**1. How can each of these parameters be fine-tuned?**

**• Number of hidden layers :** Experimenting with different numbers of hidden layers allows you to find the optimal balance between model complexity and generalization. Too few hidden layers may lead to underfitting, while too many may lead to overfitting.

**• Network architecture (network depth) :** Adjusting the depth of the network involves adding or removing layers to improve model performance. Deeper networks can capture more complex relationships in the data but may also suffer from vanishing gradients or overfitting if not properly regularized.

**• Each layer's number of neurons (layer width) :** Tuning the number of neurons in each layer affects the model's capacity to learn intricate patterns in the data. Increasing the width of layers can improve the model's ability to represent complex functions, but it may also increase the risk of overfitting.

**• Form of activation :** Choosing appropriate activation functions for each layer affects the non-linear mapping capabilities of the network. Common choices include ReLU, sigmoid, and tanh functions, each with different properties that may be more suitable for certain tasks or architectures.

**• Optimization and learning :** Selecting the right optimization algorithm (e.g., SGD, Adam, RMSprop) and learning strategy (e.g., momentum, cyclical learning rates) influences the efficiency and effectiveness of model training. Experimenting with different optimization techniques can lead to faster convergence and better generalization.

**• Learning rate and decay schedule :** Tuning the learning rate and its decay schedule controls the step size of parameter updates during training. Learning rate schedules such as exponential decay or step decay can help balance rapid initial progress with fine-tuning towards the end of training.

**• Mini batch size :** Adjusting the mini batch size affects the stability of the optimization process and the quality of parameter updates. Larger batch sizes can accelerate training but may lead to poorer generalization or convergence to suboptimal solutions.

**• Algorithms for optimization :** Exploring alternative optimization algorithms such as adaptive learning rate methods or second-order optimization techniques can help overcome challenges like slow convergence or poor generalization.

**• The number of epochs (and early stopping criteria) :** Determining the appropriate number of training epochs and implementing early stopping based on validation performance helps prevent overfitting and ensures the model generalizes well to unseen data.

**• Overfitting that be avoided by using regularization techniques. :**

Regularization techniques like L2 normalization (weight decay) penalize large weights to prevent overfitting by encouraging simpler models with smoother decision boundaries.

**• Drop out layers :** Adding dropout layers during training randomly deactivates a fraction of neurons, preventing co-adaptation and improving the generalization ability of the network.

**• Data augmentation :** Augmenting the training data with transformations like rotation, flipping, or scaling increases the diversity of the dataset, helping the model learn more robust features and reducing overfitting.