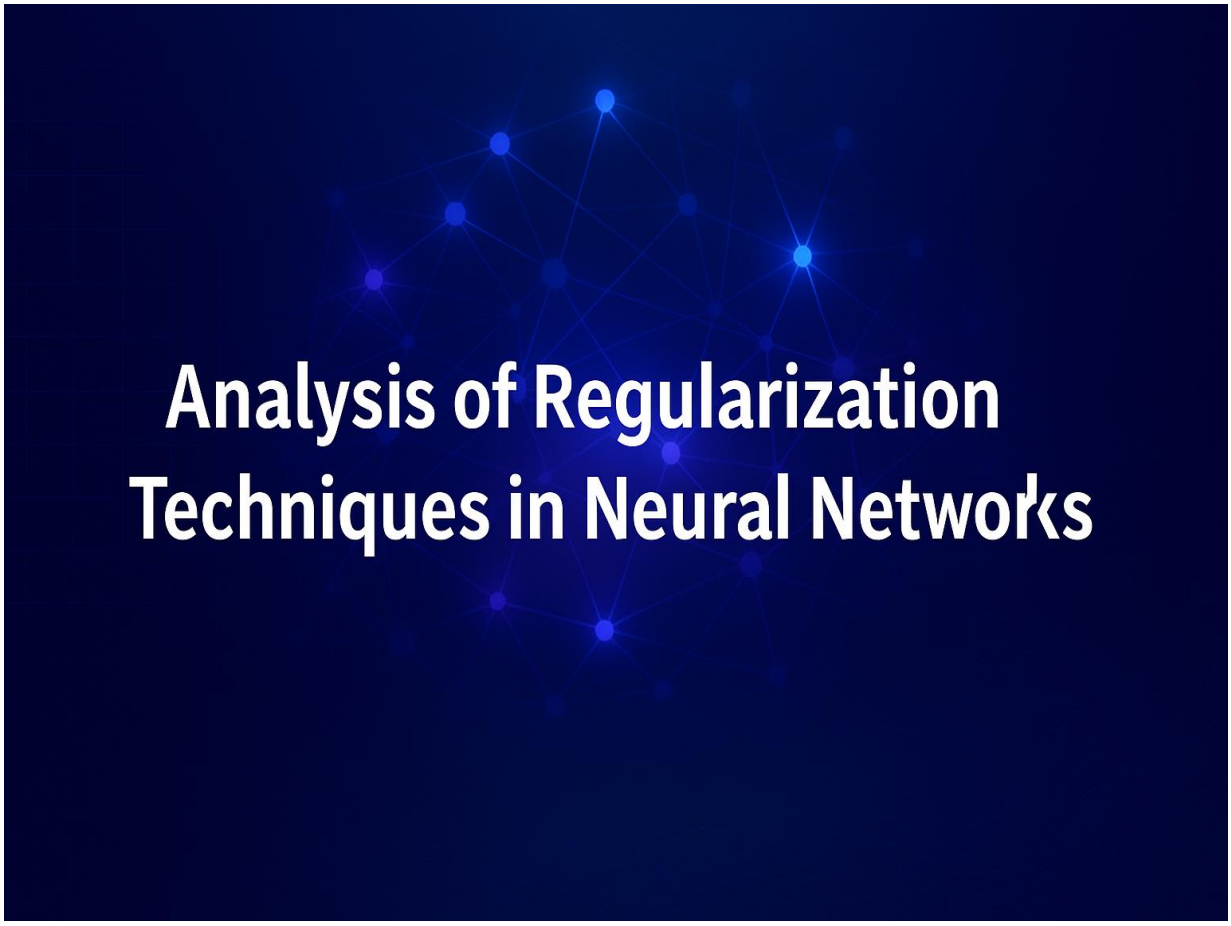
Regularization-Tradeoffs-and-Insights



bias-variance tradeoff

The **bias-variance tradeoff** is a fundamental concept in machine learning that involves finding a balance between two types of errors a model can make:

- **Bias** is the error introduced by approximating a real-world problem, which may be complex, with a simplified model. A model with **high bias** pays little attention to the training data and oversimplifies the model. This leads to **underfitting**, where the model fails to capture the underlying patterns in the data and performs poorly on both training and test data.

- **Variance** is the error introduced because the model is too sensitive to small fluctuations in the training data. A model with **high variance** pays too much attention to the training data, including its noise. This leads to **overfitting**, where the model performs very well on the training data but poorly on new, unseen data.

