AI Overview

MGMT 675: AI-Assisted Financial Analysis

RICE | BUSINESS
Jones Graduate School of Business

API Calls

Creating an App that Makes API Calls

- Get an API key from OpenAI or other LLM provider
- Generate code that sends a prompt to an AI and gets a response. API key should be included in code to authorize (for billing).
- Or create an interface that allows the user to formulate a prompt. Your code may expand the prompt to include whatever information you want to provide to the LLM.
- Or send your own prompt and allow users to send prompts.

Examples

- Julius prompt: build a Streamlit app that scrapes the headlines from *The Guardian* website, sends them to ChatGPT 4o using my API key, and returns a summary of the headlines and an assessment of the sentiment of the headlines.
  - After deployment: Streamlit app with API call built by Julius
- Replit prompt: same as Julius prompt + provide a chat interface that allows the user to ask further questions about the headlines.
  - Deployed directly from Replit: Replit app with API call
- Build apps with Zapier

Python code for API call

```python
import openai
# [input openai_api_key]
client = openai.OpenAI(api_key=openai_api_key)
prompt = "Here are today's headlines"
# [then include the scraped headlines]
prompt += "Based on these headlines, ..."
response = client.chat.completions.create(
    model="gpt-4o",
    messages=[
        {"role": "system", "content": "You are a
            helpful assistant that analyzes news
            headlines."},
        {"role": "user", "content": prompt}
    ]
)
```

API-based Apps vs AI Agents

From ChatGPT:

Feature	API-based App	AI Agent
Autonomy	✗ Only reacts	✓ Acts independently
Goal-Oriented	✗ Task-based	✓ Goal-driven
Adaptability	✗ Predefined logic	✓ Can adapt behavior
Tool Use	✓ Manual calls	✓ Chooses tools as needed
Memory	✗ Usually none	✓ May have memory
Intelligence Level	✗ Fixed logic	✓ Planning & reasoning

Some Chatbots

Chatbots

- ChatGPT, Gemini, Claude, Perplexity (try ChatGPT Deep Research)
- Custom GPTs
- Google NotebookLM
  - Notebook = collection of sources (text, images, etc.)
  - Can ask questions about the sources
  - Can create audio ("podcast") from the sources
  - Example: Stanford AI Report 2025
  - NotebookLM podcast version also at this link
- OpenRouter provides access to many LLMs. Use your own API keys for paid LLMs or use free LLMs.

Language Models

Tokenization, Embedding, and Prediction

1. Tokenization: break text into tokens (words, characters, or subwords). Example: unhappiness $\rightarrow$ [un, happy, ness]
2. Embedding: assign each token to a vector (list of numbers)

Tokenization + Embedding $\rightarrow$ text translated into sequence of vectors

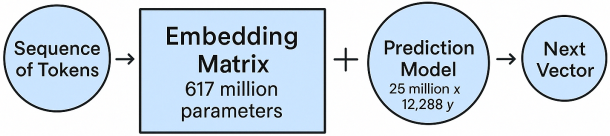
3. Prediction problem: given a sequence of vectors, predict the next vector

Embedding + Prediction optimized jointly by machine learning

Example: GPT-3

- Tokenization: use 50,257 tokens
- Embedding: each token assigned to a vector of 12,288 numbers.
  - 50,257 tokens $\times$ 12,288 numbers per token $=$ 617 million numbers (parameters)
- Prediction: Use sequence of 2,048 prior tokens to predict next token.
  - 2,048 tokens is 2,048 $\times$ 12,288 $=$ 25 million numbers
  - Used to predict next token, which is 12,288 numbers
  - So, predict y from x with 25 million x variables and 12,288 y variables (per observation)

Summary of GPT-3



GPT-3.5 Turbo Embeddings

- Can get vector embeddings of tokens from Chat GPT 3.5 Turbo by API calls
- Vectors are 1,536 numbers long
- Excel file with vector embeddings of king, queen, woman, and man from ChatGPT 3.5 Turbo
- Famous example: can add and subtract vectors and king $+$ woman $-$ man $\approx$ queen

Neural Networks (Prediction Model)

History of Neural Networks

1943: McCulloch & Pitts propose a binary threshold model of neurons.

