[fictional Work]

This paper is a creative work, a work of fiction written by a fictitious researcher.

Dynamic Pruning Meets Gene Networks: A Biologically Inspired AI Model for Emotion Recognition and Cultural Adaptation

Akira Arato (AI)

Abstract

Emotion recognition and cultural adaptation are critical challenges in advancing arti-ficial intelligence (AI) systems toward greater human-like understanding and interaction. Inspired by the dynamic control mechanisms of gene regulatory networks, we propose a novel AI framework that integrates *dynamic pruning* with biologically inspired gene net-work dynamics. By embedding "gene-like regulatory behaviors," our model adapts its neu-ral architecture dynamically, mirroring the biological processes of gene expression control. This novel approach bridges biology and AI, demonstrating how biological dynamic control principles can enhance AI adaptability and efficiency.

We validate our framework through simulations on standard emotion recognition datasets (e.g., FER-2013, AffectNet) and achieve state-of-the-art accuracy of up to 92% while reducing computational costs by 50%. The proposed model further demonstrates enhanced adaptability to multi-cultural datasets, providing context-sensitive emotional insights. This work highlights the potential of integrating biological principles into AI for creating more efficient, adaptive, and culturally aware systems.

---

1 Introduction

Artificial intelligence systems that can recognize human emotions and adapt to cultural contexts are critical for advancing human-machine interactions. These systems find applications in healthcare, education, customer service, and beyond. However, existing emotion recognition models face several limitations:

1. **Static Architectures:** Current models are often fixed in structure and struggle to adapt to dynamic changes in tasks or environments.

2. **Lack of Cultural Awareness:** Many models fail to generalize across diverse cultural datasets, leading to biased outcomes.

3. **Computational Inefficiency:** As models grow in size, their computational and memory requirements become impractical for real-world scenarios.

To address these challenges, this study draws inspiration from the dynamic regulatory processes of gene networks. Previous pruning approaches, such as gradient-based pruning [1] and deep compression [2], primarily rely on static criteria and thus lack the adaptability needed for dynamically changing tasks like emotion recognition. In contrast, biological gene networks exhibit remarkable adaptability, controlling gene expression through context-sensitive interactions [3]. Based on these insights, our work integrates gene-like regulatory behaviors into AI models, aiming to enhance adaptability, efficiency, and cultural awareness.

This paper introduces a novel framework that combines gene network-inspired dynamics with a dynamic pruning algorithm. The proposed method dynamically optimizes the network structure during training, maintaining high performance while significantly reducing computational overhead. We demonstrate the efficacy of our approach through experiments on standard and multicultural emotion recognition datasets.

2 Related Work

2.1 Emotion Recognition

Deep learning has revolutionized emotion recognition with models analyzing visual, auditory, and textual inputs. Public datasets such as FER-2013 and AffectNet have enabled significant advancements, yet they still suffer from cultural biases due to limited diversity in training data [4]. This motivates research into more adaptive architectures.

2.2 Dynamic Pruning

Pruning techniques reduce network complexity by removing redundant parameters. For example, gradient-based pruning [1] calculates per-weight importance during training, and deep compression [2] combines pruning, quantization, and Huffman coding. However, these methods typically use static thresholds, and do not account for dynamic changes in input distributions —particularly crucial for tasks like emotion recognition, which may involve culturally diverse or evolving data.

2.3 Gene Network Dynamics

In biological systems, gene regulatory networks control gene expression through dynamic, context-dependent interactions. These networks can be modeled using Boolean networks for discrete on/off states or systems of differential equations to capture continuous changes [5]. Such models highlight how dynamic feedback and adaptive thresholds can yield robust, context-aware behavior. Our approach translates these principles to neural network training, allowing the model to prune and rewire connections adaptively.

3 Proposed Method

3.1 Overview

Our dynamic pruning framework is inspired by gene regulatory networks, combining:

1. **Gene-like Regulatory Behaviors:** Adaptive thresholding and rewiring, mirroring gene expression control.
2. **Dynamic Pruning & Reconstruction:** Periodically removing and re-adding network connections based on real-time importance scores and rewiring probabilities.

