Module 3: How to Tune Your Models

Welcome to the next phase of our Neural Network series: How to Tune Your Models. This module will detail the core mechanism that powers learning in neural networks. This module will focus on gradient descent and how it optimizes neural networks to make more precise predictions. We will touch briefly on different modifications to gradient descent and other optimizer options. But, in general, remember that they all share the goal of determining how to adjust the parameters to train the model to make better predictions given the input training data. You'll learn how error functions and optimizers work together to measure and minimize (respectively). Finally, you'll experiment with hyperparameter tuning in training a model.

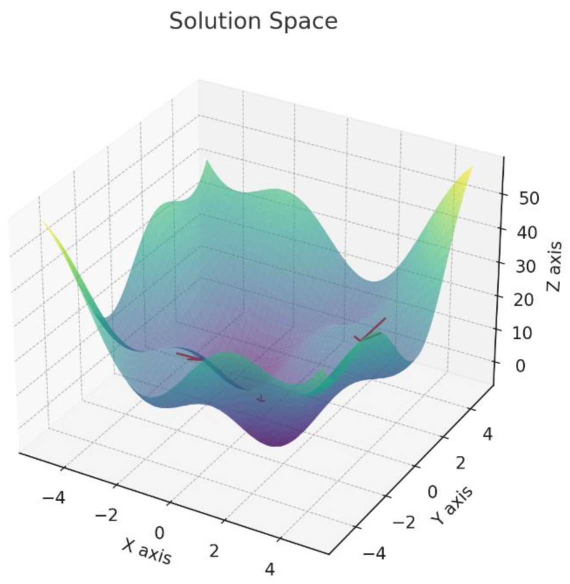
Module 3 Objectives:

By the end of this module, you will be able to:

1. Describe the purpose and process of gradient descent.
2. Discuss the error loss function.
3. Describe optimizers.
4. Experiment with hyperparameter tuning.

Understanding Gradient Descent

Gradient Descent is a foundational algorithm in machine learning that is essential for training various models, mainly neural networks. It is an optimization strategy, guiding models to minimize error or loss. At its core, Gradient Descent operates on the principle of iterative improvement. The algorithm starts with initial estimates for the parameters of a model and then incrementally adjusts them to find the combination that minimizes the error. It uses the gradient, a mathematical concept representing the direction and rate of the fastest increase of a function, to navigate the multi-dimensional landscape of the model's error function.



The Gradient Descent Process

- **Error Function and Its Importance:** The error function, or the cost or loss function, quantifies how far a model's predictions are from the actual outcomes. In the context of Gradient Descent, it's the terrain we navigate to find the lowest point, representing the smallest error. While the appropriate error function may be relatively easy in simple models like linear regression, how do you best quantify the error in image classification if the model classifies an image of pizza as a cat? It's wrong, but how bad? Is it better to classify pizza as bread?

- **The Gradient and How It's Computed:** The gradient is the vector of partial derivatives of the error function for each parameter. Computing it involves calculating how much a slight change in each parameter affects the error. This calculation is pivotal, as it points each parameter's vector in the direction where the function decreases most steeply.

- **The Descent: Step Size and Direction:** Gradient Descent moves in the opposite direction of the gradient, towards lower error. The step size, controlled by the learning rate parameter, determines each step's size. If the steps are too large, the algorithm might overshoot the minimum; if too small, the process becomes slow and may get stuck in local minima. The art of Gradient Descent lies in balancing these factors to efficiently reach the lowest point of the error function.

Introduction to Error and Loss Functions

Error and loss functions are fundamental to machine learning, acting as the guiding stars for models during their training phase. These functions quantify the difference between the predicted outputs of a model and the actual target values, offering a measurable way to evaluate and improve model performance.

Types of Loss Functions Used in Machine Learning

Different machine learning problems call for other loss functions. Here are some commonly used ones:

- **Mean Squared Error (MSE):** A staple in regression tasks, MSE measures the average squared difference between the estimated and actual values. It's handy when larger errors are more significant, as the squaring operation emphasizes larger differences (in addition to making all values positive).
- **Cross-Entropy:** Widely used in classification tasks, cross-entropy measures the difference between two probability distributions - the actual distribution and the model predictions. It's especially effective in scenarios where the prediction of the probability of a class is required.
- **Hinge Loss:** Often used in support vector machines (SVMs, a ML method) and some types of neural networks, particularly for "maximum-margin" classification.
- **Huber Loss:** A combination of MSE and Mean Absolute Error, it's less sensitive to outliers than MSE.
- **Logistic Loss:** Used in logistic regression, measuring the difference between binary or multiclass labels and predictions.

Optimizers and Advanced Gradient Descent Techniques

Optimization algorithms vary in their approach to navigating the loss landscape. Remember, not only do we not know what the entire landscape looks like (all the algorithms know is from where I stand now, how can I move to a lower point), but it is highly dimensional. Navigating the hyperdimensional universe in the dark can be a challenge! Here's a look at some common types:

Stochastic Gradient Descent (SGD): This algorithm represents the simplest form of gradient descent. It updates the model's weights using only one sample at each iteration. While this can make SGD faster and more memory-efficient, it can also make the optimization path noisy and inconsistent.

