Data Mining - Deep Learning, Transformers, and Transfer Learning

Joe Burdis
Fall 2024
CUNY Graduate Center

Outline

1. HW2 discussion
2. Week 6 recap
3. Deep Learning
   a. CNNs (for computer vision)
   b. Audio
   c. Transformers for NLP
4. using BERT (or BERT-like models)
   a. https://huggingface.co/docs/transformers/notebooks
   b. https://colab.research.google.com/github/tensorflow/text/blob/master/docs/tutorials/classify_text_with_bert.ipynb
5. General discussion of transfer learning and fine-tuning

Docker Tutorial

For next week:

https://docs.docker.com/language/python/build-images/

BM25 vs. Embedding Models

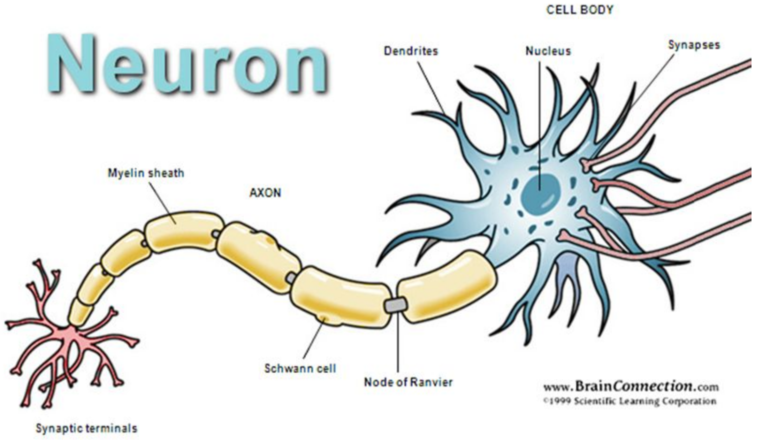
- BM25: Bag-of-words model, term frequency focus.
- Embeddings: Dense vector representations capturing semantic meaning.
- BM25: Efficient for large-scale search, interpretable.
- Embeddings: Better at capturing context and synonyms.
- BM25: Requires careful tuning of parameters (defaults aren’t bad though)
- Embeddings: Pre-trained models available, reducing training needs.
- Hybrid approaches can leverage strengths of both.

The Biological Neuron

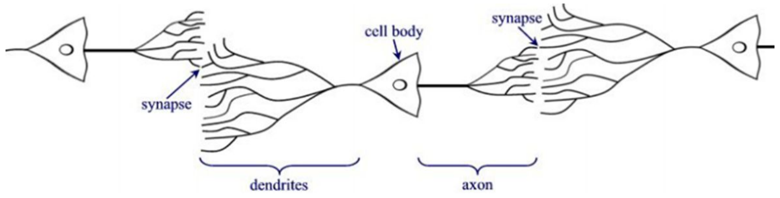
The behavior occurs at the membrane between the inside and outside of the cell.

- Dendrites: appendages that are designed to receive communications via impulses from other cells and transmit them to the cell body.
- Soma (or cell body): combines the impulses receive and triggers action on the axons.
- Axons: the long thread-like part of a neuron that impulses are sent to other cells.

When the synapse releases neurotransmitters, it opens some sodium channels. If ENOUGH channels are opened, a region of the membrane will depolarize. **Action potential! So whether an AP occurs or not is a sum!**



Neuron Networks:



(Artificial) Neural Networks

The initial neural network is often cited as Rosenblatt's Perceptron for binary classification which consisted of input values, weights (and bias), a weighted sum, and activation function. "Perceptron" and "neuron" are often used interchangeably, but "perceptron" refers so something more specific (so use "neuron").

