Deep Learning from Scratch

Lecture 1

EN.705.743: ChatGPT from Scratch

Lecture Outline

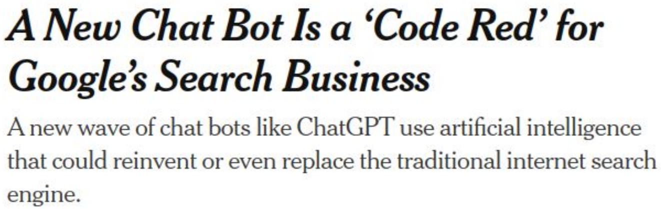
- What is ChatGPT - what is this class all about?
- Syllabus and homeworks, etc

- Deep Learning Refresher
- MLP from Scratch / Intro to HW1

What is ChatGPT?

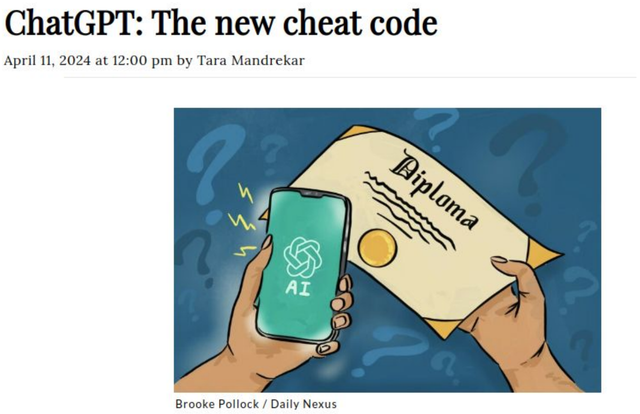
(What is this class?)

ChatGPT



A New Chat Bot Is a 'Code Red' for Google's Search Business

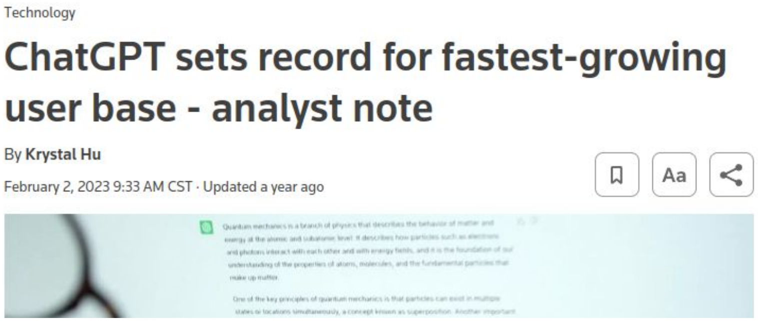
A new wave of chat bots like ChatGPT use artificial intelligence that could reinvent or even replace the traditional internet search engine.



ChatGPT: The new cheat code

April 11, 2024 at 12:00 pm by Tara Mandrekar

Brooke Pollock / Daily Nexus

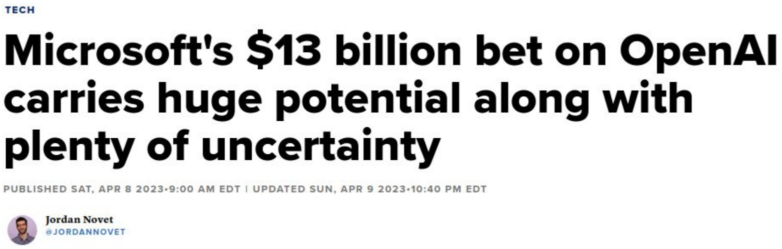


Technology

ChatGPT sets record for fastest-growing user base - analyst note

By **Krystal Hu**

February 2, 2023 9:33 AM CST · Updated a year ago

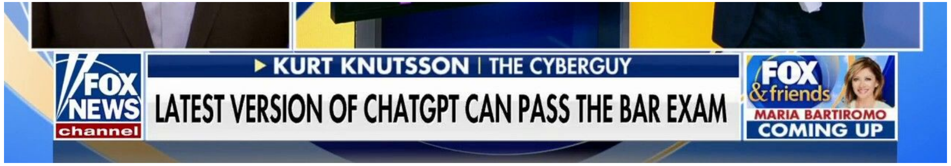


TECH

Microsoft's $13 billion bet on OpenAI carries huge potential along with plenty of uncertainty

PUBLISHED SAT, APR 8 2023·9:00 AM EDT | UPDATED SUN, APR 9 2023·10:40 PM EDT

Jordan Novet
@JORDANNOVET

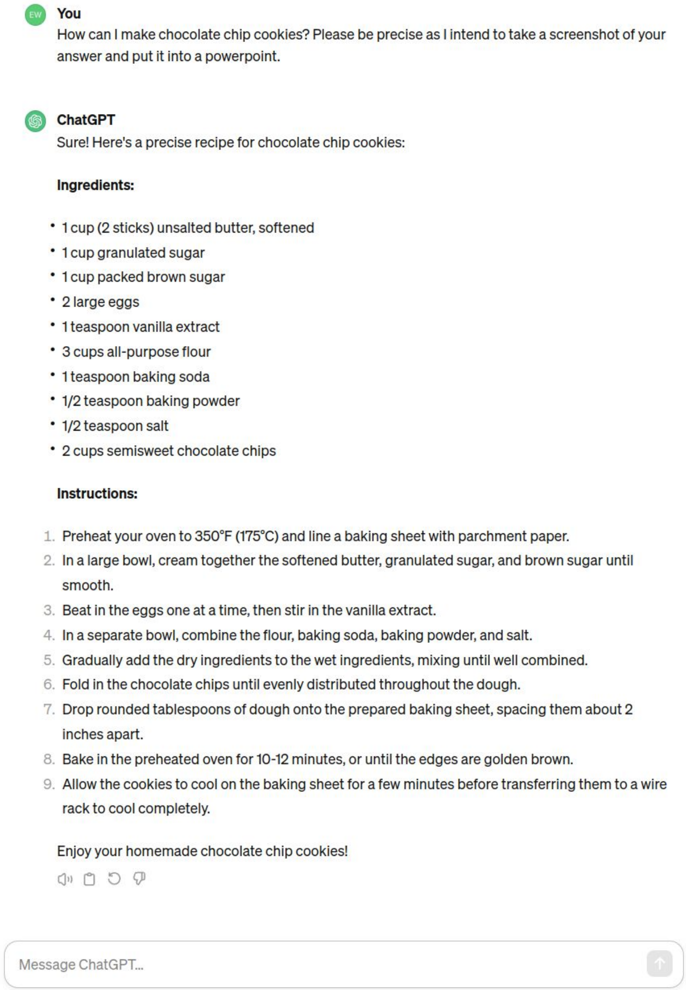


▶ KURT KNUTSSON | THE CYBERGUY

LATEST VERSION OF CHATGPT CAN PASS THE BAR EXAM

ChatGPT

ChatGPT is (to a user) a text interface that allows a user to ask for information, analysis, recommendations, even creative writing outputs.



You

How can I make chocolate chip cookies? Please be precise as I intend to take a screenshot of your answer and put it into a powerpoint.

ChatGPT

Sure! Here's a precise recipe for chocolate chip cookies:

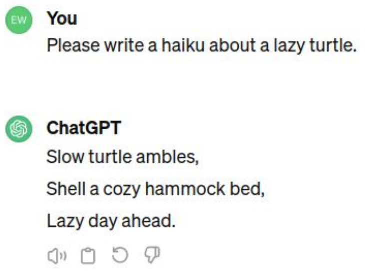
Ingredients:

- 1 cup (2 sticks) unsalted butter, softened
- 1 cup granulated sugar
- 1 cup packed brown sugar
- 2 large eggs
- 1 teaspoon vanilla extract
- 3 cups all-purpose flour
- 1 teaspoon baking soda
- 1/2 teaspoon baking powder
- 1/2 teaspoon salt
- 2 cups semisweet chocolate chips

Instructions:

1. Preheat your oven to 350°F (175°C) and line a baking sheet with parchment paper.
2. In a large bowl, cream together the softened butter, granulated sugar, and brown sugar until smooth.
3. Beat in the eggs one at a time, then stir in the vanilla extract.
4. In a separate bowl, combine the flour, baking soda, baking powder, and salt.
5. Gradually add the dry ingredients to the wet ingredients, mixing until well combined.
6. Fold in the chocolate chips until evenly distributed throughout the dough.
7. Drop rounded tablespoons of dough onto the prepared baking sheet, spacing them about 2 inches apart.
8. Bake in the preheated oven for 10-12 minutes, or until the edges are golden brown.
9. Allow the cookies to cool on the baking sheet for a few minutes before transferring them to a wire rack to cool completely.

