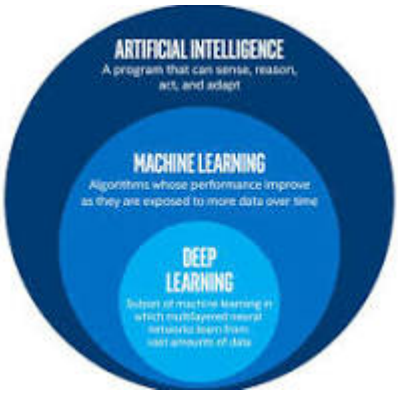
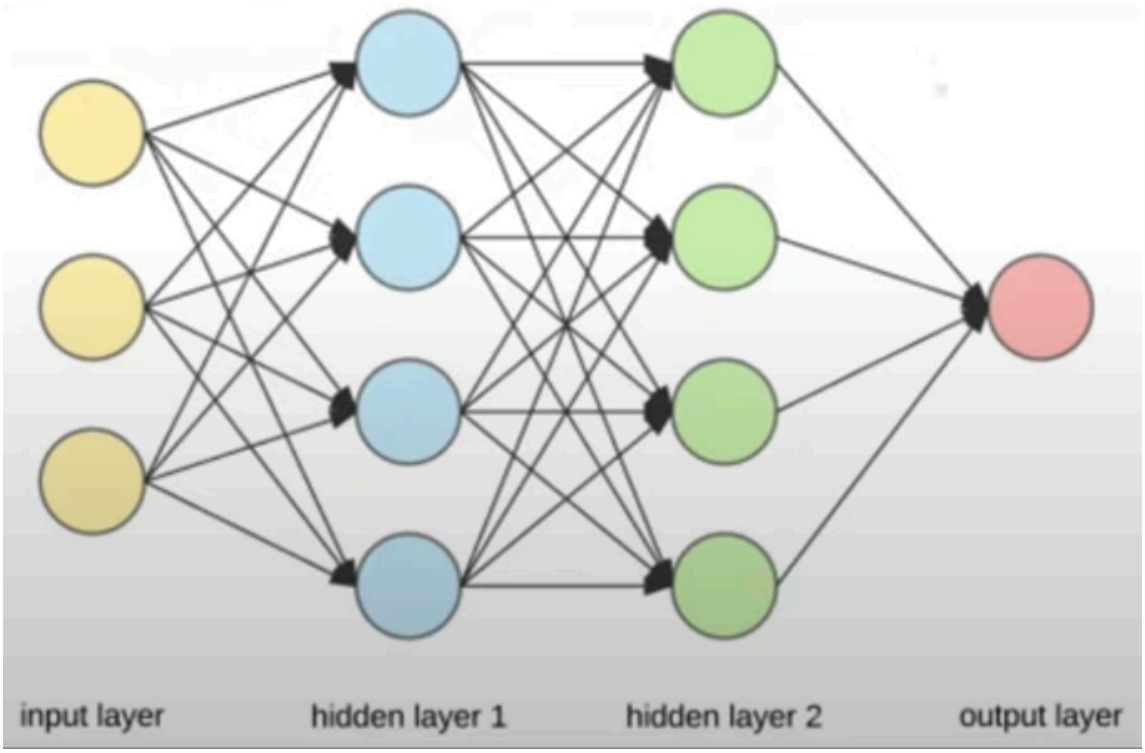
What is Deep Learning?

Deep Learning is a subfield of Artificial Intelligence and Machine Learning that is inspired by the structure of a human brain.

Deep learning algorithms attempt to draw similar conclusions as humans would by continually analyzing data with a given logical structure called Neural Network.





Why is it getting so famous?

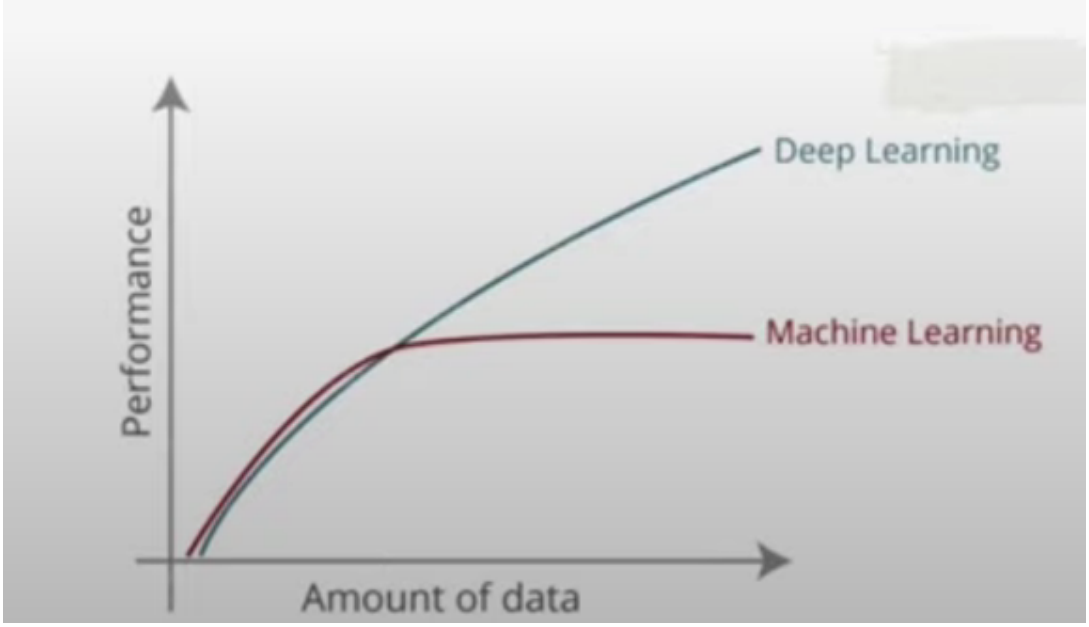
- Applicability: It has a broad range of applications across various fields and industries. Its versatility allows it to be used in diverse scenarios, making it a valuable tool for many professionals and organizations.
- Performance: It delivers high performance, often exceeding expectations. Its efficiency and effectiveness in accomplishing tasks make it a preferred choice, contributing to its growing popularity.

