Preliminaries

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)

Review/Preview of concepts from Intro Course

Notation, concepts

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)

Colab

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: in-depth

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)](https://arxiv.org/pdf/1706.03762.pdf)
- Transfer Learning
  - [Sebastian Ruder: Transfer Learning (https://ruder.io/transfer-learning/)](https://ruder.io/transfer-learning/)

Further reading

- Attention

Week 2: Review/Preview (continued); Technical

Preview

We continue with our "review" of Attention and follow up by showing how to implement it.

This will enable us to "review" the Transformer

- a fundamental architecture for processing sequences
- built very heavily upon Attention

The Transformer is important because it is the tool upon which Large Language Models are built.

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.

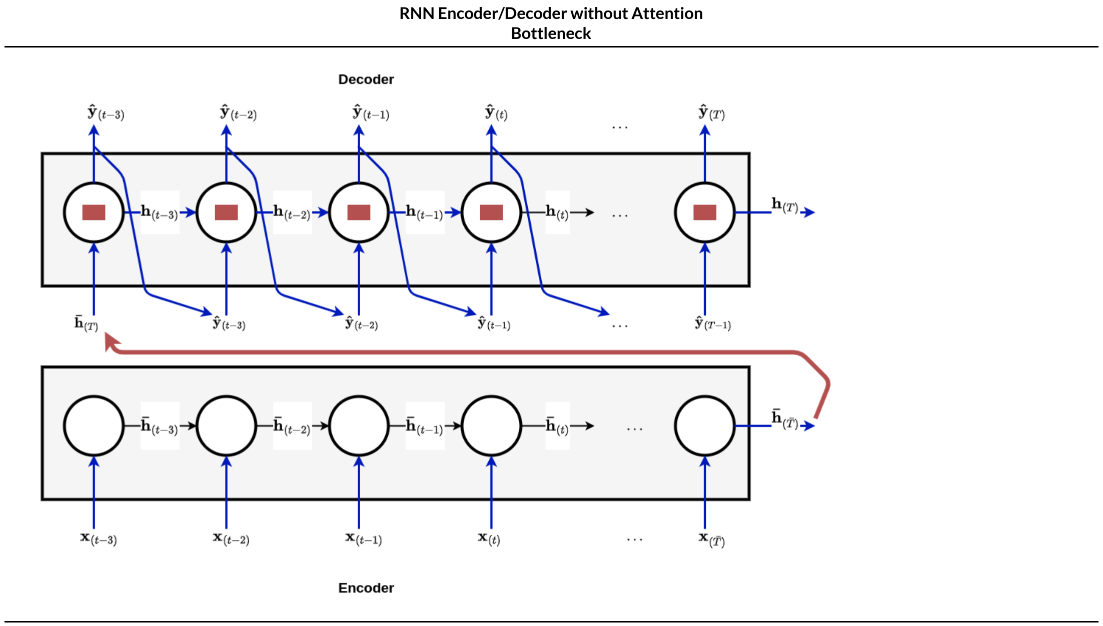
Once we understand the technical prerequisites, we will examine the code for the Transformer.

