AI INSTITUTE #AIISC
UNIVERSITY OF SOUTH CAROLINA

UNIVERSITY OF SOUTH CAROLINA

CSCE 771: Computer Processing of Natural Language

Lecture: Language Modeling Variants – BERT, Mamba

PROF. BIPLAV SRIVASTAVA, AI INSTITUTE

26th SEP 2024

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

Organization of Lecture 12

- Opening Segment
  - Review of Lecture 11
- Main Lecture

Main Section
- BERT and LLM Evaluation
- Evaluating LMs
- RNNs and Family
  - RNNs
  - Mamba
  - Live Coding - Two CV tasks

- Concluding Segment
  - About Next Lecture – Lecture 13

Recap of Lecture 11

- We covered
  - PyTorch
  - Data and Tasks
  - BERT
- We saw
  - how model's are constructed and trained using a convenience library called pytorch
  - how BERT can be used in two ways for two CV-related tasks
- We discussed handling of abstract mathematical objects

Sep 24 (Tu)	Language Model – PyTorch, BERT, {Resume data, two tasks} **– Guest Lecture**
Sep 26 (Th)	Language Model – Finetuning, Mamba - **Guest Lecture**
Oct 1 (Tu)	Language model – comparing arch, finetuning - **Guest Lecture**
Oct 3 (Th)	Language model – comparison of results, discussion, ongoing trends– **Guest Lecture**

Focused Classes

GUEST LECTURES ON LANGUAGE MODELS

About Me

https://github.com/kauroy1994/CSCE-771-NLP-Class/

South Carolina

University of South Carolina

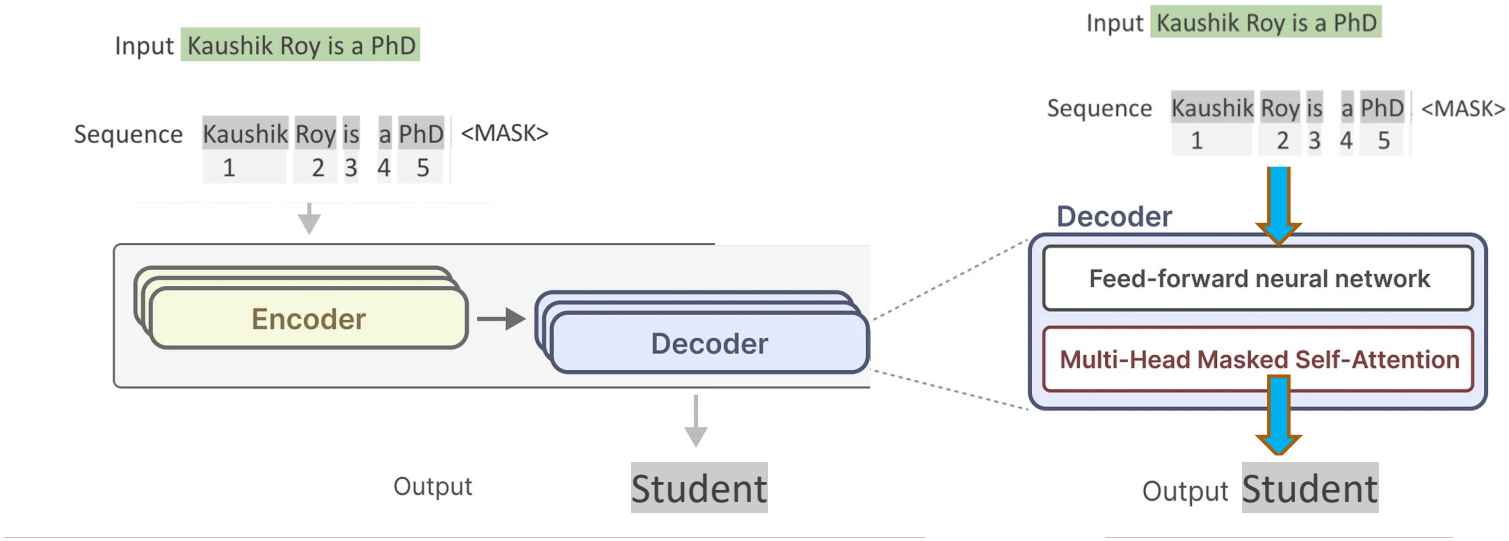
Visit My Webpage

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.

