Unit 4. Artificial Neural Network (12 Hrs.)

Objective:

- ✓ Design and implement Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs)

4.1. Introduction to Neural Network

Artificial neural networks (ANNs) are powerful tools / algorithm in machine learning that are modeled after the structure of the human brain and designed to function as an artificial human brain.

Neural networks, also called artificial neural networks or simulated neural networks, are a subset of machine learning and are the backbone of deep learning algorithms. They are called "neural" because they mimic how neurons in the brain signal one another.

Artificial Neural Networks contain artificial neurons, which are called units. These units are arranged in a series of layers that together constitute the whole Artificial Neural Network in a system. A layer can have only a dozen units or millions of units, as this depends on how the complex neural networks will be required to learn the hidden patterns in the dataset.

Basic Components of ANN:

1. **Neurons / Nodes / Units**

- ✓ A neuron is a unit that takes input values, applies a weighted sum, adds a bias, and passes the result through an activation function.
- ✓ Similar to the nucleus of brain cells, each neuron, except those in the Input layer, contains a bias parameter that the Neural Network learns and adjusts during the training process

2. **Layers:**

- ✓ Layers are groups of neurons that process and pass information through the network
- ✓ Basically 3 layers - Input, hidden, output

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

3. **Connections**

- ✓ Each neuron in one layer is connected to every neuron in the next layer (in a fully connected ANN).
- ✓ Weights are assigned to these connections.
- ✓ These weights are learned during training.
- ✓ Think of connections as memory- the way the network learns and stores patterns.

4. **Activation Function**

- ✓ It decides whether a neuron should "fire" or activate, and adds non-linearity to the network. Without it, the network would just be a linear model — not capable of learning complex patterns.
- ✓ Common activation functions are: Sigmoid, ReLU (Rectified Linear Unit), Tanh, Softmax.

Application of ANN:

Artificial neural networks find applications across a wide range of fields, including:

- **Computer Vision:** Analyzing and interpreting image and videos such as image classification, object detection, and image segmentation. E.g. Google Photos uses CNNs for image categorization and face recognition.
- **Natural Language Processing:** Recurrent Neural Networks (RNNs) and their variants are used for tasks like machine translation, sentiment analysis, and text generation, chatbots, email filters, and voice assistants.
- **Finance:** Such as stock market prediction, risk assessment, algorithmic trading, and fraud detection, to determine whether to approve or deny a loan application based on risk. E.g. PayPal uses NN to detect fraudulent transactions.
- **Healthcare:** They play a crucial role in medical image analysis, disease diagnosis, drug discovery, and personalized medicine.
- **Autonomous Systems:** ANNs power self-driving cars, drones, and robotics by enabling them to perceive and navigate their environments. CNN is used. E.g. Tesla's Autopilot uses neural networks to make real-time driving decisions.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

- **Entertainment and media**

  Neural networks find their use in video streaming, content recommendation, and content generation. They help personalize user experiences by analyzing viewing patterns and preferences. E.g. Netflix uses neural networks to recommend movies and shows based on viewing history.

- **Agriculture**

  Neural networks are used for crop prediction, precision farming, disease detection, and automated harvesting. They can analyze satellite and drone imagery to detect patterns and predict crop yields. E.g. John Deere uses neural networks in their farming equipment for automated planting and harvesting.

Challenges / limitation of ANN

- **Data size requirements:** ANNs need big amounts of labeled data to train well. If you don't have enough data, the network might not have enough input to make accurate predictions. For instance, if you train a CNN on only a few images, it might not be able to recognize new images at a high level.
- **Change sensitivity:** ANNs can be sensitive to differences in the input data, which can cause mistakes or unexpected results. For example, if you're using a CNN to identify an animal and the animal is hiding or in a weird position, the algorithm might not recognize it. If you use the same data to train ANNs as you do to test them, you may limit the algorithm's accuracy.
- **Flexibility limitations:** ANNs that you trained for one task might not work well for other tasks or areas. For instance, if you train a CNN to recognize faces, it might not be very good at recognizing things in other kinds of pictures, like landscapes.

4.1.1. Neural Network Architectures

A neural network architecture represents the structure and organization of an artificial neural network (ANN), which is a computational model inspired by the workings of a biological neural network.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

The Artificial Neural Network (ANN) architecture refers to the structured arrangement of nodes (neurons) and layers that define how an artificial neural network processes and learns from data. The design of ANN influences its ability to learn complex patterns and perform tasks efficiently.

Just like the human brain processes information through interconnected neurons, ANNs use layers of artificial neurons to learn patterns and make predictions.

The architecture explains how data flows through the network, how neurons (units) are connected, and how the network learns and makes predictions.

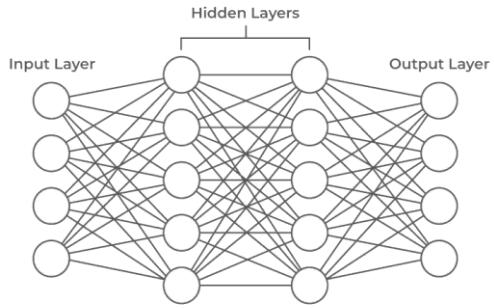


Figure: Architecture of Neural Network

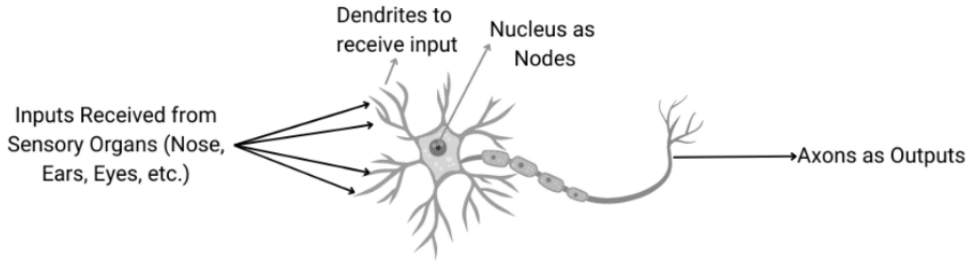


Figure: Biological Neuron to Artificial Neuron

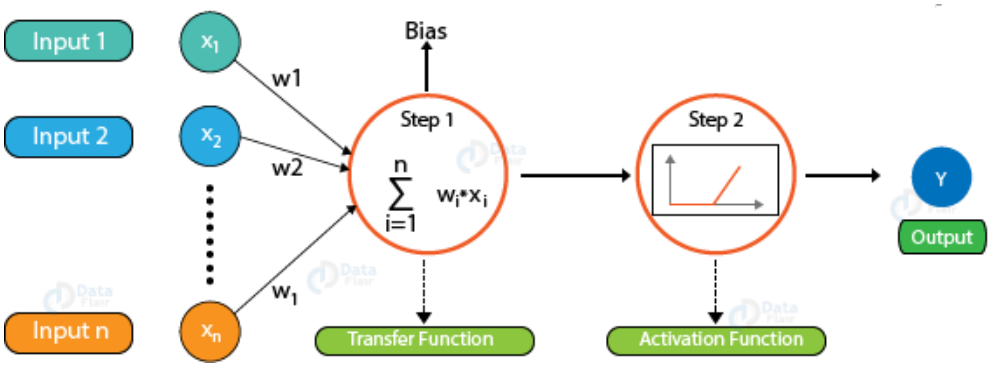


Figure: Functional Architecture of Neural Network [IS: Dataflair]

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

key components of a neural network architecture.

- **Layers:** Neural networks consist of layers of neurons, which include the input layers, hidden layers, and output layers.
    - ✓ **Input Layer**

      The input layer is where data enters the system. Each neuron corresponds to a feature in the input dataset. It passes raw inputs (such as pixel values in an image) to the next layers for further processing.
    - ✓ **Hidden Layer**

      The hidden layers are present between the input and output layers and are where the actual learning and computation happen. Hidden layers transform the input data, allowing the network to learn complex patterns and abstract representations. The more hidden layers, the more complicated patterns it can learn.
    - ✓ **Output Layer**

      The output layer is responsible for producing the network's predictions or classifications based on the transformed data from the hidden layers. The output layer maps the learned features, such as class labels in classification tasks, from the hidden layers to the final output.

- **Neurons (Nodes):**

  building blocks of every layer. Each neuron performs a mathematical operation: Takes weighted inputs, applies a bias, Passes the result through an activation function

- **Weights and biases:**

  Weights are the parameters that control the strength of the connections between neurons, while biases are additional parameters that allow the network to shift the activation function (i.e. biases allow neurons to make predictions even when all inputs are zero) The network adjusts weights and biases during training to minimize the mistakes between predicted and actual outputs.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

Together, weights and biases are updated during training to improve the network's predictions

- **Activation function:**

  Enabling it to learn complex patterns. Common activation functions include: Sigmoid: Used in binary classification. ReLU (Rectified Linear Unit): Popular for hidden layers. Softmax: Ideal for multi-class classification.

- **Loss Function:**

  Quantifies how well the network's predictions match the actual outputs. It guides the network's learning process. Common loss functions include: Mean Squared Error (MSE): For regression tasks. Cross-Entropy Loss: For classification tasks. The loss function is critical for training, as it guides the optimization process.

- **optimization algorithm**

  It adjusts the weights and biases to minimize the loss function. The most commonly used optimizer is Stochastic Gradient Descent (SGD), often enhanced with variants like Adam.

how the architecture of neural networks defines its capabilities?

1. Model's capacity: model with high depth (number of layers) and width (number of neurons in each layer) can handle complex relationships. A network with few layers cannot handle tasks like image or speech recognition.
2. Efficiency: The neural network's architecture affects the efficiency of the model. For instance, convolutional neural networks (CNNs) have lower computational costs compared to fully connected networks.
3. Optimization: The network's structure affects its optimization. For instance, deeper networks may face issues like vanishing gradients, where the gradients become too small for effective learning in early layers.
4. Task-specific design: The architecture of networks can be tailored to specific tasks, such as CNNs are suitable for image data and RNNs or transformers are preferred for sequence prediction or sequential functions like speech and text analysis.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

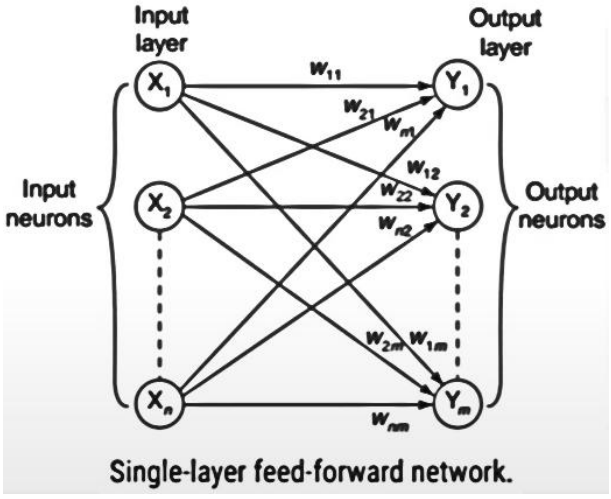
There are three different types of Neural Network Architecture:

Neural networks are highly task-specific, and no single architecture works for all types of problems. Choosing the right architecture is critical to achieving high performance, increasing the model's ability to learn from data and make accurate predictions.

4.1.1.1. Feedforward

Feedforward Neural Networks (FNNs) are the simplest of neural networks, with one or more layers of neurons that you use for sorting and predicting things. where data flows from the input layer to the output layer without any cycles or loops (i.e. unidirectional flow of information)

- The term "feedforward" highlights how data travels strictly forward through layers, without feedback connections (i.e. the data or the input provided travels in a single direction. It enters into the ANN through the input layer and exits through the output layer while hidden layers may or may not exist)
- These types of neural networks are mostly used in ***supervised learning*** for instances such as classification, regression, image recognition etc.
- In this architecture, the interconnected neurons are arranged in layers, with each layer fully connected to the next.
- These networks follow a simple but powerful structure, making them ideal for classification, regression, and pattern recognition tasks.
- Used in speech recognition, medical imaging, and financial forecasting.



