Optimize Neural Networks- Factors affecting NN

1. Model Architecture Optimization

Your network architecture dictates what the model can learn. A poorly chosen architecture may underfit (too simple) or overfit (too complex).

- **Start Simple**: Begin with a baseline model and measure its performance.
- **Architectures to Know**:
  - **CNNs** (e.g., ResNet, VGG, EfficientNet): Good for images.
  - **RNNs / LSTMs / GRUs**: Good for time-series or sequences.
  - **Transformers (e.g., BERT, ViT, GPT)**: Excellent for NLP and image classification with large data.

2. Hyperparameter Tuning

Hyperparameters control learning behavior and directly affect training time and final accuracy.

Key Hyperparameters:

Hyperparameter	Role	Common Ranges
Learning rate	Controls update step	1e-4 to 1e-1
Batch size	Controls gradient noise	16 to 512
Dropout rate	Regularization	0.1 to 0.5
Optimizer	How gradients are applied	Adam, SGD, RMSprop

- **Manual tuning** (trial and error).
- **Grid search**: Try all combinations — slow but systematic.
- **Random search**: More efficient than grid.
- **Bayesian Optimization**: Smarter, learns from past trials.

3. Training Techniques

Proper training strategies can make or break your network's ability to converge and generalize.

Techniques to Use:

- **Early stopping**: Stop training when validation loss doesn't improve for X epochs.
- **Learning rate scheduling**: Reduce LR every few epochs.
- **Batch normalization**: Normalizes activations to stabilize learning.
- **Gradient clipping**: Keeps gradient magnitude in check (important in RNNs).

Learning rate reducing/scheduling:

The **learning rate** controls how much the model updates its weights in response to the error it sees. If it's **too high**, training becomes unstable. If it's **too low**, training becomes slow or stuck.

`ReduceLROnPlateau` is a **Keras callback** that **automatically reduces the learning rate** when the model's performance **stops improving** (i.e. plateaus).

If the **validation loss** (or accuracy) hasn't improved for a few epochs, we reduce the learning rate by a **factor**.

- **Prevents overshooting** near minima.
- **Improves convergence** when the learning plateaus.
- **Automates fine-tuning** of learning rate during training.

Parameter	Description
`monitor`	What to watch (e.g., `'val_loss'`, `'val_accuracy'`)
`factor`	Factor to reduce learning rate (e.g., `0.5`)
`patience`	How many epochs to wait before reducing
`min_lr`	Lower bound on learning rate
`cooldown`	Wait time after LR is reduced before resuming monitoring

This is useful when the model gets stuck at a local minimum — a smaller learning rate can help it fine-tune further.

Early stopping:

Early Stopping is a regularization technique used to prevent **overfitting** in neural networks by **halting training when the model stops improving** on the validation set.

Rather than training for a fixed number of epochs, we let the model **decide when to stop** based on validation performance.

- **Training loss** typically decreases.
- **Validation loss** may decrease at first, but can increase later (overfitting begins).
- **Early stopping halts training at the point where validation performance stops improving**, preventing the model from "memorizing" the training data.

How It Works:

1. Monitor a metric (usually `val_loss` or `val_accuracy`).
2. Define **patience** — how many epochs to wait for improvement.
3. If no improvement after `patience` epochs, training stops.
4. Optionally, **restore the best weights** from before performance dropped.

Parameter	Purpose
`monitor`	Metric to track (`val_loss`, `val_accuracy`, etc.)
`patience`	Number of epochs with no improvement before stopping
`min_delta`	Minimum change to qualify as an improvement (default is `0`)
`restore_best_weights`	If `True`, model returns to the best weights seen during training
`mode`	`'auto'`, `'min'`, or `'max'` depending on whether you want to minimize or maximize the metric
`verbose`	Set to `1` to print messages when early stopping is triggered

Model Check pointing:

Model checkpointing is a technique used during the training of machine learning models to **periodically save the model's state**—including its parameters, optimizer state, and sometimes other metadata—so that training can resume later or the model can be restored for inference or evaluation.

Why Use Check pointing?

1. **Fault Tolerance**: If training is interrupted (e.g., due to system failure), you can resume from the last checkpoint instead of starting over.

2. **Best Model Selection**: You can save the model with the **best validation accuracy or lowest loss**, not just the final state.
3. **Training Analysis**: Checkpoints allow you to analyze model performance at different training stages.
4. **Incremental Improvements**: Continue training later or fine-tune from a particular checkpoint.

4. Regularization to Prevent Overfitting

Overfitting happens when your model learns noise instead of patterns. Regularization helps generalize.

- **Dropout**: Randomly "drops" neurons during training.
- **Weight decay (L2 regularization)**: Penalizes large weights.
- **L1 regularization**: Encourages sparsity.
- **Data augmentation**: Increase training data variety (e.g., rotate/flip images, paraphrase text).

5. Data Quality & Pre-processing

Better data = better model.
Garbage in = garbage out.

- **Clean your data**: Remove duplicates, fix mislabeled examples.
- **Balance classes**: Use SMOTE, oversampling, undersampling for class imbalance.
- **Feature scaling**: Normalize (e.g., MinMax, StandardScaler).
- **Augmentation**: Random cropping, flipping, noise injection, etc.
- **Tokenization**: For text, use BERT tokenizer or SentencePiece for subword units.

6. Evaluation and Debugging

Training loss going down ≠ good model. You need robust evaluation.

- **Cross-validation**: Evaluate performance on multiple data splits.
- **Confusion matrix**: For classification.
- **Precision, Recall, F1-score**: For imbalanced classes.
- **Learning curves**: Visualize training vs. validation loss.
- **Explainability**:
    - **Grad-CAM**: For CNNs.
    - **SHAP / LIME**: For tabular or NLP models.