Implementation Tips and Best Practices

📊 Data & Training

1. **Normalize Inputs**
   Scale features to [0,1] or standardize (mean=0, std=1)
2. **Weight Initialization**
   Use Xavier/He initialization, avoid zeros
3. **Split Data Properly**
   Train (70-80%), Validation (10-15%), Test (10-15%)
4. **Monitor Training**
   Track both training and validation loss
5. **Use Early Stopping**
   Stop when validation loss stops improving

🐛 Debugging

1. **Start Simple**
   Begin with small network, overfit small dataset first
2. **Check Gradient Flow**
   Verify gradients are not vanishing or exploding
3. **Visualize Activations**
   Plot activation distributions across layers
4. **Numerical Gradient Check**
   Verify backprop implementation with finite differences
5. **Log Everything**
   Track loss, accuracy, learning rate, gradients

⚙️ Hyperparameters

🚀 Optimization

1. **Learning Rate**
   Start with 0.001-0.01, use learning rate schedules
2. **Batch Size**
   Common: 32, 64, 128. Larger = more stable but slower
3. **Number of Epochs**
   Start high, use early stopping to prevent overfitting
4. **Regularization**
   Try L2 (λ=0.01), dropout (p=0.5), or both

Grid/Random Search

1. **Use Adam Optimizer**
   Good default choice with adaptive learning rates
2. **Batch Normalization**
   Normalize layer inputs for faster convergence
3. **Learning Rate Decay**
   Reduce LR when validation loss plateaus
4. **Gradient Clipping**
   Prevent exploding gradients (clip at norm=5)

Data Augmentation

Before Training

- ☑ Data normalized
- ☑ Train/val/test split
- ☑ Weights initialized
- ☑ Loss function chosen

During Training

- ☑ Monitor loss curves
- ☑ Check gradient norms
- ☑ Save best model
- ☑ Log metrics regularly

After Training

- ☑ Evaluate on test set
- ☑ Analyze errors
- ☑ Document results
- ☑ Compare baselines

⚠ **Common Pitfalls to Avoid**

Not shuffling data • Forgetting to normalize • Using test set for hyperparameter tuning • Setting learning rate too high • Ignoring validation metrics • Not saving checkpoints

🎯 Hands-On: TensorFlow Playground

Visualize and understand how neural networks work! Use TensorFlow Playground to observe the training process in real-time by adjusting various hyperparameters. Start with the Circle dataset and experiment with learning rates, layer structures, activation functions, and more.

🚀 Launch TensorFlow Playground