EXPAND YOUR MACHINE LEARNING KNOWLEDGE
WITH 30 QUESTIONS AND ANSWERS

MACHINE LEARNING Q and AI

Sebastian Raschka, PhD

Machine Learning Q and AI

Expand Your Machine Learning & AI Knowledge With 30 In-Depth Questions and Answers

Sebastian Raschka, PhD

This book is for sale at
http://leanpub.com/machine-learning-q-and-ai

This version was published on 2023-06-11

This book is dedicated to those who tirelessly contribute to advancing the field of machine learning through research and development. Your passion for discovery and innovation and your commitment to sharing knowledge and resources through the open-source community is an inspiration to us all.

Contents

Preface

Over the years, I shared countless educational nuggets about machine learning and deep learning with hundreds of thousands of people. The positive feedback has been overwhelming, and I continue to receive requests for more. So, in this book, I want to indulge both your desire to learn and my passion for writing about machine learning[1].

[1] I will use *machine learning* as an umbrella term for machine learning, deep learning, and artificial intelligence.

Who Is This Book For?

This book is for people with a beginner or intermediate background in machine learning who want to learn something new. This book will expose you to new concepts and ideas if you are already familiar with machine learning. However, it is not a math or coding book. You won't need to solve any proofs or run any code while reading. In other words, this book is a perfect travel companion or something you can read on your favorite reading chair with your morning coffee.

What Will You Get Out of This Book?

Machine learning and AI are moving at a rapid pace. Researchers and practitioners are constantly struggling to keep up with the breadth of concepts and techniques. This book provides bite-sized nuggets for your journey from machine learning beginner to expert, covering topics from various machine learning areas. Even experienced machine learning researchers and practitioners will encounter something new that they can add to their arsenal of techniques.

How To Read This Book

The questions in this book are mainly independent, and you can read them in any desired order. You can also skip individual questions for the most part. However, I organized the questions to bring more structure to the book.

For instance, the first question deals with embeddings, which we refer to in later questions on self-supervised learning and few-shot learning. Therefore, I recommend reading the questions in sequence.

The book is structured into five main chapters to provide additional structure. However, many questions could appear in different chapters without affecting the flow.

Chapter 1, Deep Learning and Neural Networks covers questions about deep neural networks and deep learning that are not specific to a particular subdomain. For example, we discuss alternatives to supervised learning and techniques for reducing overfitting.

Chapter 2, Computer Vision focuses on topics mainly related to deep learning but are specific to computer vision, many of which cover convolutional neural networks and vision transformers.

Chapter 3, Natural Language Processing covers topics around working with text, many of which are related to transformer architectures and self-attention.

Chapter 4, Production, Real-World, And Deployment Scenarios contains questions pertaining to practical scenarios, such as increasing inference speeds and various types of distribution shifts.

Chapter 5, Predictive Performance and Model Evaluation dives a bit deeper into various aspects of squeezing out predictive performance, for example, changing the loss function, setting up k-fold cross-validation, and dealing with limited labeled data.

If you are not reading this book for entertainment but for machine

learning interview preparation, you may prefer a spoiler-free look at the questions to quiz yourself before reading the answers. In this case, you can find a list of all questions, without answers, in the appendix.

Sharing Feedback and Supporting This Book

I enjoy writing, and it is my pleasure to share this knowledge with you.

If you obtained a free copy and like this book, you can support me by buying a digital copy on Leanpub at https://leanpub.com/machine-learning-q-and-ai/[2].

For an author, there is nothing more valuable than your honest feedback. I would really appreciate hearing from you and appreciate any reviews! And, of course, I would be more than happy if you recommend this book to your friends and colleagues or share some nice words on your social channels.

Moreover, I value feedback of every type, both positive and negative. Positive feedback serves as motivation, while constructive criticism provides insights to learn and grow. If you have questions, please don't hesitate to reach out, preferably on the Discussions page on GitHub[3].

[2] https://leanpub.com/machine-learning-q-and-ai/
[3] https://github.com/rasbt/MachineLearning-QandAI-book/discussions

Acknowledgements

Writing a book is an enormous undertaking. This project would not have been possible without the help of the open source and machine learning communities who collectively created the technologies that this book is about. Moreover, I want to thank everyone who encouraged me to share my flashcard decks, as this book is an improved and polished version of these.

I also want to thank the following readers for helpful feedback on the manuscript:

- Anton Reshetnikov for suggesting a cleaner layout for the supervised learning flowchart in Q30.
- Juan M. Bello-Rivas for sharing various typographical errors.

About the Author

Sebastian Raschka is a machine learning and AI researcher with a strong passion for education. As Lead AI Educator at Lightning AI, he is excited about making AI and deep learning more accessible and teaching people how to utilize these technologies at scale.

Before dedicating his time fully to Lightning AI, Sebastian held a position as Assistant Professor of Statistics at the University of Wisconsin-Madison, where he specialized in researching deep learning and machine learning. You can find out more about his research on his website[4].

Moreover, Sebastian loves open-source software and has been a passionate contributor for over a decade. Next to coding, he also loves writing and authored the bestselling *Python Machine Learning* book and *Machine Learning with PyTorch and Scikit-Learn.*

If you like to find out more about Sebastian and what he is currently up to, please visit his personal website at https://sebastianraschka.com. You can also find Sebastian on Twitter (@rasbt)[5] and LinkedIn (sebastianraschka)[6].

[4] https://sebastianraschka.com/publications/
[5] https://twitter.com/rasbt
[6] https://www.linkedin.com/in/sebastianraschka

Copyright and Disclaimer

Credits

Cover image by ECrafts / stock.adobe.com.

Introduction

Thanks to rapid advancements in deep learning, we have seen a significant expansion of machine learning and AI in recent years.

On the one hand, this rapid progress is exciting if we expect these advancements to create new industries, transform existing ones, and improve the quality of life for people around the world.

On the other hand, the rapid emergence of new techniques can make it challenging to keep up, and keeping up can be a very time-consuming process. Nonetheless, staying current with the latest developments in AI and deep learning is essential for professionals and organizations that use these technologies.

With this in mind, I began writing this book in the summer of 2022 as a resource for readers and machine learning practitioners who want to advance their understanding and learn about useful techniques that I consider significant and relevant but often overlooked in traditional and introductory textbooks and classes.

I hope readers will find this book a valuable resource for obtaining new insights and discovering new techniques they can implement in their work.

Happy learning,
Sebastian

Chapter 1. Neural Networks and Deep Learning

Q1. Embeddings, Representations, and Latent Space

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q2. Self-Supervised Learning

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q3. Few-Shot Learning

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q4. The Lottery Ticket Hypothesis

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q5. Reducing Overfitting with Data

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q6. Reducing Overfitting with Model Modifications

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q7. Multi-GPU Training Paradigms

> Q:

What are the different multi-GPU training paradigms, and what are their respective advantages and disadvantages?

> A:

The multi-GPU training paradigms can be categorized into two groups: (1) dividing data for parallel processing with multiple GPUs and (2) dividing the model among multiple GPUs to handle memory constraints when the model size surpasses that of a single GPU.

