Workshop: Introduction to Large Language Models with Applications in Text Classification

Gradon Nicholls, PhD Student, University of Waterloo

May 28, 2026

Objectives

1. Gain exposure to some of the core concepts and terminology regarding neural networks and large language models.
2. Put the concepts into practice using python (hugging face and pytorch packages), with text classification as the motivating example.

Outline

1. Intro to Neural Networks
2. Intro to Large Language Models
3. Fine-tuning Pre-trained LLMs with Coding Example

A Note on Terminology

$$y_i = \beta_0 + \beta_1 x_{1i} + \cdots + \beta_p x_{pi} + \varepsilon_i$$

- y_i: outcome/dependent variable vs. **label**
- $x_{1i}, \ldots, x_{pi}$: independent variables vs. **features**
- i: observations vs. **examples**
- β_0: intercept vs. **bias**
- $\beta_1, \ldots, \beta_p$: coefficients vs. **weights**

Classification

- **supervised** learning: data includes the outcome y
- **classification:** supervised learning where the outcome variable is categorical
  - binary: e.g. yes/no, true/false
  - multi-class: e.g. positive/negative/neutral
  - multi-label: e.g. tagging a news article with topics (tech, sports, politics, etc.)
- **text classification:** classification where the inputs x are text
  - complication: how to convert x to numeric variables?
- examples of techniques for classification: logistic regression, k nearest neighbours, naive bayes, regression trees, support vector machines

As our motivating example, we will consider multi-class classification into one of C classes.

Neural Networks

Basic setup of machine learning:

- output y
- input(s) x
- truth: $y = f^*(x)$
- model: $y = f(x; \beta)$

Neural networks (NNs):

- also called feedforward NNs (FFN) or multi-layer perceptrons (MLP)
- $f(x; \beta) = \cdots f_3(f_2(f_1(x; \beta_1); \beta_2); \beta_3)$
- f_i: activation function for the ith layer

Neural Networks

Basic setup of machine learning:

- output y
- input(s) x
- truth: $y = f^*(x)$
- model: $y = f(x; \beta)$

Neural networks (NNs):

- also called feedforward NNs (FFN) or multi-layer perceptrons (MLP)
- $f(x; \beta) = \cdots f_3(f_2(f_1(x; \beta_1); \beta_2); \beta_3)$
- f_i: activation function for the ith layer

Neural Networks

Basic setup of machine learning:

- output y
- input(s) x
- truth: $y = f^*(x)$
- model: $y = f(x; \beta)$

Neural networks (NNs):

- also called feedforward NNs (FFN) or multi-layer perceptrons (MLP)
- $f(x; \beta) = \cdots f_3(f_2(f_1(x; \beta_1); \beta_2); \beta_3)$
- f_i: activation function for the ith layer

Neural Networks

Basic setup of machine learning:

- output y
- input(s) x
- truth: $y = f^*(x)$
- model: $y = f(x; \beta)$

Neural networks (NNs):

- also called feedforward NNs (FFN) or multi-layer perceptrons (MLP)
- $f(x; \beta) = \cdots f_3(f_2(f_1(x; \beta_1); \beta_2); \beta_3)$
- f_i: activation function for the ith layer

Neural Networks

Basic setup of machine learning:

- output y
- input(s) x
- truth: $y = f^*(x)$
- model: $y = f(x; \beta)$

Neural networks (NNs):

- also called feedforward NNs (FFN) or multi-layer perceptrons (MLP)
- $f(x; \beta) = \cdots f_3(f_2(f_1(x; \beta_1); \beta_2); \beta_3)$
- f_i: activation function for the ith layer

Neural Networks

Basic setup of machine learning:

- output y
- input(s) x
- truth: $y = f^*(x)$
- model: $y = f(x; \beta)$

Neural networks (NNs):

- also called feedforward NNs (FFN) or multi-layer perceptrons (MLP)
- $f(x; \beta) = \cdots f_3(f_2(f_1(x; \beta_1); \beta_2); \beta_3)$
- f_i: activation function for the ith layer

Neural Networks

Basic setup of machine learning:

- output y
- input(s) x
- truth: $y = f^*(x)$
- model: $y = f(x; \beta)$

Neural networks (NNs):

- also called feedforward NNs (FFN) or multi-layer perceptrons (MLP)
- $f(x; \beta) = \cdots f_3(f_2(f_1(x; \beta_1); \beta_2); \beta_3)$
- f_i: activation function for the ith layer

Neural Networks

Basic setup of machine learning:

- output y
- input(s) x
- truth: $y = f^*(x)$
- model: $y = f(x; \beta)$

Neural networks (NNs):

- also called feedforward NNs (FFN) or multi-layer perceptrons (MLP)
- $f(x; \beta) = \cdots f_3(f_2(f_1(x; \beta_1); \beta_2); \beta_3)$
- f_i: activation function for the ith layer

Neural Networks II

$$f(x;\theta) = f_3(f_2(f_1(x;\beta_1);\beta_2);\beta_3)$$

We can break this up into parts:

