**Direct preference optimization (DPO)**

Yes! **DPO updates model weights using fine-tuning**, but **without reinforcement learning (RL)**.

Unlike RLHF, which uses **a reward model and PPO**, DPO **directly fine-tunes** the language model using a contrastive loss function.

### **🔹 How Does DPO Fine-tune the Model?**

DPO fine-tunes the model by making **preferred responses** more likely than **rejected responses** using a supervised learning approach.

**Step-by-step process:**  
1️⃣ **Collect Human Preferences:** Get pairs of responses where one is preferred.  
2️⃣ **Compare Probabilities:** Compute the probability of both preferred & rejected responses.  
3️⃣ **Use a Contrastive Loss Function:** Fine-tune the model to make the preferred response **more likely** than the rejected one.  
4️⃣ **Optimize with Supervised Learning:** Update the model weights using gradient descent.

### **🔹 RLHF vs. DPO Weight Updates**

| **Feature** | **RLHF (PPO)** | **DPO (Fine-tuning)** |
| --- | --- | --- |
| **Uses Reward Model?** | ✅ Yes | ❌ No |
| **Uses RL (PPO)?** | ✅ Yes | ❌ No |
| **Updates Weights?** | ✅ Yes (via PPO) | ✅ Yes (via fine-tuning) |
| **Training Stability** | ❌ Unstable | ✅ More stable |
| **Computational Cost** | 🔥 High | 💡 Lower |

### **🔹 Key Takeaway**

✔ **DPO fine-tunes the model** by adjusting weights with a contrastive loss function.  
✔ **No reinforcement learning or reward model is needed.**  
✔ **Simpler, more stable, and cheaper than RLHF.**

### **🔹 What is Q-Learning?**

**Q-Learning** is a **reinforcement learning (RL) algorithm** that helps an agent learn the best actions to take in an environment **to maximize rewards over time**.

It is an **off-policy, model-free** RL method, meaning:  
✅ It does **not require a model** of the environment.  
✅ It **learns from past experiences** and can use stored data to improve future actions.

## **🔹 How Q-Learning Works**

Q-Learning revolves around a **Q-table (or Q-function)**, which stores the **expected reward** for taking an action aaa in a given state sss:

Q(s,a)=Expected Future RewardQ(s, a) = \text{Expected Future Reward}Q(s,a)=Expected Future Reward

### **🛠️ Steps in Q-Learning**

1️⃣ **Initialize the Q-table** 📋

* Create a table where rows represent **states** and columns represent **actions**.
* Fill it with random values.

2️⃣ **Choose an Action (Exploration vs. Exploitation)** 🎯

* Use an **ε-greedy policy**:
  + With probability **ε**, choose a random action (exploration).
  + Otherwise, pick the **best action** from the Q-table (exploitation).

3️⃣ **Take the Action & Observe the Reward** 🎮

* Move to a new state and receive a **reward** from the environment.

4️⃣ **Update the Q-value (Using the Bellman Equation)** 🔄

* The Q-value is updated using:

Q(s,a)=Q(s,a)+α(r+γmax⁡Q(s′,a′)−Q(s,a))Q(s, a)

Where:

* α\alphaα = Learning rate (how fast we update Q-values).
* γ\gammaγ = Discount factor (how much future rewards matter).
* rrr = Immediate reward received.
* max⁡Q(s′,a′)\max Q(s', a')maxQ(s′,a′) = Best future reward from the next state.

5️⃣ **Repeat Until Convergence** 🔄

* Keep updating Q-values until the agent **learns the best actions**.

## **🔹 Example: Q-Learning in Action**

Let's say we have a **robot in a grid world**, and it needs to find the shortest path to a **goal** while avoiding obstacles.

| **State** | **Action** | **Q-Value (Before)** | **Reward** | **Q-Value (After)** |
| --- | --- | --- | --- | --- |
| 🏁 Goal | Any | 0 | 100 | 100 |
| 🤖 A → Right | Right | 0 | -1 | -1 |
| 🤖 A → Down | Down | 0 | 10 | 10 |

Over time, the Q-table updates and the agent **learns to take better paths**!

## **🔹 Q-Learning vs. Deep Q-Learning (DQN)**

| **Feature** | **Q-Learning** | **Deep Q-Learning (DQN)** |
| --- | --- | --- |
| **Stores Q-values?** | ✅ In a table | ❌ In a neural network |
| **Handles Large State Spaces?** | ❌ No | ✅ Yes |
| **Uses Deep Learning?** | ❌ No | ✅ Yes |
| **Works in Complex Environments?** | ❌ Limited | ✅ Better |

DQN is an advanced version of Q-Learning that **uses neural networks** to approximate Q-values **instead of a table**, making it work for large problems like **video games and robotics**.

## **🔹 Key Differences Between Q-Learning Rewards & RLHF Reward Model**

| **Feature** | **Q-Learning Reward** | **RLHF Reward Model** |
| --- | --- | --- |
| **Who Gives the Reward?** | **Predefined by humans** (hardcoded rules in the environment). | **Learned from human feedback** using a machine learning model. |
| **How is Reward Computed?** | **Explicit function** (e.g., +10 for reaching a goal, -5 for hitting a wall). | **Trained model** predicts reward scores based on human-labeled preferences. |
| **Use Case** | Used in **reinforcement learning for games, robotics, etc.** | Used in **fine-tuning LLMs (e.g., ChatGPT, Bard) to align with human preferences**. |
| **Is it Fixed or Learned?** | **Fixed (hardcoded function).** | **Learned (trained from human feedback).** |
| **Optimization Method** | **Bellman equation** updates Q-values. | **PPO (Proximal Policy Optimization)** adjusts the LLM’s responses. |

### **🔹 What Are Transformers?**

Transformers are a type of **deep learning architecture** introduced in the paper **"Attention Is All You Need" (2017, Vaswani et al.)**. They revolutionized NLP (Natural Language Processing) by using the **self-attention mechanism** to process entire sequences **in parallel** rather than sequentially.

## **🔹 What Problems Did Transformers Solve?**

Before transformers, NLP models relied on architectures like **RNNs (Recurrent Neural Networks) and LSTMs (Long Short-Term Memory networks)**, which had several issues:

### **1️⃣ Problem: Sequential Processing in RNNs**

**❌ Issue:** RNNs process words **one by one** in a sequence, making training **slow** and limiting parallel computation.  
**✅ Solution:** Transformers **process the entire sequence at once**, allowing for **parallelization** and **faster training**.

### **2️⃣ Problem: Long-Term Dependencies in RNNs & LSTMs**

**❌ Issue:** RNNs struggle to retain information from **earlier words** in long sentences.  
**✅ Solution:** The **self-attention mechanism** in transformers allows the model to **focus on relevant words**, no matter how far apart they are.

