[SOLUTION] Exercise 10: Model Improvement Strategy

Estimated Time: 20-25 minutes

This exercise trains you to diagnose different model failure modes and recommend the right improvements without writing training code.

Here's the outline:

Together, the four tasks develop the kind of strategic thinking that's critical for building reliable, production-ready models.

Feel free to skip the scenario and goal if you prefer to jump straight to the tasks.

Scenario

You are the new ML lead at a fast-growing startup building a wine-quality prediction system. To speed up development, three sub-teams trained three different experimental variants of the model in parallel. These were meant to compete as candidates for deployment.

However, when you review the experiment dashboard this morning, all three models show different kinds of failures:

- **Model Variant A:** Shows excellent training performance, but validation performance deteriorates as training continues. The loss curves drift apart over time.
- **Model Variant B:** Training and validation metrics barely improve. Loss remains nearly flat across epochs.
- **Model Variant C:** Training repeatedly fails to complete. Logs show unstable behavior, with loss values jumping sharply between steps.

▸ **Dataset used for all variants: mstz/wine.**

Your job: Diagnose each model's failure mode and recommend specific improvements, choosing from techniques such as dropout, weight decay, learning-rate tuning, capacity adjustments, gradient clipping, or other relevant interventions.

You will not write code; instead, you will guide the team toward the right decisions and build a generalizable framework they can reuse.

Goal: Develop the ability to diagnose model failures, map problems to appropriate solution techniques, and build generalizable decision-making frameworks for improving ML models across architectures and domains.

Tasks

Part 1: Match Symptoms to Fixes

Complete the decision matrix below by checking (✓) which techniques help solve each problem. Some techniques may help multiple problems, while others are specific to one issue.

Problem Type	Dropout	Increase LR	Reduce LR	Add Layers	Gradient Clipping
Overfitting	✓				
Underfitting		✓		✓	
Training Instability			✓		✓
Slow Convergence		✓			

> 💡 ***Hint:*** *Think about what each problem means. Overfitting = model is too flexible and memorizing. Underfitting = model is too simple. Training instability = updates are too aggressive. Slow convergence = learning steps are too conservative.*

Rationale:

- Overfitting needs regularization.
- Underfitting needs more capacity or faster learning.
- Training instability needs more stable updates.
- Slow convergence needs faster learning.

Part 2: Analyze Model Variants

For each model variant below, review the provided metrics and loss curve, then **answer the questions**.

Note: Use your loss curve diagnosis skills from *Exercise 8*. The curve patterns will guide which category of techniques to recommend.

Model Variant A: Drifting Loss

Model: 3-layer MLP with architecture [128 → 64 → 32 → 1]

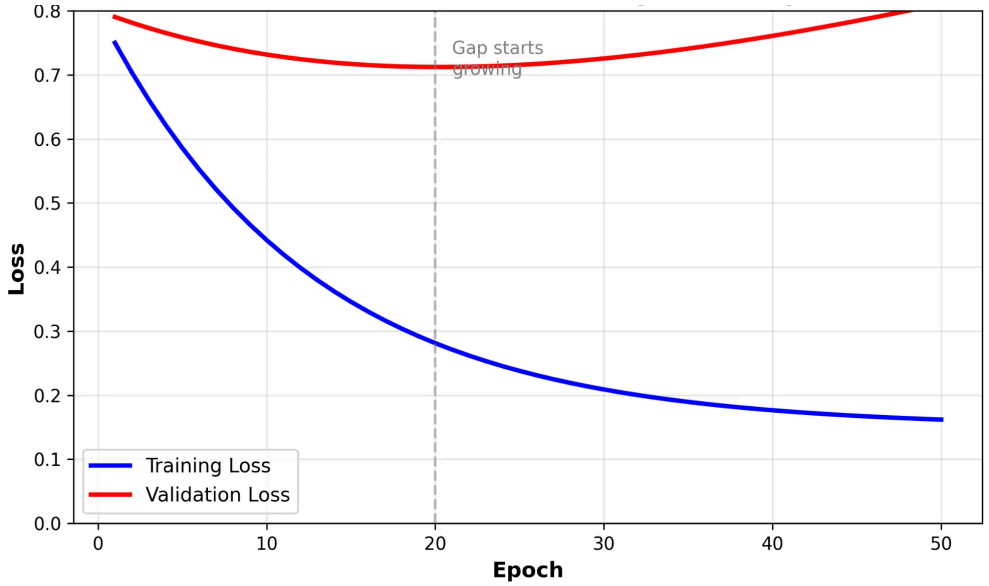
Dataset: 12,000 training samples, 3,000 validation samples