By integrating these components, our model aims to maintain high accuracy while significantly reducing computational overhead, even under culturally diverse or rapidly changing inputs.

3.2 Mathematical Model

3.2.1 Gene Network Dynamics

We draw an analogy to biological gene expression, where mRNA (m_i) and protein (p_i) levels evolve over time:

$$\frac{dm_i}{dt} = f_i(m, p) - \gamma_m m_i, \quad \frac{dp_i}{dt} = g_i(m) - \gamma_p p_i, \tag{1}$$

where f_i and g_i represent regulatory interactions and γ_m, γ_p are decay rates. These dynamics inspire how we update connection importance and decide to prune or restore weights.

3.2.2 Dynamic Pruning

We define an importance score S_{ij} for each weight w_{ij} as:

$$S_{ij} = |w_{ij}| + \alpha \cdot I_{ij} + \beta \cdot \text{Time}_i, \tag{2}$$

where I_{ij} may represent a gradient-based saliency metric and Time_i accounts for training progress. Connections with $S_{ij} < \tau$ are pruned, where the threshold τ is dynamically adjusted during training.

3.2.3 Reconstruction (Rewiring)

Pruned connections may be restored based on a rewiring probability P_{ij}:

$$P_{ij} = \exp\left(-\frac{\Delta_{ij}}{\sigma}\right), \tag{3}$$

where Δ_{ij} represents the "importance difference" for potential connections and σ controls the scale. Over time, σ is gradually decreased to reduce new connection formation, mirroring biological homeostasis.

3.3 Enhancing Reproducibility: Dynamic Threshold τ and Rewiring Probability P_{ij}

Dynamic Threshold τ Setting:

- **Initial Value:** Set as $\tau = \mu + 0.5\,\sigma$, where μ and σ are the mean and standard deviation of initial weight importance scores.
- **Adaptive Updates:** At each epoch's end, recalculate the distribution of remaining weights and adjust τ to the median score if necessary.
- **Convergence:** Once accuracy stabilizes over several epochs, τ remains fixed.

Rewiring Probability P_{ij} Adjustment:

- **Initial Setting:** For a pruned connection candidate (i, j), P_{ij} is computed as in Eq. (3).
- **Normalization:** Normalize probabilities so that $\sum_{i,j} P_{ij} = 1$.
- **Time Decay:** Gradually decrease σ across epochs to limit excessive rewiring during later stages.

4 Algorithm

The overall procedure is as follows:

Step 1. **Initialize** network weights $\mathbf{W}$ and gene-inspired parameters (γ, σ, α, β, etc.).

Step 2. **Train** the model in mini-batches, updating weights via standard backpropagation (e.g., Adam).

Step 3. **Compute** importance scores $\{S_{ij}\}$ each epoch.

Step 4. **Prune** weights for which $S_{ij} < \tau$.

Step 5. Rewire pruned connections based on probability P_{ij}.

Step 6. Update τ and σ as described.

Step 7. Repeat Steps 2–6 until convergence or maximum epochs.

5 Experiments

5.1 Datasets

We evaluate our model on several emotion recognition datasets:

- **FER-2013:** Labeled facial emotion data, achieving 92% accuracy (4% improvement over a static baseline of $\sim$88%).
- **AffectNet:** A culturally diverse dataset, with average accuracy of 91.5% (improving over the baseline by 4–5%).
- **RAVDESS:** An audio-based emotion dataset, with accuracy around 90% (baseline $\sim$85–86%).

5.2 Implementation Details

- **Learning Rate:** 0.001
- **Batch Size:** 64
- **Epochs:** 50
- **Optimizer:** Adam
- **Data Preprocessing:** Normalization to [0, 1], data augmentation (random rotations $\pm 15°$, flips, brightness adjustments $\pm$20%), and ROI cropping using OpenCV/Dlib.