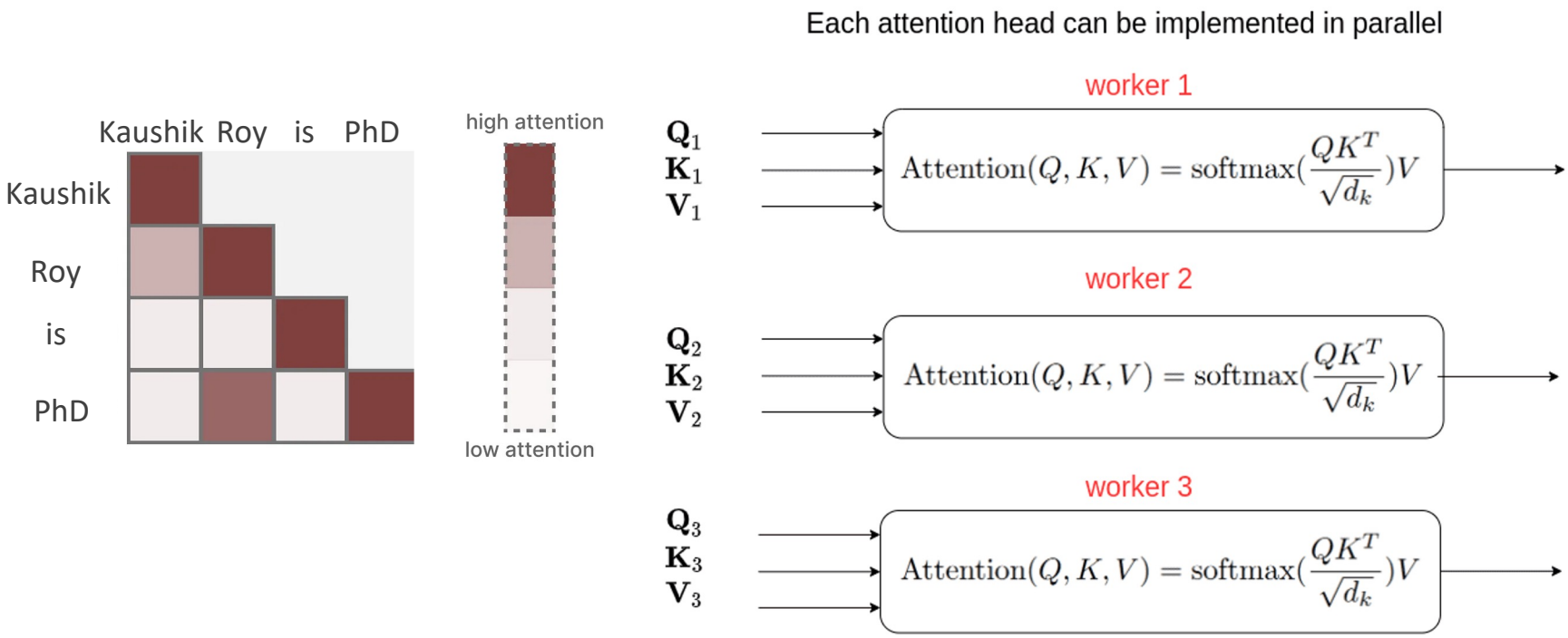
Topics of Interest to Me: Neurosymbolic AI Knowledge-infused Learning AI for Social Good Healthcare Informatics

Main Lecture

BERT and family



BERT and family - Multi-headed Self-Attention



Self Attention Snippet and Live Coding - BERT from Scratch

```python
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/CSCE-771-NLP-Class-11/tree/main (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)
```

Using BERT in Practice – Huggingface Libraries

- Transformers – https://github.com/huggingface/transformers
- APIs to download and use pre-trained models, fine-tune them on own datasets and tasks
  - Code Sample

```
# Loading BERT
model_class, tokenizer_class, pretrained_weights =  (ppb.DistilBertModel,
ppb.DistilBertTokenizer, 'distilbert-base-uncased')

# Load pretrained model/tokenizer
tokenizer = tokenizer_class.from_pretrained(pretrained_weights)
model = model_class.from_pretrained(pretrained_weights)
```

- Provides pretrained models in 100+ languages.
- Use with popular deep learning libraries, PyTorch and TensorFlow,
  - Possible to train / fine-tune models with one, and load it for inference with another

Using BERT in Practice – Huggingface Libraries

- DistilBERT
  - Details: https://medium.com/huggingface/distilbert-8cf3380435b5
  - Teacher-student learning, also called model distillation
    - Teacher: bert-base-uncased
    - Student: dstilBERT - BERT without *the token-type embeddings and the pooler* , and half the layers
  - **"DistilBERT,** has **about half** the total number of parameters of BERT base and retains 95% of BERT's performances on the language understanding benchmark GLUE"