Some basic features of Neural Networks

1. Each activation is differentiable (usually nonlinear)
2. Contains hidden layers (non-input or output layer)
3. High degree of connectivity (E.g. "Fully Dense")

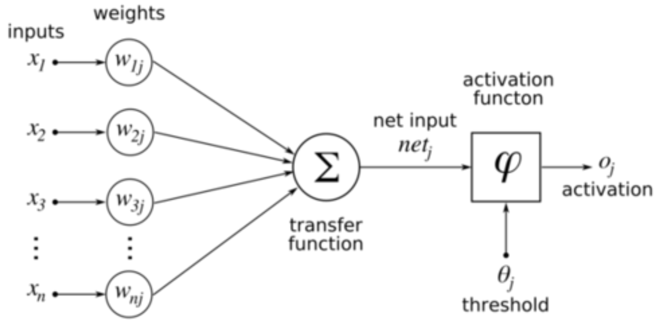
Note: "deep" refers to more than 1 hidden layer

Activation function for neuron:

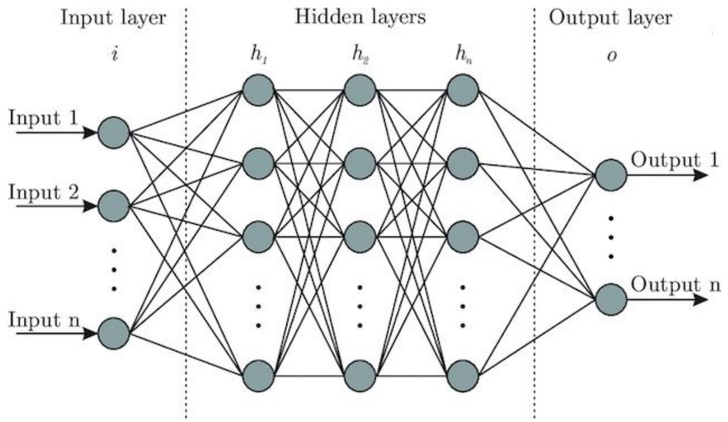
$$h_{11}(X) = \phi\left(\underbrace{W_{11}^T X + b_{11}}_{1\times 1}\right)$$

$$\phi : \mathbb{R} \to \mathbb{R}$$

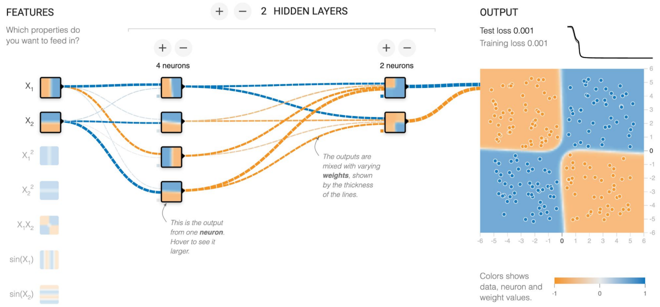
Rosenblatt's Perceptron:



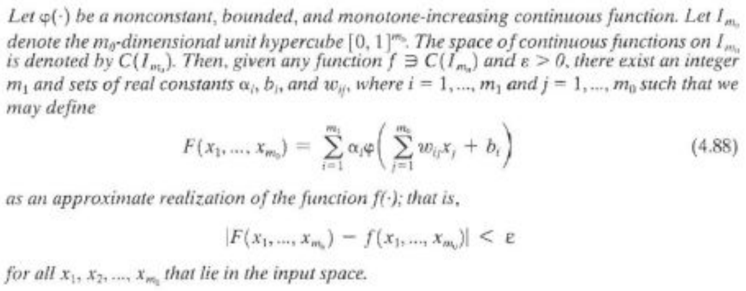
Neural Network:



Universal Approximation Theorem


FEATURES
Which properties do you want to feed in?
+ − 2 HIDDEN LAYERS
+ − 4 neurons
+ − 2 neurons
OUTPUT
Test loss 0.001
Training loss 0.001
X_1
X_2
X_1^2
X_2^2
X_1X_2
$\sin(X_1)$
$\sin(X_2)$
The outputs are mixed with varying weights, shown by the thickness of the lines.
This is the output from one neuron. Hover to see it larger.
Colors shows data, neuron and weight values.
-1 0 1