The Tradeoff:

There is an inverse relationship between bias and variance.

- Increasing a model's complexity will **decrease its bias** (it can learn more complex patterns) but **increase its variance** (it's more likely to overfit).
- Decreasing a model's complexity (making it simpler) will **increase its bias** (it might underfit) but **decrease its variance**.

The goal is to find the right level of model complexity that minimizes the total error by achieving a good balance between bias and variance. The regularization techniques discussed in the Canvas are methods used to manage this tradeoff, typically by adding a bit of bias to the model to significantly reduce its variance and prevent overfitting.

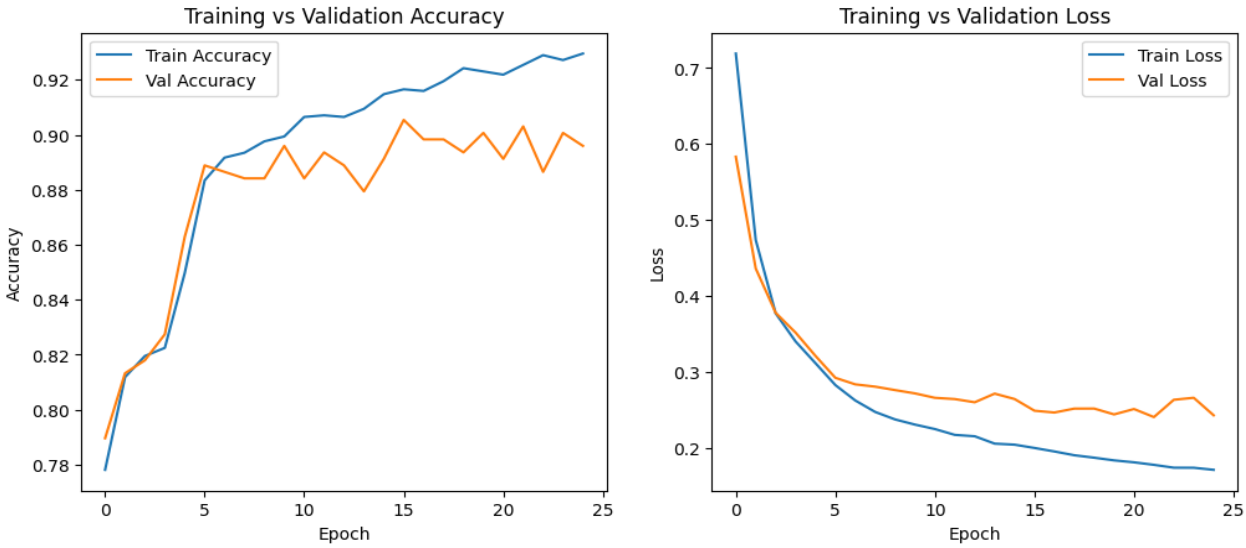
Model Without Regularization

This model serves as a baseline to observe the effects of overfitting. Without any regularization, the model is likely to learn the training data too well, including the noise, which can lead to poor performance on unseen data.

Model Architecture:

```
model = Sequential([
  Dense(32, input_shape=(X_train.shape[1],), activation='relu'),
  Dense(16, activation='relu'),
  Dense(6, activation='relu'),
  Dense(3, activation='softmax')
])
```

Plots:



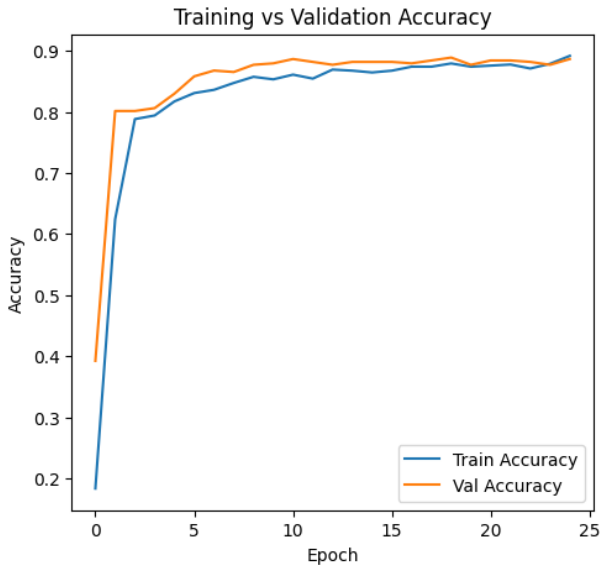
Model with Dropout Regularization

Dropout is a technique where randomly selected neurons are ignored during training. This prevents the network from becoming too reliant on any single neuron and helps in learning a more robust representation of the data.

