UNIVERSITY OF MORATUWA

Faculty of Engineering

Registered Module No: CS4690

Prospective coding as an alternative to back-propagation

Final Report

Date of Submission:

November 13, 2024

Team Members

190346A - Laksika Tharmalingam
190287R - Kajanan Selvanesan
190019B - Abinesh Thaventhirarajah
190049P - Kesavi Aravinthan

Department of:

Computer Science and Engineering

November 13, 2024

Contents

1 Introduction

Any learning system needs to figure out how each of its parameters or components contribute to the error at its output. This is known as the credit assignment problem. For a neural network, this amounts to determining the change in the error due to a change of an individual weight [2]. Back-propagation [3] is the most widely used algorithm to solve this problem in artificial neural networks. However, it is not considered to be biologically plausible [2]. In this work, we will consider a more biologically plausible learning algorithm called prospective configuration [4]. The key idea here is that neurons in a network adjust their activity in such a way so that the output neurons better predict the desired output. Subsequently, the weights are updated to consolidate this activity. Note that this is in reverse order to back-propagation, where the weights are updated first and the changes in neural activity follow. Unlike back-propagation, prospective configuration relies only on local computations that are better supported by biological evidence.

2 Background

Neural networks, inspired by the brain's functioning, aim to solve the credit assignment problem—attributing error to individual weights for learning. Back-propagation [2], while effective in optimizing artificial neural networks, raises concerns regarding biological plausibility due to its global error propagation and weight adjustment mechanisms. In biological brains, the mechanisms of learning and adaptation appear to involve more localized and activity-dependent processes rather than global error signals and weight updates across layers simultaneously.

Prospective coding [4] emerges as a biologically inspired alternative to address these concerns. The key idea behind prospective coding is to emulate the brain's approach where neurons adjust their activity based on predictive errors before updating synaptic strengths (weights). This approach mirrors a predictive coding framework observed in neurobiology, where neural circuits anticipate sensory inputs and refine their internal representations accordingly. By focusing on local computations and prediction errors, prospective coding offers a more biologically plausible model for learning in neural networks.

Research by Song et al. [4] has shown promising results in various learning scenarios, demonstrating that prospective coding can achieve lower test errors compared to back-propagation, especially in settings with limited data, online learning environments, and tasks involving continual learning or adaptation to changing environments. These findings suggest that prospective coding not only aligns better with biological principles but also offers practical advantages in terms of adaptability and efficiency in learning tasks.

Understanding and comparing these two paradigms—back-propagation and prospective coding—can provide valuable insights into the design and optimization of neural networks, potentially leading to more robust and efficient learning algorithms that better mimic biological learning processes.

3 Previous Work

Song et al. [4] have conducted experiments comparing prospective configuration against back-propagation in both supervised and reinforcement learning setups. For supervised learning, fully connected (FCNs) as well as convolutional neural networks

(CNNs) have been evaluated. Their experiments show that prospective configuration achieves lower test error than back-propagation in cases such as learning with small amounts of data, online learning, learning multiple tasks continually, and learning in changing environments [4]. However, the networks they have used are rather shallow and small (for FCNs, 4 layers with 32 neurons each and for CNNs, 2 convolutional layers and a fully connected layer).

4 Project Overview

Our project aims to explore prospective coding as a biologically plausible alternative to back-propagation in neural networks, particularly focusing on enhancing and evaluating its performance in challenging deep learning tasks. Inspired by recent advancements and the biological principles of neural adaptation, we plan to extend existing architectures beyond their conventional limits.

Our primary objective is to delve deeper and wider into the architectural complexity of neural networks. We intend to construct models with up to 5 convolutional layers, incorporating additional components such as max pooling, dropout layers, and batch normalization. By systematically increasing the depth and breadth of these networks, we aim to assess how prospective configuration adapts and optimizes in complex learning scenarios.

A key aspect of our project involves comparing the performance of prospective configuration against traditional back-propagation. We will conduct comprehensive experiments to evaluate and benchmark these algorithms across various tasks, emphasizing their efficacy in learning from diverse datasets and adapting to changing environments.

Furthermore, our project extends into the domain of transfer learning, where we plan to leverage the learned representations from our enhanced prospective configuration models. Specifically, we aim to apply transfer learning techniques to a challenging medical imaging task—covid detection. By initializing our models with pretrained weights from our optimized prospective configuration, we aim to enhance learning efficiency and improve performance on this critical healthcare application.

Through these endeavors, we seek not only to validate the biological plausibility of prospective coding but also to contribute insights towards developing more robust and adaptive neural network architectures. Our project's findings will illuminate the potential of prospective coding in advancing the field of deep learning, with implications for both theoretical neuroscience and practical applications in healthcare and beyond.

5 Experimental Setup

To evaluate the efficacy of prospective coding versus back-propagation, we conducted experiments using the CIFAR-10 dataset, a widely used benchmark in image classification tasks. Due to resource constraints, we limited our dataset to 100 samples per class, ensuring a manageable scale while maintaining diversity across classes.

5.1 Datasets

- **CIFAR-10**: A dataset comprising 10 classes of natural images, each with 60,000 32x32 color images in total [1].

- **COVID-CT-Dataset**: For transfer learning experiments about COVID-19, we utilized the COVID-CT dataset [5]. This dataset contains 349 COVID-19 CT images from 216 patients and 463 non-COVID-19 CT images, validated for AI-based diagnosis models.

5.2 Training Parameters

We explored a range of hyperparameters to optimize model performance:

5.2.1 Initial Experimentation for Pre training

- Learning Rates: Tested initial learning rates of **0.1, 0.01, 0.001, and 0.0001** to determine the optimal rate for convergence and accuracy.
- Iterations: Trained models for up to **21 epochs** initially to observe convergence and stability across different learning rates.
- Batch Size: Used a batch size of **20** for all experiments.
- Dataset Sizes: Used **100** samples per class to maintain a manageable scale and diversity across classes.

5.2.2 Further Assessment for Pre training

After selecting the best experiments, we conducted further evaluations with the following adjustments:

- Learning Rates: Extended our evaluation to include learning rates of **0.00001 and 0.000001** to refine our model's performance.
- Iterations: Increased the training duration to **42 epochs** to ensure comprehensive assessment of model convergence and stability.
- Batch Size: Increased the batch size to **100** to analyze its impact on model training and performance.
- Dataset Sizes: Explored larger dataset sizes, specifically **500** samples per class, to evaluate scalability and performance improvements.

This structured approach allowed us to systematically refine our model and assess its performance across a broader range of conditions.

5.3 Hardware and Frameworks

Our experiments were conducted using Google Colab, leveraging the powerful NVIDIA T4 GPU for accelerated training. The choice of Colab allowed us to manage resource-intensive computations effectively within a collaborative environment.

5.4 Transfer Learning for Covid Detection using Lung Scans

For the transfer learning aspect of our project, we utilized the COVID-CT dataset [5], specifically curated for covid detection tasks. This dataset was chosen to assess the applicability and performance gains of prospective coding in medical imaging tasks.

By systematically varying these parameters and utilizing state-of-the-art infrastructure, we aimed to conduct rigorous evaluations that shed light on the comparative advantages of prospective coding over traditional back-propagation in both standard and specialized learning tasks.

6 Experiments

In our study, We conducted a series of **seven** experiments aimed at optimizing the performance of neural networks by systematically adding convolutional layers, linear layers, batch normalization, and max pooling. After thoroughly evaluating the results of these seven experiments, we found that Experiment 3 performed the best. Table 1 shows the differences of each experiment than the previous experience.

Building on the success of Experiment 3, we proceeded with further enhancements, resulting in three additional experiments labeled as 3a, 3b, and 3c. These subsequent experiments focused on refining the architecture and training parameters of Experiment 3 to achieve even better performance. Table 2 shows modifications made further from the experiment 3.

Experiment	**New Layers/Changes**
1	2 Conv2d layers (without any pooling layers) 2 Linear layers
2	Pooling layers added to Convolutional layers
3	Number of convolution layer incresed by 1
4	Number of convolution layer incresed by 1 Number of linear layer incresed by 1
5	Droput layer adder Batch Normalization layer added
6	PCLayer location changed
7	Number of convolution layer incresed by 1 Number of linear layer decreased by 1

Table 1: Comparison of each experiment

Experiment	**New Layers/Changes**
3 a	Hyperparameter optimization to Exp3
3 b	Normalization and Dropout layers added to Exp3
3 c	PCLayer location changed from Exp 3(b)

Table 2: More experiments with best results of Exp3

6.1 Experiment 1: Deepening Architecture

6.1.1 Model Architecture

The model architecture used in this experiment involves deeper convolutional layers followed by fully connected layers for classification. Both back-propagation and prospective coding configurations are tested.

- **Back-Propagation Configuration**:
  - **Convolutional Layers**:
    - `Conv2d`: 3 input channels, 64 output channels, kernel size 3x3, stride 2, padding 1.
    - `Conv2d`: 64 input channels, 128 output channels, kernel size 3x3, stride 2, padding 1.

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
  (1): ReLU()
  (2): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
  (3): ReLU()
  (4): Flatten(start_dim=1, end_dim=-1)
  (5): Linear(in_features=8192, out_features=512, bias=True)
  (6): ReLU()
  (7): Linear(in_features=512, out_features=10, bias=True)
)
```

Figure 1: Back propagation model architecture used in Experiment 1.

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
  (1): PCLayer()
  (2): ReLU()
  (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
  (4): PCLayer()
  (5): ReLU()
  (6): Flatten(start_dim=1, end_dim=-1)
  (7): Linear(in_features=8192, out_features=512, bias=True)
  (8): PCLayer()
  (9): ReLU()
  (10): Linear(in_features=512, out_features=10, bias=True)
)
```

Figure 2: Prospective configuration model architecture used in Experiment 1.

  - **Fully Connected Layers**:
    - `Linear`: 8192 input features (from flattened convolutions), 512 output features.
    - `Linear`: 512 input features, 10 output classes (for CIFAR-10).

- **Prospective Coding Configuration**:
  Additinal to the above layers, here we added Predictive Coding Layers.
  - **Predictive Coding Layers**:
    - `PCLayer`: Placed between convolutional and fully connected layers to implement predictive coding mechanism.

6.1.2 Observations

- There is a high change in test classification error where we increase the value of the learning rate after 0.0001 for both PC and BP.
- There is not much of a change in the test classification error throughout the iterations for all the Learning rate values

6.2 Experiment 2: Deepening Architecture

6.2.1 Model Architecture

The model architecture used in this experiment involves deeper convolutional layers followed by fully connected layers for classification. Both back-propagation and prospective coding configurations are tested.

