Deep Dive into ANN-Parameter Experiments

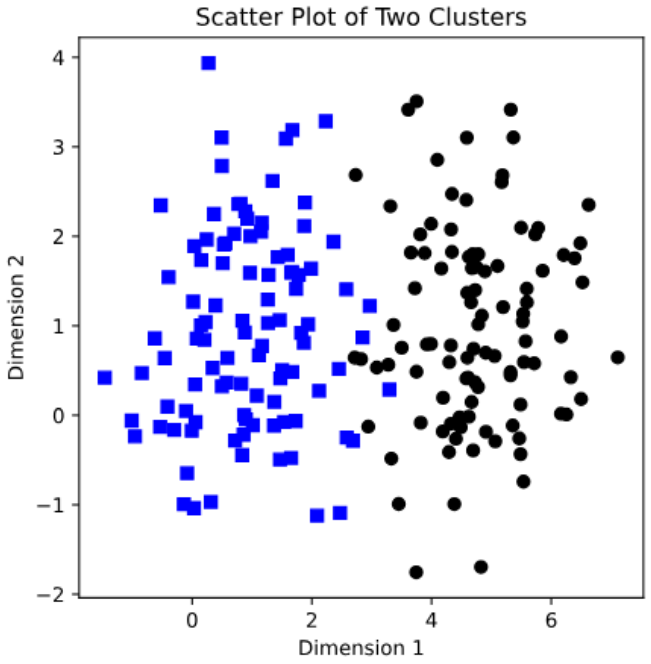
Farid Afzali, Ph.D., P.Eng.

Learning Rate's Role in Deep Learning

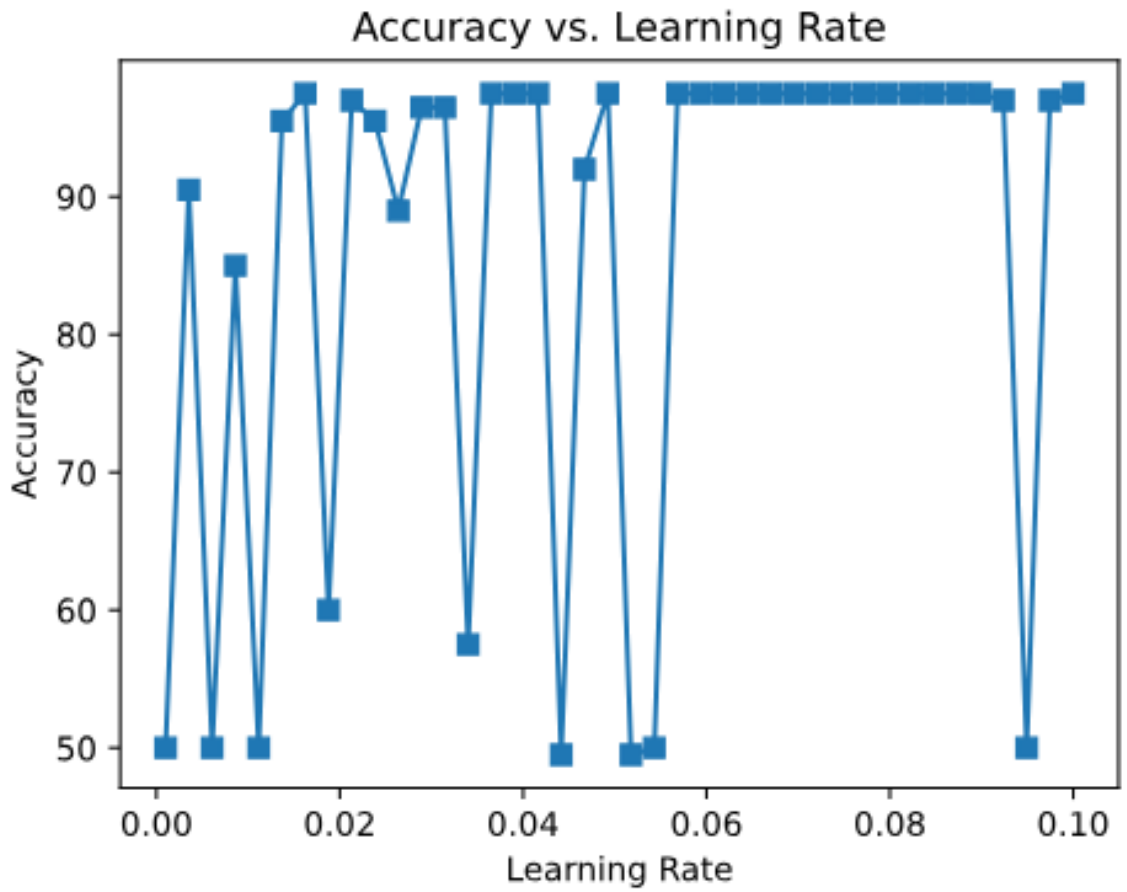
- In this section, we discuss the effects of learning rates on deep learning models.
- Different learning rates influence model performance, and we will gain insights into the learning process.
- The learning rate, a crucial hyperparameter, plays a significant role in the training of deep learning models.
- It determines the step size during gradient descent, impacting how quickly the model converges to a solution.
- Understanding the Error Landscape
  - Imagine the error landscape as a multi-dimensional space where each dimension represents a model parameter. Visualizing this landscape is complex due to the high dimensionality, but it's essential to comprehend how learning rates affect the model.

