

3.1 Lesson: Monitoring Training Using Learning Curves



Monitoring Training Using Learning Curves

In Module 3, we created validation curves to study how performance metrics such as the Mean Squared Error (MSE) changed as we varied a specific hyperparameter (e.g., the depth of a decision tree). These curves helped us select hyperparameter values to build models that generalize well.

In deep learning, we typically track performance over training epochs, rather than over hyperparameter ranges one at a time. The resulting plots — showing how metrics like loss or accuracy evolve in the training and validation sets over the epochs — are called **learning curves**.

Using learning curves, we can identify several scenarios familiar from Module 3:

Underfitting:

Both training and validation loss remain high. This usually means the model is too simple (not enough capacity) or hasn't trained long enough.

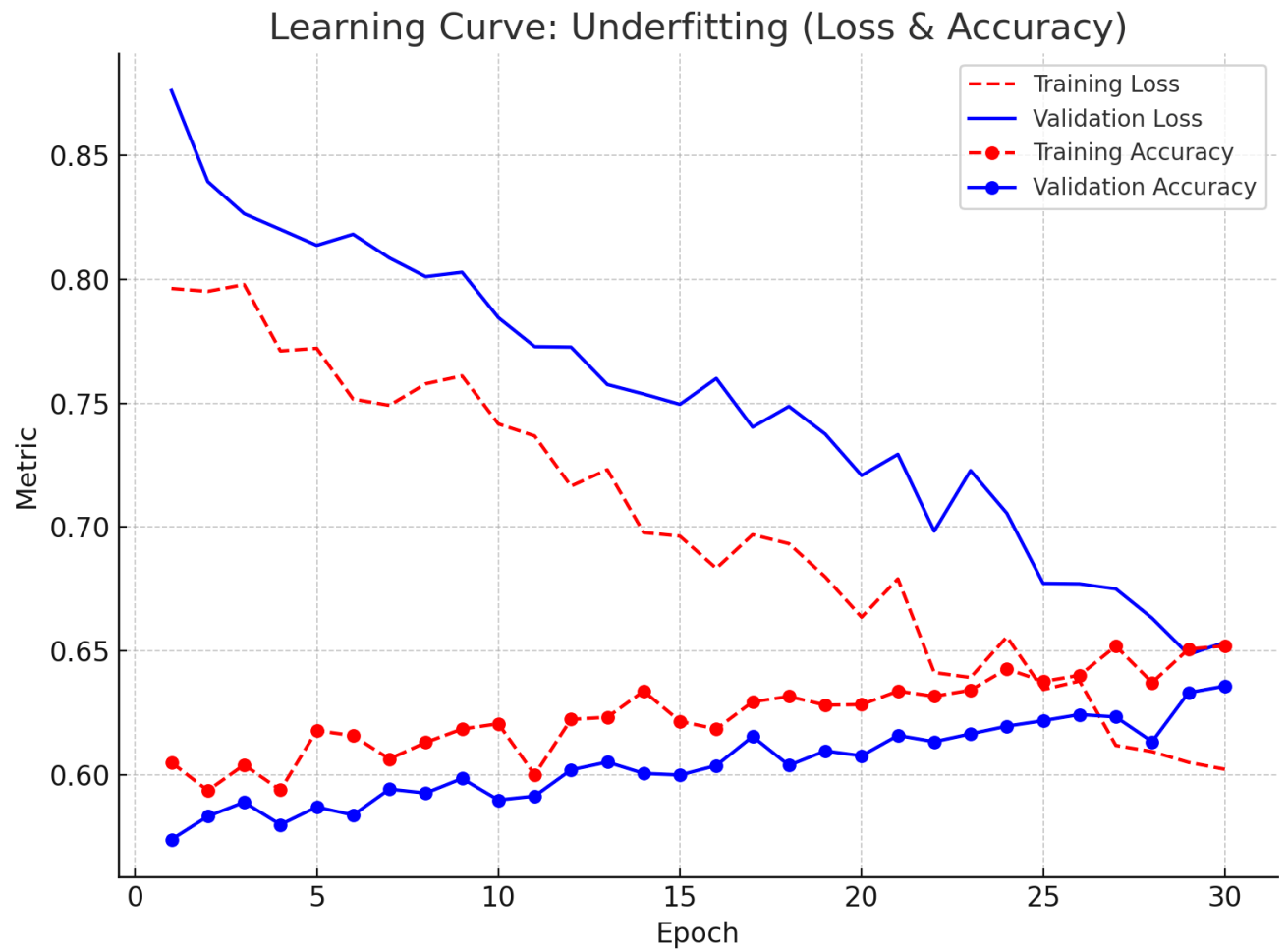


Figure 1. Learning Curve: Underfitting (Loss & Accuracy)