Plan

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

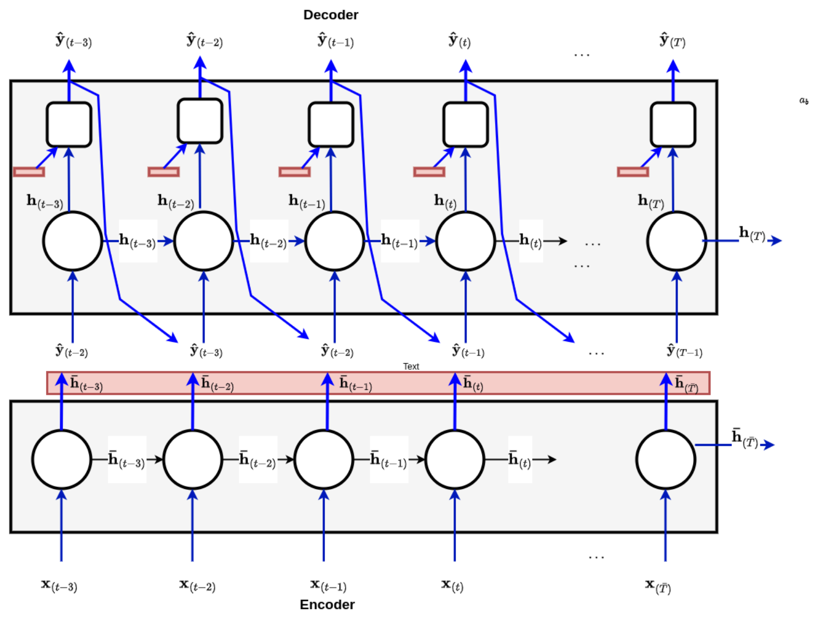
Our ultimate goal is to introduce the Transformer (which uses Attention heavily) in theory, and demonstrate its use in Large Language Models.

Review/preview continued



$$\bar{\mathbf{h}}_{(\bar{T})} = \left\{ \begin{array}{l} \text{Subject: Professor Perry} \\ \text{Pronoun: he} \\ \text{Object: Machine Learning} \\ \text{Indirect Object: them} \\ \text{Verb: taught} \end{array} \right\}$$

RNN Encoder/Decoder with Attention



$\bar{\mathbf{h}}_{(\bar{T})} =$

Subject: Professor Perry

Pronoun: he

Object: Machine Learning

Indirect Object: them

Verb: taught

Transformers: Review continued

- Attention review: continued (Intro_to_Attention.ipynb#Visualizing-Attention)
    - visualizing Attention; flavors
- Implementing Attention (Attention_Lookup.ipynb)
- Transformer (Transformer.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)

Week 3

Plan

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

Functional Models (continued)

- Functional API (continued) (Functional_Models.ipynb#Example:-Nested-Models)

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

Threading

In our Autoencoders for Conditional Risk Factors (Autoencoder_for_conditional_risk_factors.ipynb), 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)

Week 4

Putting it all together: Code: the Transformer (continued)

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)

- more in-depth than our presentation
- more background

The tutorial is especially recommended for those without the basics of the Transformer from my Intro course

- 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

- Implementing Attention (Attention_Lookup.ipynb)
- Residual connections (RNN_Residual_Networks.ipynb)

Advanced Keras (Deeper dive)

We will not cover this notebook (Keras_Advanced.ipynb) in class

- most of the material will be introduced as part of our study of different interesting models
- but this notebook is one convenient place to see them all
- consider it as a **reference** that collects multiple techniques in one place

Deeper dives

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)

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)](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) **(local machine)**
    - linked notebook: Using a pretrained Sequence Classifier (https://colab.research.google.com/github/kenperry-public/ML_Advanced_Spring_2024/blob/master/HF_quick_intro_to_model **(Google Colab)**
        - Examining a model

Suggested reading

Fine Tuning at low cost

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 (https://arxiv.org/pdf/1902.00751.pdf)
    - LLM Adapters (https://arxiv.org/pdf/2304.01933.pdf)

Fine Tuning by Proxy

Can we fine-tune a large model

- **without** adapting its weights
- by fine-tuning a **small** model
    - much lower cost than fine-tuning a large model

Fine Tuning by Proxy (FineTuning_by_Proxy.ipynb)

Suggested reading

Tuning Language Models by Proxy (https://arxiv.org/pdf/2401.08565.pdf)

Week 5

Last week's lecture was hampered by Internet connectivity issues.

We also skipped over some nice background material in the race to cover material that would enable you to start the Final Project.

We begin this week

- cover the background material
- re-start the topic of Parameter Efficient Transfer Learning.

We will then move on to new material on the topic of Large Language Models.

