Early Stopping & Data Scaling

Early Stopping

`from keras.` `callbacks` `import` `EarlyStopping`

`model.` `fit` `(` `callbacks` `=[early_stopping] )`

- **Early Stopping** is a regularization technique that halts training when the model stops improving on a **validation set**, **preventing overfitting and saving computation time**.

📌 How Early Stopping Works

1. **Monitor Validation Metric** (e.g., `val_loss`, `val_accuracy`).
2. **Stop Training** if the metric doesn't improve for `N` epochs (`patience`).
3. **Restore Best Weights** (optional) to the epoch with the optimal performance.

Key Parameters:

Parameter	Description	Recommended Value
`monitor`	Metric to track (e.g., `val_loss`, `val_accuracy`)	`val_loss` (default)
`patience`	Epochs to wait before stopping	`5-10`
`restore_best_weights`	Revert to best model weights	`True`
`min_delta`	Minimum change to qualify as improvement	`0.001`
`mode`	`auto` / `min` / `max` (e.g., minimize loss or maximize accuracy)	`auto`

Implementation in Keras

```
from keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(
    monitor='val_loss',  # Track validation loss
    patience=5,         # Stop after 5 epochs without improvement
    restore_best_weights=True,  # Revert to best weights
    min_delta=0.001     # Minimum change to count as improvement
)

model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=100,
    callbacks=[early_stopping]  # Add to training
)
```

- `monitor`: What metric to monitor (`'val_loss'`, `'val_accuracy'`, etc.).
- `patience`: The number of epochs to wait for improvement before stopping.
- `min_delta`: Minimum change in the monitored metric to count as an improvement.
- `restore_best_weights`: Ensures the model returns to the weights from the epoch with the best performance on the validation set.

When to Use Early Stopping

Early stopping is especially useful when:

- You don't have a large amount of training data.
- You're working with deep learning models that have a risk of overfitting due to their complexity.
- You want to ensure that your model generalizes well without manually tuning for too many epochs.

📊 Why Use Early Stopping?

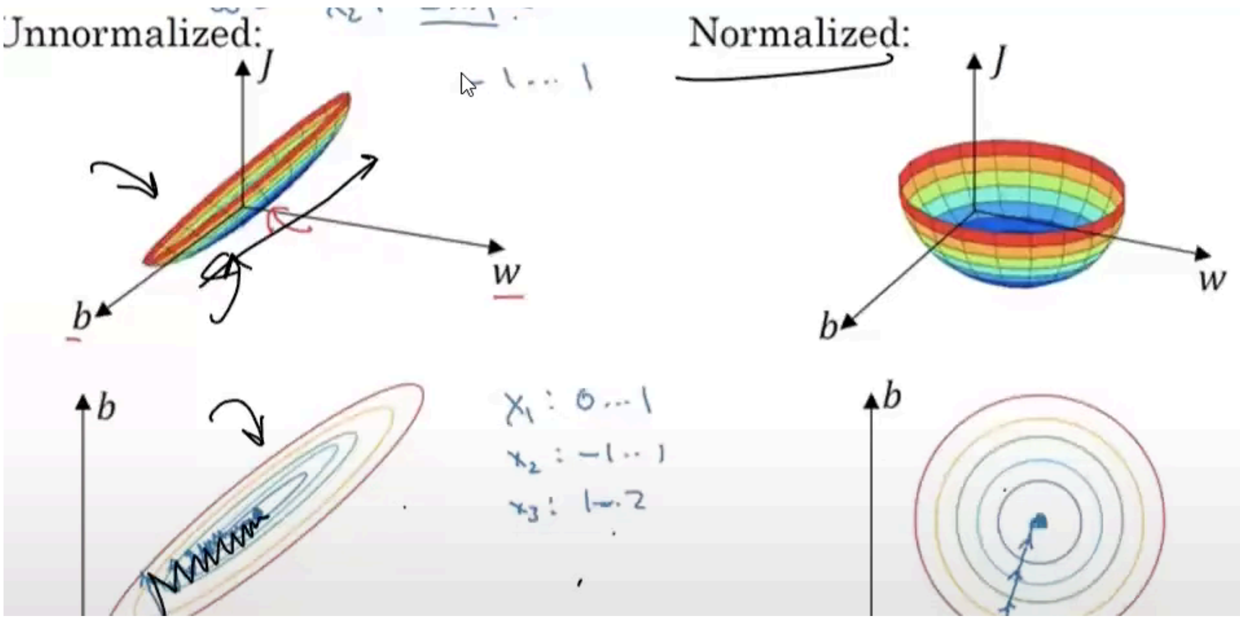
✅ **Prevents Overfitting**: Stops training before the model memorizes noise.