Data parallelism falls into the first category, model parallelism and tensor parallelism fall into the second category, and techniques like pipeline parallelism borrow ideas from both categories. In addition, current software implementations such as DeepSpeed[7], Colossal AI[8], and others blend multiple approaches into a hybrid technique.

Model parallelism

Model parallelism (also referred to as *inter op parallelism*) is perhaps the most intuitive form of parallelization across devices. For example, suppose you have a simple neural network that only consists of 2 layers: a hidden layer and an output layer. Here, we keep one layer on one GPU and the other layer on another GPU. Of course, this can scale to an arbitrary number of layers and GPUs.

This is a good strategy for dealing with limited GPU memory where the complete network does not fit into one GPU. However, there are more efficient ways of using multiple GPUs because of the chain-like structure (layer 1 on GPU 1 $\rightarrow$ layer 2 on GPU 2 $\rightarrow$ ...).

A major disadvantage of model parallelism is that the GPUs have to wait for each other – they cannot efficiently work parallel as they depend on each other's outputs.

[7] https://github.com/microsoft/DeepSpeed
[8] https://github.com/hpcaitech/ColossalAI

Data parallelism

Data parallelism has been the default mode for multi-GPU training for several years. Here, we divide a minibatch into smaller microbatches. Then, each GPU processes a microbatch separately to compute the loss and loss gradients for the model weights. After the individual devices process the microbatches, the gradients are combined to compute the weight update for the next round.

An advantage of data parallelism over model parallelism is that the GPUs can run in parallel – each GPU processes a portion of the training minibatch, a microbatch. However, a caveat is that each GPU requires a full copy of the model. This is obviously not feasible if we have large models that don't fit into the GPU's VRAM.

Tensor parallelism

Tensor parallelism (also referred to as *intra op parallelism*) is a more efficient form of model parallelism (*inter op parallelism*). Here, the weight and activation matrices are spread across the devices instead of distributing whole layers across devices. Specifically, the individual matrices are split, so we split an individual matrix multiplication across GPUs.

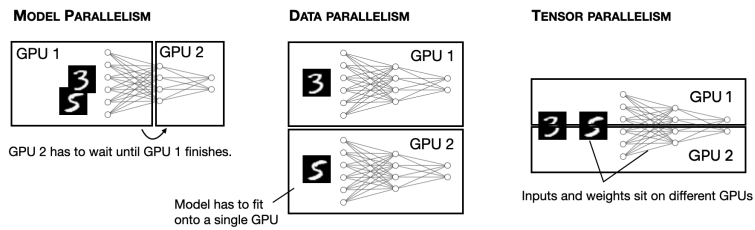

Figure 7.1. Comparison of model parallelism (left), data parallelism (center), and tensor parallelism (right). In model parallelism, we put different layers onto different GPUs to work around GPU memory limitations. In data parallelism, we split batch across GPUs to train copies of the model in parallel, averaging gradients for the weight update afterwards. In tensor parallelism, we split matrices (inputs and weights) across different GPU for parallel processing when models are too large to fit into GPU memory.

There are several ways we can implement tensor parallelism. For example, using basic principles of linear algebra, we can split a matrix multiplication across two GPUs in a row- or column-wise fashion, as illustrated in the figure below. (Note that this concept can be extended to an arbitrary number of GPUs.)

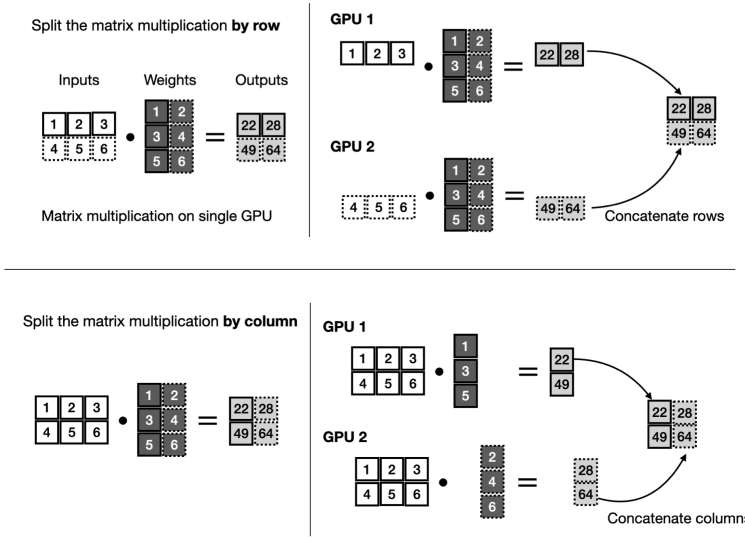

Figure 7.2. Illustration how we can distribute matrix multiplication across different devices. For simplicity, the figure depicts two GPUs, but the concepts extends to an arbitrary number of devices.

An advantage of tensor parallelism is that we can work around memory limitations similar to model parallelism. And at the same time, we can also execute operations in parallel, similar to data parallelism.

A small weakness of tensor parallelism is that it can result in high communication overhead between the multiple GPUs across which the matrices are split or sharded. For instance, tensor parallelism requires frequent synchronization of the model parameters across the devices, which can slow down the overall training process.

Pipeline parallelism

Pipeline parallelism can be seen as a form of model parallelism that tries to minimize the sequential-computation bottleneck. In that sense, we can think of pipeline parallelism as a form of model parallelism that enhances the parallelism between the individual layers sitting on different devices. However, note that it also borrows ideas from data parallelism, such as splitting minibatches further into microbatches.

How does it work? During the forward pass, the activations are passed similar to model parallelism, however, the twist is that the gradients of the input tensor are passed backwards to prevent the devices from being idle. In a sense, pipeline parallelism is a sophisticated hybrid between data and model parallelism, which is described in more detail in the GPipe paper[9] or DeepSpeed pipeline parallelism tutorial[10].

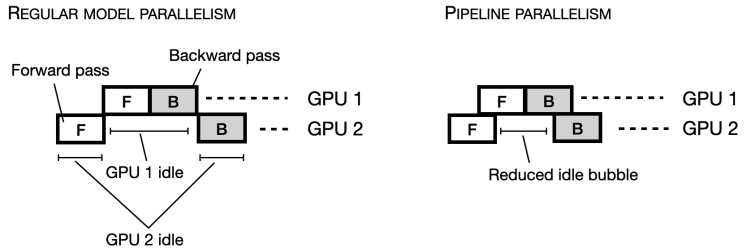

Figure 7.3. A conceptual illustration of pipeline parallelism, which aims to reduce the idle time of GPUs compared to regular model parallelism.

A disadvantage of pipeline parallelism is that it may require significant effort to design and implement the pipeline stages and associated communication patterns. Additionally, the performance gains from pipeline parallelism may not be as substantial as those from other parallelization techniques, such as pure data parallelism, especially for small models or in cases where the communication

[9]Huang, Cheng, Bapna, Firat, Chen, Chen, Lee, Ngiam, Le, Wu, and Chen (2018). *GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism*, https://arxiv.org/abs/1811.06965.

[10]Pipeline Parallelism: https://www.deepspeed.ai/tutorials/pipeline/

overhead is high.