Enjoy your homemade chocolate chip cookies!

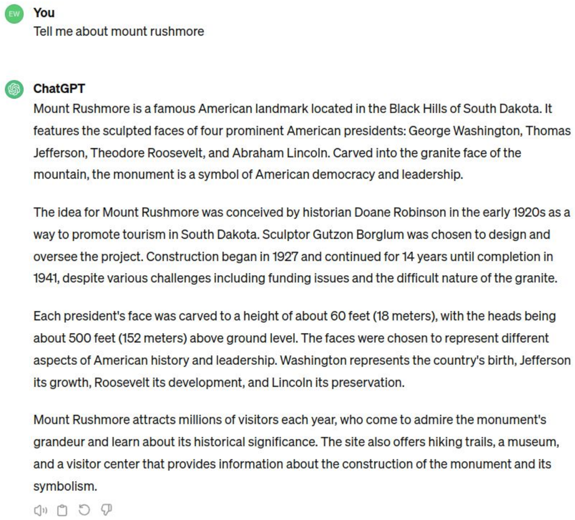


You

Please write a haiku about a lazy turtle.

ChatGPT

Slow turtle ambles,
Shell a cozy hammock bed,
Lazy day ahead.



You

Tell me about mount rushmore

ChatGPT

Mount Rushmore is a famous American landmark located in the Black Hills of South Dakota. It features the sculpted faces of four prominent American presidents: George Washington, Thomas Jefferson, Theodore Roosevelt, and Abraham Lincoln. Carved into the granite face of the mountain, the monument is a symbol of American democracy and leadership.

The idea for Mount Rushmore was conceived by historian Doane Robinson in the early 1920s as a way to promote tourism in South Dakota. Sculptor Gutzon Borglum was chosen to design and oversee the project. Construction began in 1927 and continued for 14 years until completion in 1941, despite various challenges including funding issues and the difficult nature of the granite.

Each president's face was carved to a height of about 60 feet (18 meters), with the heads being about 500 feet (152 meters) above ground level. The faces were chosen to represent different aspects of American history and leadership. Washington represents the country's birth, Jefferson its growth, Roosevelt its development, and Lincoln its preservation.

Mount Rushmore attracts millions of visitors each year, who come to admire the monument's grandeur and learn about its historical significance. The site also offers hiking trails, a museum, and a visitor center that provides information about the construction of the monument and its symbolism.

What *is* ChatGPT?

ChatGPT is a software system built around a very large, carefully trained deep neural network. Specifically, this neural network is a large language model (LLM).

What *is* ChatGPT?

ChatGPT is a software system built around a very large, carefully trained deep neural network. Specifically, this neural network is a large language model (LLM).

Language Model: A neural network that performs computations on text.

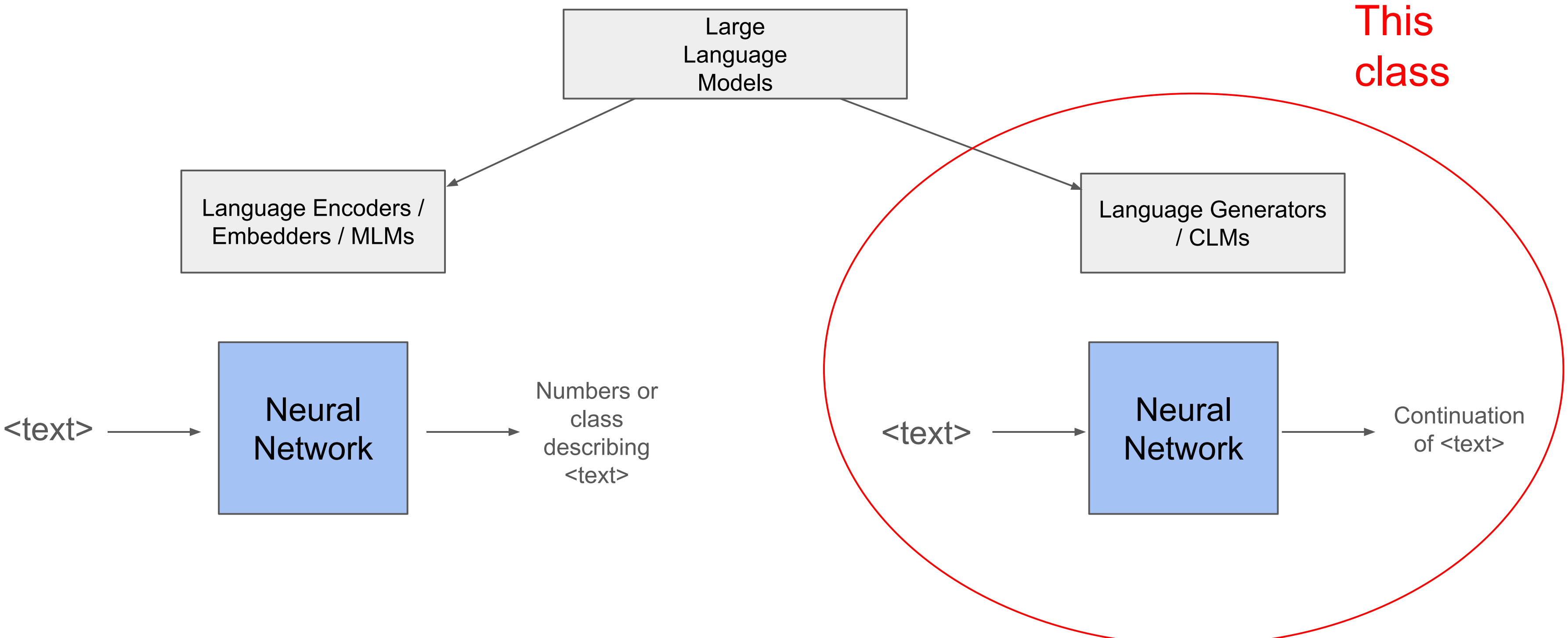
LLM: Large Language Model, a gigantic language model that is trained on so much text that it becomes useful as a general text processing engine.

Typically:

- Built on the transformer architecture
- Billions of parameters

Aside

There are actually two families of LLMs:



Large Language Models

Language Encoders / Embedders / MLMs

Language Generators / CLMs

This class

<text>

Neural Network

Numbers or class describing <text>

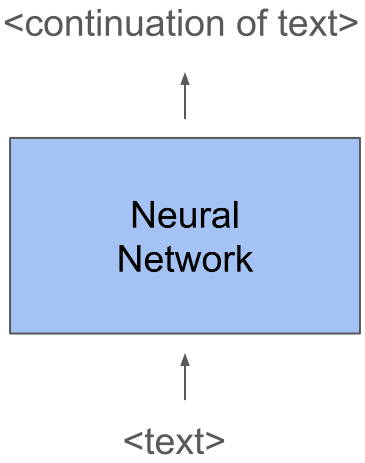
<text>

Neural Network

Continuation of <text>

Brief Overview

Carried over from the last slide. This is not super descriptive…



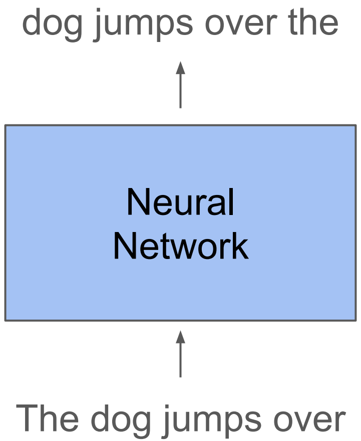
<continuation of text>

Neural Network

<text>

Brief Overview

It is better to imagine a sequence of words. The model supplies the next word the input.