### **3️⃣ Problem: Vanishing Gradient in RNNs**

**❌ Issue:** As sequences get longer, the gradients shrink, making it **hard to learn long-term dependencies**.  
**✅ Solution:** Transformers **don’t rely on recurrence**, so they **avoid vanishing gradient issues**.

### **4️⃣ Problem: Fixed Context in LSTMs (Fixed-Length Memory)**

**❌ Issue:** LSTMs store a **fixed** amount of past information, meaning long-range dependencies may be **lost**.  
**✅ Solution:** **Self-attention dynamically attends** to all words in a sequence, regardless of distance.

### **5️⃣ Problem: Poor Scaling in RNNs**

**❌ Issue:** Training large RNNs on large datasets is **computationally expensive**.  
**✅ Solution:** **Transformers scale better** because they use **parallelization**, making them ideal for large models like GPT and BERT.

## **🔹 How Do Transformers Work?**

Transformers rely on three main components:

1️⃣ **Self-Attention Mechanism**

* Helps the model focus on **important words** in a sequence.
* Example: In **"The cat sat on the mat"**, "cat" is **strongly related** to "sat" but **weakly related** to "mat".
* The self-attention mechanism **assigns higher weights** to relevant words.

2️⃣ **Positional Encoding**

* Since transformers **don’t process words sequentially**, they add **position embeddings** so the model knows the **order** of words.

3️⃣ **Feed-Forward Networks (FFN)**

* Each transformer block has a **fully connected neural network** that processes the attention outputs.

PPO

### **🔹 Why is Distillation More Effective for Encoder-Only Models?**

Knowledge distillation is a technique where a **smaller (student) model** learns from a **larger (teacher) model**. While distillation can be used for any model, it is **especially effective for encoder-only models like BERT**. Here’s why:

## **1️⃣ Encoders Compress Information Efficiently**

🔹 **Encoder-only models** (e.g., BERT) **compress input representations** into meaningful embeddings.  
🔹 During distillation, the **student model learns these compressed embeddings**, making the process **more effective**.  
🔹 **Decoders, on the other hand, generate new tokens, making distilled knowledge harder to retain.**

📌 **Example:** Distilling BERT into a smaller version (e.g., DistilBERT) is more effective than distilling a decoder-based model like GPT.

## **2️⃣ Distillation Works Well for Representation Learning**

🔹 Encoder models (like BERT) **focus on feature extraction** and **semantic representation**.  
🔹 Since the **goal of distillation is to transfer knowledge efficiently**, it works **best for models that rely on good representations** rather than generative capabilities.  
🔹 Decoder-based models (like GPT) **generate text**, making it harder to distill because **generation involves more randomness and diverse outputs**.

📌 **Example:** A distilled BERT model can maintain **90-95% accuracy** while reducing model size significantly.

## **3️⃣ Loss Function Aligns Better with Encoders**

🔹 Distillation typically uses a **soft-label loss (KL divergence)** that helps the student **match the teacher’s probability distributions**.  
🔹 In **encoder models, these probability distributions** are stable and well-structured, making distillation **more effective**.  
🔹 **Decoder models, however, have variable-length outputs**, which make it **harder for the student to match the teacher’s outputs**.

📌 **Example:** In BERT, each token **gets a fixed representation**, but in GPT, **each output depends on previous outputs**, making distillation harder.

## **4️⃣ Faster and More Efficient Training for Encoders**

🔹 Distilling an encoder-only model like BERT leads to **significant speed improvements** with minimal performance loss.  
🔹 Since **decoders generate text autoregressively (word by word)**, they require **sequential processing**, making distillation less efficient.

📌 **Example:** DistilBERT runs **60% faster** than BERT with **only a slight drop in accuracy**.

Homonyms was the big problem and transformer architecture is there to create dynamic contextual embedding

BERT is encoder only architecture

Encoder only architectures are being used in applications like classification and many much....

GPT BLOOM LLAMA jurrasic are decoderonly models

BART T5 are encoder decoder architecture

**Transfer learning** is a machine learning technique where a model trained on one task is reused or ***fine-tuned*** for a different but related task. Instead of training a model from scratch, transfer learning leverages knowledge from a pre-trained model, which helps improve performance and reduces training time, especially when limited data is available.

When using Reinforcement Learning with Human Feedback (RLHF) to align large language models with human preferences, what is the role of human labelers?

In RLHF, human labelers score a dataset of completions by the original model based on alignment criteria like helpfulness, harmlessness, and honesty. This dataset is used to train the reward model that scores the model completions during the RLHF process.

Although compute optimization is important, it can be challenging to obtain a sufficient amount of data tokens. In the case of BloombergGPT, they had a limited token count and even used fewer tokens due to early stopping. While chinchilla laws offer valuable guidance, they should not be strictly followed as rigid rules.

What is the name of the technique used in the output dense layer that is used to predict Bounding Boxes ? (Hint: It is a one word answer)

Enter answer here

Regressiion, it gives dimentions of bounding box

Catastrophic forgetting occurs when a machine learning model forgets previously learned information as it learns new information.

*Correct*

The assertion is true, and this process is especially problematic in sequential learning scenarios where the model is trained on multiple tasks over time.

Catastrophic forgetting is a common problem in machine learning, especially in deep learning models.

*Correct*

This assertion is true because these models typically have many parameters, which can lead to overfitting and make it more difficult to retain previously learned information.

This assertion is true because these models typically have many parameters, which can lead to overfitting and make it more difficult to retain previously learned information.

Catastrophic forgetting only occurs in supervised learning tasks and is not a problem in unsupervised learning.

*Un-selected is correct*

One way to mitigate catastrophic forgetting is by using regularization techniques to limit the amount of change that can be made to the weights of the model during training.

*You should have selected this*

Q) Check all the scenarios in which Transfer Learning could be beneficial.

1. To ensure better performance
2. To reduce computation and processing cost
3. When you don’t have enough data for the task you want to perform, which resembles another same or similar, already trained task.
4. When the task you want to perform is a sub-task of an already trained, larger, model.

***2,3,4***

Q) What does “include\_top=False” mean ?

1. It sets the top most layers as untrainable of ResNet50 when initializing my\_layer using it.
2. It randomly sets up the weights, instead of using that of ImageNet, for the top most dense layers of ResNet50 when initializing my\_layer using it.
3. It discards the first layer of ResNet50 when initializing my\_layer using it.
4. It discards the top most layers of ResNet50 when initializing my\_layer using ResNet50.

***D***

Q) In the context of Transfer Learning, the initial training task where the model learns reusable patterns is called a downstream task.

Ans) The statement is false.