- inputs: x
- hidden layers:
  - $h_1 = f_1(x;\beta_1)$
  - $h_2 = f_2(h_1;\beta_2)$
- output layer: $\hat{y} = f_3(h_2;\beta_3)$

Neural Networks II

$$f(x;\theta) = f_3(f_2(f_1(x;\beta_1);\beta_2);\beta_3)$$

We can break this up into parts:

- inputs: x
- hidden layers:
  - $h_1 = f_1(x;\beta_1)$
  - $h_2 = f_2(h_1;\beta_2)$
- output layer: $\hat{y} = f_3(h_2;\beta_3)$

Neural Networks II

$$f(x;\theta) = f_3(f_2(f_1(x;\beta_1);\beta_2);\beta_3)$$

We can break this up into parts:

- inputs: x
- hidden layers:
  - $h_1 = f_1(x;\beta_1)$
  - $h_2 = f_2(h_1;\beta_2)$
- output layer: $\hat{y} = f_3(h_2;\beta_3)$

Neural Networks II

$$f(x;\theta) = f_3(f_2(f_1(x;\beta_1);\beta_2);\beta_3)$$

We can break this up into parts:

- inputs: x
- hidden layers:
  - $h_1 = f_1(x;\beta_1)$
  - $h_2 = f_2(h_1;\beta_2)$
- output layer: $\hat{y} = f_3(h_2;\beta_3)$

Neural Networks II

$$f(x;\theta) = f_3(f_2(f_1(x;\beta_1);\beta_2);\beta_3)$$

We can break this up into parts:

- inputs: x
- hidden layers:
  - $h_1 = f_1(x;\beta_1)$
  - $h_2 = f_2(h_1;\beta_2)$
- output layer: $\hat{y} = f_3(h_2;\beta_3)$

Neural Networks II

$$f(x;\theta) = f_3(f_2(f_1(x;\beta_1);\beta_2);\beta_3)$$

We can break this up into parts:

- inputs: x
- hidden layers:
  - $h_1 = f_1(x;\beta_1)$
  - $h_2 = f_2(h_1;\beta_2)$
- output layer: $\hat{y} = f_3(h_2;\beta_3)$

Neural Networks II

$$f(x;\theta) = f_3(f_2(f_1(x;\beta_1);\beta_2);\beta_3)$$

We can break this up into parts:

- inputs: x
- hidden layers:
  - $h_1 = f_1(x;\beta_1)$
  - $h_2 = f_2(h_1;\beta_2)$
- output layer: $\hat{y} = f_3(h_2;\beta_3)$

Neural Networks II

$$f(x;\theta) = f_3(f_2(f_1(x;\beta_1);\beta_2);\beta_3)$$

We can break this up into parts:

- inputs: x
- hidden layers:
  - $h_1 = f_1(x;\beta_1)$
  - $h_2 = f_2(h_1;\beta_2)$
- output layer: $\hat{y} = f_3(h_2;\beta_3)$

Neural Networks III

a bit more technical:

- inputs $\underset{p\times 1}{x}$
- hidden layers:
  - $\underset{n_1\times 1}{h_1} = f_1(\underset{n_1\times p}{W_1}\ \underset{p\times 1}{x} + \underset{n_1\times 1}{b_1})$
  - $\underset{n_2\times 1}{h_2} = f_2(\underset{n_2\times n_1}{W_2}\ \underset{n_1\times 1}{h_1} + \underset{n_2\times 1}{b_2})$
- output layer
  - $\underset{C\times 1}{\hat{y}} = f_3(\underset{C\times n_2}{W_3}\ \underset{n_2\times 1}{h_2} + \underset{C\times 1}{b_3})$

- design choices:
  - **depth**, the total number of layers
  - **width** of each layer (n_i) $\rightarrow$ determines number of parameters
  - **activation function** in each layer (f_i)

Design Choices: Width and Depth

- **architecture**: overall design of the NN
- **Universal Approximation Theorem**: NN with one hidden layer and one output layer is enough to approximate a large class of functions, given the hidden layer is "wide enough"
  - does *not* mean our algorithm is guaranteed to **learn** the correct function
  - width of the layer may need to be extremely large
- no "one-size-fits-all" approach
  - empirical evidence that $\uparrow$ depth is more effective than $\uparrow$ width
  - look for previous research using the same or similar data as starting point

Design Choices: Activation Functions

- what about activation functions?
- linear activation functions $f(x) = x$?
  - our NN becomes a linear model, i.e linear regression
- power of NNs comes from non-linear activation functions in hidden layers
- common choice is the "almost linear" ReLU function:
  - $\text{ReLU}(x) = \max(0, x)$

- activation function in the output layer will depends on the type of output
  - in multi-class classification (C classes), typically use:

$$\text{softmax}(x_1, \ldots, x_C) = \left(\frac{e^{x_1}}{\sum e^{x_i}}, \ldots, \frac{e^{x_C}}{\sum e^{x_i}} \right)$$

How do we actually "learn"?