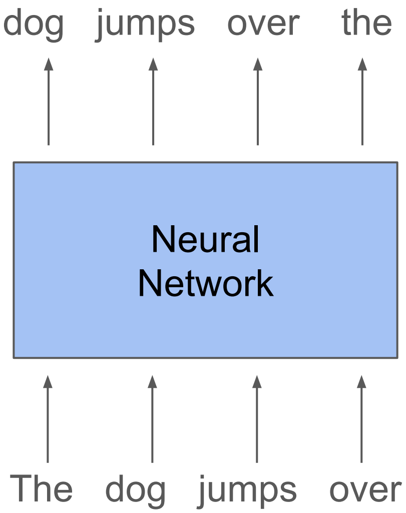
dog jumps over the

Neural Network

The dog jumps over

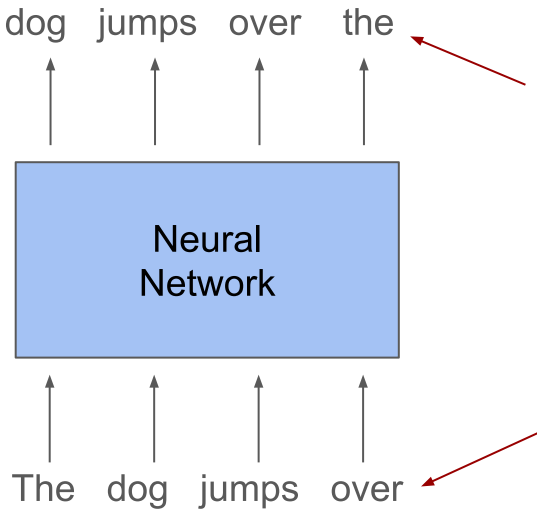
Brief Overview

In practice, the model actually processes a sequence. The whole sequence is fed through at once.



Brief Overview

In practice, the model actually processes a sequence. The whole sequence is fed through at once.

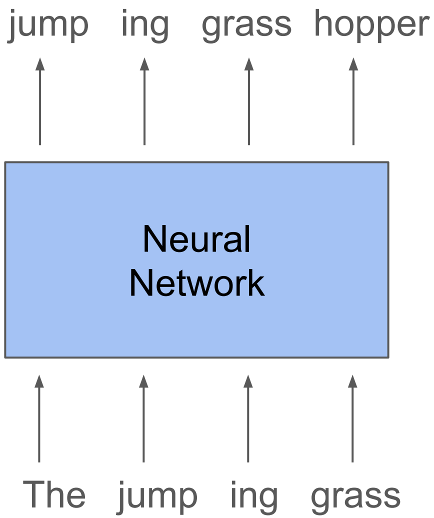


Every output is the next word in the sequence, including a new word!

The individual chunks of language that we feed in and out are called “tokens”.

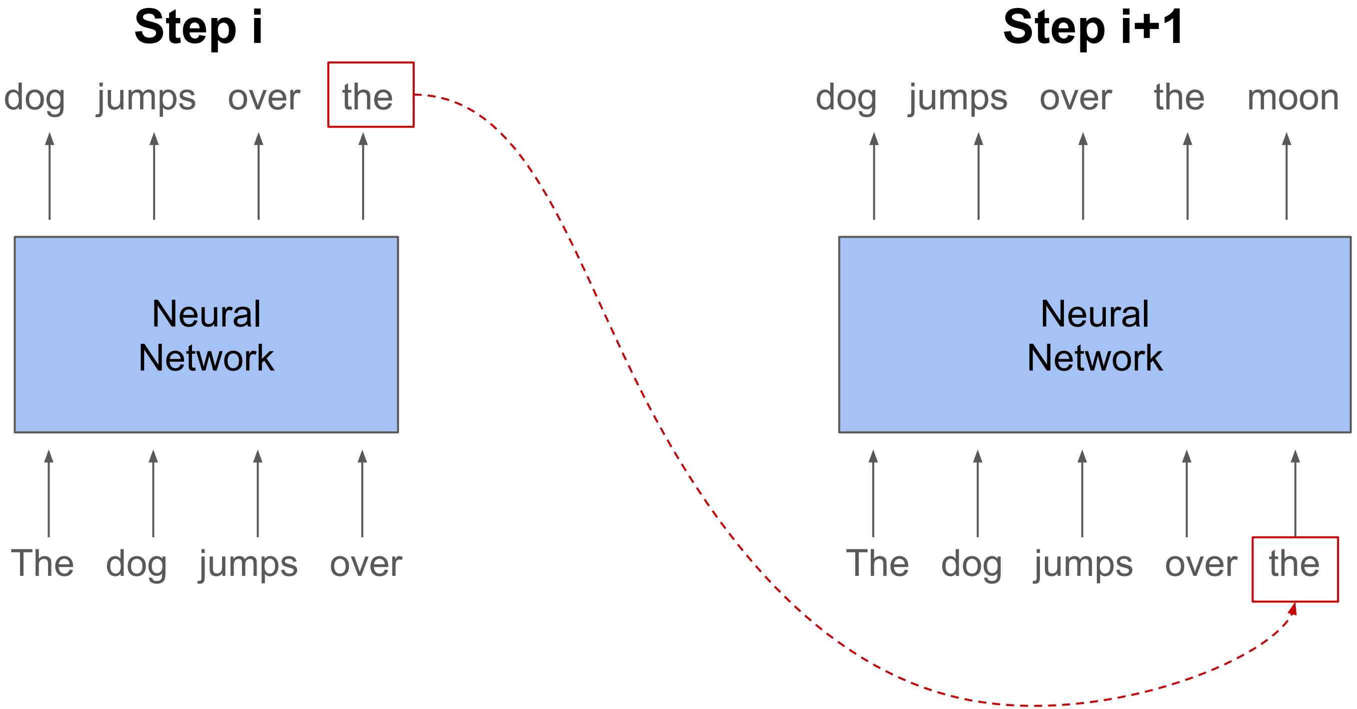
Brief Overview

We don’t call them “words” because sometimes it is easier to split up words into multiple chunks (roots, suffixes, etc):



Brief Overview

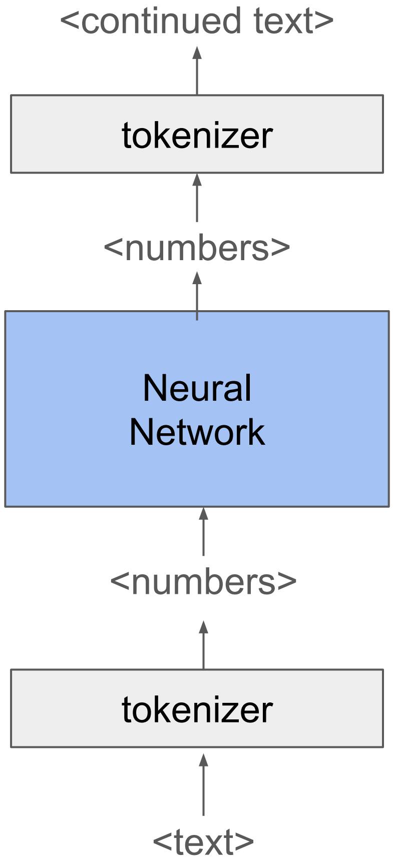
To generate text, we simply feed the new word back into the model:



Brief Overview

A neural network cannot process text directly: it needs to be represented as numbers.

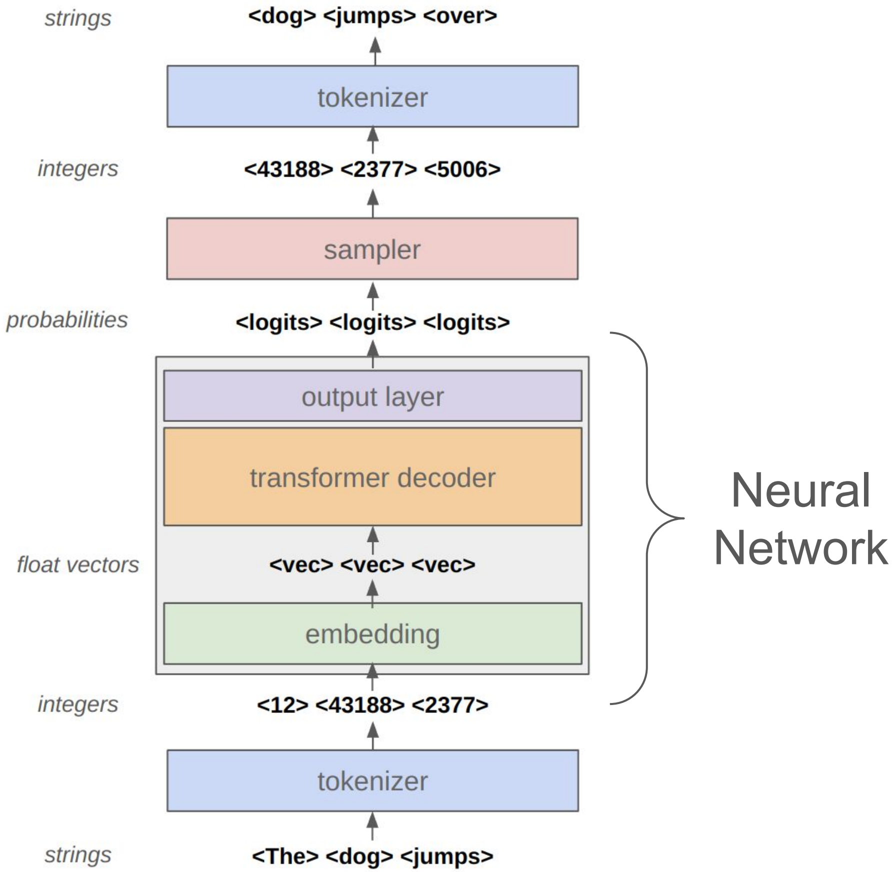
We have an additional piece of software, called a tokenizer, to convert back and forth between strings and numbers.



