## Overview

A Deep Learning model

- is a combination of components
- each component parameterized
- is fit (finding optimal values for parameters) based on a training dataset
- by optimizing a Loss Function

In the Intro course

- things were "small": datasets, number of parameters
- components were assembled in a simple but rigid order: sequence of Layers (the "Sequential" architecture)
- Loss functions were few and pre-specified: Mean Squared Error, Cross Entropy

In the Advanced course we depart from the simplifications of the Intro course

- How to use Big Data in Small Memory
- Deal with models with parameters numbering in the billions
- Introduce an unrestricted (technically: Directed Acyclic Graph) architecture for organizing components (the "Functional" model)
- Re-using advanced models that are too big to construct ourselves

## Most importantly

Loss Functions are complex

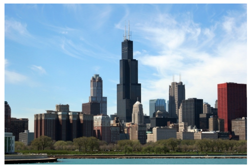
- less "pure math" (e.g., Mean Squared Error)
- More: a mathematical formulation that captures the "semantics" of the task

Here are some non-trivial tasks characterized by these more complex Loss functions

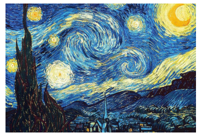
## Tasks with interesting Loss Functions

**Synthetic Data: Create a timeseries of a cross section of returns of US equities**

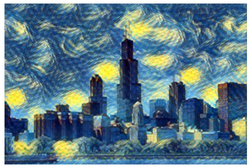
**Neural Style Transfer: Transform a photo into a painting in the style of Van Gogh**



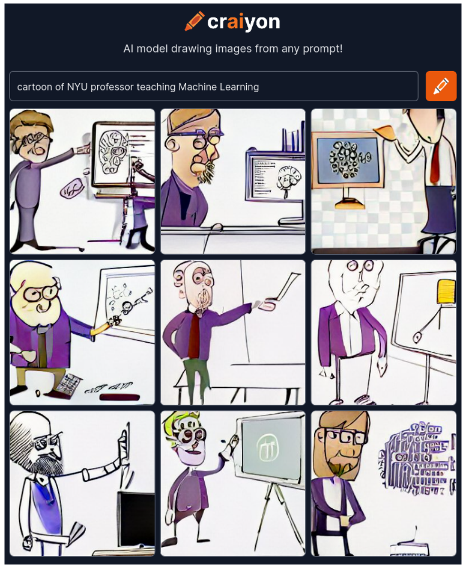
+



=



**Text to image: create cartoon of NYU professor teaching Machine Learning**



cartoon of NYU professor teaching Machine Learning

By the end of this course

- You will now how to solve each of these tasks
- Will be surprised by the relative ease

## Major Themes

- Advanced Models
    - Technical background
    - Novel architectures
- Modern Transfer Learning
    - Unsupervised Pre-training with Supervised Fine-Tuning
- Generative AI
    - emphasis on Synthetic Data

## Theme - Advanced Models

We focus mostly on *concepts* but their utility is not maximized without *technical proficiency*.

- being able to read code
    - eliminates ambiguity
- enables you to
    - modify and adapt existing models
    - create new models

## Technical

This is (to me) the least interesting, but necessary, part of the course

- technical prerequisite for *understanding* and *implementing* state of the art models
- necessary to understand the code behind the model

Technical: so that **you** can code and use these models

- How to build Functional models
- How to use Big Data: the Dataset API

## Novel architectures

Conceptual: State of the art models.

We will understand, analyze, and use the "AI" that is breathlessly reported in the popular press.

- ChatGPT
- DALL-E, Stable Diffusion

Some highlights

- Transformers
    - architecture that forms the basis of many of the most advanced models
- Autoencoders, Generative Adversarial Networks

## Theme - Modern Transfer Learning

This theme will be motivated by recent advances in Natural Language Processing: *Large Language Models (LLM)*

- but is applicable for other tasks as well

The problem:

- Models are getting so big
- That it is *impractical* for individuals/small organizations to compete with better-endowed players

GPT is a family of Large Language Models.

These models have recently captured the popular imagination (e.g., ChatGPT).

GPT-3 is a newer member of the family

- 175 billion parameters
- trained on vast quantities of data

For the most part: the techniques have been published and are well-known.

Can you train your own GPT-3 ?

**Cost of Training GPT-3 on your own**

- Amazon Cloud G5 instance
- NVidia A10G Tensor Core GPUs @ 250 Tflops/GPU
- 8 GPU instance (2 Pflops) @$10/hour (with yearly contract; \$16/hour on-demand)
    - $240 per 2Pflops-day

Training GPT-3 takes $\approx$ 3000 Pflop-days

- $3000/2 = 1500$ days G5 instances to get 3000 Pflops-days
- Cost = 1500 * $240/day = \$360K

How much does a typo in your code cost !

