**Why do we need to explicitly specify loss functions in Deep Learning but not always in Machine Learning?**

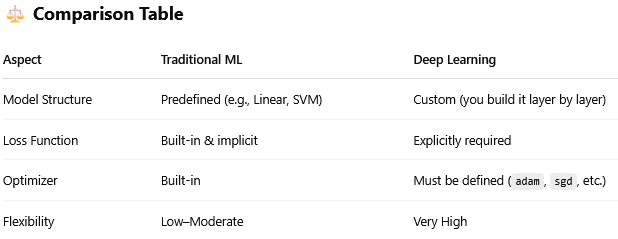
--------------------------------------------------------------------------------------------------------------

A: In traditional ML libraries like scikit-learn, you often don’t need to explicitly specify the loss function, because:

* The loss function is built into the algorithm itself.
* For example:
  + LinearRegression() uses Mean Squared Error (MSE) internally.
  + LogisticRegression() uses Log Loss (Cross-Entropy).
  + SVM uses Hinge Loss.
* These algorithms have predefined objectives, and the library handles optimization under the hood.

In deep learning, models are custom architectures (layers, activation functions, etc.), so you must specify the loss function.

* Deep learning frameworks are low-level and highly customizable.
* The model is not tied to any specific task by default (like classification or regression).
* So, you must define what it means to “make a good prediction.” That’s the loss function.



🎯 Final Thought:

* In ML: You use high-level models where the loss and optimization are abstracted away.
* In Deep Learning: You’re in charge of everything, so you must define the loss function.

**Q: What do we specify in model.compile()?**

----------------------------------------------------------------------------------------------

📌 Full Syntax:

*model.compile(*

*optimizer=...,*

*loss=...,*

*metrics=...,*

*loss\_weights=..., # Optional*

*weighted\_metrics=..., # Optional*

*run\_eagerly=..., # Optional (for debugging)*

*jit\_compile=... # Optional (for performance via XLA)*

*)*

----------------------------------------------------------------------------------------------

1. optimizer (✅ Required): This tells the model how to update weights based on gradients.

* String or optimizer object:

*optimizer='adam'*

*# or*

*optimizer=tf.keras.optimizers.SGD(learning\_rate=0.01)*

2. loss (✅ Required): Defines how the model’s prediction is evaluated — it measures the "error".

* Examples:

*loss='categorical\_crossentropy'*

*loss='sparse\_categorical\_crossentropy'*

*loss='mse' # Mean Squared Error for regression*

3. metrics (✅ Optional but Recommended): Metrics to evaluate the performance during training & validation.

* Examples:

*metrics=['accuracy']*

*metrics=['mae', 'mse'] # for regression*

4. loss\_weights (Optional): Used when your model has multiple outputs — you can assign importance (weights) to each output’s loss.