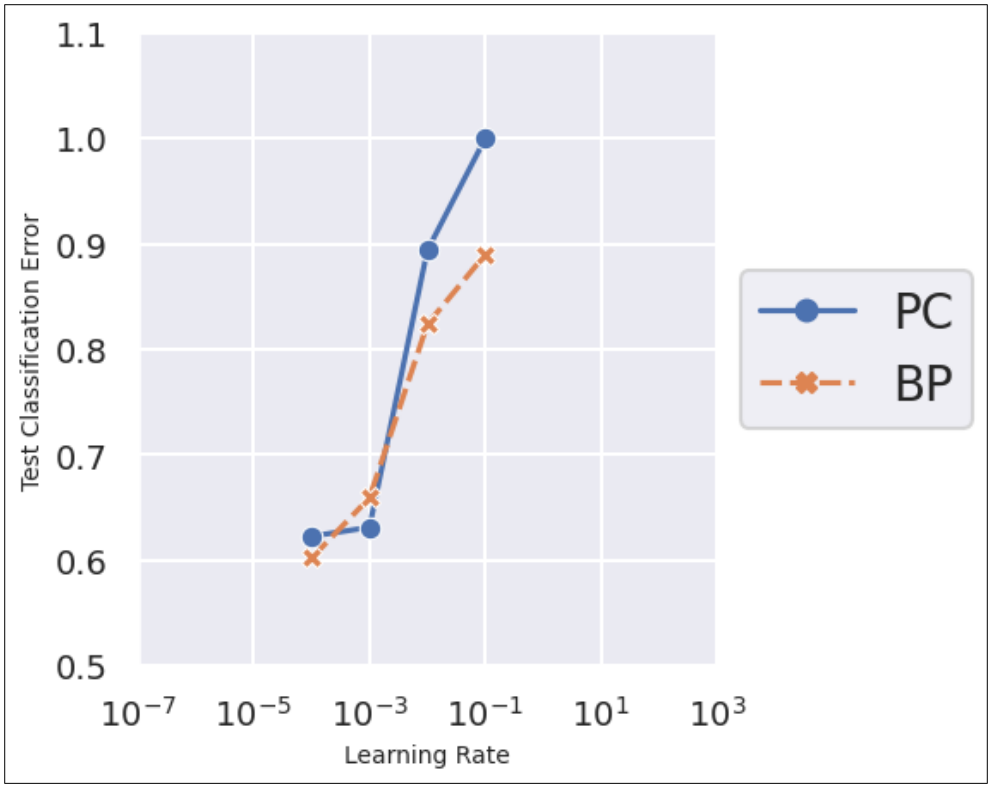
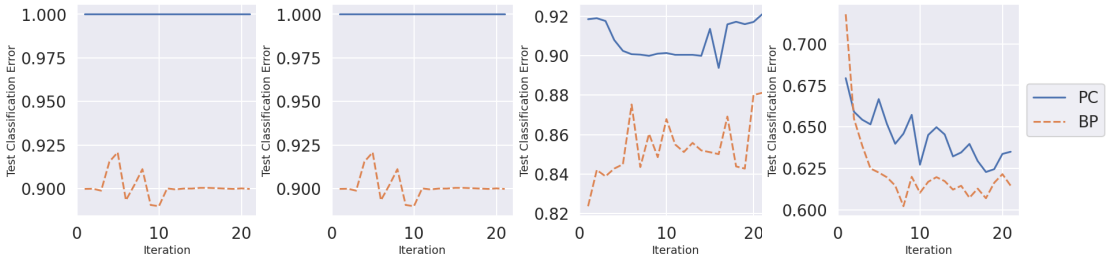


Figure 3: Learning rate vs. test classification error for Experiment 1.

Figure 4: Iteration vs. test classification error for Experiment 1.



(a) Lr: 0.1 (b) Lr: 0.01 (c) Lre: 0.001 (d) Lr: 0.0001

Figure 5: Exp 1: Iteration vs. Test Classification Error

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (1): ReLU()
  (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (4): ReLU()
  (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (6): Flatten(start_dim=1, end_dim=-1)
  (7): Linear(in_features=8192, out_features=512, bias=True)
  (8): ReLU()
  (9): Linear(in_features=512, out_features=10, bias=True)
)
```

Figure 6: Back propagation model architecture used in Experiment 2.

- **Back-Propagation Configuration**:
  - **Convolutional Layers**:
    - `Conv2d`: 3 input channels, 64 output channels, kernel size 3x3, stride 1, padding 1.
    - `Conv2d`: 64 input channels, 128 output channels, kernel size 3x3, stride 1, padding 1.
  - **Fully Connected Layers**:
    - `Linear`: 8192 input features (from flattened convolutions), 512 output features.
    - `Linear`: 512 input features, 10 output classes (for CIFAR-10).
- **Prospective Coding Configuration**:
  Additinal to the above layers, here we added Predictive Coding Layers.
  - **Predictive Coding Layers**:
    - `PCLayer`: Placed between convolutional and fully connected layers to implement predictive coding mechanism.
    - `ReLU`: Activation function used after each PC layer.

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (1): PCLayer()
  (2): ReLU()
  (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (5): PCLayer()
  (6): ReLU()
  (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (8): Flatten(start_dim=1, end_dim=-1)
  (9): Linear(in_features=8192, out_features=512, bias=True)
  (10): PCLayer()
  (11): ReLU()
  (12): Linear(in_features=512, out_features=10, bias=True)
)
```

Figure 7: Prospective configuration model architecture used in Experiment 2.

6.2.2 Observations

- There is a high change in test classification error where we increase the value of the learning rate after 0.0001 for both PC and BP.
- There is no change in test classification error for learning rate values 0.1 and 0.01 throughout the iteration. But for other values the error keeps changing in zig-zag for every iteration

6.3 Experiment 3: Deepening Architecture

6.3.1 Model Architecture

The model architecture used in this experiment involves deeper convolutional layers followed by fully connected layers for classification. Both back-propagation and prospective coding configurations are tested.

- **Back-Propagation Configuration**:
  - **Convolutional Layers**:
    - `Conv2d`: 3 input channels, 64 output channels, kernel size 5x5, stride 1, padding 2.
    - `Conv2d`: 64 input channels, 128 output channels, kernel size 3x3, stride 1, padding 1.
    - `Conv2d`: 128 input channels, 256 output channels, kernel size 3x3, stride 1, padding 1.
  - **Fully Connected Layers**:
    - `Linear`: 4096 input features (from flattened convolutions), 512 output features.
    - `Linear`: 512 input features, 10 output classes (for CIFAR-10).
- **Prospective Coding Configuration**:
  Additinal to the above layers, here we added Predictive Coding Layers.
  - **Predictive Coding Layers**:

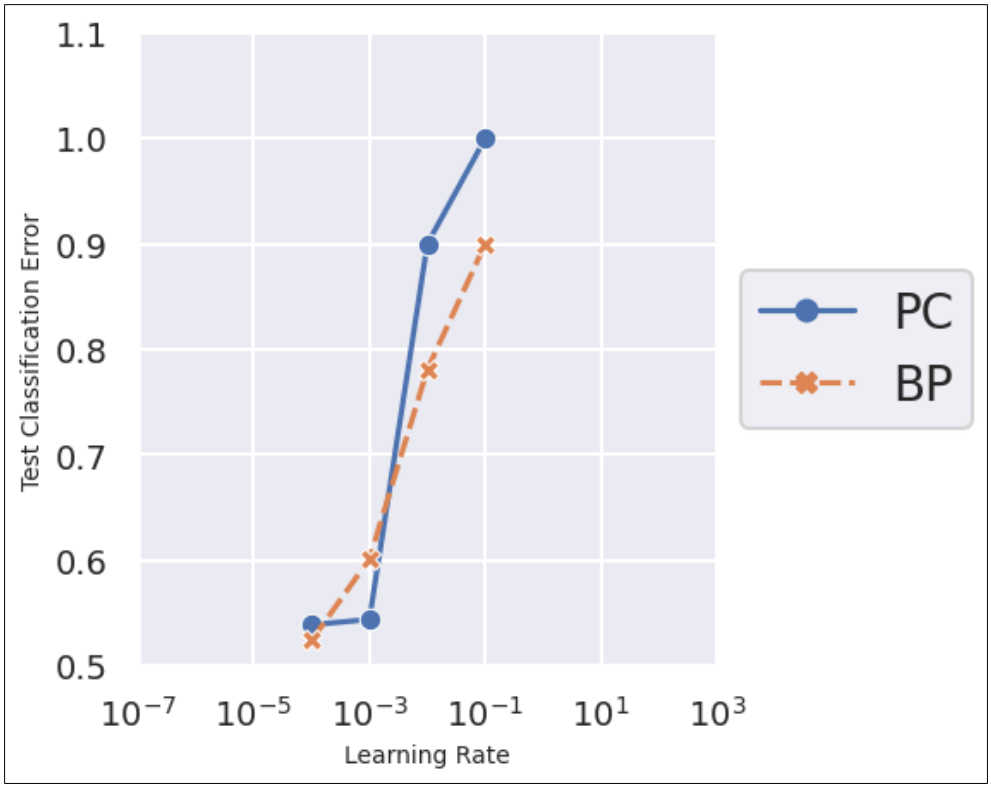


Figure 8: Learning rate vs. test classification error for Experiment 2.

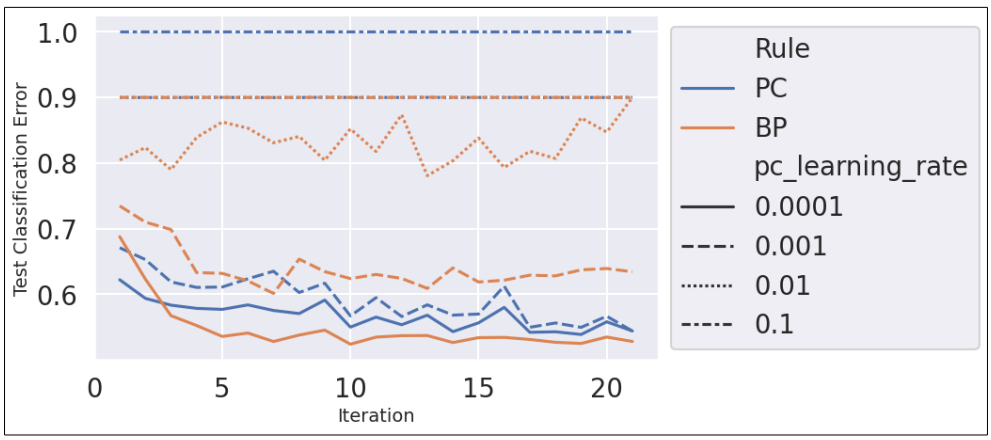
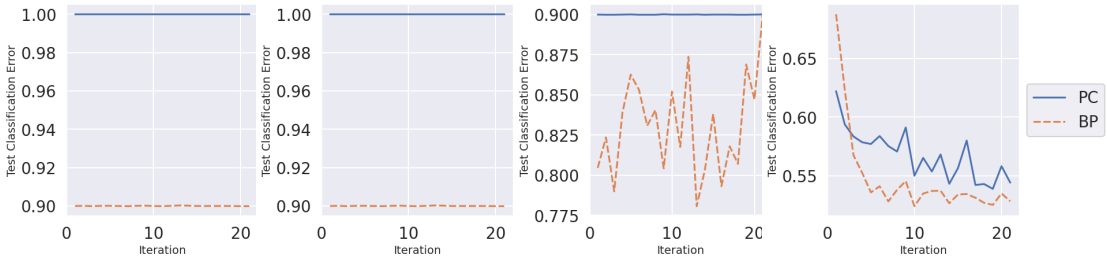


Figure 9: Iteration vs. test classification error for Experiment 2.



(a) Lr: 0.1 (b) Lr: 0.01 (c) Lr: 0.001 (d) Lr: 0.0001

Figure 10: Exp 2 : Iteration vs. Test Classification Error

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bias=False)
  (1): ReLU()
  (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (4): ReLU()
  (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (6): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (7): ReLU()
  (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (9): Flatten(start_dim=1, end_dim=-1)
  (10): Linear(in_features=4096, out_features=512, bias=True)
  (11): ReLU()
  (12): Linear(in_features=512, out_features=10, bias=True)
)
```

Figure 11: Back propagation model architecture used in Experiment 3.

* `PCLayer`: Placed between convolutional and fully connected layers to implement predictive coding mechanism.
* `ReLU`: Activation function used after each PC layer.

6.3.2 Observations

- There is a high change in test classification error where we increase the value of the learning rate after 0.0001 for both PC and BP.
- There is no change in test classification error for learning rate values 0.1 and 0.01 after the some initial iterations. For Lre value 0.001 both BP and PC error values goes the same values after some initial iterations. For learning rate 0.0001 the error follows a zig-zag patter for continues iterations.

6.4 Experiment 4: Deepening Architecture

6.4.1 Model Architecture

The model architecture used in this experiment involves deeper convolutional layers followed by fully connected layers for classification. Both back-propagation and prospective coding configurations are tested.

- **Back-Propagation Configuration**:

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bias=False)
  (1): PCLayer()
  (2): ReLU()
  (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (5): PCLayer()
  (6): ReLU()
  (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (8): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (9): PCLayer()
  (10): ReLU()
  (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (12): Flatten(start_dim=1, end_dim=-1)
  (13): Linear(in_features=4096, out_features=512, bias=True)
  (14): PCLayer()
  (15): ReLU()
  (16): Linear(in_features=512, out_features=10, bias=True)
)
```

Figure 12: Prospective configuration model architecture used in Experiment 3.