Pipeline parallelism is definitely an improvement over model parallelism, even though it is not perfect, and there will be idle bubbles. However, for modern architectures that are too large to fit into GPU memory, it is nowadays more common to use a blend of data parallelism and tensor parallelism (as opposed to model parallelism) techniques.

Sequence parallelism

Sequence parallelism is a new concept developed for transformer models[11]. One shortcoming of transformers is that the self-attention mechanism (the original scaled-dot product attention) scales quadratically with the input sequence length. There are, of course, more efficient alternatives to the original attention mechanism that scales linearly[12][13]; however, they are less popular, and most people prefer the original scaled-dot product attention mechanism.

Sequence parallelism splits the input sequence into smaller chunks that can be distributed across GPUs to work around memory limitations as illustrated in the figure below.

[11] Li, Xue, Baranwal, Li, and You (2021). *Sequence Parallelism: Long Sequence Training from [a] System[s] Perspective*, https://arxiv.org/abs/2105.13120.

[12] Tay, Dehghani, Bahri, and Metzler (2020). *Efficient Transformers: A Survey*, https://arxiv.org/abs/2009.06732.

[13] Zhuang, Liu, Pan, He, Weng, and Shen (2023). *A Survey on Efficient Training of Transformers*, https://arxiv.org/abs/2302.01107.

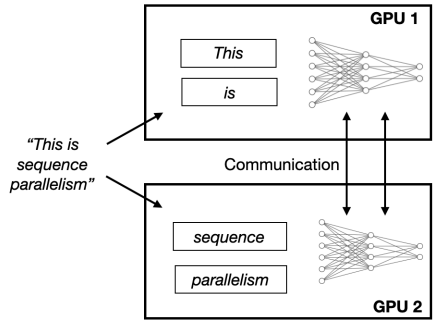

Figure 7.4. A conceptual illustration of sequence parallelism, which aims to reduce computation memory constraints of self-attention mechanisms.

Sequence parallelism is less well-studied than other parallelization techniques. However, conceptually, it would have similar advantages and disadvantages as tensor parallelism. It enables us to train larger models when memory is a constraint due to the sequence length; however, it also introduces additional communication overheads. On the other hand, it also shares shortcoming of data parallelism where we have to duplicate the model and make sure it fits into the device memory.

Another disadvantage of sequence parallelism (depending on the implementation) for multi-GPU training of transformers is that breaking up the input sequence into smaller subsequences can decrease the model's accuracy – mainly when the model is applied to longer sequences.

Which techniques should we use in practice?

Practical recommendations depend on the context. If we train small models that fit onto a single GPU, then data parallelism strategies may be the most efficient. Performance gains from pipeline parallelism may not be as significant as those from other parallelization techniques such as data parallelism, especially for small models or in cases where the communication overhead is high.

If models are too large to fit into the memory of a single GPU, we

need to explore model or tensor parallelism. Tensor parallelism is naturally more efficient since the GPUs can work in parallel – there is no sequential dependency as in model parallelism.

Modern multi-GPU strategies typically combine data parallelism and tensor parallelism (popular examples include DeepSpeed stages 2 and 3[14]).

> Reader quiz:

7-A

Suppose we are implementing our own version of tensor parallelism, which works great when we train our model with an SGD (standard stochastic gradient descent) optimizer.
However, when we try the Adam[15] optimizer, we encounter an out-of-memory device. What could be a potential problem explaining this issue?

7-B

Suppose we don't have access to a GPU and are considering using data parallelism on the CPU. Is this a good idea?

[14] https://www.deepspeed.ai/tutorials/zero/

[15] Kingma and Ba (2014). *Adam: A Method for Stochastic Optimization*, https://arxiv.org/abs/1412.6980.

Q8. The Keys to Success of Transformers

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q9. Generative AI Models

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q10. Sources of Randomness

Chapter 2. Computer Vision

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q11. Calculating the Number of Parameters

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q12. The Equivalence of Fully Connected and Convolutional Layers

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q13. Large Training Sets for Vision Transformers

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Chapter 3. Natural Language Processing

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q14. The Distributional Hypothesis

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q15. Data Augmentation for Text

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q16. "Self"-Attention

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q17. Encoder- And Decoder-Style Transformers

> Q:

What are the differences between encoder- and decoder-based language transformers?

> A:

Fundamentally, both encoder- and decoder-style architectures use the same self-attention layers to encode word tokens. However, the main difference is that encoders are designed to learn embeddings that can be used for various predictive modeling tasks such as classification. In contrast, decoders are designed to e new texts, for example, answering user queries.

The original transformer

The original transformer architecture[16], which was developed for English-to-French and English-to-German language translation, utilized both an encoder and a decoder, as illustrated in the figure below.

[16]Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, and Polosukhin (2017). *Attention Is All You Need*, https://arxiv.org/abs/1706.03762

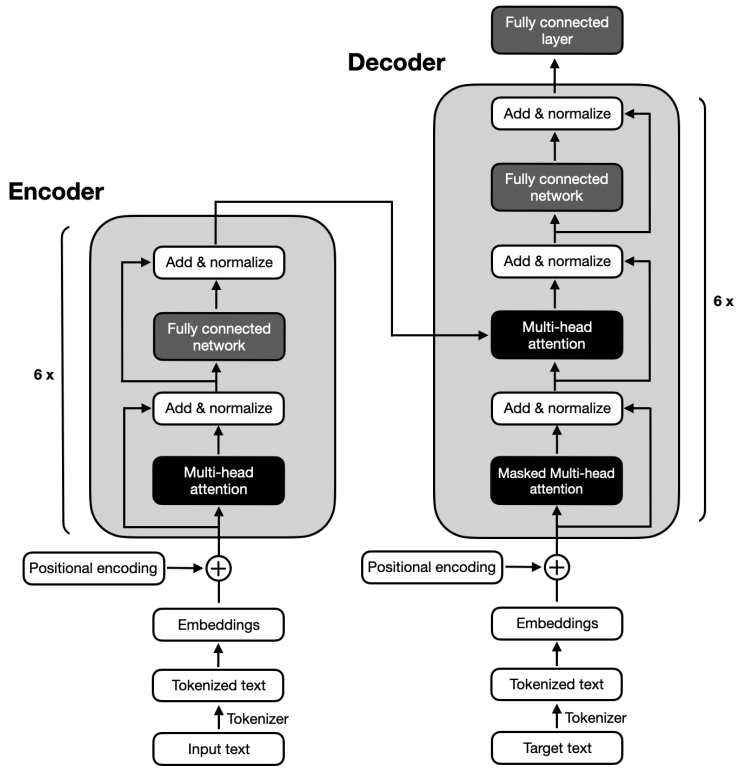

Figure 17.1. Illustration of the original transformer architecture.

In the figure above, the input text (that is, the sentences of the text that is to be translated) is first tokenized into individual word tokens, which are then encoded via an embedding layer[17] before it enters the decoder part. Then, after adding a positional encoding vector to each embedded word, the embeddings go through a multi-head self-attention layer. The multi-head attention layer is followed by an "Add & normalize" step, which performs a layer normalization and adds the original embeddings via a skip connection (also known as a residual or shortcut connection). Finally, after entering a "fully connected layer," which is a small multilayer perceptron

[17] See Q1 for more details about embeddings.

consisting of two fully connected layers with a nonlinear activation function in between, the outputs are again added and normalized before they are passed to a multi-head self-attention layer of the decoder part.

