

# 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 \(https://github.com/kenperry-public/ML\\_Spring\\_2023\)](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 \(https://arxiv.org/pdf/1706.03762.pdf\)](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
  - [Neural Machine Translation by Jointly Learning To Align and Translate \(https://arxiv.org/pdf/1409.0473.pdf\)](https://arxiv.org/pdf/1409.0473.pdf)
  - Geron Chapter 16
  - [An Analysis of BERT's Attention \(https://arxiv.org/pdf/1906.04341.pdf\)](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: customizing 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](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 \(Computation\\_Graphs.ipynb\)](#)
- [Fager vs Graph Execution \(TF\\_Graph.invnb\)](#)

# 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) (<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 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 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 \(HF quick intro to models.ipynb\)](#)

### **Suggested reading**

[HuggingFace course \(https://huggingface.co/course\)](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 \(https://arxiv.org/pdf/2001.08361.pdf\)](https://arxiv.org/pdf/2001.08361.pdf)

## Further reading

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



## 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 Autoencoder \(VAE\) \(VAE Generative.ipynb\)](#)

## **Suggested Reading**

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

## **Further reading**

[Tutorial on VAE \(https://arxiv.org/pdf/1606.05908.pdf\)](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 \(https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/dcgan/index.ipynb\)](#)
    - 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





# Week 6

GANs, continued

## Synthetic Data: GANs for Timeseries

Timeseries data

- [Time GAN \(TimeGAN Generative.ipynb\)](#)

Suggested reading

- [TimeGAN](#)  
(<https://proceedings.neurips.cc/paper/2019/file/c9efe5f26cd17ba6216bbe2a7d26Paper.pdf>)
  - [supplement \(https://www.vanderschaarlab.com/papers/NIPS2019\\_TGAN\\_Supplementary.pdf\)](https://www.vanderschaarlab.com/papers/NIPS2019_TGAN_Supplementary.pdf)
  - [github](https://github.com/vanderschaarlab/mlforhealthlabpub/tree/main/alg/tim)  
(<https://github.com/vanderschaarlab/mlforhealthlabpub/tree/main/alg/tim>)

Further reading

[Quant GAN \(https://arxiv.org/pdf/1907.06673.pdf\)](https://arxiv.org/pdf/1907.06673.pdf)

- Similar goal as TimeGAN
  - Uses *Temporal Convolution Networks (TCN)* for Generator and Discriminator
    - Is Dilated Convolution
    - effectively creates a longer window
  - [Quant GAN used for Risk Management: lecture slides](https://cfe.columbia.edu/sites/default/files/content/slides/Modelling%20/)  
(<https://cfe.columbia.edu/sites/default/files/content/slides/Modelling%20/>)

## Synthetic Data: Evaluating the quality

[Synthetic data: Evaluation \(SyntheticData\\_Evaluation.ipynb\)](#)

- [case study: Evaluation of Time GAN \(TimeGAN\\_evaluation.ipynb\)](#)

### **Further reading**

[Frechet Inception Distance \(FID\) \(https://arxiv.org/pdf/1706.08500.pdf#page=39\)](#)

# 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)  
([https://cdn.openai.com/papers/Learning\\_Transferable\\_Visual\\_Models\\_From\\_Natur](https://cdn.openai.com/papers/Learning_Transferable_Visual_Models_From_Natur);
- [DALL-E paper](https://arxiv.org/pdf/2102.12092.pdf) (<https://arxiv.org/pdf/2102.12092.pdf>),
- [OpenAI DALL-E 2 announcement](https://openai.com/dall-e-2/) (<https://openai.com/dall-e-2/>),
- [Vision Transformer](https://arxiv.org/pdf/2010.11929.pdf) ([[paper](https://arxiv.org/pdf/2010.11929.pdf)])(<https://arxiv.org/pdf/2010.11929.pdf>),
- [LiT paper](https://arxiv.org/pdf/2111.07991.pdf) (<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 \(Embeddings\\_similarity.ipynb\)](#)



## 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 \(https://arxiv.org/pdf/2205.01068.pdf\)](https://arxiv.org/pdf/2205.01068.pdf)

# Switch Transformer/Mesh TensorFlow

The Scaling laws suggest the advantage of larger models but at the cost of increased computation.

- Is it possible to increase the number of parameters without increasing the computation ?
- How do we *physically* handle
  - extremely large number of examples
  - parameter tensors that are too big to fit on a single processor
- How do we *logically* divide a large computational/memory load
  - using simple high-level abstractions
- [Scaling to 1 trillion parameters \(Switch Transformer.ipynb\)](#)
- [Mesh-TensorFlow \(Mesh\\_TensorFlow.ipynb\)](#)

## Suggested reading

- [Switch Transformer \(https://arxiv.org/pdf/2101.03961.pdf\)](https://arxiv.org/pdf/2101.03961.pdf)

## Further reading

- [Towards Understanding Mixture of Experts in Deep Learning \(https://arxiv.org/pdf/2208.02813.pdf\)](https://arxiv.org/pdf/2208.02813.pdf)
- [Mesh TensorFlow paper \(https://arxiv.org/pdf/1811.02084.pdf\)](https://arxiv.org/pdf/1811.02084.pdf)
- [Mesh TensorFlow github \(https://github.com/tensorflow/mesh#mesh-tensorflow--model-parallelism-made-easier\)](https://github.com/tensorflow/mesh#mesh-tensorflow--model-parallelism-made-easier)





# Week 7

## Stop-gradient operator and Straight-through Estimator

How do we handle operations that are **not** differentiable ?