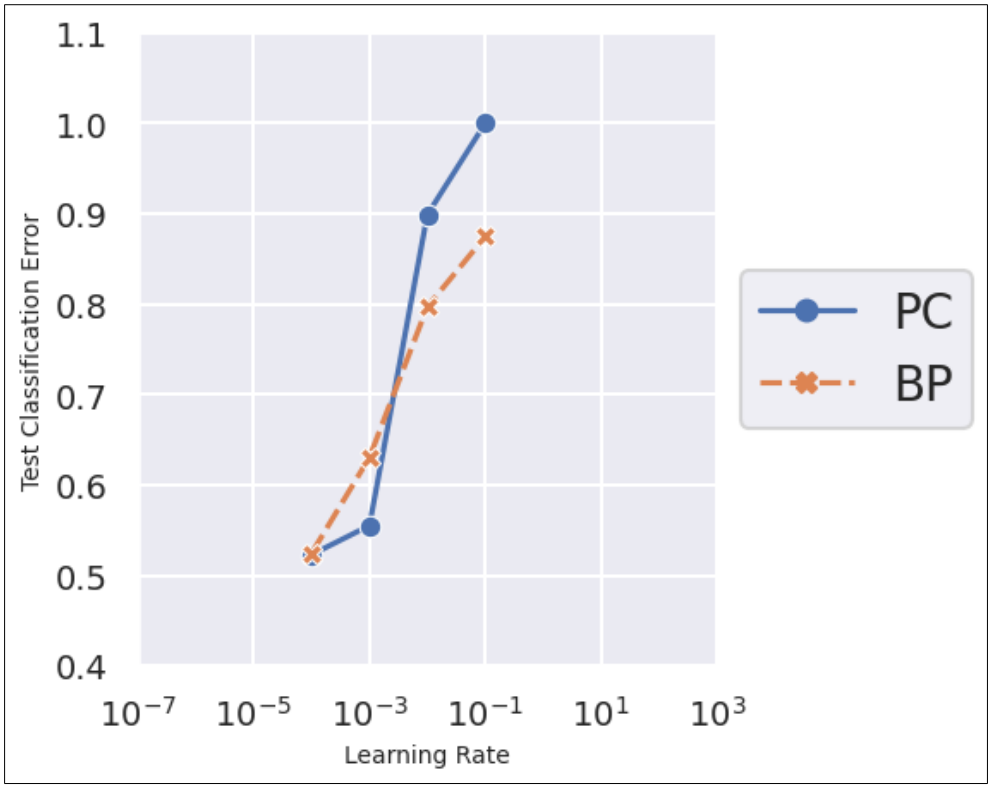


Figure 13: Learning rate vs. test classification error for Experiment 3.

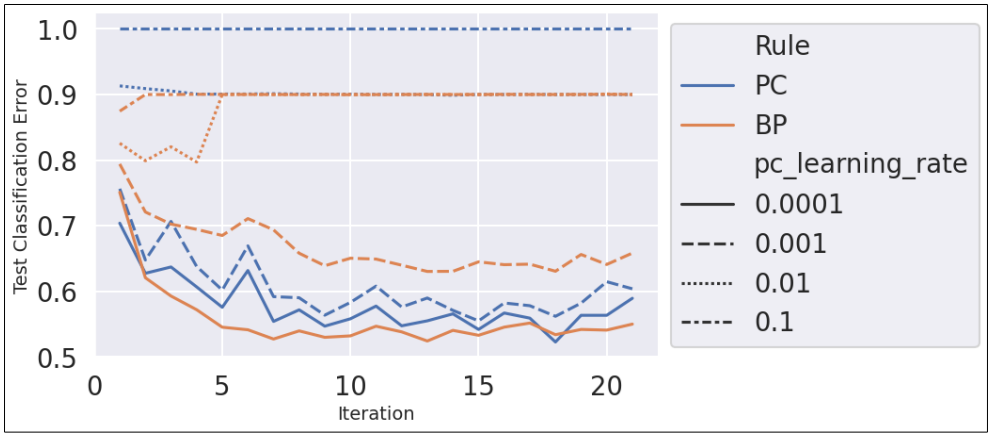
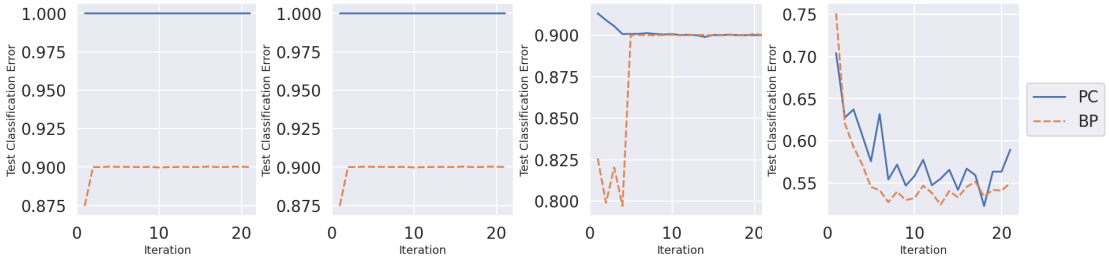


Figure 14: Iteration vs. test classification error for Experiment 3.



(a) Lr: 0.1 (b) Lr: 0.01 (c) Lre: 0.001 (d) Lr: 0.0001

Figure 15: Exp 3: Iteration vs. Test Classification Error

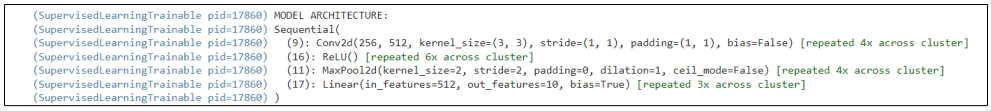


Figure 16: Back propagation model architecture used in Experiment 4.

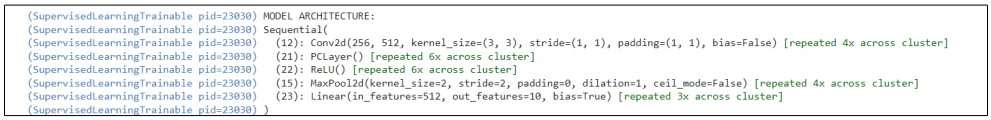


Figure 17: Prospective configuration model architecture used in Experiment 4.

  - **Convolutional Layers**:
    - `Conv2d`: 256 input channels, 512 output channels, kernel size 3x3, stride 1, padding 1.
  - **Fully Connected Layers**:
    - `Linear`: 8192 input features (from flattened convolutions), 512 output features.
    - `Linear`: 512 input features, 10 output classes (for CIFAR-10).

- **Prospective Coding Configuration**:
  Additinal to the above layers, here we added Predictive Coding Layers.
  - **Predictive Coding Layers**:
    - `PCLayer`: Placed between convolutional and fully connected layers to implement predictive coding mechanism.
    - `ReLU`: Activation function used after each PC layer.

6.4.2 Observations

- There is a high change in test classification error where we increase the value of the learning rate after 0.0001 for both PC and BP.
- For Lr 0.1 and 0.01 the error follows a similar patter and the error is constent after some initial iterations. For Lr 0.001 the error settles in a Constance value for both PC and BP at 0.9. For BP ,PC the error keeps decreasing for each iteration For Lr value 0.0001.

6.5 Experiment 5: Deepening Architecture

6.5.1 Model Architecture

The model architecture used in this experiment involves deeper convolutional layers followed by fully connected layers for classification. Both back-propagation and prospective coding configurations are tested.

- **Back-Propagation Configuration**:
  - **Convolutional Layers**:

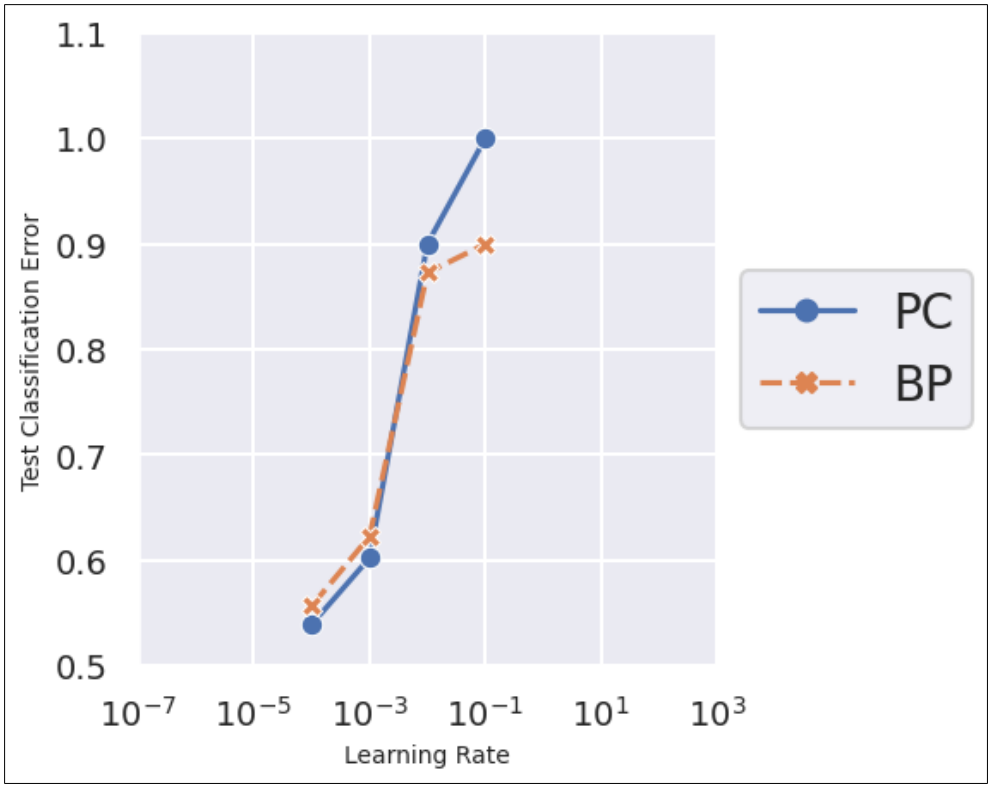


Figure 18: Learning rate vs. test classification error for Experiment 4.

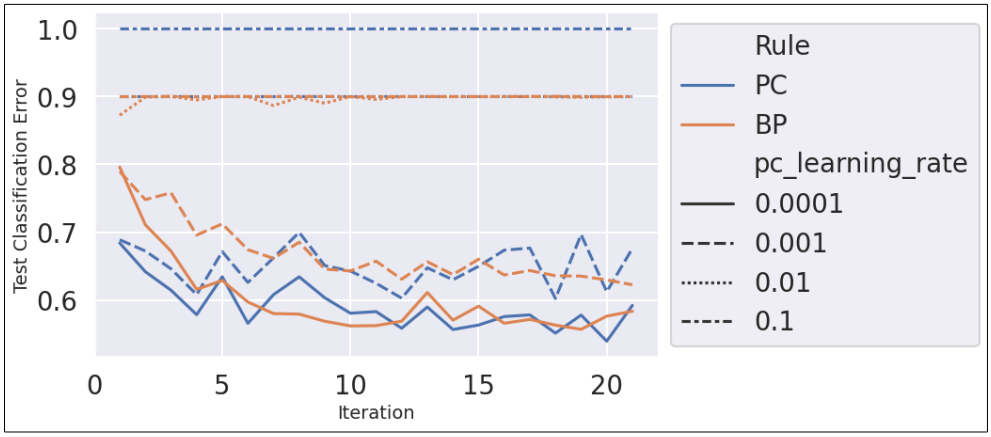
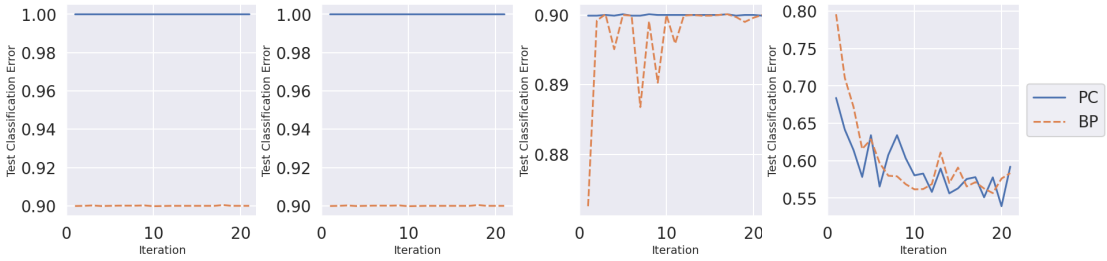


Figure 19: Iteration vs. test classification error for Experiment 4.



