Classical Machine Learning

Week 0

Plan

- Setting up your learning and programming environment

Getting started

- Setting up your ML environment (Setup_NYU.ipynb)
    - Choosing an ML environment (Choosing_an_ML_Environment_NYU.ipynb)
- Quick intro to the tools (Getting_Started.ipynb)

Week 1

Plan

We give a brief introduction to the course.

We then present the key concepts that form the basis for this course

- For some: this will be review

Intro to Advanced Course

- Introduction to Advanced Course (Intro_Advanced.ipynb)

Here is a *quick reference* of key concepts/notations from the Intro course

- For some: it will be a review, for others: it will be a preview.
- We will devote a sub-module of this lecture to elaborate on each topic in slightly more depth.
    - For a more detailed explanation: please refer to the material from the Intro course (repo (https://github.com/kenperry-public/ML_Spring_2023))
- Review and Preview (Review_Advanced.ipynb)

You may want to run your code on Google Colab in order to take advantage of powerful GPU's.

Here are some useful tips:

Google Colab tricks (Colab_practical.ipynb)

Review/Preview of concepts from Intro Course

Transfer Learning: Review (Review_TransferLearning.ipynb)

Transformers: Review (Review_Transformer.ipynb)

Suggested reading

- Attention
  - Attention is all you need (https://arxiv.org/pdf/1706.03762.pdf)
- Transfer Learning
  - Sebastian Ruder: Transfer Learning (https://ruder.io/transfer-learning/)

Further reading

- Attention
  - Neural Machine Translation by Jointly Learning To Align and Translate (https://arxiv.org/pdf/1409.0473.pdf)
  - Geron Chapter 16
  - An Analysis of BERT's Attention (https://arxiv.org/pdf/1906.04341.pdf)

Week 2: Review/Preview (continued); Technical

Preview

There is lots of interest in Large Language Models (e.g., ChatGPT). These are based on an architecture called the Transformer. We will introduce the Transformer and demonstrate some amazing results achieved by using Transformers to create Large Language Models.

Attention is a mechanism that is a core part of the Transformer. We will begin by first introducing Attention.

We will then take a detour and study the Functional model architecture of Keras. Unlike the Sequential model, which is an ordered sequence of Layers, the organization of blocks in a Functional model is more general. The Advanced architectures (e.g., the Transformer) are built using the Functional model.

Once we understand the technical prerequisites, we will examine the code for the Transformer.

Plan

We continue the review/preview of key concepts that we started last week.

Our ultimate goal is to introduce the Transformer (which uses Attention heavily) in theory, and demonstrate its use in Large Language Models.

Review/preview continued

Transformers: Review continued

- Attention (continued) (Intro_to_Attention.ipynb#Attention)
- Transformer (Transformer.ipynb)

Natural Language Processing: Review (Review_NLP.ipynb)

Functional Models

Plan

Enough theory (for the moment) !

The Transformer (whose theory we have presented) is built from plain Keras.

Our goal is to dig into the **code** for the Transformer so that you too will learn how to build advanced models.

Before we can do this, we must

- go beyond the Sequential model of Keras: introduction to the Functional model
- understand more "advanced" features of Keras: customomizing layers, training loops, loss functions
- The Datasets API

Basics

We start with the basics of Functional models, and will give a coding example of such a model in Finance.

- Functional API (Functional_Models.ipynb)

Functional Model Code: A Functional model in Finance: "Factor model"

We illustrate the basic features of Functional models with an example

- does not use the additional techniques of the next section (Advanced Keras)

Autoencoders for Conditional Risk Factors (Autoencoder_for_conditional_risk_factors.ipynb)

- code (https://github.com/stefan-jansen/machine-learning-for-trading/blob/main/20_autoencoders_for_conditional_risk_factors/06_conditional_a

Week 3

Plan

We continue our study of Functional Models, introducing Advanced capabilities of Keras.