Model Architecture:

```
model = Sequential([
  Dense(32, input_shape=(X_train.shape[1],), activation='relu'),
  Dropout(0.3),
  Dense(16, activation='relu'),
  Dropout(0.2),
  Dense(6, activation='relu'),
  Dropout(0.1),
  Dense(3, activation='softmax')
])
```

Plots:



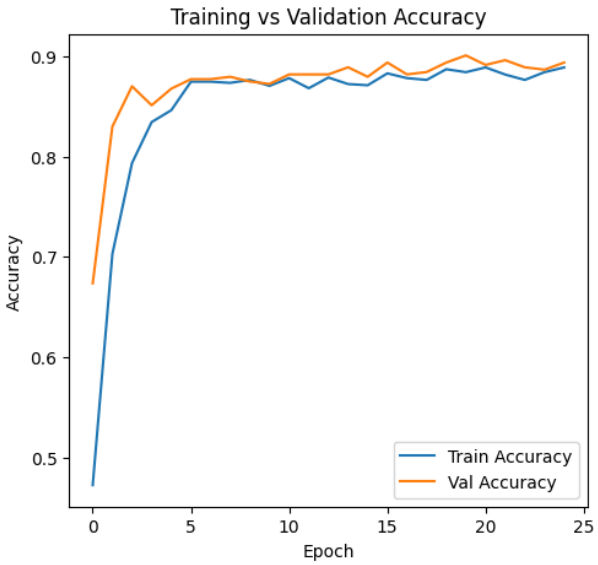
. Model with L1 Regularization

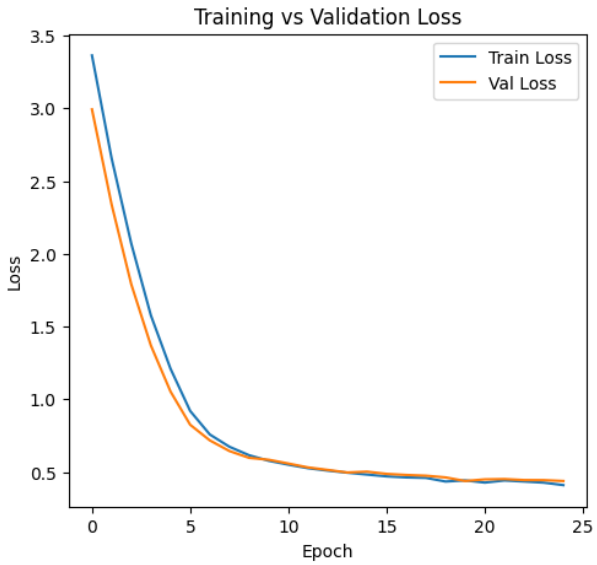
L1 regularization, also known as Lasso regularization, adds a penalty to the loss function that is proportional to the absolute value of the weights. This encourages the model to learn sparse weights, effectively performing feature selection by driving some weights to zero.

Model Architecture:

```
model1 = Sequential([
  Dense(64, input_shape=(X_train.shape[1],),
kernel_regularizer=regularizers.l1(0.001)),
  Activation('relu'),
  Dense(32, kernel_regularizer=regularizers.l1(0.005)),
  Activation('relu'),
  Dense(16),
  Activation('relu'),
  Dense(1, activation='sigmoid')
])
```

Plots:





