# 19CSE456 Neural Network and Deep Learning Laboratory

# List of Experiments

| Week # | Experiment Title |
|---|---|
| 1 | Introduction to the lab and Implementation of a simple Perceptron (Hardcoding) |
| 2 | Implementation of Perceptron for Logic Gates (Hardcoding, Sklearn, TF) |
| 3 | Implementation of Multilayer Perceptron for XOR Gate and other classification problems with ML toy datasets (Hardcoding & TF) |
| 4 | Implementation of MLP for Image Classification with MNIST dataset (Hardcoding & TF) |
| 5 | Activation Functions, Loss Functions, Optimizers (Hardcoding & TF) |
| 6 | Lab Evaluation 1 (based on topics covered from w1 to w5) |
| 7 | Convolution Neural Networks for Toy Datasets (MNIST & CIFAR) |
| 8 | Convolution Neural Networks for Image Classification (Oxford Pets, Tiny ImageNet, etc.) |
| 9 | Recurrent Neural Networks for Sentiment Analysis with IMDB Movie Reviews |
| 10 | Long Short Term Memory for Stock Prices (Yahoo Finance API) |

# List of Experiments contd.

| Week # | Experiment Title |
|---|---|
| 11 | Implementation of Autoencoders and Denoising Autoencoders (MNIST/CIFAR) |
| 12 | Boltzmann Machines (MNIST/CIFAR) |
| 13 | Restricted Boltzmann Machines (MNIST/CIFAR) |
| 14 | Hopfield Neural Networks (MNIST/CIFAR) |
| 15 | Lab Evaluation 2 (based on CNN, RNN, LSTM, and AEs) |
| 16 | Case Study Review (Phase 1) |
| 17 | Case Study Review (Phase 1) |

# Optimization Techniques in TensorFlow

TensorFlow offers various gradient-based optimizers to minimize loss functions efficiently:

### Gradient-Based Optimizers

1. Stochastic Gradient Descent
   *tf.keras.optimizers.SGD*()

2. Adam
   *tf.keras.optimizers.Adam()*

3. RMSprop
   *tf.keras.optimizers.RMSprop()*

4. Adagrad
   *tf.keras.optimizers.Adagrad()*

5. Adadelta
   *tf.keras.optimizers.Adadelta()*

### Learning Rate Schedulers

1. Exponential Decay
   *tf.keras.optimizers.schedules.ExponentialDecay()*

2. Piecewise Constant Decay
   *tf.keras.optimizers.schedules.PiecewiseConstantDecay()*

3. Cosine Decay *tf.keras.optimizers.schedules.CosineDecay()*

4. Inverse Time Decay
   *tf.keras.optimizers.schedules.InverseTimeDecay()*

5. Polynomial Decay
   *tf.keras.optimizers.schedules.PolynomialDecay()*

# Gradient-Based Optimizers

```
optimizers = {
"Batch Gradient Descent (BGD)": tf.keras.optimizers.SGD(learning_rate=0.01),  # Full batch size
"Mini-Batch Gradient Descent (MBGD)": tf.keras.optimizers.SGD(learning_rate=0.01),  # Mini-batch size (e.g., 32)
"SGD": tf.keras.optimizers.SGD(learning_rate=0.01),  # Stochastic GD (batch_size = 1)
"SGD with Momentum": tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9),
"NAG (Nesterov)": tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9, nesterov=True),
"Adam": tf.keras.optimizers.Adam(learning_rate=0.001),
"RMSprop": tf.keras.optimizers.RMSprop(learning_rate=0.001),
"Adagrad": tf.keras.optimizers.Adagrad(learning_rate=0.01),
"Adadelta": tf.keras.optimizers.Adadelta(learning_rate=1.0),
"Nadam": tf.keras.optimizers.Nadam(learning_rate=0.001),}
```

# Gradient-Based Optimizers

```
# Model Creation
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(input_shape,)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(2, activation='softmax')
  ])
```

```
# Compile model
  model.compile(
    optimizer=optimizer,
    loss='categorical_crossentropy',
    metrics=['accuracy']
  )
```

```
# Train model
  model.fit(
    X_train, y_train,
    epochs=10,
    batch_size=batch_size,
    verbose=0,
    callbacks=[custom_callback]
  )
```

# Learning Rate Schedulers

## Step Decay

```
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=0.01, decay_steps=10000, decay_rate=0.9, staircase=True )
optimizer = tf.keras.optimizers.SGD(learning_rate=lr_schedule)
```

The learning rate is reduced at specific intervals or "steps" during training

## Exponential Decay

```
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=0.01, decay_steps=1000, decay_rate=0.96, staircase=False )
optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule)
```

