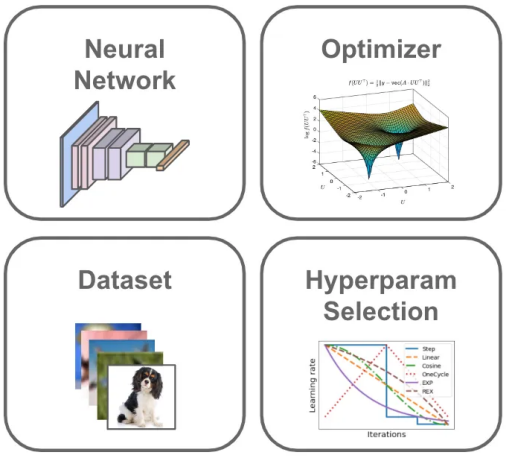


Learning Rate Schedule

Dr. Sabzi

Advisor

Leili Motahari

BSc Student

2024

Contents

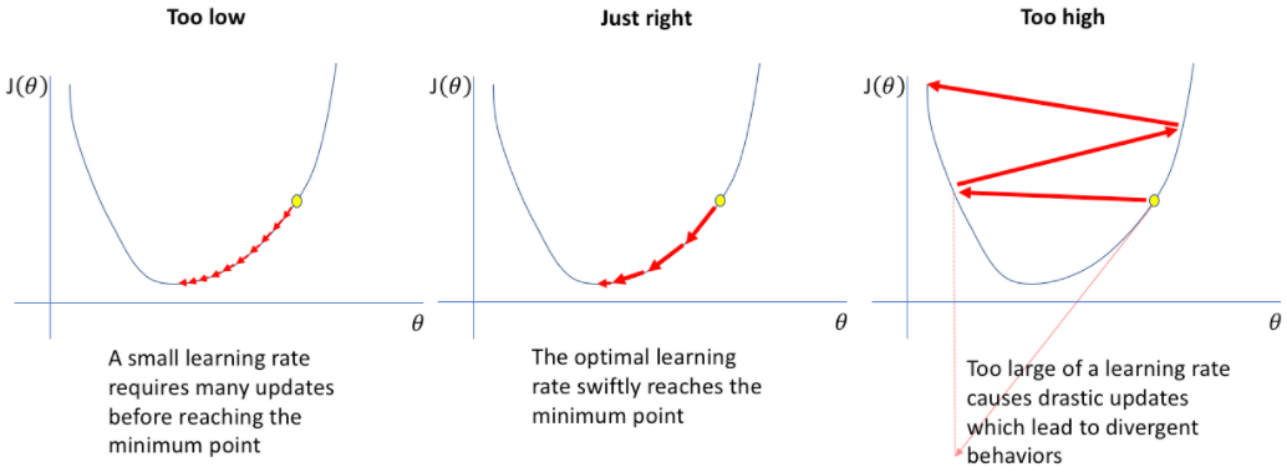
Chapter 1

Introduction

1.1 What is the learning rate in neural networks?

What is the learning rate, and what does it do to a neural network? The learning rate (or step-size) is explained as the magnitude of change/update to model weights during the backpropagation training process. As a configurable hyperparameter, the learning rate is usually specified as a positive value less than 1.0.

In back-propagation, model weights are updated to reduce the error estimates of our loss function. Rather than changing the weights using the full amount, we multiply it by some learning rate value. For example, setting the learning rate to 0.5 would mean updating (usually subtract) the weights with 0.5*estimated weight errors (i.e., gradients or total error change w.r.t. the weights).
The learning rate controls how big of a step for an optimizer to reach the minima of the loss function. What does this do to our optimization algorithm? Look at these graphs:



- With a large learning rate (on the right), the algorithm learns fast, but it may also cause the algorithm to oscillate around or even jump over the minima. Even worse, a high learning rate equals large weight updates, which might cause the weights to overflow;
- On the contrary, with a small learning rate (on the left), updates to the weights are small, which will guide the optimizer gradually towards the minima. However, the optimizer may take too long to converge or get stuck in a plateau or undesirable local minima;
- A good learning rate is a tradeoff between the coverage rate and overshooting (in the middle). It's not too small so that our algorithm can converge swiftly, and it's not too large so that our algorithm won't jump back and forth without reaching the minima.

Although the theoretical principle of finding an appropriate learning rate is straightforward (not too large, not too small), it's easier said than done! To solve this problem, the learning rate schedule is introduced[1].

1.2 Importance of Learning Rate

In machine learning, models are governed by two types of parameters: machine-learnable parameters and hyper-parameters. Machine-learnable parameters are estimated by the algorithm during training, while hyper-

parameters, such as the learning rate (denoted by), are set by the data scientist or engineer to influence the training process. The learning rate plays a crucial role in controlling how quickly the algorithm updates the model's weights to minimize the error between predicted and actual outputs.