Re-cap: Using Hugging Face

- HuggingFace intro (Transfer_Learning_HF.ipynb)
    - linked notebook: Using a pretrained Sequence Classifier (HF_quick_intro_to_models.ipynb) **(local machine)**
    - linked notebook: Using a pretrained Sequence Classifier (https://colab.research.google.com/github/kenperry-public/ML_Advanced_Spring_2024/blob/master/HF_quick_intro_to_model **(Google Colab)**
        - Examining a model

Additional Neural Network topics in the Transformer code

- Residual connections (Transformer_Understanding_the_Pieces.ipynb#Residual-connections)
- Embeddings (Transformer_Understanding_the_Pieces.ipynb#Embedding)

- Layer normalization (Transformer_Understanding_the_Pieces.ipynb#Layer-Normalization-(part-of-Add-and-Norm))

- NLP: following sub-section

- LLM: following sub-section

Fine Tuning at low cost

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 (https://arxiv.org/pdf/1902.00751.pdf)
    - LLM Adapters (https://arxiv.org/pdf/2304.01933.pdf)

Fine Tuning by Proxy

Can we fine-tune a large model

- **without** adapting its weights
- by fine-tuning a **small** model
    - much lower cost than fine-tuning a large model

Fine Tuning by Proxy (FineTuning_by_Proxy.ipynb)

Suggested reading

Tuning Language Models by Proxy (https://arxiv.org/pdf/2401.08565.pdf)

Defer remaiing two subsections until after we present LLM, Universal API

Fine Tuning by Prompt Engineering

Can we adapt a base LLM to solve a new Target task just by changing the prompt ?

Surprisingly: yes !

- Fine Tuning by Prompt Tuning (Prompt_Engineering_Tuning.ipynb)
- Few-shot and Zero shot learning are included in the prompting techniques studied. Why do they work ?
    - In Context Learning: Theory (In_Context_Learning_Theory.ipynb)

Fine-Tuning "without** fine-tuning" learning a new task from exemplars

- In Context Learning (Review_LLM.ipynb#Universal-API/In-context-Learning)

Review/Preview: Language Models

Plan

We will spend a lot of time dealing with inputs/outputs that are sequences of tokens ("words").

We start with a quick review of the basics of Natural Language Processing.

We then do a quick review of Large Language Models

- a specific kind of NLP model, that solves a very simple task
- is the basis for the Assistants (e.g., ChatGPT) that have become increasingly prevalent

Natural Language Processing: Review (Review_NLP.ipynb)

LLM review

Large Language Models (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 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)](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)

Further reading

- Inference budget
    - Transformer Inference Arithmetic (https://kipp.ly/transformer-inference-arithmetic/)
    - LLaMA: Open and Efficient Foundation Language Models (https://arxiv.org/pdf/2302.13971.pdf)
    - Large language models aren't trained enough (https://finbarr.ca/llms-not-trained-enough/)

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)

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 6

LLM’s continued

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)

Further reading

- Inference budget
    - Transformer Inference Arithmetic (https://kipp.ly/transformer-inference-arithmetic/)
    - LLaMA: Open and Efficient Foundation Language Models (https://arxiv.org/pdf/2302.13971.pdf)
    - Large language models aren't trained enough (https://finbarr.ca/llms-not-trained-enough/)

Fine-tuning a low cost

Fine Tuning by Proxy

Can we fine-tune a large model

- **without** adapting its weights
- by fine-tuning a **small** model
    - much lower cost than fine-tuning a large model

Fine Tuning by Proxy (FineTuning_by_Proxy.ipynb)

Suggested reading

Tuning Language Models by Proxy (https://arxiv.org/pdf/2401.08565.pdf)

Fine Tuning by Prompt Engineering

Can we adapt a base LLM to solve a new Target task just by changing the prompt ?

Surprisingly: yes !

- Fine Tuning by Prompt Tuning (Prompt_Engineering_Tuning.ipynb)

Fine-Tuning "without fine-tuning": learning a new task from exemplars

- In Context Learning (Review_LLM.ipynb#Universal-API/In-context-Learning)
- Few-shot and Zero shot learning are included in the prompting techniques studied. Why do they work ?
    - In Context Learning: Theory (In_Context_Learning_Theory.ipynb)

Synthetic Data

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)

Week 7

Plan

We continue with the topic of Synthetic Data.