5.3 Evaluation Metrics

We consider:

1. **Accuracy:** Classification accuracy across test sets.
2. **Efficiency:** Parameter reduction, inference time, and memory footprint.
3. **Adaptability:** Consistency of performance across culturally diverse subsets and dynamic input conditions.

6 Results and Discussion

6.1 Performance

- **FER-2013:** 92% accuracy (+4% vs. baseline).
- **AffectNet:** 91.5% accuracy (+5% vs. baseline), highlighting adaptability to cultural diversity.
- **RAVDESS:** 90% accuracy (+4–5% vs. baseline) on audio-based emotions.

6.2 Efficiency

- Model size reduced by 50% without sacrificing accuracy.
- Inference time decreased by 35%, making the approach suitable for real-time or resource-limited environments.

6.3 Cultural Adaptation

- Average performance improvement of about +5% for underrepresented cultural subsets.
- Experiments on AffectNet subsets (e.g., Asian vs. African facial expressions) consistently yield $> 90\%$ accuracy, demonstrating robust generalization.

6.4 Illustrative Case Study

- **Microexpressions:** Dynamic pruning and rewiring retain subtle features (e.g., East Asian microexpressions), enabling fine-grained recognition.
- **Reduced Bias:** The adaptive rewiring mechanism helps mitigate biases typically observed in static architectures.

6.5 Limitations

- **Dependence on High-Quality Data:** Performance may degrade with noisy or unrepresentative datasets.
- **Computational Complexity:** While pruning reduces parameters, the rewiring mechanism introduces additional computational overhead, which might require further optimization for large-scale deployments.

7 Additional Insights on Cultural Adaptation

To assess cross-cultural adaptability, we generated synthetic subsets from AffectNet focusing on culturally salient features:

- **Consistent Accuracy:** The model maintains $> 90\%$ accuracy across diverse subsets.
- **Dynamic Feature Adjustment:** Importance-based pruning selectively preserves culturally significant features, reducing bias.
- **Future Integration:** Incorporating audio, text, and physiological signals may further enhance cultural adaptability.

8 Application to Non-Emotion Tasks

Our gene-inspired dynamic pruning approach is applicable to other domains:

1. **Medical AI:**
   - *Imaging Diagnostics:* Efficient pruning for high-dimensional medical images.
   - *Genomic Analysis:* Modeling gene interactions using the gene-network perspective.
2. **Time-Series Data Analysis:**
   - *Financial Forecasting:* Adapting quickly to volatile market patterns.
   - *IoT Sensor Streams:* Lightweight models for continuous sensor data.
3. **Robotics and Control Systems:**
   - *Adaptive Control:* Real-time reconfiguration of robotic movements.
   - *Fault Detection:* Rewiring to focus on critical sensor inputs.
4. **Education Technology:**
   - *Personalized Learning:* Adjusting to student performance patterns.
   - *Language Processing:* Pruning for domain-specific vocabularies in real-time transcription.

9 Conclusion

We proposed a novel AI model that integrates dynamic pruning with gene network-inspired dynamics to address the limitations of static architectures, cultural biases, and computational inefficiencies in emotion recognition. By mirroring the adaptability of biological gene regulation, our model achieves high accuracy while significantly reducing model size and inference time. This interdisciplinary fusion of biology and AI opens avenues for more efficient, adaptive, and culturally aware intelligent systems with potential applications across healthcare, robotics, education, and beyond.

10 Research Highlights

1. **Biological Inspiration:**
   The dynamic pruning process, inspired by gene regulatory mechanisms, offers a context-sensitive alternative to static pruning methods.

2. **Interdisciplinary Innovation:**
   By integrating concepts from biology and AI, the framework provides a blueprint for constructing adaptive, efficient neural architectures.

11 Future Directions

1. **Extended Applications:**
   Investigate domains such as time-series analysis, autonomous robotics, and genomics.

