Neural Forecasting Model Improvement Guide

Current Baseline

Monkey	Val MSE	Test MSE	Channels	Parameters
Beignet	0.439	0.396	89	207,690
Affi	0.586	0.608	239	207,690

---

1. Easy/Immediate Fixes

1.1 Increase Model Capacity

Current: d_model=64, n_heads=4, n_layers=3

Suggested: d_model=128 or 192, n_heads=8, n_layers=4-6

```
# In train_and_evaluate.py:97-105
model = get_model_for_monkey(
    monkey_name,
    n_channels=n_channels,
    d_model=128,      # was 64
    n_heads=8,        # was 4
    n_layers=4,       # was 3
    dropout=0.2       # slightly higher for larger model
)
```

Aspect	Notes
Expected improvement	5-15% MSE reduction
Difficulty	Trivial (change 3 numbers)
Pros	More capacity to model comp...
Cons	~4x memory, ~3x slower trai...

1.2 Enable Refinement During Training

Current: `n_refinement_iters=0` during training (memory reasons)

Your model has iterative refinement but it's disabled during training. The model never learns to use this capability.

```
# Gradual approach: enable 1 iteration with gradient checkpointing
model = get_model_for_monkey(
    monkey_name,
    n_refinement_iters=1,  # Start with 1
)
```

Aspect	Notes
Expected improvement	3-8% (model learns to refin...
Difficulty	Easy, but requires memory m...
Pros	Trains what you evaluate; r...
Cons	2x memory per refinement it...

1.3 Longer Training with Lower Learning Rate

Current: 100 epochs max, lr=1e-3, early stopped at ~41-44 epochs

```
  train_and_evaluate(
      monkey,
      n_epochs=200,
      learning_rate=5e-4,  # Lower peak LR
  )
```

Aspect	Notes
Expected improvement	2-5%
Difficulty	Trivial
Pros	More thorough optimization
Cons	Longer training time

---

2. Straightforward Hyperparameter Improvements

2.1 Learning Rate Tuning

Your current warmup/cosine schedule is reasonable, but the peak LR may be too high for this problem.

Try: Grid search over {1e-4, 3e-4, 5e-4, 1e-3}

Aspect	Notes
Expected improvement	3-10%
Difficulty	Low (run 4 experiments)

2.2 Adjust Smoothness Loss Weight

Current: `smoothness_weight=0.05`

The smoothness regularizer penalizes temporal discontinuities. This may be over-constraining neural signals which can have abrupt changes.

Try: {0.0, 0.01, 0.02, 0.05, 0.1}

Aspect	Notes
Expected improvement	2-5%
Difficulty	Low
Pros	May allow model to capture ...
Cons	Predictions may become noisier

2.3 Data Augmentation Tuning

Current: electrode_dropout=0.1, noise_std=0.02, each applied 50% of time

```
  # In dataset.py, try different augmentation strengths
  NeuralDataAugmentation(
      electrode_dropout_prob=0.15,  # Slightly more aggressive
      noise_std=0.05,               # More noise tolerance
  )
```

Aspect	Notes
Expected improvement	2-8% on generalization
Difficulty	Low

Pros	Better robustness to electr...
Cons	Too aggressive may hurt tra...

2.4 Batch Size Optimization

Current: 16 for Affi, 32 for Beignet

Larger batch sizes with linear LR scaling often help transformers.

Aspect	Notes
Expected improvement	2-5%
Difficulty	Low (may need gradient accu...

---

3. Problem-Specific Architectural Improvements

3.1 Feature-Specific Encoding (High Priority)

Current issue: All 9 features are treated identically through the same MLP. But feature[0] (the target) has different semantics than features[1-8] (frequency bands).