The choice of learning rate significantly impacts both the speed of convergence and the model's final performance. If the learning rate is too high, the model may overshoot optimal values, leading to poor performance and unstable training. On the other hand, a learning rate that is too low can cause the training process to be slow or get stuck in suboptimal solutions (local minima).

Moreover, the learning rate is critical in balancing underfitting and overfitting. It affects how well a model generalizes to new data beyond the training set. Finding the right learning rate often requires careful tuning based on the model architecture and training scenario, as there is no universally optimal value[**?**, **?**, 2].

1.2.1 Example :

Imagine you are learning to play a video game that involves jumping over obstacles. If you always jump too early or too late you will keep failing and have to restart the game. But if you try to adjust your timing by a small amount each time we can eventually find the sweet spot where you can consistently get over the obstacles.

Similarily in machine learning, low learning rate will result in longer training times and increased costs. On the other hand high learning rate can cause the model to overshoot or fail to converge. Therefore, finding the right learning rate is crucial to achieving best results without wasting time and resources.

Discovering the ideal learning rate for a particular problem can be a challenging task, but there are several well-established techniques available that can help. Adaptive learning rate techniques involve dynamically adjusting the learning rate instead of using a constant value[3].

1.3 Impact of learning rate on model performance

The selection of the learning rate can significantly affect a model's performance. A learning rate that's too high might lead to rapid convergence, but the model's results may be suboptimal. Conversely, a low learning rate can cause the training to be slow and yield poor-quality solutions. Therefore, choosing a learning rate that balances the speed of convergence with solution quality is critical [2].

If the learning rate is too high, several issues may arise:

- **Oscillations:** The model's weights may update too quickly, causing it to fluctuate around the optimal solution.
- **Divergence:** The model may move away from the ideal outcome, never reaching convergence.
- **Poor performance:** The model might settle on a suboptimal solution, leading to reduced accuracy or effectiveness.

On the other hand, if the learning rate is too low:

- **Slow convergence:** The optimization process could take too long to find a solution, resulting in sluggish training.
- **Poor performance:** The model might get stuck in a local minimum, failing to achieve optimal performance.

Chapter 2

Learning Rate Strategies

In neural network training, the choice of learning rate strategy is crucial for optimizing model performance. Each approach offers distinct benefits and is suited to different training scenarios.

2.1 Fixed Learning Rate

he fixed learning rate is one of the simplest strategies, where the learning rate remains constant throughout the training process. This approach offers simplicity and stability but may not be optimal in all situations.

- **Pros:**
  1. **Simplicity:** Easy to implement and understand.
  2. **Stability:** Provides a consistent update rate for the model's weights.
- **Cons :**
  1. **Lack of Adaptability:** Fixed rates do not adjust to changing training dynamics. If the learning rate is too high, the model might overshoot optimal solutions, leading to instability or divergence. If too low, the training process can be sluggish, and the model may become stuck in local minima.

2.1.1 What are use cases of fixed learning rate

- Ideal for simpler or baseline models where a constant rate is sufficient.
- Example: Setting the learning rate to a constant value such as 0.01.

2.1.2 Challenges and Enhancements

Choosing an appropriate fixed learning rate involves balancing convergence speed with solution quality. A rate set too high can cause the model to overshoot the ideal outcome, while a rate set too low may result in slow convergence and poor performance.

To address some limitations of fixed learning rates, a "Loss Decay" method can be employed. This technique dynamically adjusts the loss function during training to improve convergence, without altering the learning rate itself. The method focuses on controlling gradient sizes through loss adjustments, thereby mitigating common issues associated with fixed learning rates.

2.1.3 Comparison with Adaptive Learning Rates

While adaptive learning rate methods, which adjust rates dynamically during training, can accelerate convergence, a fixed learning rate still plays a critical role. It requires careful manual tuning to achieve good performance. Although adaptive methods offer flexibility, combining a fixed learning rate with techniques like loss decay can also yield effective results in various scenarios [4, 5, 6, 2, 3].

2.2 Adaptive learning rates

Adaptive learning rates are techniques that dynamically adjust the learning rate during training to optimize model performance. By monitoring the cost function's gradient, these methods can speed up or slow down learning based on the steepness of the cost function curve.

2.2.1 Key Algorithms:

1. **AdaGrad**: This algorithm tailors the learning rate for each parameter based on historical gradient information, reducing the learning rate for frequently updated parameters. It performs well with sparse data but can stop learning too early due to its aggressive decay.
2. **RMSprop**A modification of AdaGrad, RMSprop addresses the issue of overly aggressive learning rate decay by maintaining a moving average of squared gradients, allowing for a more balanced adjustment of the learning rate.
3. **Adam** Combining the strengths of both AdaGrad and RMSprop, Adam incorporates momentum and adjusts the learning rate based on an exponentially decaying average of past gradients. This results in faster convergence and improved performance, especially in high-dimensional spaces and recurrent networks.