Batch Gradient Descent: This algorithm simultaneously updates the model parameters using the entire dataset. Unlike SGD, which updates parameters using only one data point at a time, batch gradient descent computes the average gradient from all data points, leading to more stable and consistent parameter updates. However, this can be computationally intensive for large datasets and less efficient than SGD.

Mini-batch Gradient Descent: Striking a balance between batch gradient descent and SGD, this approach uses a subset of the data at each step. This middle ground makes it more stable than SGD and often more efficient than using the entire dataset.

Momentum-based Optimizers:

- **Adam (Adaptive Moment Estimation):** Adam combines the ideas of momentum (considering past gradients) and adaptive learning rates (adjusting the learning rate for each parameter) to make more effective updates.
- **RMSprop (Root Mean Square Propagation):** This optimizer dynamically adjusts each parameter's learning rate, making it effective for problems with noisy or sparse gradients.

Advanced Concepts

- **Learning Rate Decay:** Over time, reducing the learning rate can help the optimizer settle into the minimum of the loss function. This technique is especially useful to prevent overshooting in the later stages of training.
- **Adaptive Learning Rates:** Algorithms like Adam and RMSprop adjust the learning rate for each parameter, which can lead to more effective training by customizing the step size based on the behavior of each parameter.

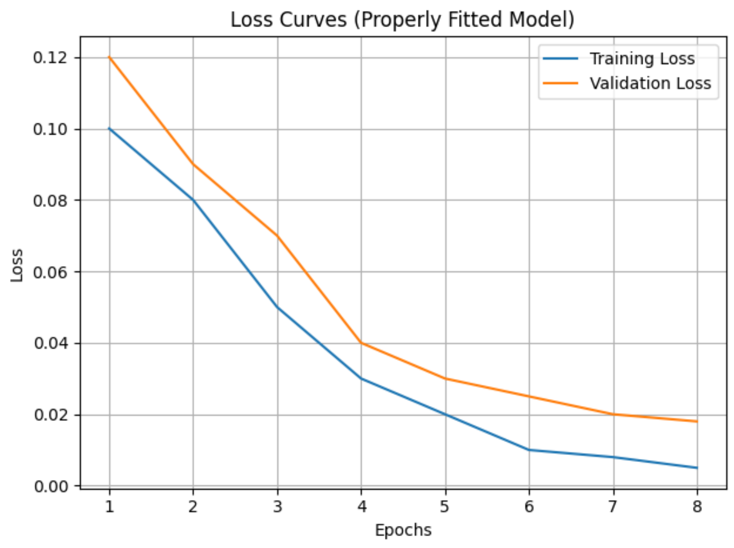
Choosing the Right Optimizer: A Guide

Selecting the right optimizer for a specific problem can significantly impact model performance. Here are some guidelines:

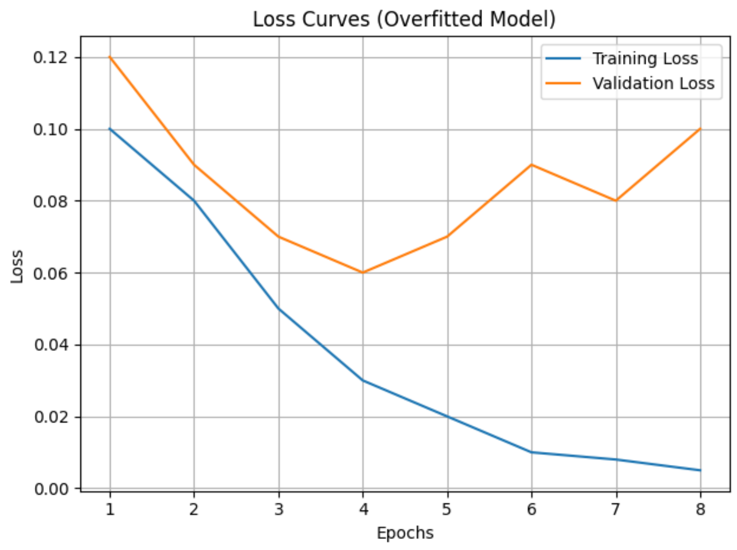
- Consider optimizers like Adam or RMSprop for sparse data, as they handle noise well.
- Traditional SGD can be effective and easier to control for simpler models or smaller datasets.
- Experiment with different learning rates and observe the training process. Sometimes, a combination of optimizers used in different training phases can yield the best results.
- Stay informed about the latest developments in optimization algorithms, as this is an actively evolving area in machine learning research.