The decoder part in the figure above has a similar overall structure as the encoder part. The key difference is that the inputs and outputs are different. The encoder receives the input text that is to be translated, and the decoder generates the translated text.

Encoders

The encoder part in the original transformer, illustrated in the preceding figure, is responsible for understanding and extracting the relevant information from the input text. It then outputs a continuous representation (embedding) of the input text that is passed to the decoder. Finally, the decoder generates the translated text (target language) based on the continuous representation received from the encoder.

Over the years, various encoder-only architectures have been developed based on the encoder module of the original transformer model outlined above. Notable examples include BERT[18] and RoBERTa[19].

BERT (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers) is an encoder-only architecture based on the Transformer's encoder module. The BERT model is pretrained on a large text corpus using masked language modeling (illustrated in the figure below) and next-sentence prediction tasks.

[18] *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* (2018). Devlin, Chang, Lee, and Toutanova, https://arxiv.org/abs/1810.04805.

[19] *RoBERTa: A Robustly Optimized BERT Pretraining Approach* (2018). Liu, Ott, Goyal, Du, Joshi, Chen, Levy, Lewis, Zettlemoyer, and Stoyanov https://arxiv.org/abs/1907.11692.

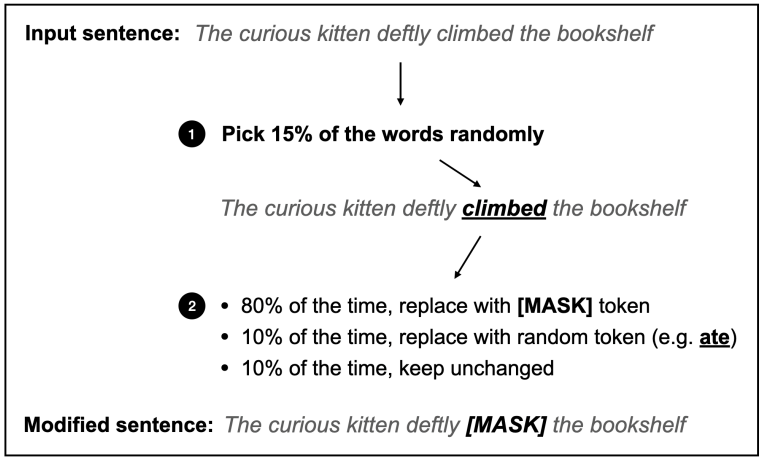

Figure 17.2. Illustration of the masked language modeling pretraining objective used in BERT-style transformers.

The main idea behind masked language modeling is to mask (or replace) random word tokens in the input sequence and then train the model to predict the original masked tokens based on the surrounding context.

Next to the masked language modeling pretraining task illustrated in the figure above, the next-sentence prediction task asks the model to predict whether the original document's sentence order of two randomly shuffled sentences is correct. For example, two sentences, in random order, are separated by the [SEP] token:

- [CLS] Toast is a simple yet delicious food [SEP] It's often served with butter, jam, or honey.
- [CLS] It's often served with butter, jam, or honey. [SEP] Toast is a simple yet delicious food.

The [CLS] token is a placeholder token for the model, prompting the model to return a *True* or *False* label indicating whether the sentences are in the correct order or not.

The masked language and next-sentence pretraining objective[20] allow BERT to learn rich contextual representations of the input texts, which can then be finetuned for various downstream tasks like sentiment analysis, question-answering, and named entity recognition.

RoBERTa (**R**obustly **o**ptimized **BERT** **a**pproach) is an optimized version of BERT. It maintains the same overall architecture as BERT but employs several training and optimization improvements, such as larger batch sizes, more training data, and eliminating the next-sentence prediction task. These changes resulted in RoBERTa achieving better performance on various natural language understanding tasks than BERT.

Decoders

Coming back to the original transformer architecture outlined at the beginning of this section, the multi-head self-attention mechanism in the decoder is similar to the one in the encoder, but it is masked to prevent the model from attending to future positions, ensuring that the predictions for position i can depend only on the known outputs at positions less than i. As illustrated in the figure below, the decoder generates the output word by word.

[20] It shall be noted that this pretraining is a form of self-supervised learning (see Q2 for more details).

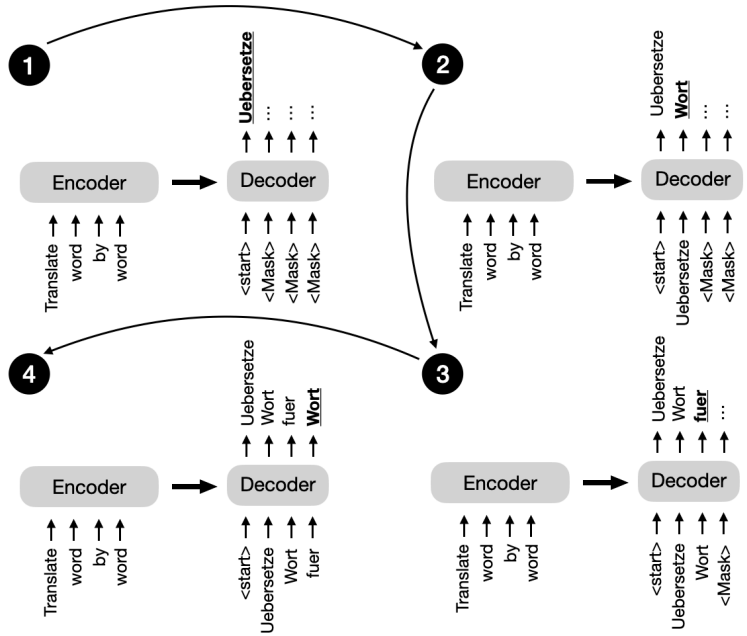

Figure 17.3. Illustration of the next-sentence prediction task used in the original transformer.

This masking (shown explicitly in the figure above, although it happens internally in the decoder's multi-head self-attention mechanism) is essential to maintain the autoregressive property of the transformer model during training and inference. The autoregressive property ensures that the model generates output tokens one at a time and uses previously generated tokens as context for generating the next word token.

Over the years, researchers have built upon the original encoder-decoder transformer architecture and developed several decoder-only models that have proven to be highly effective in various natural language processing tasks. The most notable models include the GPT family.

The GPT (**G**enerative **P**re-trained **T**ransformer) series are decoder-

only models pretrained on large-scale unsupervised text data and finetuned for specific tasks such as text classification, sentiment analysis, question-answering, and summarization. The GPT models, including GPT-2, GPT-3[21], and the more recent GPT-4, have shown remarkable performance in various benchmarks and are currently the most popular architecture for natural language processing.

One of the most notable aspects of GPT models is their emergent properties. Emergent properties refer to the abilities and skills that a model develops due to its next-word prediction pretraining. Even though these models were only taught to predict the next word, the pretrained models are capable of text summarization, translation, summarization, question answering, classification, and more. Furthermore, these models can perform new tasks without updating the model parameters via in-context learning, which is discussed in more detail in Q18.