Single-layer feed-forward network.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

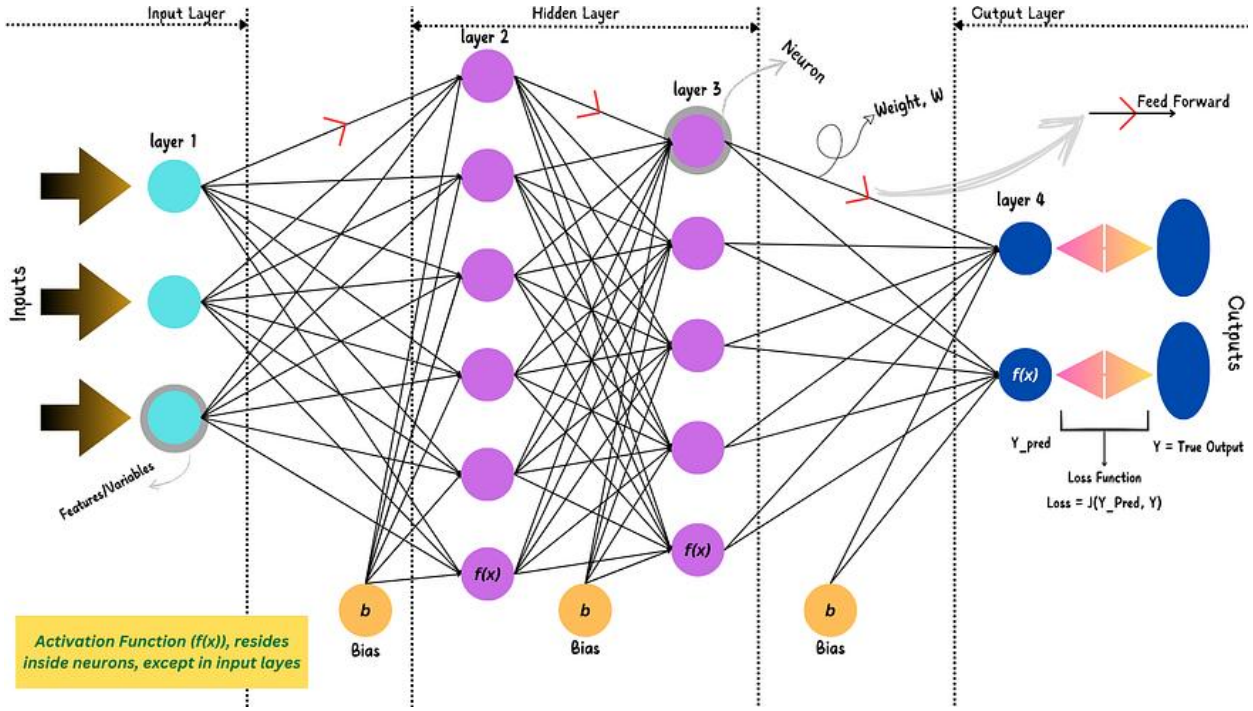


Figure: Feed-forward Neural Network with two hidden layers [IS: Medium]

Feedforward neural networks process data through multiple layers, refining it at each stage.

The network applies weights and biases to the input before passing it through activation functions to make predictions.

Components of FNNs

Input Layer

It receives raw data and forwards it to the next layer without modifying it. Each neuron in this layer represents one feature of the input data, ensuring the network processes all relevant information:

- ✓ Passes Information Without Processing: Unlike other layers, it does not perform computations.
- ✓ Determines Network Input Size: The number of neurons in this layer depends on the input data's dimensions. A grayscale image of 28x28 pixels requires 784 neurons, one for each pixel
- ✓ Ensures Structured Data Flow

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

Example: For a house price prediction problem, the input layer may include nodes for features like square footage, number of bedrooms, location, etc.

Hidden Layer

The hidden layer is where most of the computations occur in a feedforward neural network. It transforms raw data into meaningful patterns using weights, biases, and activation functions. Multiple hidden layers in networks allow better feature extraction and hidden layers extract hierarchical features that improve classification accuracy

- **Depth** refers to the number of hidden layers.
- **Width** refers to the number of neurons in a hidden layer

how hidden layers contribute to processing?

- ✓ Applies Weights and Biases: Each neuron modifies input values using weights and biases, improving the network's ability to detect patterns.
- ✓ Uses Activation Functions: Functions like ReLU and sigmoid introduce non-linearity, enabling complex pattern recognition.
- ✓ Extracts Features at Multiple Levels: Early layers detect simple edges, while deeper layers recognize objects. In a handwriting recognition task, the network first identifies lines and later distinguishes entire letters.
- ✓ Connects Input to Output for Prediction: Hidden layers bridge the gap between raw input and final classification. For instance, in a spam detection model, hidden neurons identify suspicious patterns in emails.

Output Layer:

The output layer produces the final result of a feedforward neural network. It takes processed data from the hidden layers and converts it into a prediction. The number of neurons in this layer depends on the task, such as classification or regression.

What is the role of the output layer in NN?

- ✓ Converts Processed Data into Output
- ✓ Applies Activation Functions for Probability Scores: Softmax and sigmoid functions are common in classification tasks. In a digit recognition model, Softmax assigns probabilities to numbers from 0 to 9.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

- ✓ Adapts to Task Requirements: The number of neurons depends on the output type. A binary classification task, like spam detection, has one neuron, while a multi-class problem, like sentiment analysis, has multiple neurons.
- ✓ Completes the Learning Process: The output layer finalizes the network's decision based on learned patterns. In fraud detection, it classifies transactions as either genuine or fraudulent.

How Does a Feedforward Neural Network Work?

Step-1: Forward Pass:

Data flows through the network from the input layer to the output layer. At each layer, inputs are multiplied by weights, added to biases, and passed through an activation function.

Step-2: Loss Calculation:

The loss function calculates the difference between predicted and actual values.

Step-3: Backward Pass / Backpropagation (Training):

In backpropagation the error is propagated back through the network to update the weights. The gradient of the loss function with respect to each weight is calculated and the weights are adjusted using gradient descent.

Using an algorithm like backpropagation, the network calculates the gradients of the loss function with respect to the weights and biases. These gradients are used to update the weights and biases to reduce the loss.

Gradient Descent

Gradient Descent is an optimization algorithm used to minimize the loss function by iteratively updating the weights in the direction of the negative gradient. Common variants of gradient descent include:

- **Batch Gradient Descent:** Updates weights after computing the gradient over the entire dataset.
- **Stochastic Gradient Descent (SGD):** Updates weights for each training example individually.

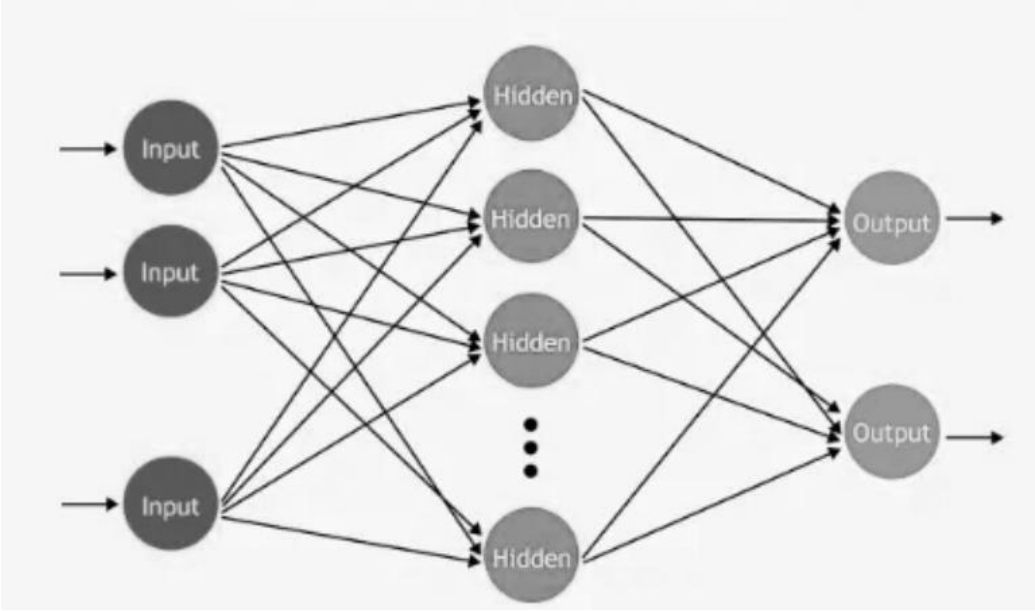
By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

- **Mini-batch Gradient Descent:** It Updates weights after computing the gradient over a small batch of training examples.

Repeat:

Steps 1–3 are repeated for multiple iterations (epochs) until the loss is minimized.

4.1.1.2. Convolution



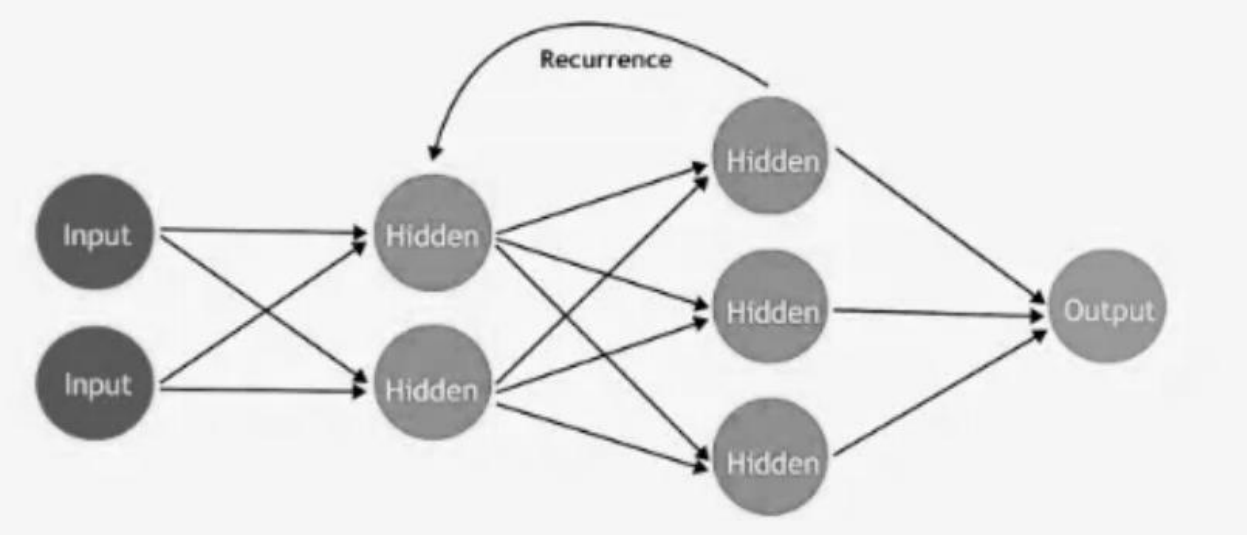
Convolutional Neural Networks (CNNs) are designed to process grid-like data, especially images. The layers used in CNNs perform convolutions to automatically extract features from images, such as edges, textures, and shapes. These features are then used to recognize objects, patterns, or classify images.

