (\sum_{i=1}^{n} |x_i|^2)^{1/2}$$

Let L and $\tilde{L}$ be the loss functions without and with regularization, respectively. The regularized loss function is given by:

$$\tilde{\mathcal{L}}(\mathbf{w}) = \mathcal{L}(\mathbf{w}) + \alpha||\mathbf{w}||_1$$

where, α is the regularization constant. Suppose α is a sufficiently large constant.

When the weights become too large, the second term $\alpha||w||_1$ increases. However, as the goal of optimization is to minimize the loss function, the weights should decrease.

TRAIN THE MODEL

- **Batch size**
  - Batch size is the number of samples that you feed into your model at each iteration of the training process.
  - It determines how often you update the model parameters based on the gradient of the loss function.
  - A larger batch size means more data per update, but also more memory and computation requirements.
  - A smaller batch size means less data per update, but also more noise and variance in the gradient.

TRAIN THE MODEL

- **Batch size**
  - What are some practical guidelines?
    - Finding the best batch size for optimizing deep learning models is not an exact science, as it depends on many factors like model architecture, data distribution, hardware configuration, and optimization algorithm.
    - There are some practical guidelines that can be followed to select a reasonable batch size. For instance, it is recommended to start with a small batch size such as 32 or 64 and increase it gradually until the validation performance decreases or the training time increases.
    - Using a batch size that is a power of 2 can improve the efficiency and compatibility of the computation on GPUs and other hardware devices.
    - Furthermore, the batch size should be divisible by the number of devices used for parallel or distributed training in order to avoid wasting resources and ensure a balanced workload.
    - The batch size should be large enough to fit in the memory of your device but not too large to cause memory overflow or underutilization.

TRAIN THE MODEL

- **Batch size**
  - What are some practical guidelines?
    - It should be consistent with any regularization techniques used like dropout, or batch normalization
    - It should be suitable for the optimization algorithm used such as SGD, Adam, or RMSprop; the learning rate and other hyperparameters should be tuned accordingly if the batch size is changed.
    - Finally, experimenting with different batch sizes and comparing results on validation and test data will help determine which one gives the best trade-off between speed, accuracy, and generalization.

TEST THE MODEL

- **Evaluation metrics:** A metric is a function that is used to judge the performance of your model.
  - Metric functions are similar to loss functions, except that the results from evaluating a metric are not used when training the model.
  - Note that you may use any loss function as a metric.

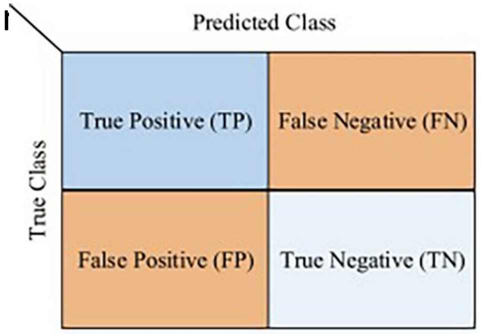
- **Available metrics**
  - Accuracy metrics
  - Probabilistic metrics
  - Regression metrics
  - Classification metrics based on True/False positives & negatives
  - Image segmentation metrics
  - …

TEST THE MODEL

- **Evaluation metrics in classification models**
  - **What is confusion matrix**

Confusion matrix shows the classification performance of a model in tabular format in terms of four parameters, **True Positive (TP)**, **True Negative (TN), False Positive (FP)**, and **False Negative (FN)**.

- **True negative** measures how many predicted 0s (that is, no cat) are actually correct.
- **False positive** measures how many predicted 1s are actually 0; that is, the classifier predicted that the image contains a cat, but actually, there is no cat.
- **False negative** measures how many predicted 0s are actually 1; that is, the classifier has predicted that the image does not contain any cat, but there are actually cats in the i



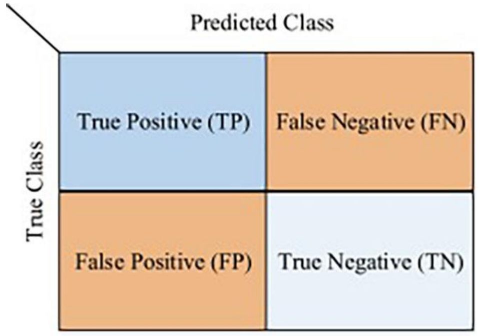
Confusion matrix for a binary classifier

TEST THE MODEL

- **Evaluation metrics in classification models**
  - **What is confusion matrix**

    Confusion matrix shows the classification performance of a model in tabular format in terms of four parameters, **True Positive (TP)**, **True Negative (TN), False Positive (FP)**, and **False Negative (FN)**.