Source: OpenAI. (2025). *ChatGPT* (May version) [Large language model]. <https://chat.openai.com/>

Good Fit:

Training loss decreases steadily, and validation loss follows a similar pattern. At some point, both stabilize at low values.

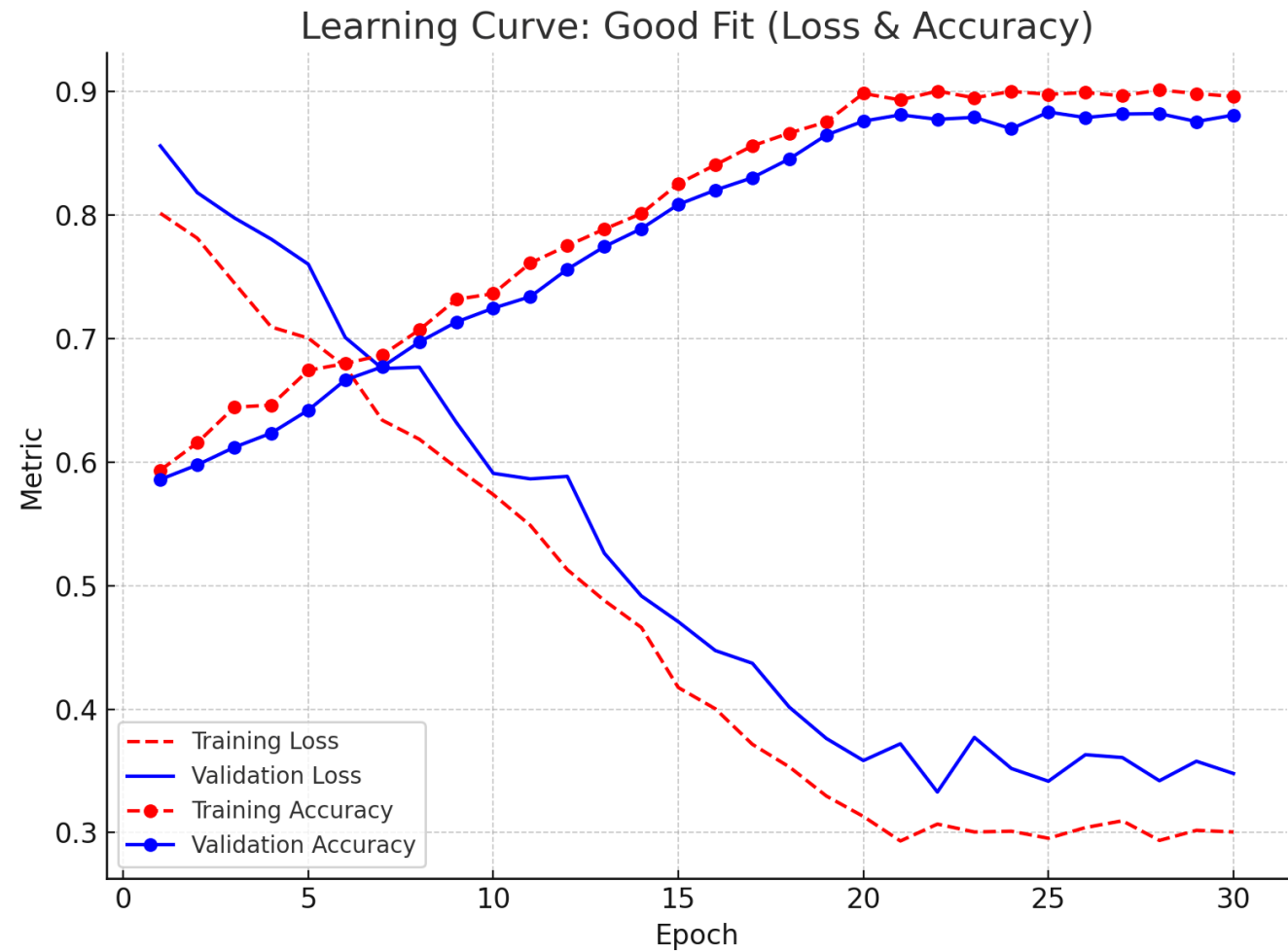


Figure 2. Learning Curve: Good Fit (Loss & Accuracy)