Improvement: Separate pathways for target signal vs. auxiliary features:

```
class ImprovedFeatureEncoder(nn.Module):
    def __init__(self, n_features=9, d_model=64):
        super().__init__()
        # Separate encoding for target (feature 0) and frequency bands (1-8)
        self.target_encoder = nn.Sequential(
            nn.Linear(1, d_model // 2),
            nn.GELU(),
            nn.LayerNorm(d_model // 2),
        )
        self.freq_encoder = nn.Sequential(
            nn.Linear(n_features - 1, d_model // 2),
            nn.GELU(),
            nn.LayerNorm(d_model // 2),
        )
        self.fusion = nn.Linear(d_model, d_model)

    def forward(self, x):
        # x: [B, T, C, F]
        target = self.target_encoder(x[..., :1])      # [B, T, C, d/2]
        freq = self.freq_encoder(x[..., 1:])          # [B, T, C, d/2]
        fused = torch.cat([target, freq], dim=-1)     # [B, T, C, d]
        return self.fusion(fused)
```

Aspect	Notes
Expected improvement	5-15%
Difficulty	Moderate
Pros	Model can learn different r...
Cons	Slightly more parameters

3.2 Learnable Spatial Embeddings

Current: No spatial positional encoding-spatial attention is permutation-equivariant.

This is a deliberate design choice for robustness, but you're losing the ability to learn electrode-specific biases.

Many electrodes may have consistent characteristics.

```
class SpatialEmbedding(nn.Module):
    def __init__(self, n_channels, d_model):
        super().__init__()
        # Learnable per-electrode embedding
        self.spatial_embed = nn.Parameter(
            torch.randn(1, 1, n_channels, d_model) * 0.02
        )

    def forward(self, x):
        # x: [B, T, C, d]
        return x + self.spatial_embed
```

Aspect	Notes
Expected improvement	3-8%
Difficulty	Easy
Pros	Model can learn electrode-s...
Cons	Less robust to electrode re...

3.3 Causal Masking in Temporal Attention

Current: Full bidirectional temporal attention-each timestep can attend to future timesteps.

For forecasting, causal masking may help the model learn better temporal dynamics:

```
def forward(self, x, temporal_mask=None):
    B, T, C, d = x.shape

    # Create causal mask if not provided
    if temporal_mask is None:
        temporal_mask = torch.triu(
            torch.ones(T, T, device=x.device) * float('-inf'),
            diagonal=1
        )

    # Temporal attention with causal mask
    x_t = x.permute(0, 2, 1, 3).reshape(B * C, T, d)
    attn_out, _ = self.temporal_attn(x_t, x_t, x_t, attn_mask=temporal_mask)
    ...
```

Aspect	Notes
Expected improvement	2-8%
Difficulty	Easy
Pros	More aligned with forecasti...
Cons	May hurt if bidirectional c...

3.4 Attention-Based Prediction Head

Current: Only uses the last timestep for prediction.

```
class AttentionPredictionHead(nn.Module):
    def __init__(self, d_model, n_future=10):
        super().__init__()
        self.query = nn.Parameter(torch.randn(1, 1, d_model) * 0.02)
        self.attn = nn.MultiheadAttention(d_model, 4, batch_first=True)
        self.head = nn.Sequential(
            nn.Linear(d_model, d_model),
            nn.GELU(),
            nn.Linear(d_model, n_future),
        )
```

```
    def forward(self, x):
        # x: [B, T, C, d]
        B, T, C, d = x.shape
        x_flat = x.permute(0, 2, 1, 3).reshape(B * C, T, d)
        query = self.query.expand(B * C, 1, d)
        attended, _ = self.attn(query, x_flat, x_flat)
        attended = attended.reshape(B, C, d)
        return self.head(attended)  # [B, C, n_future]
```

Aspect	Notes
Expected improvement	3-10%
Difficulty	Moderate
Pros	Learns to weight relevant t...
Cons	More parameters; slightly s...

3.5 Multi-Scale Temporal Modeling

Neural signals often have patterns at multiple timescales. Add parallel pathways:

```
class MultiScaleTemporalBlock(nn.Module):
    def __init__(self, d_model, n_heads):
        super().__init__()
        # Fine-grained attention (full sequence)
        self.fine_attn = nn.MultiheadAttention(
            d_model, n_heads, batch_first=True
        )
        # Coarse attention (pooled by 2)
        self.coarse_attn = nn.MultiheadAttention(
            d_model, n_heads, batch_first=True
        )
        self.fusion = nn.Linear(d_model * 2, d_model)
```

Aspect	Notes
Expected improvement	5-12%
Difficulty	Moderate-High
Pros	Captures both fast transien...
Cons	More complex; needs careful...