  - Suppose you have designed a binary classifier for identifying images of cats.
    - Your classifier gives an output **1** if it predicts a **cat** in the image and
    - returns **0** if **no cat** is detected.
    - **True positive** measures how many 1s predicted by the classifier are correct based on their actual labels.



Confusion matrix for a binary classifier

TEST THE MODEL

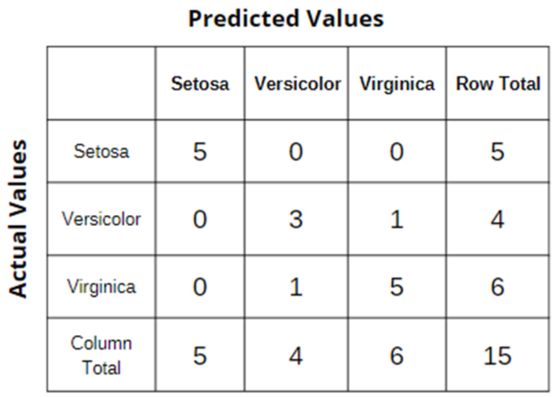
➢Evaluation metrics in classification models

- **What is confusion matrix**

Confusion matrix can also be derived for both binary and multi-class classifiers. For multi-class classifiers, the parameters TP, TN, FP, and FN are measured in one-vs-all strategy. Where we spilt the problem into multiple binary classification problems per class.

- **How to Read a Classification Matrix for Multiclass**

Now that we have our confusion matrix ready, let's calculate the TP, TN, FP, and FN values for **Setosa** species.
TP -> 5 (case where the predicted values match the actual values)
TN -> 10 (cases except for the values of the class for which we are computing the values, the sum of all columns and rows)
FP -> 0 (sum of values of the columns except TP)
FN -> 0 (sum of values of the row except for TP)
Next, we will calculate the TP, TN, FP and FN values for **versicolor** species.
TP -> 3, TN -> 10, FP -> 1, FN -> 1
Lastly we will calculate the TP, TN, FP and FN values for **virginica** species.
TP -> 5, TN -> 8, FP -> 1, FN -> 1



Predicted Values

Actual Values

	Setosa	Versicolor	Virginica	Row Total
Setosa	5	0	0	5
Versicolor	0	3	1	4
Virginica	0	1	5	6
Column Total	5	4	6	15

TEST THE MODEL

- **Evaluation metrics in classification models**
  - **Classification report**

    With the confusion matrix, we can define a number of metrics to evaluate the classification performance.

    - **Recall or Sensitivity**: It measures how many positive samples have been correctly predicted by the classifier. It is given by the following equation:

      True Positive Rate (TPR) or Hit Rate or Sensitivity or Recall = $\frac{TP}{TP+FN}$

    - **Precision**: It measures out of all predicted positive samples by the classifier how many are actually correct. It is given by the following:

      Precision $= \frac{TP}{TP+FP}$

    - **Specificity**: It measures the fraction of negative samples that are correctly predicted. Specificity= $\frac{TN}{TN+FP}$

TEST THE MODEL

- **Evaluation metrics in classification models**
  - **Classification report**
    - **Classification accuracy**: It is a measure of the fraction of all correctly predicted samples by the classifier. Classification accuracy = $\frac{TP+TN}{TP+FP+TN+FN}$
    - **F1 score**: This is a weighted average of precision and recall. It is calculated as the harmonic mean of precision and recall. Refer to the following equation: F1 score = $\frac{2*Precision\ .Recall}{Precision+Recall}$

      F1-score takes into account both precision and recall in a single metric. Hence, it is widely considered as a better performance metric compared to classification accuracy, particularly to evaluate on imbalanced datasets.
    - **False Positive Rate(FPR)** or False Alarm Rate = 1 - Specificity = 1 - (TN / (TN + FP))