"A neural network with one hidden layer containing a sufficient but finite number of neurons can approximate any continuous function to a reasonable accuracy, under certain conditions for activation functions"



Let $\varphi(\cdot)$ be a nonconstant, bounded, and monotone-increasing continuous function. Let I_{m_0} denote the m_0-dimensional unit hypercube $[0, 1]^{m_0}$. The space of continuous functions on I_{m_0} is denoted by $C(I_{m_0})$. Then, given any function $f \ni C(I_{m_0})$ and $\varepsilon > 0$, there exist an integer m_1 and sets of real constants α_i, b_i, and w_{ij}, where $i = 1, \ldots, m_1$ and $j = 1, \ldots, m_0$ such that we may define

$$F(x_1, \ldots, x_{m_0}) = \sum_{i=1}^{m_1} \alpha_i \varphi\left(\sum_{j=1}^{m_0} w_{ij} x_j + b_i \right) \qquad (4.88)$$

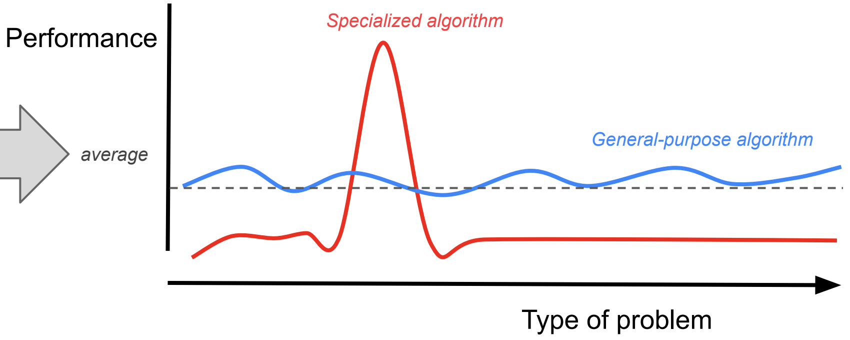
as an approximate realization of the function $f(\cdot)$; that is,

$$|F(x_1, \ldots, x_{m_0}) - f(x_1, \ldots, x_{m_0})| < \varepsilon$$

for all $x_1, x_2, \ldots, x_{m_0}$ that lie in the input space.

So just use neural networks for everything, right?

No!



There is No Free Lunch

What are Neural Networks used for?

Computer Vision (CV) / Image Processing

- Image classification
- Facial recognition
- Object detection
- Image (and semantic) segmentation
- Any model using images as inputs (text featurizers)

<Optical Character Recognition(OCR)>

Natural Language Processing (NLP)

- Translation
- Sentiment analysis
- Text summarization or extraction (NER)
- Topic modeling
- Any model using text as inputs (text featurizers)

Tabular Data

- Model using an arrangement of information or data, typically in rows and columns
- Historically only good for very large data and rare instances, but recent innovations have proved quite useful

Other

- Reinforcement Learning (DQN)
- Generative models
- Knowledge graphs
- Art, Music, Gaming
- Dimensionality reduction / compression

Why Neural Networks and why "deep"?

Pros:

- NNs can outperform other ML techniques on large and complex problems and are SOTA in many modalities (images, audio, text, etc.)
- The tremendous increase in data availability and computing power has made it possible to train large neural networks in a reasonable amount of time.
- The training algorithms have been improved
  - Activation functions: Relu
  - Heuristics: batch normalization and learning rate scheduling/annealing
  - Architectures: CNN, GRU/LSTM, Transformers/Attention, and others
- Some theoretical limitations of NNs have turned out to be benign in practice.

Cons:

- Huge number of parameters: slow to use, prone to overfitting.
- Black box model: hard to understand the learned function
- High Variance and lack of performance guarantee

Mitigations:

- **Proper experimentation techniques!!!**
- **Use good baselines!!!**
- Early stopping (careful / double descent)
- New explanation methods
  - Activations
  - Integrated gradients
  - Model agnostic methods / testing

Let's go through XOR on the board!