The gist:

1) start with initial "guess" $\hat{\theta}_0$
2) given loss function $L(y, \hat{y})$, compute $\frac{\partial L}{\partial \theta}(\hat{\theta}_0)$
3) update "guess": $\hat{\theta}_1 = \hat{\theta}_0 - r\frac{\partial L}{\partial \theta}(\hat{\theta}_0)$
   - r: "learning rate"
4) repeat steps 2 and 3

Questions:

- How do we compute the derivative? **beyond the scope of this workshop**
- What loss function to use? **for classification, cross-entropy/logistic loss is natural choice**
- What learning rate to use? **try different values**
- How many times do we repeat steps 2 and 3? **next slide**

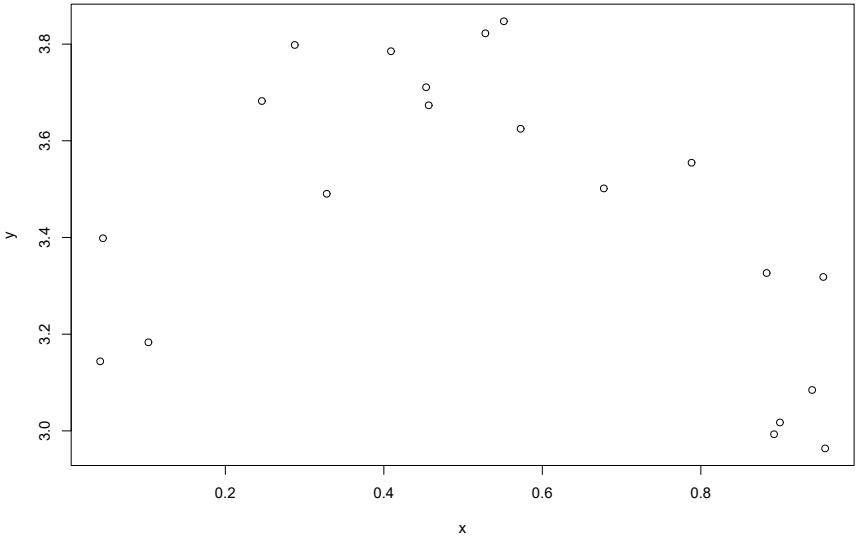
Stochastic Gradient Descent: Batches and Epochs

- in methods like logistic regression:
  - compute derivative for entire dataset before updating estimate
  - run algorithm until convergence is reached
- with NNs:
  - randomly divide data into **batches**, update after each batch
  - specify how many times we iterate through entire data (number of **epochs**)
  - we do **not** want to let the algorithm run too long $\rightarrow$ overfitting

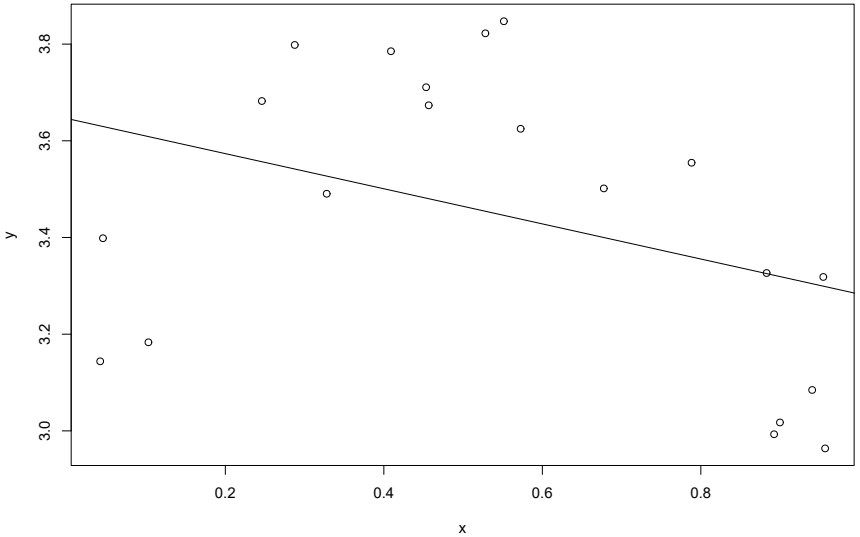
Summary So Far

- design decisions:
  - depth: total number of layers
  - width of each layer
  - activation function at each layer
- computational decisions:
  - loss function
  - learning rate
  - batch size (how frequently we update the model estimates)
  - number of epochs (how many times we pass through entire dataset)

