AI INSTITUTE #AIISC
UNIVERSITY OF SOUTH CAROLINA

UNIVERSITY OF SOUTH CAROLINA

CSCE 771: Computer Processing of Natural Language
Lecture: Pytorch and BERT

PROF. BIPLAV SRIVASTAVA, AI INSTITUTE

24th SEP 2024

Carolinian Creed: "I will practice personal and academic integrity."

Organization of Lecture 11

- Opening Segment
  - Review of Lecture 10
- Main Lecture
- Concluding Segment
  - About Next Lecture – Lecture 12

Main Section
- PyTorch
- BERT

Recap of Lecture 10

- We reviewed Machine Learning methods
  - Data preparation is the key
  - Watch out for evaluation
  - ML is just a step, what happens to the model is also important
- Language models
  - We reviewed connections between parsing and language model
  - We discussed language models and are focusing on pre-requisites needed to understand them
  - We discussed contextual word representations as a stepping stone

Main Lecture

Kaushik Roy

South Carolina

University of South Carolina

[Webpage]

I am Kaushik Roy, a Ph.D. candidate at the Artificial Intelligence Institute, University of South Carolina. My research focuses on developing neurosymbolic methods for declarative and process knowledge-infused learning, reasoning, and sequential decision-making, with a particular emphasis on social good applications. My academic journey has taken me from R.V. College of Engineering in Bangalore for my Bachelor's to Indiana University Bloomington for my Master's, and briefly to the University of Texas at Dallas before settling at the University of South Carolina for my doctoral studies. My research interests span machine learning, artificial intelligence, and their application in social good settings. I'm passionate about pushing the boundaries of AI, particularly in areas where it intersects with human understanding and decision-making.

Topics of Interest to Me: Neurosymbolic AI Knowledge-infused Learning AI for Social Good Healthcare Informatics

Organization of This Lecture
Part 1: Pytorch

- Introduction Section
  - Pytorch
- Main Section

Main Section

- What is a Pytorch?
  - Old-school differentiation
  - Automatic differentiation
- Processing data
  - Scalars
  - Vectors, Matrices
  - Tensors

Organization of This Lecture
PART 2: BERT

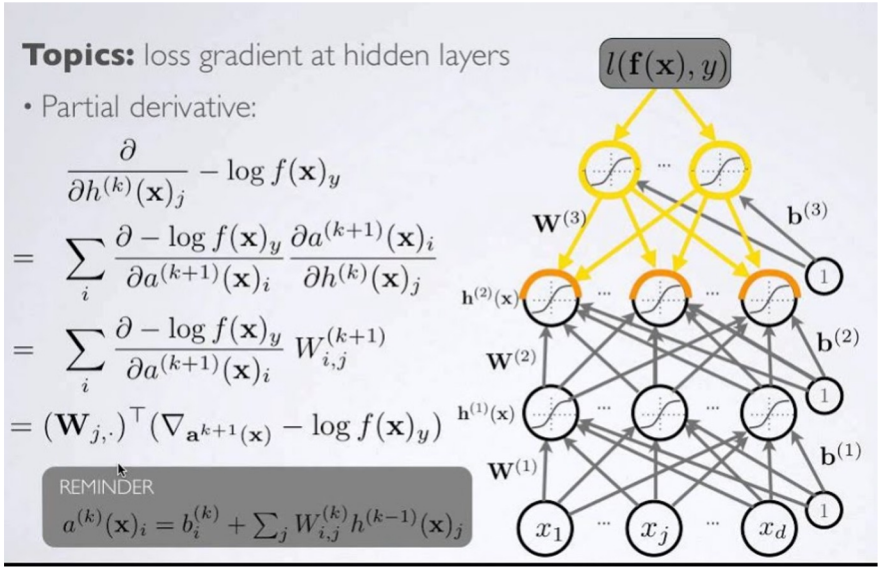
- Introduction Section
  - Language Modeling
- Main Section 
- Concluding Section
    - Next lecture Mamba

Main Section

- What is Language Modeling?
  - BERT-and family
  - Live Coding - Two CV tasks
- Concluding Comments
  - BERT lib on CV
  - BERT scratch on CV
  - Abstract Math Objects

Image Source

What is Pytorch?