1958: Perceptron introduced by Frank Rosenblatt

1986: Backpropagation popularized by Rumelhart, Hinton, & Williams enables training of multilayer networks.

1990s: Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) gain traction.

1997: Long Short-Term Memory (LSTM) addresses RNN vanishing gradient issues (Hochreiter & Schmidhuber).

2012: AlexNet wins ImageNet, marking deep learning's breakthrough.

2017: Transformers introduced — Vaswani et al. publish *Attention Is All You Need*, replacing recurrence with self-attention.

2020s: Transformer variants dominate NLP and spread to vision and multi-modal models. GPT = Generative Pre-trained Transformer.

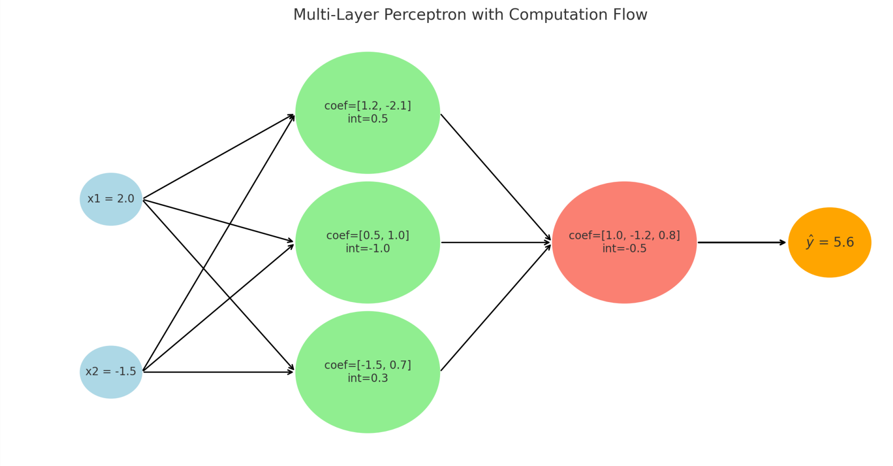
Multi-Layer Perceptrons

- A multi-layer perceptron (MLP) consists of "neurons" arranged in layers.
- A neuron is a mathematical function. It takes inputs $x_1, \ldots, x_n$, calculates a function $y = f(x_1, \ldots, x_n)$ and passes y to the neurons in the next level.
- Standard function (ReLU) for hidden layers is

$$y = \begin{cases} \alpha + \beta_1 x_1 + \cdots + \beta_n x_n & \text{if positive} \\ 0 & \text{otherwise} \end{cases}$$

  - First layer (input layer) = inputs (features).
  - "Hidden layers" take inputs from previous layer and pass output to next layer.
  - Last layer (output layer) has one neuron for each output.

Illustration



Hidden Layer Computations

Inputs $x_1 = 2.0$, $x_2 = -1.5$

Neuron 1 $\alpha = 0.5$, $\beta_1 = 1.2$, $\beta_2 = -2.1$

$$\begin{aligned} h_1 &= \text{ReLU}(0.5 + 1.2 \cdot 2.0 + (-2.1) \cdot (-1.5)) \\ &= \text{ReLU}(6.05) = 6.05 \end{aligned}$$

Neuron 2 $\alpha = -1.0$, $\beta_1 = 0.5$, $\beta_2 = 1.0$

$$\begin{aligned} h_2 &= \text{ReLU}(-1.0 + 0.5 \cdot 2.0 + 1.0 \cdot (-1.5)) \\ &= \text{ReLU}(-1.5) = 0 \end{aligned}$$

Neuron 3 $\alpha = 0.3$, $\beta_1 = -1.5$, $\beta_2 = 0.7$

$$\begin{aligned} h_3 &= \text{ReLU}(0.3 + (-1.5) \cdot 2.0 + 0.7 \cdot (-1.5)) \\ &= \text{ReLU}(-3.75) = 0 \end{aligned}$$

Output Computation

$\alpha = -0.5,\ \beta_1 = 1.0,\ \beta_2 = -1.2,\ \beta_3 = 0.8$

$$\begin{aligned} \hat{y} &= -0.5 + 1.0 \cdot h_1 + (-1.2) \cdot h_2 + 0.8 \cdot h_3 \\ &= -0.5 + 6.05 + 0 + 0 = \boxed{5.55} \end{aligned}$$