Loss function: BCEWithLogitsLoss (binary cross-entropy)

Optimizer: Adam, LR=1e-3, batch_size=64

Training: 100 epochs, NO regularization

Performance: Train accuracy: 92% / Validation accuracy: 73%, Train loss: 0.18 / Validation loss: 0.52



Questions:

1. What is the primary issue with this model? *(1 sentence)*

The model is overfitting: training loss drops to 0.18 while validation loss climbs to 0.52, indicating that the model is memorizing the training data rather than learning generalizable patterns.

2. Recommend 2-3 techniques in priority order with brief justification. *(2-3 sentences)*

1. **Dropout (p=0.3)**: Forces the network to learn robust features by randomly disabling neurons during training, preventing co-adaptation.
2. **Early stopping with patience=5-10**: Automatically halts training when validation loss stops improving, capturing the model at peak generalization.
3. **Weight decay (1e-4)**: Penalizes large weights to encourage simpler models that generalize better.

3. Implementation: Show a 2-3 line code snippet demonstrating your top recommended technique.

```
# Add dropout between layers
layers.append(nn.Linear(128, 64))
layers.append(nn.ReLU())
layers.append(nn.Dropout(p=0.3))  # Drop 30% of neurons
```

Model Variant B: Flat loss

Model: 2-layer MLP with architecture [64 → 32 → 1]

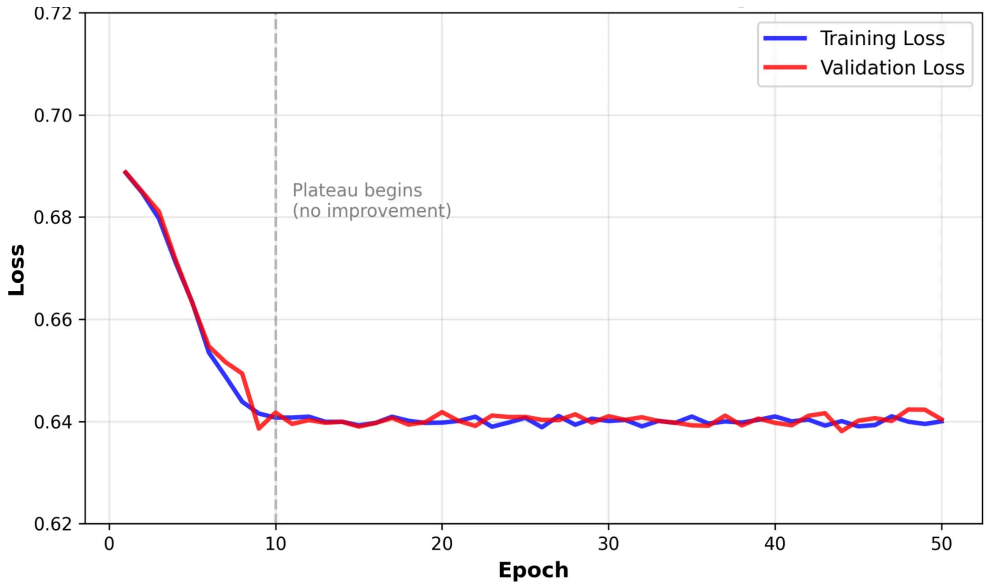
Dataset: 8,000 training samples, 2,000 validation samples