2.2.2 Pros and Cons

- **Pros:** These algorithms offer parameter-specific adaptability, enhancing convergence speed and solution quality.
- **Cons:** They can be more computationally expensive and may involve varying complexity, with potential issues of diminishing learning rates.

2.2.3 Challenges and Enhancements

The challenge of using learning rate schedules is that their hyperparameters have to be defined in advance and they depend heavily on the type of model and problem. Another problem is that the same learning rate is applied to all parameter updates. If we have sparse data, we may want to update the parameters in different extent instead.

Adaptive gradient descent algorithms such as Adagrad, Adadelta, RMSprop, Adam, provide an alternative to classical SGD. These per-parameter learning rate methods provide heuristic approach without requiring expensive work in tuning hyperparameters for the learning rate schedule manually.

In brief, Adagrad performs larger updates for more sparse parameters and smaller updates for less sparse parameter. It has good performance with sparse data and training large-scale neural network. However, its monotonic learning rate usually proves too aggressive and stops learning too early when training deep neural networks. Adadelta is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate. RMSprop adjusts the Adagrad method in a very simple way in an attempt to reduce its aggressive, monotonically decreasing learning rate. Adam is an update to the RMSProp optimizer which is like RMSprop with momentum.

2.2.4 Comparison with Adaptive Learning Rates

Compared to fixed learning rate approaches, adaptive learning rate methods can be more computationally expensive while simultaneously enhancing convergence speed and solution quality [4, 6, 2, 3].

2.3 Learning rate decay

Learning rate decay is a technique used in machine learning models, especially deep neural networks. It is sometimes referred to as learning rate scheduling or learning rate annealing. Throughout the training phase, it entails gradually lowering the learning rate. Learning rate decay is used to gradually adjust the learning rate, usually by lowering it, to facilitate the optimization algorithm's more rapid convergence to a better solution. This method tackles problems that are frequently linked to a fixed learning rate, such as oscillations and sluggish convergence [7].

Learning rate decay can be accomplished by a variety of techniques, such as step decay, exponential decay, and 1/t decay. Degradation strategy selection is based on the particular challenge and architecture. When training deep learning models, learning rate decay is a crucial hyperparameter that, when used properly, can result in faster training, better convergence, and increased model performance.

2.3.1 How Learning Rate Decay Works

Learning rate decay is like driving a car towards a parking spot. At first, you drive fast to reach the spot quickly. As you get closer, you slow down to park accurately. In machine learning, the learning rate determines how much the model changes based on the mistakes it makes. If it's too high, the model might miss the best fit; too low, and it's too slow. Learning rate decay starts with a higher learning rate, letting the model learn fast. As training progresses, the rate gradually decreases, making the model adjustments more precise. This ensures the model finds a good solution efficiently. Different methods reduce the rate in various ways, either stepwise or smoothly, to optimize the training process.

2.3.2 Mathematical Representation of Learning Rate Decay

A basic learning rate decay plan can be mathematically represented as follows:

Assume that the starting learning rate is η_0 and that the learning rate at epoch t is η_t.

A typical decay schedule for learning rates is based on a constant decay rate α, where $\alpha \in (0, 1)$, applied at regular intervals (e.g., every n epochs):

$$\eta_t = \frac{\eta_0}{1 + \alpha \cdot t} \tag{2.1}$$

Where,

- η_t is the learning rate at epoch t.
- η_0 is the initial learning rate at the start of training.
- α is the fixed decay rate, typically a small positive value, such as 0.1 or 0.01.
- t is the current epoch during training.

The learning rate η_t decreases as t increases, leading to a smaller step size as training progresses. The learning rate is decreased by a percentage of its previous value at each epoch in this formula, which depicts a basic learning rate decay schedule. A timetable like this facilitates the optimization process by enabling the model to converge more quickly at first, then fine-tuning in smaller increments as it gets closer to a local minimum.