Source: OpenAI. (2025). *ChatGPT* (May version) [Large language model]. <https://chat.openai.com/>

Overfitting:

Training loss continues to decrease, while validation loss either starts to rise or stops

improving, causing the gap between them to widen. This indicates that the model is starting to memorize the training data rather than learning general patterns that apply to new data.

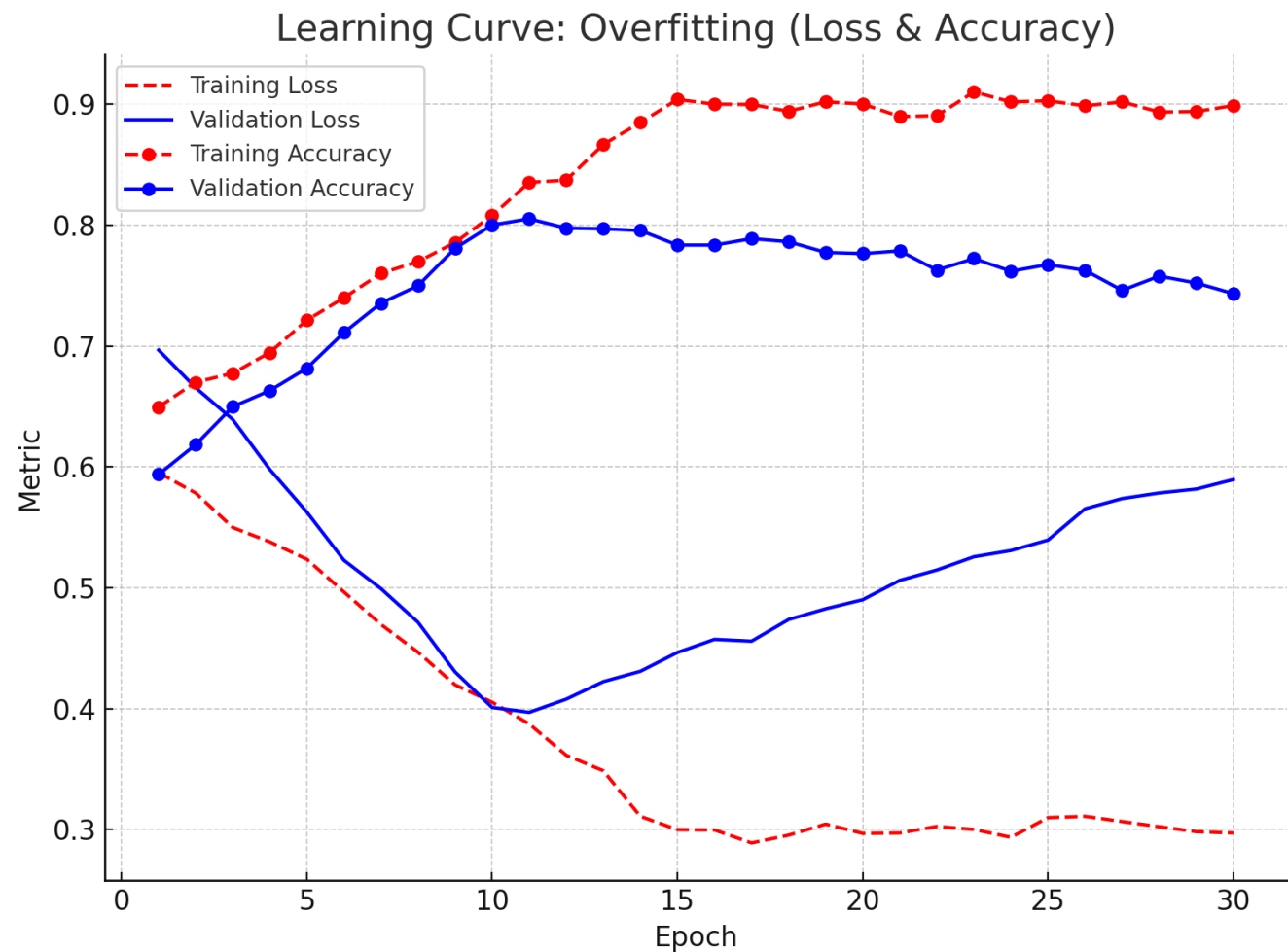


Figure 3. *Learning Curve: Overfitting (Loss & Accuracy)*

Source: OpenAI. (2025). *ChatGPT* (May version) [Large language model].