Encoder-decoder hybrids

Next to the traditional encoder and decoder architectures, there have been advancements in the development of new encoder-decoder models that leverage the strengths of both components. These models often incorporate novel techniques, pre-training objectives, or architectural modifications to enhance their performance in various natural language processing tasks. Some notable examples of these new encoder-decoder models include BART[22] and T5[23].

Encoder-decoder models are typically used for natural language processing tasks that involve understanding input sequences and

[21]Brown et al. (2020). *Language Models are Few-Shot Learners*, https://arxiv.org/abs/2005.14165

[22]Lewis, Liu, Goyal, Ghazvininejad, Mohamed, Levy, Stoyanov, and Zettlemoyer (2018). *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*, https://arxiv.org/abs/1910.13461.

[23]Raffel, Shazeer, Roberts, Lee, Narang, Matena, Zhou, Li, and Liu (2019). *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*, https://arxiv.org/abs/1910.10683.

generating output sequences, often with different lengths and structures. They are particularly good at tasks where there is a complex mapping between the input and output sequences and where it is crucial to capture the relationships between the elements in both sequences. Some common use cases for encoder-decoder models include text translation and summarization.

Terminology and jargon

All of these methods, encoder-only, decoder-only, and encoder-decoder models, are sequence-to-sequence models (often abbreviated as *seq2seq*). Note that while we refer to BERT-style methods as encoder-only, the description *encoder-only* may be misleading since these methods also *decode* the embeddings into output tokens or text during pretraining.

In other words, both encoder-only and decoder-only architectures are "decoding." However, the encoder-only architectures, in contrast to decoder-only and encoder-decoder architectures, are not decoding in an autoregressive fashion. Autoregressive decoding refers to generating output sequences one token at a time, conditioning each token on the previously generated tokens. Encoder-only models do not generate coherent output sequences in this manner. Instead, they focus on understanding the input text and producing task-specific outputs, such as labels or token predictions.

Conclusion

In brief, encoder-style models are popular for learning embeddings used in classification tasks, encoder-decoder-style models are used in generative tasks where the output heavily relies on the input (for example, translation and summarization), and decoder-only models are used for other types of generative tasks including Q&A. Since the first transformer architecture emerged, hundreds of encoder-only, decoder-only, and encoder-decoder hybrids have been developed, as summarized in the figure below.

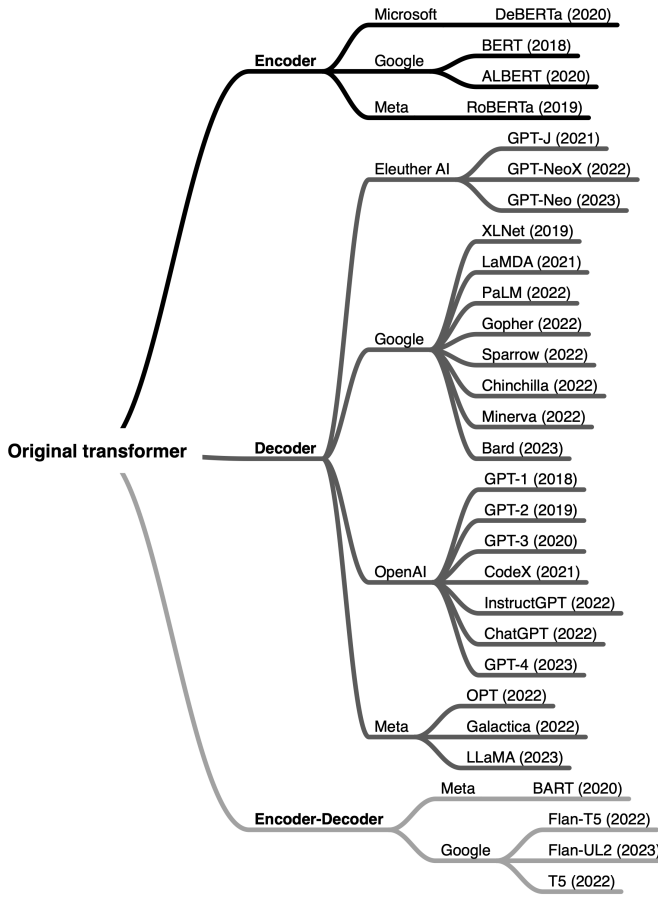

Figure 17.4. An overview of some of the most popular large language transformers organized by architecture type and developers.

While encoder-only models gradually lost in popularity, decoder-only models like GPT exploded in popularity thanks to breakthrough in text generation via GPT-3, ChatGPT, and GPT-4. However, encoder-only models are still very useful for training predictive models based on text embeddings versus generating texts.

> Reader quiz:

17-A

As discussed earlier, BERT-style encoder models are pretrained using masked language modeling and next-sentence prediction pretraining objectives. How could we adopt such a pretrained model for a classification task, for example, predicting whether a text has a positive or negative sentiment?

17-B

Can we finetune or a decoder-only model like GPT for classification?

Q18. Using and Finetuning Pretrained Transformers

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q19. Evaluating Generative Language Models

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Chapter 4. Production, Real-World, And Deployment Scenarios

Q20. Stateless And Stateful Training

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q21. Data-Centric AI

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q22. Speeding Up Inference

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q23. Data Distribution Shifts

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Chapter 5. Predictive Performance and Model Evaluation

Q24. Poisson and Ordinal Regression

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q25. Confidence Intervals

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q26. Confidence Intervals Versus Conformal Predictions

Q27. Proper Metrics

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q28. The *K* in *K*-Fold Cross-Validation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q29. Training and Test Set Discordance

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Q30. Limited Labeled Data

> Q:
Suppose we plotted a learning curve[24] and found that the machine learning model overfits and could benefit from more training data. Name different approaches for dealing with limited labeled data in supervised machine learning settings.

> A:

Next to collecting more data, there are several methods more or less related to regular supervised learning that we can use in limited-labeled data regimes.

1) Label more data

Collecting additional training examples is often the best way to improve the performance of a model[25]. However, this is often not feasible in practice. Listed below are various alternative approaches.

2) Bootstrapping the data

Similar to the topics discussed in *Q5, Reducing Overfitting with Data*, it can be helpful to "bootstrap" the data by generating modified (augmented) or artificial (synthetic) training examples to boost the performance of the predictive model.

Of course, improving the quality of data can also lead to improved predictive performance of a model, as discussed in *Q21, Data-Centric AI*.

3) Transfer learning

Transfer learning describes training a model on a general dataset (e.g., ImageNet) and then finetuning the pretrained target dataset (e.g., a specific dataset consisting of different bird species). Transfer

[24] See Q5, Figure 14 for a refresher of what a learning curve looks like.

[25] A learning curve is a good diagnostic to find out whether the model can benefit from more data. See Figure 15 in Q5 for details.

learning is usually done in the context of deep learning, where model weights can be updated. This is in contrast to tree-based methods since most decision tree algorithms are nonparametric models that do not support iterative training or parameter updates[26].