Old-school differentiation

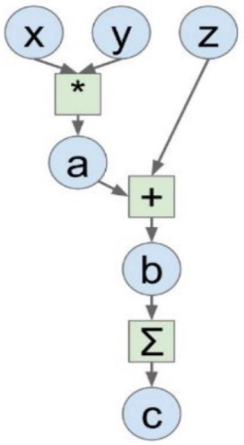
- Processing data
  - Scalars?
  - Vectors, Matrices?
  - Tensors?
- Abstract Mathematical Objects?

- Prior to libraries such as pytorch, we needed to hand derive the derivatives of neural network architectures, which could end up being prone to manual errors!

Image Source

Why is Pytorch?

Computation Graph




```
import torch

N, D = 3, 4

x = torch.rand((N, D),requires_grad=True)
y = torch.rand((N, D),requires_grad=True)
z = torch.rand((N, D),requires_grad=True)

a =x * y
b =a + z
c=torch.sum(b)

c.backward()
```

Automating differentiation

- Processing data
  - Scalars ✓
  - Vectors, Matrices ✓
  - Tensors ✓
- Abstract Mathematical Objects?

- Pytorch allows arbitrary compositions of mathematical object as functions, while also simultaneously and **automatically** maintaining the derivative of the function with respect to it's parameters!

Brief History

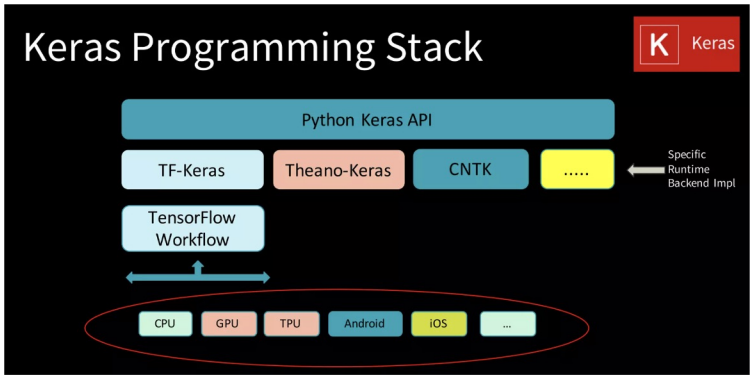
- **Big data**
  - Very large volumes of raw data
- **Deep Learning boom**

Image Source

Automatic Differentiation

- Tensorflow
  - Released by Google
  - Used software artifacts such as constants, variables, placeholders, operations, sessions, tensors, graphs
  - Hugely popular, but a bit difficult to manage (complex) – APIs built to make it easier (Keras)

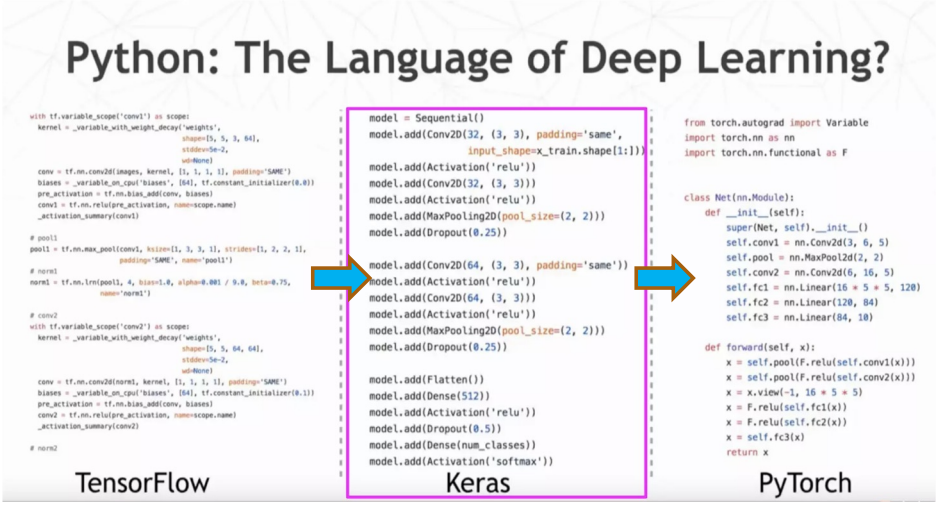
Image Source



Brief History

- **Python as language of deep learning**
  - Many libraries in python
- **Python-inspired syntax**