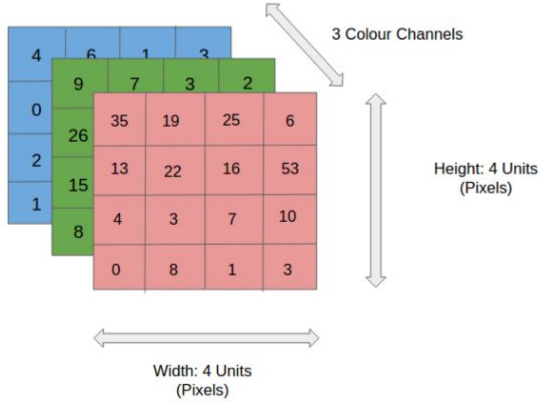
Backpropagation

1. **Initialization of weights (and biases):** Choose weights randomly(see initializers)
    a. Range of the weights should be determined so the induced fields of the first hidden layer are near the midrange of the activation function.
    b. **Note**: it is common to adjust the range of the input vectors have magnitudes of the input terms in the range [-1, 1] or standardized (aligns with adjustment of the initial weights and any scaling parameters)
2. **Forward computation / propagation (or pass)**: Present $\mathbf{x}_i$ to the input and d_i at the output to compute error.
3. **Backward computation / propagation:** Compute local gradients.

The math:
https://www.youtube.com/watch?v=tIeHLnjs5U8

The intuition:
https://www.youtube.com/watch?v=Ilg3gGewQ5U

Computer Vision

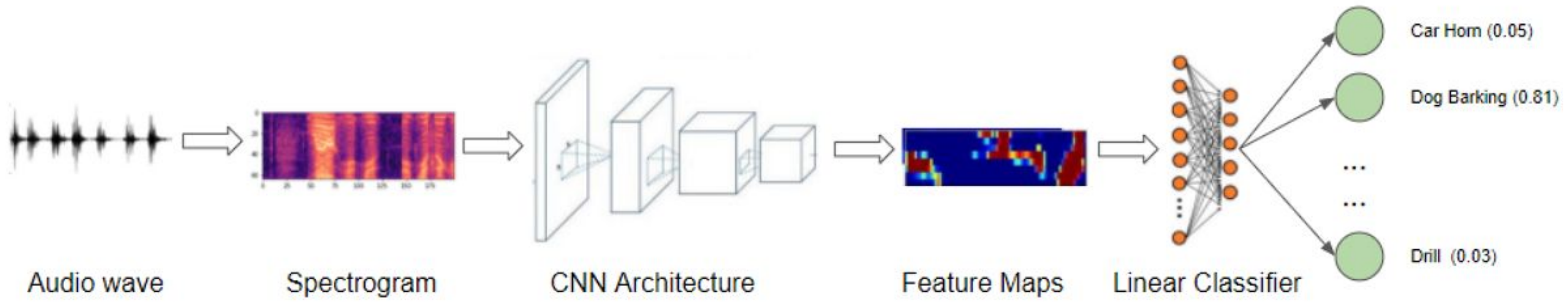


CNN Explainer: https://poloclub.github.io/cnn-explainer/

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

https://www.tensorflow.org/tutorials/images/transfer_learning

Audio Data



https://towardsdatascience.com/audio-deep-learning-made-simple-sound-classification-step-by-step-cebc936bbe5

Multi-Head Attention

Computing $e_1, \dots, e_n \in \mathbb{R}$ from $k_1, \dots, k_n \in \mathbb{R}^{d_1}$ and q $\in \mathbb{R}^{d_2}$

- Basic dot-product attention:

$$e_i = q^T k_i \ \in \mathbb{R}$$

  where $d_1 = d_2$

- Multiplicative attention:

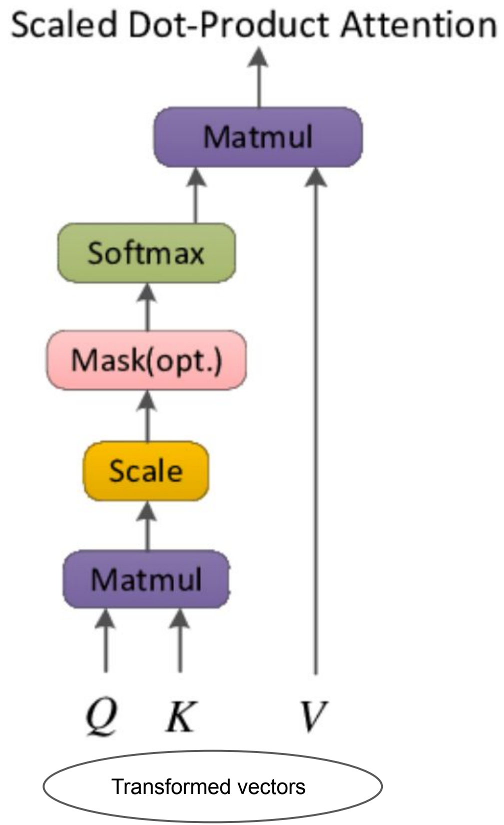
$$e_i = q^T W k_i \ \in \mathbb{R}$$

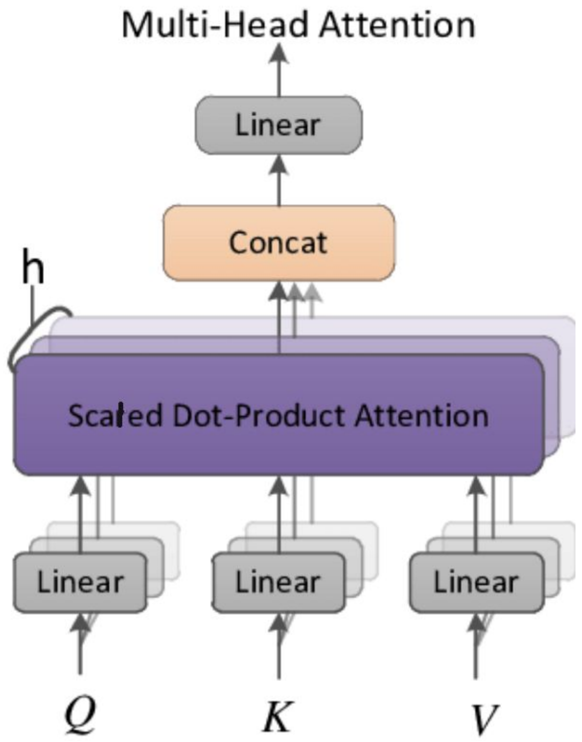
  where $W \in \mathbb{R}^{d_2 x d_1}$ is a weight matrix

- Additive attention:

$$e_i = {W_3}^T \tanh(W_1 k_i + W_2 q) \in \mathbb{R}$$

  where $W_1 \in \mathbb{R}^{d_3 x d_1}$, $W_2 \in \mathbb{R}^{d_3 x d_2}$ are weight matrices and $W_3 \in \mathbb{R}^{d_3}$ is a weight vector.



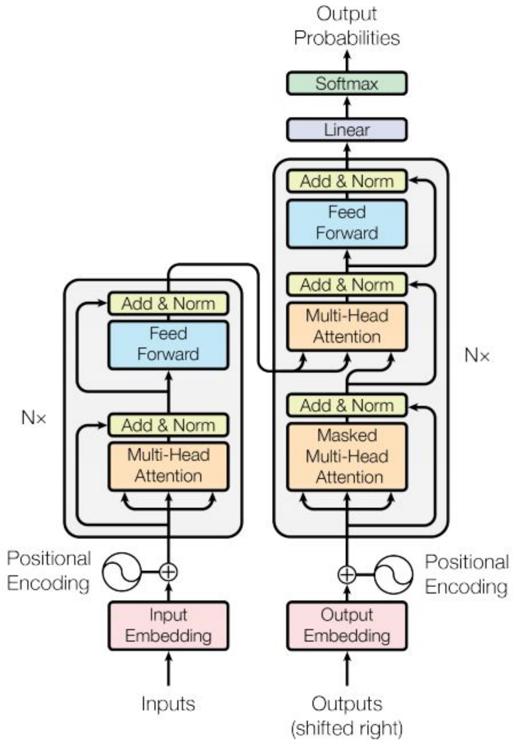


Transformer (used in NLP and other tasks)