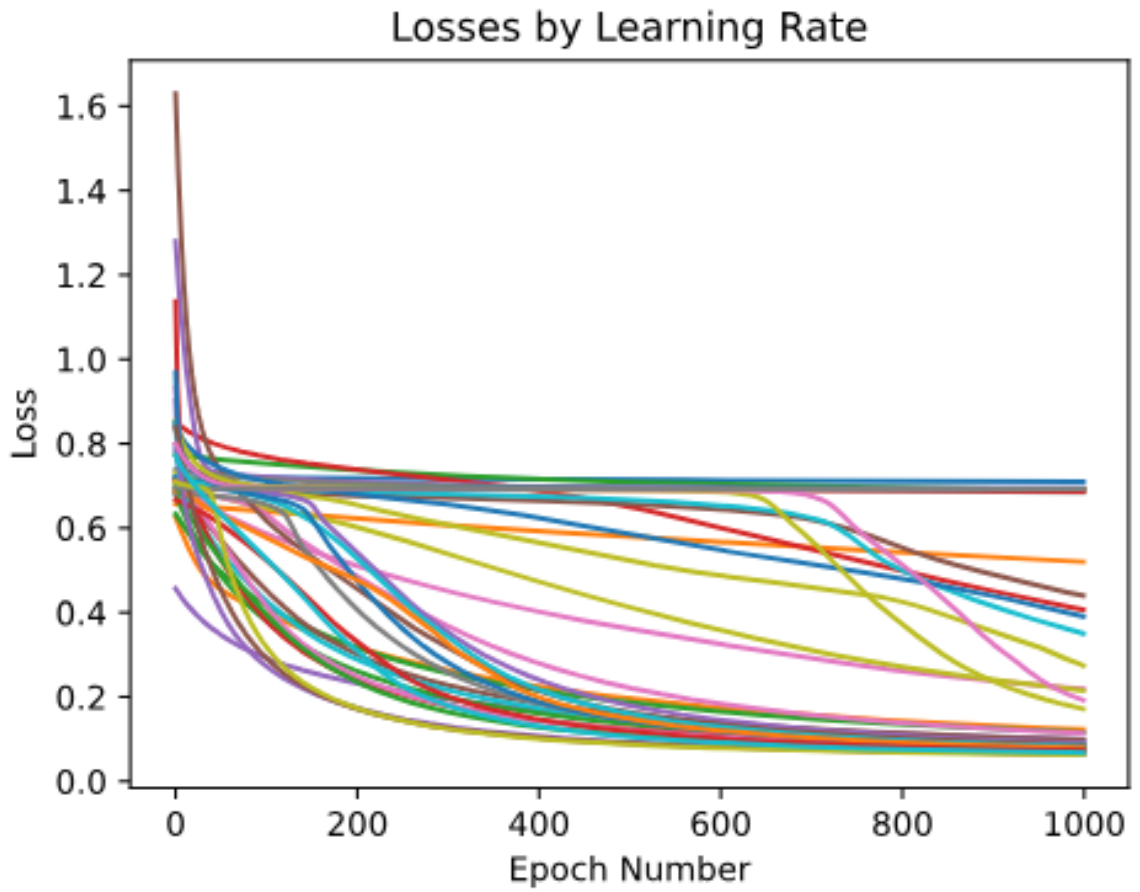
Importance of Learning Rate

- The learning rate directly influences the model's convergence.
  - A large learning rate may lead to rapid convergence, but it risks overshooting optimal solutions.
  - In contrast, a small learning rate may slow down learning or trap the model in local minima.
- Variability in Model Performance
  - ❑ Our experiments reveal a puzzling pattern of model performance.
  - ❑ Deep learning models seem to either excel or fail dramatically, with little middle ground.
  - ❑ This bimodal distribution suggests that chance plays a significant role in model outcomes.



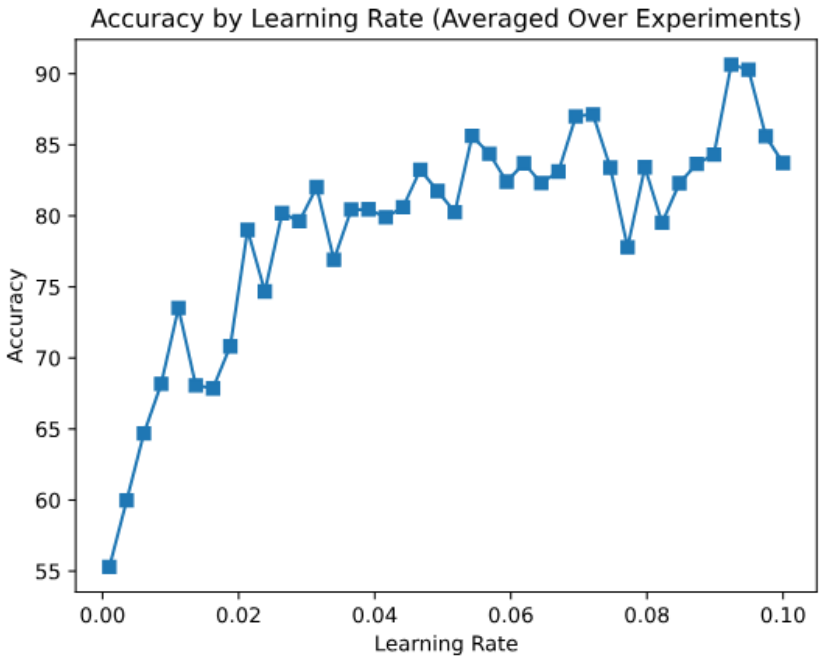
Importance of Learning Rate (PP#1)





Meta-Experiments for Clarity

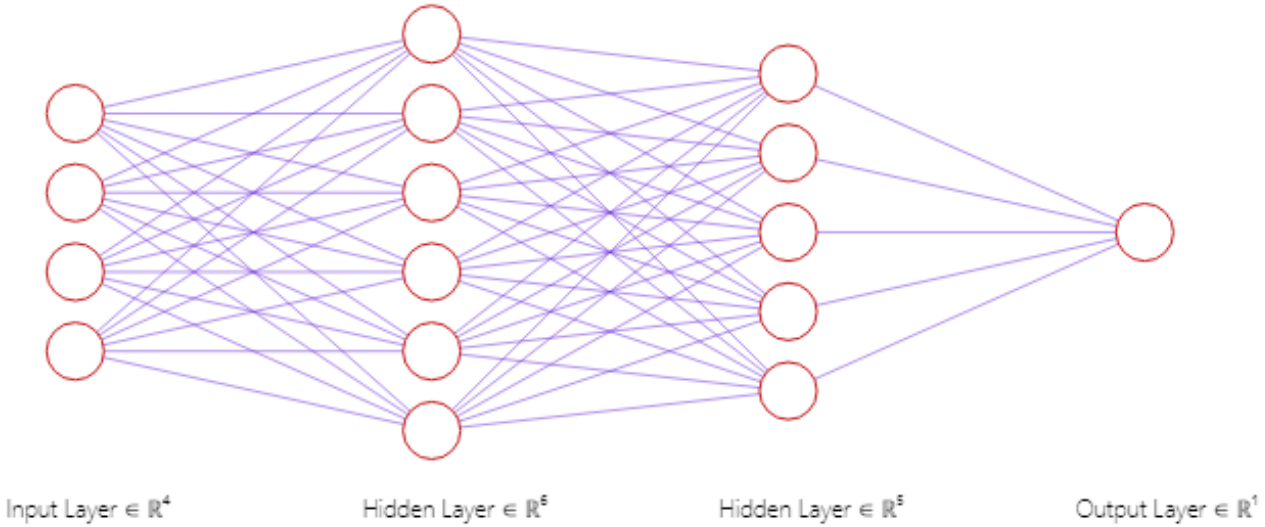
- To overcome the variability in individual experiment results, we conducted meta-experiments.
  - ❑ Repeating the experiment multiple times and averaging the outcomes, providing a clearer view of the relationship between learning rates and accuracy.
- Revealing the Learning Rate Effect
  - ❑ Extremely small learning rates generally impede learning, while larger learning rates facilitate better model performance.
  - ❑ This finding sheds light on the delicate balance between learning rates and model convergence.
- Conclusion
  - ❑ By conducting meta-experiments and exploring the impact of learning rates on model performance, we've gained valuable insights into the intricacies of deep learning.