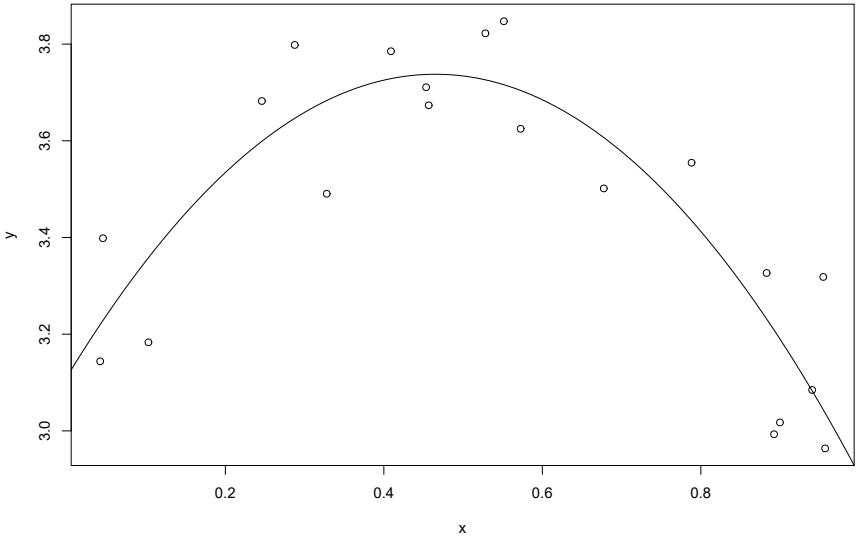
Overfitting



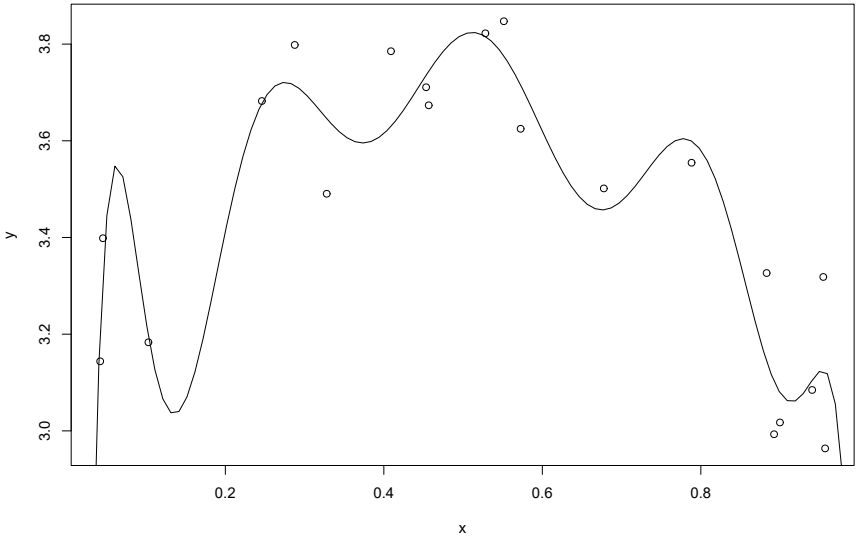
Overfitting



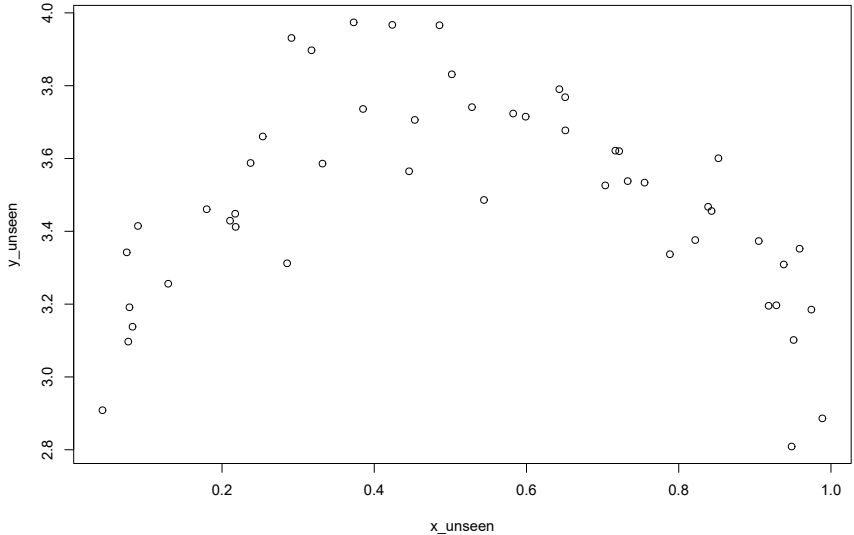
Overfitting



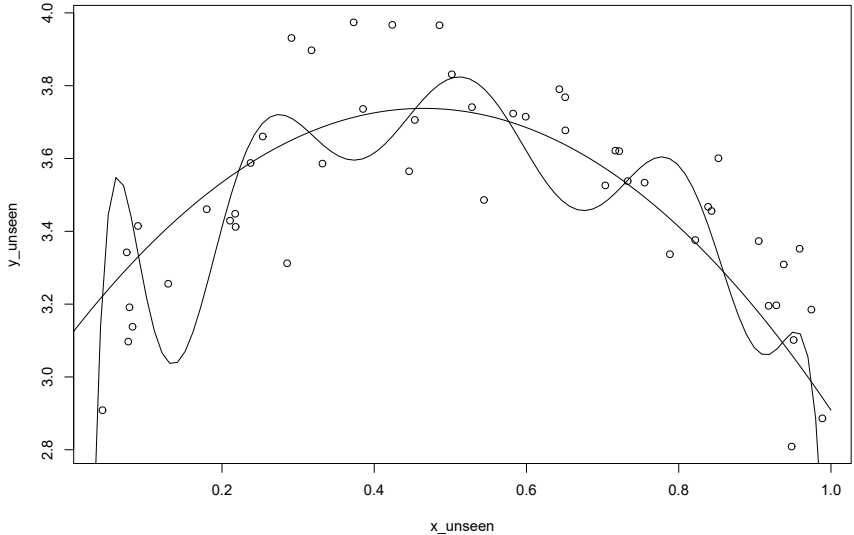
Overfitting



Overfitting



Overfitting



Data Splitting

- overfitting means your model fits the data it is trained on, but performs poorly on "unseen" data
  - **generalization** error or **test** error
- so, let's simulate having unseen data!
- randomly split into three:
  - training set: used to train the model
  - validation set: used to compare performance for choices of depth, width, learning rate, etc.
  - test set: used only at the very end to assess performance of the final chosen model

Regularization