To reduce the learning rate of a model during training gradually

## Piecewise Decay

```
lr_schedule = tf.keras.optimizers.schedules.PiecewiseConstantDecay(
    boundaries=[5000, 10000], values=[0.01, 0.005, 0.001] )
optimizer = tf.keras.optimizers.SGD(learning_rate=lr_schedule)
```

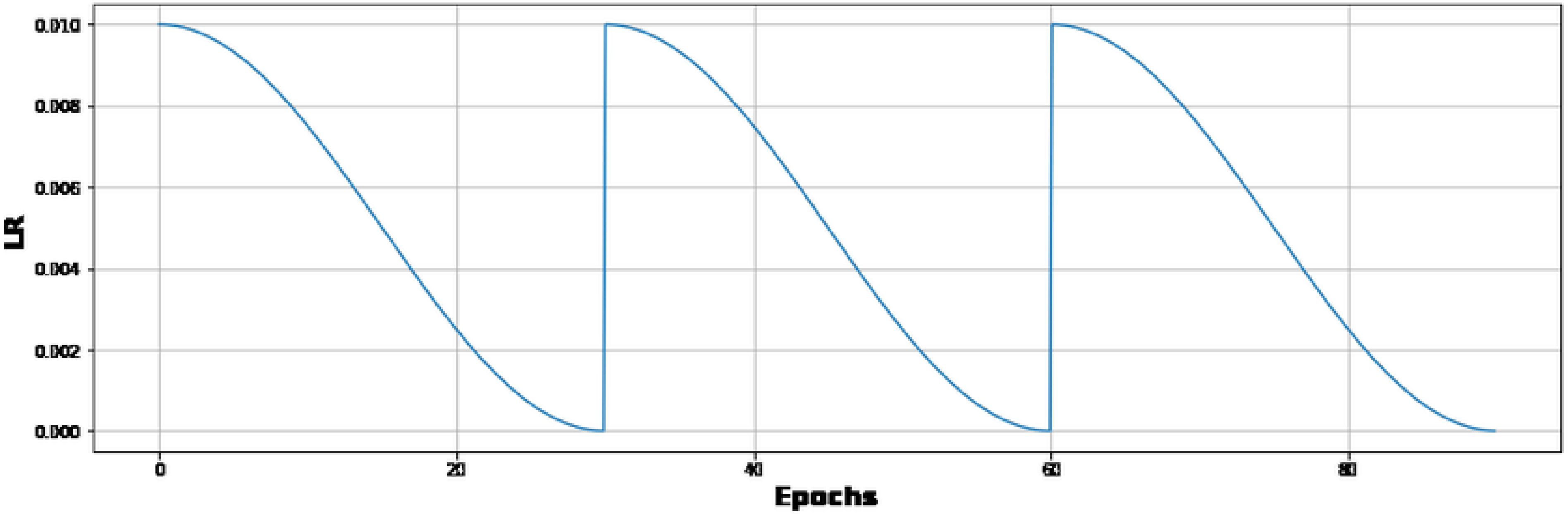
The learning rate remains constant within specified intervals (or "pieces") of training steps, but changes to a different value at the boundaries between these intervals

# Learning Rate Schedulers

## Cosine Decay

```
lr_schedule = tf.keras.optimizers.schedules.CosineDecay(
    initial_learning_rate=0.01, decay_steps=10000, alpha=0.1 )
optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule)
```

Reduces the learning rate following a cosine function



# Gradient-Based Optimizers

```
# Model Creation
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(input_shape,)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(2, activation='softmax')
  ])
```

```
# Compile model
  model.compile(
    optimizer=optimizer,
    loss='categorical_crossentropy',
    metrics=['accuracy']
  )
```

```
# Train model
  model.fit(
    X_train, y_train,
    epochs=10,
    batch_size=batch_size,
    verbose=0,
    callbacks=[custom_callback]
  )
```

# Week 6 Exercises

## 1. Optimizers & Learning Rate Schedule in MLP Classifier

Objective: Implement and compare various optimizers and learning rate scheduling schemes in TensorFlow using the Titanic dataset.

Tasks:

- Load and preprocess the Titanic dataset.
- Implement a MLP classifier in TensorFlow.
  - Train the model using the following optimizers:
  - SGD (Stochastic )
  - Momentum
  - Adam
  - RMSprop
- Apply the following learning rate scheduling schemes:
  - Step Decay
  - Exponential Decay
  - Piecewise Constant Decay
  - Cosine Decay
- Compare the performance of the different optimizers and learning rate schedules by evaluating metrics such as accuracy and loss.