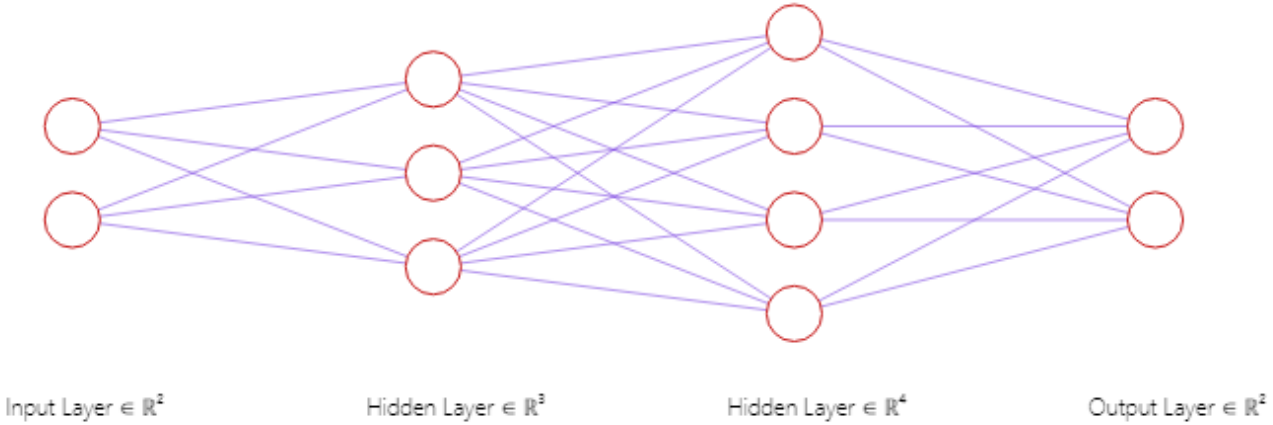


Constructing Multi-Layer Artificial Neural Networks

- In the previous practice, the model's performance exhibited a binary pattern – achieving high accuracy or operating at a chance level when categorizing the quartz dataset.
- An exploration was conducted to investigate whether the model's complexity was a limiting factor.
- **Architecture of the Deep Learning Network**
  - ❑ ANN layers comprising an input layer, hidden layers, and an output layer.
  - ❑ The input layer establishes a connection with the external world, accepting data from diverse sources.
  - ❑ The hidden layers, conversely, remain confined within the network, devoid of interaction with external data. They receive inputs from preceding layers.

Constructing Multi-Layer Artificial Neural Networks





Python-Based Model Construction (PP#2)

- ❑ Incorporation of linear layers succeeded by non-linear activation functions.
- ❑ Variation in the choice of activation functions – ReLU within input and hidden layers, and a sigmoid function within the output layer.
- ❑ Definition of the number of input and output features for each layer, guaranteeing compatibility.
- ❑ Discourse surrounding the inclusion of a sigmoid function in the output layer and its underlying significance.

- **Model Training**
  - ➢ A noteworthy change lies in the determination of the decision boundary. While previous code stipulated a boundary of 0.0, the current code establishes it at 0.5, attributing this alteration to the presence of the sigmoid function within the model.
- **Testing and Experimental Insights**
  - ○ Proceeding with model testing and experimentation, we systematically alter learning rates, spanning from 0.001 to 0.1 in a linear progression. The outcomes reveal that the model's performance predominantly conforms to two extremities – either achieving high accuracy or operating at a chance-level, with limited instances of intermediate performance.

Unveiling the Influence of Linearity in Deep Learning Models

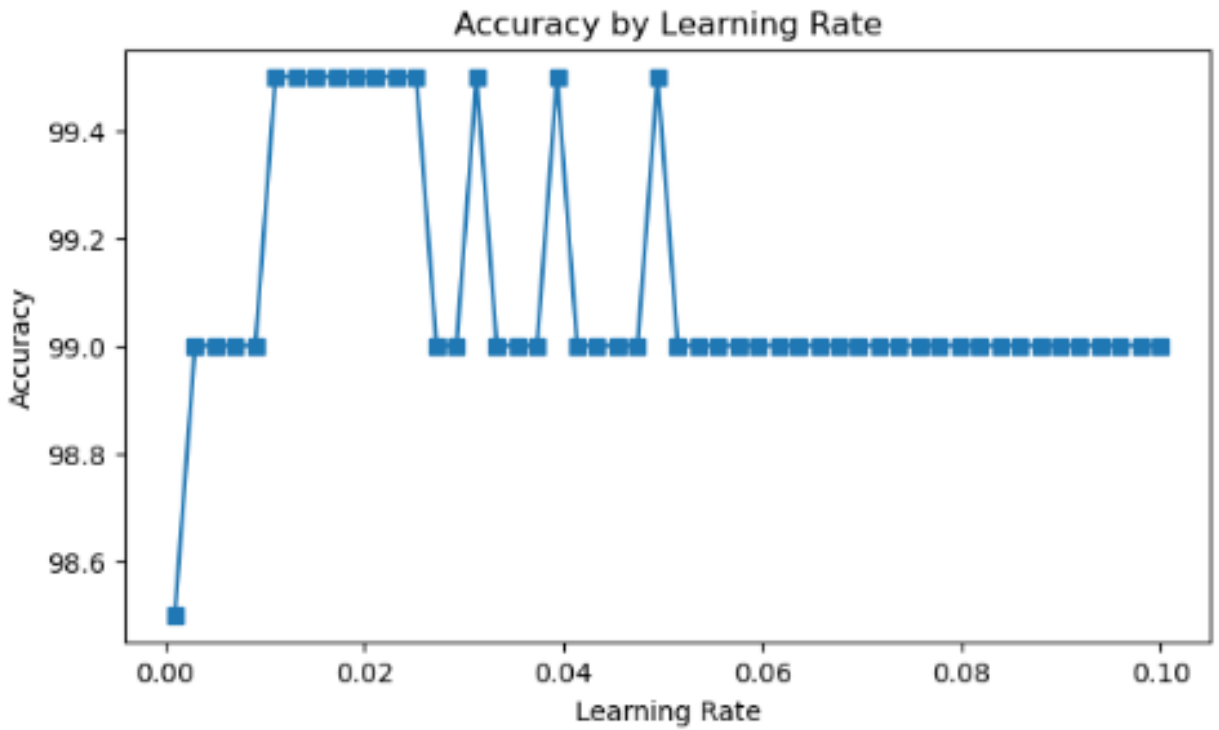
- In the realm of deep learning, it is imperative to recognize the potential ramifications of model complexity, a subject matter that this discourse aims to shed light upon.
- **Revelations in Model Transformation**
  - ❑In a noteworthy revelation, we explore the remarkable impact of a seemingly minor adjustment to our existing model.
  - ❑This transformation pertains to the removal of nonlinear activation functions within the hidden layers while preserving the non-linearity within the output layer.
  - ❑With this minimalistic alteration, we embark on a quest to decode the enigma surrounding keyword classification.

Code Challenge: A Deceptive Simplicity(PP#3)

- **An Engaging Coding Challenge**
  - This process may resemble a coding challenge, but it's surprisingly straightforward.
  - Start with the existing code and make a simple alteration: remove the non-linear components.
  - Subsequently, evaluate how this streamlined model performs compared to its more intricate counterpart.
- **Analyzing the Outcomes**
  - ❖ When subjecting this simplified model to testing, a remarkable phenomenon emerges.
  - ❖ It exhibits outstanding performance, achieving nearly flawless accuracy, whereas the complex model often struggled to yield consistent results.
- **Deciphering the Significance**
  - ❑ This leads us to a crucial insight: for uncomplicated problems, simpler models often outperform their complex counterparts.
  - ❑ Although intricate non-linear models may seem enticing, they can be excessive for tasks that don't demand their sophistication.

Essential Takeaways

- The lesson here is not to employ a complex model merely because it's feasible.
- Approach each problem with a critical, open mind.
- Occasionally, a basic model—or an entirely different approach—proves to be the correct choice.
- **In Closing**
  - ❑ Ultimately, this revelation serves as a reminder to carefully consider the complexity of our models.
  - ❑ While deep learning wields immense power, it's not always the ideal solution.
  - ❑ Recognizing when to opt for simpler models is as vital as understanding how to construct intricate ones.



The Essence of Non-Linearity in Deep Learning Models

- In the preceding discourse, a fundamental proposition emerged: within the domain of deep learning, models can be reduced to a solitary layer if devoid of non-linear activation functions.