Image Source



Automatic Differentiation

- Pytorch
  - Released by Meta (Formerly Facebook)
  - Used software artifacts such as Tensors of various dimensions, optimizers, and extremely intuitive due to it's **pythonic** nature
  - Hugely popular, and today most popular for **rapid research prototyping**

Image Source

Tiny Example

Autograd

- Automatic Differentiation Package
- Don't need to worry about partial differentiation, chain rule etc.
  - `backward()` does that
- Gradients are accumulated for each step by default:
  - Need to zero out gradients after each update
  - `tensor.grad_zero()`

```
# Create tensors.
x = torch.tensor(1., requires_grad=True)
w = torch.tensor(2., requires_grad=True)
b = torch.tensor(3., requires_grad=True)

# Build a computational graph.
y = w * x + b    # y = 2 * x + 3

# Compute gradients.
y.backward()

# Print out the gradients.
print(x.grad)    # x.grad = 2
print(w.grad)    # w.grad = 1
print(b.grad)    # b.grad = 1
```

- This code shows a simple composition of scalars into a linear function
- Note how we never specify the gradients manually, and **pytorch automatically calculates it during a call to .backward()**

Image Source

Tiny Example

Optimizer and Loss

Optimizer

- Adam, SGD etc.
- An optimizer takes the parameters we want to update, the learning rate we want to use along with other hyper-parameters and performs the updates

Loss

- Various predefined loss functions to choose from
- L1, MSE, Cross Entropy

```
a = torch.randn(1, requires_grad=True, dtype=torch.float, device=device)
b = torch.randn(1, requires_grad=True, dtype=torch.float, device=device)


# Defines a SGD optimizer to update the parameters
optimizer = optim.SGD([a, b], lr=lr)

for epoch in range(n_epochs):
    yhat = a + b * x_train_tensor
    error = y_train_tensor - yhat
    loss = (error ** 2).mean()

    loss.backward()

    optimizer.step()

    optimizer.zero_grad()

print(a, b)
```

- Pytorch allows a wide range of optimization techniques, such as **stochastic gradient descent (SGD)**, already built in it's optimizer toolkit

Image Source

Tiny Example

Model

In PyTorch, a model is represented by a regular Python class that inherits from the Module class.

- Two components
  - `__init__(self)` : it defines the parts that make up the model- in our case, two parameters, a and b
  - `forward(self, x)` : it performs the actual computation, that is, it outputs a prediction, given the inputx

```
class ManualLinearRegression(nn.Module):
    def __init__(self):
        super().__init__()
        # To make "a" and "b" real parameters of the model, we need to wrap them with nn.Parameter
        self.a = nn.Parameter(torch.randn(1, requires_grad=True, dtype=torch.float))
        self.b = nn.Parameter(torch.randn(1, requires_grad=True, dtype=torch.float))

    def forward(self, x):
        # Computes the outputs / predictions
        return self.a + self.b * x
```

- A pytorch model is easily specified as a class
- Next we will demonstrate how effective pytorch is using the CV dataset

Data - CV data as a PDF

Follow Links

Links to my Google Scholar and Socials

South Carolina
University of South Carolina
Email
Google Scholar
Github
LinkedIn

Curriculum Vitae, Updated September 14th 2024.