In the context of Transfer Learning, the initial training task where the model learns reusable patterns is called the pretraining task or source task. The task where the model is fine-tuned or adapted to a specific application is called the downstream task.

Q) What is the method that locates an object(s) by labelling the pixels, where each similar object(s) is assigned to the same class? Type your response here (two words, all lower case).

Ans) semantic segmentation

Q) Which of the following statements correctly describes the difference between object detection and object localization? ***2***

1. Object detection refers to detecting the object within an image, while object localization gives us the bounding box around that object.
2. They both are the same.
3. Object localization is where you get a bounding box around the main subject of the image, while in object detection you get a bounding box around all of the objects within an image.
4. Object detectionis where you get a bounding box around the main subject of the image, while in object localization you get a bounding box around all of the objects within an image.

Q) In a Multi-Class classification scenario, your model can identify all the different items and people that are present in a given input image.

True

***False***

Q) Which of the following is true about training your model using data parallelism technique? Check all that are true.

All of the data is on 1 master machine, and copies of the data are then distributed to machines having different model architectures based on their capacity of processing the data.

The full data set is split up and subsets of the data are stored across multiple machines

Correct

Correct! Data parallelism is meant to improve efficiency by not having to store or process all of the data on the same machine.

The same model architectures are used on different machines, and each machine processes the entire data set.

Weights from different machines are aggregated and updated into a single model.

Correct

Correct! All the learnings from training on multiple machines should be used to update a single model.

Q) In TensorFlow version 2, tf.distribute.Strategy class supports \_\_\_\_\_\_\_. Check all that apply.

Graph Mode

Correct

Correct!

Eager Mode

Correct

Correct!

**Question 3**

Which of the following are true of both MirroredStrategy and TPU Strategy? Check all that are true.

Uses multiple machines

The same model is replicated on each core.

Correct

Correct! Both of these strategies use multiple cores on the same machine (either GPU for Mirrored Strategy or TPU for TPU strategy)

Uses a single machine

Variables are synchronized (mirrored) across each replica of the model

Correct

Correct! Variables are mirrored across the copies of the model.

You didn't select all the correct answers

**Question 4**

To modify training code to work with Mirrored Strategy, which of the following should we do? Choose all that apply.

Put the code that creates, compiles and fits the model inside the scope of “with strategy.scope()”.

Increase the batch size as long as the number is 2^n (e.g. 64, 128, 256 etc).

Put code that creates the model object inside the scope of “with strategy.scope()”.

Correct

Correct: the model creation code should be written within the scope of the strategy.

Adjust the batch size to equal the batch size per replica times the number of replicas

Correct

Correct! The batch size that the model can handle is now the number of examples that can be processed across all replicas of the model.

**Question 5**

To modify training code to work with distributed data, which of the following should we do? Choose all that apply.

Use strategy.run to run the code that updates the model weights (calculating loss, calculating the gradients, and applying the gradients).

Correct

Correct! Use strategy.run and pass in a function that contains the code which updates the model weights and returns the calculated loss.

Use strategy.reduce to aggregate the losses across the replicas.

Correct

Correct! After the replicas all train, update their weights, and return their losses, their losses are aggregated using strategy.reduce

Use strategy.experimental\_distribute\_dataset to convert training and test sets into distributed datasets.

Correct

Correct!

Replace the code that updates the model weights (calculating loss, calculating gradients, and applying the gradients) so that each training step handles all replicas at once.

in custom training, for multiple GPU mirror strategy, distributed train step function with call train step function for every single processor

R-CNN uses support vector machines to classifoy regions

Question 2

Which of the following statements is true for tf.cond ?

1. Graph execution does not support if/else statements. To replicate that effect you use tf.co
2. tf.cond is an alternative to using if/else statements in Graphs, as its execution is much faster than if/else statements.

***A***

Question 4

Function written in Eager mode when converted to Graph accommodates different data types all in one, so you don’t have to define similar functions for different data types.

**True**

***Explanation***:

In TensorFlow, when a function written in Eager mode is converted to a Graph (using @tf.function), it can handle different data types by creating separate versions of the graph for each data type. This feature is called AutoGraph polymorphism or trace polymorphism.

**@tf.function**

**def add(x, y):**

**return x + y**

# This works for both float and int types.

print(add(1, 2)) # Integer inputs

print(add(1.0, 2.0))   # Float inputs

Question 3

You have learned that Sequential and Functional APIs have their limitations.

How can you build dynamic networks where the architecture changes on the fly, or networks where recursion is used? Check all that are true:

* Using Functional API
* Using model subclassing
* Using Sequential API

Question 8

What is the purpose of using a custom contrastive loss function for a siamese model?

1. A custom built function is required because it is not possible to use a built-in loss function with the Lambda layer.
2. It is a custom built function that can calculate the loss on similarity comparison between two items.
3. A custom loss function is required for using the RMSprop() optimizer.
4. As a custom built function, it provides better results and it is faster to run.

***B***

Question

What is the output of each twin network inside a Siamese Network architecture?

1. A number
2. A softmax probability
3. An output vector
4. Binary value, 1 or 0

***C***

Question 6

A siamese network architecture has:

* 2 inputs, 1 output
* 1 input, 2 outputs
* 1 input, 1 output
* 2 inputs, 2 outputs

***A***

Q) One of the advantages of the Functional API is the option to build branched models with multiple outputs, where different loss functions can be implemented for each output.

**True**

Question 4

What are Branch Models ?

* A model architecture with a single recurring path.
* A model architecture with linear stack of layers.
* A model architecture with non-linear topology, shared layers, and even multiple inputs or outputs.
* A model architecture where you can split the model into different paths, and cannot merge them later.

***C***

Question 1

Which of these steps are needed for building a model with the Functional API? (Select three from the list below)

1. Explicitly define an input layer to the model.
2. Define the input layer of the model using any Keras layer class (e.g., Flatten(), Dense(), ...)
3. Define disconnected intermediate layers of the model.
4. Connect each layer using python functional syntax.
5. Define the model using the input and output layers.
6. Define the model using only the output layer(s).

***The correct answer is: 1, 4, 5***

Explanation:

***Explicitly define an input layer to the model:***

***The Functional API requires explicitly defining an input layer using keras.layers.Input.***

***Connect each layer using python functional syntax:***

***Layers are connected using functional syntax, where the output of one layer is passed as the input to the next layer.***

***Define the model using the input and output layers:***

***The model is defined by specifying both the input and output layers using keras.Model(inputs=..., outputs=...).***

Q) A Tensor is a flexible data structure that can hold data in a variety of different ways.

True

Q) One type of mode in TensorFlow allows for immediate evaluation of values. What is this mode called?

Eager Mode is the answer

Question 1

What are the names of the two functions which are part of the built in solution for training a model in TensorFlow and Keras?