- **The Crucial Need for Non-Linearity**
  - The cardinal tenet of this exposition is that the inclusion of non-linear elements between layers is imperative. Whether in the form of non-linear activations or other constructs, this non-linearity serves as the linchpin that imparts depth to deep learning models.

- **Unveiling the Perceptron Model**
- ????

A Numerical Insight

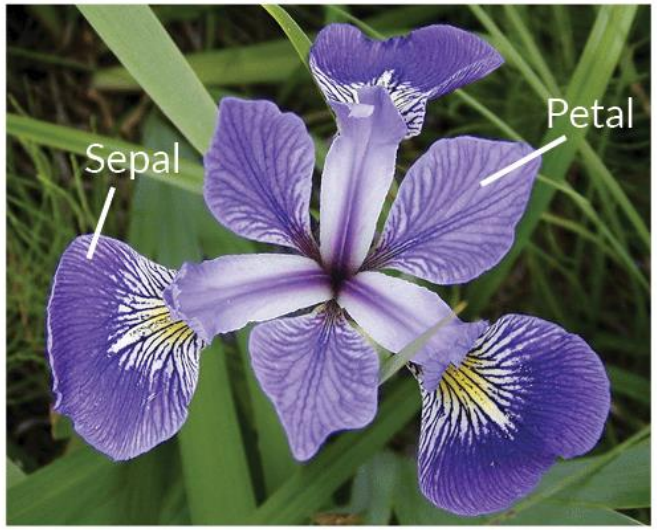
- To illustrate this concept more concretely, consider a numerical example contrasting linear and non-linear operations.
- The non-linear operation—a logarithm—does not yield separable outcomes when addition intervenes.
- Conversely, linear operations possess the distributive property, permitting the decomposition of the equation.
- **The Imperative of Non-Linearity Reiterated**
  - ❑This elucidation underscores the inescapable necessity of incorporating non-linear transformations between layers in deep learning models.
  - ❑ Without these vital non-linear elements, the intricate layers dissolve into a monolithic entity, undermining the essence of deep learning's depth.

Multi-Class Classification

- In this section, we are going to tackle multi-class classification problems. We'll explore network architecture, key terminologies in deep learning, and practical implementation in PyTorch.
- **The Famous Iris Dataset**
  - ❑ We commence with the venerable **iris** dataset, a classic in the realm of machine learning and statistics.
  - ❑ Over nearly a century, it has been instrumental in multivariate classification studies. We measure various attributes of iris flowers to predict their species based on these measurements.
- **Model Architecture**
  - ❑ Our model architecture is depicted as a fully connected neural network with four input nodes corresponding to the four feature dimensions.
  - ❑ We then have a hidden layer with 64 units, and finally, an output layer with three units, representing the three iris species.
  - ❑ Activation functions, specifically ReLU, provide the non-linearity between layers.

Multi-Class Classification/IRIS Dataset(PP#4)