(a) Lr: 0.1 (b) Lr: 0.01 (c) Lre: 0.001 (d) Lr: 0.0001

Figure 20: Exp 4: Iteration vs. Test Classification Error

  - `Conv2d`: 3 input channels, 64 output channels, kernel size 5x5, stride 2, padding 1.
  - `Conv2d`: 64 input channels, 128 output channels, kernel size 3x3, stride 1, padding 1.
  - `Conv2d`: 128 input channels, 256 output channels, kernel size 3x3, stride 1, padding 1.
  - `Conv2d`: 256 input channels, 512 output channels, kernel size 3x3, stride 1, padding 1.
  - `Dropout2d`, `BatchNorm2d`, `ReLU`, `MaxPool2d` added in between each `Conv2d` layers.
  - `Dropout2d`: probablity 0.5
  - `BatchNorm2d`: 64 input channels.
  - `ReLU`: Activation function used after BatchNorm Layer.
  - `MaxPool2d`: kernel 2, stride 2, padding 0, dilation 1.
- **Fully Connected Layers**:
  - `Linear`: 2048 input features (from flattened convolutions), 1024 output features.
  - `Linear`: 1024 input features (from flattened convolutions), 512 output features.
  - `Linear`: 512 input features, 10 output classes (for CIFAR-10).

- **Prospective Coding Configuration**:
  Additinal to the above layers, here we added Predictive Coding Layers.
  - **Predictive Coding Layers**:
    - `PCLayer`: Placed between convolutional and fully connected layers to implement predictive coding mechanism.
    - `ReLU`: Activation function used after each PC layer.

6.5.2 Observations

- There is a high change in test classification error where we increase the value of the learning rate after 0.0001 for both PC and BP.

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bias=False)
  (1): Dropout2d(p=0.5, inplace=False)
  (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (3): ReLU()
  (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (6): Dropout2d(p=0.5, inplace=False)
  (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (8): ReLU()
  (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (11): Dropout2d(p=0.5, inplace=False)
  (12): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (13): ReLU()
  (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (15): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (16): Dropout2d(p=0.5, inplace=False)
  (17): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (18): ReLU()
  (19): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (20): Flatten(start_dim=1, end_dim=-1)
  (21): Dropout(p=0.5, inplace=False)
  (22): Linear(in_features=2048, out_features=1024, bias=True)
  (23): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (24): ReLU()
  (25): Dropout(p=0.5, inplace=False)
  (26): Linear(in_features=1024, out_features=512, bias=True)
  (27): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (28): ReLU()
  (29): Linear(in_features=512, out_features=10, bias=True)
)
```

Figure 21: Back propagation model architecture used in Experiment 5.

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bias=False)
  (1): PCLayer()
  (2): Dropout2d(p=0.5, inplace=False)
  (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (4): ReLU()
  (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (7): PCLayer()
  (8): Dropout2d(p=0.5, inplace=False)
  (9): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (10): ReLU()
  (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (12): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (13): PCLayer()
  (14): Dropout2d(p=0.5, inplace=False)
  (15): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (16): ReLU()
  (17): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (18): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (19): PCLayer()
  (20): Dropout2d(p=0.5, inplace=False)
  (21): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (22): ReLU()
  (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (24): Flatten(start_dim=1, end_dim=-1)
  (25): Dropout(p=0.5, inplace=False)
  (26): Linear(in_features=2048, out_features=1024, bias=True)
  (27): PCLayer()
  (28): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (29): ReLU()
  (30): Dropout(p=0.5, inplace=False)
  (31): Linear(in_features=1024, out_features=512, bias=True)
  (32): PCLayer()
  (33): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (34): ReLU()
  (35): Linear(in_features=512, out_features=10, bias=True)
)
```

Figure 22: Prospective configuration model architecture used in Experiment 5.

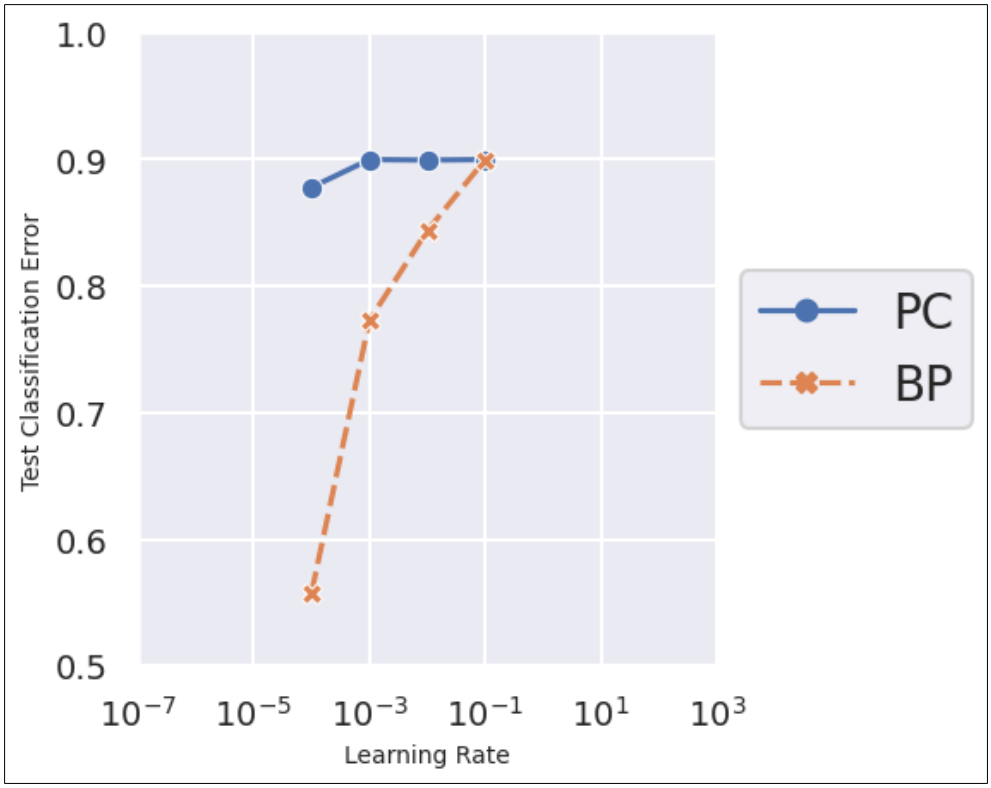


Figure 23: Learning rate vs. test classification error for Experiment 5.

- For Lr 0.1 and 0.01 the error follows a similar patter and the error is constent after some initial iterations. For Lr 0.001 the error settles in a Constance value for both PC and BP at 0.9. For BP ,PC the error keeps decreasing for each iteration For Lr value 0.0001.

6.6 Experiment 6: Deepening Architecture

6.6.1 Model Architecture

The model architecture used in this experiment involves deeper convolutional layers followed by fully connected layers for classification. Both back-propagation and prospective coding configurations are tested.

- **Back-Propagation Configuration**:
  - **Convolutional Layers**:
    - `Conv2d`: 3 input channels, 64 output channels, kernel size 5x5, stride 1, padding 2.
    - `Conv2d`: 64 input channels, 128 output channels, kernel size 3x3, stride 1, padding 1.
    - `Conv2d`: 128 input channels, 256 output channels, kernel size 3x3, stride 1, padding 1.
    - `Conv2d`: 256 input channels, 512 output channels, kernel size 3x3, stride 1, padding 1.

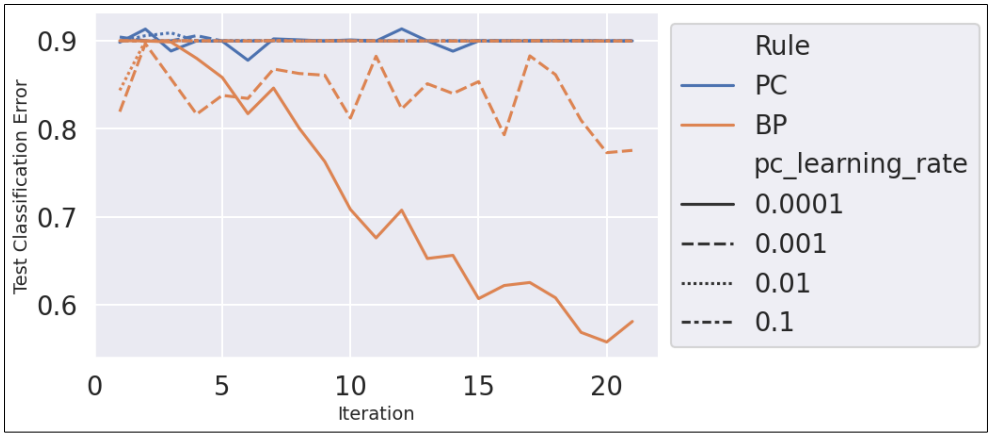
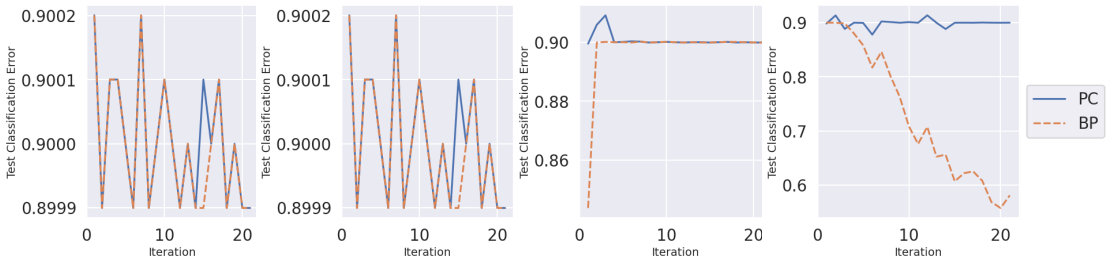


Figure 24: Iteration vs. test classification error for Experiment 5.



(a) Lr: 0.1 (b) Lr: 0.01 (c) Lre: 0.001 (d) Lr: 0.0001

Figure 25: Exp 5: Iteration vs. Test Classification Error

    * `Dropout2d`, `BatchNorm2d`, `ReLU`, `MaxPool2d` added in between each `Conv2d` layers.
  - **Fully Connected Layers**:
    * `Linear`: 2048 input features (from flattened convolutions), 1024 output features.
    * `Linear`: 1024 input features (from flattened convolutions), 512 output features.
    * `Linear`: 512 input features, 10 output classes (for CIFAR-10).

- **Prospective Coding Configuration**:
  Additinal to the above layers, here we added Predictive Coding Layers.
  - **Predictive Coding Layers**:
    * `PCLayer`: Placed between convolutional and fully connected layers to implement predictive coding mechanism.
    * `ReLU`: Activation function used after each PC layer.

6.6.2 Observations

- There is a high change in test classification error where we increase the value of the learning rate after 0.0001 for both PC and BP.
- For Lr 0.1 and 0.01 the error follows a similar patter and the error is constant after for all iterations. For Lr 0.001 the error settles in a Constance value for both PC and BP at 0.9. For BP ,PC the error keeps decreasing for each iteration For Lr value 0.0001.

6.7 Experiment 7: Deepening Architecture

6.7.1 Model Architecture

The model architecture used in this experiment involves deeper convolutional layers followed by fully connected layers for classification. Both back-propagation and prospective coding configurations are tested.