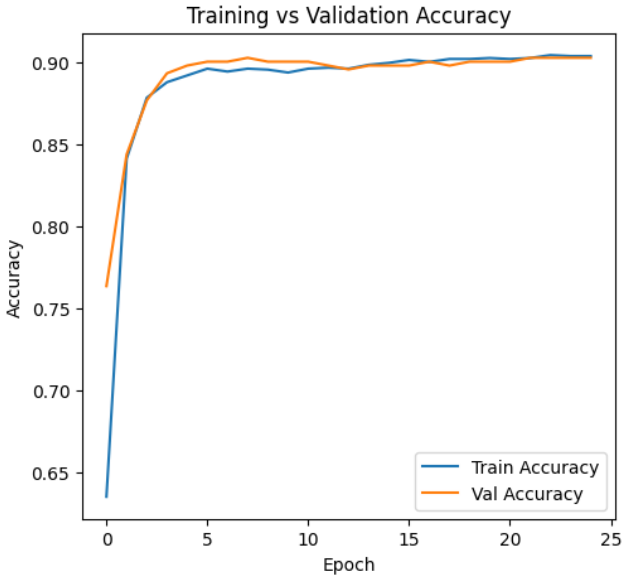
Model with L2 Regularization

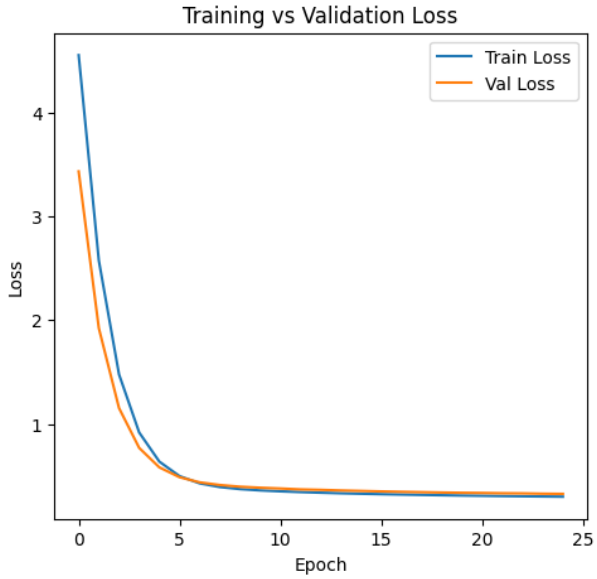
L2 regularization, also known as Ridge regularization, adds a penalty to the loss function that is proportional to the square of the weights. This encourages the model to learn smaller weights, which helps in preventing the model from becoming too complex and overfitting.

Model Architecture:

```
model1 = Sequential([
  Dense(64, input_shape=(X_train.shape[1],),
kernel_regularizer=regularizers.l2(0.2)),
  Activation('relu'),
  Dense(32, kernel_regularizer=regularizers.l2(0.002)),
  Activation('relu'),
  Dense(16, kernel_regularizer=regularizers.l2(0.005)),
  Activation('relu'),
  Dense(1, activation='sigmoid')
])
```

Plots:





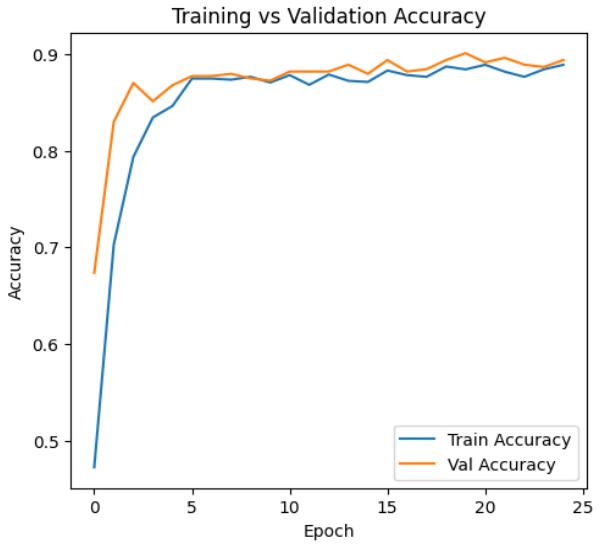
Model with L1 Regularization and Dropout

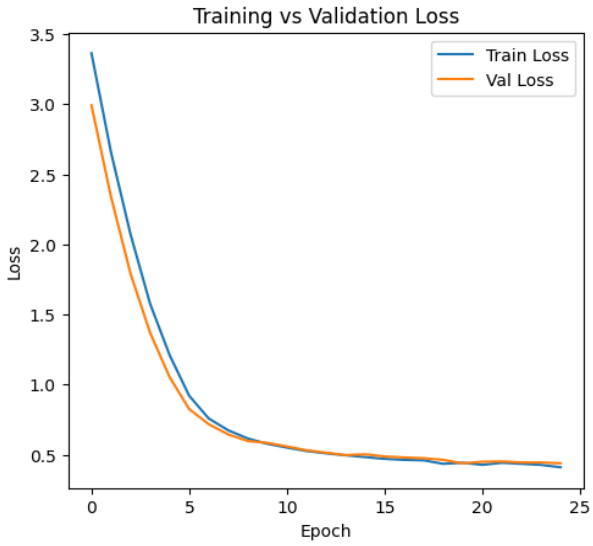
This model combines L1 regularization with dropout. This can be a powerful combination, as L1 regularization encourages sparse weights, while dropout helps in learning a more robust representation of the data.