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

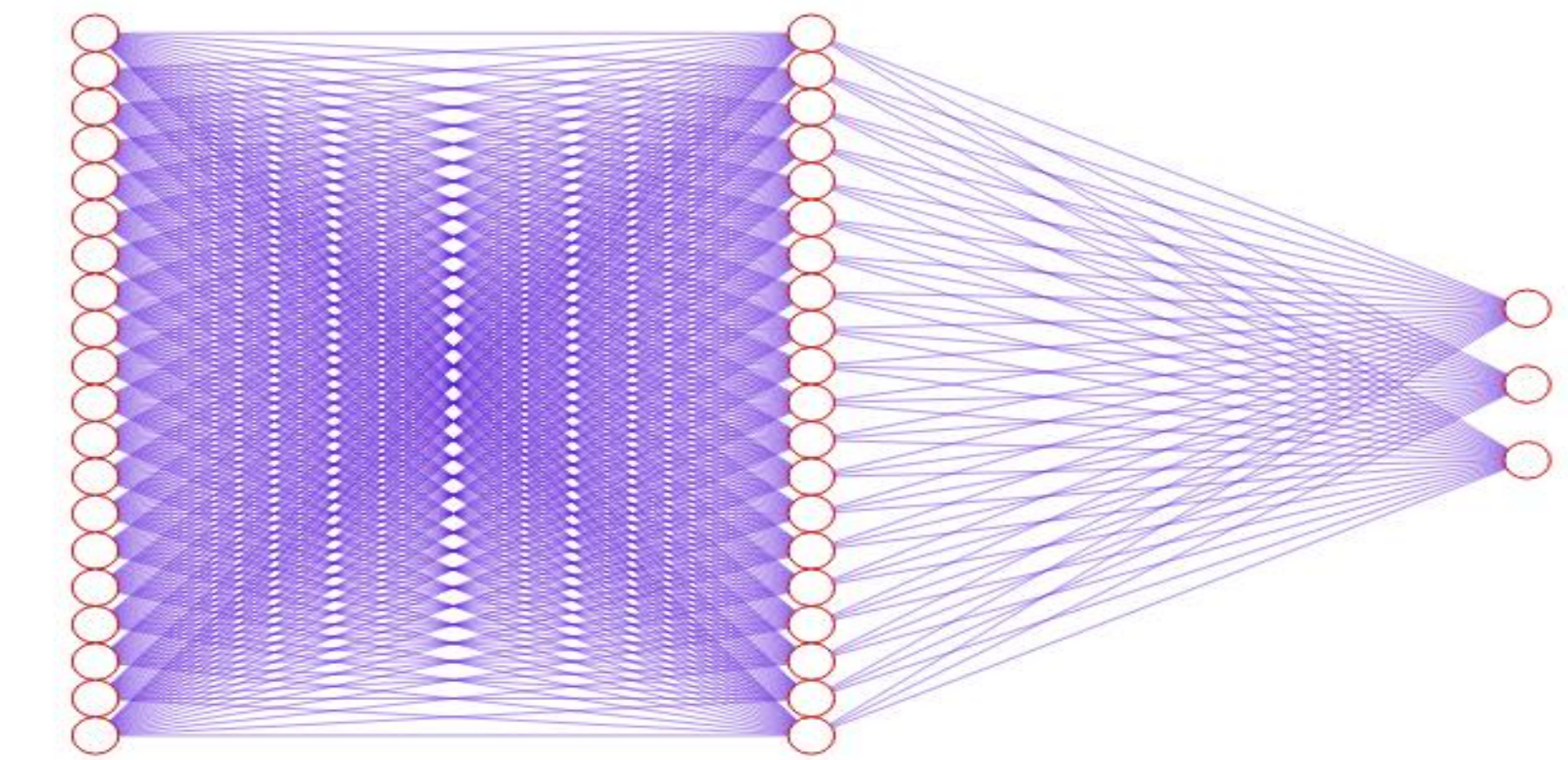


Iris Versicolor

Iris Setosa

Iris Virginica

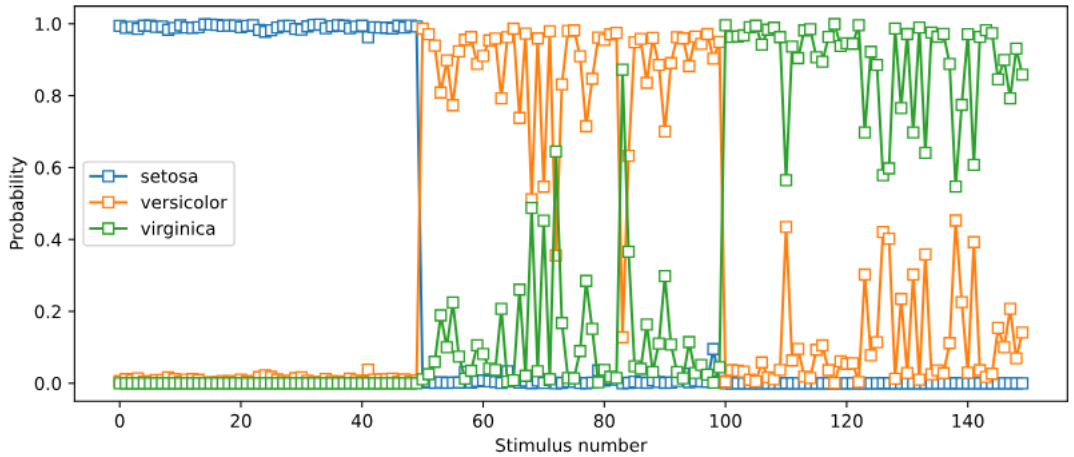
Multi-Class Classification



The Role of Softmax

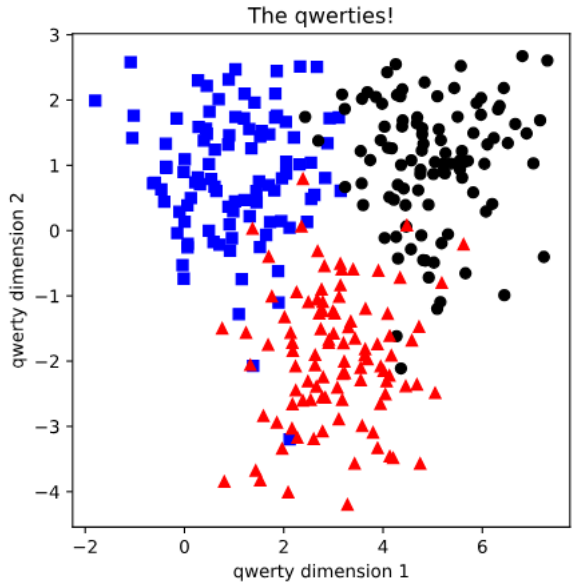
- ✓ A pivotal aspect in multi-class classification is the use of the softmax function after the output layer.
- ✓ Unlike binary classification where we employ sigmoid functions, softmax is essential here.
- ✓ It transforms raw outputs into probabilities for each class. It ensures the sum of these probabilities equals one, forming a true probability distribution.

- **Training the Model**
  - ❑ We employ stochastic gradient descent to train our model, aiming to minimize the cross-entropy loss.
  - ❑ During training, we compute accuracy at each iteration, revealing how the model's performance evolves over time.
  - ❑ This dynamic learning process is enlightening, showcasing how the model refines its weights to achieve higher accuracy.



Combining Code (PP#5)

- In this section, we'll explore a challenging coding task aimed at enhancing your expertise in deep learning.
- The main goal here is to get better at combining code from different sources to create a single, working codebase.
- **The Challenge: Combining Code**
  - The main challenge is all about merging code from two separate Jupyter notebooks that you've used in previous tutorials.
  - **Please write the code using two best codes you already solved**
  - Instead of working with just two groups of keywords, we're now aiming to modify the code to handle three groups: blue squares, black circles, and red triangles.
- **The Design and Setup**
  - For the design of the neural network, we've chosen a two-layer structure.
  - This decision is based on the fact that we're dealing with two-dimensional input data (X and Y coordinates) and three categories for classification.
  - The input layer has two features, and the hidden layer has four units. The output layer has three units, matching the three categories.



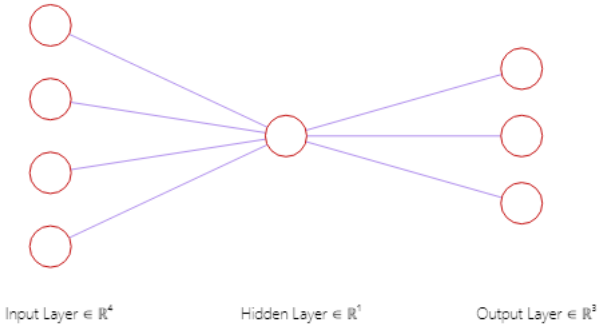
Combining Code

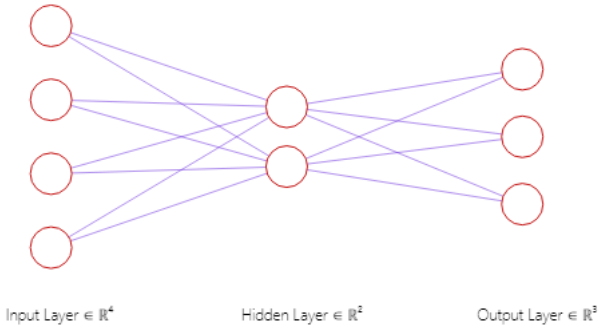
- **An Interesting Point**
  - ❑We included something called the softmax function in the output layer of the model.
  - ❑This function helps with classifying data into different categories.
  - ❑But here's the interesting part: we also have a similar function called the cross-entropy loss function in PyTorch that already does this job.
  - ❑So, we've kept it here to see if explicitly using softmax makes a difference in how well the model performs. It's something worth exploring.
- **In Conclusion**
  - This coding challenge is a chance for you to improve your deep learning skills by tackling a real-world problem of combining code from different sources. It's like solving a puzzle.

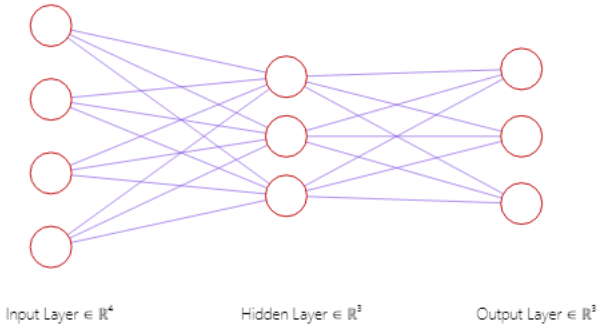
How do we ascertain the ideal number of units for our deep learning models?(PP#6)

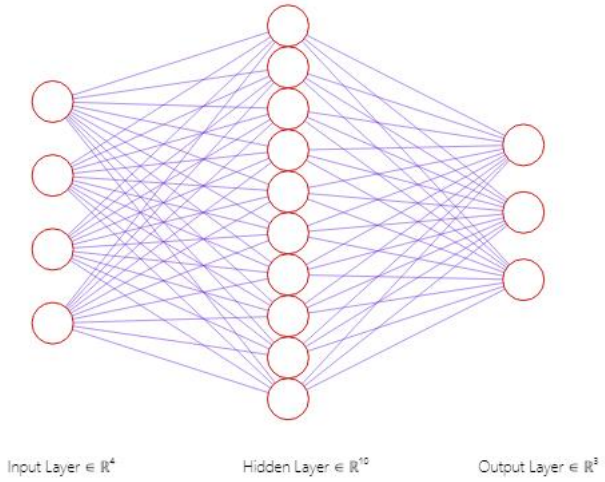
- In this section, we will delve into a critical aspect of deep learning: determining the appropriate number of units in hidden layers.
- This prompts the question: How do we ascertain the ideal number of units for our deep learning models?
- **The Challenge: Experimentation**
  - ❖ Unfortunately, there's no straightforward answer to this question.
  - ❖ Determining the optimal number of nodes in your network can be elusive. However, we can tackle this challenge through a systematic parametric experiment.
  - ❖ Our focus will remain on the iris dataset—a classic dataset used in machine learning.
  - ❖ Specifically, we aim to classify flowers using a model featuring a single hidden layer.
  - ❖ However, we will vary the number of units within that hidden layer from 1 to 128.
  - ❖ Our objective is to record the classification accuracy and visualize how it changes with the model's complexity.