DEFINE

- Deep learning is a subfield of machine learning that uses artificial neural networks with representation learning. Here's a bit more detail on how it works:
- Deep learning algorithms use multiple layers of processing units (artificial neurons) to progressively extract higher-level features from raw input data.
- In image recognition, for example, the lower layers of a deep learning network might identify edges and corners, while higher layers might recognize faces, objects, and scenes.
- This ability to learn complex representations from data is what makes deep learning so powerful. It has led to breakthroughs in a wide range of fields, including computer vision, speech recognition, natural language processing, and more.

DIFFERENCE BETWEEN ML AND DL

- DATA



- HARDWARE DEPEND (GPU)
- TRANING TIME (HIGH)
- FEATURE SELECTION (REPRESENTATION LEARNING)
- INTERPRETABILITY (BLACK BOX)

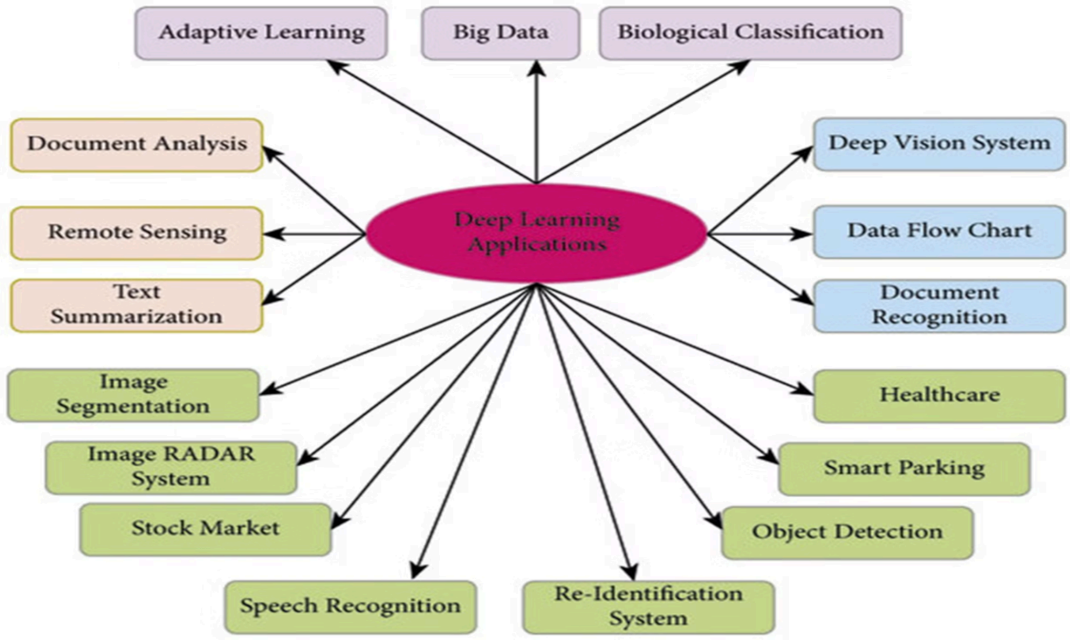
WHY DL FAMOUS NOW?

- Datasets: The availability of large, high-quality datasets allows for better training of deep learning models. These datasets provide the necessary data for models to learn patterns and make accurate predictions.
- Frameworks: Robust and user-friendly deep learning frameworks (such as TensorFlow, PyTorch, and Keras) have made it easier for developers and researchers to design, implement, and experiment with deep learning models.
- Community: A growing and active community of researchers, developers, and enthusiasts supports the rapid advancement of deep learning. This community shares knowledge, tools, and best practices, fostering innovation and collaboration.
- Architecture: Advances in neural network architectures, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), have significantly improved the performance of deep learning models on various tasks.
- Hardware: Improvements in hardware, particularly GPUs and TPUs, have made it possible to train complex deep learning models faster and more efficiently. This increased computational power has enabled the handling of larger datasets and more sophisticated models.

TYPE OF NEURAL NETWORK

- ANN
- CNN
- RNN
- GAN
- AUTO ENCODER

APPLICATION OF DL



Deep learning has a wide range of applications across various fields. Here are some key areas where deep learning is making a significant impact:

1.Computer Vision:

- Image and Video Recognition: Used in social media, security, and autonomous vehicles.
- Medical Imaging: Assists in diagnosing diseases from X-rays, MRIs, and CT scans.
- Facial Recognition: Utilized for security and authentication purposes.
- Natural Language Processing (NLP):

2. Language Translation: Enables real-time translation services.

- Sentiment Analysis: Helps in understanding customer feedback and social media sentiments.
- Chatbots and Virtual Assistants: Powers systems like Siri, Alexa, and customer service bots.

3. Speech Recognition:

- Voice Assistants: Improves the accuracy of systems like Google Assistant and Cortana. Transcription Services: Converts speech to text for various applications. Healthcare:

4. Drug Discovery: Accelerates the process of finding new drugs.

- Personalized Medicine: Tailors treatments based on individual genetic information.
- Predictive Analytics: Forecasts disease outbreaks and patient outcomes.

5. Finance:

- Fraud Detection: Identifies fraudulent transactions and activities.
- Algorithmic Trading: Enhances trading strategies based on deep learning models.
- Risk Management: Assesses and mitigates financial risks. Autonomous Vehicles:

6. Self-Driving Cars: Enables cars to navigate and make decisions without human intervention.

- Advanced Driver Assistance Systems (ADAS): Provides features like lane detection and collision avoidance.

7. Gaming:

- Game Development: Creates more realistic and intelligent game characters.
- Player Behavior Analysis: Understands and predicts player actions and preferences.

8. Robotics:

- Automation: Enhances the capabilities of industrial and service robots.
- Navigation: Helps robots understand and navigate their environments.