- **Back-Propagation Configuration**:
  - **Convolutional Layers**:
    * `Conv2d`: 3 input channels, 64 output channels, kernel size 5x5, stride 1, padding 2.
    * `Conv2d`: 64 input channels, 128 output channels, kernel size 3x3, stride 1, padding 1.
    * `Conv2d`: 128 input channels, 256 output channels, kernel size 3x3, stride 1, padding 1.
    * `Conv2d`: 256 input channels, 512 output channels, kernel size 3x3, stride 1, padding 1.
    * `Conv2d` 512 input channels, 1024 output channels, kernel size 3x3, stride 1, padding 1.
    * `ReLU`, `MaxPool2d` added after each `Conv2d` layers.

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bias=False)
  (1): Dropout2d(p=0.5, inplace=False)
  (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (3): ReLU()
  (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (6): Dropout2d(p=0.5, inplace=False)
  (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (8): ReLU()
  (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (11): Dropout2d(p=0.5, inplace=False)
  (12): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (13): ReLU()
  (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (15): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (16): Dropout2d(p=0.5, inplace=False)
  (17): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (18): ReLU()
  (19): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (20): Flatten(start_dim=1, end_dim=-1)
  (21): Dropout(p=0.5, inplace=False)
  (22): Linear(in_features=2048, out_features=1024, bias=True)
  (23): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (24): ReLU()
  (25): Dropout(p=0.5, inplace=False)
  (26): Linear(in_features=1024, out_features=512, bias=True)
  (27): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (28): ReLU()
  (29): Linear(in_features=512, out_features=10, bias=True)
)
```

Figure 26: Back propagation model architecture used in Experiment 6.

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bias=False)
  (1): Dropout2d(p=0.5, inplace=False)
  (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (3): PCLayer()
  (4): ReLU()
  (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (7): Dropout2d(p=0.5, inplace=False)
  (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (9): PCLayer()
  (10): ReLU()
  (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (12): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (13): Dropout2d(p=0.5, inplace=False)
  (14): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (15): PCLayer()
  (16): ReLU()
  (17): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (18): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (19): Dropout2d(p=0.5, inplace=False)
  (20): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (21): PCLayer()
  (22): ReLU()
  (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (24): Flatten(start_dim=1, end_dim=-1)
  (25): Dropout(p=0.5, inplace=False)
  (26): Linear(in_features=2048, out_features=1024, bias=True)
  (27): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (28): PCLayer()
  (29): ReLU()
  (30): Dropout(p=0.5, inplace=False)
  (31): Linear(in_features=1024, out_features=512, bias=True)
  (32): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (33): PCLayer()
  (34): ReLU()
  (35): Linear(in_features=512, out_features=10, bias=True)
)
```

Figure 27: Prospective configuration model architecture used in Experiment 6.

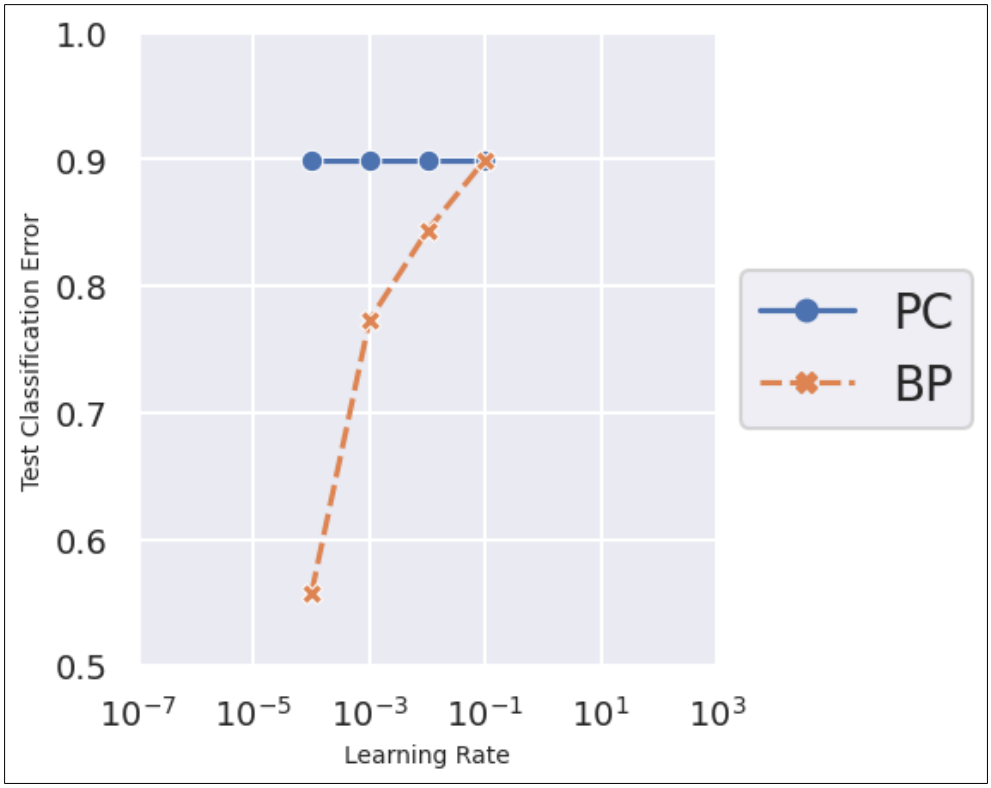


Figure 28: Learning rate vs. test classification error for Experiment 6.

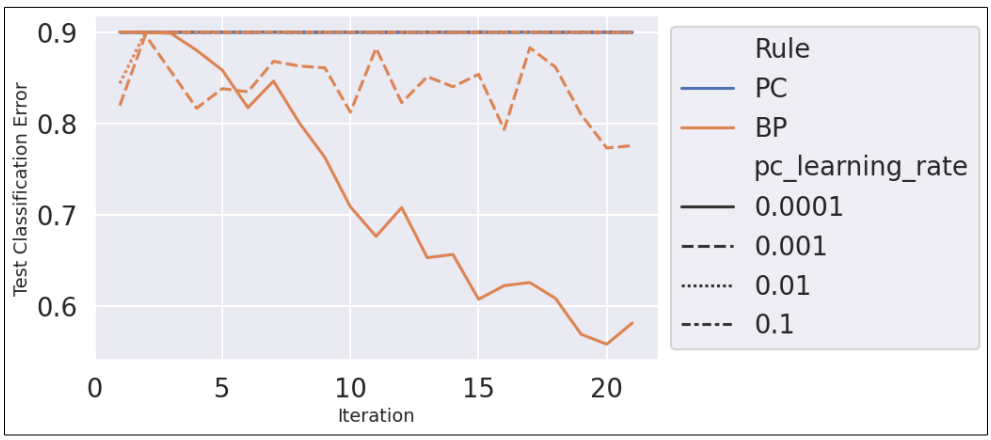
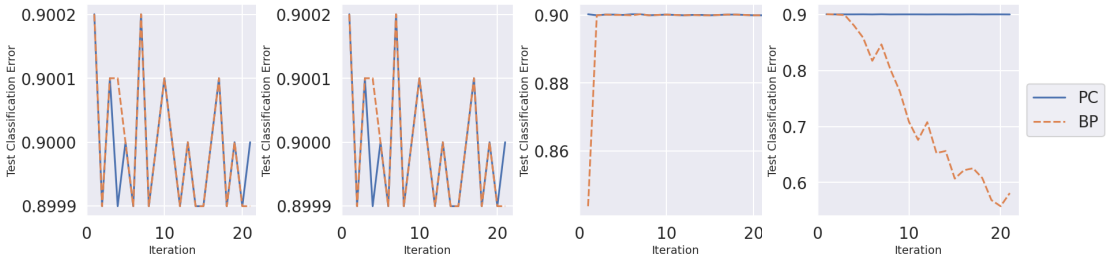


Figure 29: Iteration vs. test classification error for Experiment 6.



(a) Lr: 0.1 (b) Lr: 0.01 (c) Lre: 0.001 (d) Lr: 0.0001

Figure 30: Exp 6: Iteration vs. Test Classification Error

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bias=False)
  (1): ReLU()
  (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (4): ReLU()
  (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (6): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (7): ReLU()
  (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (9): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (10): ReLU()
  (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (12): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (13): ReLU()
  (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (15): Flatten(start_dim=1, end_dim=-1)
  (16): Linear(in_features=1024, out_features=512, bias=True)
  (17): ReLU()
  (18): Linear(in_features=512, out_features=10, bias=True)
)
```

Figure 31: Back propagation model architecture used in Experiment 7.

  - **Fully Connected Layers**:
    - `Linear`: 1024 input features (from flattened convolutions), 512 output features.
    - `Linear`: 512 input features, 10 output classes (for CIFAR-10).

- **Prospective Coding Configuration**:
  Additinal to the above layers, here we added Predictive Coding Layers.
  - **Predictive Coding Layers**:
    - `PCLayer`: Placed between convolutional and fully connected layers to implement predictive coding mechanism.

6.7.2 Observations

- There is not that much of of change in error value for PC but for BP the error increases in a higher rate and reaches the same value as the error for PC.

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bias=False)
  (1): PCLayer()
  (2): ReLU()
  (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (5): PCLayer()
  (6): ReLU()
  (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (8): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (9): PCLayer()
  (10): ReLU()
  (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (12): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (13): PCLayer()
  (14): ReLU()
  (15): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (16): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (17): PCLayer()
  (18): ReLU()
  (19): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (20): Flatten(start_dim=1, end_dim=-1)
  (21): Linear(in_features=1024, out_features=512, bias=True)
  (22): PCLayer()
  (23): ReLU()
  (24): Linear(in_features=512, out_features=10, bias=True)
)
```

Figure 32: Prospective configuration model architecture used in Experiment 7.

- For Lr 0.1 and 0.01 the error follows a similar patter and the error values keeps in the the range of 0.9002 - 0.8999. For Lr 0.001 the error settles in a Constance value for both PC and BP at 0.9000 after some initial iterations . for BP the error remains almost in 0.9 but for PC the error keeps decreasing for each iteration For Lr value 0.0001.

6.8 Experiment 3a: Deepening Architecture

6.8.1 Model Architecture

The model architecture used in this experiment involves deeper convolutional layers followed by fully connected layers for classification. Both back-propagation and prospective coding configurations are tested.

- **Back-Propagation Configuration**:
  - **Convolutional Layers**:
    - `Conv2d`: 3 input channels, 64 output channels, kernel size 3x3, stride 2, padding 1.
    - `Conv2d`: 64 input channels, 128 output channels, kernel size 3x3, stride 1, padding 1.
    - `Conv2d`: 128 input channels, 256 output channels, kernel size 3x3, stride 1, padding 1.
  - **Fully Connected Layers**:
    - `Linear`: 4096 input features (from flattened convolutions), 512 output features.