🎓 Education

- **2020 – Present**: Ph.D. Candidate, University of South Carolina
  - Topic: Process Knowledge-infused Learning and Reasoning
- **2017 – 2020**: Ph.D. Student, University of Texas at Dallas (Transferred in 2020)
  - Topic: Relational Sequential Decision Making [Qualifier Document]
- **2015 – 2017**: M.Sc. Computer Science, Indiana University Bloomington
  - Specialization: Artificial Intelligence and Machine Learning
- **2011 – 2015**: B.E. Computer Science, RV College of Engineering
  - Thesis title: Computer Vision Algorithms for Background Understanding [Thesis Document]

📚 Publications

📘 Book Chapters

Tasks

- CV-related tasks
  - Extractive Summarization – Extract Institution, Degree, and Year
    - Institution: UofSC
    - Degree: Ph.D.
    - Year: 2024
  - Abstractive Summarization - Masked Language Filling with Partial Queries
    - Kaushik Roy is a ?__

Data - Preprocessing Steps

Key Steps in Preprocessing the CV Text:

1. **Convert the PDF into Plain Text**:
   - Use a library like **pdfplumber** to extract the text from the PDF file.
2. **Text Cleaning**:
   - Remove or replace unnecessary characters such as newline characters (\n), multiple spaces, tabs (\t), etc.
   - Normalize any irregular characters.
3. **Segment the Text**:
   - Break the text into sentences or paragraphs, which can be useful when preparing data for sequence-based tasks.
4. **Tokenization**:
   - Use a tokenizer like **BERT's tokenizer** to convert the cleaned text into a format that can be used by your model.

Step 1 - Convert PDF into Plain Text

You can use **pdfplumber** to extract text from a PDF file. The extracted text may contain newline characters, extra spaces, and other artifacts that need to be cleaned.

Code Snapshot:

```python
import pdfplumber

def extract_text_from_pdf(pdf_path):
    """
    Extract text from a PDF file using pdfplumber.

    Args:
        pdf_path (str): Path to the PDF file.

    Returns:
        str: Extracted text from the PDF.
    """
    with pdfplumber.open(pdf_path) as pdf:
        text = ''
        for page in pdf.pages:
            text += page.extract_text()  # Extract text from each p
    return text
```

Step 2 - Text Cleaning

Once you've extracted the text from the PDF, it might contain newlines, multiple spaces, and other irregular characters that need to be cleaned. Here's what we can do:

- **Remove newline characters** (\n) and replace them with spaces (or punctuation marks like periods, depending on how the text is structured).
- **Normalize white spaces**: Collapse multiple spaces into a single space.
- **Remove special characters**: Remove characters like tabs, and optionally remove non-alphabetic characters.

Code Snapshot:

```python
import re

def clean_text(text):
    """
    Clean the extracted text by removing newlines, multiple spaces,

    Args:
        text (str): The raw extracted text.

    Returns:
        str: Cleaned text.
    """
    # Replace newline characters with spaces
    text = text.replace('\n', ' ').replace('\r', ' ')

    # Remove tabs and multiple spaces
    text = re.sub(r'\s+', ' ', text)  # Replace multiple spaces wi

    # Optionally remove non-alphanumeric characters (if needed)
    # text = re.sub(r'[^a-zA-Z0-9.,!?\'\s]', '', text)

    # Strip leading/trailing spaces
    text = text.strip()

    return text
```

Step 3 - Segment the Text

- Depending on how you want to use the text in your model, you might want to segment it into smaller units like **sentences** or **paragraphs**.
- If the extracted text is structured, you can split it into sentences using **nltk** or **spaCy** to make the text more manageable.
- **Code Example for Segmenting Text into Sentences Using nltk:**

Code Snapshot:

```
import nltk
nltk.download('punkt')  # Download the sentence tokenizer

def segment_text_into_sentences(text):
    """
    Segment the cleaned text into sentences.

    Args:
        text (str): The cleaned text.

    Returns:
        list: A list of sentences.
    """
    from nltk.tokenize import sent_tokenize
    sentences = sent_tokenize(text)  # Tokenize the text into sent
    return sentences
```

Step 4 - Tokenization

- Once the text is cleaned and segmented, you can tokenize it using **BERT's tokenizer** or any other suitable tokenizer.