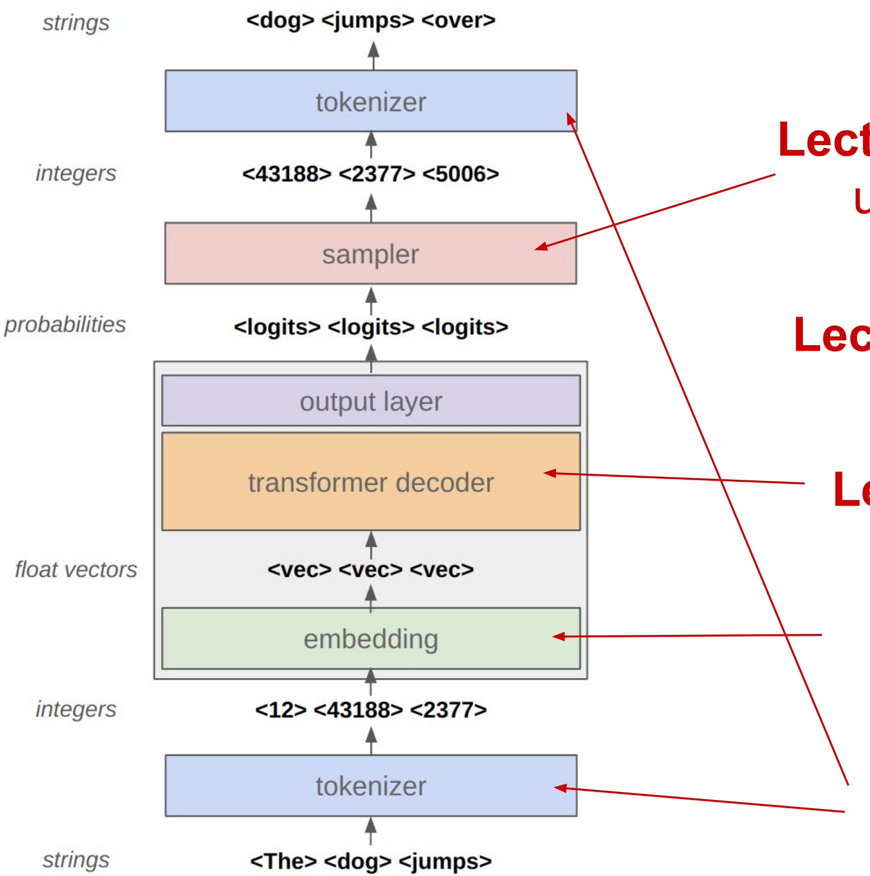
Brief Overview

Expanding all the way out, we have a complete diagram of a ChatGPT-style LLM.

We are going to see this diagram many times! (No need to understand it all now)



Course Overview



Lecture 7 (sampler not used in training)

Lecture 6: Training

Lectures 4 & 5

Lecture 3

Lecture 2

Course Outline

The first half of the course will breakdown the ChatGPT architecture in detail:

Lecture 1: Overview

Lecture 2: Tokenization

Lecture 3: Embeddings

Lectures 4 & 5: Transformers

Lecture 6: Training the model

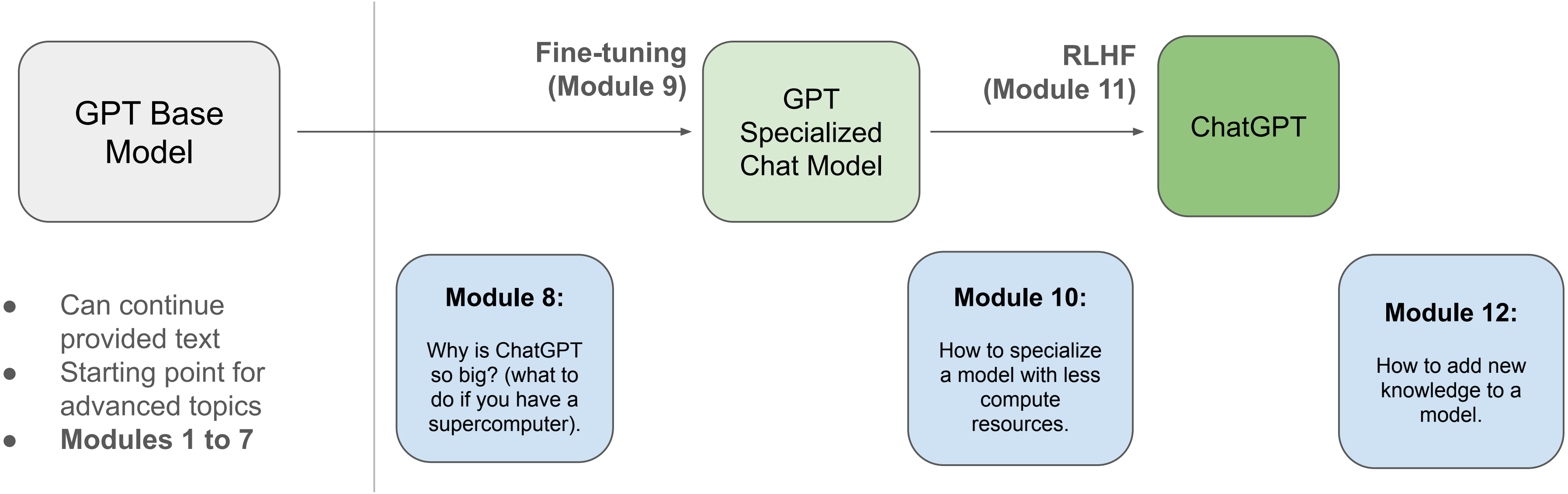
Lecture 7: Sampling from a model

Homework: Your homework after each of these lectures will be to implement the relevant part of the model (and also some readings and short answers).

After week 7, you will have each implemented an entire language model!

What about the rest of the course?

There is a lot more to ChatGPT than this model! The rest of the course will focus on advanced topics that turn a simple LLM into a powerful chatbot or app:



GPT Base Model

- Can continue provided text
- Starting point for advanced topics
- **Modules 1 to 7**

Fine-tuning (Module 9)

GPT Specialized Chat Model

RLHF (Module 11)

ChatGPT

Module 8:

Why is ChatGPT so big? (what to do if you have a supercomputer).

Module 10:

How to specialize a model with less compute resources.

Module 12:

How to add new knowledge to a model.

Course Outline (Continued)

The second half will focus on advanced topics.

Lecture 8: Scale

Lecture 9: Fine-tuning

Lecture 10: PEFT and Quantization

Lecture 11: RLHF

Lecture 12: Knowledge Distillation

Homework: For the last half of the course you will have a class project.

Your project will focus on changing some aspect of your LLM and how this impacts the model.

Summary of Class

- First 7 modules: Details of LLM architecture. Homeworks focus on implementing this yourself, in pieces.
- Last 5 modules: Advances uses of LLMs. Homework is to work on a project, which will be due at the end of the course (details will come later).
- Additionally, there are short readings (a few pages) each week and 1-2 short answer questions about the readings (a few sentences each).
- There are no quizzes or exams. This course is really about learning through programming things yourself- an exam is not of much help here.

Syllabus Review

Deep Learning Refresher

Supervised Learning

We are given a dataset of (x,y) pairs, and we want to learn some function f(x) that predicts y.

Each x in X is typically a vector of features that describe the characteristics of a sample.

Each y in Y might be a selection from a set of labels (classification), or a real-valued vector (regression).