9. Agriculture:

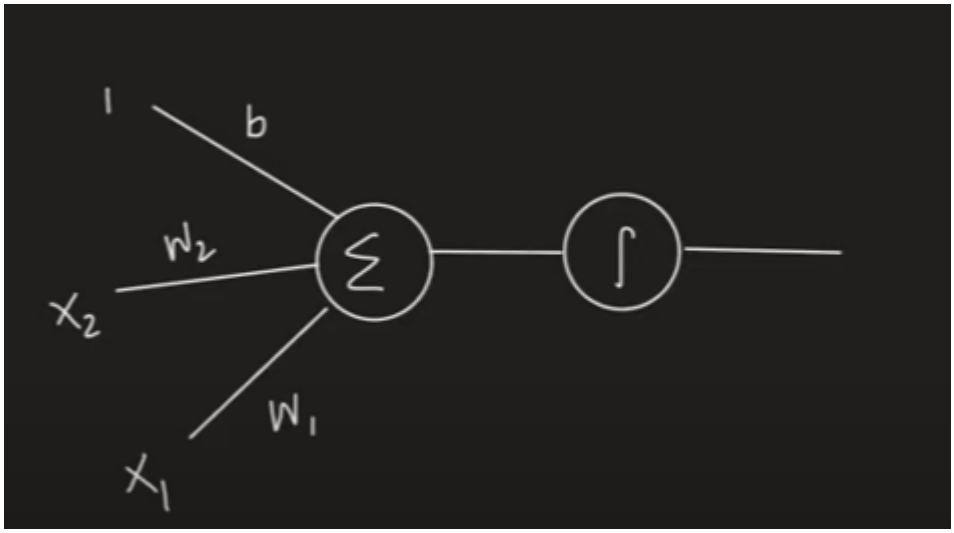
- Crop Monitoring: Analyzes satellite images to monitor crop health.

- Precision Farming: Optimizes planting, watering, and harvesting based on data analysis.

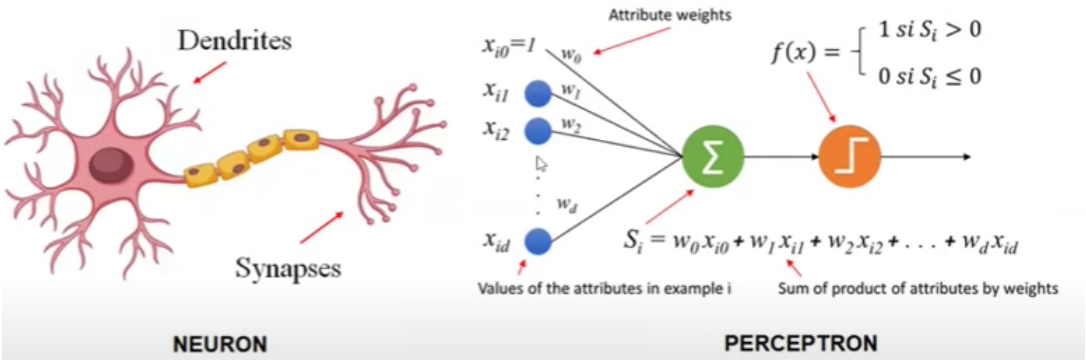
10. Marketing and Advertising:

- Customer Segmentation: Identifies target audiences for personalized marketing.
- Ad Placement: Optimizes the placement of ads for better engagement and conversion rates.

WHAT IS PERCEPTRON?

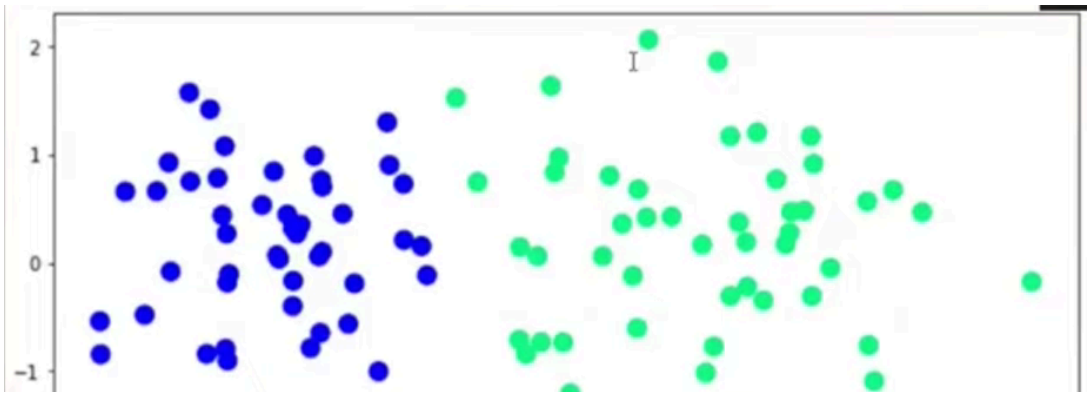


- DIFF. BETWEEN NEURON AND PERCEPTRON



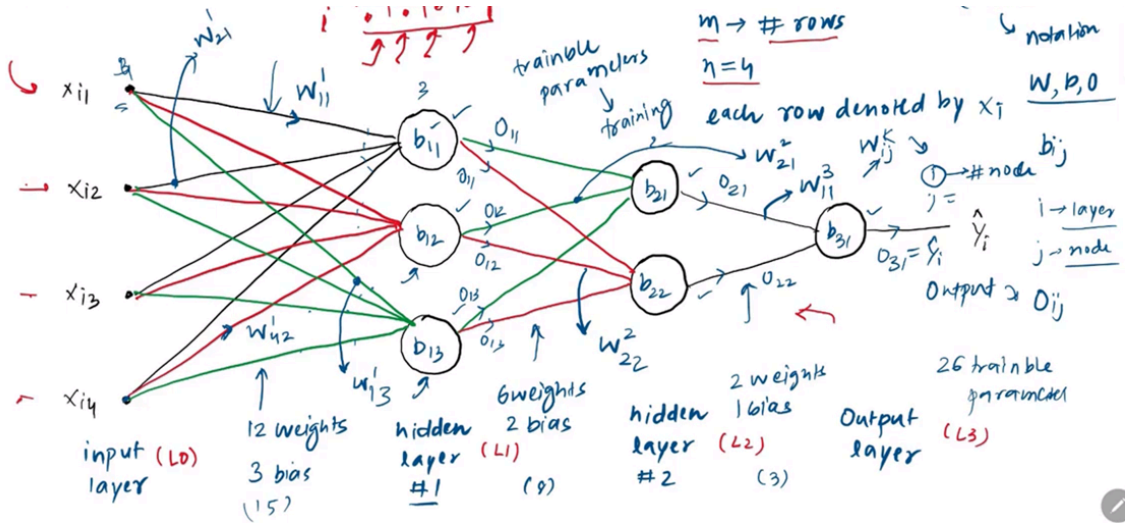
look perceptron-demo.ipynb in deep learning folder

- how to train PERCEPTRON?