Fortunately: pre-trained versions of these large models are often published

- Model hubs/Model Zoos

You can "fine-tune" these costly models (developed for broad tasks) to your *narrow* tasks

- Unsupervised Pre-training with Supervised Fine-Tuning
    - Fine-tune on small number of narrow task-specific examples

We will focus on the "Model Hub" from Hugging Face (https://huggingface.co)

- located just down the road !

There is another way of "re-using" costly pre-trained models

*Zero Shot Learning*

- is a method allowing a model trained for one task
- to solve other tasks
- *without* being trained on the other tasks
- *often* with **no coding**

For example, GPT (and its relatives) can form the basis of "no programming" new apps

- exploiting Zero Shot Learning
- *Prompt engineering*: specially engineered "prompts"
    - often pre-pended to the feature vector for the new task to be solved

Some new tasks that can be derived by using GPT and Zero Shot Learning

- Summarize an article
- Write an article !
- ChatGPT

## Theme - Generative AI

In the Intro course, many of our tasks were *discriminative*

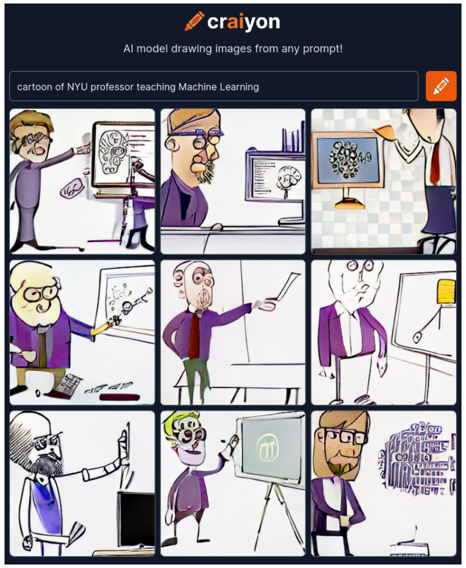
- learning a relationship between features and targets
    - Classification: discriminating among a finite number of possible target labels
    - Regression: continuous target

Many of the tasks we will address in this course (including many listed above) are *generative*

- Learning a distribution of feature vectors
    - Answer is a sample from the learned distribution
- Output is often structured, rather than label or numeric
    - A block of text
    - An image

## Many Generative Models we will study are based on Large Language Models

- text input to text output
- text input to image output
    - DALL-E
    - Stable Diffusion



cartoon of NYU professor teaching Machine Learning

A different (but interesting for Finance) use of Generative AI

- Creating synthetic training examples

Large models (many parameters) need lots of training data

- Finance data (particularly at daily frequency) just not big enough

Being able to create synthetic examples

- facilitates large models for Finance
- is a way of dealing with *class imbalance*
    - lots of the interesting Finance phenomena are rare
        - Risk Management

## Organization

The themes are *not orthogonal*: many are linked

- a Novel Architecture (the Transformer) is the basis of many models used for Transfer Learning
- we need the Technical tools to understand the code (and to build) Advanced Models

We also want to front-load the Technical tools presentation so that you may be started on the Project.

So we will probably wind up weaving back and forth between topics.

The goal

- to give you the knowledge and tools
- and intellectual background and curiousity

to participate (and maybe lead) advances in ML and Finance.

You will need both a solid conceptual basis and technical coding skills

**Concepts** come first. For example

- Background motivating the Transformer
- The Transformer in concept

**Code** comes second. For example

- we will examine the code behind a Transformer implementation

## Course organization

We will be dealing with a lot of new concepts

- many introduced only in the past 1-3 years

The lectures will cover the *most important* points

- function of limited time
- *real* understanding and proficiency will come from reading the papers

## Academic papers are dense and require effort to understand

Academic papers have a particular style

- more science and math
- as opposed to engineering
- assume the reader has substantial background

This makes them a bit dense and requires effort to grasp.

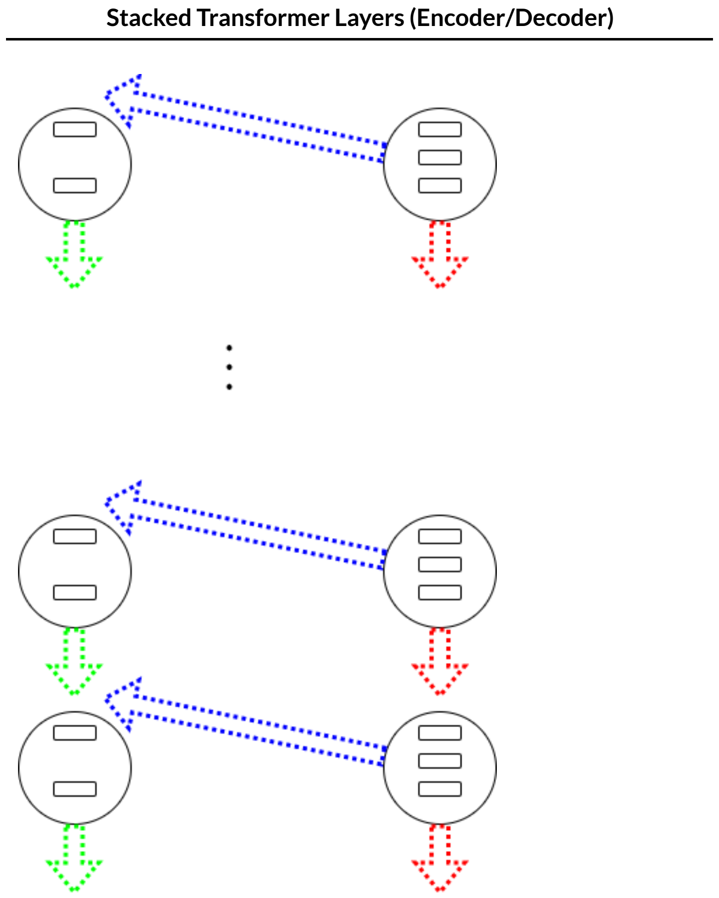
- lots of discussion and analysis
- almost no code