Loss function: BCEWithLogitsLoss (binary cross-entropy)

Optimizer: Adam, LR=1e-3, batch_size=128

Training: 100 epochs, no regularization applied

Performance: Train accuracy 64% / Val accuracy 63%, Train loss 0.68 / Val loss 0.69



Questions:

1. What is the primary issue with this model? *(1 sentence)*

The model is underfitting: both training and validation losses plateau at high values (~0.68), indicating the model lacks sufficient capacity to learn the underlying patterns.

2. Recommend 2-3 techniques in priority order with brief justification. *(2-3 sentences)*

1. **Increase model capacity**: Add more layers or wider layers (e.g., [128, 256, 128] instead of [64, 32]) to give the model enough parameters to capture complex patterns.
2. **Train longer**: Extend training to 150-200 epochs since the model hasn't converged yet.
3. **Reduce regularization**: If any dropout or weight decay is active, reduce or remove it as it's constraining an already weak model.

3. Implementation: Show a 2-3 line code snippet demonstrating your top recommended technique.

```
# Increase model capacity
hidden_sizes = [128, 256, 128]  # Wider layers
model = MyModel(input_size=12, hidden_sizes=hidden_sizes, output_size=1)
```

Model Variant C: Unstable loss

Model: 4-layer MLP with architecture [256 → 128 → 64 → 32 → 1]

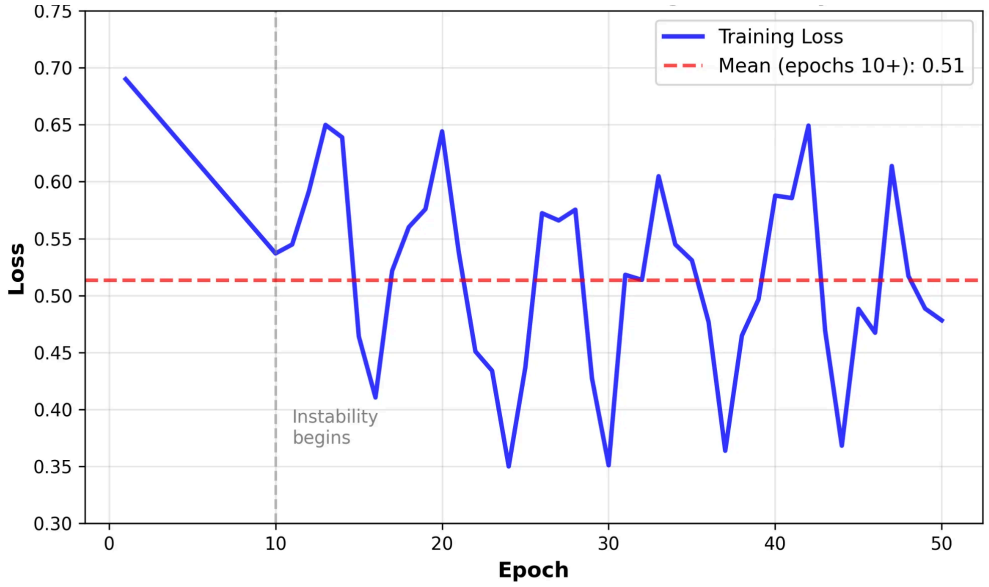
Dataset: 15,000 training samples, 5,000 validation samples

Loss function: BCEWithLogitsLoss (binary cross-entropy)

Optimizer: SGD, LR=0.1, batch_size=32

Training: 100 epochs attempted, training never stabilized

Performance: Loss oscillates between 0.3-0.7, never converging



Questions:

1. What is the primary issue with this model? *(1 sentence)*

The model has training instability: loss oscillates erratically without convergence, indicating that the learning rate is too high or gradients are exploding.