CNNs consist of three main types of layers:

- **Convolutional layers:** Detect local features using convolution operations.
- **Pooling layers:** Minimizes computational load and avoids overfitting by reducing spatial dimensions.
- **Fully-connected layers:** These layers, present at the end of the network, integrate the features learned in previous layers to make final predictions.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

4.1.1.3. Recurrent



RNNs have a "memory" component, where information can flow in cycles through the network.

Recurrent Neural Networks (RNNs) can process sequential data where the order of the input's matters. RNNs contain loops that allow information to be passed from one step to the next, making them suitable for tasks that involve time-series data or sequences of information.

4.1.2. Perceptron

Perceptron is the first neural network model designed in 1958, which is linear binary classifier used for supervised learning. This algorithm enables neurons to learn and processes elements in the training set one at a time.

How perceptron work?

Perceptron is considered a single-layer neural link with four main parameters. The perceptron model begins with multiplying all input values and their weights, then adds these values to create the weighted sum. Further, this weighted sum is applied to the activation function 'f' to obtain the desired output. This activation function is also known as the step function and is represented by 'f.'

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

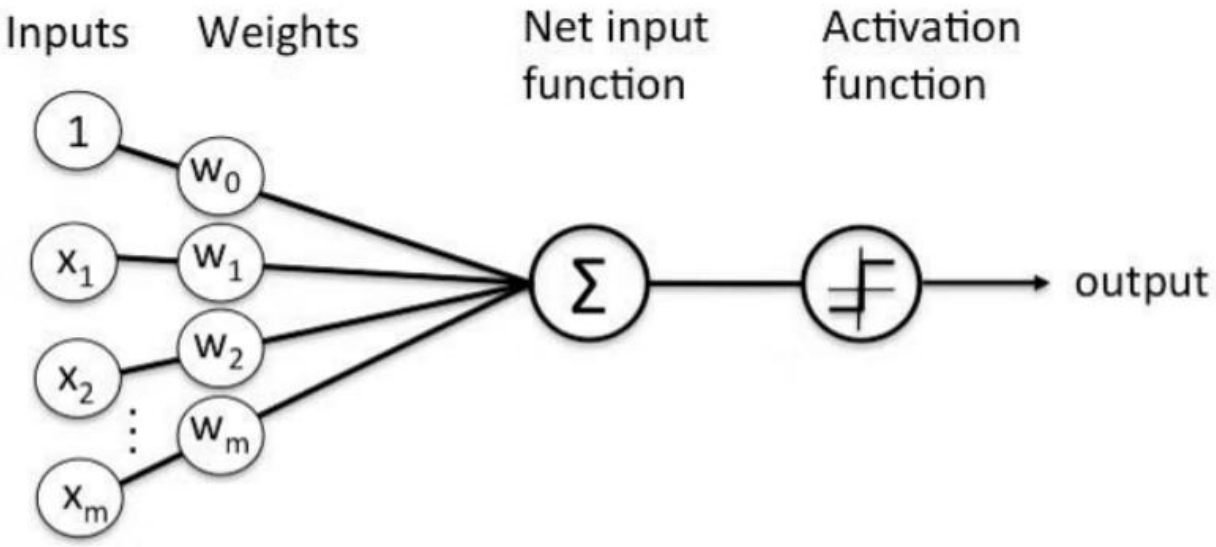


Figure: General diagram of perceptron

There are three different types of perceptron models:

4.1.2.1. Single layer perceptron

A single layer perceptron is the simplest, feed forward and based on threshold transfer function type of artificial neural network used primarily for binary classification tasks. It consists of a single layer of output nodes directly connected to the input nodes, without any hidden layers. The main goal of the single layer perceptron is to classify linearly separable data by learning a linear decision boundary.

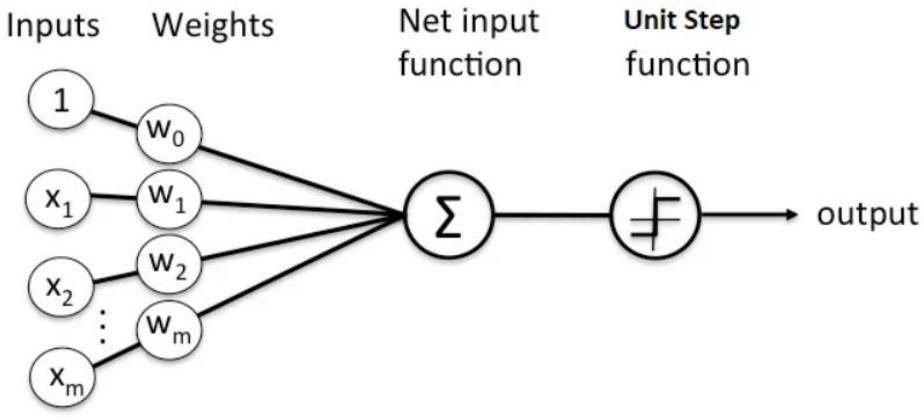


Figure: Single layer perceptron with unit step function during training

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

Components of a Single Layer Perceptron

- **Input Layer** consists of input neurons that receive the input features. Each input is associated with a weight that signifies the importance or strength of that input to the output
- Input feature is multiplied by a corresponding **weight**. These weights are parameters that the perceptron learns during training to correctly classify inputs
- **Bias** term is added to the weighted sum of inputs to shift the decision boundary away from the origin, allowing more flexibility in classification
- The perceptron computes the **weighted sum** of the inputs plus the bias $\sum w_i\, x_i$ + b
- The output of the net sum is passed through an **activation function** (in case of single layer perceptron a step function (Heaviside function), which outputs either 0 or 1). If the weighted sum is above a threshold (usually zero), the output neuron fires (outputs 1); otherwise, it outputs.
- The final **output** is a binary value indicating the class to which the input belongs.

Working Principle

The single layer perceptron works by taking the input vector x, computing the dot product with the weight vector w, adding the bias b, and then applying the activation function: f(x)=h (w· x)+b

where h is the step function that outputs 1 if the argument is positive, and 0 otherwise.

Training the Single Layer Perceptron

1. label the positive and negative class, in binary classification setting as 1 and *-1*
2. linear combination of the input values *x* and weights *w* as $(z=w_1x_1+\cdots+w_mx_m)$
3. define an activation function *g(z)*, where if *g(z)* is greater than a defined threshold θ we predict 1 and -1 otherwise; in this case, this activation function *g* is an alternative form of a simple Unit step function, which is sometimes also called Heaviside step function

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

$$g(z) = \begin{cases} 1, & \text{if } z \geq \theta \\ -1, & \text{otherwise} \end{cases}$$

Now,

$$z = \sum_{j=1}^{m} x_j w_j = w^T x$$

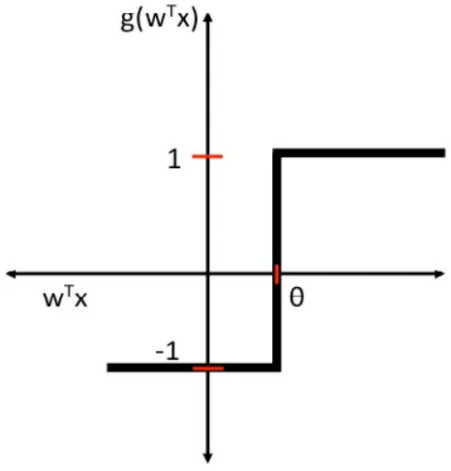
where,

- ***w*** is the feature (weight) vector,
- ***x*** is an ***m***-dimensional sample from the training dataset:

$$[w = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

In order to simplify the notation, we bring $\boldsymbol{\theta}$ to the left side of the equation and define $\boldsymbol{w_0 = -\theta}$ and $\boldsymbol{x_0 = 1}$ (also known as **bias**)

$$g(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ -1, & \text{otherwise} \end{cases}$$



By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

So that,

$$z = \sum_{j=0}^{m} x_j w_j = w^T x$$

The perceptron is trained using a supervised learning algorithm known as the perceptron learning algorithm can be summarized by following steps more clearly:

Step-1: Initialize weights and bias randomly.

Step-2: For each training example, compute the output using the current weights and bias.

Step-3: Calculate the error as the difference between the predicted output and the actual output.

Step-4: Update the weights and bias to reduce the error, typically using the rule: $w_i \leftarrow w_i + \Delta w_i$

Where, $\Delta w_i = \eta(y - y\hat{})x_i$

Here, η is the learning rate, y is the true label, and $y\hat{}$ is the predicted output.

Step-5: Repeat this process iteratively until the error is minimized or a maximum number of iterations is reached.

Limitations of SLP

- The single layer perceptron can only solve problems where the classes are linearly separable. It cannot solve problems like XOR where the classes are not linearly separable
- It has no hidden layers, so it cannot capture complex patterns in data.

4.1.2.2. Multilayer Perceptron

A multilayer perceptron (MLP) is a type of artificial neural network that extends the single layer perceptron by including one or more hidden layers between the input and output layers. This architecture allows MLPs to learn and represent complex, non-linear relationships in data, overcoming the limitations of single layer perceptron which can only solve linearly separable problems.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

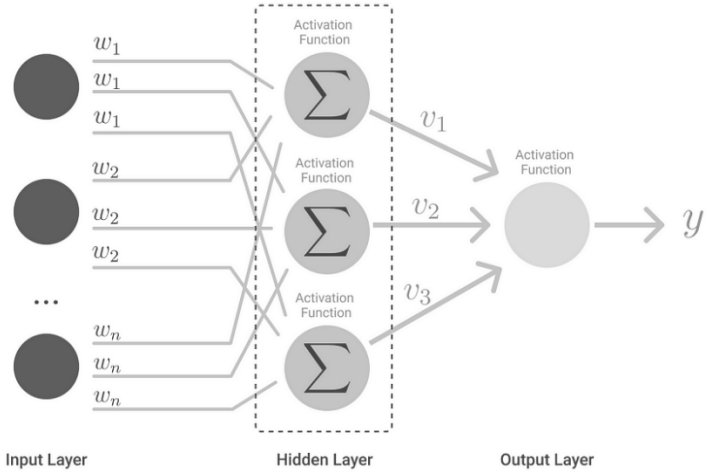


Figure: multilayer perceptron

- Multilayer Perceptron falls under the category of feedforward algorithms, because inputs are combined with the initial weights in a weighted sum and subjected to the activation function, just like in the Perceptron.
- But the difference is that each linear combination is propagated to the next layer.
- Each layer is feeding the next one with the result of their computation, their internal representation of the data. This goes all the way through the hidden layers to the output layer.

If the algorithm only computed the weighted sums in each neuron, propagated results to the output layer, and stopped there, it wouldn't be able to learn the weights that minimize the cost function. If the algorithm only computed one iteration, there would be no actual learning.

- Therefore, multi-layer perceptron is trained using **backpropagation**.
- A requirement for backpropagation is a differentiable activation function. That's because backpropagation uses **gradient descent** on this function to update the network weights.

4.1.2.3. Backpropagation

Backpropagation is short for "backward propagation of errors."

- In the context of backpropagation, SGD involves updating the network's parameters iteratively based on the gradients computed during each batch of training data. Instead of computing the gradients using the entire training dataset (which can be computationally

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

expensive for large datasets), SGD computes the gradients using small random subsets of the data called mini-batches.

Backpropagation is the learning mechanism that allows the Multilayer Perceptron to iteratively adjust the weights in the network, with the goal of minimizing the cost function.

- There is one hard requirement for backpropagation to work properly. The function that combines inputs and weights in a neuron, for instance the weighted sum, and the threshold function, for instance ReLU, must be differentiable.
- These functions must have a **bounded derivative**, because Gradient Descent is typically the optimization function used in Multilayer Perceptron.