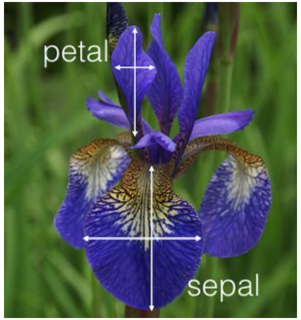
Supervised Learning

We are given a dataset of (x,y) pairs, and we want to learn some function f(x) that predicts y.

Each x in X is typically a vector of features that describe the characteristics of a sample.

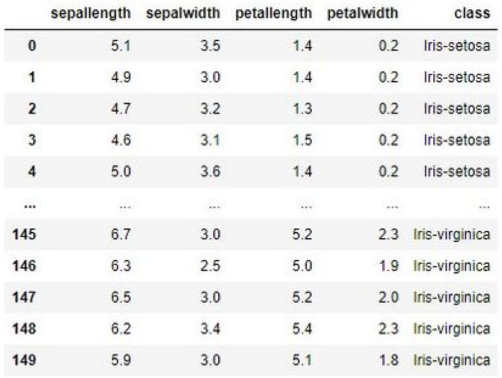
Each y in Y might be a selection from a set of labels (classification), or a real-valued vector (regression).

Example: IRIS Dataset



x: 4 features

y: class

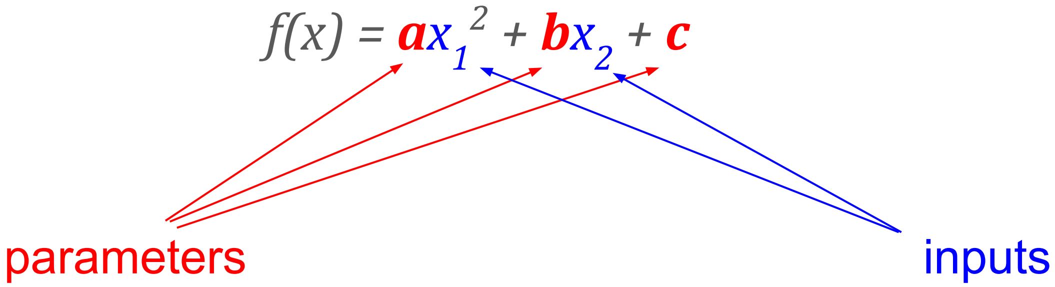


	sepallength	sepalwidth	petallength	petalwidth	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

Supervised Learning

We are given a dataset of (x,y) pairs, and we want to learn some function f(x) that predicts y.

Our first step is to decide on the form of the function f(x). The function's form will stay fixed, but we will adjust it by learning values for parameters.



$$f(x) = \mathbf{a}x_1^2 + \mathbf{b}x_2 + \mathbf{c}$$

parameters

inputs

Supervised Learning

One of the simplest forms we can choose is a linear function:

$y = mx + b$ for x with only one dimension. Parameters: m and b.

$y = \mathbf{W}x + \mathbf{b}$ for x being a vector. Parameters: **W** and **b**.

If y has N values and x has M values, **W** is a (NxM) matrix. **b** is a vector of size N:

$$y^{(N\times1)} = \mathbf{W}^{(N\times M)}\, x^{(M\times1)} + \mathbf{b}^{(N\times1)}$$

Activation Functions

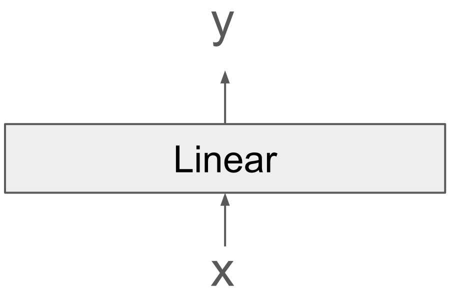
Our function is linear, but we can add an additional function to our output to make our function have useful properties:

$y = \mathbf{W}x + \mathbf{b}$	$y = \mathrm{sigmoid}(\mathbf{W}x + \mathbf{b})$	$y = \mathrm{softmax}(\mathbf{W}x + \mathbf{b})$
Linear Regression	Logistic Regression	General Classification
Learns a linear relationship between x and y.	Learns a boundary between two classes, represented as 0 and 1.	Learns a probability distribution over multiple discrete labels.

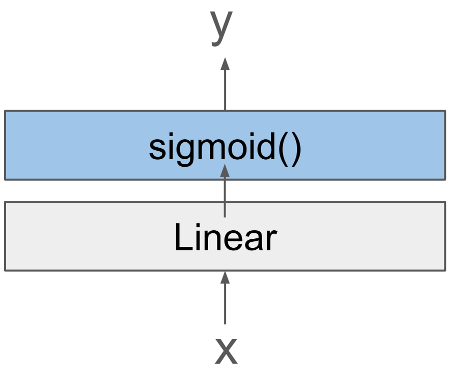
Activation Functions

Our function is linear, but we can add an additional function to our output to make our function have useful properties:

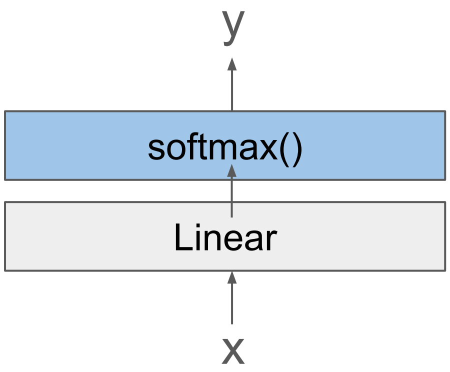
y = **W**x + **b**



y = sigmoid(**W**x + **b**)



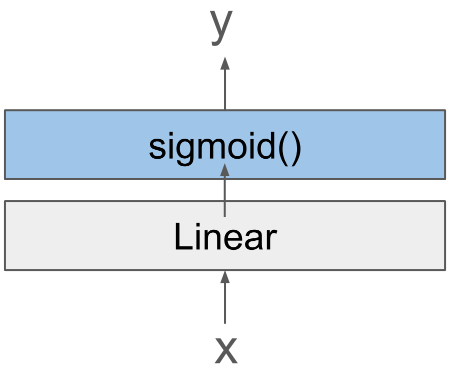
y = softmax(**W**x + **b**)



Deep Functions

Once we think about constructing functions by chaining together lower-level functions (“layers”), we can easily construct all sorts of complex functions:

When our function has more than one layer which has parameters, it is considered “deep”.



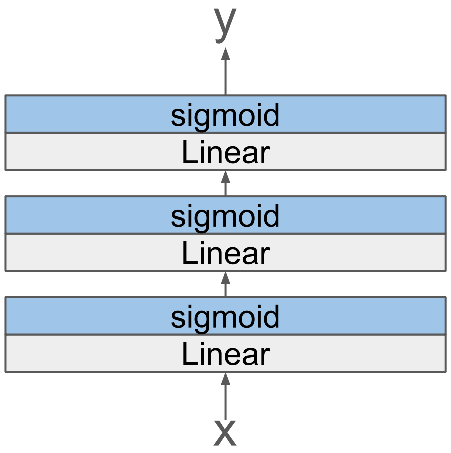
y

sigmoid()

Linear

x

$y = \sigma(Wx+b)$



y

sigmoid

Linear

sigmoid

Linear

sigmoid

Linear

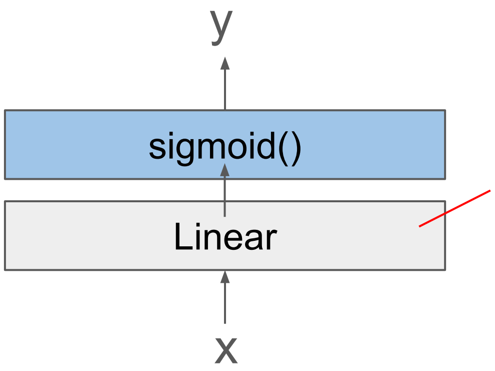
x

$y = \sigma(W_3\sigma(W_2\sigma(W_1x+b_1)+b_2)+b_3)$

Deep Functions

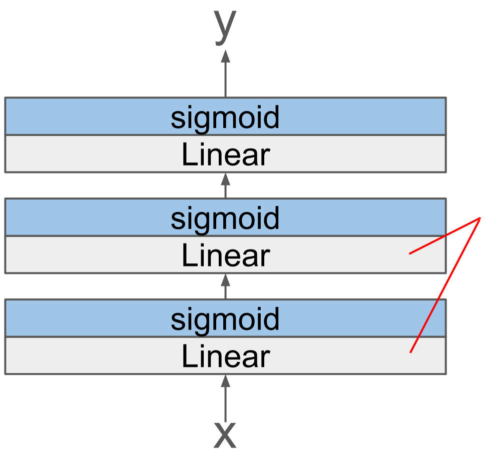
Once we think about constructing functions by chaining together lower-level functions (“layers”), we can easily construct all sorts of complex functions:

When our function has more than one layer which has parameters, it is considered “deep”.