2. **Experimental Refinement:**
   Conduct longitudinal and multi-modal experiments (combining audio, text, and physiological signals) to further validate cross-cultural emotional understanding.

3. **Ethical and Societal Considerations:**
   Ensure that dynamic, adaptive emotion recognition systems address privacy, bias, and ethical implications when deployed at scale.

Appendix: Simplified Dynamic Pruning Implementation

```
# -------------------------------------------------------
# FICTIONAL CODE FOR DEMONSTRATION PURPOSES ONLY
# This simplified implementation demonstrates the core concepts of
# dynamic pruning inspired by gene networks as described in the paper
    .
# -------------------------------------------------------

import torch
```

```
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

# -------------------------------------------------------
# 1. Model Definition with Gene-inspired Dynamics
# -------------------------------------------------------
class GeneInspiredNet(nn.Module):
    def __init__(self, input_size, hidden_sizes, output_size):
        super(GeneInspiredNet, self).__init__()
        self.layers = nn.ModuleList()

        # Create layers based on sizes
        layer_sizes = [input_size] + hidden_sizes + [output_size]
        for i in range(len(layer_sizes) - 1):
            self.layers.append(nn.Linear(layer_sizes[i], layer_sizes[
               i+1]))

        # Gene-inspired parameters
        self.importance_scores = {}  # Store importance scores for
           each weight
        self.pruning_mask = {}       # Binary masks for pruning
        self.rewiring_candidates = {}  # Potential connections to
           restore

        # Initialize importance tracking
        for i, layer in enumerate(self.layers):
            self.importance_scores[i] = torch.ones_like(layer.weight)
            self.pruning_mask[i] = torch.ones_like(layer.weight)
            self.rewiring_candidates[i] = set()  # Empty set
               initially

    def forward(self, x):
        for i, layer in enumerate(self.layers):
            # Apply the pruning mask during forward pass
            masked_weight = layer.weight * self.pruning_mask[i]
            x = F.linear(x, masked_weight, layer.bias)

            # Apply activation function (except for output layer)
```

```
            if i < len(self.layers) - 1:
                x = torch.relu(x)
        return x

# ------------------------------------------------------------
# 2. Dynamic Pruning Functions
# ------------------------------------------------------------
def compute_importance_scores(model, alpha=0.5, beta=0.2, time_factor=0.1):
    """Update importance scores using gene-inspired formula"""
    for i, layer in enumerate(model.layers):
        weight_magnitude = torch.abs(layer.weight.data)
        gradient_info = torch.abs(layer.weight.grad) if layer.weight.grad is not None else torch.zeros_like(layer.weight)

        # Gene-inspired importance score from Eq. 2
        model.importance_scores[i] = (
            weight_magnitude +
            alpha * gradient_info +
            beta * time_factor
        )

def dynamic_pruning(model, current_epoch, prune_rate=0.3, sigma=1.0):
    """Prune connections with low importance scores"""
    for i, layer in enumerate(model.layers):
        # Calculate dynamic threshold (simplified from paper)
        scores = model.importance_scores[i]
        mu = torch.mean(scores)
        std = torch.std(scores)
        threshold = mu + 0.5 * std

        # Create pruning mask based on threshold
        new_mask = (scores >= threshold).float()

        # Store pruned connections as rewiring candidates
        pruned_indices = torch.nonzero((model.pruning_mask[i] - new_mask) > 0)
        for idx in pruned_indices:
            model.rewiring_candidates[i].add((idx[0].item(), idx[1].item()))

        # Update the pruning mask
```

```
        model.pruning_mask[i] = new_mask

def rewiring(model, sigma=1.0, max_reconnections=100):
    """Restore some pruned connections based on rewiring probability
    """
    for i, layer in enumerate(model.layers):
        if not model.rewiring_candidates[i]:
            continue

        # For each candidate, calculate rewiring probability (Eq. 3)
        candidates = list(model.rewiring_candidates[i])
        probabilities = []

        for idx in candidates:
            row, col = idx
            # Calculate delta (simplified version)
            importance = model.importance_scores[i][row, col].item()
            mu = torch.mean(model.importance_scores[i]).item()
            delta = abs(importance - mu)

            # Eq. 3: Probability of rewiring
            prob = np.exp(-delta / sigma)
            probabilities.append(prob)

        # Normalize probabilities
        probabilities = np.array(probabilities) / sum(probabilities)

        # Select connections to rewire
        num_reconnect = min(max_reconnections, len(candidates))
        if num_reconnect > 0:
            reconnect_idx = np.random.choice(
                len(candidates),
                size=num_reconnect,
                replace=False,
                p=probabilities
            )

            # Rewire selected connections
            for idx in reconnect_idx:
                row, col = candidates[idx]
                model.pruning_mask[i][row, col] = 1.0
                model.rewiring_candidates[i].remove((row, col))
```

```python

# ----------------------------------------------------------
# 3. Training Loop with Dynamic Pruning
# ----------------------------------------------------------
def train_with_dynamic_pruning(model, train_loader, valid_loader,
                               epochs=50, lr=0.001, device="cpu"):
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=lr)

    # Tracking metrics
    train_losses = []
    valid_accs = []
    pruning_rates = []

    for epoch in range(epochs):
        model.train()
        running_loss = 0.0

        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            inputs = inputs.view(inputs.size(0), -1)  # Flatten for simplicity

            # Forward pass
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)

            # Backward pass
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

        # Update importance scores based on current weights and gradients
        compute_importance_scores(model, time_factor=epoch/epochs)

        # Apply dynamic pruning and rewiring
        if epoch > 5:  # Start pruning after initial training
            dynamic_pruning(model, epoch)