TensorFlow Playground - https://playground.tensorflow.org/ (https://playground.tensorflow.org/)

multi network



churn.ipynb

mnist_classification.ipynb

what is loss function?

- it is a method of evaluating how well yor algorithmm is modelling your dataset?

differance between cost and loss function

**Loss Function:

- Focus: Measures the error for a single data point (sample) in your training set.
- Calculation: Compares the predicted value for that data point with the actual ground truth value.
- Example: Mean Squared Error (MSE) for a single data point. **Cost Function:
- Focus: Calculates the average error across a set of data points, typically the entire training batch or epoch.
- Calculation: Usually involves averaging the loss function values for all data points in the set. Sometimes, it might include additional terms like regularization penalties.

- Example: Average of the MSE values for all data points in a training batch. **Here's an analogy to illustrate the difference:
- Loss Function: Imagine grading a single student's exam. The loss function is the number of points they missed.
- Cost Function: Now, imagine calculating the average score for the entire class. The cost function is the average number of points missed by all students. **Key Points:
- Loss function: Single data point error.
- Cost function: Average error over a set of data points (often the entire training batch or epoch).
- In many cases, the cost function is simply the average of the loss function values.
- Some argue that cost function might encompass additional terms like regularization penalties, while loss function strictly refers to the error calculation.
- If you're referring to the error for a single data point, use "loss function."
- If you're referring to the overall error across a set of data points, "cost function" is generally appropriate.

Deep Learning Loss Functions: Classification & Regression

Loss functions play a critical role in deep learning by guiding the training process. They measure the discrepancy between a model's predictions and the ground truth labels. By minimizing the loss, we aim to improve the model's performance.

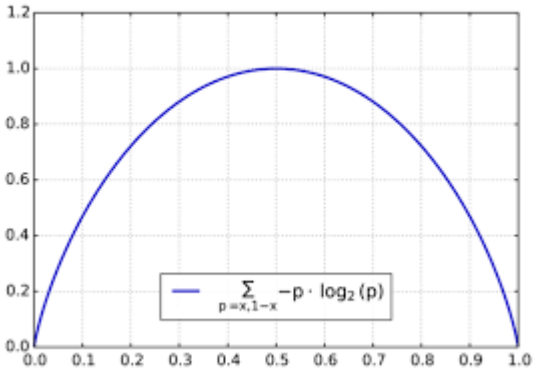
Classification Loss Functions

These functions evaluate the difference between the model's predicted class probabilities and the actual class labels. Here are two common choices:

1. Cross-Entropy Loss (Categorical Cross-Entropy)

- output layer active function softmax
- use in multi classification(more than two)
- in output column one hot encode
- in output layer number of node = no of label or number of class
- **Formula:** `- Σ (y_i * log(p_i))` where:
  - y_i is the true label (1 for the correct class, 0 for others)
  - p_i is the predicted probability for class i
- **Pros:**
  - Well-suited for multi-class problems.
  - Encourages focusing on difficult-to-classify examples.
- **Cons:**
  - Sensitive to class imbalance (more data for some classes).

2. Binary Cross-Entropy Loss



- **Formula:** Same as Cross-Entropy for binary classification (two classes).
- use in two class
- output activation function sigmoid
- **Pros:**
  - Efficient for two-class problems.
  - Simpler to interpret.
  - differentiable
- **Cons:**
  - Not applicable to multi-class problems.
  - multiple local minimum
  - not intuitive

**3.sparse categorical cross-entropy loss and categorical cross-entropy loss are very similar, but with a key difference in how the true labels (ground truth) are represented. Here's an explanation of sparse categorical cross-entropy loss, including the formula:

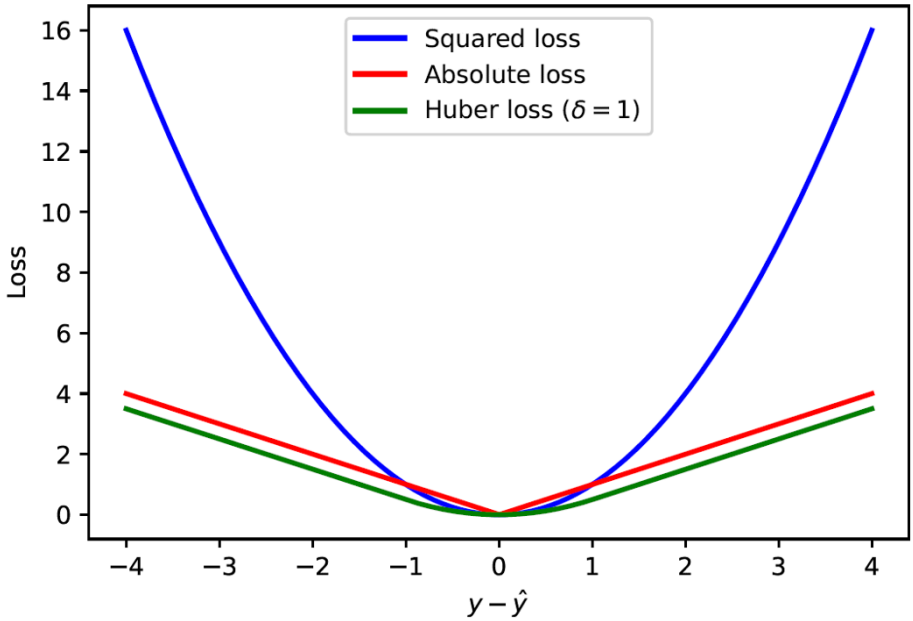
- This loss function is used in multi-class classification problems where the true labels are represented as integers (indices) rather than one-hot encoded vectors.