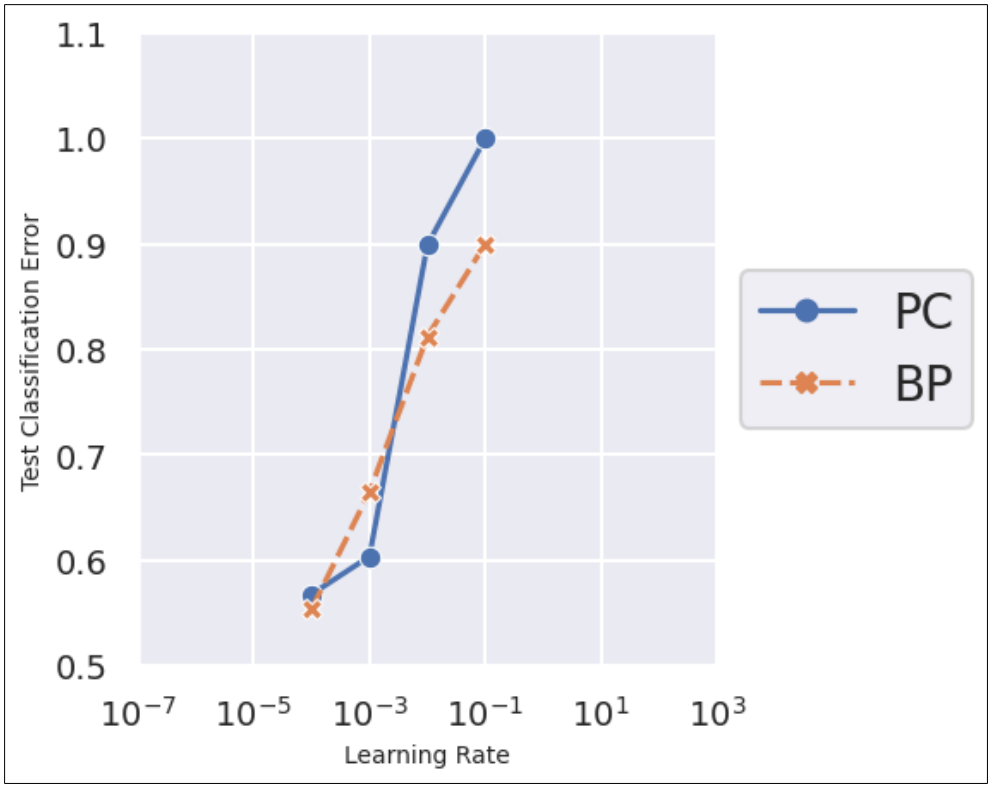


Figure 33: Learning rate vs. test classification error for Experiment 7.

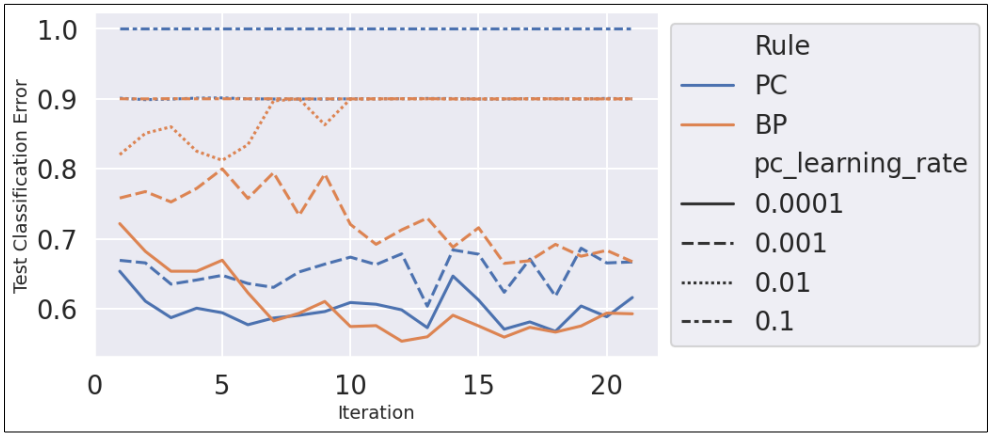
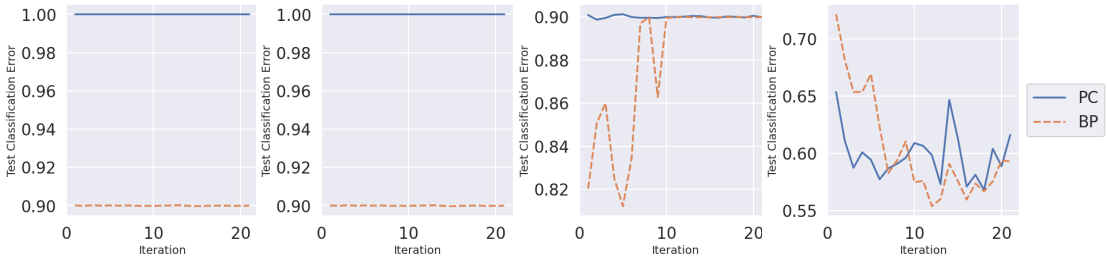


Figure 34: Iteration vs. test classification error for Experiment 7.



(a) Lr: 0.1 (b) Lr: 0.01 (c) Lre: 0.001 (d) Lr: 0.0001

Figure 35: Exp 7: Iteration vs. Test Classification Error

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bias=False)
  (1): ReLU()
  (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (4): ReLU()
  (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (6): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (7): ReLU()
  (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (9): Flatten(start_dim=1, end_dim=-1)
  (10): Linear(in_features=4096, out_features=512, bias=True)
  (11): ReLU()
  (12): Linear(in_features=512, out_features=10, bias=True)
)
```

Figure 36: Back propagation model architecture used in Experiment 3a.

    - `Linear`: 512 input features, 10 output classes (for CIFAR-10).

- **Prospective Coding Configuration**:
  Additinal to the above layers, here we added Predictive Coding Layers.
  - **Predictive Coding Layers**:
    - `PCLayer`: Placed between convolutional and fully connected layers to implement predictive coding mechanism.
    - `ReLU`: Activation function used after each PC layer.

6.8.2 Observations

- There is not that much of of change in error value for PC but for BP the error increases in a higher rate and reaches the same value as the error for PC.
- For Lr 0.1 and 0.01 the error follows a similar patter and the error values keeps in the the range of 0.9002 - 0.8999. For lr 0.001 the error settles in a Constance value for both PC and BP at 0.9000

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bias=False)
  (1): PCLayer()
  (2): ReLU()
  (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (5): PCLayer()
  (6): ReLU()
  (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (8): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (9): PCLayer()
  (10): ReLU()
  (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (12): Flatten(start_dim=1, end_dim=-1)
  (13): Linear(in_features=4096, out_features=512, bias=True)
  (14): PCLayer()
  (15): ReLU()
  (16): Linear(in_features=512, out_features=10, bias=True)
)
```

Figure 37: Prospective configuration model architecture used in Experiment 3a.

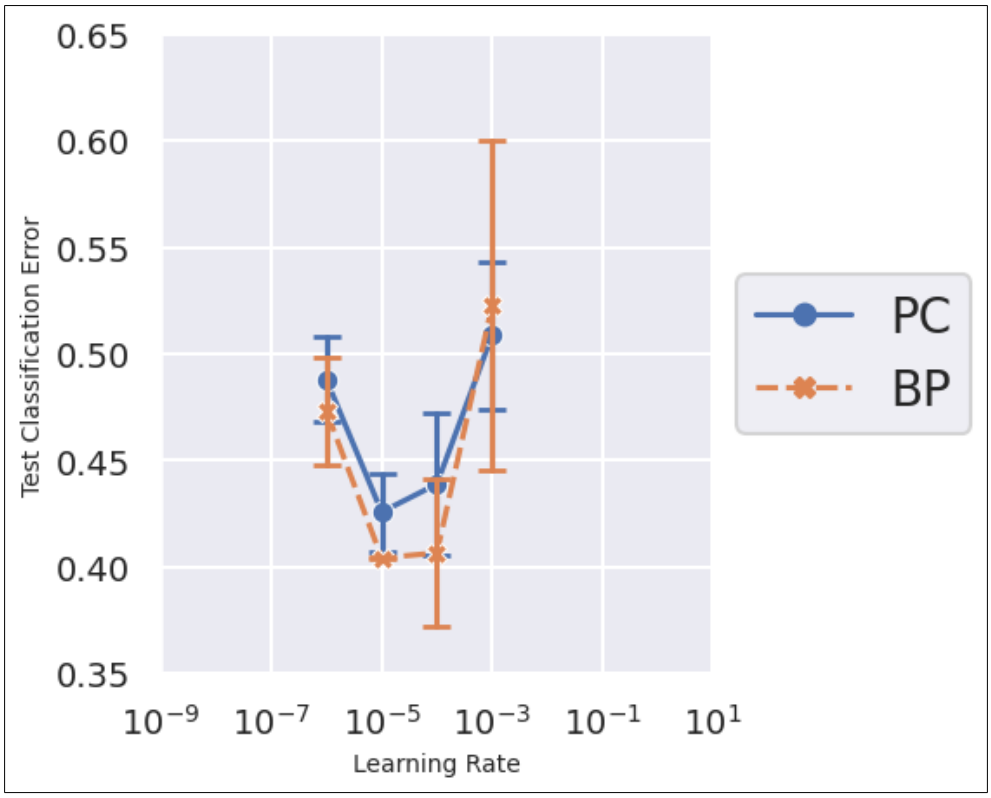


Figure 38: Learning rate vs. test classification error for Experiment 3a.

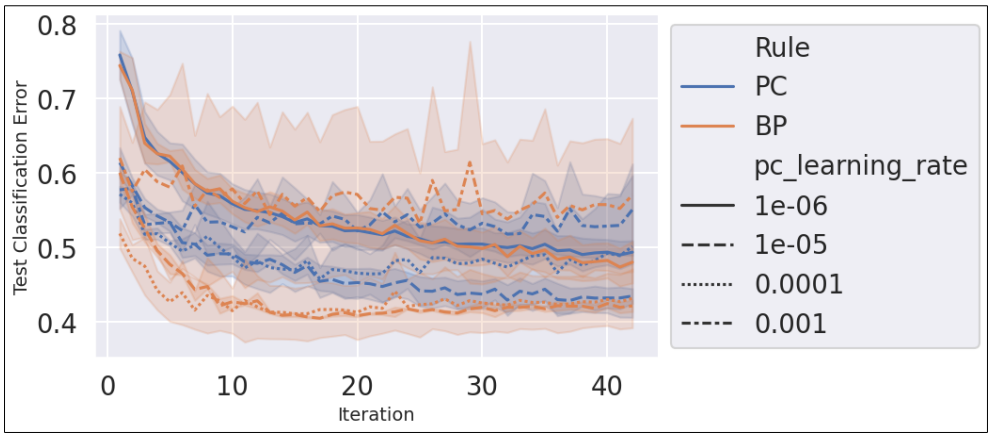
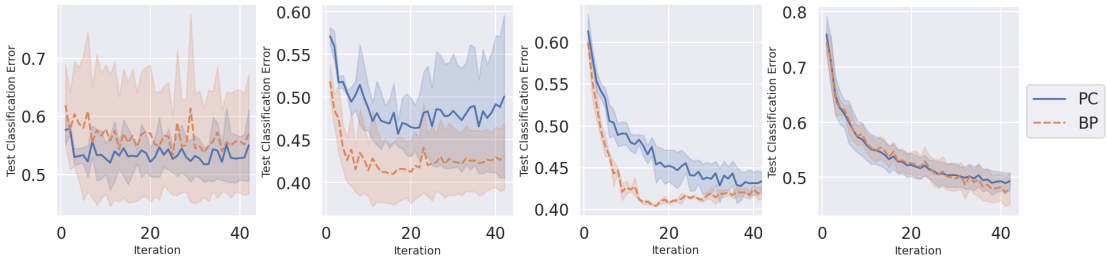
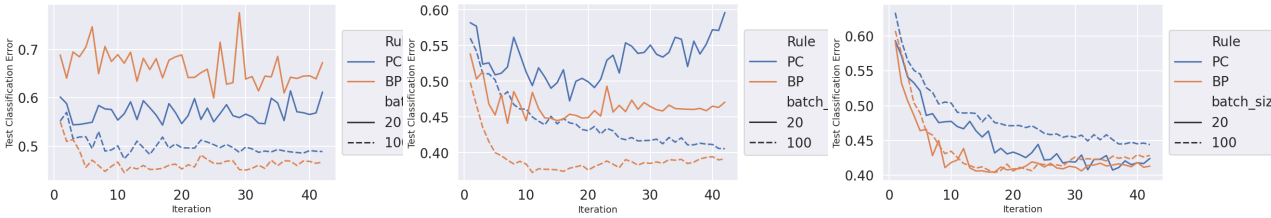


Figure 39: Iteration vs. test classification error for Experiment 3a.