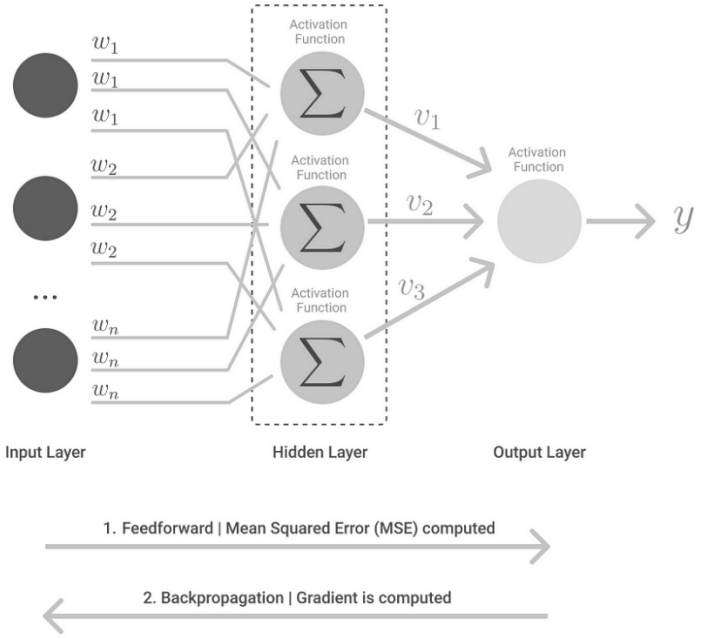
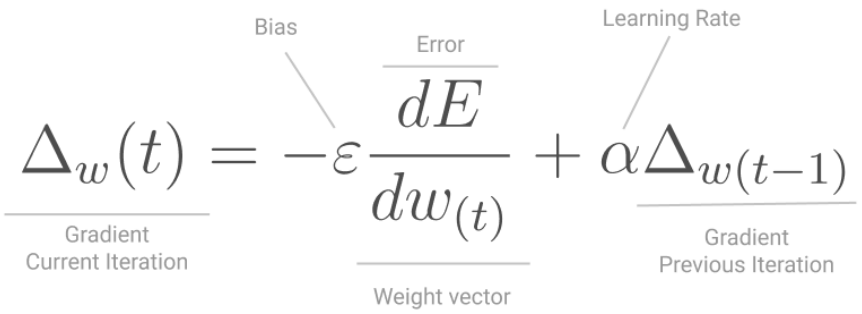


Figure: Multilayer Perceptron, with feedforward and Backpropagation

In each iteration, after the weighted sums are forwarded through all layers, the gradient of the **Mean Squared Error** is computed across all input and output pairs. Then, to propagate it back, the weights of the first hidden layer are updated with the value of the gradient. That's how the weights are propagated back to the starting point of the neural network!

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com



This equation is used in Stochastic Gradient Descent (SGD) with momentum for training a Multilayer Perceptron (MLP) using backpropagation.

Where, Δw (t) = Change in weight at time step t (this is what we use to update weights).

ϵ = Learning rate (a small positive constant that controls how much we adjust weights).

$\frac{dE}{dw(t)}$ = Derivative of the error function E with respect to the weights, this is the gradient (how the error changes with a change in weight).

α= Momentum term (usually between 0 and 1) — it helps to smooth out updates

Δw(t−1) = Change in weight from the previous iteration.

This process keeps going until gradient for each input-output pair has converged, meaning the newly computed gradient hasn't changed more than a specified *convergence threshold*, compared to the previous iteration.

How backpropagation algorithm works?

Step-1: Forward pass: During the forward pass, input data is fed into the neural network, and the network's output is computed layer by layer. Each neuron computes a weighted sum of its inputs, applies an activation function to the result, and passes the output to the neurons in the next layer.

Step-2: Loss computation: After the forward pass, the network's output is compared to the true target values, and a loss function is computed to measure the discrepancy between the predicted output and the actual output.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

Step-3: Backward pass (gradient calculation): In the backward pass, the gradients of the loss function with respect to the network's parameters (weights and biases) are computed using the chain rule of calculus. The gradients represent the rate of change of the loss function with respect to each parameter and provide information about how to adjust the parameters to decrease the loss.

Step-4: Parameter update: Once the gradients have been computed, the network's parameters are updated in the opposite direction of the gradients in order to minimize the loss function. This update is typically performed using an optimization algorithm such as stochastic gradient descent (SGD), that we discussed earlier.

Step-5: Iterative process: Steps 1-4 are repeated iteratively for a fixed number of epochs or until convergence criteria are met. During each iteration, the network's parameters are adjusted based on the gradients computed in the backward pass, gradually reducing the loss and improving the model's performance.

Note: *section 4.2. (sub section 4.2.1 (4.2.1.1, 4.2.1.2) are completely covered in SLP and MLP.*

4.2.2. Loss functions

Also called error function, is the degree of error in machine learning model's outputs.

- A loss function applies to a single training example and is part of the overall model's learning process that provides the signal by which the model's learning algorithm updates the weights and parameters

The cost function, sometimes called the objective function, is an **average of the loss function of an entire training set** containing several training examples

- It quantifies the difference ("loss") between a predicted value (i.e. model's output) for a given input and the actual value or ground truth.
- If a model's predictions are accurate, the loss is small. If its predictions are inaccurate, the loss is large.

In case of loss function ERM (empirical Risk Management) is an approach to selecting the optimal parameters of a machine learning algorithm that minimizes the empirical risk.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

4.2.2.1. Role of loss function

The role of the loss function is crucial in the training of machine learning models and includes the following:

- **Performance measurement**: Loss functions offer a clear metric to evaluate a model's performance by quantifying the difference between predictions and actual results.
- **Direction for improvement**: Loss functions guide model improvement by directing the algorithm to adjust parameters(weights) iteratively to reduce loss and improve predictions.
- **Balancing bias and variance**: Effective loss functions help balance model bias (oversimplification) and variance (overfitting), essential for the model's generalization to new data.
- **Influencing model behavior**: Certain loss functions can affect the model's behavior, such as being more robust against data outliers or prioritizing specific types of errors.

How Loss Function Works?

In a typical training setup, a model makes predictions on a batch of sample data points drawn from the training data set and a loss function measures the average error for each example. This information is then used to optimize model parameters.

Loss functions are specific to supervised learning, whose training tasks assume the existence of a correct answer: the *ground truth*. Conventional unsupervised learning algorithms, such as clustering or association, do not involve "right" or "wrong" answers, as they solely seek to discover intrinsic patterns in unlabeled data.

Types of Loss Functions:

Loss functions in machine learning can be categorized based on the machine learning tasks to which they are applicable. Most loss functions apply to regression and classification machine learning problems.

- In supervised learning, there are two main types of loss functions, these correlate to the 2 major types of neural networks: regression and classification loss functions

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

1. **Regression Loss Functions**: Used in regression neural networks; given an input value, the model predicts a corresponding output value (rather than pre-selected labels).

   Loss function includes,

   - ✓ Mean Squared Error (MSE) / L2 loss,
   - ✓ Mean Absolute Error (MAE) / L1 loss,
   - ✓ Huber loss / Smooth Mean Absolute Error.

2. **Classification Loss Functions:** Used in classification neural networks; given an input, the neural network produces a vector of probabilities of the input belonging to various pre-set categories can then select the category with the highest probability of belonging.

   Loss Function includes,

   - ✓ Binary Cross-Entropy loss / Log loss
   - ✓ Categorical Cross-Entropy,
   - ✓ Hinge loss,

4.2.2.2. Mean Squared Error (MSE)

The mean squared error loss function, also called **L2 loss** or **quadratic loss**, is generally the default for most regression algorithms.

- As its name suggests, MSE is calculated as the average of the squared differences between the predicted value and the true value across all training examples.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Where, n is the number of samples in the dataset

y_i is the predicted value for the i-th sample

$\bar{y}$ is the target value for the i-th sample

- Squaring the error means that the resulting value is always positive: as such, MSE evaluates only the magnitude of error and not its direction.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

- Squaring the error also gives large mistakes a disproportionately heavy impact on overall loss, which strongly punishes outliers and incentivizes the model to reduce them.
- The difference is squared, which means it does not matter whether the predicted value is above or below the target value; however, values with a large error are penalized.
- MSE is thus suitable when the target outputs are assumed to have a normal (Gaussian) distribution.

MSE is always differentiable, making it practical for optimizing regression models through gradient descent.

Mean squared logarithmic error (MSLE) *

For regression problems where the target outputs have a very wide range of potential values, such as those involving exponential growth, heavy penalization of large errors might be counterproductive.

$$\text{MSLE} = \frac{1}{n}\sum_{i=1}^{n}((\log(1+\hat{y}_i) - \log(1+y_i))^2$$

- Mean squared logarithmic error (MSLE) offsets this problem by averaging the squares of the natural logarithm of the differences between the predicted and average values.
- However, it's worth noting that MSLE assigns a greater penalty to predictions that are too low than to predictions that are too high.

Root mean squared error (RMSE)*

Root mean squared error is the square root of the MSE, which makes it closely related to the formula for standard deviations.

$$\text{RMSE} = \sqrt{\frac{\sum_i^n (y_i - \hat{y}_i)^2}{n}}$$

- RMSE thus largely mirrors the qualities of MSE in terms of sensitivity to outliers but is easier to interpret because it expresses loss in the same units as the output value itself.
- This benefit is somewhat tempered by the fact that calculating RSME requires another step compared to calculating MSE, which increases computation costs.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

Mean absolute error (MAE)*

Mean absolute error or *L1 loss*, measures the average absolute difference between the predicted value and actual value.

$$\text{MAE}= \frac{1}{n}\sum_{i=1}^{n} | (y_i - \hat{y}_i|$$

- Like MSE, MAE is always positive and doesn't distinguish between estimates that are too high or too low.
- It's calculated as the sum of the absolute value of all errors divided by the sample size

Because it doesn't square each loss value, MAE is more robust to outliers than MSE.

- MAE is thus ideal when the data might contain some extreme values that shouldn't overly impact the model.
- L1 loss also penalizes small errors more than L2 loss.

The MAE loss function is not differentiable in cases where the predicted output matches the actual output. Therefore, MAE requires more workaround steps during optimization.

4.2.2.3. Cross-Entropy Loss

In most cases, classification loss is calculated in terms of entropy. Entropy is a measure of uncertainty within a system.

For example, compare flipping coins to rolling dice: the former has lower entropy, as there are fewer potential outcomes in a coin flip (2) than in a dice toss (6).

- In supervised learning, model predictions are compared to the ground truth classifications provided by data labels. Those ground truth labels are certain and thus have low or no entropy. As such, we can measure loss in terms of the difference in certainty we'd have using the ground truth labels to the certainty of the labels predicted by the model.

The formula for cross-entropy loss (CEL) is derived from that of Kullback-Leibler divergence (KL divergence), which measures the difference between two probability distributions. Ultimately, minimizing loss entails minimizing the difference between the ground truth distribution of

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

probabilities assigned to each potential label and the relative probabilities for each label predicted by the model.

Binary cross-entropy (log loss) (BCEL)

Binary cross-entropy loss, also called log loss, is used for binary classification.

For example, in an email spam detection model, email inputs that result in outputs closer to 1 might be labeled "spam." Inputs yielding outputs closer to 0 would be classified as "not spam." An output of 0.5 would indicate maximum uncertainty or entropy.

- Though the algorithm will output values between 0 and 1, the ground truth values for the correct predictions are exactly "0" or "1."
- Minimizing binary cross-entropy loss thus entails not only penalizing incorrect predictions but also penalizing predictions with low certainty.
- This incentivizes the model to learn parameters that yield predictions that are not only correct but also confident.
- Furthermore, focusing on the logarithms of predicted likelihood values results in the algorithm more heavily penalizing predictions that are confidently wrong.