**Formula:

- Loss = - Σ (y_i * log(p_i))

Regression Loss Functions

These functions assess the difference between a model's predicted continuous values and the actual target values. Here are two common choices:



1. Mean Squared Error (MSE)

- **Formula:** `(1/n) * Σ (y_i - y_hat_i)^2` where:
  - y_i is the true value
  - y_hat_i is the predicted value
  - n is the number of samples
- **Pros:**
  - Simple to calculate.
  - Interpretable (error units squared).
  - differenciate
  - one local minma
- **Cons:**
  - Sensitive to outliers (can heavily influence the loss).
  - May not be ideal for non-normal distributions.
  - error unit in squared

2. Mean Absolute Error (MAE)

- **Formula:** `(1/n) * Σ |y_i - y_hat_i|`
- **Pros:**
  - Less sensitive to outliers than MSE.
  - Robust to noise.
  - unit same
- **Cons:**
  - Not as differentiable (can lead to slower convergence in training).

Choosing the Right Loss Function

The best loss function depends on the specific problem and data characteristics:

- **Classification:**
  - Use Cross-Entropy for multi-class problems, especially when class probabilities are important.
  - Use Binary Cross-Entropy for two-class problems.
- **Regression:**
  - Use MSE for normally distributed, continuous targets where accurate predictions are vital.
  - Use MAE for non-normal distributions or datasets with outliers where robustness is important.

Additional Considerations:

- **Huber Loss:** This loss combines MAE and MSE, offering a balance between robustness and accuracy.

Huber Loss Formula

The Huber loss function offers a compromise between Mean Squared Error (MSE) and Mean Absolute Error (MAE), providing robustness to outliers while maintaining smoothness for smaller errors.

Formula: Huber loss(y_true, y_pred, delta) = {

- 0.5 * (y_true - y_pred)^2 if |y_true - y_pred| <= delta
- delta * |y_true - y_pred| - 0.5 * delta^2 if |y_true - y_pred| > delta }

- **Loss Normalization:** Techniques like dividing by the number of samples (n) can be used for better interpretability and gradient calculations.

Remember: Experiment with different loss functions and hyperparameter tuning to find the

When and Why to Use Deep Learning Loss Functions

The selection of a loss function in deep learning is crucial for steering your model towards optimal performance. The choice hinges on two primary factors:

1. **Task Type:** Classification (predicting discrete categories) vs. Regression (predicting continuous values).
2. **Data Characteristics:** Distribution (normal vs. non-normal), presence of outliers.

Classification Loss Functions

Cross-Entropy (Categorical Cross-Entropy):

- **Use Case:** Ideal for multi-class classification problems where you care about predicted class probabilities. It emphasizes focusing on harder-to-classify examples.
- **Justification:** The logarithmic term in the formula penalizes the model more for being wrong on difficult classes, encouraging it to learn better distinctions.

Binary Cross-Entropy:

- **Use Case:** Specifically designed for binary classification (two classes). It's computationally efficient and easier to interpret.
- **Justification:** Simplifies calculation and interpretation compared to cross-entropy due to the two-class nature. It directly measures the discrepancy between the predicted probability of the positive class and the actual label (0 or 1).

Regression Loss Functions

Mean Squared Error (MSE):

- **Use Case:** A popular choice for regression problems when you prioritize accurate predictions and have likely normally distributed data. It minimizes the squared difference between predicted and actual values.
- **Justification:** Squaring the error amplifies larger deviations, guiding the model to reduce significant prediction errors. However, it's sensitive to outliers.

Mean Absolute Error (MAE):

- **Use Case:** A robust alternative to MSE for datasets with outliers or non-normal distributions. It minimizes the absolute difference between predicted and actual values.
- **Justification:** The absolute value of the error makes it less susceptible to outliers. However, MAE can lead to slower convergence due to its non-differentiability at zero.

Key Takeaways

- **Classification:**
  - Cross-entropy (multi-class)

    - Binary cross-entropy (two classes)
- **Regression:**
    - MSE (normally distributed data)
    - MAE (outliers, non-normal distributions)
- **Experimentation is Key:** Try different loss functions to find the best fit for your specific problem and data.

Additional Considerations:

- **Huber Loss:** Combines MAE and MSE for a balance between robustness and accuracy.
- **Class Imbalance Techniques:** Consider these if your classes are not equally represented.

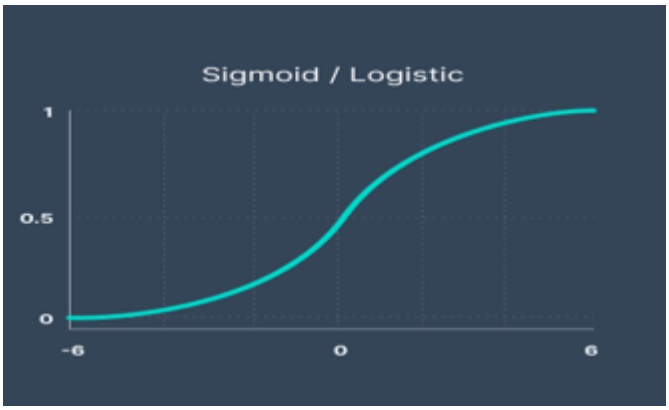
By understanding these factors and the trade-offs of each loss function, you can effectively guide your deep learning models towards superior performance.