For example, we take a look at one of the early and important papers (https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf) on Large Language Models.

- The architecture is described (https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf#page=3) a *multi-layer Transformer Decoder*
    - reference to existing concept
    - which is *clarified* via a set of recursive equations

$$\begin{aligned} h_0 &= UW_e + W_p \\ h_i &= \text{transformer\_block}(h_{i-1}) \\ & \qquad\qquad\qquad\qquad \text{for } 1 \le i \le n \\ p(U) &= \text{softmax}(h_n W_e^T) \end{aligned}$$

In the Intro course, I would likely explain this via a diagram



And I would likely add explanation to the equations

$$
\begin{array}{lll}
h_0 & = U W_e + W_p & \text{concatenate Input Embedding and Positi} \\
h_i & = \text{transformer\_block}(h_{i-1}) & \text{connect output of layer } (i-1) \text{ to input o} \\
 & & \text{for } 1 \le i \le n \\
p(U) & = \text{softmax}(h_n W_e^T) & \text{Final output is probability distribution o} \\
 & & h_n \text{ is output of top transformer block} \\
 & & h_n W_e^T \text{reverses the embedding to obtain t}
\end{array}
$$

where

| | |
|---|---|
| $U$ | context of size $k : [u_{-k}, \ldots, u_{-1}]$ |
| $W_e$ | token embedding matrix |
| $W_p$ | position encoding matrix |
| $h_i$ | Output of transformer block $i$ |
| $n$ | number of transformer blocks/layers |

The model "details" are given in a singe dense paragraph (https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf#page=5)

- full of references to other concepts

And the optimization objective is described as maximizing

$$\mathcal{L}_2(\mathcal{C}) = \sum_{(\mathbf{x},\mathbf{y})} \log p(\mathbf{y}|\mathbf{x}_1, \ldots, \mathbf{x}_m)$$

- where $\mathcal{C}$ is the training dataset

The experienced reader (the intended audience for the author) will recognize

- the sum of logs
- as log-likehood

and immediately understand that the optimization is maximization of log-likelihood.

In the Introductory course

- I would have described the objective as *Cross Entropy*
    - which helps to relate it to the code needed for training
- and would have helpfully pointed out

$$\mathbf{y} = \mathrm{softmax}(h_l^m W_y)$$

to connect the mathematical concept (target $\mathbf{y}$) to the *output of the final layer* of the multi-layer Transformer Decoder.

Be patient: the effort will be rewarded

- faster to read a dense paper
- deeper understanding

## Academic papers focus on results and analysis rather than code

In our limited time, we focus on concepts and code.

But academic papers spend the bulk of their text on *analysis and discussion*.

- lots of experimental results
- discussion of results

These are really important to those of you interested in deeper understanding of the field

- but less important to those of you with an engineering focus

One common concept seen often is the *abltation study*

- The present paper has introduced one or more novel concepts or features
- Remove them one at a time (*ablation*)
    - to try to understand the relative importance of each novel concept

## The HuggingFace course

There will be a Final Project which involves coding.

You will need to understand

- Transfer Learning
- Transformers
- Natural Language Processing concepts
- Keras code to implement all of the above

In our lectures

- you will learn all the necessary concepts
- be introduced to the coding techniques

We will be using HuggingFace (https://https://huggingface.co/) as our "model hub"

- the source of pre-trained models
- that we will adapt (fine-tune) to solve a particular task
    - e.g., the Final Project

I *strongly suggest* that you follow the excellent HuggingFace course (https://huggingface.co/)

- reinforces the concepts
- but *emphasizes the coding* that you will need for the Final Project
    - worth being familiar with their Hub by the middle of this course

There are no assignments (other than the Final Project) in this course

- you can consider the time on the HuggingFace course as being time spent in lieu of homework

The HuggingFace course contains code in *either* Keras or Pytorch

- we will use Keras in this course
- there is a toggle (upper right) on most HuggingFace pages to switch between Keras and Pytorch
- some advanced features (e.g., the Trainer) are not (yet) available in Keras
    - but are not necessary for our course (convenience rather than necessity)

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

```
Done
```