To maintain the common convention of lower loss values meaning less error, the result is multiplied by -1.

- Log loss for a single example i is thus calculated as: $(y_i * \log(p(y_i)) + (1-y_i) * \log(1-p(y_i)))$

where y_i is the true likelihood (either 0 or 1) and

$p(y_i)$ is the predicted likelihood.

Average loss across an entire set of n training examples is thus calculated as:

$$\frac{1}{n}\sum_{i=1}^{n}(yi \cdot \log(p(yi)) + (1 - yi) \cdot \log(1 - p(yi)))$$

Categorical cross-entropy loss (CCEL)

Categorical cross-entropy loss (CCEL) applies this same principle to multi-class classification.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

- A multi-class classification model will usually output a value for each potential class, representing the probability of an input belonging to each respective category (i.e. output predictions as a probability distribution).

Neural network classifiers typically use a **softmax** activation function for neurons in the output layer.

- Each output neuron's value is mapped to a number between 0 and 1, with the values collectively summing up to 1.

For example, in a data point containing only one potential category, the actual values for each prediction thus comprise "1" for the true class and "0" for each incorrect class.

- Minimizing CCEL entails increasing the output value for the correct class and decreasing output values for incorrect classes, thereby bringing the probability distribution closer to that of the ground truth. For each example, log loss must be calculated for each potential classification predicted by the model.

4.2.3. Regularization Techniques

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 are:

- Early stopping
- L1 and L2 regularization
- Data augmentation
- Addition of noise
- Dropout

4.2.3.1. Overfitting and underfitting

In neural networks**,**

Overfitting occurs when a model learns the training data too well, including noise and outliers, resulting in poor performance on unseen data.

Characteristics of Overfitting:

- Low training error, high validation/test error.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

- High accuracy on training data, low accuracy on new data.
- The model essentially memorizes the training data instead of learning the underlying patterns.

Causes of overfitting:

- Too complex model (too many layers or neurons).
- Insufficient training data.
- Training for too many epochs.
- Lack of regularization.

Solutions:

- **Regularization:** Techniques like L1 or L2 regularization add a penalty term to the loss function, discouraging overly complex models.
- **Dropout:** Randomly dropping out neurons during training prevents over-reliance on specific neurons.
- **Data Augmentation:** Increasing the amount of training data by creating slightly modified versions of existing data.
- **Early Stopping:** Monitoring the validation loss and stopping training when it starts to increase.
- **Cross-validation:** Splitting the data into multiple folds and training/validating on different combinations to get a more robust estimate of performance.

Underfitting, happens when the model is too simple to capture the underlying patterns in the data, leading to poor performance on both training and new data.

Characteristics:

- High training error, high validation/test error.
- Low accuracy on both training and new data.

Causes:

- Too simple model (not enough layers or neurons).
- Insufficient training epochs.
- Poor feature engineering.

Solutions:

- **Increase model complexity:** Add more layers or neurons.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

- **Increase training time:** Train for more epochs.
- **Improve feature engineering:** Ensure the model has relevant features to learn from.
- **Adjust learning rate:** Experiment with different learning rates to find the optimal one.
- **Use more appropriate algorithms:** Consider using more complex models if the problem requires it.

4.2.3.2. Regularization methods: L1, L2, Dropout, Batch Normalization

Dropout:

Dropout is a regularization technique commonly used in neural network architectures by randomly dropping one or several neurons inside a layer during the training process. This means that during training, the weights of dropped neurons won't be updated during the backpropagation process.

To implement Dropout, we first need to define a dropout probability for each layer. As an example, let's say we're setting a 50% dropout probability in a layer. This means that during training, each neuron in that layer has a 50% chance of being dropped. This process occurs independently for each training batch, meaning different subsets of data in the same iteration will most likely have different model architectures due to random neurons being dropped in each layer.

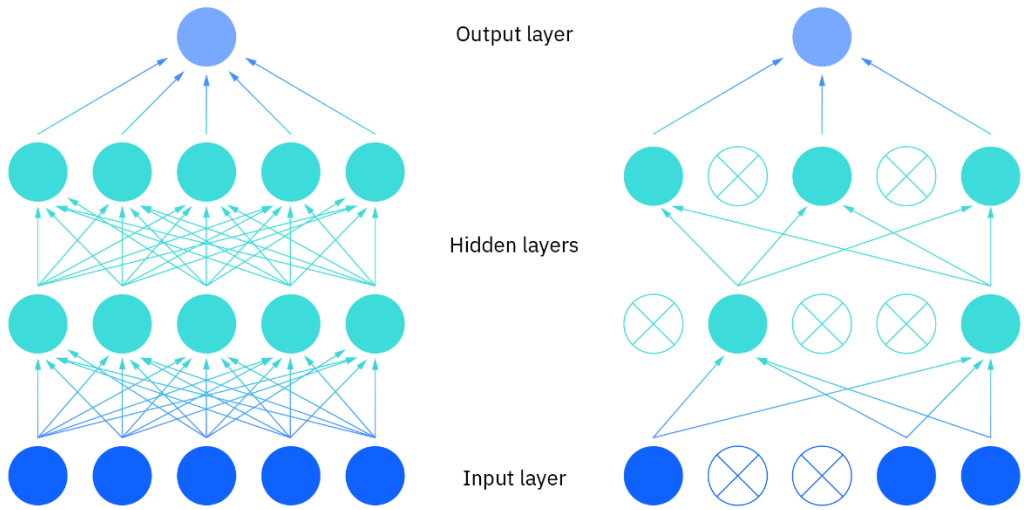


Figure: Architecture of a Neural Network before and after dropout

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

- This regularization method is quite destructive, but it's proven to be effective in improving a model's generalization on unseen data.
- One cause of overfitting is an overly complex model. and if we randomly drop neurons in each layer, we'll end up with a much simpler model.
- Also, if neurons are randomly dropped during training, other neurons need to step in and make necessary weight adjustments to be able to predict the training data correctly. This, in turn, will make the overall model less sensitive to specific weights of neurons in a particular layer, resulting in a model with better generalization capability.

IN TESTING, all neurons are active, but their outputs are scaled down to compensate for the absence of dropout.

Batch Normalization:

Batch Normalization helps to stabilize, reduction in risk of overfitting and speed up the learning process by normalizing inputs to their mean and standard deviation.

- It operates by calculating the mean and variance of the activations for each feature in the mini-batch and then normalizing the activations using these statistics.
- The normalized activations are then scaled and shifted using learnable parameters, allowing the model to adapt to the optimal activation distribution.

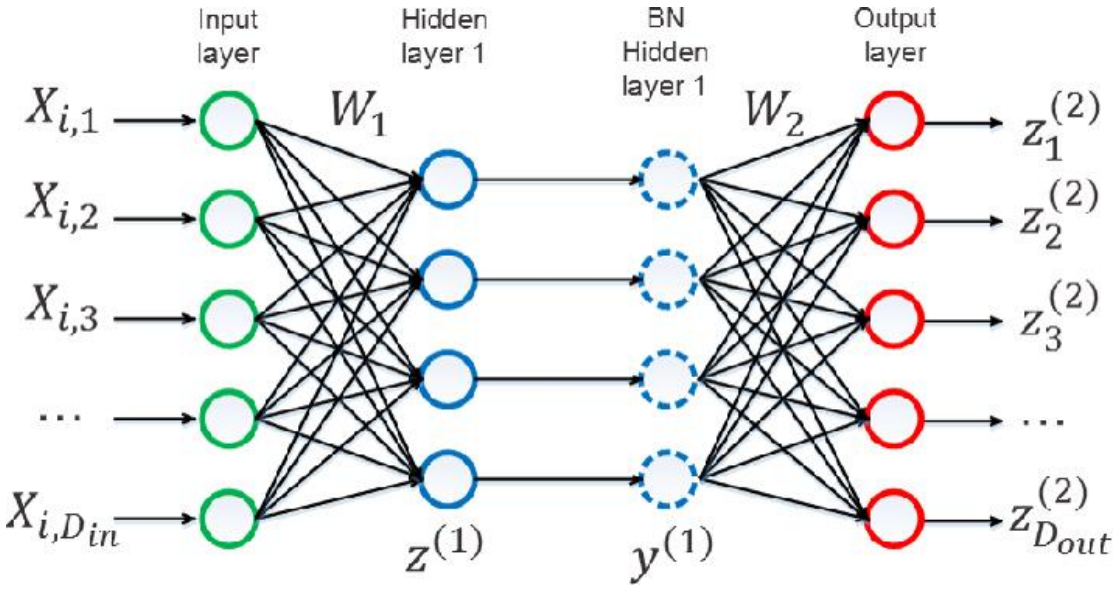


Figure: Neural Network (Batch Normalization)

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

Batch Normalization is typically applied after the linear transformation of a layer (e.g., after the matrix multiplication in a fully connected layer or after the convolution operation in a convolutional layer) and before the non-linear activation function (e.g., ReLU).

The key components of batch normalization are:

1. **Mini-batch statistics**: The mean and variance of the activations are calculated for each feature within the mini-batch.
2. **Normalization**: The activations are normalized by subtracting the mini-batch mean and dividing by the mini-batch standard deviation.
3. **Scaling and shifting**: Learnable parameters (γ and β) are introduced to scale and shift the normalized activations, allowing the model to learn the optimal activation distribution.

Generalization Mindmap

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

4.3. Advanced Neural Network Architecture

4.3.1. Convolution Neural Network (CNNs)

Neural networks are a subset of machine learning, and they are at the heart of deep learning algorithms.

- They are comprised of node layers, containing an input layer, one or more hidden layers, and an output layer.
- Each node connects to another and has an associated weight and threshold.
- If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

Convolutional neural networks are distinguished from other neural networks by their superior performance with image, speech or audio signal inputs.

- Convolutional neural networks (CNNs) are one of the main types of neural networks particularly well-suited for analyzing visual imagery (i.e. image recognition and classification) and other data with a grid-like topology.
- CNNs have several uses, some of which are object recognition, image processing, computer vision, and face recognition. Input for convolutional neural networks is provided through images.
- Convolutional neural networks are used to automatically learn a hierarchy of features that can then be utilized for classification, as opposed to manually creating features. In achieving this, a hierarchy of feature maps is constructed by iteratively convolving the input image with learned filters.
- Because of the hierarchical method, higher layers can learn more intricate features that are also distortion and translation invariant.
- CNNs excel at automatically extracting features from raw input data, making them powerful tools for tasks like image recognition, object detection, and classification.
- Common CNN architectures are: LeNet, AlexNet, VGGNet, GoogLeNet / Inception, ResNet, ZFNet.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

4.3.1.1. CNNs and their components

Convolutional Neural Networks (CNN, or ConvNet) are a type of multi-layer neural network that is meant to discern visual patterns from pixel images.

- In CNN, 'convolution' is referred to as the mathematical function.
- It's a type of linear operation in which you can multiply two functions to create a third function that expresses how one function's shape can be changed by the other.
- In simple terms, two images that are represented in the form of two matrices, are multiplied to provide an output that is used to extract information from the image.
- CNN is similar to other neural networks, but because they use a sequence of convolutional layers, they add a layer of complexity to the equation.
- CNN cannot function without convolutional layers.
- CNN can also be quite effective for classifying non-image data such as audio, time series, and signal data.