An **activation function** in a neural network introduces non-linearity to the model, enabling it to learn and represent complex patterns in the data. It determines the output of a neuron by transforming the weighted sum of its inputs. Common activation functions include ReLU, sigmoid, and tanh.

ideal activation function

1. non-lineae
2. diffrentiable
3. computationally(simple,easy and fast)
4. zero centered/normalize means next layer means=0
5. non-saturating (in saturating vanish gradiant problem)

**1. Sigmoid Function:



Formula: f(x) = 1 / (1 + exp(-x))

Output Range: (0, 1)

- Properties: Outputs a value between 0 and 1, suitable for tasks like probability prediction (logistic regression). Suffers from vanishing gradients in deep networks, making it less preferred for complex architectures

advantage

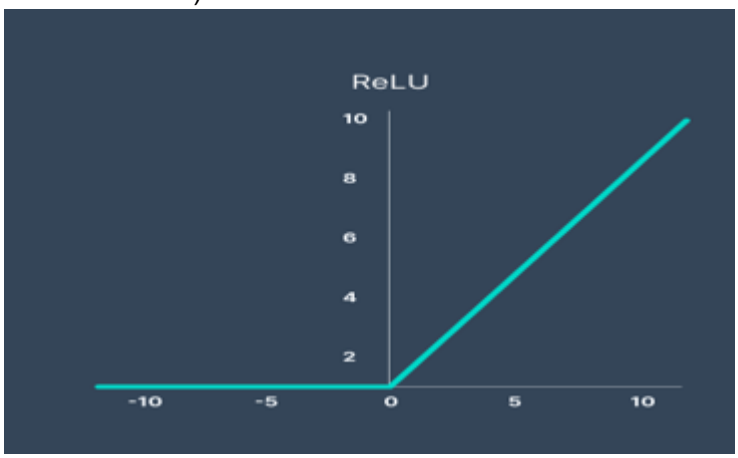
- [0,1] binary classification
- non linear

- diffrentiable

disadvantage

- saturating function (vanishing gradiant)
- non zero centerd (not normalize ,mean not zero,tranning slow)
- computation

**2. ReLU (Rectified Linear Unit):



Formula: f(x) = max(0, x)

Output Range: (0, infinity)

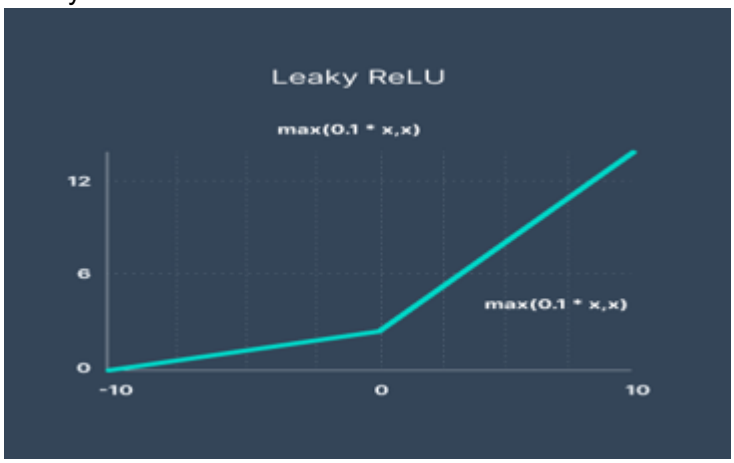
- Properties: Simpler and computationally efficient compared to sigmoid. Addresses vanishing gradients, making it a popular choice for many deep learning tasks. Can suffer from "dying neurons" if the learning rate is too high, where some neurons never activate and their weights never get updated.

advantage

- non linear
- not saturated in posative sign
- computationally
- faster other

dis advantage

- diffrentiable
- dying relu **3. Leaky ReLU:

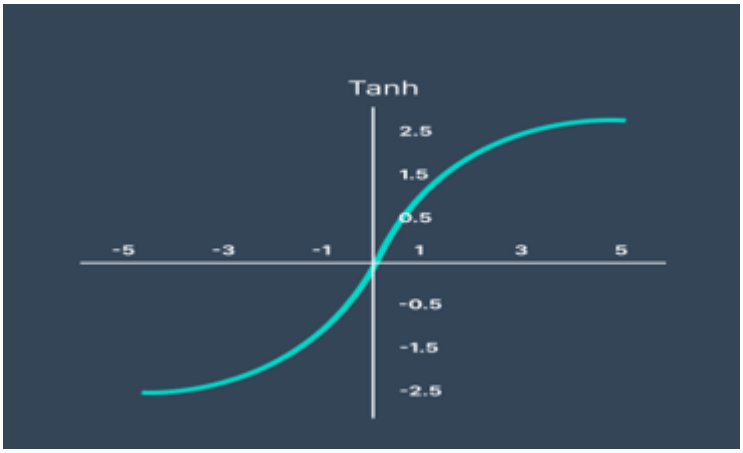


Formula: f(x) = max(leak * x, x) (where leak is a small value, typically between 0.01 and 0.1)

Output Range: (-infinity, infinity)

- Properties: Combines the benefits of ReLU (non-vanishing gradients) and avoids the dying neuron issue of ReLU by allowing a small non-zero gradient for negative inputs.

**4. tanh (Hyperbolic Tangent):



Formula: f(x) = (exp(x) - exp(-x)) / (exp(x) + exp(-x))

Output Range: (-1, 1)

- Properties: Outputs values between -1 and 1, similar to sigmoid but with a steeper slope around zero. Can be useful when you want the output to be centered around zero.

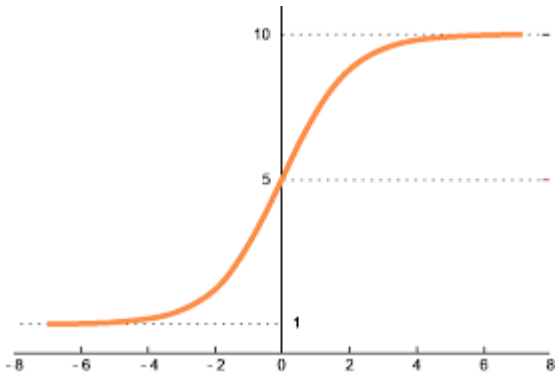
advantage

- non linear
- zero centered
- - diffrentiable

disadvantage

- saturating function (vanishing gradiant)
- computation

5. Softmax:



Formula: f(x_i) = exp(x_i) / sum(exp(x_j)) for all j (applied element-wise)

Output Range: (0, 1) for each element, sum to 1 across all elements

- Properties: Typically used in the output layer for multi-class classification problems. Outputs a probability distribution over all possible classes, where the highest value represents the most likely class.