Understanding where you are on this curve is essential to deciding what to do next — whether to train longer, modify the hyperparameters (including the network architecture), or adjust the learning rate and optimizer settings to improve convergence. The important takeaway compared with Module 3 is that for deep learning, **learning curves tracking performance vs. length of training replace validation curves tracking performance vs. hyperparameter values.**

K-Fold Cross-Validation or Static Validation Sets?

When tracking model performance using learning curves in deep learning, it's important to consider the computational cost of K-fold cross-validation. Because K-fold CV requires training K separate models per evaluation, applying it during each training epoch becomes prohibitively expensive in most deep learning contexts (and repeated K-fold CV would make this even worse). As a result, practitioners typically prefer using a **single static validation set** for hyperparameter tuning and early stopping, followed by evaluation on a separate held-out test set to estimate final performance.

Note: The size of the validation set is specified as a parameter of Keras's **fit** method (demonstrated in the Coding Video and notebook).

However, cross-validation can still be valuable in certain situations:

- **Small datasets:** When data is scarce (e.g., in medical imaging), performance estimates tend to have high variance due to the small sample size amplifying fluctuations — a phenomenon explained by the central limit theorem. K-fold CV helps reduce this variance by averaging results across multiple training-validation splits, yielding a more reliable estimate of model performance.
- **Final model evaluation:** Once your model architecture and hyperparameters are finalized, cross-validation can serve as a sanity check, confirming that performance is stable across different data splits before deployment.

Early Stopping

In the previous section, we saw that a key warning sign of overfitting is when the validation loss starts increasing even though the training loss continues to decrease. One of the simplest and most effective ways to prevent overfitting is through early stopping. We introduced this technique in Module 3 but did not emphasize its use; in deep learning, it is an important tool in your toolbox.

Early stopping stops training once the model's performance on the validation set stops improving. The idea is simple: Don't keep training if the model is no longer getting better at generalizing to new data.

This helps in two ways:

- **Saves time:** You don't waste epochs improving training loss without improving validation loss.
- **Improves generalization:** You stop before the model starts overfitting.

How It Works

During training, you monitor a metric on the validation set — usually **validation loss**, but sometimes **validation accuracy** or another custom metric. If that metric doesn't improve for a while, you stop training.

This process relies on the same two parameters we saw in Module 3:

- **patience:** How many epochs to wait for improvement before stopping. For example, if $\text{patience}=5$, the training will stop if the validation metric hasn't improved in 5 epochs.
- **min_delta:** The minimum amount of improvement required to reset the patience counter. If the

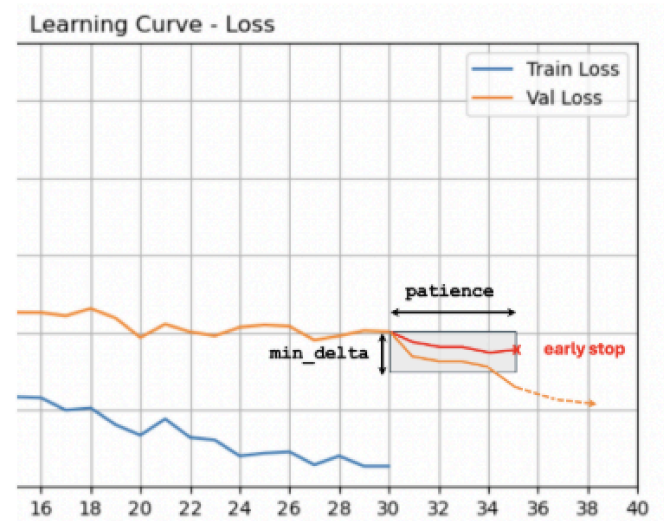


Figure 4. *Early Stopping*

metric improves
by less than
 min_delta , it
doesn't count
as
improvement.