Code Snapshot:

```
from transformers import BertTokenizer

# Load the BERT tokenizer
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

def tokenize_text(sentences):
    """
    Tokenize a list of sentences using the BERT tokenizer.

    Args:
        sentences (list): A list of sentences.

    Returns:
        list: A list of tokenized sentences (token IDs).
    """
    tokenized_sentences = [tokenizer.encode(sentence, return_tensor
    return tokenized_sentences
```

Live Coding

Github:
https://github.com/kauroy1994/Teaching
(Neural Network-based CV information filling)

- Simply using a feedforward neural network with a position embedding layer to solve the CV tasks

Code Snapshot:

```
class generator(nn.Module):

  def __init__(self,
               n_tokens = None,
               emb_size = None,
               context_size = None,
               n_layers = 2,
               h_size = 100):

    super().__init__()

    self.n_tokens = n_tokens
    self.emb_size = emb_size
    self.context_size = context_size
    self.n_layers = n_layers
    self.h_size = h_size

    self.embeddings = nn.Embedding(self.n_tokens, self.emb_size)
    self.pos_embeddings = nn.Embedding(self.context_size, self.emb_size)

    self.fc1 = nn.Linear(self.emb_size,self.h_size,bias=False)
    self.fc2 = nn.Linear(self.h_size,self.h_size,bias=False)
    self.head = nn.Linear(self.h_size,self.n_tokens)

  def forward(self,
              token_encodings):

    n_tokens = len(token_encodings)
    token_encodings = torch.tensor(token_encodings)
    token_encodings.to(device)
    token_embeddings = self.embeddings(token_encodings)
    pos_embeddings = self.pos_embeddings(torch.arange(n_tokens))
    token_embeddings += pos_embeddings

    reps = token_embeddings
    reps = F.leaky_relu(self.fc1(reps))
    reps = F.leaky_relu(self.fc2(reps))
    reps = self.head(reps)

    logits = reps[-1]
    return logits
```