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

Synthetic Data: Vector Quantized Autoencoders

While we are on the topic of Autoencoders, we present the Vector Quantized Autoencoder.

It is not so much a tool for creating Synthetic Data as a natural continuation of our Autoencoder exploration.

Vector Quantized Autoencoder (VQ_VAE_Generative.ipynb)

Suggested Reading

vanilla VQ-VAE (https://arxiv.org/pdf/1711.00937.pdf)

VQ-VAE-2 paper (https://arxiv.org/pdf/1906.00446.pdf)

DALL-E: Mixing Text and Image

We make use of the Quantized VAE technique we learned in the module on Autoencoders to enable us to mix text and image.

We discuss how a Text to Image model (convert the textual description of an image to an actual image) works.

- CLIP (CLIP.ipynb)
    - Zero shot learning, prompt engineering Colab notebook: PyTorch (https://github.com/openai/CLIP/blob/main/notebooks/Prompt_Engineerir
- DALL-E (DALL-E.ipynb)
- Vision Transformer (Vision_Transformer.ipynb)

Suggested Reading

- CLIP paper (https://cdn.openai.com/papers/Learning_Transferable_Visual_Models_From_Natur

- DALL-E paper (https://arxiv.org/pdf/2102.12092.pdf)

- OpenAI DALL-E 2 announcement (https://openai.com/dall-e-2/)

- Vision Transformer ([paper](https://arxiv.org/pdf/2010.11929.pdf)
- LiT paper (https://arxiv.org/pdf/2111.07991.pdf)

Contrastive Learning Objective (skipped)

CLiP used a Contrastive Learning Objective

- find embeddings to
  - maximize similarity between related examples (e.g., image and the text that describes it)
  - minimize similarity between unrelated examples (e.g., image and text that *does not* describe it).

CLiP used Binary Cross Entropy to meet the objective; we study a different approach.

Creating Embeddings for Similarity (Embeddings_similarity.ipynb)

Social Concerns

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)

Assignments

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

Final Project

Assignment notebook (assignments/FineTuning_HF/FineTune_FinancialPhraseBank.ipynb)

Additional Deep Learning resources

Intro to ML course (my section, previous semester)

For those of you who did not take my section of the Intro to ML course

- what was meant to be "review" may actually be "preview"
- I suggest
    - checking out the [Github repo (https://github.com/kenperry-public/ML_Fall_2023)](https://github.com/kenperry-public/ML_Fall_2023) for the Intro course of the semester just completed
    - The "landing page" for the course is the notebook `Index.ipynb`
    - Use that notebook to navigate the course
    - The Deep Learning part of the course starts at the end of Week 7

External resouces

Here are some resources that I have found very useful.

Some of them are very nitty-gritty, deep-in-the-weeds (even the "introductory" courses)

- For example: let's make believe PyTorch (or Keras/TensorFlow) didn't exists; let's invent Deep Learning without it !
    - You will gain a deeper appreciation and understanding by re-inventing that which you take for granted

[Andrej Karpathy course: Neural Networks, Zero to Hero (https://karpathy.ai/zero-to-hero.html)](https://karpathy.ai/zero-to-hero.html)

- PyTorch
- Introductory, but at a very deep level of understanding
    - you will get very deep into the weeds (hand-coding gradients !) but develop a deeper appreciation

fast.ai

`fast.ai` is a web-site with free courses from Jeremy Howard.

- PyTorch
- Introductory and courses "for coders"
- Same courses offered every few years, but sufficiently different so as to make it worthwhile to repeat the course !
    - Practical Deep Learning (https://course.fast.ai/)
    - Stable diffusion (https://course.fast.ai/Lessons/part2.html)
        - Very detailed, nitty-gritty details (like Karpathy) that will give you a deeper appreciation

Stefan Jansen: Machine Learning for Trading (https://github.com/stefan-jansen/machine-learning-for-trading)

An excellent github repo with notebooks

- using Deep Learning for trading

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

```
Done
```