2.3.3 Basic Decay Schedules

In order to enhance the convergence of machine learning models, learning rate decay schedules are utilized to gradually lower the learning rate during training. Here are a few simple schedules for learning rate decay:

- **Step Decay:** In step decay, after a predetermined number of training epochs, the learning rate is decreased by a specified factor (decay rate). The mathematical formula for step decay is:

$$\text{lr} = \text{lr}_{\text{initial}} \times \text{drop rate}^{\frac{\text{epoch}}{\text{step size}}} \tag{2.2}$$

- **Exponential Decay:** The learning rate is progressively decreased over time by exponential decay. At each epoch, a factor is used to adjust the learning rate. The mathematical formula for exponential decay is:

$$\text{lr} = \text{lr}_{\text{initial}} \times e^{-\text{decay rate} \times \text{epoch}} \tag{2.3}$$

- **Inverse Time Decay:** A factor inversely proportional to the number of epochs is used to reduce the learning rate through inverse decay. The mathematical formula for inverse time decay is:

$$\text{lr} = \text{lr}_{\text{initial}} \times \frac{1}{1 + \text{decay} \times \text{epoch}} \tag{2.4}$$

- **Polynomial Decay:** When a polynomial function, usually a power of the epoch number, is followed, polynomial decay lowers the learning rate. The mathematical formula for polynomial decay is:

$$\text{lr} = \text{lr}_{\text{initial}} \times \left(1 - \frac{\text{epoch}}{\text{max epoch}}\right)^{\text{power}} \tag{2.5}$$

In simple words, these schedules adjust the learning rate during training. They help in starting with big steps and taking smaller steps as we get closer to the best solution, ensuring efficiency and precision.

2.3.4 Steps Needed to Implement Learning Rate Decay

1. **Set Initial Learning Rate:** Start by establishing a base learning rate. It shouldn't be too high to cause drastic updates, nor too low to stall the learning process.
2. **Choose a Decay Method:** Common methods include exponential decay, step decay, or inverse time decay. The choice depends on your specific machine learning problem.
3. **Implement the Decay:** Apply the chosen decay method after a set number of epochs, or based on the performance of the model.
4. **Monitor and Adjust:** Keep an eye on the model's performance. If it's not improving, you might need to adjust the decay rate or the method.

2.3.5 Toy Example Visualization

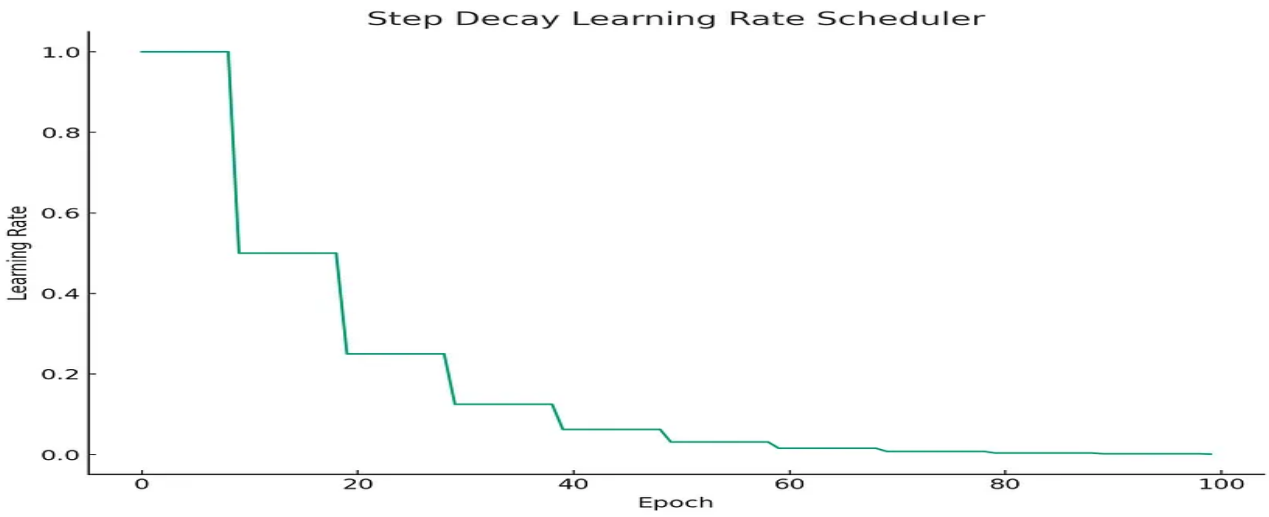
To visualize learning rate decay, consider the following toy example:

```
# Parameters
initial_lr = 1.0
decay_factor = 0.5
step_size = 10
max_epochs = 100

# Generate learning rate schedule
lr = [
  initial_lr * (decay_factor ** np.floor((1+epoch)/step_size))
  for epoch in range(max_epochs)
]

# Plot
plt.figure(figsize=(10, 7))
plt.plot(lr)
plt.title('Step Decay Learning Rate Scheduler')
plt.ylabel('Learning Rate')
plt.xlabel('Epoch')
plt.grid()
plt.show()
```

This is the result [8]:



For more information is recommended to read the artcile [9]

2.4 Exponential Decay

To better understand the impact of exponential decay on the learning rate, we can modify the parameters of the exponential decay scheduler to make the decay more visible. This allows us to observe how changes in these parameters affect the learning rate over time and the training process [8, 10].