Choosing the Right Activation Function:

The best activation function for your network depends on several factors, including:

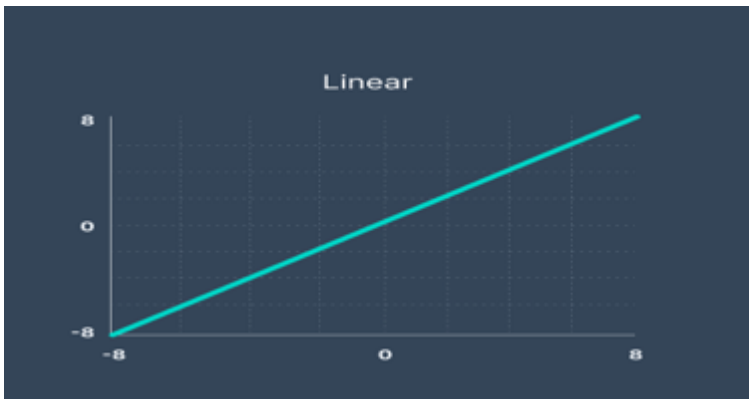
- Task: Classification, regression, or something else?
- Network Architecture: Number of layers, type of layers?
- Data Distribution: Is the data centered around zero or skewed?
- Computational Efficiency: How much computation can you afford?

Here are some general guidelines:

- ReLU: A common and good default choice for many hidden layers due to its simplicity and ability to address vanishing gradients.
- Leaky ReLU: Can be a good alternative to ReLU to avoid dying neurons.
- Sigmoid: Primarily used in output layers for binary classification (logistic regression) or when you need probability-like outputs between 0 and 1.
- tanh: Can be useful when you want the output to be centered around zero, but is less common than ReLU or Leaky ReLU in hidden layers.
- Softmax: Essential for multi-class classification problems in the output layer.

**linear **

- The linear activation function, also sometimes referred to as the identity function, is the simplest activation function used in neural networks. Here's a breakdown of its characteristics and why it's generally not the preferred choice for hidden layers:



Formula:

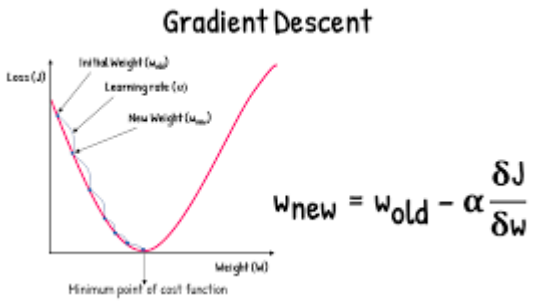
f(x) = x

- Output: The linear activation function simply outputs the weighted sum of its inputs (x) without any modification

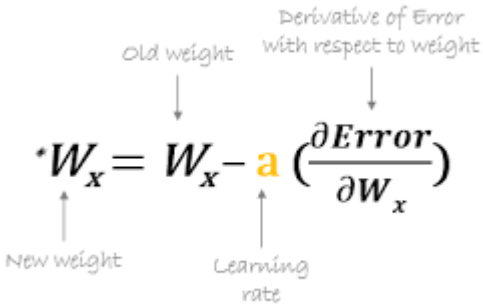
Backpropagation:

- it is algoritham train(find optimum value of weight and bias) the neural network **steps

1. select a point(row)
2. predict value for that point(forward propagation)
3. choose a loss function



4. weight and bias update based on gradient descent(partial derivative) for minimum loss
5. update weight and bias untill minimum loss



$$W = W - \alpha \frac{\partial L}{\partial W}$$

$$b = b - \alpha \frac{\partial L}{\partial b} \qquad (\alpha - Learning\ Rate)$$

backpropagation_classification.ipynb refer

- loss function is function of trainable parameter
- change of trainable parameter towards minimum loss is called gradient descent
- Backpropagation needs a learning rate to update weights effectively. It acts like a step size during training, determining how much to adjust the weights in the direction of the gradient (steepest descent).
- Too small a learning rate leads to excruciatingly slow training, while too large a learning rate can cause the model to overshoot the optimal weights or even diverge.
- The ideal learning rate balances speed and stability, allowing the model to converge

https://developers-dot-devsite-v2-prod.appspot.com/machine-learning/crash-course/backprop-scroll (https://developers-dot-devsite-v2-prod.appspot.com/machine-learning/crash-course/backprop-scroll)

https://developers.google.com/machine-learning/crash-course/fitter/graph (https://developers.google.com/machine-learning/crash-course/fitter/graph)

improve neural network

1. Fine tuning NN hyper parameter

- hidden layer
- neuron
- learning rate

- optimizer
- batchsize
- activation function
- epochs

2. exploding gradient
3. not enough data
4. overfitting

Fine-Tuning Neural Network Hyperparameters: A Summary

1. Hidden Layers and Neurons:

**Hidden Layers:

- Start simple, increase complexity gradually.
- More layers for intricate relationships, fewer for simpler tasks. -**Neurons:
- Balance capacity and overfitting.
- Consider a "fan-in" to "fan-out" ratio of 2:1 to 4:1.

2. Learning Rate:

- Controls the magnitude of weight updates.
- Too high: Unstable training (oscillation, divergence).
- Too low: Slow convergence.
- Start small (0.001-0.01), adjust gradually based on validation.
- Consider learning rate decay.

3. Optimizer:

- Algorithm for weight updates based on the loss function.
- Common choices: SGD (simple, slow), Adam (adaptive, faster), RMSprop (adaptive, addresses SGD issues).
- Experiment with different options.

4. Batch Size:

- Number of training examples used for weight updates per iteration.
- Large: Better generalization, slower convergence.
- Small: Faster training, noisier gradients, worse generalization.
- Start moderate (e.g., 32 or 64), adjust based on memory and speed.