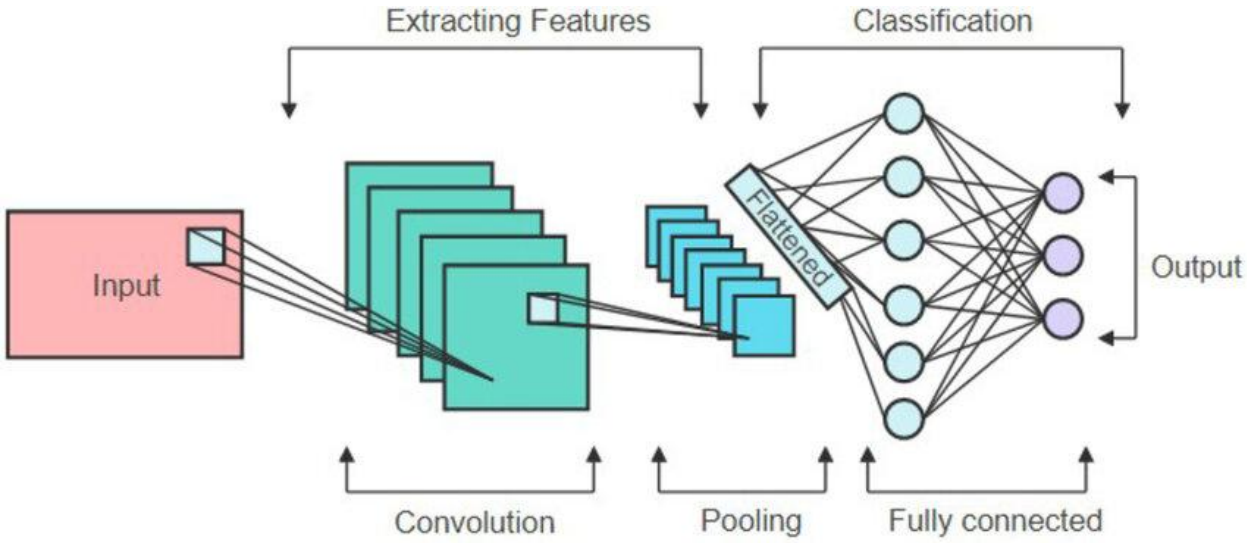


Figure: Architectural Diagram of CNN

CNN is made of three main components:

1. Convolutional layers
2. Pooling layers
3. Fully connected layers

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

4.3.1.2. Convolution, Pooling and fully connected layers

Convolution layer:

Convolution layer is the foundation of CNN, and they are in charge of executing convolution operations.

- The Kernel/Filter is the component in this layer that performs the convolution operation (matrix).
- Until the complete image is scanned, the kernel makes horizontal and vertical adjustments dependent on the stride rate.
- The kernel is less in size than a picture, but it has more depth (i.e. if the image has three (RGB) channels, the kernel height and width will be modest spatially, but the depth will span all three)

Simply, in the convolution layer, we use small grids (called filters or kernels or feature detectors) that move over the image. Each small grid is like a mini magnifying glass that looks for specific patterns in the photo, like lines, curves, or shapes. As it moves across the photo, it creates a new grid that highlights where it found these patterns.

For example, one filter might be good at finding straight lines, another might find curves, and so on. By using several different filters, the CNN can get a good idea of all the different patterns that make up the image.

Kernel or Filter or Feature Detectors

In a convolutional neural network, the kernel is nothing but **a filter that is used to extract the features from the images**.

The filters/kernels are smaller matrices usually 2x2, 3x3, or 5x5 shape

Formula for filter = [i-k] +1

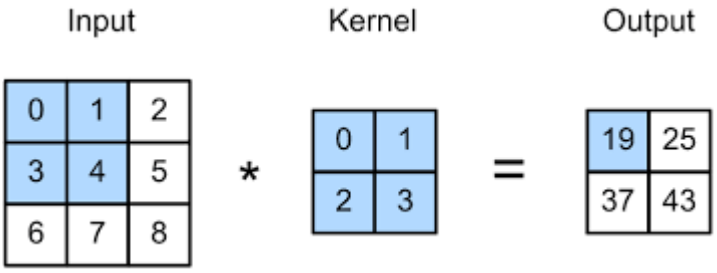
Where, i = Size of input and K= Size of kernel

What is Convolution Operation?

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

The convolution operation involves multiplying **the kernel value**s by the **original pixel values** of the image and then **summing up the results**.

This is a basic example with a 2 × 2 kernel:



Let's start in the left corner of the input: (0 × 0) + (1 × 1) + (3 × 2) + (4 × 3) = 19

Then slice one pixel to the right and perform the same operation: (1 × 0) + (2 × 1) + (4 × 2) + (5 × 3) = 25

After completed the first row we move one pixel down and start again from the left: (3 × 0) + (4 × 1) + (6 × 2) + (7 × 3) = 37

Finally, again slice one pixel to the right: (4 × 0) + (5 × 1) + (7 × 2) + (8 × 3) = 43

The output matrix of this process is known as the **Feature map.**

What is the Depth of Layer?

Depth is the number of kernels it contains. Each filter produces a separate feature map, and the collection of these feature maps forms the complete output of the layer.

Components of Convolution layer:

There are three essential components inside convolutional layers: ***Channels, Stride and Padding***

These components address the following.

1. Kernels always traverse through the image matrix one pixel at a time?
2. What happens with the pixels in the corners, we are only passing over them one time, what if they have an important feature?

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

3. And what about RGB images? We stated that they are represented in 3 dimensions, how does the kernel traverse over them?

Channels:

Each channel will detect a different feature in the image based on the values you assign to its kernel. For an RGB image, there are typically separate kernels for each color channel because different features might be more visible or relevant in one channel compared to the others.

Stride:

Stride refers to the number of pixels by which a kernel moves across the input image. In convolution layer a kernel moves through the pixel of an image in different way.

- If the stride=1, that means kernel moves across one pixel at a time (e.g. if input matrix is 5×5 then o/p matrix will be 3×3 as stride is 1)

Similarly,

- If the stride=2, that means kernel moves across two pixel at a time (e.g. if input matrix is 5×5 then o/p matrix will be 2×2 as stride is 2)

Therefore, A larger stride will produce smaller output dimensions (as it covers the input image faster), whereas a smaller stride results in a larger output dimension.

Why to change the stride?

- Increasing the stride will allow the filter to cover a larger area of the input image, which can be useful for capturing more global features.
- In contrast, lowering the stride will capture finer and more local details.
- In addition, increasing the stride will control overfitting and reduce computational efficiency as it will reduce the spatial dimensions of the feature map.

$$\text{Stride}=\frac{Input\ size-\text{Kernel size}}{Stride}+1$$

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

Padding:

Padding refers to the addition of extra pixels around the edge of the input image. If focus on the pixels in the image's edges, you'll notice that we traverse them fewer times compared to those positioned in the center.

- The purpose of padding is to adjust the spatial size of the output of a convolutional operation and to preserve spatial information at the borders.

When to use padding?

When the edges of the image contain useful information that you want to capture, use padding.

- Padding can be increased up to the size of kernel using.

How does padding affects the output field?

Padding increases the size of the output feature map.

- If you increase the padding while keeping the kernel size and stride constant, the convolution operation has more "room" to take place, resulting in a larger output.

Finally,

Output size of a convolutional layer can be calculated using the following formula:

$$\text{Output size} = \frac{Input\ size + 2 \times \text{Padding} - \text{Kernel size}}{Stride} + 1$$

Where, $2 \times$ Padding → accounts for padding applied to both the left and right sides (or top and bottom sides) of the input.

$+ 1$ → accounts for the initial position of the filter, which starts at the beginning of the padded input.

Conclusion: The main objective of Convolutional Layers is Feature Extraction although Convolutional Layers can decrease the output size, their principal objective is not Dimensionality Reduction.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

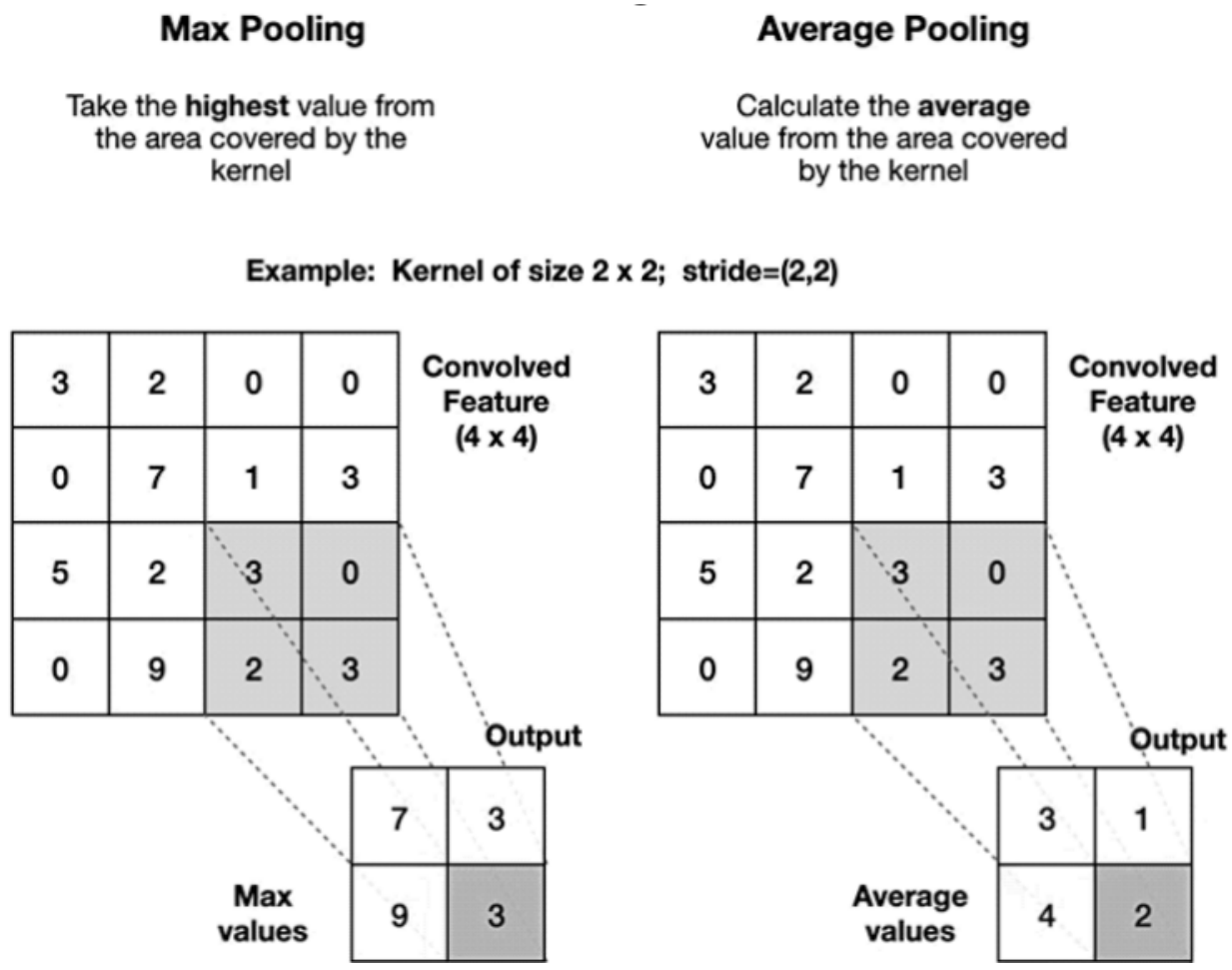
Pooling Layer

Pooling layer's main objective is indeed dimensionality reduction.

Working mechanism of Pooling Layer:

Suppose you have a large image and want to make it smaller but keep all the important features like edges and colors.

The pooling layer operates independently on every depth slice of the input. It resizes it spatially, using the **Max** or **Average** of the values in a window slid over the input data.