* model.structure() and model.compute()
* model.configure() and model.map()
* model.compile() and model.fit()
* model.build() and model.train()

***C***

Question 2

With regards to the loss function, the derivative that gives you the gradient points away from the minimum loss value, but you can still use it for direction by adding the gradient value to your weight, instead of subtracting from it.

False

Explanation:

The derivative (gradient) of the loss function indicates the direction of the steepest ascent (increasing loss).

To minimize the loss, you need to move in the opposite direction of the gradient.

This is why during gradient descent, you subtract the gradient value from the weights rather than adding to it.

For creating your own custom training loop, arrange the following steps in the correct order for each training batch:

1. Accumulate accuracy metric
2. Calculate loss
3. Calculate gradients of loss with respect to the model trainable weights
4. Calculate logits
5. Apply gradients on model using optimizer

4 2 3 5 1 Answer ha

Metrics in Keras can only be called as functions and not instantiated as Classes.

False

Why is MAE a good analytic for measuring accuracy of predictions for time series?

* It only counts positive errors
* It punishes larger errors
* It doesn’t heavily punish larger errors like square errors do
* It biases towards small errors

C

agr model learning m epoch k sth loss decrease ho but spikes curves bhi hu mean fluctuating to apka batch size chhota hoga, jbhi converge zig zag ho rha

Specialized libraries:

DeepXDE: A Python library for PINNs.

SciANN: A Keras-based library for scientific computing.

Read foundational papers for PINNS:

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving PDEs. Journal of Computational Physics. DOI

Karniadakis, G. E., et al. (2021). Physics-informed machine learning. Nature Reviews Physics. DOI

Q) What is a non-stationary time series?

1. One that has a constructive event forming trend and seasonality.
2. One that is consistent across all seasons.
3. One that moves seasonally.
4. One that has a disruptive event breaking trend and seasonality.

***D***

A non-stationary time series is one that exhibits changes in its statistical properties over time, such as trend, seasonality, or abrupt disruptions. These changes could be due to events that break the established patterns of the data, such as a major shift or a disruptive event (like a crisis or new policy).

In contrast, a stationary time series has constant statistical properties, such as mean and variance, over time.

Question 9

What is Seasonality?

1. Data aligning to the 4 seasons of the calendar
2. Weather data
3. Data that is only available at certain times of the year
4. A regular change in shape of the data

***D***

Seasonality refers to the periodic fluctuations or patterns that occur at regular intervals within a time series, often due to factors like seasons, holidays, or other recurring events. These fluctuations follow a predictable pattern over time.

Define the naive\_forecast variable below. Remember that the naive forecast simply takes the last value to predict the next one. This means that the forecast series should be identical to the validation series but delayed one time step

Question 8

A sound wave is a good example of time series data

Yes

Question 5

What is an example of a Multivariate time series?

1. Hour by hour weather
2. Fashion items
3. Baseball scores
4. Hour by hour temperature

***D***

A multivariate time series involves multiple variables being tracked over time. In the case of "hour by hour weather," several variables can be measured such as temperature, humidity, wind speed, and air pressure, making it a multivariate time series.

The other options, like "fashion items" or "baseball scores," typically focus on a single variable, and "hour by hour temperature" only tracks one variable (temperature), so they would not be considered multivariate time series.

Question 3

1. What is an example of a Univariate time series?
2. Baseball scores
3. Hour by hour temperature
4. Fashion items
5. Hour by hour weather

***D***

A univariate time series consists of data points collected or recorded at consistent time intervals, and it involves only one variable. In this case, the "hour by hour temperature" involves just one variable (temperature) over time.

The other options involve multiple variables or categories (e.g., baseball scores, fashion items, and weather, which typically includes several variables such as temperature, humidity, etc.).

Question 2

In the context of time series, what is noise?

* Unpredictable changes in time series data
* Data that doesn’t have a trend
* Sound waves forming a time series
* Data that doesn’t have seasonality

A

What is a trend?

* An overall consistent downward direction for data
* An overall direction for data regardless of direction
* An overall consistent upward direction for data
* An overall consistent flat direction for data

B

Question 5

If my training data only has people facing left, but I want to classify people facing right, how would I avoid overfitting?

* Use the RandomFlip layer and set mode='vertical'
* Use the 'flip' parameter of image\_dataset\_from\_directory and set 'horizontal'
* Use the RandomFlip layer and set mode='horizontal'
* Use the ‘flip’ parameter of image\_dataset\_from\_directory

C

What does the fill\_mode parameter do?

* There is no fill\_mode parameter
* It creates random noise in the image
* It attempts to recreate lost information after a transformation like a shear
* It masks the background of an image

C

Question 7

After adding data augmentation and using the same batch size and steps per epoch, you noticed that each training epoch became a little slower than when you trained without it. Why?

* Because the training is making more mistakes
* Because there is more data to train on
* Because the augmented data is bigger
* Because the image preprocessing takes cycles

Status: [object Object]

***D***

For train/Dev/Test sets good practice is

70/30 or 60/20/20

But nowadays for bigdata trained 98/1/1 is being used in big data we might have 1 million examples

dev set == developement set== cross validation set==validation set

apka training error and crosse validation error bhly 0 na ho but 0 k kreeb h or ak doosray s bhi kreebho togood wrna bekaar

Entropy function for measuring impurity

Also there is Gini Criteria for this purpose

Root note entropy minus current not average entropy is equal to information gain. And the information gain is how much entropy reduced so we will select that whose reduction is larger.

Of continius value feature of having 10 examples so do split on 9 mid points and find max information gain there

Stop split when num of examples in nodes are below threshold or u reached maximun depth of tree

generalization: model performing well with new data, is called generalize model

Play with cuda for machine learning, how gpu are going with machine learning

distortion function for k means alg cost function

You run K-means 100 times with different initializations. How should you pick from the 100 resulting solutions?

Pick randomly -- that was the point of random initialization.

If each example x is a vector of 5 numbers, then each cluster centroid

�

�

μ

k

is also going to be a vector of 5 numbers.