✅ **Saves Time**: Avoids unnecessary epochs.

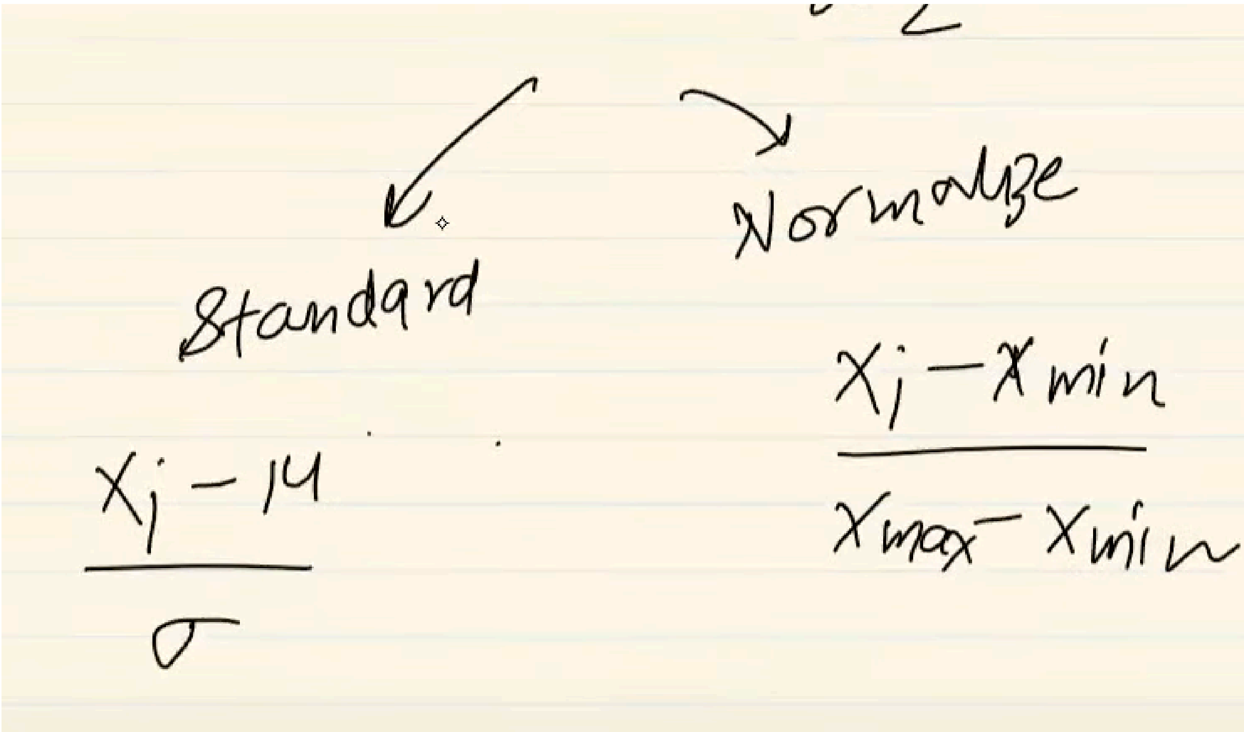
✅ **No Extra Parameters**: Unlike L2/Dropout, it doesn't modify the model.

Data Scaling (Normalization)

Neural networks are sensitive to input scale. Proper data scaling ensures:
✅ Faster convergence (optimizers work efficiently).
✅ Better performance (avients dominance of large-scale features).
✅ Numerical stability (prevents exploding/vanishing gradients).



Standardize vs Normalize



- **Standardize → -1 to 1**
- **Normalize → 0 to 1**

When to standardise & when to normalize?

- If you know the max & min values → Normalize
- Otherwise → Standardise
  - Standardise if data is normally distributed.

Standardization (Z-Score Normalization)

$$X_{\text{scaled}} = \frac{X - \mu}{\sigma}$$

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Min-Max Normalization

$$X_{\text{scaled}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

🚀 When to Scale Data?

Algorithm	Scaling Needed?	Recommended Method
Neural Networks	✅ Yes	Standardization
CNNs (Images)	✅ Yes	Min-Max (0-1)
RNNs (Time Series)	✅ Yes	Standardization
Tree-Based Models	❌ No	Not required

⚠️ Common Mistakes

❌ **Scaling Test Data Separately**: Always use `transform()` with the **same scaler** from training.

❌ **Ignoring Categorical Features**: Scale only **numerical** features (one-hot encode categoricals).

❌ **Forgetting to Scale Outputs**: In regression, scale yy if it has a large range.