In this example, feature map has been reduced from (4 × 4) to (2 × 2).

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

What is the difference between pooling and the convolution operation?

In pooling, any kernel is not applied to the input data, just simplify the information with a math operation (Max or Avg).

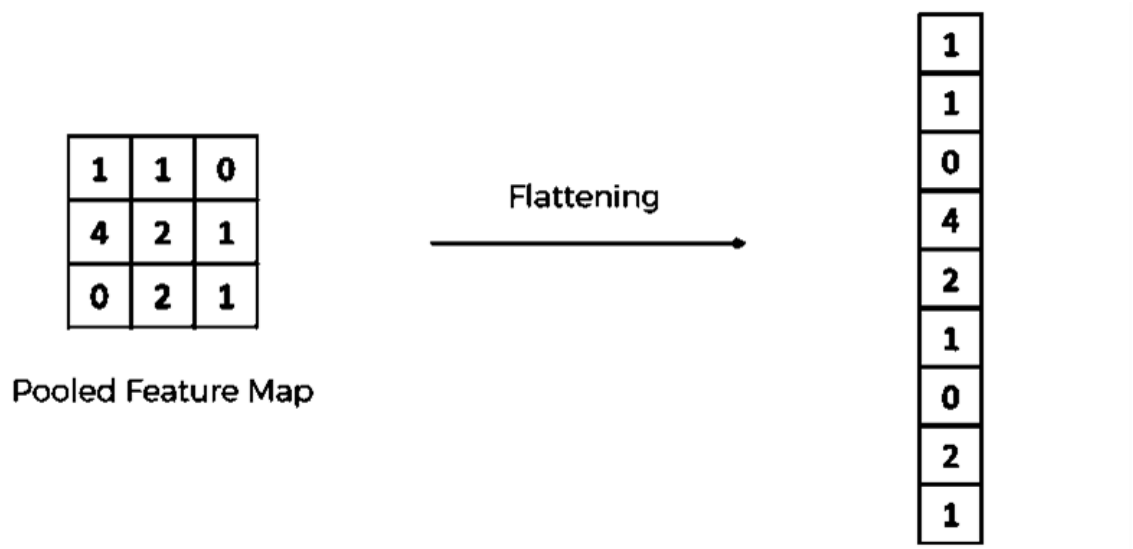
- Pooling layers do not reduce the number of channels (i.e. Channels is not reduced until the end of the architecture)
- Each pooling operation is applied independently to each channel of the input data. With Convolutional and Pooling layers, we can't reduce the number of channels, just add more to the existing ones.

After convolutional and pooling layers have extracted relevant features from the input image, we have to turn this high-dimensional feature map into a format suitable for feeding into fully connected layers. Here´s where flattening layers come into action

Flattening layers

Suppose you have a grid of data (like pixels in a feature map), and you want to line up all of these grid points in a single, long line. That's what flattening does.

- It takes the entire feature map and reorganizes it into a single, long vector.
- Flattening changes the shape of the data, it does not make any changes to the actual information.



By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

Why do we need flattening Layers?

- **Integration of features**
  By flattening the feature maps into a vector, the network can integrate the spatially distributed features extracted for tasks such as classification.
- **Compatibility with Dense Layers**
  Fully connected layers (dense layers) are designed to operate on 1-dimensional data, hence, flattening is a necessary step to transition from the multidimensional tensors produced by convolutional layers to the format required for dense layers.

Fully Connected Layers (Dense Layers)

While convolutional layers are good at detecting features in input data, dense layers are essential for integrating these features into predictions.

- Without dense layers, CNNs would not be able to perform the high-level tasks that are often required, such as classifying images, detecting objects, or making predictions based on visual inputs.

Final Note: The pooling and flattening layers DON´T have an activation function (i.e. Activation function comes under Convolution Layer)

4.3.1.3. Applications in image processing and computer vision

Image processing in the context of CNN refers to the analysis, manipulation, or extraction of information from image data using techniques like convolutional neural networks.

Application of CNN in image processing:

- Image classification
- Object detection and localization
- Image segmentation
- Image retrieval
- Image denoising / restoration
- Image super-resolution
- Facial recognition

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

- Pose estimation
- OCR (optical Character recognition)
- Scene understanding
- Recommender engine
- Drug discovery using predictive analytics
- Content based image retrieval
- Visual Question Answering (VQA)
- Image captioning
- Medical image analysis, computing, predictive analysis, healthcare data science.
- Image tagging: image tag helps to describe the image and makes them easier to find.it is fundamental elements of visual search. Used by Facebook, google, amazon.

4.3.2. Recurrent Neural Networks (RNNs)

RNNs are a type of neural network that can model sequence data.

- RNNs, which are formed from feedforward networks, are similar to human brains in their behavior.
- RNN can anticipate sequential data in a way that other algorithms can't.

A Recurrent Neural Network (RNN) is a type of artificial neural network designed to process sequential data where the order matters, like sentences, time series, or audio. Unlike regular neural networks, RNNs have a "memory" that helps them remember previous inputs, making them ideal for tasks where context is important.

4.3.2.1. Basics of RNNs

Basic Architecture and Working Principle of Standard RNNs

RNNs are designed to process sequential data by maintaining a hidden state that captures information about previous inputs.

- The basic architecture consists of an input layer, a hidden layer, and an output layer.
- Unlike feedforward neural networks, RNNs have recurrent connections, as shown in figure below, allowing information to cycle within the networks.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

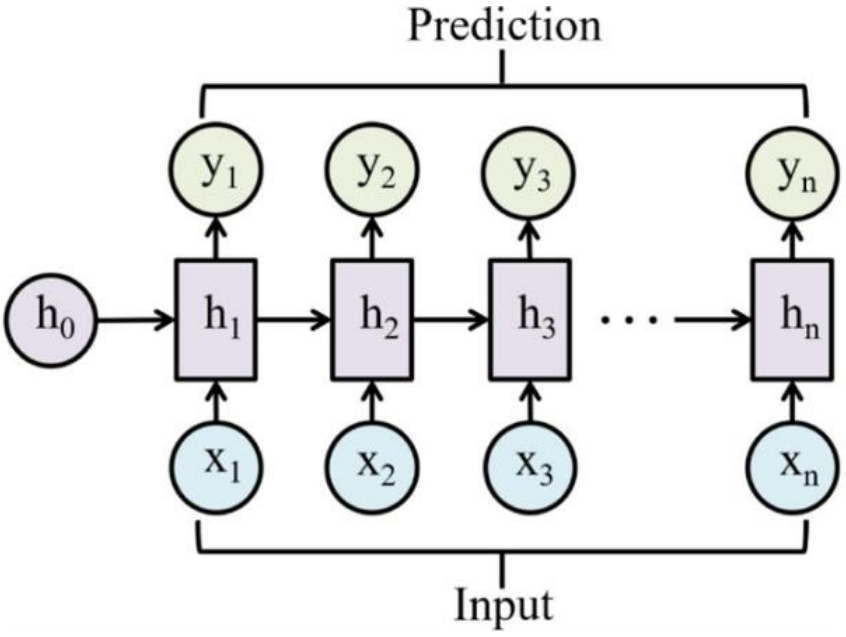


Figure: Architecture of RNN

At each time step, *t*, the RNN takes an input vector, x*t*, and updates its hidden state, h*t*, using the following equation:

$$h_t = \sigma_h\ (W_{xh}\ x_t\ +\ W_{hh}\ h_{t-1}\ +\ b_h\ )$$

where W_{hy} is the weight matrix between the input and hidden layer,

W_{hh} is the weight matrix for the recurrent connection,

h_h is the bias vector, and

σ_h is the activation function, typically the hyperbolic tangent function (tanh) or the rectified linear unit.

The output at each time step, *t*, is given by

$$y_t = \sigma_y\ (W_{hy}\ h_t\ +\ b_y\ )$$

where W_{hy} is the weight matrix between the hidden and output layers,

b_y is the bias vector, and

σ_y is the activation function for the output layer.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

Activation Function in RNNs.

The core of RNN operations involves the recurrent computation of the hidden state, which integrates the current input with the previous hidden state.

- This recurrent computation allows RNNs to exhibit dynamic temporal behavior.
- The choice of activation function plays a crucial role in the behavior of the network, introducing non-linearity that enables the network to learn and represent complex patterns in the data.
- One commonly used activation function in RNNs is the ***hyperbolic tangent (tanh)***.
- The tanh function squashes the input values to the range of [-1, 1], making it zero-centered and suitable for modeling sequences with both positive and negative values.

The tanh is represented mathematically as follows:

$$\sigma_h\ (z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Similarly, another widely used activation function are the rectified linear unit (ReLU), ELU, Sigmoid, Softmax etc. (for details visit: https://www.mdpi.com/2078-2489/15/9/517)

Applications of RNNs

- Language Translation: Translating sentences from one language to another.
- Speech Recognition: Converting spoken words into text (e.g., Siri, Google Voice).
- Time-Series Prediction: Forecasting stock prices or weather.
- Text Generation: Creating new text based on learned patterns.
- Sentiment Analysis: Understanding emotions in text (e.g., positive or negative reviews)
- Stock market recommendation, anomaly detection, video captioning etc.

Advantages of RNNs

- Handles Sequential Data: Remembers previous inputs, making it great for time-dependent tasks.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

- Flexible Input/Output Lengths: Can process sequences of varying lengths.
- Parameter Sharing: Uses the same weights for each time step, making it efficient

Disadvantages and Limitations of RNNs

- Vanishing Gradient Problem: When training, the "memory" of earlier steps can fade, making it hard to learn long-term dependencies.
- Exploding Gradient Problem: Sometimes, the training process can become unstable if the gradients get too large.
- Slow Training: Because RNNs process data step by step, they are slower to train than some other models.
- Limited Long-Term Memory: Standard RNNs struggle to remember information from many steps back in a sequence.
- Difficult Parallelization: The sequential nature makes it hard to speed up training using modern hardware

Vanishing and Exploding Gradient Problems in RNN

Training RNNs presents significant challenges due to the vanishing and exploding gradient problems. During the training process, the BPTT (Back Propagation through Time) algorithm is used to compute the gradients of the loss function with respect to the weights.

As the gradients are propagated backwards in time, they can either diminish **(vanish**) or grow exponentially **(explode),** making it difficult for the network to learn long-term dependencies or causing instability during training.

Mathematically, the hidden state at time step *t* can be expanded as follows:

$$h_t = \sigma_h\ (W_{xh}\ x_t\ +\ W_{hh}\ \sigma_h\ (W_{xh}\ x_{t-1}\ +\ W_{hh}\ h_{t-2}\ +b_h\ )+b_h\ )$$

When calculating the gradients, we encounter terms involving the product of many Jacobian matrices:

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-n}} = \prod_{k=t-n}^{t-1} \mathbf{J}_k,$$

where $\mathbf{J}_k$ is the Jacobian matrix of the hidden state at time step k.

- If the eigenvalues of $\mathbf{J}_k$ are less than 1, the product of these matrices will tend to zero as n increases, leading to vanishing gradients.
- Conversely, if the eigenvalues of $\mathbf{J}k$ are greater than 1, the gradients can grow exponentially, leading to exploding gradients, which can cause the model parameters to become unstable and result in numerical overflow during training.
- The vanishing gradient problem prevents the network from effectively learning long-term dependencies, as the gradient signal becomes too weak to update the weights meaningfully for earlier layers.
- On the other hand, the exploding gradient problem can cause the model to converge too quickly to a poor local minimum or make the training process fail entirely due to excessively large updates.