Regularization refers to methods of "constraining" the model to reduce overfitting.

Some common methods:

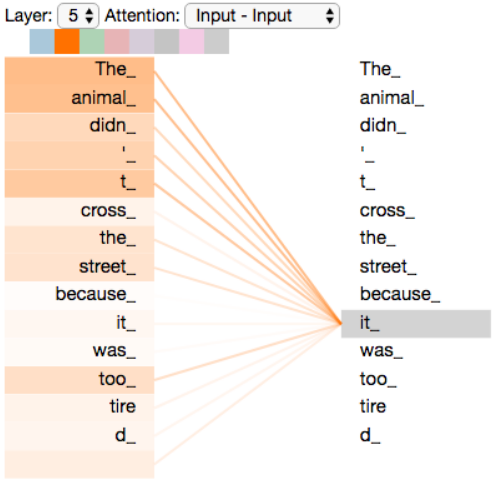
- **weight decay**: penalty placed on large parameter estimates
- **dropout**: randomly choose fraction of layer and set to 0
- **early stopping**: limit number of epochs

(Self-)Attention

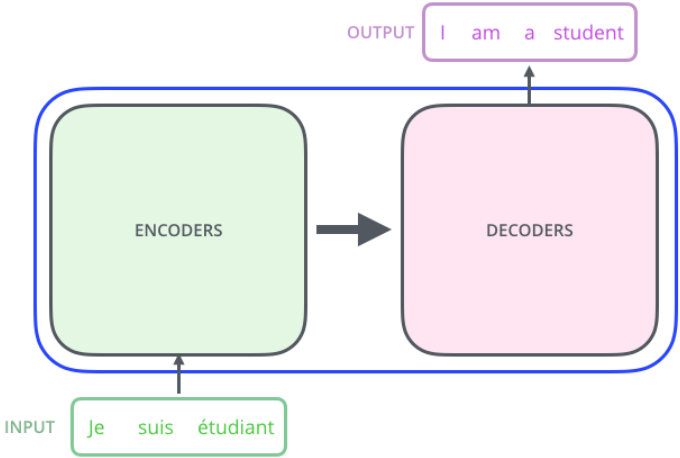
Consider the sentence:
"The animal didn't cross the street because it was too tired"

(Self-)Attention

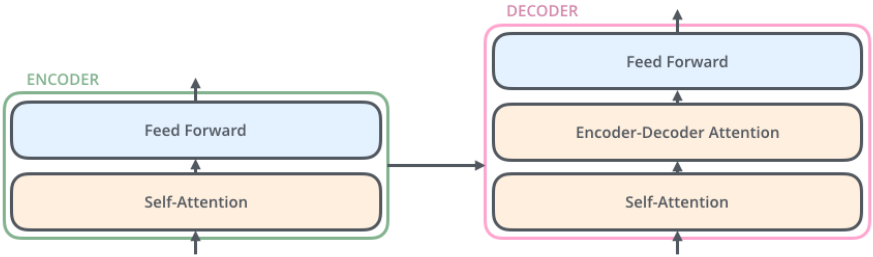
Consider the sentence:
"The animal didn't cross the street because it was too tired"



Transformers

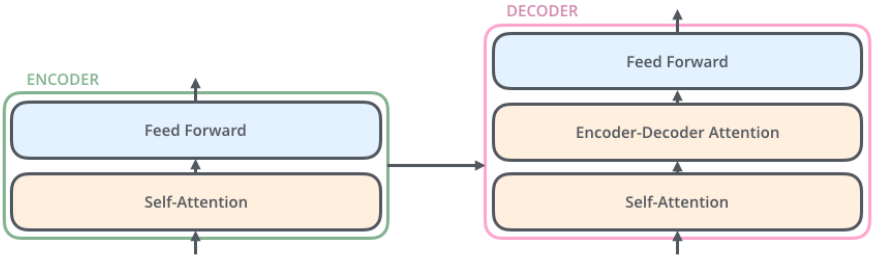


Transformers pt2



graphics: https://jalammar.github.io/illustrated-transformer/
seminal paper: "Attention is All You Need" (Vaswani et al 2017)

