Struggling with Deep Learning Frameworks? A Good Curriculum Is Likely What You Need

Taha Rostami, 2024

Navigating the complexities of deep learning frameworks can be daunting due to the vast array of resources and the diverse backgrounds of learners. This manuscript addresses these challenges by offering a structured curriculum tailored specifically for "Gray Box" learners—those with some foundational knowledge of deep learning who seek to enhance their proficiency with frameworks like PyTorch. The curriculum emphasizes practical, hands-on learning, focusing on key areas such as sequence data processing, LSTM implementation, and Transformer models. By following the outlined path, learners can efficiently build their skills in deep learning frameworks, with an estimated completion time of 2 to 4 weeks. The tutorial also includes annotated resources and supplementary code to aid in mastering these frameworks, ensuring a more targeted and effective learning experience.

Keywords: Deep Learning Frameworks, Curriculum Design, Tutorial, PyTorch

1. Challenges in Learning Deep Learning Frameworks

Despite significant advancements in deep learning frameworks and the growing number of tutorials available, learning to work with these frameworks remains challenging for a large majority of people. This observation is based on my experience over the past few years, during which I have provided consultation to students and developers from various backgrounds on their machine learning projects.

I believe this challenge is strongly linked to several factors:

1. **Diverse Backgrounds:** People from a wide variety of fields and professions are now attempting to learn deep learning.
2. **Different Goals:** Individuals have different objectives when learning deep learning, ranging from academic research to practical application development.
3. **Misconception of Learning:** Many people mistakenly believe that learning deep learning and mastering deep learning frameworks are the same or should be learned simultaneously. Deep learning is a mathematically complex field, and understanding its fundamentals is inherently challenging and time-consuming, even for those with a strong mathematical background. As a result, some learners face a paradox: they feel the need to deeply

understand deep learning to effectively use its frameworks, yet they are under pressure to quickly learn how to work with these frameworks.

4. **Inadequate Tutorials:** There is an overwhelming number of tutorials available, but many are not aligned with the specific needs of the learner.

Assuming the audience has basic to intermediate Python programming experience, I categorize learners into three groups based on their approach to deep learning frameworks:

1. **White Box Learners:** These individuals have a solid mathematical understanding of deep learning but lack experience with frameworks. For example, some Electrical Engineering students fall into this category. They usually aim to implement fine-grained ideas and often work as researchers focusing on deep learning fundamentals.
2. **Black Box Learners:** These individuals may not have a strong mathematical background, or they are not interested in the underlying mechanisms of deep learning. Instead, they want to use these frameworks as tools, often to develop AI-powered applications or integrate AI into their workflows. Many developers in the industry belong to this group.
3. **Gray Box Learners:** These individuals have some background in deep learning, enough to use it effectively but not to develop new fundamental theories. This group includes many engineering students and developers who need to maximize the efficiency and flexibility of deep learning frameworks.

2. Focus on Gray Box Learners

The White Box learners often have access to high-quality resources and guidance, given their environment. It's like doubting whether a rally champion can drive a new car—it's not a significant concern.

On the other hand, Black Box learners benefit from the standardization of deep learning architectures, like Transformers, and the development of higher-level frameworks such as LangChain. For them, learning to work with deep learning has been somewhat reduced to learning how to use Python libraries and web APIs, which are relatively easy to master.

My primary interest lies with Gray Box learners who seek flexibility in using deep learning frameworks. I further divide them into two subgroups: Light Gray and Dark Gray learners.

- **Dark Gray Learners:** These individuals have a well-rounded understanding of various deep learning components, including gradient descent, parameter optimization, different types of data and tasks (e.g., time series, graphs, tabular data, text), model architectures, and learning paradigms (e.g., conventional deep learning, reinforcement learning).
- **Light Gray Learners:** These individuals are not equally interested in all aspects of deep learning. They prefer to abstract away certain components and focus on areas of particular interest. For example, one might treat gradient descent and parameter optimization as a

black box and concentrate on model architecture, while another might delve into the details of optimization algorithms but prefer to work with a fixed model architecture.

3. Special Focus on Light Gray Box Learners

Light Gray Box learners must answer certain questions and resolve some inner conflicts before proceeding to save time and energy. Otherwise, this could increase the time they need to learn, as they may resort to trial and error if they don't address specific issues upfront.