Multiple heads and multiple transformer layer / blocks

Left: Encoder (BERT)

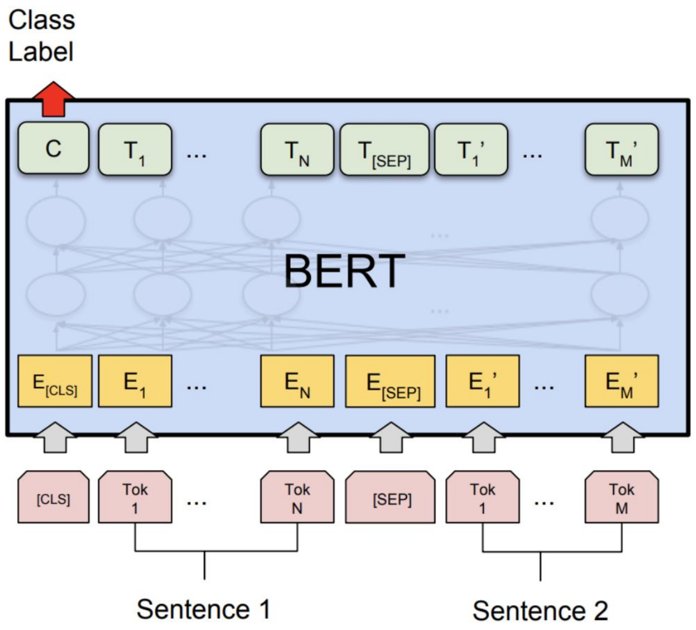
Right: Decoder (e.g. GPT)



https://arxiv.org/abs/1706.03762

NLP with BERT

Encode Only



https://pytorch.org/tutorials/intermediate/dynamic_quantization_bert_tutorial.html

https://colab.research.google.com/github/tensorflow/text/blob/master/docs/tutorials/classify_text_with_bert.ipynb