Transformers pt2



graphics: `https://jalammar.github.io/illustrated-transformer/`
seminal paper: "Attention is All You Need" (Vaswani et al 2017)

LLMs for Text Classification

Broadly speaking, two approaches to LLM-based text classification:

1. Fine-tuning
2. Prompt-based

Data

Open-ended question fielded in Dutch as a web survey in the LISS panel in 2012 (Martin et al 2011):

> *Joe's doctor told him that he would need to return in two weeks to find out whether or not his condition had improved. But when Joe asked the receptionist for an appointment, he was told that it would be over a month before the next available appointment. What should Joe do?*

$n = 1756$ texts double-coded into 4 categories:

1. Proactive
2. Somewhat proactive
3. Passive
4. Destructive

source: https://www.dataarchive.lissdata.nl/study-units/view/971

Data

Open-ended question fielded in Dutch as a web survey in the LISS panel in 2012 (Martin et al 2011):

> *Joe's doctor told him that he would need to return in two weeks to find out whether or not his condition had improved. But when Joe asked the receptionist for an appointment, he was told that it would be over a month before the next available appointment. What should Joe do?*

$n = 1756$ texts double-coded into 4 categories:

1. Proactive
2. Somewhat proactive
3. Passive
4. Destructive

source: `https://www.dataarchive.lissdata.nl/study-units/view/971`

(Fine-Tuning I) BERT Pre-Training

- BERT = Bidirectional Encoder Representations from Transformers
- Pre-training data: BooksCorpus (800M words) and English Wikipedia (2,500M words)
- Training tasks:
  - masked token prediction
  - next sentence prediction
- Source: "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" (Devlin et al 2019; original version 2018)
- Dutch-language: "BERTje: A Dutch BERT Model" (de Vries et al 2019)

(Fine-Tuning I) BERT Pre-Training

- BERT = Bidirectional Encoder Representations from Transformers
- Pre-training data: BooksCorpus (800M words) and English Wikipedia (2,500M words)
- Training tasks:
  - masked token prediction
  - next sentence prediction
- Source: "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" (Devlin et al 2019; original version 2018)
- Dutch-language: "BERTje: A Dutch BERT Model" (de Vries et al 2019)

(Fine-Tuning I) BERT Pre-Training

- BERT = Bidirectional Encoder Representations from Transformers
- Pre-training data: BooksCorpus (800M words) and English Wikipedia (2,500M words)
- Training tasks:
  - masked token prediction
  - next sentence prediction
- Source: "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" (Devlin et al 2019; original version 2018)
- Dutch-language: "BERTje: A Dutch BERT Model" (de Vries et al 2019)

(Fine-Tuning I) BERT Pre-Training

- BERT = Bidirectional Encoder Representations from Transformers
- Pre-training data: BooksCorpus (800M words) and English Wikipedia (2,500M words)
- Training tasks:
  - masked token prediction
  - next sentence prediction
- Source: "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" (Devlin et al 2019; original version 2018)
- Dutch-language: "BERTje: A Dutch BERT Model" (de Vries et al 2019)

(Fine-Tuning II) BERT Fine-Tuning

- Basic premise:
  - input text x
  - black-box model $\tilde{x} = \text{BERT}(x; \theta_{\text{pre-trained}})$
    - $\theta_{\text{pre-trained}}$ consists of 100s of millions of parameters
  - classification head $y = f(\tilde{x}; \theta_{\text{fine-tuned}})$
- During fine-tuning
  - can choose whether to update both $\theta_{\text{pre-trained}}, \theta_{\text{fine-tuned}}$ or only $\theta_{\text{fine-tuned}}$
  - choose values of hyperparameters: learning rate, batch size, number of epochs

(Fine-Tuning II) BERT Fine-Tuning

- Basic premise:
  - input text x
  - black-box model $\tilde{x} = \text{BERT}(x; \theta_{\text{pre-trained}})$
    - $\theta_{\text{pre-trained}}$ consists of 100s of millions of parameters
  - classification head $y = f(\tilde{x}; \theta_{\text{fine-tuned}})$
- During fine-tuning
  - can choose whether to update both $\theta_{\text{pre-trained}}, \theta_{\text{fine-tuned}}$ or only $\theta_{\text{fine-tuned}}$
  - choose values of hyperparameters: learning rate, batch size, number of epochs