Threading

In our Autoencoders for Conditional Risk Factors (Autoencoder_for_conditional_risk_factors.ipynb) last week, we skipped over an important detail

- the "Beta" side of the notebook takes as input a matrix rather than a vector
- how does a `Dense` layer work on higher dimensional data ?

The answer is similar to how Python works: Threading (Keras_Advanced.ipynb#Factor-Models-and-Autoencoders:-Threading)

Advanced Keras

- As part of our introduction to the Functional API (Functional_Models.ipynb) we briefly covered some advanced topics
    - Custom layer type (Keras_Advanced.ipynb#Layer-specialization)
- We continue with some other useful techniques, emphasizing interesting "Loss" functions
    - Custom loss, Custom training loop (Keras_Advanced.ipynb#Model-specialization)
    - Non-trivial Custom Loss: Neural Style Transfer (Neural_Style_Transfer.ipynb)
    - Gradient Ascent: Maximize utility for visualization (Keras_Advanced.ipynb#Visualizing-what-CNN's-learn:-Gradient-Ascent-and-the-Gradient-Tape)

Deeper dives

We have a notebook (Keras_Advanced.ipynb)that is a **reference** to all the advanced techniques.

- most will be introduced as part of our study of different interesting models
- but this notebook is one convenient place to see them all

If you *really* want to dig into the micro-details of TensorFlow, here are some important subtleties

- Computation Graphs (Computation_Graphs.ipynb)
- Eager vs Graph Execution (TF_Graph.ipynb)

Putting it all together: Code: the Transformer

Plan

With the base of advanced Keras under our belts, it's time to understand the Transformer, in code

We will examine the code in the excellent TensorFlow tutorial on the Transformer (https://www.tensorflow.org/text/tutorials/transformer)

- The Transformer: Understanding the Pieces (Transformer_Understanding_the_Pieces.ipynb)
- The Transformer: Code (Transformer_code.ipynb)
- Choosing a Transformer architecture (Transformer_Choosing_a_PreTrained_Model.ipynb)

Suggested reading

There is an excellent tutorial on Attention and the Transformer which I recommend:

- Tensorflow tutorial: Neural machine translation with a Transformer and Keras (https://www.tensorflow.org/text/tutorials/transformer)

Deeper dive

Review/Preview: Language Models (continue)

Large Language Modells (LLMs) are the basis of the amazing advances we are witnessing in NLP (e.g., GPT).

It will also be an essential part of our Final Project.

Let's introduce the topic.

Language Models, the future (present ?) of NLP: Review (Review_LLM.ipynb)

Beyond Transfer Learning: Fine-tuning a pre-trained model

Plan

We begin the "technical" part of the course: the programming tools that will enable the Course Project.

We introduce "Modern Transfer Learning": using model hubs.

The hub we will use for the final project: HuggingFace

- illustrate how to fine-tune a pre-trained model
- quick Intro to HF
    - best way to learn: through the course !
    - uses Datasets
        - will introduce later
    - PyTorch version (uses Trainer); we will focus on Tensorflow/Keras version

HuggingFace Transformers course

The best way to understand and use modern Transfer Learning is via the HuggingFace course (https://huggingface.co/course).

You will learn

- about the Transformer
- how to use HuggingFace's tools for NLP (e.g., Tokenizers)
- how to perform common NLP tasks
    - especially with Transformers
- how to fine-tune a pre-trained model
- how to use the HuggingFace dataset API

All of this will be invaluable for the Course Project.

- does not have to be done using HuggingFace
- but using at least parts of it will make your task easier

- HuggingFace intro (Transfer_Learning_HF.ipynb)
    - linked notebook: Using a pretrained Sequence Classifier (HF_quick_intro_to_models.ipynb)

Suggested reading

HuggingFace course (https://huggingface.co/course)

Week 4

Transformer wrap-up

- Choosing a Transformer architecture (Transforming_Choosing_a_PreTrained_Model.ipynb)

Intro to HuggingFace (continued)

- HuggingFace intro (Transfer_Learning_HF.ipynb)
    - linked notebook: Using a pretrained Sequence Classifier (HF_quick_intro_to_models.ipynb)

Datasets: Big data in small memory

Plan

We continue our exploration of the Functional API in Keras.