You run K-means 100 times with different initializations. How should you pick from the 100 resulting solutions? Pick the one with the lowest cost

if x is near to mean so its probablity is maximum and as it goes away means of standard deviation so probablity decreases

in normal distribution curve, it generate using meu and sigma, and both theese values tells about data set, and its area under curve always 1 so greater the devaition means liwer will be hight of curve to compensate the area, and there will be not probablity euqal to 1, it might be 0.8 or like that curves are there because of datasets and mue ad sigma,

greater the sigma lower the max hight so, obiviously showing that proablity can't reach equal to 1

for finding probablity for anamoly detection, if yoh have many much features, so find probablity for every feature and then multiply all them to find total probablity, this actulay use for statistically dependent features, but this algorithm work also well with not dependent feautres too

so we will be having as much normal distribution curves as number of features, for finding indivdual probability

Learn feature engineering for better ML/AI engineere

Anomoly detection is used for finding unlikely events on the other hand, supervised leanring is used for known events

auto diff and auto Grad in tensorfow and pytorch

Reinforcement Learning:

policy try to maximy the return and this way takes decision

in reinforement learning, replay buffer is called storing data into tuples

in recomendor system, mean normalisation m benefit yeh hua k mean ki waja s jis n koi movie rate ki ho ya nhi, bakiu k waja s us bndy k lye ratings predict ho jati hai

in content based recomendation system, we get recomendation as we like something

and in collaburative filtering algorithim we get recommendation by ratings or purchasing or liking of similar other users who are similar with me in ratings or purchasing or liking

In machine learning greater the training data greater the model accuracy but upto limit

But in deep learning greater the data greater will be accuracy of model

vectorization to avoid foorloop and faster the code

broadcasging in np, extend the size for +-x/ the second matrix for operation. broadcast is only in one direction not in two direction.

Another common technique we use in Machine Learning and Deep Learning is to normalize our data. It often leads to a better performance because gradient descent converges faster after normalization. Here, by normalization we mean changing x to ‖𝑥‖

(dividing each row vector of x by its norm).

Downside of sigmoid and tanH function is for very large and very small values the graph is horizontal and so the gradient become very small this slows the gradient descent. Thats why ReLU is being used. There is also leakyRelu which is not zero for negative value

Correct. The "cache" is used in our implementation to store values computed during forward propagation to be used in backward propagation.

If we initialize w and b in neuro network as zero it means k jitny m neuron hein ak layer m wo sb ak jesey hi hein and same output dengy

Phr to faida hi nhi Is lye randon initialize krty hein But b as zero will be OK cuz W Is making diference already. Initialize with random values multiply by 0.01

Question 7

Logistic regression’s weights should be initialized randomly rather than to all zeros, because if you initialize to all zeros, then logistic regression will fail to learn a useful decision boundary because it will fail to “break symmetry”, True/False?

False. Logistic regression's weights should typically be initialized to zeros or small random values, not all zeros. Initializing weights to all zeros can cause a symmetry problem where all weights update identically during training, resulting in suboptimal learning or failure to converge to a useful decision boundary. Random initialization helps break this symmetry.

Test error 1 %

dev error 11% means high variance

T E 15%

Dev E 16% means high bias cuz they close and error is high

T E 15 %

D E 30% is high V and High B

T E is 0.1 %

D E is 1%

Low B and Low V

One is regularization reduce overfitting

L2 regularization is used mostly not L1 is not

and for deep learning there are 2d W so l2 is replaced with frobenius norm

L2 is also called weight decay

one more technique to regularized is drop out

Data augmentation also used as regualrization technique, means increasing data by mirroring images and taking different angle and zoom position picture of images, without going out and taking pictures by your self

orthogonalization??

L1 is Lasso regularization

L2 is Ridge Regualrization

L1 and L2 is Elastic Net Regularization

how regularization works

if cost functionis zero so gradient will stop reducing w but by adding more value by increading lambda the cost function will stop late means more lesser w will be got

if weights are greater than 1 for deeper network, the values of w will go increase and if it between 0 and 1 they will get decrease for deeper network, see topic vansihing/ exploding gradients

Incorrect. In most cases, it is recommended to not use dropout if there is no overfit. Although in computer vision, due to the nature of the data, it is the default practice.

In Gradient descent there is all data set to find lose in every epoch, and it converge as possible straight

In Stochastic G D there is only one set in every epoch to find lose

End it converge in zig zag maner

So the change in Lose w.r.t W is all most equal

And this zig zag is called noisy data and to cure it there is concept called Stochastic with momentum

1 epoch means entire data set is trained by forward and backword

Iteration means one batch of entire set is trained by forward and backward prop

So means in one epoch there can be many iterations depends on num of batches and after one epoch we might have not get enough good weights so need more epochs to train model

In statistics, normalization typically refers to scaling data to have a mean of 0 and a standard deviation of 1, which is also known as z-score normalization. The formula to normalize a dataset using z-score normalization

Yes, in the problem you provided where you calculated the norm of a NumPy array x, you used a similar concept of normalization. However, the normalization performed in the given code is slightly different from the z-score normalization commonly used in statistics.

The np.linalg.norm() function calculates the Euclidean norm of the array x, which is essentially finding the magnitude or length of the vector formed by flattening the array. It doesn't directly perform the z-score normalization.

L1

loss = np.sum(np.abs(y - yhat))

L2

loss = np.sum((y - yhat)\*\*2)

Discriminiative aI ?

Deep Learning m Ak Descriptive Models

And

Doosra

Generative Models

The value of

is a hyperparameter that you can tune using a dev set.

L2 regularization makes your decision boundary smoother. If

is too large, it is also possible to "oversmooth", resulting in a model with high bias.

Divide [1]

by keep\_prob. By doing this you are assuring that the result of the cost will still have the same expected value as without drop-out. (This technique is also called inverted dropout.)

in DropOut neurorns

Dropout is a regularization technique.

You only use dropout during training. Don't use dropout (randomly eliminate nodes) during test time.

Apply dropout both during forward and backward propagation.

During training time, divide each dropout layer by keep\_prob to keep the same expected value for the activations. For example, if keep\_prob is 0.5, then we will on average shut down half the nodes, so the output will be scaled by 0.5 since only the remaining half are contributing to the solution. Dividing by 0.5 is equivalent to multiplying by 2. Hence, the output now has the same expected value. You can check that this works even when keep\_prob is other values than 0.5.

What you should remember: the implications of L2-regularization on:

The cost computation:

A regularization term is added to the cost.

The backpropagation function:

There are extra terms in the gradients with respect to weight matrices.

Weights end up smaller ("weight decay"):

Weights are pushed to smaller values.

Regularization will help you reduce overfitting.

Regularization will drive your weights to lower values.

L2 regularization and Dropout are two very effective regularization techniques.

In statistics, particularly in the context of machine learning and optimization, the norm often refers to a measure of the size or length of a vector. The Euclidean norm, also known as the L2 norm, is the most common norm used in these contexts.

vector1 = np.array([1, 2, 3])

vector2 = np.array([4, 5, 6])

# Computing Euclidean norm

norm\_vector1 = np.linalg.norm(vector1)

norm\_vector2 = np.linalg.norm(vector2

**Gradient Checking Reason**

You are part of a team working to make mobile payments available globally, and are asked to build a deep learning model to detect fraud--whenever someone makes a payment, you want to see if the payment might be fraudulent, such as if the user's account has been taken over by a hacker.