(Fine-Tuning II) BERT Fine-Tuning

- Basic premise:
  - input text x
  - black-box model $\tilde{x} = \text{BERT}(x; \theta_{\text{pre-trained}})$
    - $\theta_{\text{pre-trained}}$ consists of 100s of millions of parameters
  - classification head $y = f(\tilde{x}; \theta_{\text{fine-tuned}})$
- During fine-tuning
  - can choose whether to update both $\theta_{\text{pre-trained}}, \theta_{\text{fine-tuned}}$ or only $\theta_{\text{fine-tuned}}$
  - choose values of hyperparameters: learning rate, batch size, number of epochs

(Fine-Tuning II) BERT Fine-Tuning

- Basic premise:
  - input text x
  - black-box model $\tilde{x} = \text{BERT}(x; \theta_{\text{pre-trained}})$
    - $\theta_{\text{pre-trained}}$ consists of 100s of millions of parameters
  - classification head $y = f(\tilde{x}; \theta_{\text{fine-tuned}})$
- During fine-tuning
  - can choose whether to update both $\theta_{\text{pre-trained}}, \theta_{\text{fine-tuned}}$ or only $\theta_{\text{fine-tuned}}$
  - choose values of hyperparameters: learning rate, batch size, number of epochs

(Fine-Tuning II) BERT Fine-Tuning

- Basic premise:
  - input text x
  - black-box model $\tilde{x} = \text{BERT}(x; \theta_{\text{pre-trained}})$
    - $\theta_{\text{pre-trained}}$ consists of 100s of millions of parameters
  - classification head $y = f(\tilde{x}; \theta_{\text{fine-tuned}})$
- During fine-tuning
  - can choose whether to update both $\theta_{\text{pre-trained}}, \theta_{\text{fine-tuned}}$ or only $\theta_{\text{fine-tuned}}$
  - choose values of hyperparameters: learning rate, batch size, number of epochs

(Fine-Tuning II) BERT Fine-Tuning

- Basic premise:
  - input text x
  - black-box model $\tilde{x} = \text{BERT}(x; \theta_{\text{pre-trained}})$
    - $\theta_{\text{pre-trained}}$ consists of 100s of millions of parameters
  - classification head $y = f(\tilde{x}; \theta_{\text{fine-tuned}})$
- During fine-tuning
  - can choose whether to update both $\theta_{\text{pre-trained}}, \theta_{\text{fine-tuned}}$ or only $\theta_{\text{fine-tuned}}$
  - choose values of hyperparameters: learning rate, batch size, number of epochs

(Fine-Tuning II) BERT Fine-Tuning

- Basic premise:
  - input text x
  - black-box model $\tilde{x} = \text{BERT}(x; \theta_{\text{pre-trained}})$
    - $\theta_{\text{pre-trained}}$ consists of 100s of millions of parameters
  - classification head $y = f(\tilde{x}; \theta_{\text{fine-tuned}})$
- During fine-tuning
  - can choose whether to update both $\theta_{\text{pre-trained}}, \theta_{\text{fine-tuned}}$ or only $\theta_{\text{fine-tuned}}$
  - choose values of hyperparameters: learning rate, batch size, number of epochs

(Prompt Engineering I) GPT and ChatGPT

- GPT = Generative Pre-trained Transformer
- core of the model is still transformers (attention)
- often used for text (and image, etc) generation
- OpenAI develops "foundational models", most recently GPT5
- These are then used inside other products, like ChatGPT
- ChatGPT fine-tuned for specialized task (chat bot)
  - involves "reinforcement learning from human feedback" (RLHF)

(Prompt Engineering I) GPT and ChatGPT

- GPT = Generative Pre-trained Transformer
- core of the model is still transformers (attention)
- often used for text (and image, etc) generation
- OpenAI develops "foundational models", most recently GPT5
- These are then used inside other products, like ChatGPT
- ChatGPT fine-tuned for specialized task (chat bot)
  - involves "reinforcement learning from human feedback" (RLHF)

(Prompt Engineering I) GPT and ChatGPT

- GPT = Generative Pre-trained Transformer
- core of the model is still transformers (attention)
- often used for text (and image, etc) generation
- OpenAI develops "foundational models", most recently GPT5
- These are then used inside other products, like ChatGPT
- ChatGPT fine-tuned for specialized task (chat bot)
  - involves "reinforcement learning from human feedback" (RLHF)

(Prompt Engineering I) GPT and ChatGPT

- GPT = Generative Pre-trained Transformer
- core of the model is still transformers (attention)
- often used for text (and image, etc) generation
- OpenAI develops "foundational models", most recently GPT5
- These are then used inside other products, like ChatGPT
- ChatGPT fine-tuned for specialized task (chat bot)
  - involves "reinforcement learning from human feedback" (RLHF)

(Prompt Engineering I) GPT and ChatGPT

- GPT = Generative Pre-trained Transformer
- core of the model is still transformers (attention)
- often used for text (and image, etc) generation
- OpenAI develops "foundational models", most recently GPT5
- These are then used inside other products, like ChatGPT
- ChatGPT fine-tuned for specialized task (chat bot)
  - involves "reinforcement learning from human feedback" (RLHF)