The size of this layer is constrained to match x and y.

$y = \sigma(Wx+b)$



These layers are “hidden”, i.e. output not influenced by size of x and y. We can make them whatever we want as long as they go together.

$y = \sigma(W_3\sigma(W_2\sigma(W_1x+b_1)+b_2)+b_3)$

Training Deep Networks

Almost all functions that are constructed this way (a.k.a. “deep neural networks” a.k.a. a “model”) are trained with stochastic gradient descent (SGD).

SGD is a fairly straightforward algorithm at a high level:

```
Repeat until converged:
    Sample a few (x,y) from your dataset (a “minibatch”)
    Feed x through your model to get a predicted y-value, $y_{pred}$
    Compare $y_{pred}$ with the true y value and compute some error between them (the “loss”, L)
    For each parameter, compute dL/dp (the change in the loss with respect to the change in the parameter).
        The collection of dL/dp for all parameters is called “the gradient”, written as $\nabla L$
    Update all parameters to minimize the loss: p = p - (learning_rate)*(dL/dp)
```

Training Deep Networks

Almost all functions that are constructed this way (a.k.a. “deep neural networks” a.k.a. a “model”) are trained with stochastic gradient descent (SGD).

SGD is a fairly straightforward algorithm at a high level:

```
Repeat until converged:
    Sample a few (x,y) from your dataset (a “minibatch”)
    Feed x through your model to get a predicted y-value, $y_{pred}$
    Compare $y_{pred}$ with the true y value and compute some error between them (the “loss”, L)
    For each parameter, compute dL/dp (the change in the loss with respect to the change in the parameter).
        The collection of dL/dp for all parameters is called “the gradient”, written as $\nabla L$
    Update all parameters to minimize the loss: p = p - (learning_rate)*(dL/dp)
```

This is the most challenging step. However, tools like pytorch do this automatically via backpropogation.

Backpropogation

If we have constructed our model by chaining together functions, we can calculate our gradient in reverse order (backwards through the model) using the chain rule.

Suppose $y_{pred} = f(g(h(x)))$, with parameters W_f, W_g, W_h

First we look at f(), and pretend we just have y = f(input_to_f):

$$\frac{dL}{dW_f} \qquad \frac{dL}{d[input\_to\_f]}$$

Gradient for f()

Used later for g(). Remember, the input to f() is the output of g()!

Backpropogation

If we have constructed our model by chaining together functions, we can calculate our gradient in reverse order (backwards through the model) using the chain rule.

Suppose $y_{pred} = f(g(h(x)))$, with parameters W_f, W_g, W_h

Then we can look at g(), and pretend we have g(input_to_g). By the chain rule:

$$\frac{dL}{dW_g} = \frac{dL}{d[input\_to\_f]} \frac{d[input\_to\_f]}{dW_g}$$

Gradient for g()

$$\frac{dL}{d[input\_to\_g]} = \frac{dL}{d[input\_to\_f]} \frac{d[input\_to\_f]}{d[input\_to\_g]}$$

Used later for h()

Backpropogation

If we have constructed our model by chaining together functions, we can calculate our gradient in reverse order (backwards through the model) using the chain rule.

Suppose y_{pred} = f(g(h(x))), with parameters W_f, W_g, W_h

Then we can look at h(x). By the chain rule:

$$\frac{dL}{dW_h} = \frac{dL}{d[input\_to\_g]} \frac{d[input\_to\_g]}{dW_h}$$

Gradient for h()

$$\frac{dL}{dx} = \frac{dL}{d[input\_to\_g]} \frac{d[input\_to\_g]}{dx}$$

Not needed typically.

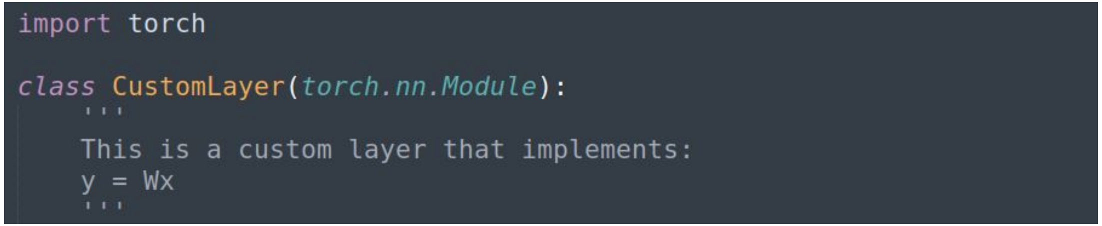
Deep Learning in Pytorch

(Intro to Homework 1)

PyTorch Modules

A layer is represented by a module.

PyTorch provides lots of layers for us, but we can easily define our own.



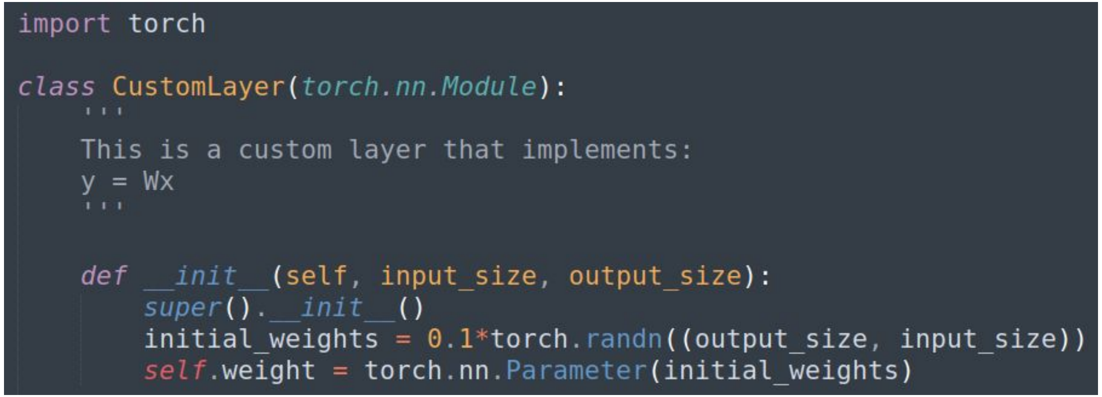
```python
import torch

class CustomLayer(torch.nn.Module):
    '''
    This is a custom layer that implements:
    y = Wx
    '''
```

PyTorch Modules

A layer is represented by a module.

PyTorch provides lots of layers for us, but we can easily define our own.

We need to instantiate parameters in the __init__() method.



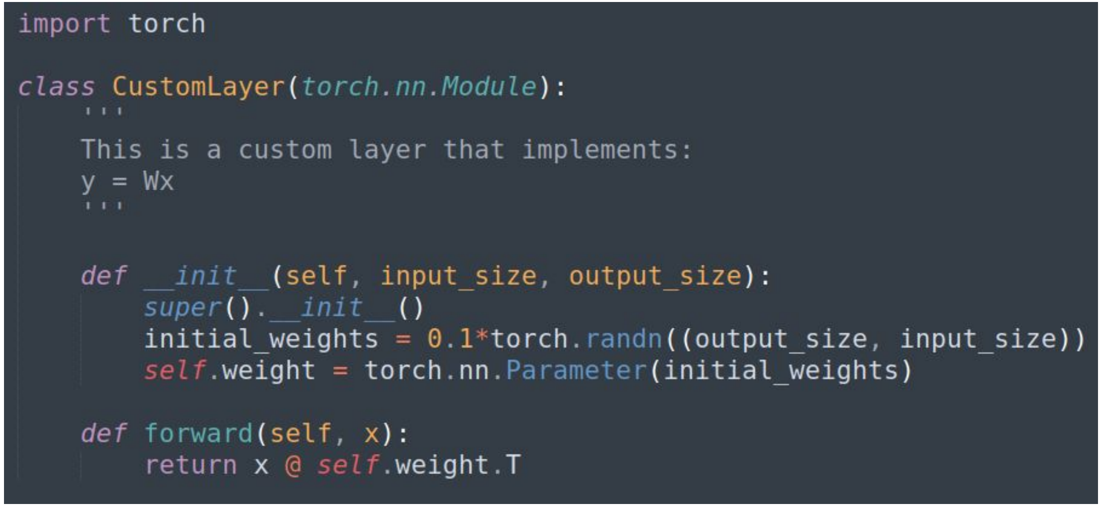
```
import torch

class CustomLayer(torch.nn.Module):
    '''
    This is a custom layer that implements:
    y = Wx
    '''

    def __init__(self, input_size, output_size):
        super().__init__()
        initial_weights = 0.1*torch.randn((output_size, input_size))
        self.weight = torch.nn.Parameter(initial_weights)
```

PyTorch Modules

A layer is represented by a module.