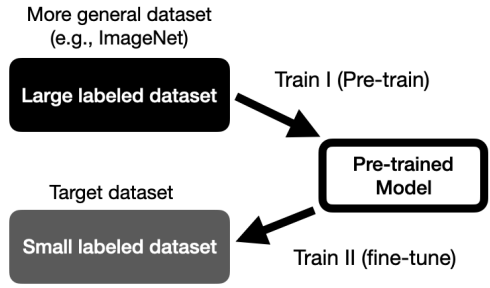

Figure 30.1. Illustration of transfer learning.

4) Self-supervised learning

Similar to transfer learning, self-supervised learning, the model is pretrained on a different task before it is finetuned to a target task for which only limited data exists. However, in contrast to transfer learning, self-supervised learning usually relies on label information that can be directly and automatically extracted from unlabeled data. Hence, self-supervised learning is also often called unsupervised pretraining. Common examples include "next word" (e.g., used in GPT) or "masked word" (e.g., used in BERT) prediction in language modeling. Or, an intuitive example from computer vision includes inpainting: predicting the missing part of an image that was randomly removed. (For more details about self-supervised learning, also see Q2.)

[26]While decision trees for incremental learning are not commonly implemented, algorithms for training decision trees in an iterative fashion do exist (https://en.wikipedia.org/wiki/Incremental_decision_tree).

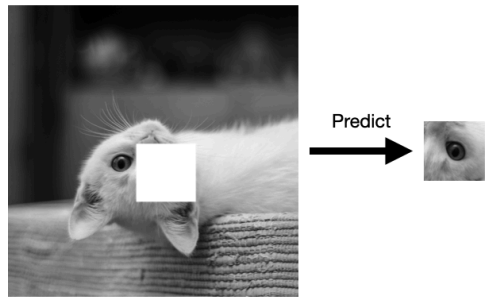

Figure 30.2. Illustration of inpainting for self-supervised learning.

5) Active learning

In active learning, we typically involve manual labelers or users for feedback during the learning process. However, instead of labeling the entire dataset upfront, active learning includes a prioritization scheme for suggesting unlabeled data points for labeling that maximize the machine learning model's performance.

The name *active learning* refers to the fact that the model is *actively* selecting data for labeling in this process. For example, the simplest form of active learning selects data points with high prediction uncertainty for labeling by a human annotator (also referred to as an *oracle*).

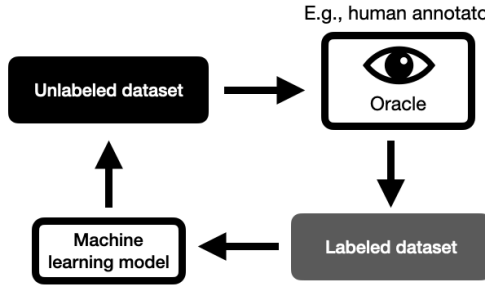

Figure 30.3. Illustration active learning.

6) Few-shot learning

In a few-shot learning scenario, we often deal with extremely small datasets where we usually only have a handful of examples per class. In research contexts, 1-shot (1 example per class) and 5-shot (5 examples per class) are very common.
An extreme case of few-shot learning is zero-shot learning, where no labels are provided. A recently popular example of zero-shot learning is GPT-3 and related language models. Here, the user has to provide all the necessary information via the input prompt, as illustrated in the figure below.

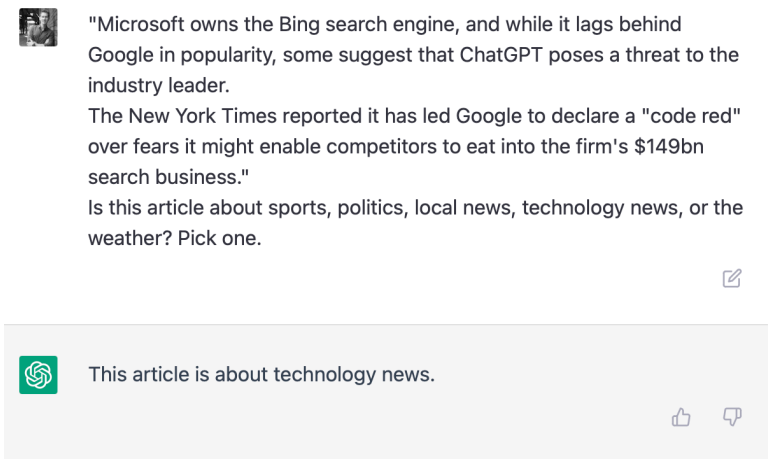

Figure 30.4. Example of zero-shot classification with ChatGPT.

(For more details about self-supervised learning, also see Q3.)

7) Meta-learning

We can think of meta-learning as "learning to learn" – we develop methods that learn how machine learning algorithms can best learn from data.
Over the years, the machine learning community developed several approaches for meta-learning. To further complicate matters, meta-learning can refer to different processes.

Meta-learning is one of the main subcategories of few-shot learning

(mentioned above). Here, the focus is on learning a *good* feature extraction module. The feature extraction module converts support and query images into vector representations. These vector representations are optimized for determining the predicted class of the query example via comparisons with the training examples in the support set. (This form of meta-learning is illustrated in Q3, Figure 13.)

Another branch of meta-learning, unrelated to the few-shot learning approach above, is focused on extracting meta-data (also called meta-features) from datasets for supervised learning tasks. The meta-features are descriptions of the dataset itself. For example, these can include the number of features and statistics of the different features (kurtosis, range, mean, etc.).

The extracted meta-features provide information for selecting a machine learning algorithm for the given dataset at hand. Using this approach, we can narrow down the algorithm and hyperparameter search spaces, which helps reduce overfitting when the dataset is small.

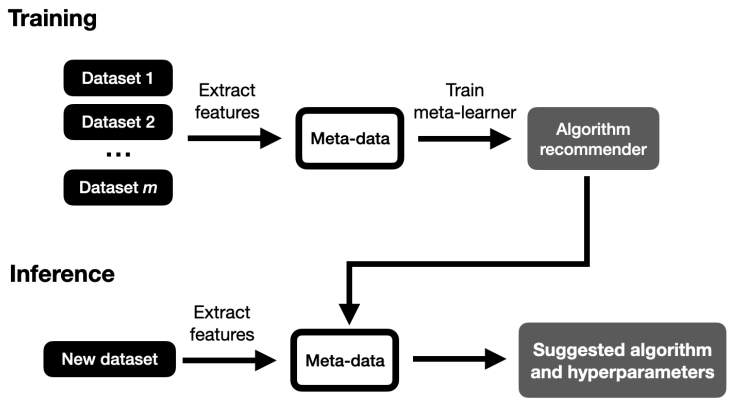

Figure 30.5. Illustration of the meta-learning process involving the extraction of meta-data.

8) Weakly supervised learning

Weakly supervised learning is a procedure where we use an external label source to generate labels for an unlabeled dataset. Often, the labels created by a weakly supervised labeling function are more noisy or inaccurate than those produced by a human or domain expert; hence, the term *weakly* supervised.

Often, we can develop or adopt a rule-based classifier to create the labels in weakly supervised learning – these rules usually only cover a subset of the unlabeled dataset.