You already know that backpropagation is quite challenging to implement, and sometimes has bugs. Because this is a mission-critical application, your company's CEO wants to be really certain that your implementation of backpropagation is correct. Your CEO says, "Give me proof that your backpropagation is actually working!" To give this reassurance, you are going to use "gradient checking"

With Mini-batch GD or Mini-batch GD with Momentum, the accuracy is significantly lower than Adam, but when learning rate decay is added on top, either can achieve performance at a speed and accuracy score that's similar to Adam.

As seen in lecture, it is important that your dev and test set have the closest possible distribution to “real”-data. It is also important for the training set to contain enough “real”-data to avoid having a data-mismatch problem.

Changing the **training set distribution** is not a problem.

It's okay if the **training set and dev set** have different distributions.

**Dev and test sets must have the same distribution** for reliable evaluation.

If dev and test distributions differ, model performance on the test set may be inaccurate.

Cat boast is good algorithm for recommendation system. It is also for gradient boasting for decision trees. upgini python libraries for finding features automatically

**CNN**

Suppose your input is a 300 by 300 color (RGB) image, and you use a convolutional layer with 100 filters that are each 5x5. How many parameters does this hidden layer have (including the bias parameters)?]

To calculate the number of parameters in a convolutional layer, you need to consider the following:

Each filter in the convolutional layer has its own set of weights.

There is a bias term associated with each filter.

Given that the input is a 300x300 RGB image (3 channels), and we have 100 filters that are each 5x5, we can calculate the number of parameters as follows:

Each filter has a size of 5x5x3 (height x width x channels), resulting in

5×5×3=75 weights per filter.

Each filter also has one bias parameter.

Therefore, the total number of parameters for the convolutional layer is:

Number of parameters=(weights per filter+bias)×number of filters

Number of parameters=(75+1)×100=76×100=7600 parameters

 Suppose your input is a 128 by 128 grayscale image, and you are not using a convolutional network. If the first hidden layer has 256 neurons, each one fully connected to the input, how many parameters does this hidden layer have (including the bias parameters)?

If each neuron in the first hidden layer is fully connected to the input, it means that each neuron has a connection to each pixel in the 128x128 input image. Since the image is grayscale, each pixel has a single value.

Therefore, each neuron in the hidden layer has 128 \* 128 weights (connections) associated with it, plus one bias parameter.

So, the total number of parameters for each neuron in the hidden layer is:

128 \* 128 (weights) + 1 (bias) = 16385 parameters.

Since there are 256 neurons in the hidden layer, the total number of parameters for the hidden layer is:

256 \* 16385 = 4194560 parameters.

You have an input volume that is 63x63x16, and convolve it with 32 filters that are each 7x7, using a stride of 2 and no padding. What is the output volume?

To calculate the output volume after convolution, we can use the following formula

Output volume size=(input volume size−filter size stride)+1

Given:

Input volume size: 63x63x16

Filter size: 7x7

Stride: 2

Using the formula:

Output volume size=(63−72)+1=(56 2)+1=28+1=29

Output volume size=( 263−7)+1=( 256)+1=28+1=29

So, the output volume size is 29x29x32, where 32 is the number of filters applied.

When padding is applied to an input volume, it adds additional rows and columns around the edges of the volume.

Input volume size: 15x15x8

Padding size: 2

For each dimension (height, width, and depth), we add 2 units of padding on both sides. Therefore, the resulting dimensions of the padded volume will be:

Height: 15+2 (padding on top)+2 (padding on bottom)=15+4=19

Width: 15+2 (padding on left)+2 (padding on right)=15+4=19

Depth: The depth remains unchanged since padding doesn't affect it.

So, the resulting dimension of the padded volume will be 19x19x8.

To achieve **"same" convolution**, the output volume size should be the same as the input volume size. We determine the required **padding** using the formula:

Output size=((Input size+2×Padding−Filter size)/Stride)+1

**Given Values:**

* **Input size** = 64
* **Filter size** = 9
* **Stride** = 1
* **Output size** = 64 (same as input)

**Substituting the values:**

64=64+2×Padding−91+164 = \frac{64 + 2 \times \text{Padding} - 9}{1} + 164=164+2×Padding−9​+1

**Simplifying:**

64=64+2×Padding−9+164 = 64 + 2 \times \text{Padding} - 9 + 164=64+2×Padding−9+1 64=64+2×Padding−864 = 64 + 2 \times \text{Padding} - 864=64+2×Padding−8 2×Padding=82 \times \text{Padding} = 82×Padding=8 Padding=4\text{Padding} = 4Padding=4

**Conclusion:**

To maintain the **same output size (64×64×40)** with a **9×9 filter** and **stride of 1**, we need to apply **a padding of 4**. ✅

Convolutional layers exhibit translation invariance, meaning they can detect features regardless of their exact location in the input image.

You have an input volume that is 32x32x16, and apply max pooling with a stride of 2 and a filter size of 2. What is the output volume?

To calculate the output volume after max pooling, we use the following formula:

Output volume size=(input volume size−filter size stride )+1

Output volume size=( stride input volume size−filter size )+1

Given:

Input volume size: 32x32x16

Filter size: 2

Stride: 2

Using the formula:

Output volume size= (32−22)+1=(30/2+1)=15+1==16

So, the output volume size is 16x16x16.

"Sparsity of connections" refers to the characteristic of convolutional layers where each neuron in the layer is connected only to a small region of the input volume, rather than being fully connected to all neurons in the previous layer. This sparsity arises from the use of filters (also known as kernels) in convolutional layers.

In a convolutional layer, each filter is applied to a small region of the input volume, and its weights are shared across all spatial locations within that region. As a result, each neuron in the convolutional layer is only affected by a local region of the input volume. This local connectivity pattern leads to sparsity in the connections between neurons, as compared to fully connected layers where each neuron is connected to all neurons in the previous layer.

The sparsity of connections in convolutional layers offers several benefits:

Parameter Sharing: By sharing weights across different spatial locations, convolutional layers have fewer parameters compared to fully connected layers, leading to a more efficient use of memory and computation.

Translation Invariance: Convolutional layers are able to detect features regardless of their exact location in the input volume, making them robust to translations in the input data.

Feature Learning: The local connectivity pattern allows convolutional layers to learn hierarchical representations of features from the input data, capturing patterns and spatial relationships effectively.

Overall, the sparsity of connections in convolutional layers contributes to their effectiveness in various tasks such as image classification, object detection, and image segmentation.

The pooling (POOL) layer reduces the height and width of the input. It helps reduce computation, as well as helps make feature detectors more invariant to its position in the input.