PyTorch provides lots of layers for us, but we can easily define our own.

We need to instantiate parameters in the __init__() method, and how they are used computationally in the forward() method.



```
import torch

class CustomLayer(torch.nn.Module):
    '''
    This is a custom layer that implements:
    y = Wx
    '''

    def __init__(self, input_size, output_size):
        super().__init__()
        initial_weights = 0.1*torch.randn((output_size, input_size))
        self.weight = torch.nn.Parameter(initial_weights)

    def forward(self, x):
        return x @ self.weight.T
```

Note: The @ operator is a shorthand in torch and numpy for matrix multiplication. See torch.matmul() as an alternative notation.

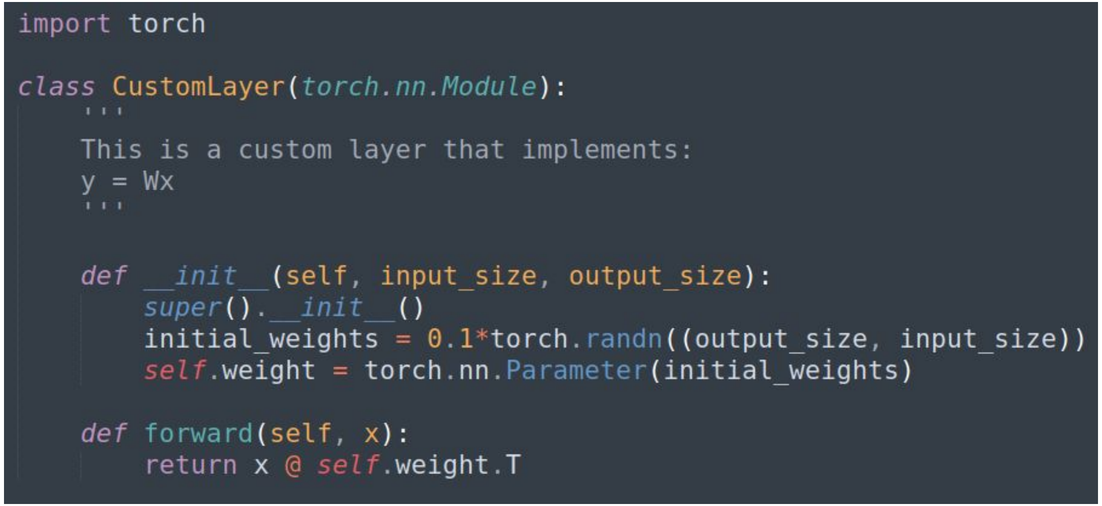
Note on shapes

Note the way the forward method is written:

We typically write Wx on paper, but it is convenient in code to write xW^T.

This lets us add dimensions to x (i.e. batch dimension) without trouble:

$$y^{(B,N)} = x^{(B,M)} W^{T\ (M,N)}$$



```python
import torch

class CustomLayer(torch.nn.Module):
    '''
    This is a custom layer that implements:
    y = Wx
    '''

    def __init__(self, input_size, output_size):
        super().__init__()
        initial_weights = 0.1*torch.randn((output_size, input_size))
        self.weight = torch.nn.Parameter(initial_weights)

    def forward(self, x):
        return x @ self.weight.T
```

Note: The @ operator is a shorthand in torch and numpy for matrix multiplication. See torch.matmul() as an alternative notation.

Homework 1

The previous slide is 90% of homework 1.

You will need to implement $y = Wx + b$ instead.

For this class we will not implement backpropogation (that is a bit too low-level). We will let pytorch do this for us.

Training in Pytorch

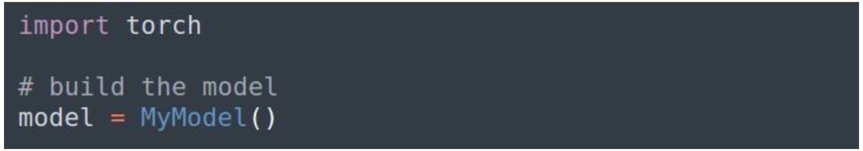
(For Completeness)

Training Modules in PyTorch

You are not responsible for training your custom layer in homework 1, but we will do training in later weeks.

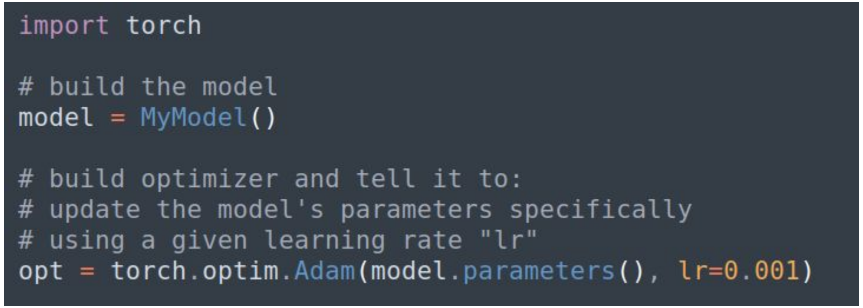
To train a module in pytorch, we:

1) Build the model



```python
import torch

# build the model
model = MyModel()
```

Training Modules in PyTorch



```python
import torch

# build the model
model = MyModel()

# build optimizer and tell it to:
# update the model's parameters specifically
# using a given learning rate "lr"
opt = torch.optim.Adam(model.parameters(), lr=0.001)
```

You are not responsible for training your custom layer in homework 1, but we will do training in later weeks.

To train a module in pytorch, we:

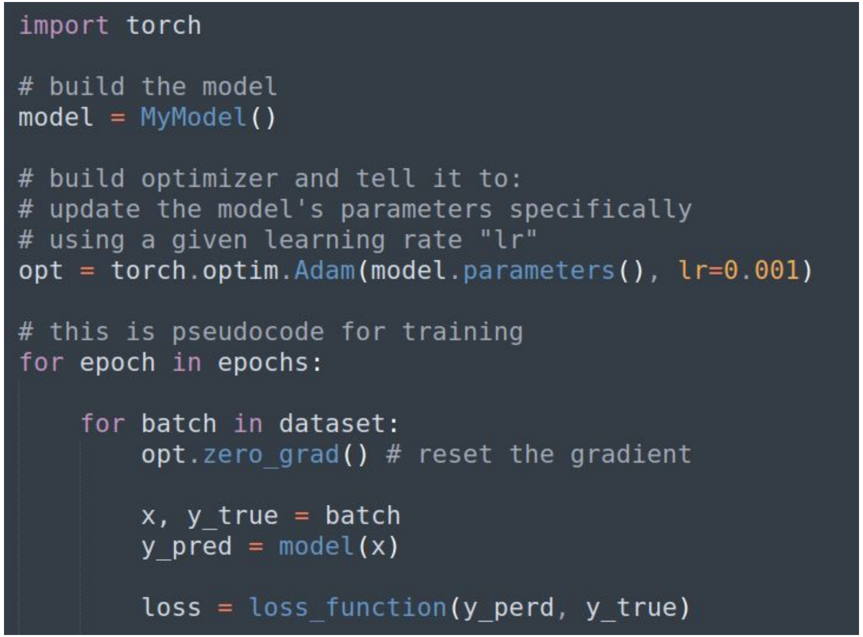
1) Build the model
2) Build an optimizer (handles SGD for us)

Training Modules in PyTorch

You are not responsible for training your custom layer in homework 1, but we will do training in later weeks.