2.5 Exponential Decay Scheduler

The exponential decay learning rate scheduler is defined by the following formula:

$$\eta_t = \eta_0 \times e^{-k \cdot \text{epoch}} \tag{2.6}$$

where:

- η_0 is the initial learning rate,
- k is the decay rate, and
- epoch is the index of the epoch.

2.5.1 Parameters and Their Impact

Initial Learning Rate (η_0)

- **Definition**: The initial learning rate η_0 determines the magnitude of the updates to the model's parameters in the early stages of training.
- **Impact**: A larger initial learning rate allows for more aggressive parameter updates in the beginning. This can accelerate learning initially but may lead to overshooting the optimal solution if set too high. For example, setting $\eta_0 = 2.0$ results in a higher starting learning rate.

2.5.2 Decay Rate (k)

- **Definition**: The decay rate k controls how quickly the learning rate decreases over time.
- **Impact**: A larger decay rate results in a faster decrease in the learning rate. This facilitates quicker reductions in the learning rate as training progresses, which can help in fine-tuning the model more precisely in later stages. For instance, setting $k = 0.1$ results in a more rapid decline in the learning rate compared to a smaller k value.

2.5.3 Epoch Index

- **Definition**: The epoch index represents the current iteration or cycle of training.
- **Impact**: As the number of epochs increases, the learning rate decreases exponentially. This setup allows for larger updates initially and finer adjustments as the model approaches a potential solution.

2.6 Visualizing the Effects

Using a larger initial learning rate and a larger decay rate will make the learning rate decay more visible and allow us to observe its effects more clearly. For example, with $\eta_0 = 2.0$ and $k = 0.1$, the learning rate will start at a higher value and decay more rapidly, showing a steeper decline in the learning rate curve.

2.7 Practical Considerations

- **Choosing** η_0: While a larger η_0 speeds up convergence initially, it may increase the risk of diverging from the optimal solution if not tuned properly. Balancing the initial learning rate is essential to ensure efficient learning without instability.
- **Tuning** k: A larger decay rate k results in a quicker decrease of the learning rate, which might be beneficial for early aggressive learning followed by fine-tuning. However, an excessively large k might make the learning rate too small too soon, leading to slow convergence later in training.

- **Empirical Testing**: Empirical testing of different η_0 and k values in your specific training scenario is crucial. Optimal values can vary based on model complexity and dataset characteristics.

By experimenting with these parameters, you can gain insights into how exponential decay affects the training process and adjust your learning rate schedule to improve model performance and convergence.

