How to Improve the Performance of a Neural Network

Hyperparameter Tuning

- **No. of hidden layers**
- **No. of neurons**
- **Learning Rate**:
  - Use learning rate schedulers (`ReduceLROnPlateau`).
  - Try cyclical learning rates.
- **Batch Size**: Smaller batches (32-128) often generalize better.
- **Optimizers**:
  - **Adam**: Default choice for most cases.
  - **SGD with Momentum**: Better for some tasks (e.g., CNNs).
- **Activation function**
- No. of epochs

Solve Problems

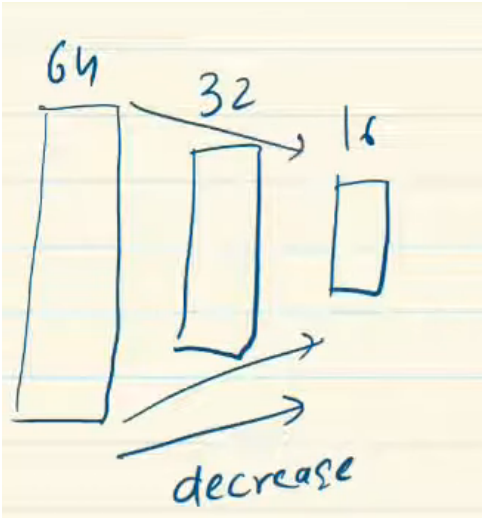
- Vanishing/exploding gradient problem
- Not enough data
- Slow training
- Overfitting

No. of hidden layers

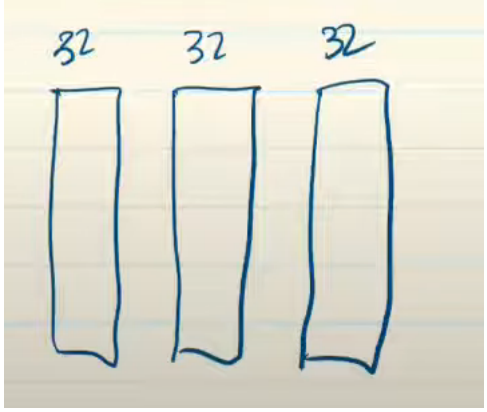
- Instead of taking 1 hidden layer with lot of neurons, taking multiple hidden layer with few neurons usually gives good results.

No. of neurons

- Pyramid structure



- Same neurons



💡 **Both give same results.**

- No. of neurons should be **sufficient**.
  - Start with more number

Batch Size

- Smaller batch size shows better results on newer datasets (**8 to 32**)
- But large batch size gives faster results (Usually → **8192**)
- Use → **Learning Rate Scheduler**
  - Keep LR small for initial epochs
  - ⬆️Increase **LR** as the epochs increase
  - 👆This approach is called **warming up the learning rate**

Wise Approach:

- First, try the **warming up the learning rate**
- If it doesn't work → Go for small batch size

No. of epochs

Early Stopping & Data Scaling

- Try max value
- Use **early stopping → Keras feature:** `callback`
  - It's a mechanism which stops the training if results aren't improving

Vanishing/exploding gradient problem

- Wight initialization
- Change activation function
- Batch normalization
- Gradient clipping → Used for exploding gradient
  - (for RNNs/Transformers)

Not enough data

- Transfer learning
  - Use some other model's data to train yours
  - Use pre-trained models (e.g., ResNet, BERT).
- Unsupervised pre-training

Slow training

- Use different optimizers
- Use learning rate scheduler (`ReduceLROnPlateau`)

Overfitting

- Use L1 & L2 Regularization
- Dropouts

✨Overview:

Improving Neural Network Performance:

1. **Vanishing Gradients**
   - Activation Functions
   - Weight Initialization
2. **Overfitting**
   - Reduce Complexity/Increase Data
   - Dropout Layers
   - Regularization (L1 & L2)
   - Early Stopping
3. **Normalization**

- Normalizing Inputs
- Batch Normalization
- Normalizing Activations

4. **Gradient Checking and Clipping**
5. **Optimizers**
    - Momentum
    - Adagrad
    - RMSprop
    - Adam
6. **Learning Rate Scheduling**
7. **Hyperparameter Tuning**
    - No. of hidden layers
    - nodes/layer
    - Batch size