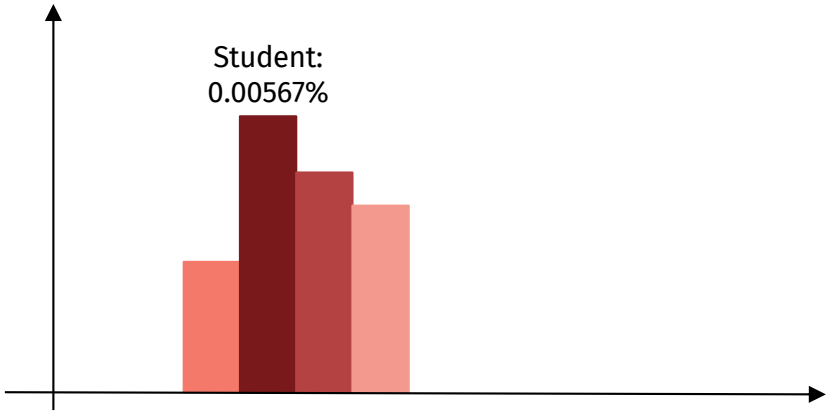
Language Modeling

Input Kaushik Roy is a PhD Student

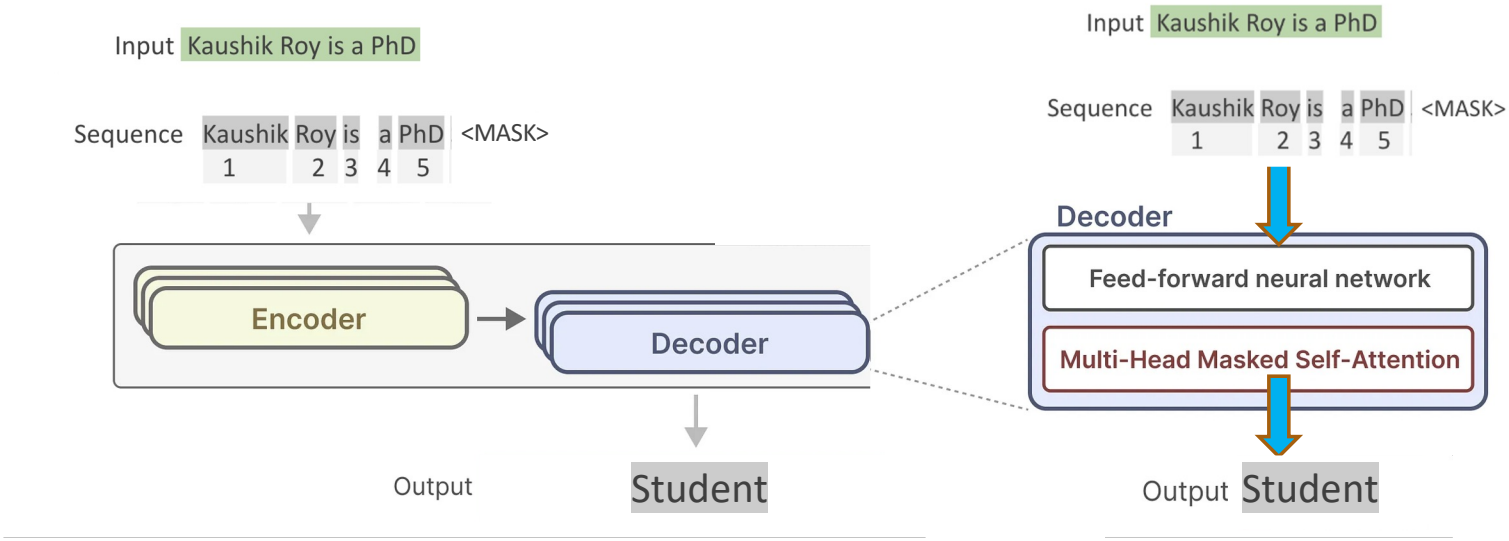
Sequence	Kaushik	Roy	is	a	PhD	Student
	1	2	3	4	5	6

Student

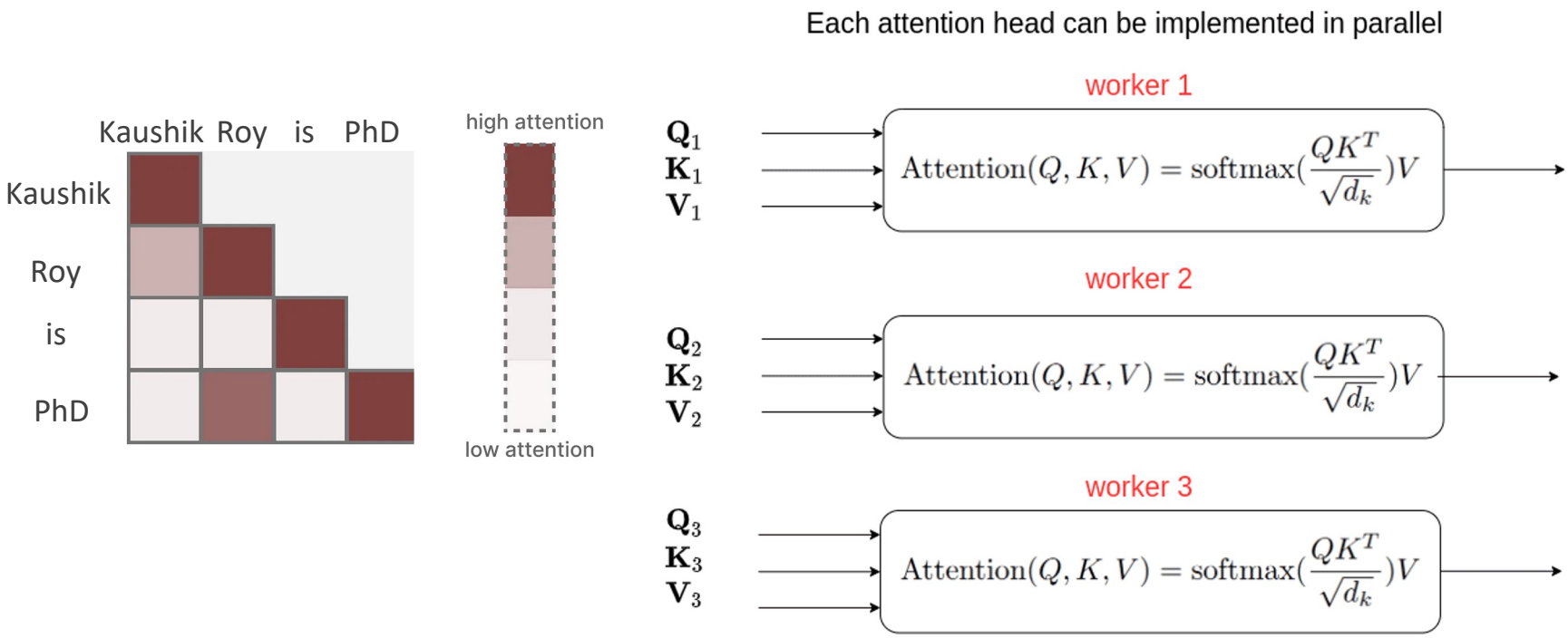
Did you mean: Student
Did you mean: Student Square
Did you mean: Student Success