(Prompt Engineering I) GPT and ChatGPT

- GPT = Generative Pre-trained Transformer
- core of the model is still transformers (attention)
- often used for text (and image, etc) generation
- OpenAI develops "foundational models", most recently GPT5
- These are then used inside other products, like ChatGPT
- ChatGPT fine-tuned for specialized task (chat bot)
  - involves "reinforcement learning from human feedback" (RLHF)

(Prompt Engineering I) GPT and ChatGPT

- GPT = Generative Pre-trained Transformer
- core of the model is still transformers (attention)
- often used for text (and image, etc) generation
- OpenAI develops "foundational models", most recently GPT5
- These are then used inside other products, like ChatGPT
- ChatGPT fine-tuned for specialized task (chat bot)
  - involves "reinforcement learning from human feedback" (RLHF)

(Prompt Engineering II) How Do We Train Humans To Code?

Human coders given a codebook:

Somewhat active: *call me if*

Patient accepts the appointment but asks to be called, or patient accepts the appointment and asks the staff to check with the doctor later, or patient asks staff to make an exception. Examples are:

- Leaving his/her name and number with the receptionist in case an appointment comes up.
- Asking to be called if there is a cancellation
- Leaving the office with the intention to later call the office to check that the appointment is ok with the doctor.
- Asking to squeeze patient in
- Asking to double book
- Asking for other alternatives

Idea: train ChatGPT the same way.

(Prompt Engineering II) How Do We Train Humans To Code?

Human coders given a codebook:

Somewhat active: *call me if*
Patient accepts the appointment but asks to be called, or patient accepts the appointment and asks the staff to check with the doctor later, or patient asks staff to make an exception. Examples are:

- Leaving his/her name and number with the receptionist in case an appointment comes up.
- Asking to be called if there is a cancellation
- Leaving the office with the intention to later call the office to check that the appointment is ok with the doctor.
- Asking to squeeze patient in
- Asking to double book
- Asking for other alternatives

Idea: train ChatGPT the same way.

(Prompt Engineering III) How Do We Train ChatGPT to Code?

- system specification:
  - The codebook text, including definition of each category and examples.
  - Replace labels (Proactive, Somewhat Proactive, Passive, Destructive) with generic (Group A through D)
  - End with "Give only one of the four categories (Group A, Group B, Group C, Group D) as your answer."
- user input:
  - "Here is the survey response which I need you to classify: [insert text]"

(Prompt Engineering III) How Do We Train ChatGPT to Code?

- system specification:
  - The codebook text, including definition of each category and examples.
  - Replace labels (Proactive, Somewhat Proactive, Passive, Destructive) with generic (Group A through D)
  - End with "Give only one of the four categories (Group A, Group B, Group C, Group D) as your answer."
- user input:
  - "Here is the survey response which I need you to classify: [insert text]"

(Prompt Engineering III) How Do We Train ChatGPT to Code?

- system specification:
  - The codebook text, including definition of each category and examples.
  - Replace labels (Proactive, Somewhat Proactive, Passive, Destructive) with generic (Group A through D)
  - End with "Give only one of the four categories (Group A, Group B, Group C, Group D) as your answer."
- user input:
  - "Here is the survey response which I need you to classify: [insert text]"

(Prompt Engineering III) How Do We Train ChatGPT to Code?

- system specification:
  - The codebook text, including definition of each category and examples.
  - Replace labels (Proactive, Somewhat Proactive, Passive, Destructive) with generic (Group A through D)
  - End with "Give only one of the four categories (Group A, Group B, Group C, Group D) as your answer."
- user input:
  - "Here is the survey response which I need you to classify: [insert text]"

(Prompt Engineering III) How Do We Train ChatGPT to Code?

- system specification:
  - The codebook text, including definition of each category and examples.
  - Replace labels (Proactive, Somewhat Proactive, Passive, Destructive) with generic (Group A through D)
  - End with "Give only one of the four categories (Group A, Group B, Group C, Group D) as your answer."
- user input:
  - "Here is the survey response which I need you to classify: [insert text]"

(Prompt Engineering III) How Do We Train ChatGPT to Code?

- system specification:
  - The codebook text, including definition of each category and examples.
  - Replace labels (Proactive, Somewhat Proactive, Passive, Destructive) with generic (Group A through D)
  - End with "Give only one of the four categories (Group A, Group B, Group C, Group D) as your answer."
- user input:
  - "Here is the survey response which I need you to classify: [insert text]"

References

- resources used in developing slides:
  - "Deep Learning" by Ian Goodfellow and Yoshua Bengio and Aaron Courville
    - `https://www.deeplearningbook.org/`
  - "Applied Statistical Learning" By Matthias Schonlau
  - seminal paper: "Attention is All You Need" (Vaswani et al 2017)
  - BERT:
    - "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" (Devlin et al 2019; original version 2018)
    - Dutch-language: "BERTje: A Dutch BERT Model" (de Vries et al 2019)
- data: `https://www.dataarchive.lissdata.nl/study-units/view/971`

Thank you!

Thank you!