5. Activation Function:

- Introduces non-linearity for learning complex relationships.
- Common choices: Sigmoid (vanishing gradients), ReLU (dying neurons), Leaky ReLU (mitigates dying neurons), tanh (scaled output).
- Experiment for each layer/layer type (e.g., ReLU for hidden, softmax for classification output).

6. Epochs:

- One complete pass through the entire training dataset.
- Too few: Underfitting (insufficient learning).
- Too many: Overfitting (memorizes training data, poor generalization).

- Use validation set, stop training with early stopping to avoid overfitting.

- Data augmentation (expand dataset with transformations).
- Regularization (L1/L2, dropout) to prevent overfitting.
- Normalization/standardization (preprocess data for better training).

how to prevent overfitting in Convolutional Neural Networks (CNNs):

1. Data Augmentation: Artificially expand your training data ith random transformations (flipping, rotation, etc.) to improve generalization.
2. Regularization: Penalize complex weights (L1/L2) or randomly drop neurons **(Dropout)** to prevent the model from memorizing training data.
3. Reduce Model Complexity: Start with a simpler architecture (fewer layers/filters) and increase complexity only if necessary.
4. Early Stopping: Stop training when validation loss increases to avoid overfitting to the training data.
5. Transfer Learning: Leverage pre-trained models on large datasets to reduce the number of parameters your model needs to learn.
6. Batch Normalization: Improve training stability and potentially reduce overfitting by normalizing activations in hidden layers.

Reduce overfitting

1. add more varity of data
2. reduce complexity by reduce node
3. early stopping
4. regularization
5. Dropout

Dropout

is a powerful regularization technique used in neural networks to prevent overfitting, particularly in deep architectures with many layers and parameters. Here's a breakdown of how dropout works:

- each epoch make a nn

Concept:

During training, dropout randomly "drops" (temporarily ignores) a certain percentage of neurons along with their connections from the network at each iteration. This forces the network to learn using different subsets of neurons in each pass, preventing it from relying too heavily on any specific set of features or connections.

Benefits:

Reduces Overfitting: By making the network learn robust features that are independent of specific neurons, dropout prevents it from memorizing the training data and improves its ability to generalize to unseen examples.

Ensemble Effect: The random nature of dropping neurons during training can be seen as training an ensemble of slightly different networks simultaneously. When you test the model, all neurons are active again, effectively approximating the average behavior of the ensemble, often leading to better performance.

Implementation:

Define dropout rate (p): This value represents the probability of a neuron being dropped. A common range is 0.2 to 0.5, meaning 20% to 50% of neurons are dropped in each training iteration.

Randomly drop neurons: During training, for each neuron in each layer (except the input and output layers), a random number between 0 and 1 is generated. If the number is less than the dropout rate (p), the neuron is dropped along with its incoming and outgoing connections for that iteration.

Scale activations: To compensate for the fact that only a portion of neurons are active on average during training, the activations of the remaining neurons are scaled by a factor of 1 / (1 - p). This ensures that the average activation remains the same at test time when all neurons are active.

if dropout rate 0.2 how to calculate value of weight

In dropout, the weight values themselves are not directly modified during the random dropping process. However, the activations of the remaining neurons are scaled to compensate for the fact that only a portion of the network is active during training. Here's how the dropout rate (p, in this case 0.2) affects the calculation of the activation values:

1. Weight Updates Remain Unchanged:

Dropout focuses on temporarily disabling neurons during training, not modifying the weights themselves. The weights are still updated based on the backpropagation algorithm using the scaled activations.

2. Scaling Activations:

Scaling Factor: To maintain the average activation level during training (when some neurons are dropped) and testing (when all neurons are active), the activations of the remaining active neurons are scaled by a factor of 1 / (1 - p). Example: With a dropout rate of p = 0.2 (20% dropout), the scaling factor becomes 1 / (1 - 0.2) = 1 / 0.8 = 1.25. 3. Calculation of Scaled Activation:

Let x be the original activation value of a neuron before dropout. If the neuron is not dropped (active during training), its scaled activation (y) is calculated as: y = 1.25 * x // Assuming dropout rate p = 0.2 This scaling ensures that the average activation reaching the next layer remains the same during training, even though fewer neurons are contributing.

4. Weight Update:

During backpropagation, the scaled activation (y) is used along with the weights and the error signal to update the weights using the standard backpropagation rule.

Key Points:

Dropout rate (p) affects the scaling factor (1 / (1 - p)). This scaling factor is applied to the activations of the remaining active neurons. Weight updates still occur based on the scaled activations and the error signal. In summary, dropout doesn't directly change weight values. It scales the activations of the surviving neurons during training to compensate for the inactive ones, maintaining the average activation level and promoting robustness against overfitting.

dropout_classification_example.ipynb

- if overfiting p increase
- underfiting p decrease,
- cnn p range 0.2 to 0.5

Batch Normalization

1. What is Batch Normalization (BatchNorm)?

Function: A technique used during training in deep learning to normalize the activations (outputs) of hidden layers.

Normalization: Standardizes the activation distribution to have a mean of zero and a standard deviation of one.

2. Benefits of Batch Normalization:

Improves Gradient Flow:

Addresses vanishing/exploding gradients during backpropagation in deep networks.

Allows gradients to flow more effectively, leading to faster and more stable training.

Reduces Internal Covariate Shift: Mitigates the shift in activation distributions between layers during training.

Makes the training process less sensitive to the order of data presentation.

Allows Higher Learning Rates:

Enables using higher learning rates without encountering gradient issues.

Accelerates the training process.

3. Importance of Batch Normalization:

Faster and More Stable Training:

Addresses vanishing gradients and internal covariate shift.

Makes training deep networks more efficient and reliable.

Reduces Need for Manual Tuning: Lessens the need for extensive hyperparameter tuning (e.g., learning rate).

Simplifies the training process and minimizes human error.

Improves Regularization:

Acts as a form of regularization, reducing overfitting. Makes the network less sensitive to the specific training data distribution.

4. Additional Notes:

Application: Primarily applied to hidden layers, not input or output layers.

Necessity: May not be essential for very shallow networks or specific tasks.

Experimentation: Evaluate its effectiveness for your specific model and dataset.

In [ ]:
```
1
```