2.7.1 Toy Example Visualization

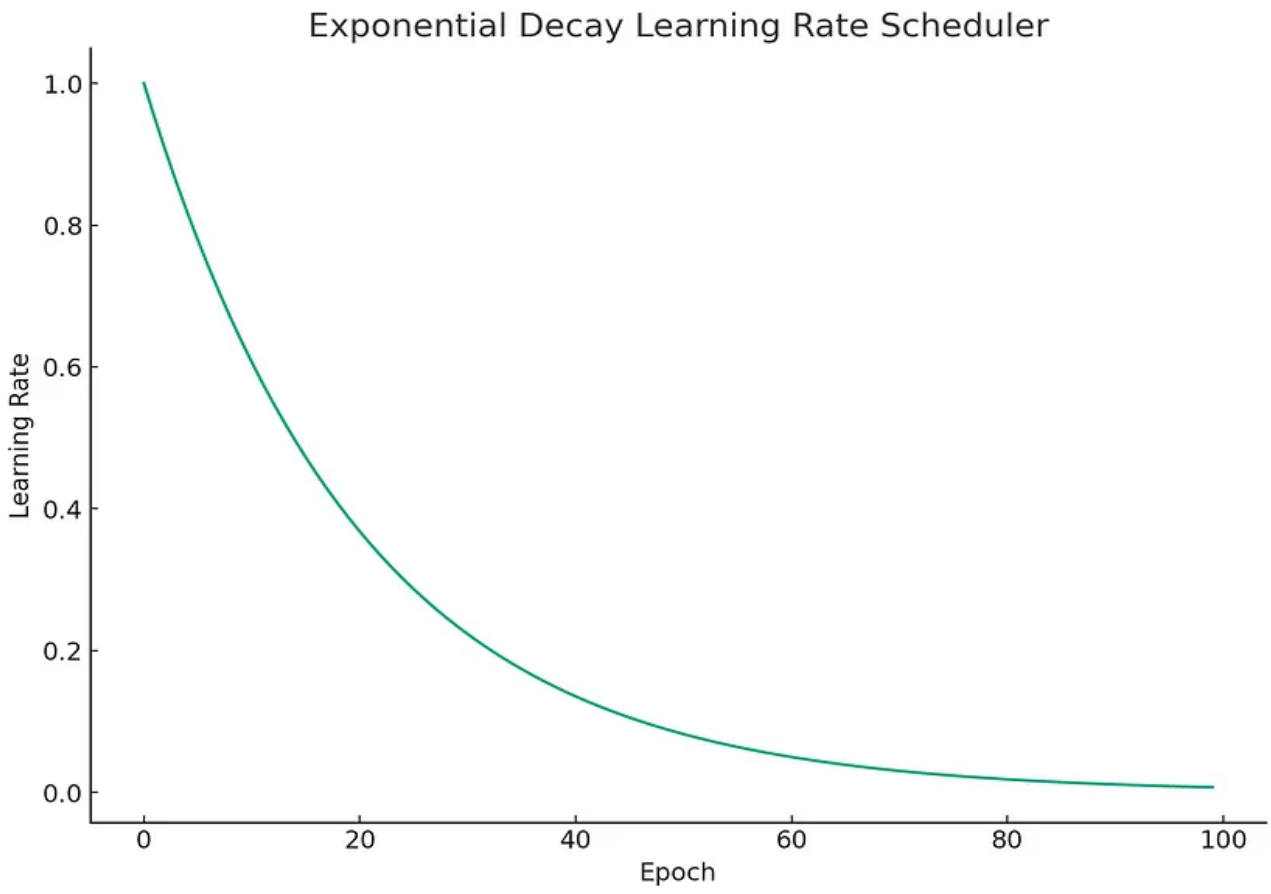
To visualize learning rate decay, consider the following toy example:

```
# Parameters
initial_lr = 1.0
decay_rate = 0.05
max_epochs = 100

# Generate learning rate schedule
lr = [
  initial_lr * np.exp(-decay_rate * epoch)
  for epoch in range(max_epochs)
]

# Plot
plt.figure(figsize=(10, 7))
plt.plot(lr)
plt.title('Exponential Decay Learning Rate Scheduler')
plt.ylabel('Learning Rate')
plt.xlabel('Epoch')
plt.grid()
plt.show()
```

This is the result [8]:



2.8 Cyclical Learning Rate (CLR)

Cyclical Learning Rate (CLR) is a technique that varies the learning rate cyclically within a predefined range during training. This approach oscillates the learning rate between a minimum and maximum value in a triangular pattern, enabling exploration of different rates and avoiding poor local minima while accelerating convergence [11].

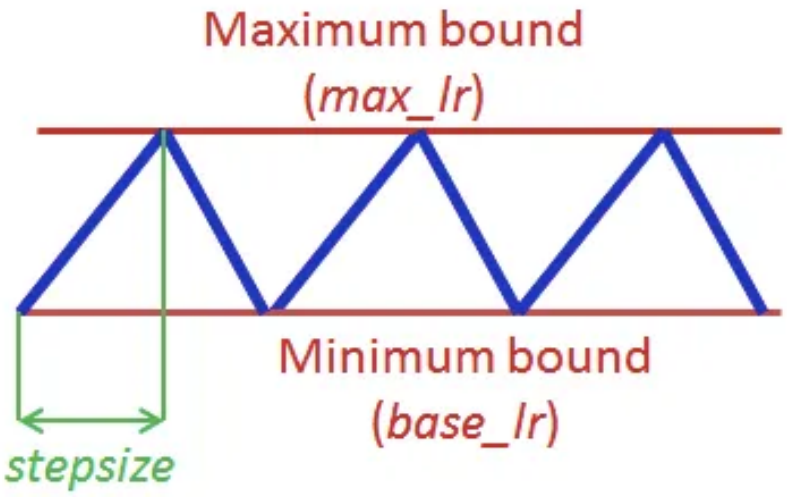
2.8.1 Mechanism and Benefits

- **Cyclic Variation**: CLR maintains a constant frequency of learning rate fluctuations, typically in a triangular pattern where the rate increases linearly to a maximum and then decreases back to a minimum. This cyclical nature helps the model escape from local minima and prevents it from converging too slowly or diverging.
- **Avoiding Saddle Points**: According to Dauphin et al. (2014), saddle points, not local minima, are significant obstacles in optimizing deep neural networks. CLR helps address this issue by rapidly traversing saddle point plateaus due to its cyclic nature, which maintains both high and low learning rates.
- **Function Types**: The oscillation pattern of CLR can be based on various functions, including triangular (linear), Welch window (parabolic), or Hann window (sinusoidal). The triangular window is a simpler approach where the learning rate increases and decreases linearly.