We will spend some time examining the code for the Transformer.

We will also introduce the TensorFlow Dataset (TFDS) API, a way to consume large datasets using a limited amount of memory.

Plan

Last piece of technical info to enable the project

- TensorFlow Dataset (TF_Data_API.ipynb)

Background

- Python generators (Generators.ipynb)

Notebooks

- Dataset API: play around (TFDatasets_play_v1.ipynb)

Beyond the LLM

Plan

The LLM is trained to "predict the next" token.

Although this sounds boring, it is the basis of a Universal API for problem solving.

We also show the remarkable ability to adapt on Pre-Trained LLM to a new task *without* fine-tuning

- and without changing any parameters !

Beyond the LLM (Review_LLM.ipynb#Beyond-the-LLM)

Transformers: Scaling

We now have the capabilities to build models with extremely large number of weights. Is it possible to have too many weights ?

Yes: weights, number of training examples and compute capacity combine to determine the performance of a model.

There is an empirical result that suggests that in order to take advantage of GPT-3's use of 175 billion weights

- 1000 times more compute is required than what was used
- 10 times more training examples is required compared to what was used

How large should my Transformer be ? (Transformers_Scaling.ipynb)

Suggested reading

- Scaling laws (https://arxiv.org/pdf/2001.08361.pdf)

Re-visiting the Transformer code: deeper details

- Implementing Attention (Attention_Lookup.ipynb)
- Transformer code: parts that we skipped over (Transformer_Understanding_the_Pieces.ipynb#General)
    - Residual connections, Embeddings, Positional Encoding
- Residual connections (RNN_Residual_Networks.ipynb)

Week 5

New major topic: Synthetic data.

After last week's "code-heavy" modules, we are back to "theory" !

We will address several ways to create new examples, starting with the simplest model and moving on to models that are more complex.

We wrap up by demonstrating a new trend

- using Large Language Models
- to create training data
- to improve Large Language Models !

Synthetic Data: Autoencoders

Generating synthetic data using Autoencoders and its variants.

"Vanilla" Autoencoder

Autoencoder (Autoencoders_Generative.ipynb)

Suggested Reading

TensorFlow Tutorial on Autoencoders (https://www.tensorflow.org/tutorials/generative/autoencoder)

Variational Autoencoder (VAE)

We now study a different type of Autoencoder

- that learns a *distribution* over the training examples
- by sampling from this distribution: we can create synthetic examples

Variational Autoeconder (VAE) (VAE_Generative.ipynb)

Suggested Reading

TensorFlow tutorial on VAE (https://www.tensorflow.org/tutorials/generative/cvae)

Further reading

Tutorial on VAE (https://arxiv.org/pdf/1606.05908.pdf)

Synthetic Data: GANs

We introduce a new type of model that can be used to generate synthetic data: the Generative Adversarial Network (GAN). It uses a competitive process involving two Neural Networks in order to iteratively produce synthetic examples of increasing fidelity to the true data.

- GAN: basic (GAN_Generative.ipynb)
- GAN loss (GAN_Loss_Generative.ipynb)
- Wasserstein GAN (Wasserstein_GAN_Generative.ipynb)

Notebooks

- GAN to Generate Faces (CelebA_01_deep_convolutional_generative_adversarial_network.ipynb)

Suggested reading

- Generative Adversarial Nets (https://arxiv.org/pdf/1406.2661.pdf)
- TensorFlow Tutorial DCGAN (https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tut
    - this is a tutorial from which our code notebook was derived

Synthetic Data: Self-improvement of an LLM by generating examples

We present a way of synthesizing examples to improve a Large Language Model. In this case: the examples we create are *text*.