BERT and family



BERT and family - Multi-headed Self-Attention



Self Attention Snippet and Live Coding - BERT from Scratch

```
class SelfAttention(nn.Module):
    def __init__(self, embed_size, heads):
        super(SelfAttention, self).__init__()
        self.embed_size = embed_size
        self.heads = heads
        self.head_dim = embed_size // heads

        assert (
            self.head_dim * heads == embed_size
        ), "Embedding size needs to be divisible by heads"

        self.values = nn.Linear(self.head_dim, embed_size, bias=False)
        self.keys = nn.Linear(self.head_dim, embed_size, bias=False)
        self.queries = nn.Linear(self.head_dim, embed_size, bias=False)
        self.fc_out = nn.Linear(embed_size, embed_size)
```

BERT-based CV Processing

Live Coding - BERT using Libraries

Github: https://github.com/kauroy1994/Teaching (BERT-based CV Processing)

- Use the transformers library to extract information from the CV

```python
import pdfplumber
from transformers import AutoTokenizer, AutoModelForTokenClassification
from transformers import pipeline

# Step 1: Load the CV PDF and extract text
def extract_text_from_pdf(pdf_path):
    with pdfplumber.open(pdf_path) as pdf:
        pages = [page.extract_text() for page in pdf.pages]
    return ''.join(pages)

# Extract text from the CV
pdf_path = "CV.pdf"
cv_text = extract_text_from_pdf(pdf_path)

# Step 2: Load pre-trained BERT model and tokenizer
tokenizer = AutoTokenizer.from_pretrained("dslim/bert-base-NER")
model = AutoModelForTokenClassification.from_pretrained("dslim/bert-base-NER")

# Step 3: Use pipeline for Named Entity Recognition
nlp = pipeline("ner", model=model, tokenizer=tokenizer, grouped_entities=True)

# Step 4: Extract entities from the CV text
ner_results = nlp(cv_text)

# Display the recognized entities
for entity in ner_results:
    print(f"Entity: {entity['word']}, Label: {entity['entity_group']}")

# Step 5: Post-process the entities for CV data extraction (optional)
# For example, grouping entities like degree, institution, and dates
def extract_education_details(ner_results):
    education = []
    current_education = {}
    for entity in ner_results:
        if entity['entity_group'] == 'ORG':
            current_education['institution'] = entity['word']
        elif entity['entity_group'] == 'MISC':  # Assuming degrees are labeled as MISC
            current_education['degree'] = entity['word']
        elif entity['entity_group'] == 'DATE':
            current_education['year'] = entity['word']

        # Save the current education entry
        if 'institution' in current_education and 'degree' in current_education and 'year' in current_education:
            education.append(current_education)
            current_education = {}
    return education

education_details = extract_education_details(ner_results)

# Display the structured education data
print("Extracted Education Details:", education_details)
```

Concluding Segment

Concluding Comments

- We saw how model's are constructed and trained using a convenience library called pytorch
- We saw how BERT can be used in two ways for two CV-related tasks
- What about abstract mathematical objects?