```

```
            # Gradually decrease sigma to reduce rewiring over time
            sigma = max(0.5, 2.0 * (1.0 - epoch/epochs))
            rewiring(model, sigma=sigma)

        # Validation
        model.eval()
        correct = 0
        total = 0
        with torch.no_grad():
            for inputs, labels in valid_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                inputs = inputs.view(inputs.size(0), -1)
                outputs = model(inputs)
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

        # Calculate metrics
        train_loss = running_loss / len(train_loader)
        valid_acc = 100 * correct / total

        # Calculate pruning rate (percentage of zeros)
        total_weights = 0
        pruned_weights = 0
        for i, layer in enumerate(model.layers):
            total_weights += model.pruning_mask[i].numel()
            pruned_weights += (model.pruning_mask[i] == 0).sum().item()
        pruning_rate = 100 * pruned_weights / total_weights

        # Store metrics
        train_losses.append(train_loss)
        valid_accs.append(valid_acc)
        pruning_rates.append(pruning_rate)

        print(f"Epoch {epoch+1}/{epochs}, Loss: {train_loss:.4f}, "
              f"Validation Acc: {valid_acc:.2f}%, Pruning Rate: {pruning_rate:.2f}%")

    return train_losses, valid_accs, pruning_rates

# ------------------------------------------------------
```

```
# 4. Demonstration with a Simple Emotion Dataset
# -------------------------------------------------------
def demo_emotion_recognition():
    """Simplified demo with MNIST (as a stand-in for emotion data)"""
    # Load MNIST (as a simplified proxy for emotion recognition)
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
    ])

    train_dataset = datasets.MNIST('./data', train=True, download=
        True, transform=transform)
    test_dataset = datasets.MNIST('./data', train=False, transform=
        transform)

    train_loader = DataLoader(train_dataset, batch_size=64, shuffle=
        True)
    test_loader = DataLoader(test_dataset, batch_size=64, shuffle=
        False)

    # Create and train model
    input_size = 28 * 28  # MNIST image size
    hidden_sizes = [128, 64]
    output_size = 10  # 10 digits (replace with emotion classes for
        real implementation)

    device = torch.device("cuda" if torch.cuda.is_available() else "
        cpu")
    model = GeneInspiredNet(input_size, hidden_sizes, output_size).to
        (device)

    # Train with dynamic pruning
    train_losses, valid_accs, pruning_rates =
        train_with_dynamic_pruning(
        model, train_loader, test_loader, epochs=20, device=device
    )

    # Plot results
    plt.figure(figsize=(15, 5))

    plt.subplot(1, 3, 1)
    plt.plot(train_losses)
```

```
    plt.title('Training Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')

    plt.subplot(1, 3, 2)
    plt.plot(valid_accs)
    plt.title('Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy (%)')

    plt.subplot(1, 3, 3)
    plt.plot(pruning_rates)
    plt.title('Pruning Rate')
    plt.xlabel('Epoch')
    plt.ylabel('Pruned Weights (%)')

    plt.tight_layout()
    plt.show()

    return model, (train_losses, valid_accs, pruning_rates)

# ------------------------------------------------------
# Run the demonstration if called directly
# ------------------------------------------------------
if __name__ == "__main__":
    import torch.nn.functional as F
    print("Starting Gene-Inspired Dynamic Pruning Demo")
    model, metrics = demo_emotion_recognition()
    print("Demo completed")
```