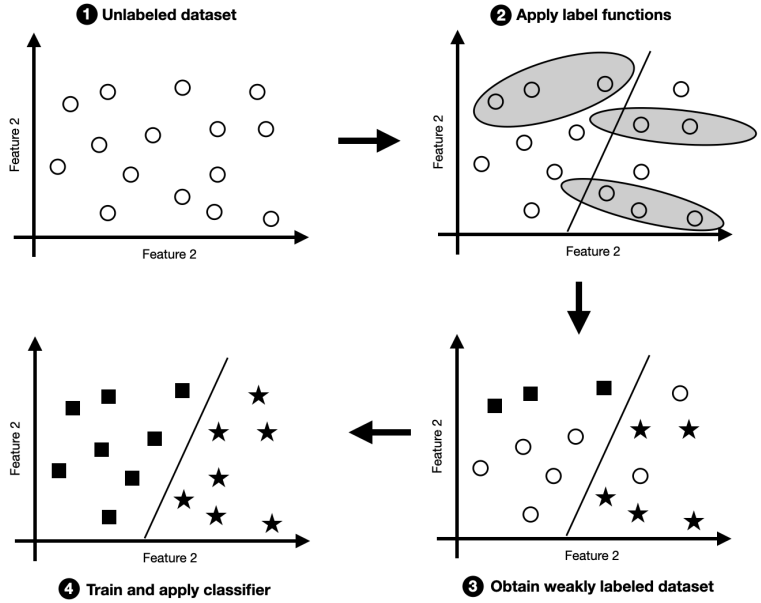

Figure 30.6. Illustration of weakly supervised learning.

Imagine the context of email spam classification as an example of a rule-based approach for data labeling. In weak supervision, we could design a rule-based classifier based on the keyword "SALE" in the email subject header line to identify a subset of spam emails. Note that while we may use this rule to label certain emails as spam-positive, we should not apply this rule to label emails without SALE as non-spam but leave those either unlabeled or apply a different

rule to these[27].

In short, weakly supervised learning is an approach for increasing the number of labeled instances in the training set. Hence, other techniques, such as semi-supervised, transfer, active, and zero-shot learning, are fully compatible with weakly supervised learning.

9) Semi-supervised learning

Semi-supervised learning is closely related to weakly supervised learning described above: we create labels for unlabeled instances in the dataset. The main difference between weakly supervised and semi-supervised learning is *how* we create the labels[28].

In weak supervision, we create labels using an external labeling function that is often noisy, inaccurate or only covers a subset of the data. In semi-supervision, we do not use an external label function but leverage the structure of the data itself.

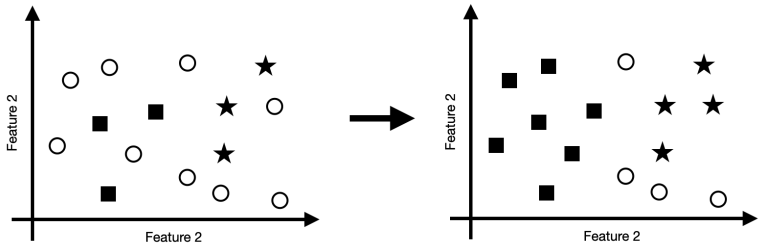

Figure 30.7. Illustration of semi-supervised learning.

In semi-supervised learning, we can, for example, label additional data points based on the density of neighboring labeled data points, as illustrated in the figure below.

While we can apply weak supervision to an entirely unlabeled dataset, semi-supervised learning requires at least a portion of

[27]There is a subcategory of weakly supervised learning referred to as PU-learning. In PU-learnining, which is short for positive-unlabeled learning, we only label and learn from positive examples.

[28]Semi-supervised learning is sometimes referred to as a subcategory of weakly supervised learning and vice versa.

the data to be labeled. In practice, it is possible first to apply weak supervision to label a subset of the data and then use semi-supervised learning to label instances that were not captured by the labeling functions.

10) Self-training

Self-training is a category that falls somewhere between semi-supervised learning and weakly supervised learning. In self-training, we train a model or adopt an existing model to label the dataset. This model is also referred to as a *pseudo-labeler*.

Since the model used in self-training does not guarantee accurate labels, self-training is related to weakly supervised learning. Moreover, while we use or adopt a machine learning model for this pseudo-labeling, self-training is also related to semi-supervised learning.

An example of self-training is knowledge distillation, discussed in Q6.

11) Multi-task learning

Multi-task learning trains neural networks on multiple, ideally related tasks. For example, suppose we are training a classifier to detect spam emails; here, spam classification is the main task. In multi-task learning, we can add one or more related tasks the model has to solve. These additional tasks are also referred to as *auxiliary tasks*. If the main task is email spam classification, an auxiliary task could be classifying the email's topic or language.

Typically, multi-task learning is implemented via multiple loss functions that have to be optimized simultaneously – one loss function for each task. The auxiliary tasks serve as an inductive bias, guiding the model to prioritize hypotheses that can explain multiple tasks. This approach often results in models that perform better on unseen data[29].

[29]Caruana (1997). *Multi-task learning.* Machine Learning. 28: 41–75. https://doi.org/10.1023%2FA%3A1007379606734

There are two subcategories of multi-task learning: (1) multi-task learning with *hard* parameter sharing and (2) multi-task learning with *soft* parameter sharing[30].

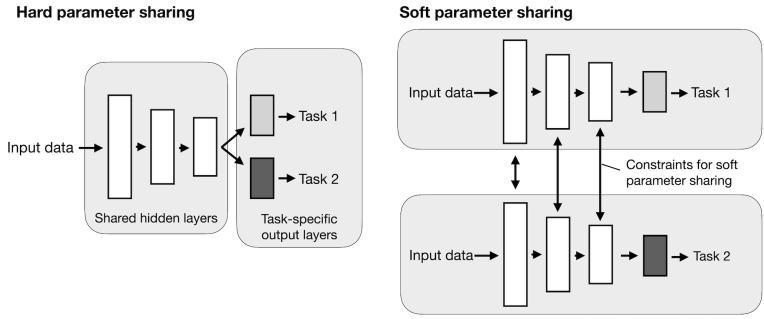

Figure 30.8. Illustration of the two main types of multi-task learning. For simplicity, the figure depicts only two tasks, but multitask learning can be used for any number of tasks.

The figure above illustrates the difference between hard and soft parameter sharing. In hard parameter sharing, only the output layers are task-specific, while all tasks share the same hidden layers and neural network backbone architecture. In contrast, soft parameter sharing uses separate neural networks for each task, but regularization techniques such as distance minimization between parameter layers are applied to encourage similarity among the networks.

12) Multi-modal learning

While multi-task learning involves training a model with multiple tasks and loss functions, multi-modal learning focuses on incorporating multiple types of input data.

Common examples of multi-modal learning are architectures that take both image and text data as input[31]. Depending on the task,

[30] Ruder (2017). *An Overview of Multi-Task Learning in Deep Neural Networks.* https://www.ruder.io/multi-task/.

[31] Multi-modal learning is not restricted two only two modalities and can be used for any number of inputs modalities.

we may employ a matching loss that forces the embedding vectors (Q1) between related images and text to be similar, as shown in the figure below.