3.1. Which Framework Should You Learn?

There are a few well-recognized deep learning frameworks like PyTorch and TensorFlow. Nowadays, there are also extensions to these frameworks that make coding for common scenarios easier, such as PyTorch Lightning. A common pattern I've observed is that students often don't commit to a single framework. Here's what typically happens: they start with one framework, say PyTorch, but after learning some concepts, they struggle with something specific they need for their current work. They then switch to another framework, like TensorFlow, hoping it will be easier or better documented for that particular task. They might learn that thing in TensorFlow or Keras, but after some time, they face a situation where they can't learn comfortably in TensorFlow either. So, they switch back to PyTorch, and this loop continues. In the end, they learn bits and pieces from different frameworks but aren't confident or able to work with any of them comfortably or with the flexibility they initially wanted.

So, my first and foremost advice is to fix a framework you want to learn. The next natural question is which framework to choose. I know that both TensorFlow and PyTorch are quite similar nowadays, and each has its strengths. But I can say for sure that, generally, it doesn't matter much which one you choose. However, if I had to recommend, I'd suggest PyTorch. The reason is simple: most advanced models are developed in PyTorch (for example, see the support for models on Hugging Face). This gives you more freedom to work with advanced models without compatibility issues and allows you to learn from their source code.

3.2. One Tutorial vs. Multiple Tutorials

There are many available videos, books, blogs, and other resources that teach deep learning frameworks. Is there any single tutorial or reference that you can choose to learn everything you need? In principle, yes, but in practice, I don't think so. This is a bitter truth because learning from multiple resources can be overwhelming. However, it is the reality we have to deal with.

No matter if you attend online courses from Stanford, MIT, Microsoft, Google, Coursera, Udacity, or some YouTube channels, the issue lies not in the quality or comprehensiveness of the tutorial but in the type of tutorial. When learning to work with a deep learning framework, it's often the case that no matter how much effort the teacher puts into making things comprehensive and clear, you will still have questions and ambiguities that arise as you learn. If you can't ask these questions directly to the instructor or to other students who are taking or have taken the same course, these

unanswered questions can accumulate. By the end of the course, you may feel like you didn't learn certain things you wanted to.

If you can attend interactive courses, then using a single tutorial might be sufficient. However, my focus here is on those who want or need to learn on their own, offline, and preferably through free materials. For this group, I can say from experience that it's very unlikely you'll be able to learn everything you want from just one tutorial. This is because, in this case, you have more limited resources to ask your questions (although tools like ChatGPT have alleviated this challenge somewhat). Additionally, there are certain concepts that one tutorial may explain much better than another.

In conclusion, you will most likely need to use multiple resources to learn what you want. However, this doesn't mean you need to go through each resource entirely. With a well-planned curriculum, you should be able to focus on different parts of various resources and even within each resource, you'll be able to concentrate on some sections while skipping or skimming others. I will discuss a specific curriculum later.

3.3. Learn Then Practice vs. Practice Alongside Learning

I've noticed a common pattern among those who attempt to learn deep learning frameworks through non-interactive materials: they often read or watch multiple sections without implementing what they've learned. This approach is generally ineffective, especially if you plan to follow the curriculum I'll describe later.

This behavior often stems from previous unsuccessful attempts to master deep learning frameworks. When people repeatedly fail to achieve their learning goals, their confidence and expectations diminish. Coupled with the anxiety of needing to learn quickly due to time constraints, they may rush through materials without practicing, hoping to absorb knowledge passively. Unfortunately, this approach rarely leads to true understanding or proficiency.

To counter this, I strongly recommend that after reading or watching a section, you immediately implement what you've learned. At a minimum, you should code the primary concepts and architectures covered in the material. This practice not only solidifies your understanding but also builds the practical skills needed to work effectively with deep learning frameworks.

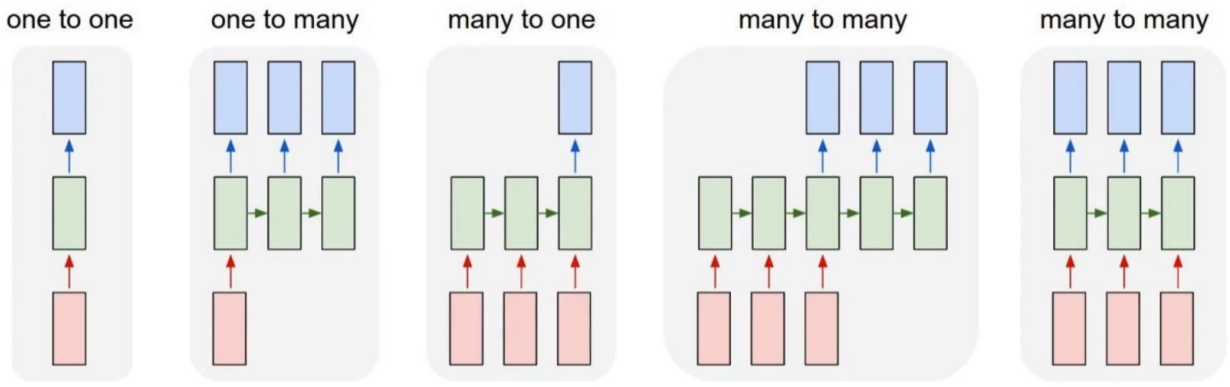
In the curriculum I'll introduce later, I'll highlight the essential implementations you should focus on to ensure you're well-equipped to use your chosen framework. Implementing these key elements as you progress will greatly enhance your learning experience and ensure you gain the necessary hands-on expertise.