GPT Architecture

Decoder Only

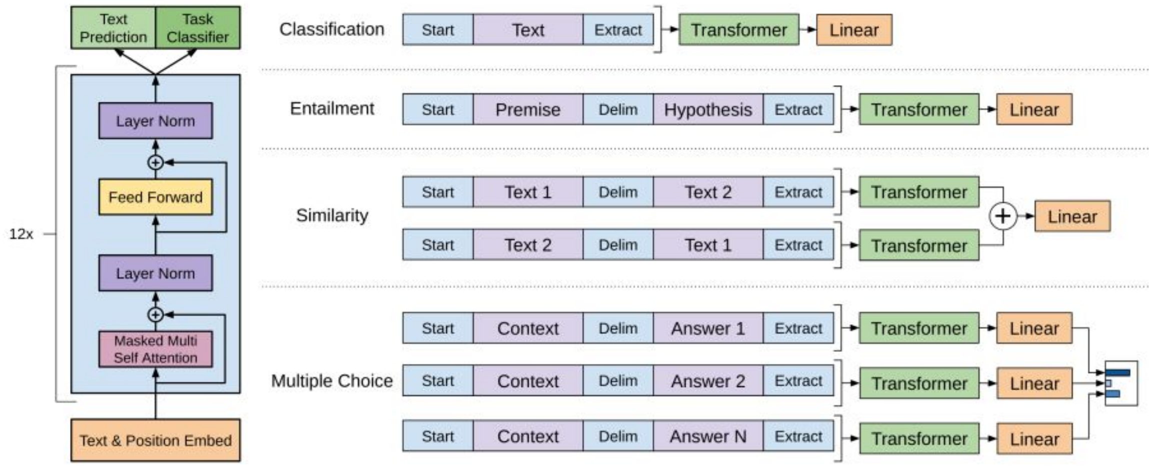
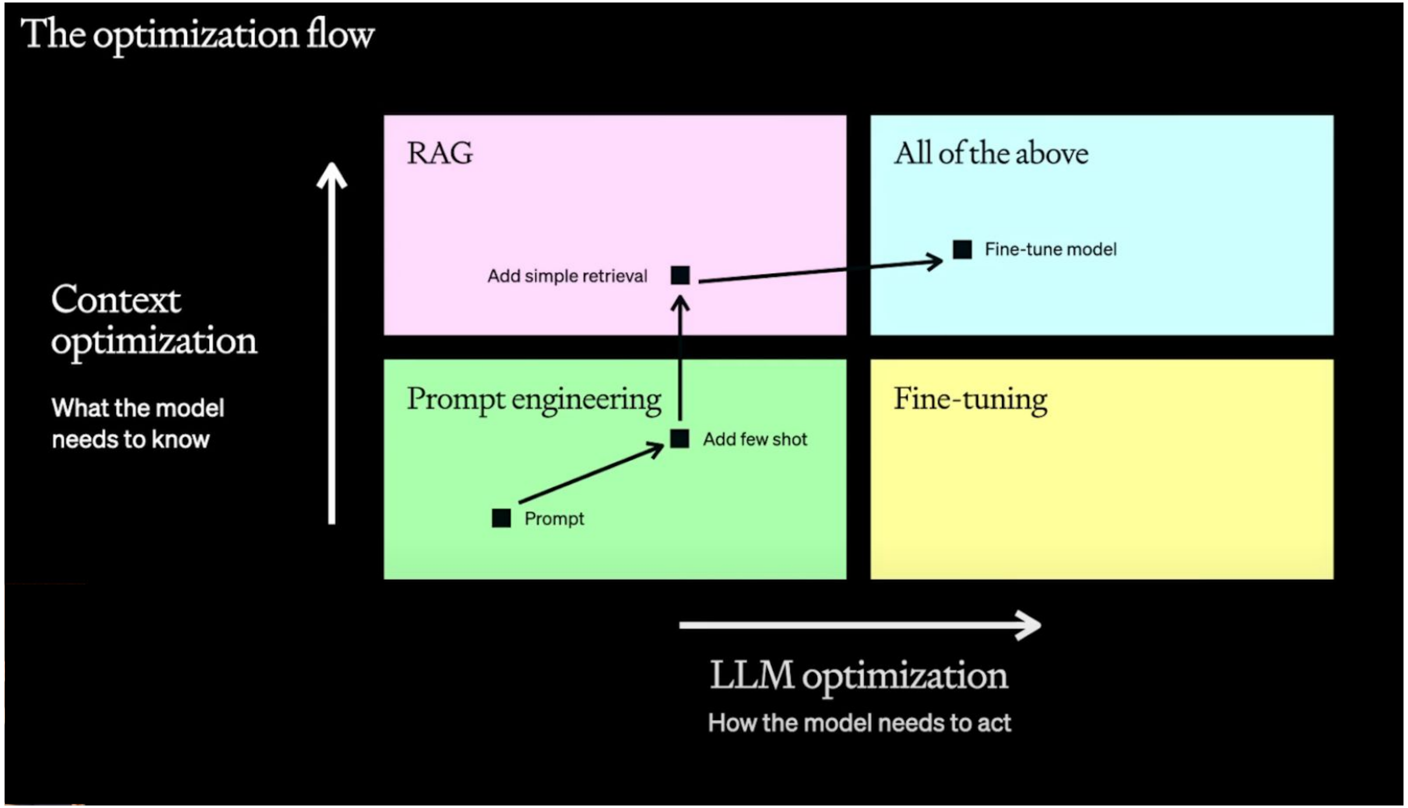