CLR is beneficial because it allows for periodic adjustments in learning rates, aiding in faster convergence and better exploration of the solution space.

2.8.2 Triangular window

In this, the learning rate changing logic is linear that is we will increase the learning rate with some constant from min learning rate to max learning rate and will decrease the learning rate with the same constant from max learning rate to minimum learning rate [12].



2.9 Cosine Annealing

Cosine Annealing is a learning rate scheduling technique that reduces the learning rate using a cosine-based schedule. The learning rate η_t at epoch t is defined by the following formula:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min}) \left(1 + \cos\left(\frac{t\pi}{\text{max\_epochs}}\right)\right) \tag{2.7}$$

where:

- $\eta_{\min}$ is the minimum learning rate,
- $\eta_{\max}$ is the maximum learning rate,

- t is the current epoch, and
- max_epochs is the maximum number of epochs.

2.9.1 Mechanism and Benefits

- **Cosine Decay**: The learning rate starts at the maximum value and decreases following a cosine function, reaching the minimum value at the end of the schedule. This gradual decay allows for a smooth transition and stabilization towards the end of training.
- **Smoother Adjustment**: Cosine Annealing provides a smoother adjustment of the learning rate compared to linear or triangular schedules. The cosine function ensures that the learning rate decays gently, which can improve convergence and training stability.
- **Application in Training**: This schedule is particularly useful in scenarios where a gradual reduction in the learning rate is beneficial. It has been shown to be effective in various training regimes, especially for deep learning models where stability in the final stages of training is crucial [8].

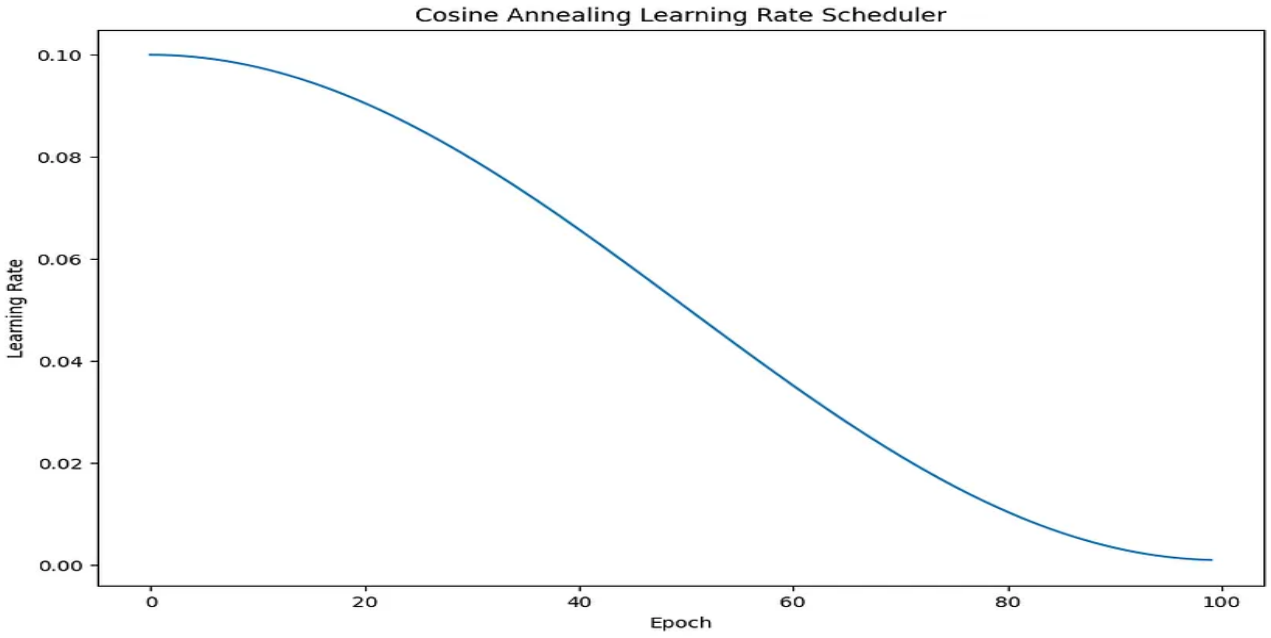
2.9.2 Toy Example Visualization

```
# Parameters
lr_min = 0.001
lr_max = 0.1
max_epochs = 100

# Generate learning rate schedule
lr = [
    lr_min + 0.5 * (lr_max - lr_min) * (1 + np.cos(epoch / max_epochs * np.pi))
    for epoch in range(max_epochs)
]

# Plot
plt.figure(figsize=(10, 7))
plt.plot(lr)
plt.title("Cosine Annealing Learning Rate Scheduler")
plt.ylabel("Learning Rate")
plt.xlabel("Epoch")
plt.show()
```