4. Key Characteristics of a Good Curriculum for Learning Deep Learning Frameworks

When I first started learning deep learning frameworks, nearly every tutorial I encountered followed this exact curriculum:

1. Review the basics of neural networks.
2. Train a feedforward network.
3. Learn to work with CNNs.
4. Move on to advanced architectures like GANs, and so on.

If the tutorial covered working with sequence data, it would typically use time series prediction as the example and, of course, reference the usual "bible" of sequence data.



Honestly, I think this approach is terrible. The reason I say this is that such a curriculum violates the key principles that a good curriculum for learning deep learning frameworks should have. For instance, this particular curriculum assumes all learners are equally interested in every aspect of deep learning. It wastes a lot of time on CNNs before even getting to more advanced topics that some learners might actually need. For example, if a student in medical engineering is already working on signal processing data, how relevant is this curriculum for them? Is CNN really what they need to learn first? And why should GANs be their next focus?

So, how do we address the issues with this curriculum—and others like it? I have a proposal that, while it hasn't been tested on others yet, has worked well for me. Before sharing my proposed curriculum, let me outline the key principles or characteristics that I believe a good curriculum should have, but which many existing references often lack. [You can interpret this section as {the all characteristics my proposed curriculum aims to incorporate} - { all the characteristics commonly found in well-recognized curricula}.]

Minimizing Unnecessary Background

A good curriculum should have one key property: it should provide just enough background—neither too much nor too little—for the learner to grasp what they need to learn. Given the nature of AI-powered research and applications, students, developers, and other learners often need to get up to speed with deep learning frameworks in a relatively short amount of time. Therefore, a good curriculum should be as minimal as possible, including only the information that's truly necessary to help the learner achieve their goals.

Generalization

Another important property of a good curriculum—especially for deep learning frameworks—is that it should be as general as possible. People from diverse backgrounds and with different goals want to learn these frameworks, and overly specific content can be a hurdle.

For example, generalization should be applied not only to deep learning concepts (e.g., CNNs might not be relevant for everyone and could be deferred to later) but also to coding examples. Take PyTorch's documentation as an example: while it offers excellent code recipes, it often includes ad-hoc code, like specific preprocessing steps, using specific datasets, or loading them in an ad-hoc manner, that aren't necessary for understanding the overall workflow. For instance, in its NLP documentation, PyTorch uses libraries like torchtext and spaCy and includes steps like converting Unicode to ASCII. These details might not be relevant for people outside of NLP and can cause confusion.

Of course, PyTorch's documentation is designed for those interested in NLP and is more of a tutorial than a curriculum. However, imagine if it were intended as a general reference for working with sequence data. The diverse backgrounds of learners might make niche topics like Unicode-to-ASCII conversion a confusing distraction. This highlights why generalization in a curriculum is crucial—it should focus on core concepts that are broadly applicable, rather than getting bogged down in specific details that may not be relevant to everyone.

5. Proposed Curriculum

The proposed curriculum is structured as follows:

1. Review the basics of neural networks.
2. Train a feedforward network.
3. Learn to work with sequence data.
4. Advance with Specific Interests.

The curriculum focuses on a streamlined approach by omitting less relevant topics such as Convolutional Neural Networks (CNNs) and Generative Adversarial Networks (GANs). After covering the basics and training simple feedforward neural networks, the emphasis shifts to sequence data. Sequence data presents unique challenges and opportunities:

1. **Historical Significance**: Numerous revolutionary architectures, such as LSTM, Seq2Seq, and Transformers, have been developed to enhance the accuracy of sequence data processing.
2. **Complex Dependencies**: The relationships within sequence data are often more complex than those in image data, which primarily concerns locality and rotation invariance. This complexity is comparable to that seen in graph data and Graph Neural Networks.