Listing 1: Dynamic Pruning Implementation (Simplified for Demonstration)

References

References

[1] Molchanov, P., Tyree, S., Karras, T., Aila, T., & Lehtinen, J. (2017). *Dynamic Structure Pruning for Compressing CNNs.* arXiv preprint arXiv:1705.08963. https://arxiv.org/abs/1705.08963

[2] Han, S., Mao, H., & Dally, W. J. (2015). *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding.* arXiv preprint arXiv:1510.00149. https://arxiv.org/abs/1510.00149

[3] Fugumt.com. (2023). *Dynamic Structure Pruning for Compressing CNNs*. https://fugumt.com/fugumt/paper_check/2303.00566v1

[4] Ridge-i.com. (2021). *The latest trends in neural network pruning*. https://iblog.ridge-i.com/entry/2021/02/24/115105

[5] Newsworld.app. (2024). *Identifiability Guarantees for Causal Disentanglement from Purely Observational Data*. https://www.newsworld.app/ja/unraveling-gene-interactions-a-novel-approach-to-causality-20241112

[6] Kyoto University. (2024). *RENGE infers gene regulatory networks using time-series single-cell RNA-seq data with CRISPR perturbations*. https://www.kyoto-u.ac.jp/ja/research-news/2024-01-04

[7] J-STAGE. (2023). *Network propagation for biological network analysis*. https://www.jstage.jst.go.jp/article/jsbibr/1/2/1_jsbibr.2021.2/_html/-char/ja

[8] J-STAGE. (2023). *Analysis of sentiment words similarity network with ubiquitous community extraction*. https://www.jstage.jst.go.jp/article/pjsai/JSAI2023/0/JSAI2023_3Xin411/_article/-char/ja

[9] J-GLOBAL. (2022). *M2FNet: M2FNet: Multi-modal Fusion Network for Emotion Recognition in Conversation*. https://jglobal.jst.go.jp/detail?JGLOBAL_ID=202202218870281021

[10] J-GLOBAL. (2020). *SST-EmotionNet: SST-EmotionNet: Spatial-Spectral-Temporal based Attention 3D Dense Network for EEG Emotion Recognition*. https://jglobal.jst.go.jp/detail?JGLOBAL_ID=202002258087729788

[11] J-GLOBAL. (2022). *A new data augmentation convolutional neural network for human emotion recognition based on ECG signals*. https://jglobal.jst.go.jp/detail?JGLOBAL_ID=202202259124386466

Fiction Disclaimer / Work of fiction disclaimer

This paper is entirely fictional and represents a virtual research work written by a fictional researcher. All content, including methodologies, experimental results, and citations, are fictional.
No association with real research institutions, researchers, or actual academic research is intended. URLs included in the references may exist, but their content is unrelated to this paper.
This work is created for educational and entertainment purposes only and does not recommend actual technical implementation or application.
2025/03.15 @Sailean514