Suppose that in a MobileNet v2 Bottleneck block we have an

�×�×5

n×n×5 input volume, we use

30

30 filters for the expansion, in the depthwise convolutions we use

3×3 filters, and

20 filters for the projection. How many parameters are used in the complete block, suppose we don't use bias?

Incorrect. When adding a ResNet block it can easily learn to approximate the identity function, thus in a worst-case scenario, it will not affect the performance of the network at all.

The speed of learning decreases very rapidly for the shallower layers as the network trains

Large dataset mein

Skip gram

And small dataset m CBOW

Prefere krty

Due to dying Relu Problem

All negative input is outputting zero, which is flat graph, and all flat graphs derivative is zero, means no learning in backpropagation it provides

Thats why Leaky Relu is introduced

The sigmoid is not used in hidden layers because, it has flat graph at value 0 and 1, which is similar again problematic in backpropagation causing Vanishing gradient and saturation problem. Tanh has also same problem.

What is covariant shift?

Means k training set ka mean and std nkl jata ak dafa and wohi use hota always for testing. But training k waqt hr dafa scaling factor and shift factor apy krny k bd activation m jata data...

Convolutiin apply learned filters to detect features

Padding q krty, Reason GANS course 1 week 2 k ander padding ki video m ha

Islye use krty k, hr pixel ko equal.importance mly, q k filter apply krty we, filters ziadah center pixels ko use krty, padding k bd corners walay center ki trf ho jatay to woh bhi ziadah dafa convolution filters s interact krty

Pooling decrease the size of image and blur it and gets its featutes which are most important

Unsampling is opposite, its increases the size of the image

Transpose convolution is also enlarge the size

But it is learnable feature on the other hand un-sampling is predefined method

Mode collapse is generator stuck to one local minima or stuck in one mode

In Transformer huggingface heads are use to convert the ***hidden states/ features*** into ***contextual understanding.***

*print*(outputs.logits.shape)

(all 🤗 Transformers models output the logits, as the loss function for training will generally fuse the last activation function, such as SoftMax, with the actual loss function, such as cross entropy):

*import* torch

predictions = torch.nn.functional.softmax(outputs.logits, dim=-1)

*print*(predictions)

To get the labels corresponding to each position, we can inspect the id2label attribute of the model config (more on this in the next section):

model.config.id2label

model.save\_pretrained("directory\_on\_my\_computer")

This saves two files to your disk:

ls *directory\_on\_my\_computer*

*config*.*json* pytorch\_model.*bin*

Translating text to numbers is known as encoding. Encoding is done in a two-step process: the tokenization, followed by the conversion to input IDs.

There are word based tokenizer and then character based tokenizer then better way is subword tokenizer.

This tokenizer is a subword tokenizer: it splits the words until it obtains tokens that can be represented by its vocabulary.

Unsurprisingly, there are many more techniques out there. To name a few:

* Byte-level BPE, as used in GPT-2
* WordPiece, as used in BERT
* SentencePiece or Unigram, as used in several multilingual models

*Attention masks* are tensors with the exact same shape as the input IDs tensor, filled with 0s and 1s: 1s indicate the corresponding tokens should be attended to, and 0s indicate the corresponding tokens should not be attended to (i.e., they should be ignored by the attention layers of the model).

Let’s complete the previous example with an attention mask:

batched\_ids = [

[200, 200, 200],

[200, 200, tokenizer.pad\_token\_id],

]

attention\_mask = [

[1, 1, 1],

[1, 1, 0],

]

outputs = model(torch.tensor(batched\_ids), attention\_mask=torch.tensor(attention\_mask))

*print*(outputs.logits)

tensor([[ 1.5694, -1.3895],

[ 0.5803, -0.4125]], grad\_fn=<AddmmBackward>)

### **Blue Benchmark NLP – Explained Super Simply**

Alright, imagine you have different AI models that can read and understand text. Some might be good at answering questions, some at summarizing, and others at translating languages. But how do we know which model is actually the best? 🤔

That’s where **Blue Benchmark NLP** comes in! It's a way to **compare and rank NLP (Natural Language Processing) models** based on how well they perform on different tasks.

### **Why Does It Matter?**

* If you're building a chatbot, you’d want the best model for answering questions.
* If you need a translation tool, you'd want the most accurate translator.
* If you want AI to summarize books, you’d pick a model that gives the best summaries.

So, Blue Benchmark NLP is like a **report card** 📜 for AI models, helping people choose the best one for their tasks.

### **How does it Evaluate NLP Models?**

It ranks models based on:  
✅ **Accuracy** – How well the model understands and generates text  
✅ **Efficiency** – How fast the model runs on different hardware  
✅ **Robustness** – How well it performs on diverse datasets  
✅ **Fairness** – Avoiding biases in predictions

It provides a **scorecard** for various NLP models (like GPT, BERT, T5, Llama, etc.), helping researchers and developers choose the **best** model for their specific needs.

### **Scenario 1: Choosing the Best Model for Your Task**

Let’s say you’re building a **chatbot** for customer service. You want to pick the best NLP model for answering customer queries.

🔹 Instead of **randomly picking** a model, you can check the **Blue Benchmark NLP leaderboard** and see which model performs best for **question-answering tasks**.

📌 **Example of checking a benchmark leaderboard:**  
You can visit **Hugging Face’s Leaderboard** or **Papers With Code** to see how models rank on different tasks.

### **Scenario 2: Comparing Performance of Different Models**

Imagine you’re working on a **text summarization project**. You can test multiple models and compare their **speed, accuracy, and response quality** using Blue Benchmark.

#### ****Steps to compare models yourself:****

1. **Pick 2 or more models** from Hugging Face (e.g., google/pegasus-xsum vs. facebook/bart-large-cnn).
2. **Use the same dataset** (like CNN/DailyMail for summarization).
3. **Run them on the same text input** and compare:
   * How well they summarize
   * How accurate the summary is
   * How fast they generate responses

### **Scenario 3: Automating Benchmarking in Code**

If you want to evaluate an NLP model programmatically, you can use the **Hugging Face evaluate library**.

#### ****Example: Comparing BERT and GPT on Text Classification****

from transformers import pipeline

from evaluate import load

bert\_model = pipeline("text-classification", model="nlptown/bert-base-multilingual-uncased-sentiment")

gpt\_model = pipeline("text-generation", model="gpt2")

sentence = "I love this product! It's amazing."

bert\_result = bert\_model(sentence)

gpt\_result = gpt\_model(sentence, max\_length=30)

print("BERT Prediction:", bert\_result)

print("GPT Prediction:", gpt\_result)

👉 **You can compare accuracy, speed, and efficiency** to decide which model to use.