Different number of hidden nodes







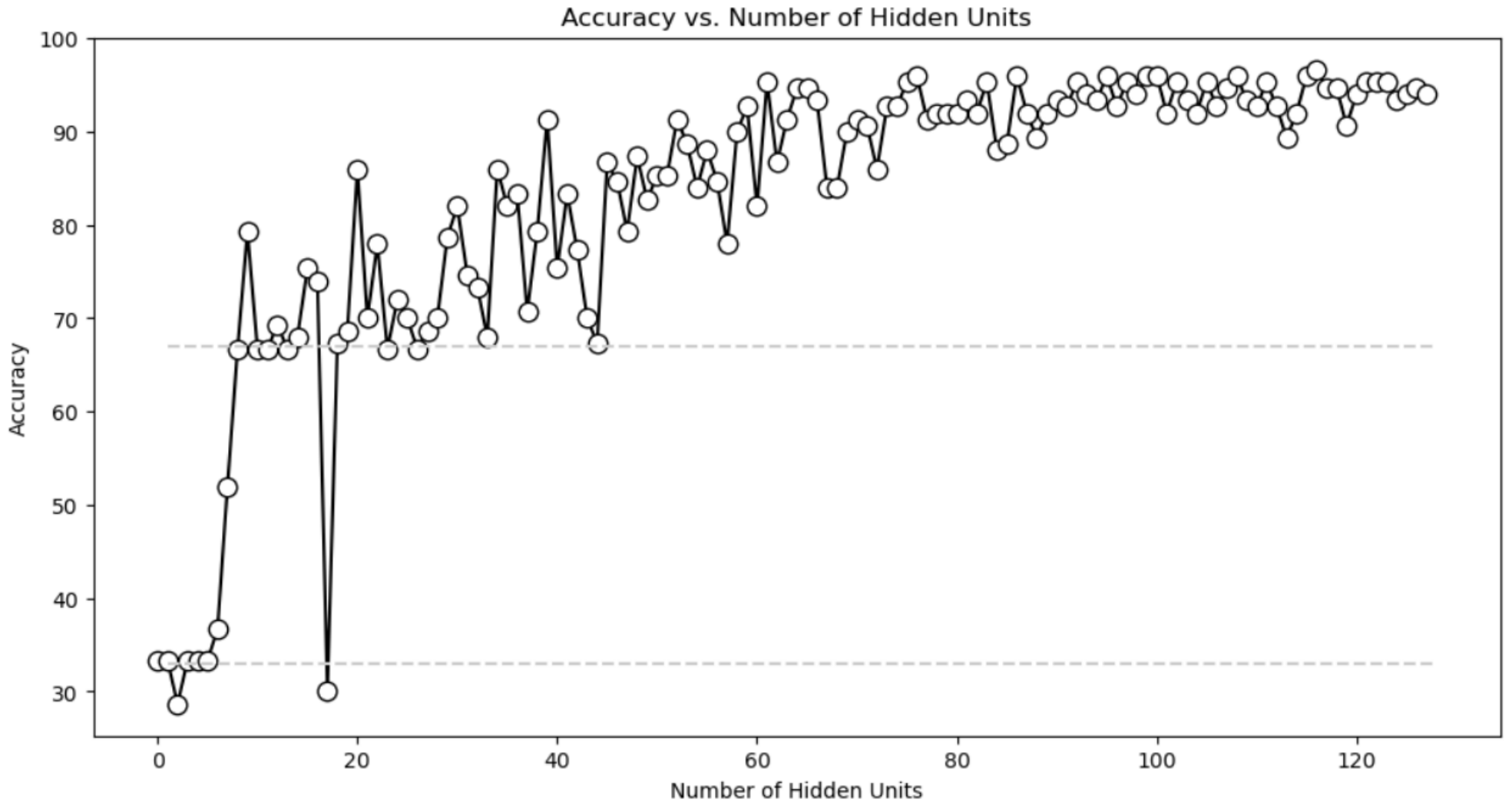


Hypothesize and Commit

- **Hypothesize and Commit**

  - ❑Before unveiling the results, I encourage you to form a hypothesis. What do you anticipate the accuracy curve will look like?
  - ❑Will it exhibit high accuracy throughout, or might it follow an inverted U-shape, indicating that having too few or too many hidden units could impede performance?
  - ❑Take a moment to commit to your hypothesis. Remember, embracing the possibility of being proven wrong is an integral part of the learning process.