3.6 Fix the Refinement Loop

Current issue: During refinement, you fill auxiliary features with zeros:

```
pred_features = torch.zeros(B, self.n_future, C, F, device=x.device)
pred_features[:, :, :, 0] = pred  # Only fill in predicted feature[0]
```

This creates a distribution shift-the model was trained on normalized data where all features have ~0 mean and ~1 std, but refinement uses zeros for 8/9 features.

Fix: Predict all features, or use learned embeddings for missing features:

```
# Option 1: Use mean of observed features for predicted timesteps
pred_features[:, :, :, 1:] = x[:, -1:, :, 1:].expand(
    -1, self.n_future, -1, -1
)

# Option 2: Learnable placeholder embeddings
self.placeholder_features = nn.Parameter(torch.zeros(1, 1, 1, F-1))
pred_features[:, :, :, 1:] = self.placeholder_features.expand(
    B, n_future, C, -1
```

```
)
```

Aspect	Notes
Expected improvement	2-5% during inference
Difficulty	Easy
Pros	Reduces train/inference dis...
Cons	Minor change

---

4. Advanced/Research-Level Improvements

4.1 Auxiliary Loss on Frequency Bands

Add a secondary objective to predict frequency bands, not just feature[0]:

```
class MultiTaskLoss(nn.Module):
    def __init__(self, aux_weight=0.1):
        super().__init__()
        self.aux_weight = aux_weight

    def forward(self, pred_main, pred_aux, target_main, target_aux):
        main_loss = F.mse_loss(pred_main, target_main)
        aux_loss = F.mse_loss(pred_aux, target_aux)
        return main_loss + self.aux_weight * aux_loss
```

Aspect	Notes
Expected improvement	5-15%
Difficulty	Moderate (requires model ch...
Pros	Richer gradients; better fe...
Cons	More output heads; need to ...

4.2 Cross-Electrode Attention with Distance Weighting

If you have electrode position information, incorporate spatial priors:

```
# Bias attention by electrode distance
spatial_bias = -distance_matrix / temperature
attn_weights = softmax(QK^T + spatial_bias)
```

Aspect	Notes
Expected improvement	5-10%
Difficulty	High (requires electrode co...
Pros	Physically meaningful atten...
Cons	Needs spatial metadata

4.3 Curriculum Learning for Refinement

Train with increasing refinement iterations:

- Epochs 1-30: 0 iterations
- Epochs 31-60: 1 iteration
- Epochs 61+: 2 iterations

Aspect	Notes

Expected improvement	3-8%
Difficulty	Moderate
Pros	Stable training; model lear...
Cons	More complex training loop

4.4 Per-Channel Normalization Instead of Per-Sample

Current: Normalizes across all channels and timesteps together.

Different electrodes may have very different scales. Per-channel normalization preserves relative magnitudes between electrodes:

```
# Per-channel normalization
mean = x.mean(dim=1, keepdim=True)  # Mean over time only
std = x.std(dim=1, keepdim=True) + eps
x_norm = (x - mean) / std
```

Aspect	Notes
Expected improvement	2-8%
Difficulty	Easy
Pros	Preserves cross-electrode r...
Cons	May be more sensitive to ou...

Summary: Recommended Priority Order

Priority	Change	Expected Gain	Effort
1	Increase model size (d=128,...	5-15%	Trivial
2	Feature-specific encoding	5-15%	Moderate
3	Lower learning rate (3e-4)	3-10%	Trivial
4	Attention-based prediction ...	3-10%	Moderate
5	Causal temporal masking	2-8%	Easy
6	Fix refinement with non-zer...	2-5%	Easy
7	Learnable spatial embeddings	3-8%	Easy
8	Tune smoothness weight	2-5%	Trivial
9	Multi-task auxiliary loss	5-15%	Moderate
10	Enable training with refine...	3-8%	Moderate

Recommendation: Start with #1, #3, and #5-6 as quick wins, then implement #2 and #4 for more substantial architectural improvements.