- Quantization in the VQ-VAE
- Routing in the Mixture of Experts

There is a special operator in Keras that helps us with the task.

- [Quantization in the VQ-VAE: dealing with non-differentiable operations \(VQ\\_VAE\\_Generative.ipynb#Quantization-is-not-differentiable\)](#)

## Combining a LLM with external capabilities

The module on Scaling argued that perhaps some models have *too many* parameters (given the size of the training dataset). Many model parameters probably encode "world knowledge" (facts) rather than "task knowledge" (how to solve a task).

What if we allow world knowledge to be acquired at *inference time* by connecting the model to an external knowledge source (e.g., the Web) ? The result would be smaller models with more current knowledge.

- [Extra parametric capabilities \(LLM\\_plus\\_Extra\\_Parametric.ipynb\)](#)

### Suggested reading

- [Retriever-Generator paper \(https://arxiv.org/pdf/2005.11401.pdf\)](https://arxiv.org/pdf/2005.11401.pdf)
- [MRKL Systems \(https://arxiv.org/pdf/2205.00445.pdf\)](https://arxiv.org/pdf/2205.00445.pdf)
- [Program Aided Language Model \(https://arxiv.org/pdf/2211.10435.pdf\)](https://arxiv.org/pdf/2211.10435.pdf)

# Retriever-Generator examples: WebGPT, RETRO

- [Non-parametric knowledge \(Retriever\\_plus\\_LLM.ipynb\)](#)

## Suggested reading

- [WebGPT \(https://arxiv.org/abs/2112.09332\)](https://arxiv.org/abs/2112.09332)

## Further reading

# Prompt Engineering

We can adapt the behavior of a LLM trained on the text-continuation task ("predict the next") to solve a different Target Task

- by crafting a text "prompt" that conditions the LLM to produce output consistent with the Target Task

Is the exact wording of the prompt important ? That is what *Prompt Engineering* is about.

The methods of improving performance by care prompt crafting vary from

- suggestions to users, based on anecdotal evidence
- empirical measurements
- machine automated prompt improvement
- optimization via Loss minimization

In this module, we explore the various methods.

- [Prompt Engineering Overview \(Prompt\\_Engineering.ipynb\)](#)
- [Using an LLM as a Prompt Engineer \(Prompt\\_Engineering\\_APE.ipynb\)](#)
  - module also referenced in [Synthetic Data: Instruction Following \(LLM\\_Instruction\\_Following\\_Synthetic\\_Data.ipynb\)](#)
- Optimizing the prompt by minimizing the Loss
  - [Prompt Engineering: 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 \(In\\_Context\\_Learning\\_Theory.ipynb\)](#)

Can humans even write good prompts ? Will they need to ?

- [Automatic "upsampling" of human text prompts for Image creation \(Prompt\\_Engineering\\_Text\\_Upsampling.ipynb\)](#)

Some theories behind prompt engineering

- [Prompt Engineering suggestions \(Prompt\\_Engineering\\_Suggestions.ipynb\)](#)

### Suggested reading

- [How does in-context learning work ? summary of paper \(http://ai.stanford.edu/blog/understanding-incontext/\)](http://ai.stanford.edu/blog/understanding-incontext/)
- [LLMs are Human-Level Prompt Engineers: summary \(https://sites.google.com/view/automatic-prompt-engineer\)](https://sites.google.com/view/automatic-prompt-engineer)
- [Prompt Tuning, blog \(https://ai.googleblog.com/2022/02/guiding-frozen-language-models-with.html\)](https://ai.googleblog.com/2022/02/guiding-frozen-language-models-with.html)
- [Prefix-Tuning: Optimizing Continuous Prompts for Generation \(https://arxiv.org/pdf/2101.00190.pdf\)](https://arxiv.org/pdf/2101.00190.pdf)
- [APE: summary \(https://sites.google.com/view/automatic-prompt-engineer\)](https://sites.google.com/view/automatic-prompt-engineer)

# 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/abs/1410.0275](#)).

# Interpretation

As we've hinted at before: the art of Deep Learning is writing a Loss function that captures the semantics of the problem you want to solve.

- We've spent several weeks learning about Neural Networks. At best, we suggested theories for how they are able to achieve what they do.

This week: we will explore methods for testing theories as to how Neural Networks work.

## What is a Neural Network really doing ? Interpretation

- [Interpretation: preview \(Interpretation\\_preview.ipynb\)](#)
- [Introduction to Interpretation of Deep Learning \(Intro to Interpretation of DL.ipynb\)](#)
- [Interpretation: Simple Methods \(Interpretation of DL Simple.ipynb\)](#)
- [Interpretation: Saliency Maps \(Interpretation of DL Deconv.ipynb\)](#)
- [Interpretation: Gradient Ascent \(Interpretation of DL Gradient Ascent.ipynb\)](#)
- [Adversarial Examples \(Adversarial Examples.ipynb\)](#)

# 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 \(https://mlco2.github.io/impact/#compute\)](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:  $5 \cdot 10^{15}$  Flops
    - 3600 seconds/hour
    - $\frac{10^{23}}{3600 \cdot 5 \cdot 10^{15}} \approx 28000$  hours to train on single A100
    - Calculated CO<sub>2</sub> equivalent:  $1.7 \cdot 10^4$  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 \(https://arxiv.org/pdf/2005.14165.pdf#page=39\)](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 \(Alignment\\_Anthropic.ipynb\)](#)

# 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

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 exist; 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 \(https://course.fast.ai/\)](https://course.fast.ai/)
  - [Stable diffusion \(https://course.fast.ai/Lessons/part2.html\)](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

→ [Machine Deep Learning for trading](#)

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

Done