Overfitting and Underfitting

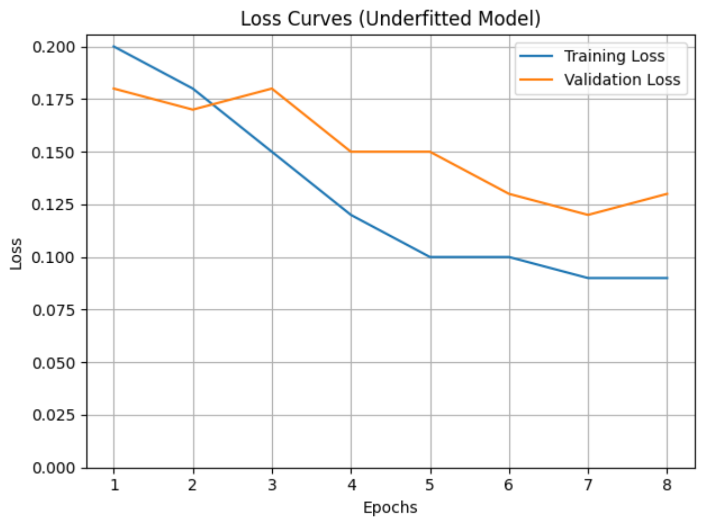
Supervised learning models aim to learn the relationship between input data and desired outputs. Striking the right balance between complexity and accuracy is crucial. Here, we explore two common issues in model training: overfitting and underfitting. As a guideline, if a model’s accuracy improves on training *and* validation, the model is generalizing well (though it could still have other issues, such as overall low accuracy…). Here is an example loss graph showing a model that is probably fitting properly:



Overfitting occurs when a model becomes too focused on the specific training examples it has been shown. It essentially memorizes the training data rather than capturing the underlying patterns that generalize to unseen data. Symptoms include high accuracy on the training set but poor performance on new data. Here is an example loss graph showing a model that is probably overfitting:



Underfitting, on the other hand, happens when a model is too simple and lacks the capacity to learn the complexities within the training data itself. This leads to low accuracy on both the training and validation sets. Here is an example loss graph showing a model that is probably underfitting:



Tips for Tackling Overfitting and Underfitting

To combat overfitting, we can employ techniques like:

- Data augmentation, where we artificially create new training examples. This is usually done by making common-sense changes to existing data (like taking a sample picture and mirroring it, thus giving us two useful samples from one).
- Regularization, where we penalize overly complex models. Some popular examples of regularization are Dropout (having the model randomly "turn off" some number of neurons during each epoch to prevent overreliance on them) and Early Stopping (having the model stop training early if it detects overfitting).

For underfitting, we can employ strategies like:

- Using more complex models (with more layers, neurons, more sophisticated activation functions, etc.).
- Increasing the amount of training data.
- Employing feature engineering to reduce the model's complexity. We want to remove redundant or highly correlated features!

Transfer Learning & Fine Tuning

Transfer Learning and Fine Tuning are both topics we'll cover in much more detail in our Intermediate Series. For now, we'll provide you with a quick primer, so you'll be familiar with the concepts when you hear them out in the wild.

Transfer Learning

Transfer learning is a machine learning method where a model developed for one task is repurposed as the starting point for a model on a second task. It is a popular approach in deep learning where pre-trained models are used as the basis for performing new tasks. Technically, this involves taking layers from a model (their architecture, hyperparameters and trained parameters) already trained on a dataset and leveraging that learned feature map for a new task with potentially less data. An example of this you may have already used is Google's Teachable Machine (TM), which uses Google's MobileNet model as the base. When you provide new images to TM, it adds a layer to the top of the pre-trained model that is trained on your new images. This is how TM can recognize images with just a few hundred examples, rather than the tens of thousands (or even more) typically required for a computer vision model.

Fine-Tuning

Fine-tuning a model refers to the process of taking a pre-trained model (like those used in transfer learning) and continuing the training process to adjust the weights for a specific task. It usually involves unfreezing the entire model or a portion of it, and training it again with a smaller learning rate, to not destroy the previously learned features. The fine-tuning adjusts the specialized features to make the model more relevant for the specific task at hand. This process typically requires less data and computation than training a model from scratch. An example would be taking a pre-trained large language model and fine-tuning it with legal documents to specialize the model to reproduce legal-sounding text.

Practice and Apply

Through this exercise, you'll gain hands-on experience applying deep learning to solve another image recognition task. As discussed above, the loss function, optimizer, and more are all hyperparameters that can be adjusted to try to train a better model. You will get some experience with hyperparameter optimization.

Work on notebook **03_bees_vs_wasps.ipynb**.

In this module, we explored essential concepts and techniques in machine learning optimization:

1. **Basics of Gradient Descent:** Introduced the algorithm as a method for minimizing loss functions in machine learning models.
2. **Loss Functions:** Discussed various types, such as Mean Squared Error and Cross-Entropy, highlighting their role in model accuracy.
3. **Optimizers:** Covered different optimizers like Stochastic Gradient Descent (SGD) and Adam, focusing on their application in refining the learning process.
4. **Transfer Learning & Fine Tuning:** Touched on these important, advanced deep learning techniques.
5. **Deep Learning Implementation:** Explored hyperparameter optimization to train better models.

This module provided a foundational understanding of how gradient descent drives the learning process in machine learning and deep learning models.