The Experiment



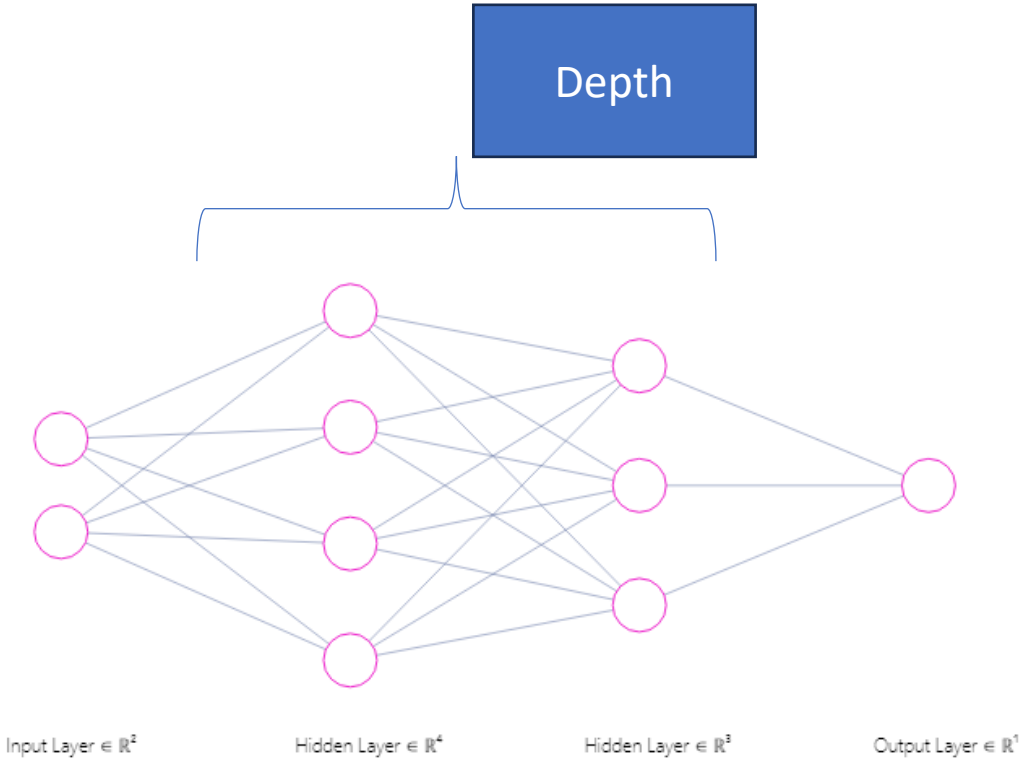
Dimensions of Deep Learning Complexity

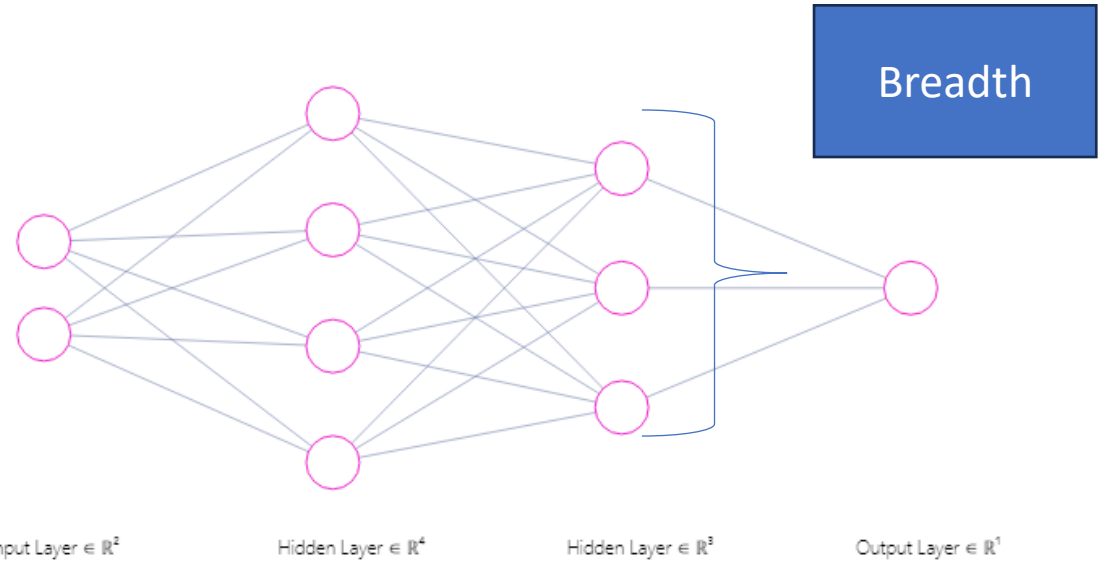
✓This section explores the multifaceted nature of complexity in deep learning models, with a particular focus on the depth versus breadth dimensions.

✓These dimensions wield significant influence over a model's architecture and its capacity to encapsulate abstract concepts.

- **Defining Breadth and Depth**
  - Deep learning models can be categorized along two dimensions: **breadth and depth**.
  - Breadth refers to the number of nodes in a layer, exemplified by the quantity of nodes within a given layer. Notably, breadth is not constrained to a uniform value across all layers; it can vary.
  - Conversely, depth signifies the number of layers, primarily hidden layers, bridging the input features and the model's output.
  - It is crucial to note that there are no strict thresholds designating models as deep or shallow; instead, they are deemed relatively deep or shallow based on the context.

Dimensions of Deep Learning Complexity





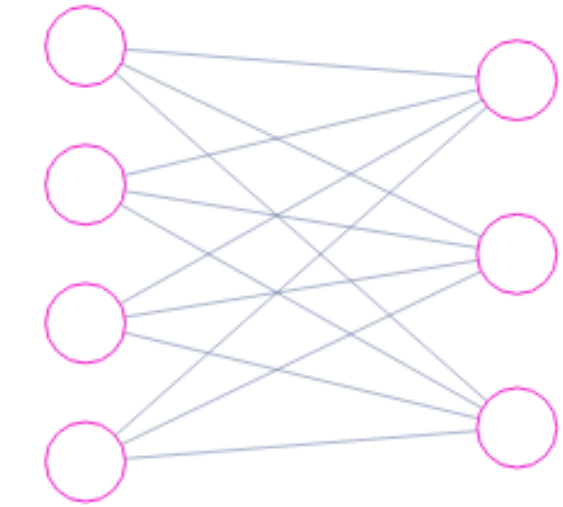
Dimensions of Deep Learning Complexity

✓This section explores the multifaceted nature of complexity in deep learning models, with a particular focus on the depth versus breadth dimensions.

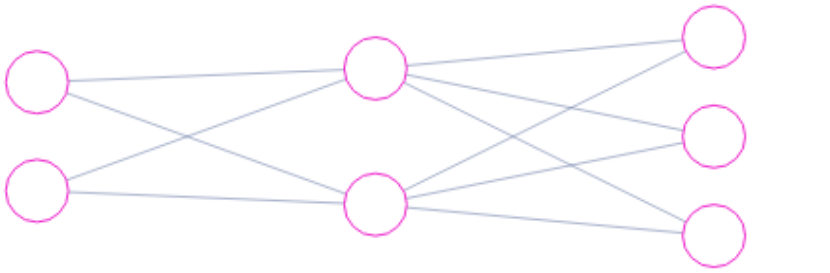
✓These dimensions wield significant influence over a model's architecture and its capacity to encapsulate abstract concepts.

- **Defining Breadth and Depth**
  - Deep learning models can be categorized along two dimensions: **breadth and depth**.
  - Breadth refers to the number of nodes in a layer, exemplified by the quantity of nodes within a given layer. Notably, breadth is not constrained to a uniform value across all layers; it can vary.
  - Conversely, depth signifies the number of layers, primarily hidden layers, bridging the input features and the model's output.
  - It is crucial to note that there are no strict thresholds designating models as deep or shallow; instead, they are deemed relatively deep or shallow based on the context.

Counting Parameters



Input Layer $\in \mathbb{R}^4$ Output Layer $\in \mathbb{R}^3$



Input Layer $\in \mathbb{R}^2$ Hidden Layer $\in \mathbb{R}^2$ Output Layer $\in \mathbb{R}^3$

The Impact on Parameters(PP#7)

- The central question that arises is: How do these dimensions affect the total number of model parameters? To comprehend this, we must delve into counting nodes and parameters.
- Consider two models, one broad and one deep. To count the nodes, we focus on the biases. Each node in the network is associated with precisely one bias parameter, simplifying node enumeration.
  - ❑In the wide model, we count a total of seven nodes and, consequently, seven biases. Moreover, there are 20 weights connecting nodes across layers, alongside 20 bias terms, summing up to 27 trainable parameters.
  - ❑Comparatively, the deep model also encompasses seven nodes, thus seven biases, but the structure differs in the weights' distribution. The total number of weights, including biases, amounts to 21.

Remarkable Insights

- **Summary Function**
  - ❑Introducing the summary function from the Torch Summary library, we obtain a comprehensive overview of the model.
  - ❑This function reveals the layers, components, and most importantly, the total number of parameters. While the models explored here are relatively small, in practical applications, deep learning models can burgeon into megabytes as they incorporate thousands, even millions, of parameters.
- This section equips you with an understanding of deep learning model complexity, offering a glimpse into parameter counting techniques and the tools available in PyTorch for model analysis.

Customizing Deep Learning Models

- In the realm of deep learning, we've primarily employed the user-friendly "**N.N.Sequential**" function.
- While this approach is simple and intuitive, it also possesses certain limitations in terms of flexibility and customization.
- As we delve deeper into the intricacies of deep learning, we'll encounter scenarios where the conventional "**N.N.Sequential**" falls short.
- To address these limitations and unlock a broader spectrum of possibilities, we must embark on a journey to construct our own classes for building deep learning models.