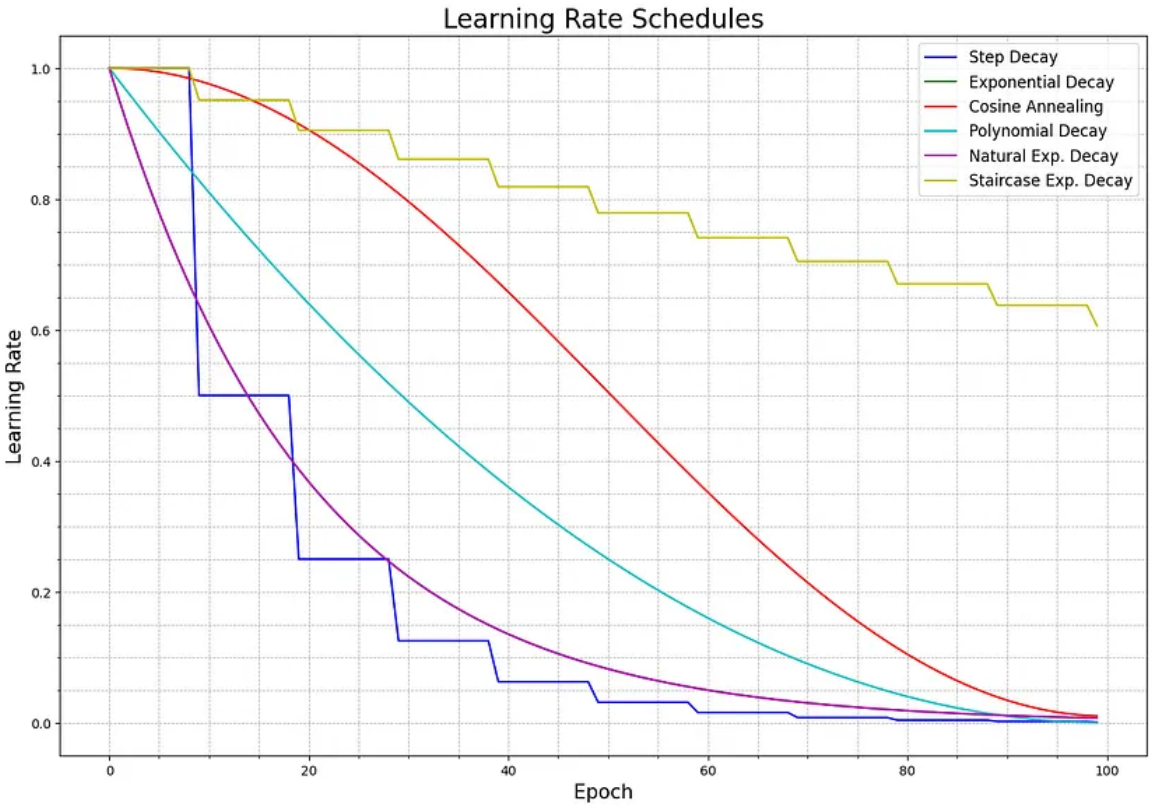
This is the result :



Learning rate schedulers are an important tool in the machine learning practitioner's toolkit, providing a mechanism to adjust the learning rate over time, which can help to improve the efficiency and effectiveness of the training process. The best learning rate scheduler to use can depend on the specific problem and dataset, and it is often helpful to experiment with different schedulers to see which one works best.

2.9.3 Conclusion

Some more learning rates in one plot [8].



References

[1] Katherine (Yi) Li.

[2] https://www.tutorialspoint.com/what-is-learning-rate-in-neural networks. https://www.tutorialspoint.com/what-is-learning-rate-in-neural-networks.

[3] https://www.geeksforgeeks.org/impact-of-learning-rate-on-a model/. https://www.geeksforgeeks.org/impact-of-learning-rate-on-a-model/.

[4] Suki Lau.

[5] Jiakai Wei. Forget the learning rate, decay loss.

[6] Vrunda Bhattbhatt. https://medium.com/thedeephub/learning-rate-and-its-strategies-in-neural-network-training-270a91ea0e5c.

[7] https://www.geeksforgeeks.org/learning-rate decay/. https://www.geeksforgeeks.org/learning-rate-decay/.

[8] Théo Martin. https://medium.com/@theom/a-very-short-visual-introduction-to-learning-rate-schedulers-with-code-189eddffdb00.

[9] Yimin Ding et al. The impact of learning rate decay and periodical learning rate restart on artificial neural network.

[10] Chat GPT-4.

[11] Sanket Doshi.

[12] Leslie N. Smith. Cyclical learning rates for training neural networks.