TextGrad: Automatic "Differentiation" via Text

An autograd engine -- for textual gradients!

TextGrad is a powerful framework building automatic ``differentiation'' via text. TextGrad implements backpropagation through text feedback provided by LLMs, strongly building on the gradient metaphor

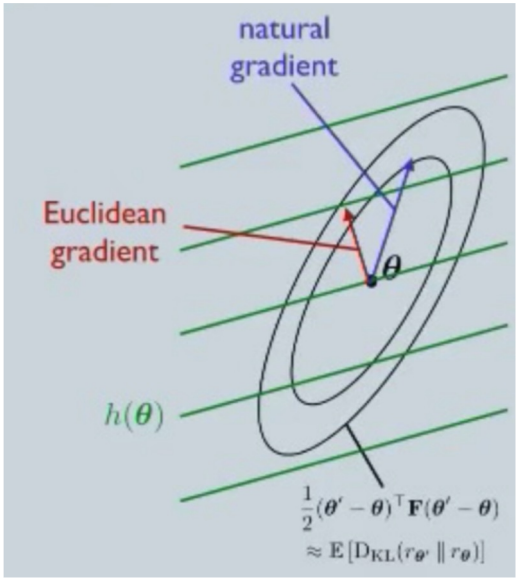
1 Analogy in abstractions

	Math	PyTorch	TextGrad
Input	x	`Tensor(image)`	`tg.Variable(article)`
Model	$\hat{y} = f_\theta(x)$	`ResNet50()`	`tg.BlackboxLLM("You are a summarizer.")`
Loss	$L(y, \hat{y}) = \sum_i y_i \log(\hat{y}_i)$	`CrossEntropyLoss()`	`tg.TextLoss("Rate the summary.")`
Optimizer	$\mathrm{GD}(\theta, \frac{\partial L}{\partial \theta}) = \theta - \frac{\partial L}{\partial \theta}$	`SGD(list(model.parameters()))`	`tg.TGD(list(model.parameters()))`

2 Automatic differentiation

PyTorch and TextGrad share the same syntax for backpropagation and optimization.

Forward pass	Backward pass	Updating variable
`loss = loss_fn(model(input))`	`loss.backward()`	`optimizer.step()`



Course Project

Discussion: Course Project

Theme: Analyze quality of official information available for elections in 2024 [in a state]

- Take information available from
  - Official site: State Election Commissions
  - Respected non-profits: League of Women Voters
- Analyze information
  - State-level: Analyze quality of questions, answers, answers-to-questions
  - Comparatively: above along all states (being done by students)
- Benchmark and report
  - Compare analysis with LLM
  - Prepare report

- Process and analyze using NLP
  - Extract entities
  - Assess quality – metrics
    - Content – *Englishness*
    - Content – *Domain* -- election
  - … other NLP tasks
  - Analyze and communicate overall

Major dates for project check

- Sep 10: written – project outline
- Oct 8: in class
- Oct 31: in class // LLM
- Dec 5: in class // Comparative

About Next Lecture – Lecture 12

Lecture 12 Outline

- Mamba, Finetuning

7	Sep 10 (Tu)	Statistical parsing, QUIZ
8	Sep 12 (Th)	Evaluation, Semantics
9	Sep 17 (Tu)	Semantics, Machine Learning for NLP, Evaluation - Metrics
10	Sep 19 (Th)	Towards Language Model: Vector embeddings, Embeddings, CNN/ RNN
11	Sep 24 (Tu)	Language Model – PyTorch, BERT, {Resume data, two tasks} **– Guest Lecture**
12	Sep 26 (Th)	Language Model – Finetuning, Mamba - **Guest Lecture**
13	Oct 1 (Tu)	Language model – comparing arch, finetuning - **Guest Lecture**
14	Oct 3 (Th)	Language model – comparison of results, discussion, ongoing trends– **Guest Lecture**
15	Oct 8 (Tu)	PROJ REVIEW