(a) Lr: 0.1 (b) Lr: 0.01 (c) Lre: 0.001 (d) Lr: 0.0001

Figure 40: Exp 3a: Iteration vs. Test Classification Error



(a) Lr: 0.001 (b) Lr: 0.00001 (c) Lre: 0.000001

Figure 41: Exp 3a: Iteration vs. Test Classification Error

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bias=False)
  (1): ReLU()
  (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (4): ReLU()
  (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (6): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (7): ReLU()
  (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (9): Flatten(start_dim=1, end_dim=-1)
  (10): Linear(in_features=4096, out_features=512, bias=True)
  (11): ReLU()
  (12): Linear(in_features=512, out_features=10, bias=True)
)
```

Figure 42: Back propagation model architecture used in Experiment 3b.

6.9 Experiment 3b: Deepening Architecture

6.9.1 Model Architecture

The model architecture used in this experiment involves deeper convolutional layers followed by fully connected layers for classification. Both back-propagation and prospective coding configurations are tested.

- **Back-Propagation Configuration**:
  - **Convolutional Layers**:
    - `Conv2d`: 3 input channels, 64 output channels, kernel size 3x3, stride 2, padding 1.
    - `Conv2d`: 64 input channels, 128 output channels, kernel size 3x3, stride 1, padding 1.
    - `Conv2d`: 128 input channels, 256 output channels, kernel size 3x3, stride 1, padding 1.
  - **Fully Connected Layers**:
    - `Linear`: 4096 input features (from flattened convolutions), 512 output features.
    - `Linear`: 512 input features, 10 output classes (for CIFAR-10).
- **Prospective Coding Configuration**:
  Additinal to the above layers, here we added Predictive Coding Layers.
  - **Predictive Coding Layers**:
    - `PCLayer`: Placed between convolutional and fully connected layers to implement predictive coding mechanism.
    - `ReLU`: Activation function used after each PC layer.

6.9.2 Observations

- There is not that much of of change in error value for PC but for BP the error increases in a higher rate and reaches the same value as the error for PC.

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bias=False)
  (1): PCLayer()
  (2): ReLU()
  (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (5): PCLayer()
  (6): ReLU()
  (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (8): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (9): PCLayer()
  (10): ReLU()
  (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (12): Flatten(start_dim=1, end_dim=-1)
  (13): Linear(in_features=4096, out_features=512, bias=True)
  (14): PCLayer()
  (15): ReLU()
  (16): Linear(in_features=512, out_features=10, bias=True)
)
```

Figure 43: Prospective configuration model architecture used in Experiment 3b.

- For Lr 0.1 and 0.01 the error follows a similar patter and the error values keeps in the the range of 0.9002 - 0.8999. For Lr 0.001 the error settles in a Constance value for both PC and BP at 0.9. The error for PC almost floats around the value of 0.9 but for BP the error keeps decreasing for each iteration For Lr value 0.0001.

6.10 Experiment 3c: Deepening Architecture

6.10.1 Model Architecture

The model architecture used in this experiment involves deeper convolutional layers followed by fully connected layers for classification. Both back-propagation and prospective coding configurations are tested.

- **Back-Propagation Configuration**:
  - **Convolutional Layers**:
    - `Conv2d`: 3 input channels, 64 output channels, kernel size 5x5, stride 1, padding 2.
    - `Conv2d`: 64 input channels, 128 output channels, kernel size 3x3, stride 1, padding 1.
    - `Conv2d`: 128 input channels, 256 output channels, kernel size 3x3, stride 1, padding 1.
  - **Fully Connected Layers**:
    - `Linear`: 4096 input features (from flattened convolutions), 512 output features.
    - `Linear`: 512 input features, 10 output classes (for CIFAR-10).
- **Prospective Coding Configuration**:
  Additinal to the above layers, here we added Predictive Coding Layers.
  - **Predictive Coding Layers**:

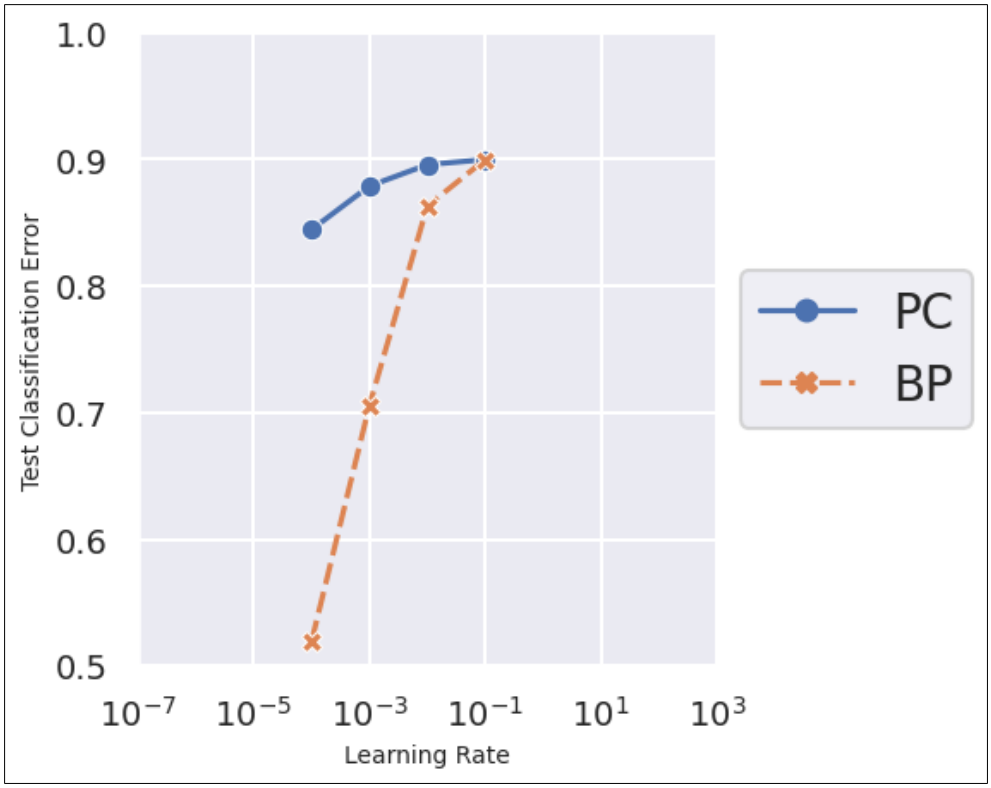


Figure 44: Learning rate vs. test classification error for Experiment 3b.

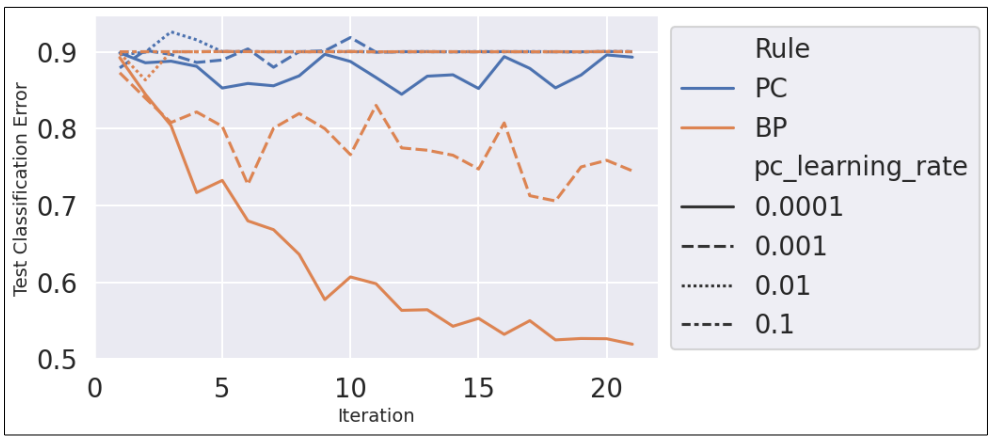
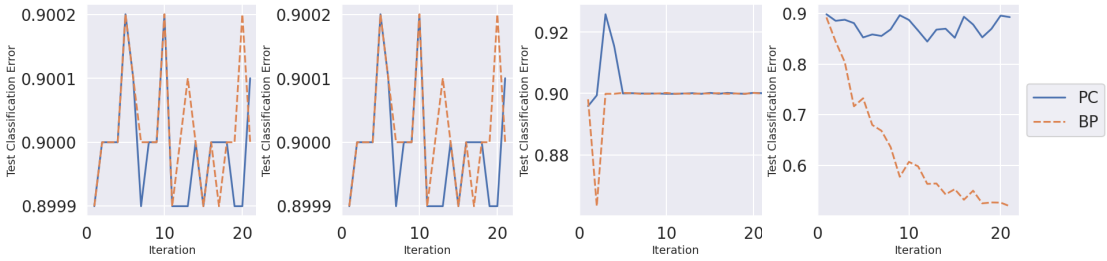


Figure 45: Iteration vs. test classification error for Experiment 3b.



(a) Lr: 0.1 (b) Lr: 0.01 (c) Lre: 0.001 (d) Lr: 0.0001

Figure 46: Exp 3b: Iteration vs. Test Classification Error

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bias=False)
  (1): Dropout2d(p=0.5, inplace=False)
  (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (3): ReLU()
  (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (6): Dropout2d(p=0.5, inplace=False)
  (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (8): ReLU()
  (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (11): Dropout2d(p=0.5, inplace=False)
  (12): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (13): ReLU()
  (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (15): Flatten(start_dim=1, end_dim=-1)
  (16): Dropout(p=0.5, inplace=False)
  (17): Linear(in_features=4096, out_features=512, bias=True)
  (18): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (19): ReLU()
  (20): Linear(in_features=512, out_features=10, bias=True)
)
```

Figure 47: Back propagation model architecture used in Experiment 3c.

* `PCLayer`: Placed between convolutional and fully connected layers to implement predictive coding mechanism.
* `ReLU`: Activation function used after each PC layer.

6.10.2 Observations

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bias=False)
  (1): Dropout2d(p=0.5, inplace=False)
  (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (3): PCLayer()
  (4): ReLU()
  (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (7): Dropout2d(p=0.5, inplace=False)
  (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (9): PCLayer()
  (10): ReLU()
  (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (12): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (13): Dropout2d(p=0.5, inplace=False)
  (14): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (15): PCLayer()
  (16): ReLU()
  (17): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (18): Flatten(start_dim=1, end_dim=-1)
  (19): Dropout(p=0.5, inplace=False)
  (20): Linear(in_features=4096, out_features=512, bias=True)
  (21): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (22): PCLayer()
  (23): ReLU()
  (24): Linear(in_features=512, out_features=10, bias=True)
)
```

Figure 48: Prospective configuration model architecture used in Experiment 3c.

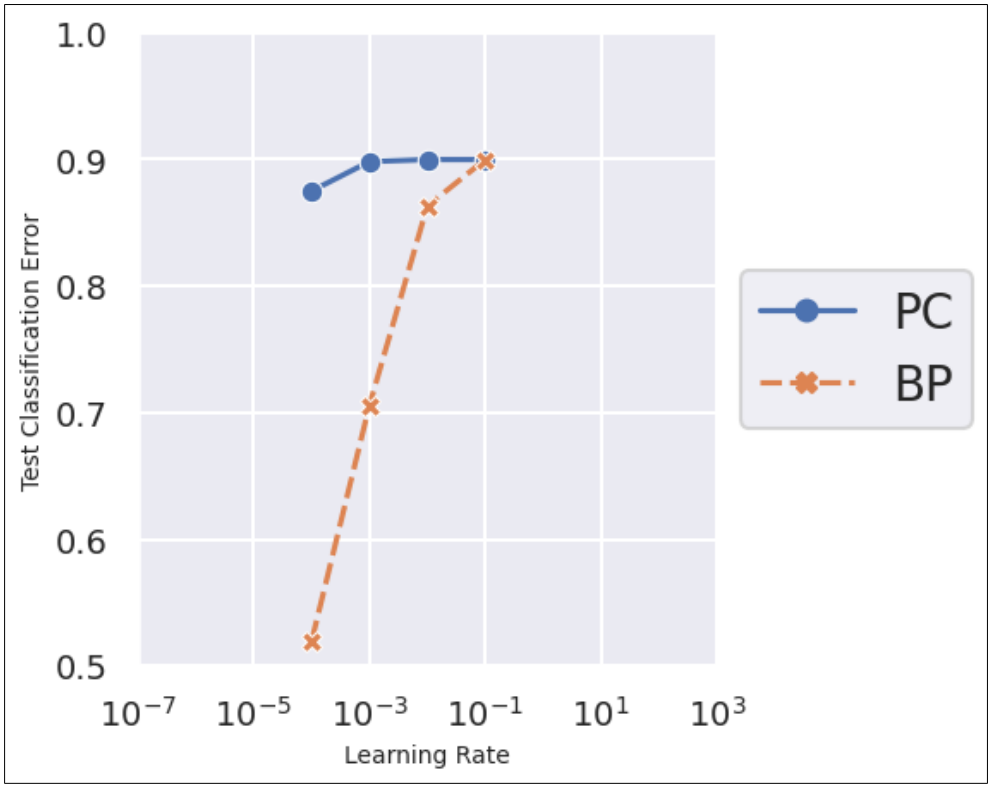


Figure 49: Learning rate vs. test classification error for Experiment 3c.

