ASSIGNMENT - 5

1. How can each of these parameters be fine-tuned? • Number of hidden layers

• Network architecture (network depth)

Number of hidden layers: Experiment with different numbers of hidden layers. Start with a simple architecture and gradually increase complexity as needed, monitoring validation performance to avoid overfitting.

Network depth: Similar to hidden layers, experiment with increasing depth (more layers) while monitoring validation performance. Deeper networks can capture more complex relationships but are also more prone to overfitting.

Layer Width (Number of Neurons):

Start with a reasonable number of neurons based on the complexity of the problem and data size.

Use techniques like grid search or random search to explore different widths for each layer and choose the setting that performs best on validation data.

• Each layer’s number of neurons (layer width)

• Form of activation

Activation functions (e.g., ReLU, sigmoid, tanh) for hidden layers and choose the one that leads to better performance on your specific task.

• Optimization and learning

Optimization algorithm: Experiment with different optimizers (e.g., SGD, Adam, RMSprop) and see which converges faster and achieves better results.

• Learning rate and decay schedule

Start with a reasonable learning rate and adjust it during training (e.g., using a decay schedule). A high learning rate can lead to unstable training, while a low rate can make training slow.

• Mini batch size

Mini-batch size can affect training speed and convergence. Experiment with different sizes and choose a value that balances efficiency and performance.

• Algorithms for optimization

Explore different optimization algorithms like stochastic gradient descent (SGD) with momentum, Adam, RMSprop, etc., and see which one works best for your specific problem.

• The number of epochs (and early stopping criteria)

Train the network for a sufficient number of epochs to allow it to learn from the data. Consider using early stopping to halt training if validation performance doesn't improve for a certain number of epochs, preventing overfitting.

• Overfitting that can be avoided by using regularization techniques.

Overfitting that can be avoided by using regularization techniques are

* L2 normalization.
* Drop out
* Data augmentation

• L2 normalization

This technique penalizes large weights in the network, encouraging smoother weight distributions and reducing overfitting.

• Drop out layers

Randomly dropping neurons during training with a certain probability can prevent them from co-adapting too much, improving model generalization.

• Data augmentation

Artificially creating new data points by applying random transformations (e.g., rotations, flips, cropping) to your existing data can increase the size and diversity of your training set, making the model less prone to overfitting to the specific training data.