2. Recommend 2-3 techniques in priority order with brief justification. *(2-3 sentences)*

1. **Reduce learning rate by 10×**: Change from 0.1 to 0.001 to make updates smaller and more stable.
2. **Add gradient clipping**: Cap gradient norms at 1.0 or 5.0 to prevent explosive updates that cause loss spikes.
3. **Check data quality**: Verify there are no NaN/inf values, extreme outliers, or missing normalization. These can cause training instability.

3. Implementation: Show a 2-3 line code snippet demonstrating your top recommended technique.

```
# Reduce learning rate
optimizer = optim.Adam(model.parameters(), lr=1e-4)  # Was 1e-3
# Add gradient clipping
torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
```

Part 3: Rank Techniques by Effort vs. Impact

Complete the table below by evaluating each technique's implementation effort and expected impact.

Technique	Implementation Effort	Expected Impact	Best Used When
Early Stopping	Low	Medium	Always (default practice)
Dropout	Low	High	Clear overfitting (large train-val gap)
Weight Decay	Low	Medium	Mild overfitting or as a preventive measure
LR Tuning	Low	High	Training instability, slow convergence, or initial setup
Collect More Data	High	High	Persistent overfitting after regularization is exhausted
Architecture Search	High	Medium	Research projects or when the compute budget allows

> 💡 ***Hint:*** *Implementation effort = how much code/time needed. Expected impact = how much improvement you'd expect. Think: Low/Medium/High for each.*

Key insight:

- Prioritize high-impact, low-effort techniques first.
- Save expensive options for when optimization techniques are exhausted.

Part 4: Build Your Improvement Playbook

Create a 5-step checklist that any team member could follow when facing an underperforming model. **Fill in each step below.**

Step 1: First, always check: Data quality and preprocessing: verify normalization, check for NaN/inf values, ensure train/val/test splits are correct, and confirm data is representative of deployment distribution.

> 💡 ***Hint:*** *What foundational issues should you rule out before trying advanced techniques? (data quality, normalization, etc.)*

Step 2: If overfitting (train-val gap), try: Enable early stopping (patience=5-10), then add dropout (start with p=0.3), then try weight decay (1e-4). Change one at a time and observe results.

> 💡 ***Hint:*** *What's the easiest, most effective regularization technique to try first?*

Step 3: If underfitting (both losses high), try: Train longer (increase epochs), then increase model capacity (add layers or neurons), then reduce/remove regularization if present. Verify data quality first.

> 💡 ***Hint:*** *The model needs more capacity or more time to learn. What's the first thing to try?*

Step 4: Before collecting more data, ensure: You have tried all optimization techniques (dropout, weight decay, LR tuning, early stopping), verified the model has appropriate capacity, and explored data augmentation if applicable. Maximize existing data first.

> 💡 ***Hint:*** *Have you extracted maximum value from your existing data? What optimization techniques are left?*

Step 5: Last resort options (expensive/time-consuming): Collect more data (labeling costs time/money), run architecture search (compute-intensive), or hire a domain expert to engineer better features. Only pursue after exhausting optimization techniques.

> 💡 ***Hint:*** *What requires significant resources or expertise? (data collection, architecture search, etc.)*

Key Takeaways

Model improvement is a systematic process, not random experimentation. By the end of this exercise, you should understand:

- How to diagnose model problems from loss curves and metrics
- Which techniques address which problems (and why)
- How to prioritize improvements based on effort vs. impact
- A reusable framework for tackling underperforming models

Remember: Start with high-impact, low-effort techniques (optimization and data improvements) before moving to expensive solutions. Change one thing at a time, observe results, then iterate.

▸ **Next Step:** Closing the Loop. You've diagnosed all three model variants and recommended specific improvements. What should each team do next to move from recommendations to deployment? *Consider model retraining, evaluation, and deployment trade-offs for final model selection.*

No need to answer, just a challenge for you to reflect on!