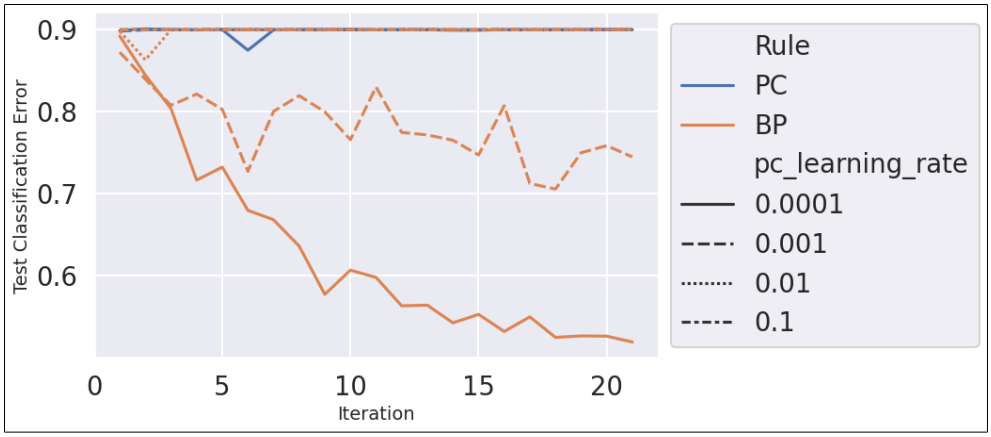
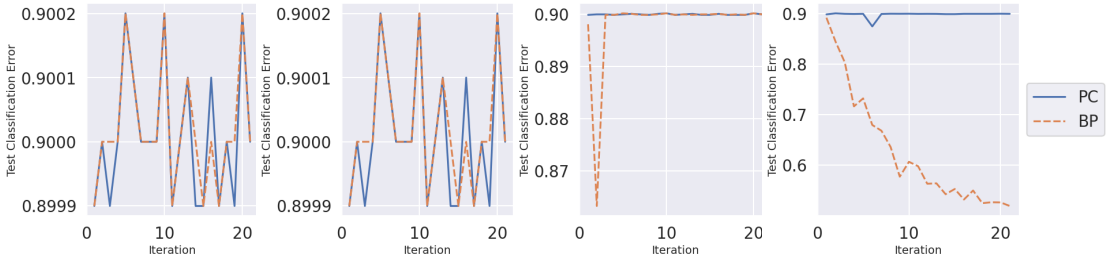


Figure 50: Iteration vs. test classification error for Experiment 3c.



(a) Lr: 0.1 (b) Lr: 0.01 (c) Lre: 0.001 (d) Lr: 0.0001

Figure 51: Exp 3c: Iteration vs. Test Classification Error

7 Performance of Transfer Learning

7.1 Best Experiment

We explored neural networks with up to 5 convolutional layers. From our initial experiments, Experiment 3 yielded the best results. Building on this, we conducted further experiments by introducing dropout and batch normalization to Experiment 3, with various placements of the prospective coding (PC) layer. These additional experiments were labeled as Experiment 3b and Experiment 3c. Unfortunately, both Experiment 3b and Experiment 3c performed worse than the original Experiment 3. Consequently, we designed Experiment 3a, which incorporates the conditions specified in Section 5.2.2.

In Experiment 3a, we determined that a learning rate of **0.0001**, a batch size of **100**, and iterations up to **42** produced optimal performance. The architecture of Experiment 3a is consistent with the architecture described in Experiment 3.(Figure 11)

7.2 Transfer Learning Experiment

Using the optimal model from Experiment 3a, we applied transfer learning to the COVID-19 lung scans dataset. This involved fine-tuning the pre-trained model on the new dataset to adapt it to the specific characteristics and features of lung scan images associated with COVID-19. The results of this transfer learning experiment are presented in the following section.

7.3 Results and Observation

We trained the same experimental architecture using both backpropagation and prospective configuration on CIFAR-10 dataset. Afterwards, we fine-tuned each using their respective trained models with COVID-CT Dataset:

- For backpropagation fine-tuning, we used a model trained with backpropagation.
- For prospective configuration fine-tuning, we used a model trained with prospective configuration.

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bias=False)
  (1): ReLU()
  (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (4): ReLU()
  (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (6): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (7): ReLU()
  (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (9): Flatten(start_dim=1, end_dim=-1)
  (10): Linear(in_features=4096, out_features=512, bias=True)
  (11): ReLU()
  (12): Linear(in_features=512, out_features=2, bias=True)
)
```

Figure 52: Model Architecture of the Back propagation transfer learning

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2), bias=False)
  (1): PCLayer()
  (2): ReLU()
  (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (5): PCLayer()
  (6): ReLU()
  (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (8): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (9): PCLayer()
  (10): ReLU()
  (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (12): Flatten(start_dim=1, end_dim=-1)
  (13): Linear(in_features=4096, out_features=512, bias=True)
  (14): PCLayer()
  (15): ReLU()
  (16): Linear(in_features=512, out_features=2, bias=True)
)
```

Figure 53: Model Architecture of the Prospective configuration transfer learning

Trial SupervisedLearningTrainable_89ad9_00000 result	
time_this_iter_s	6.06058
time_total_s	362.279
training_iteration	42
is_num_iterations_reached	1
test__classification_error	0.04545

Figure 54: Results of the Back propagation transfer learning

Trial SupervisedLearningTrainable_b87ab_00000 result	
time_this_iter_s	14.8897
time_total_s	626.536
training_iteration	42
is_num_iterations_reached	1
test__classification_error	0.28788

Figure 55: Results of the Prospective configuration transfer learning

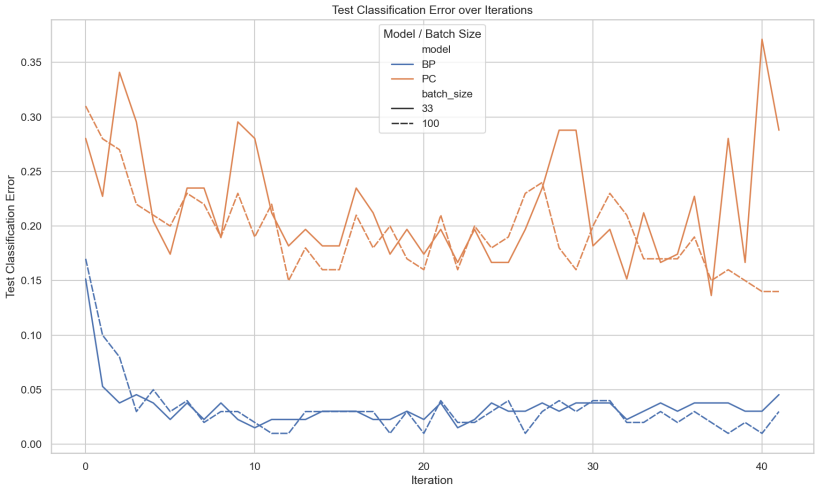
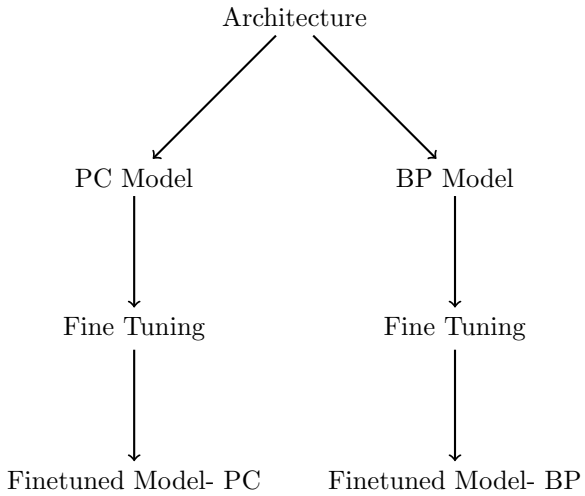


Figure 56: Results for transfer learning



When comparing backpropagation and prospective configuration, both methods yield good results. However, backpropagation consistently shows superior performance over prospective configuration.

During transfer learning using prospective configuration, we encountered issues when loading model weights in PyTorch. Specifically, the prospective configuration (PC) layer values were not correctly loaded, and we initialized them from their default values instead. This initialization process likely contributed to reduced performance of the PC layer.

8 Reproducing Our Results

We began by forking the repository of prospective configuration and recreating one of their results on our local machine. Our cloned repository can be found here. Subsequently, we initiated new experiments in the `aai-experiments` folder within

our repository, visible here.

To run the experiments, use the provided Jupyter notebook (`ipynb`) here and modify the following variables: such as exp-name, results-path and data-path

Addition to this, you can view all our experiments results and the visualizations here

```
import os
from pathlib import Path

exp_name = "08-pt-final"
results_path = "/content/drive/MyDrive/results"
data_path = "/content/prospective-configuration/data"

os.environ['RESULTS_DIR'] = results_path
os.environ['DATA_DIR'] = data_path

os.environ['EXP_NAME'] = exp_name
os.environ['MODEL_DIR'] = f"{results_path}/{exp_name}/model"
os.environ['ANALYSIS_DIR'] = f"{results_path}/{exp_name}/analysis"

os.environ['PT_MODEL_DIR'] = f"{results_path}/08-pt-final/model"
```

By executing these steps, you will obtain both the results and visualizations.

9 Conclusion

Our experiments with prospective coding as an alternative to back-propagation have demonstrated promising results across various learning scenarios. We observed that prospective configuration not only offers competitive performance in standard tasks like image classification on CIFAR-10 but also excels in challenging domains such as transfer learning for Covid19 detection using lung scans. By focusing on local computations and predictive error adjustments, prospective coding aligns more closely with biological principles of learning, presenting a compelling case for its adoption in neural network optimization..

10 Future Work

Future research on prospective coding could focus on scaling the algorithm to larger and deeper neural network architectures to assess its performance across more complex tasks. Exploring adaptive learning rate strategies specifically tailored for prospective coding could further enhance its convergence speed and stability across diverse datasets. Additionally, investigating applications in reinforcement learning and unsupervised learning settings could uncover new strengths and challenges, broadening the algorithm's applicability in machine learning. Integrating insights from neuroscience to refine biological plausibility and efficiency in learning processes remains a critical direction for advancing prospective coding as a viable alternative to traditional back-propagation.

11 References

References

[1] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto, 2009. Accessed: 2024-06-30.

[2] Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, 2020.

[3] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. *Nature*, 323(6088):533–536, 1986.

[4] Yuhang Song, Beren Millidge, Tommaso Salvatori, Thomas Lukasiewicz, Zhenghua Xu, and Rafal Bogacz. Inferring neural activity before plasticity as a foundation for learning beyond backpropagation. *Nature Neuroscience*, pages 1–11, 2024.

[5] Xingyi Yang, Xuehai He, Jinyu Zhao, Yichen Zhang, Shanghang Zhang, and Pengtao Xie. COVID-CT-Dataset: A CT Image Dataset about COVID-19. `https://arxiv.org/pdf/2003.13865`, 2020. Accessed: 2024-06-30.