This is a potential solution to one of the issues with Fine-Tuning a LLM: the lack of sufficient labeled examples for the Target task.

- LLM Instruction Following (LLM_Instruction_Following.ipynb)
- Synthetic data for Instruction Following (LLM_Instruction_Following_Synthetic_Data.ipynb)

Suggested Reading

- InstructGPT paper (https://arxiv.org/pdf/2203.02155.pdf)
- Self-instruct (https://arxiv.org/pdf/2212.10560.pdf)
- Self improvement (https://arxiv.org/pdf/2210.11610.pdf)
    - goal is to fine-tune a LLM for question answering
        - without an **a priori** fine-tuning dataset
            - use a LLM to **generate** a fine-tuning dataset
            - Use few-shot, CoT prompts:
                - Input=question; Output=answer + rationale
                - Input=question, LLM generates output
                - multiple outputs
                - extract answer from output
                - - use majority voting on answer to filter responses - hopefully: majority is accurate: "high confidence" == fraction of responses that agree ?
            - The high confidence (large fraction of generated responses to a question agree in answer) generated examples become the fine-tuning dataset

Social concerns

Model bias

- show model cards

Environmental

Training a Large Language Model uses substantial compute, which means lots of energy and, hence, environmental impact.

There is also a cost to *using* (i.e., inference) a trained model.

Let's examine the impact.

- ML Carbon impact calculator (https://mlco2.github.io/impact/#compute)
    - GPT-3:
        - $\approx 10^{23}$ Flops to train (see [Total compute: detailed calc for many models])(https://arxiv.org/pdf/2005.14165.pdf#page=46 (https://arxiv.org/pdf/2005.14165.pdf#page=46))
        - NVidia A100: 5*10^{15} Flops
        - 3600 seconds/hour
        - $\frac{10^{23}}{3600*5*10^{15}} \approx 28000$ hours to train on single A100
        - Calculated CO_2 equivalent: $1.7 * 10^4$ kilometers of car driving
        - According to the Scaling laws module (Index_Advanced.ipynb#Transformers:-Scaling)
            - we could potentially achieve the same performance of GPT-3 on a *smaller* (less environmentally impactful model)
    - Energy: train vs inference (https://arxiv.org/pdf/2005.14165.pdf#page=39)

Alignment

Language models show great capabilities but also the potential for harm: biased and offensive generated text, for example. Can we "align" a model's output with human values ?

- Alignment (Alignment.ipynb)
- Alignment Anthropic (Alignment_Anthropic.ipynb)

Training: tricks of the trade

Training, in practice, involves more than a model and a training set

- Using multiple machines/GPU's: expect something to fail in the middle
- Loss does not always decrease with increasing epoch
    - Learning rate schedule "mid-flight corrections"

Some practical lessons are found here (Training_a_LLM_practical.ipynb).

Suggested reading

- OPT: Open Pre Trained Model (https://arxiv.org/pdf/2205.01068.pdf)

Parameter Efficient Transfer Learning

Transfer learning may be the "future of Deep Learning" in that we can adapt models (that are too big for us to train on our own) to our own tasks.

But Fine-Tuning a Pre-Trained model involves training a lot of parameters when the base model is large.

This may be difficult for a variety of reasons.

Can we adapt a Pre-Trained base model to a new task *without* training a large number of parameters ?

- Parameter Efficient Transfer Learning (ParameterEfficient_TransferLearning.ipynb)

Suggested reading

- Parameter Efficient Transfer Learning - article (https://lightning.ai/pages/community/article/understanding-llama-adapters/)

Adapters

- Parameter Efficient Transfer Learning for NLP

Week 6 (tentative)

Assignments

Your assignments should follow the Assignment Guidelines (assignments/Assignment_Guidelines.ipynb)

Final Project

Assignment notebook (assignments/FineTuning_HF/FineTune_FinancialPhraseBank.ipynb)

In [1]:
```
print("Done")
```

```
Done
```