Initializing and Forwarding

- Initializing and Forwarding When designing our custom classes, we define two pivotal functions: **__init__** and **forward**.
- The **__init__** function is analogous to populating a story with characters. Here, we create the layers of our deep learning model.
- The **forward** function, on the other hand, is where all the action unfolds. It corresponds to the verbs in our story, representing the operations performed on the layers.
- Pros and Cons Choosing between the simplicity of "**N.N.Sequential**" and the versatility of custom classes boils down to your specific needs.
  - ❑ "**N.N.Sequential**" is swift, legible, and perfect for basic models.
  - ❑ However, custom classes provide unparalleled flexibility, letting you transcend the confines of standard architectures and explore innovative possibilities.
- With custom classes, you're only limited by your Python proficiency and imagination.

A Practical Example(PP#8)

- To illustrate the power of custom classes, we've incorporated them into a familiar problem: classifying queries.
- As we progress through this course, you'll encounter numerous scenarios where defining your own classes becomes indispensable.

How do we see the test of different layers and nodes??

??????

Exploration of the Impact of Model Depth and Breadth in Deep Learning

- In this informative lecture, the investigation of crucial concepts in deep learning is initiated.
- Commencement is made by discussing the advantages associated with the definition of custom classes to facilitate the development of flexible deep learning models.
- It becomes evident that a mere increase in the depth of a deep learning network is not always deemed beneficial.
- Model interpretability is also explored, with an emphasis on the notion that a model's performance is not solely dependent on the number of parameters but also on its architecture.
- **Depth vs. Breadth**: The distinction between depth and breadth in a model is elucidated..

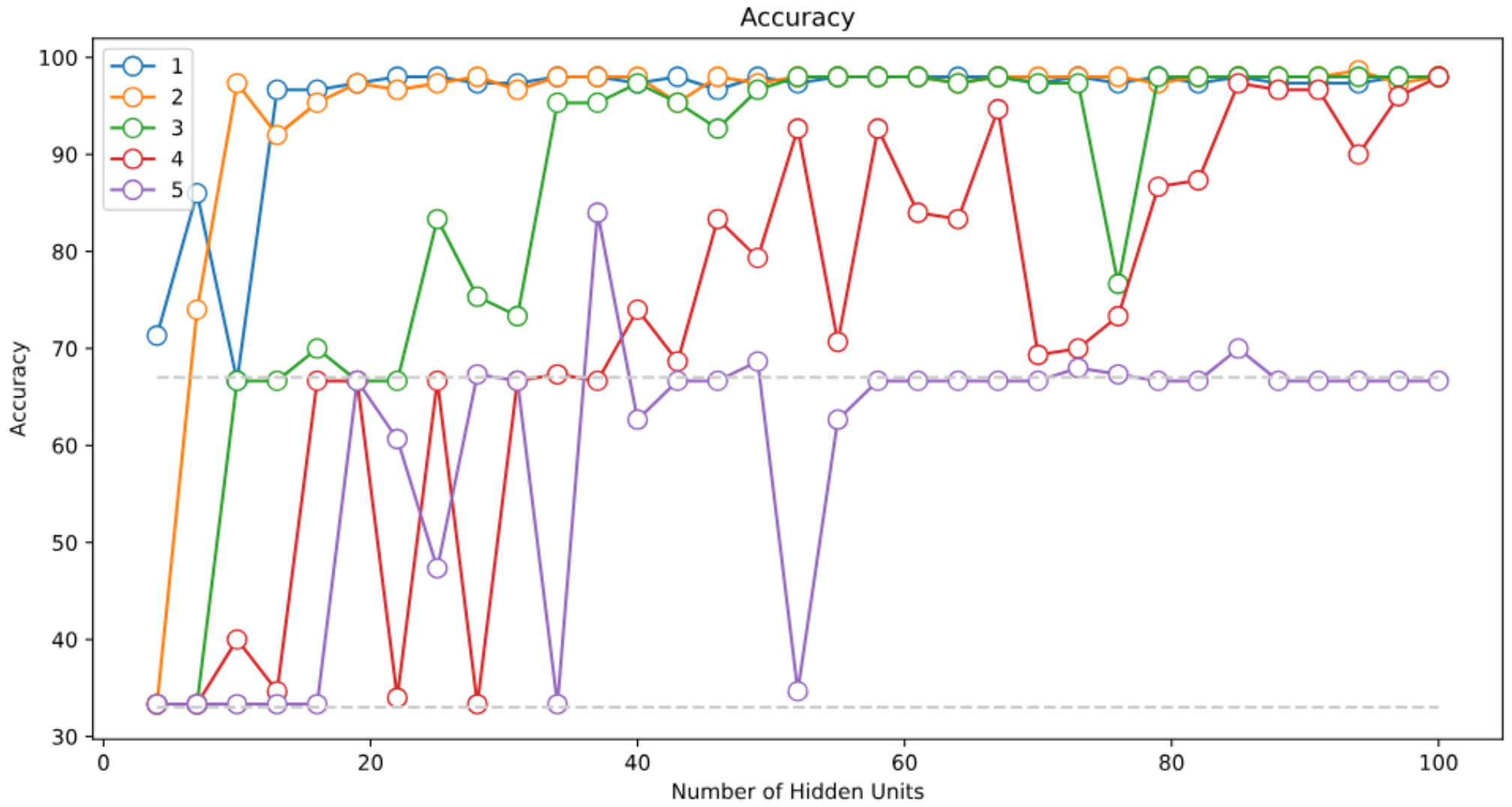
Python Implementation(PP#9)

- Python Implementation: A step-by-step Python implementation of our deep learning model is provided.
- PyTorch is utilized to create a class that defines the model architecture. This approach's flexibility allows for experimentation with different models by varying the number of layers and units per layer.
- The Experiment: A comprehensive experiment is conducted by varying the number of hidden layers and units per layer.
- The results are visually represented, showcasing the impact of different architectural choices on model performance.
- Surprisingly, it is observed that deeper models are not consistently superior, and a careful balance is necessitated.

Analysis of Parameters

- An analysis is conducted to examine the correlation between the total number of trainable parameters and model accuracy.
  - ❑ The results indicate that there is no straightforward relationship between these factors.
- Model architecture, along with other hyperparameters, plays a pivotal role in determining performance.
- Deeper models are not necessarily deemed superior; their performance is contingent upon the specific problem.
- The flexibility inherent in the creation of custom classes for model development is underscored.
- Model interpretability and architecture are identified as vital factors influencing performance.
  - ❑ Shallow models exhibit quicker learning, while deeper models possess the capacity to capture more intricate data representations.

Analysis of Parameters



Implementation of Custom Deep Learning Model Classes (PP#10)

- In this chapter, we are discussing the practice of converting existing deep learning models that are initially structured using the sequential class into custom model classes.
- This exercise not only reinforces your understanding of class-based model design but also showcases the equivalency between sequential-based and custom class-based model architectures.
- **Model Refactoring Using Custom Classes**
  - ❖ The primary objective of this coding challenge is to augment your proficiency in the conversion of sequential models into custom classes.
  - ❖ You have previously encountered numerous examples that leverage the sequential approach for model design.
- In this challenge, the emphasis is on replicating the existing model's architecture using a custom class structure, thus replacing the use of **'nn.Sequential**.'

Instructions for the Code Challenge

1. Begin by creating a copy of the notebook titled "multi-layer" to ensure the integrity of the original file.
2. Reconstruct the model, preserving its architecture precisely as it is. The key difference is the utilization of your own class instead of **'nn.Sequential**.'
3. Execute the remainder of the code to verify that the model functions correctly.