Model Architecture:

```
model1 = Sequential([
  Dense(32, input_shape=(X_train.shape[1],),
kernel_regularizer=regularizers.l1(0.03)),
  Activation('relu'),
  Dropout(0.4),
  Dense(16, kernel_regularizer=regularizers.l1(0.04)),
  Activation('relu'),
  Dense(6),
  Activation('relu'),
  Dense(1, activation='sigmoid')
])
```

Plots:





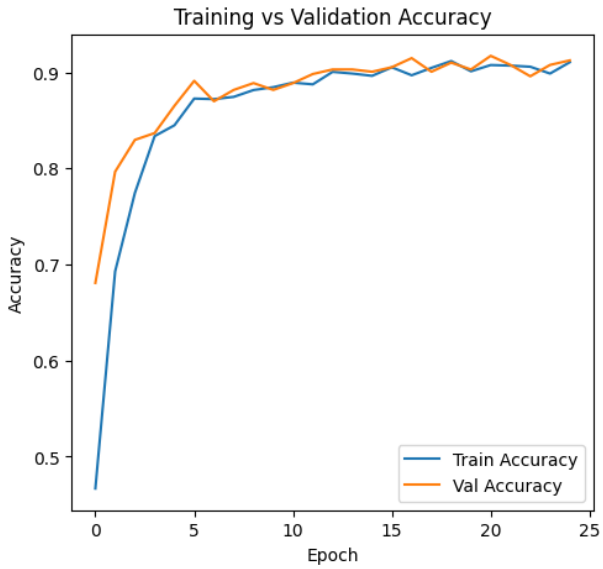
6. Model with L2 Regularization and Dropout

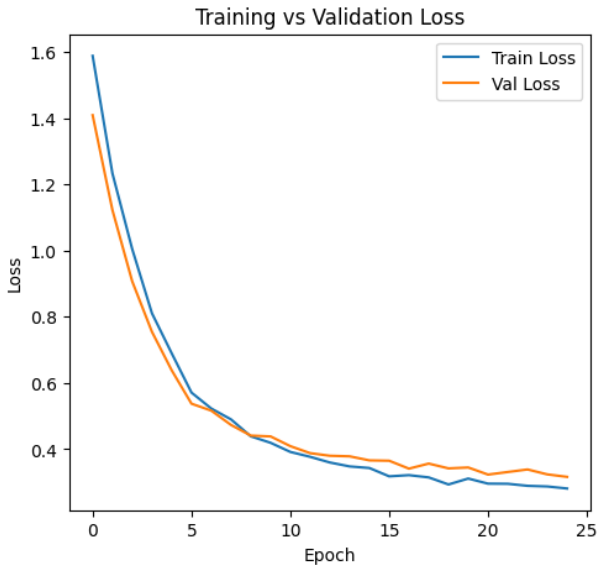
This model combines L2 regularization with dropout. This can also be a powerful combination, as L2 regularization encourages smaller weights, while dropout helps in learning a more robust representation of the data.

Model Architecture:

```
model1 = Sequential([
  Dense(32, input_shape=(X_train.shape[1],),
kernel_regularizer=regularizers.l2(0.01)),
  Activation('relu'),
  Dropout(0.4),
  Dense(16, kernel_regularizer=regularizers.l2(0.04)),
  Activation('relu'),
  Dense(6),
  Activation('relu'),
  Dense(1, activation='sigmoid')
])
```

Plots:





Comparison of L1, L2, and Dropout Regularization

Feature	L1 Regularization (Lasso)	L2 Regularization (Ridge)	Dropout
Penalty	Adds a penalty proportional to the absolute value of the weights.	Adds a penalty proportional to the square of the weights.	Randomly sets a fraction of neuron activations to zero during training.
Effect on Weights	Encourages sparse weights, driving some weights to zero.	Encourages smaller weights, but rarely drives them to zero.	Does not directly affect the weights, but encourages the network to learn a more robust representation of the data.
Feature Selection	Performs feature selection by driving some weights to zero.	Does not perform feature selection.	Does not perform feature selection.
Computational Cost	Can be more computationally	Less computationally	Adds some computational

	expensive than L2 regularization.	expensive than L1 regularization.	overhead during training, but not during inference.
Use Cases	Useful when you suspect that some features are irrelevant or redundant.	Useful when you want to prevent the model from becoming too complex and overfitting.	Useful in a wide range of neural network architectures, especially deep networks.