To mitigate these problems, various RNN variants have been developed, such as

1. LSTM and
2. GRUs.

These architectures introduce gating mechanisms that regulate the flow of information and gradients through the network, allowing for the better handling of long-term dependencies.

Additionally, gradient clipping is a common technique used to prevent exploding gradients by capping the gradients at a maximum threshold during backpropagation, ensuring that they do not grow uncontrollably

4.3.2.2. Long Short-Term Memory (LSTM)

LSTM is the advanced type of RNN.

- LSTM networks were introduced to address the vanishing gradient problem inherent to basic RNNs.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

- The key innovation in LSTM is the use of gating mechanisms to control the flow of information through the network. This allows LSTM networks to maintain and update their internal state over long periods, making them effective for tasks requiring the modeling of long-term dependencies.
- Each LSTM cell contains ***three gates: the input gate, forget gate, and output gate***, which regulate the cell state $\mathbf{c}_t$ and hidden state $\mathbf{h}_t$.
- These gates determine how much of the input to consider, how much of the previous state to forget, and how much of the cell state to output.

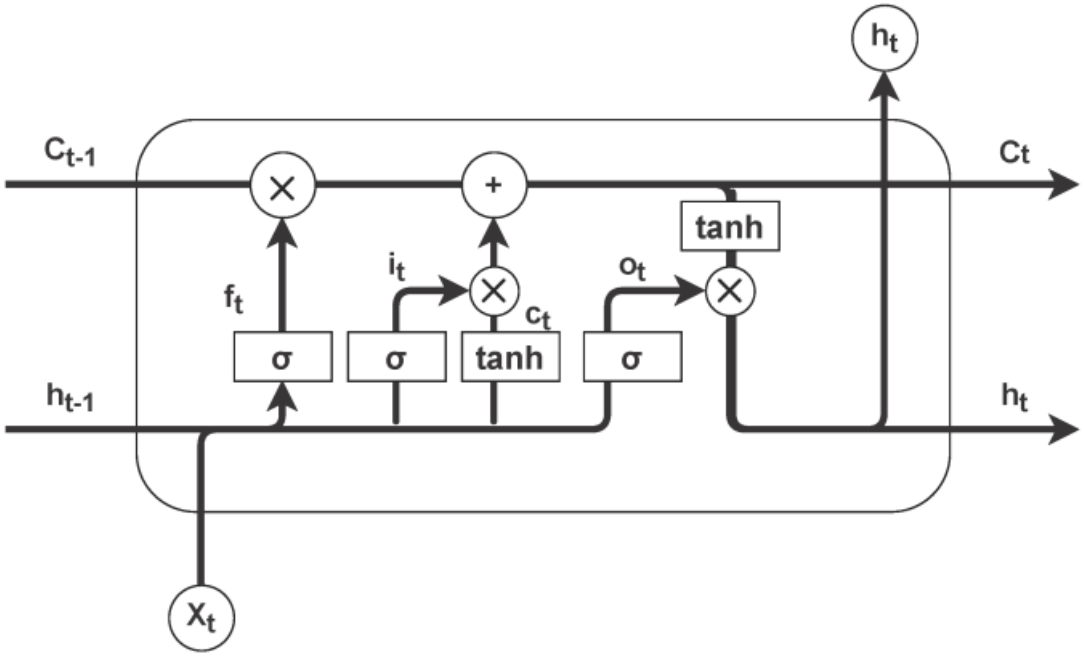


Figure: Architecture of the LSTM network

Mathematically, LSTM Updates the equations are as follows:

$$i_t = \sigma\,(W_{xi}\,x_t + W_{hi}\,h_{t-1} + b_i)$$

$$f_t = \sigma\,(W_{xf}\,x_t + W_{hf}\,h_{t-1} + b_f)$$

$$o_t = \sigma\,(W_{xo}\,x_t + W_{ho}\,h_{t-1} + b_o)$$

$$g_t = \tanh\,(W_{xg}\,x_t + W_{hg}\,h_{t-1} + b_g)$$

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

$$c_t = (f_t * c_{t-1} + i_t * g_t)$$

And $h_t = o_t * \tanh(c_t)$

where $\mathbf{i}_t$ is the input gate,

$\mathbf{f}_t$ is the forget gate,

$\mathbf{o}_t$ is the output gate,

$\mathbf{g}_t$ is the cell input,

$\mathbf{c}_t$ is the cell state,

$\mathbf{h}_t$ is the hidden state,

σ represents the sigmoid function,

tanh is the hyperbolic tangent function

Architecture of LSTM cell, effectively manages long-term dependencies in sequence data by employing three crucial gating mechanisms. Each of these gates plays a distinct role in controlling the flow of information through the cell.

1. Input gate ($\mathbf{i}_t$): Controls how much of the new input $\mathbf{x}_t$ is written to the cell state $\mathbf{c}_t$

2. Forget gate ($\mathbf{f}_t$): Decides how much of the previous cell state $\mathbf{c}t$-1 should be retained

3. Output gate ($\mathbf{o}_t$): Determines how much of the cell state $\mathbf{c}_t$ is used to compute the hidden state $\mathbf{h}_t$

The cell input $\mathbf{g}_t$ is a candidate value that is added to the cell state after being modulated via the input gate. The use of these gating mechanisms allows LSTM networks to selectively remember or forget information, enabling them to handle long-term dependencies more effectively than traditional RNNs

The internal recurrence within the LSTM cell is managed through the cell state $\mathbf{c}_t$, which acts as a conveyor belt, transferring relevant information across different time steps. This recurrence

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

mechanism allows the LSTM to maintain and update its memory over long sequences, effectively capturing long-term dependencies.

Additionally, the element-wise multiplication operations between the gates and their respective inputs ensure that the interactions between different components of the LSTM are smooth and efficient. This enables the LSTM to perform complex transformations on the input data while maintaining the stability of the learning process.

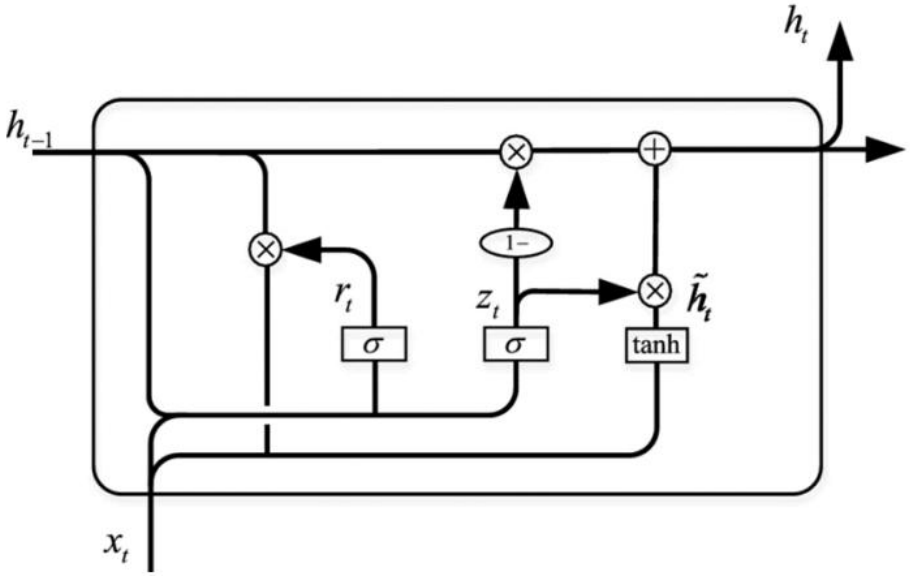
Meanwhile, LSTM networks utilize internal recurrence within each cell to manage long-term dependencies, with the recurrence happening through the cell state as information is passed from one time step to the next.

Other LSTM variants include bidirectional LSTM (BiLSTM) and stacked LSTM.

4.3.2.3. Gated Recurrent Units (GRU)

Gated recurrent units are another variant designed to address the vanishing gradient problem while simplifying the LSTM architecture.

- GRUs combine the forget and input gates into a single update gate and merge the cell state and hidden state, reducing the number of gates and parameters and thus simplifying the model and making it computationally more efficient.
- The GRU architecture consists of two gates: the update gate, $\mathbf{z}_t$, and the reset gate, $\mathbf{r}_t$



By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

Figure: Architecture of RGU network

- The gates control the flow of information to ensure that relevant information is retained and irrelevant information is discarded.
- Similar to LSTM, GRUs rely on internal recurrence within each unit as they maintain and update the hidden state across time steps to capture temporal dependencies.

The updated equations for GRUs are as follows:

$$z_t = \sigma\ (W_{xz}\ x_t\ +\ W_{hz}\ h_{t-1}\ +b_z\ )$$

$$r_t = \sigma\ (W_{xr}\ x_t\ +\ W_{hr}\ h_{t-1}\ +b_r\ )$$

$$h'_t\ = \tanh(\ W_{xh}\ x_t\ +\ r_t\ *(W_{hh}\ h_{t-1}\ ) + b_h$$

$$h_t = ((1 - z_t\ ) * (\ h_{t-1}\ + z_t\ * h'_t\ )$$

where $\mathbf{z}_t$ is the update gate,

$\mathbf{r}_t$ is the reset gate, and

$\mathbf{h}'_t$ is the candidate hidden state.

The update gate $\mathbf{z}_t$ determines how much of the previous hidden state $\mathbf{h}_{t-1}$ should be carried forward to the current hidden state $\mathbf{h}_t$, while the reset gate, $\mathbf{r}_t$, controls how much of the previous hidden state to forget

The candidate hidden state $\mathbf{h}'_t$ represents the new content to be added to the current hidden state, modulated via the reset gate.

- Furthermore, the simplified architecture of GRUs allows them to be computationally more efficient than LSTM while still addressing the vanishing gradient problem.
- This efficiency makes GRUs well-suited for tasks where computational resources are limited or when training needs to be faster.
- GRUs have been successfully applied in various sequence modeling tasks. Their ability to capture long-term dependencies with fewer parameters makes them a popular choice in many applications.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

- Additionally, studies have shown that GRUs can achieve performance comparable to LSTM, making them an attractive alternative for many use cases.

GRU vs LSTM

GRUs have fewer parameters compared to LSTM due to the absence of a separate cell state and combined gating mechanisms.

- This often leads to faster training times and comparable performance to LSTM in many tasks.
- However, despite their advantages, the choice between GRUs and LSTM often depends on the specific task and dataset.
- Some tasks may benefit more from the additional complexity and gating mechanisms of LSTM, while others may perform equally well with the simpler GRU architecture.

4.3.2.4. Applications in time-series prediction

Recurrent Neural Networks (RNNs) are powerful tools for time series analysis and forecasting due to their ability to process sequential data and retain memory of past inputs. They are widely used in various applications, including followings:

- **Financial Forecasting:**

  Predicting stock prices, currency exchange rates, and other financial variables based on historical market data.

- **Weather Forecasting:**

  Predicting temperature, rainfall, and other weather parameters based on historical weather data.

- **Energy Demand Prediction:**

  Anticipating energy consumption patterns to optimize energy resource allocation and distribution.

- **Anomaly Detection:**

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com

Identifying unusual patterns or outliers in time series data, which can be useful in various applications such as fraud detection or predictive maintenance.

- **Sales Forecasting:**

  Predicting future sales based on historical sales data and other relevant factors.

- **Healthcare:**

  Analyzing patient data over time to predict potential health issues or optimize treatment plans.

Example:

Suppose you are predicting the daily temperature using an RNN. The network would analyze historical temperature data, along with other relevant factors like humidity and wind speed, to learn the patterns that influence temperature changes. The RNN would then use this learned knowledge to predict future temperatures.

By: Er. Mukunda Paudel, *M.E. Computer*
Email: paudelmuku@gmail.com