## **Why is GLUE Important?** 🤔

Before GLUE, AI models were often tested on just **one task**, which wasn’t enough to check if they truly understood text. GLUE fixes that by testing models on **multiple real-world language tasks**.

Think of it like an **exam** for AI models, where they get a score at the end. **Higher scores mean a better model.**

### **Blue Benchmark NLP vs. GLUE Benchmark – What’s the Difference?**

Both **Blue Benchmark NLP** and **GLUE Benchmark** are used to evaluate **NLP models**, but they focus on **different goals**. Let’s break it down simply.

## **1. GLUE Benchmark – Testing Language Understanding** 🧠

👉 **Goal:** Measures how well a model **understands text** through various real-world NLP tasks.

✅ **Example:**  
Think of GLUE like a **school exam** 📚. It tests an AI model in different subjects (tasks) like:

* **Sentiment Analysis** (Is a review positive or negative?)
* **Text Similarity** (Are two sentences saying the same thing?)
* **Natural Language Inference (NLI)** (Does one sentence logically follow from another?)

📌 **Used for:** General-purpose NLP models like **BERT, RoBERTa, T5, GPT, etc.**

## **2. Blue Benchmark NLP – Ranking and Comparing Models** 📊

👉 **Goal:** Ranks NLP models based on performance in different categories (accuracy, efficiency, robustness, fairness).

✅ **Example:**  
Think of Blue Benchmark like a **sports league** 🏆. It takes multiple AI models and ranks them **against each other** to see which one is the best for different tasks (summarization, Q&A, translation, etc.).

📌 **Used for:** Finding the **best-performing** model for a specific NLP task.

**This command downloads and caches the dataset, by default in**~/.cache/huggingface/datasets**. Recall from Chapter 2 that you can customize your cache folder by setting the HF\_HOME environment variable**.

## **1. NLP Benchmarks 📖🧠 (Natural Language Processing)**

| **Benchmark** | **What It Evaluates** | **Best for Selecting...** |
| --- | --- | --- |
| **GLUE** 🧠 | General NLP understanding (Sentiment, Text Similarity, NLI, etc.) | Models for text classification, sentence similarity, and inference tasks |
| **SuperGLUE** 🔥 | Harder NLP tasks (requires reasoning, common sense) | More advanced models (BERT, T5, GPT, etc.) |
| **MMLU** 📚 (Massive Multitask Language Understanding) | Knowledge-based QA across multiple subjects | LLMs like GPT, Claude, Gemini |
| **HELLO-SWEET** 🍬 | Hallucination detection in LLMs | Finding the most reliable LLMs |
| **MT-Bench** 🏆 | AI model evaluation through human-like conversations | Selecting the best chatbot/LLM for dialogue-based tasks |
| **BLUE Benchmark NLP** 📊 | Ranking LLMs based on fairness, robustness, efficiency | Choosing the most practical LLM for real-world deployment |
| **BIG-Bench** 🏛️ | Reasoning & general intelligence in LLMs | Comparing large AI models (GPT-4, Gemini, Claude, etc.) |

🔥 **Where to check?**

* Hugging Face Open LLM Leaderboard
* Papers With Code NLP Benchmarks

## **2. Computer Vision Benchmarks 🖼️🔍 (Image & Video Processing)**

| **Benchmark** | **What It Evaluates** | **Best for Selecting...** |
| --- | --- | --- |
| **ImageNet** 🏞️ | Object classification in images | Best models for image classification |
| **COCO** 🦾 | Object detection & segmentation | Best models for detecting objects in images |
| **Open Images** 🌍 | Large-scale object detection with detailed annotations | Models for real-world image recognition |
| **Cityscapes** 🏙️ | Street scene segmentation | Self-driving car vision models |
| **ADE20K** 🏡 | Scene understanding & segmentation | General-purpose segmentation models |
| **Kinetics-700** 🎥 | Action recognition in videos | Best models for video understanding |
| **LFW (Labeled Faces in the Wild)** 👦 | Face recognition | Selecting the best face recognition models |

🔥 **Where to check?**

* Papers With Code CV Benchmarks
* Hugging Face Model Leaderboard

## **3. Multi-Modal Benchmarks 🎭 (Text + Image + Audio Models)**

| **Benchmark** | **What It Evaluates** | **Best for Selecting...** |
| --- | --- | --- |
| **MMBench** 🎭 | Vision + Language model evaluation | Selecting multimodal LLMs (GPT-4V, Gemini, LLaVA) |
| **VQAv2** 🖼️📖 | Visual Question Answering | Best models for answering questions about images |
| **LAION-400M** 🏞️🔤 | Text-to-image retrieval | Choosing models for multimodal retrieval |
| **AudioSet** 🎧 | Audio event detection | Best models for sound classification |

🔥 **Where to check?**

* Multi-Modal Leaderboard on Hugging Face

## **4. Reinforcement Learning Benchmarks 🎮🤖**

| **Benchmark** | **What It Evaluates** | **Best for Selecting...** |
| --- | --- | --- |
| **Atari-57** 🕹️ | Playing Atari games using AI | RL models for games |
| **MuJoCo** 🤸 | Simulated physics-based environments | Robotics and control-based RL |
| **Procgen** 🎮 | Generalization in RL tasks | RL models that adapt to new environments |

🔥 **Where to check?**

* [OpenAI Gym Benchmarks](https://gym.openai.com/)

### **Step 1: Identify Your NLP Task 📌**

Each NLP task has a **specific benchmark** designed to test how well models perform on it.

| **NLP Task** | **Best Benchmark to Use** | **Why This Benchmark?** |
| --- | --- | --- |
| **Text Classification** 📂 | **GLUE, SuperGLUE** | Tests sentiment, spam detection, topic categorization |
| **Sentiment Analysis** 😊😡 | **GLUE (SST-2)** | Evaluates how well models detect emotions in text |
| **Question Answering (QA)** ❓ | **SQuAD, MMLU** | Measures how well models answer questions from a passage |
| **Text Summarization** 📄➡️📝 | **BIG-Bench, HELM** | Tests how well models summarize long documents |
| **Named Entity Recognition (NER)** 🔍 | **CoNLL-2003** | Tests recognition of names, places, organizations |
| **Text Generation (Chatbots, LLMs)** 🤖 | **MT-Bench, HELLO-SWEET** | Ranks models like GPT, Claude, and Gemini for response quality |
| **Machine Translation** 🌍🗣️ | **WMT Benchmark** | Evaluates models on translating text between languages |
| **Text Similarity / Paraphrasing** 🔄 | **STS Benchmark (Semantic Textual Similarity)** | Tests whether two sentences mean the same thing |
| **Hallucination Detection in LLMs** 🚨 | **HELLO-SWEET, MT-Bench** | Checks if models generate fake/misleading info |