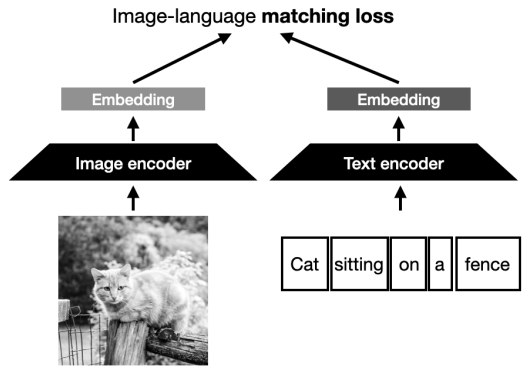

Figure 30.9. Illustration of multi-modal learning with a matching loss that forces embeddings from different types of inputs to be similar.

The figure above shows image and text encoders as separate components. The image encoder can be a convolutional backbone or a vision transformer, and the language encoder can be a recurrent neural network or language transformer. However, it's common nowadays to use a single transformer-based module that can simultaneously process image and text data[32].

Optimizing a matching loss, as shown in the previous figure, can be useful for learning embeddings that can be applied to various tasks, such as image classification or summarization. However, it is also possible to directly optimize the target loss, like classification or regression, as the figure below illustrates.

[32]For example, VideoBERT is a model that with a joint module that processes both video and text for action classification and video captioning. Reference: Sun, Myers, Vondrick, Murphy, Schmid (2019). *VideoBERT: A Joint Model for Video and Language Representation Learning*, https://arxiv.org/abs/1904.01766.

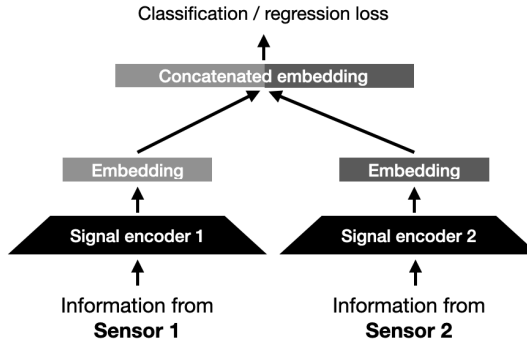

Figure 30.10. Illustration of multi-modal learning for optimizing a supervised learning objective.

Intuitively, models that combine data from different modalities generally perform better than uni-modal models because they can leverage more information. Moreover, recent research suggests that the key to the success of multi-modal learning is the improved quality of the latent space representation[33].

13) Inductive biases

Choosing models with stronger inductive biases can help to lower data requirements by making assumptions about the structure of the data. For example, due to their inductive biases, convolutional networks require less data than vision transformers as discussed in Q13.

Which techniques should we use?

Now that we covered several techniques for lowering the data requirements, which ones should we use?

Collecting more data and techniques such as data augmentation and feature engineering are compatible with all the methods discussed above. Also, multi-task learning and multi-modal inputs can be used with the other learning strategies outlined above. If the

[33]Huang, Du, Xue, Chen, Zhao, Huang, (2021). *What Makes Multi-Modal Learning Better Than Single (Provably)*, https://arxiv.org/abs/2106.04538.

model suffers from overfitting, techniques from Q5 and Q6 should also be included.

How about active learning, few-shot learning, transfer learning, self-supervised learning, semi-supervised learning, and weakly supervised learning? Which technique(s) to try highly depends on the context, and the figure below provides an overview that can be used for guidance.

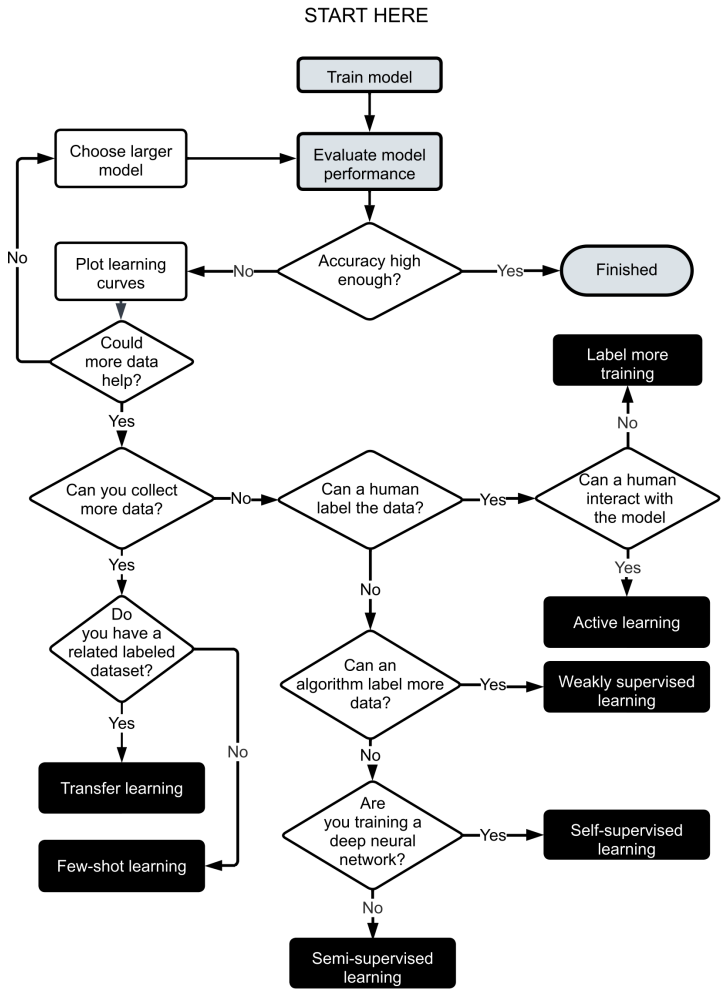

Figure 30.11. Recommendations for choosing a supervised learning techniques. The black boxes are not terminal nodes but arch back to Evaluate model performance (the arrows were omitted to avoid visual clutter).

> Reader quiz:

30-A

Given the task of constructing a machine learning model that utilizes images to detect manufacturing defects on the outer shells of tablet devices similar to iPads, we have access to the following data:

- Millions of images of various computing devices, including smartphones, tablets, and computers, which are not labeled.
- Thousands of labeled pictures of smartphones depicting various types of damage.
- Hundreds of labeled images specifically related to the target task of detecting manufacturing defects on tablet devices.

How could we approach this problem using self-supervised learning or transfer learning?

30-B

In active learning, selecting difficult examples for human inspection and labeling is often based on confidence scores. Neural networks can provide such scores by using the logistic sigmoid or softmax function in the output layer to calculate class-membership probabilities. However, it is widely recognized that deep neural networks exhibit overconfidence on out-of-distribution data[34], rendering their use in active learning ineffective. So, what are some other methods to obtain confidence scores using deep neural networks for active learning?

[34]Nguyen, Yosinski, Clune, (2015). *Deep Neural Networks Are Easily Fooled: High Confidence Predictions for Unrecognizable Image*, https://arxiv.org/abs/1412.1897.

Afterword

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Appendix A: Reader Quiz Solutions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.

Appendix B: List of Questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/machine-learning-q-and-ai.