To train a module in pytorch, we:

1) Build the model
2) Build an optimizer (handles SGD for us)
3) Sample minibatches and compute loss in a loop over our data



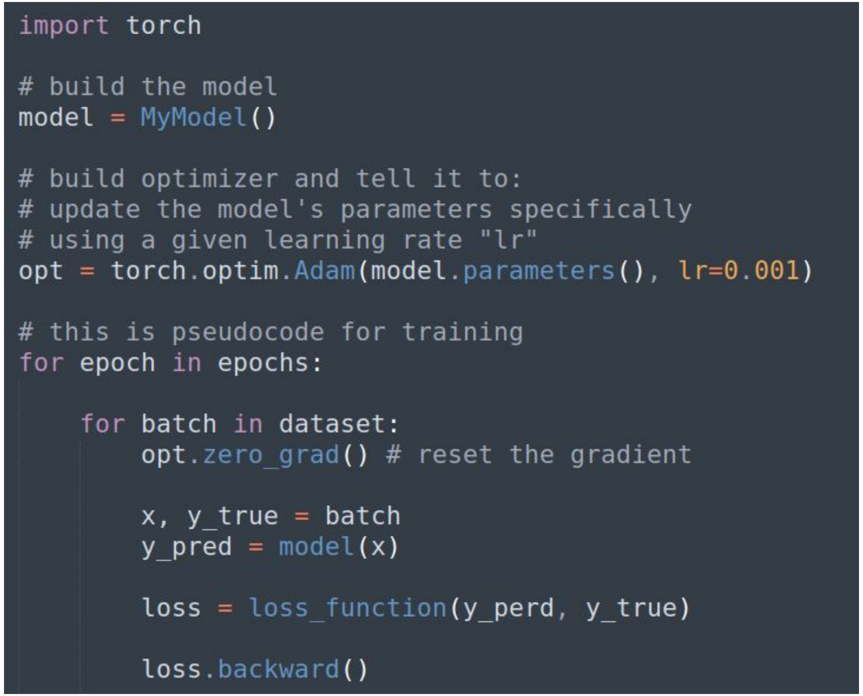
```
import torch

# build the model
model = MyModel()

# build optimizer and tell it to:
# update the model's parameters specifically
# using a given learning rate "lr"
opt = torch.optim.Adam(model.parameters(), lr=0.001)

# this is pseudocode for training
for epoch in epochs:

    for batch in dataset:
        opt.zero_grad() # reset the gradient

        x, y_true = batch
        y_pred = model(x)

        loss = loss_function(y_perd, y_true)
```

Training Modules in PyTorch

You are not responsible for training your custom layer in homework 1, but we will do training in later weeks.

To train a module in pytorch, we:

1) Build the model
2) Build an optimizer (handles SGD for us)
3) Sample minibatches and compute loss in a loop over our data
4) Backpropogate the loss to compute a gradient



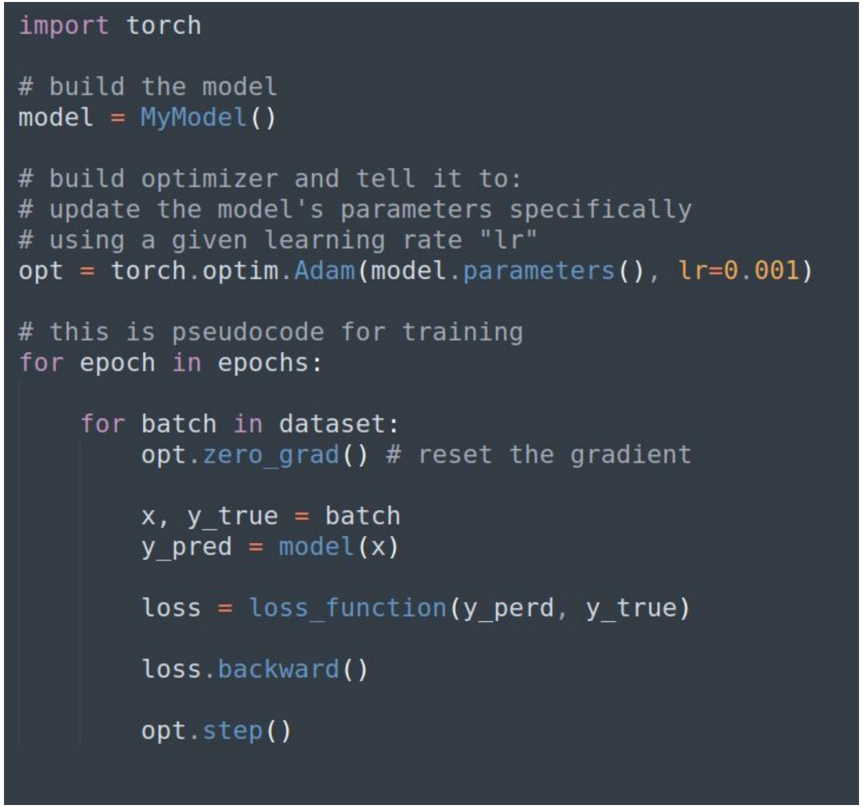
```python
import torch

# build the model
model = MyModel()

# build optimizer and tell it to:
# update the model's parameters specifically
# using a given learning rate "lr"
opt = torch.optim.Adam(model.parameters(), lr=0.001)

# this is pseudocode for training
for epoch in epochs:

    for batch in dataset:
        opt.zero_grad() # reset the gradient

        x, y_true = batch
        y_pred = model(x)

        loss = loss_function(y_perd, y_true)

        loss.backward()
```

Training Modules in PyTorch

You are not responsible for training your custom layer in homework 1, but we will do training in later weeks.

To train a module in pytorch, we:

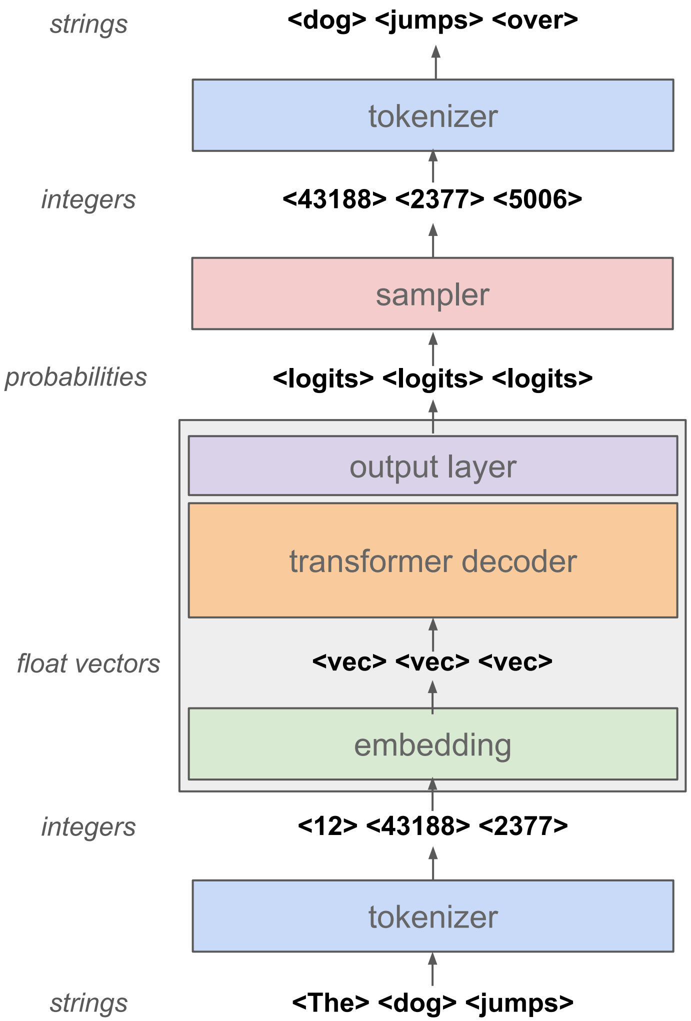
1) Build the model
2) Build an optimizer (handles SGD for us)
3) Sample minibatches and compute loss in a loop over our data
4) Backpropogate the loss to compute a gradient
5) Update the model with optimizer.step()



```
import torch

# build the model
model = MyModel()

# build optimizer and tell it to:
# update the model's parameters specifically
# using a given learning rate "lr"
opt = torch.optim.Adam(model.parameters(), lr=0.001)

# this is pseudocode for training
for epoch in epochs:

    for batch in dataset:
        opt.zero_grad() # reset the gradient

        x, y_true = batch
        y_pred = model(x)

        loss = loss_function(y_perd, y_true)

        loss.backward()

        opt.step()
```

End of Lecture 1

Review HW1 Together



strings **<dog> <jumps> <over>**

tokenizer

integers **<43188> <2377> <5006>**

sampler

probabilities **<logits> <logits> <logits>**

output layer

transformer decoder

float vectors **<vec> <vec> <vec>**

embedding

integers **<12> <43188> <2377>**

tokenizer

strings **<The> <dog> <jumps>**