In practice, high-quality tutorials on general sequence data can be scarce. Common tutorials typically focus on:

1. Manually creating tensor datasets.
2. Time series data.
3. Natural Language Processing (NLP) perspectives.

I recommend focusing on NLP because algorithms such as Word2Vec offer valuable insights into data preprocessing, which can be applied to other scenarios like time series. Additionally, NLP provides historical context for state-of-the-art models, such as Transformers.

Revised Curriculum:

1. Review the Basics of Neural Networks
2. Train a Feedforward Network
3. Learn to Work with Sequence Data Using NLP:
    - Word2Vec (including implementation)
    - LSTM (including implementation)
    - Seq2Seq (with and without Attention, including implementation)
    - Transformers (including implementation)
    - Pretraining (including implementation using Hugging face and Lang chain)
4. Advance with Specific Interests

For learners with different levels of familiarity with neural networks, the curriculum allows flexibility. Some learners should thoroughly review fundamental concepts such as gradient descent and backpropagation. Some learners might focus on understanding that neural networks involve parameters and algorithms for optimization without delving into specific details.

When training a simple feedforward neural network, ensure familiarity with basic components of your chosen deep learning framework. For example, if using PyTorch, you should understand tensors, autograd, computational graphs, datasets, data loaders, nn.Module, and the typical training/evaluation loop.

Next, explore the main architectures and important NLP tasks such as machine translation, Named Entity Recognition (NER), and classification. You may choose to skip specific NLP evaluation metrics if NLP is not your primary interest. Depending on your depth of study, you might also consider algorithms like CBOW and GloVe.

Implement each learned architecture in PyTorch before moving on to the next topic. Avoid implementing models out of order or postponing practical exercises until after completing all theoretical material.

Finally, apply your knowledge to areas of specific interest, whether it be graphs, text, audio, video, images, tabular data, or particular tasks o particular paradigms.

6. Proposed Curriculum's Concretization

There are countless resources available for learning deep learning frameworks, which can make it challenging to choose the right ones. Based on my research experience, I've explored both popular and lesser-known courses. In this curriculum, I'll annotate specific resources to provide a clear path for those who want to follow it. Additionally, I've written some code, particularly with detailed comments and explanations, especially for LSTM classification. While I won't heavily reference my own code, you may find it helpful as complementary material. By following the curriculum as outlined, the estimated learning time is between 2 to 4 weeks.

1. Review the Basics of Neural Networks & Train a Feedforward Network

The basics of neural networks and training a feedforward network are foundational concepts, and there are many good tutorials available. I won't suggest a specific resource here; instead, I recommend picking any reputable tutorial to learn these basics without overthinking it.

For specific components of PyTorch, such as tensors, autograd, datasets, and dataloaders, PyTorch's official documentation is an excellent resource. See PyTorch Documentation for detailed tutorials.

2. Learn to Work with Sequence Data Using NLP

Word2Vec

- Start with the first four videos from CS224n: Natural Language Processing with Deep Learning .
- If you need further clarification on implementation, watch the first video from NLP Lecture Series , with accompanying code available on GitHub.
- For a simplified implementation in PyTorch, refer to this GitHub repository . The goal is to understand distributed representation and how it prepares data, as well as learning about PyTorch's Embedding Layers. If you're primarily interested in NLP, you may also explore related topics like GloVe.

LSTM

- Watch lectures 5 and 6 from CS224n and try using LSTM for classification, regression, or Named Entity Recognition (NER).
- For implementation, refer to this PyTorch tutorial and Deep Learning for NLP .
- If you struggle with data preparation for LSTM, consider watching video 48 from this NLP Playlist .

Seq2Seq

- See lecture 7 from CS224n and implement a machine translation model, first without attention and then with attention.

- For implementation, refer to this PyTorch tutorial , or the code from the NLP Lecture Series [3].

Transformers

- Watch lecture 8 from the NLP Lecture Series . This lecture is dense with information, so take your time to fully understand it.
- If you are a dark gray or white box learner, consider implementing the main components of a transformer from scratch. For this, you can refer to resources that do this, such as the code from the NLP Lecture Series .
- If you are a light gray learner, try using nn.Transformer available in PyTorch. See this tutorial .

Pretraining

- Watch lecture 9 from CS224n . You can also explore videos related to Hugging Face and LangChain available from the NLP Playlist or NLP Lecture Series .

3. Advance with Specific Interests

At this stage, you should be ready to delve into any specific area of applied deep learning or frameworks that interest you. Select a topic relevant to your work and explore it in depth using the skills and knowledge you've acquired.