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

https://paperswithcode.com/paper/improving-language-understanding-by

Improving LLM Pipelines



https://www.youtube.com/watch?v=ahnGLM-RC1Y&t=1279s

TikTok Recommendation System

https://analyticsindiamag.com/tiktok-parent-bytedance-reveals-its-sota-recommendation-engine/

https://arxiv.org/pdf/2209.07663.pdf

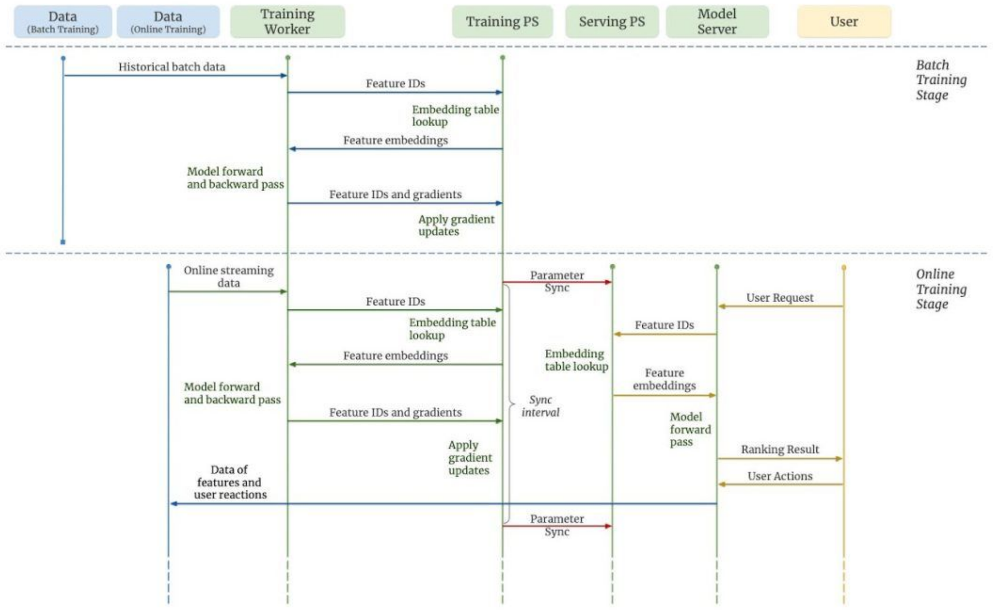
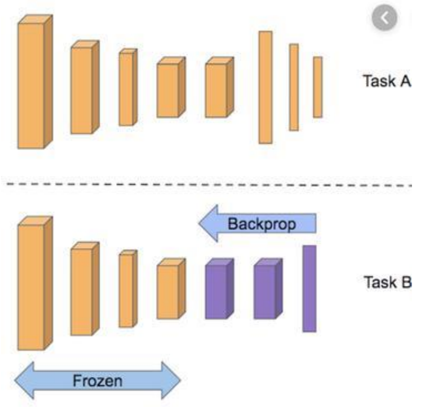


Figure 1: Monolith Online Training Architecture.

Transfer Learning

- One can think of deep learning models as learning a complex feature extractor and using that to predict a given target.
- **Issue**: In order to learn this feature extractor, we need lots of data to train from scratch.
- What about using the first X layers of a model trained on a task with lots of data and using those trained weights for another task?



(Image credit: Subodh Malgonde)

Transfer Learning

- You need less data - only need to train purple layers
- Typically works very well if new task in same domain as original task
- **Example**: ImageNet or other networks trained on ImageNet dataset may fail if using strict transfer learning to detect human emotions (no faces in dataset)
- **Solution**: Fine-tuning - i.e. the layers borrowed don't need to be frozen

CNN Example: https://www.tensorflow.org/tutorials/images/transfer_learning

NLP Example:
https://colab.research.google.com/github/tensorflow/text/blob/master/docs/tutorials/classify_text_with_bert.ipynb

Hugging Face: https://huggingface.co/docs/transformers/notebooks