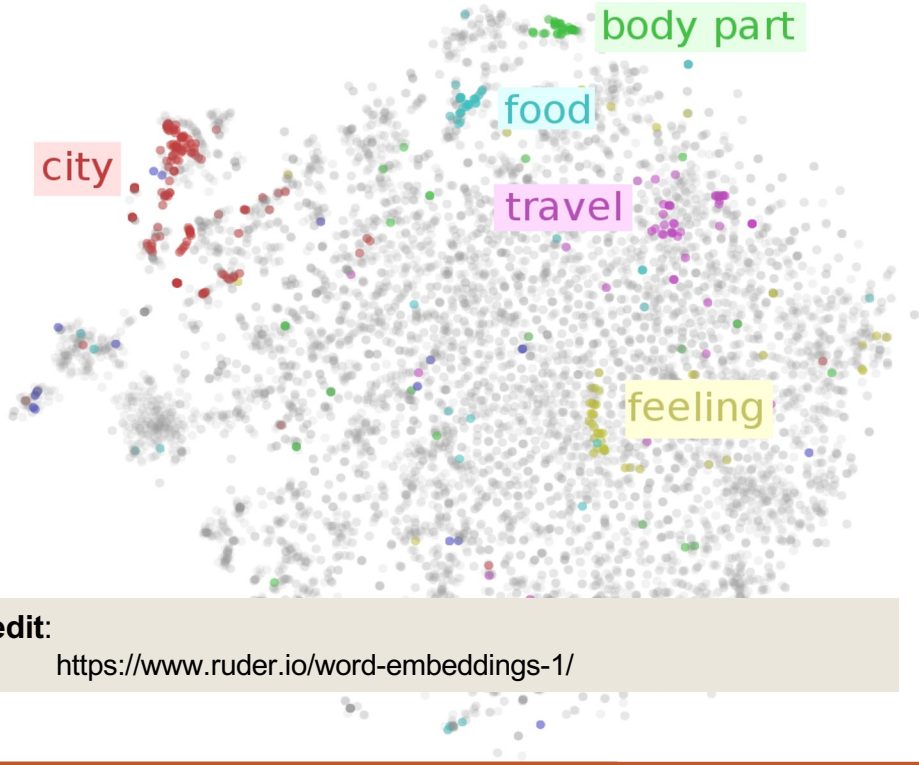
- Sample code of usage for sentiment classification: https://github.com/biplav-s/course-nl/blob/master/l12-langmodel/UsingLanguageModel.ipynb

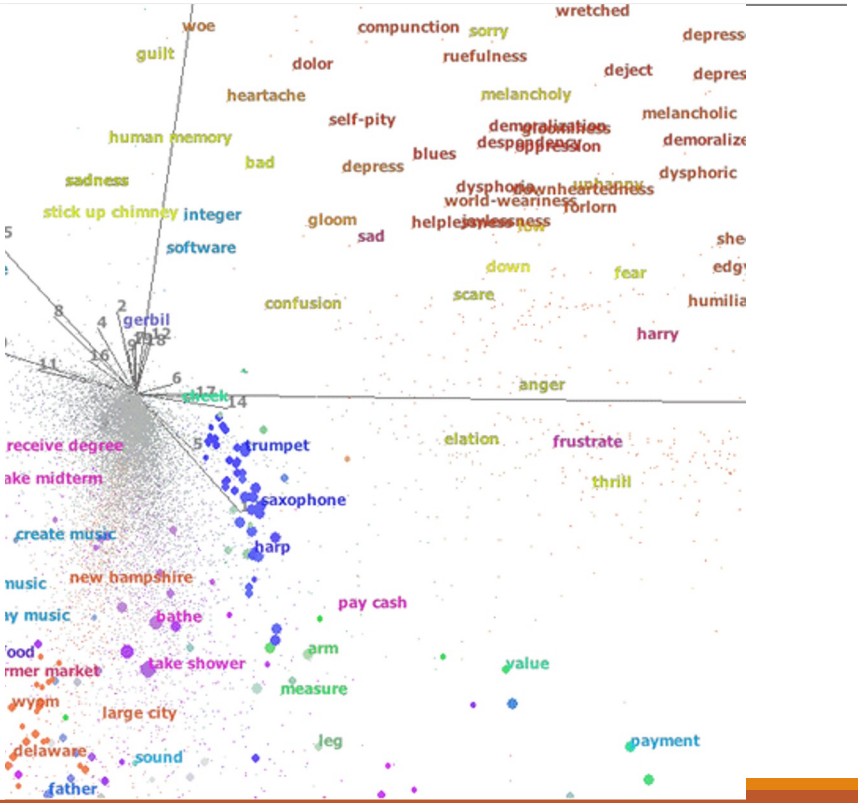
Example Pre-Trained Models

1. ALBERT (from Google Research and the Toyota Technological Institute at Chicago) released with the paper ALBERT: A Lite BERT for Self-supervised Learning of Language Representations, by Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, Radu Soricut.
2. BART (from Facebook) released with the paper BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension by Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov and Luke Zettlemoyer.
3. BERT (from Google) released with the paper BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding by Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova.
4. BERT For Sequence Generation (from Google) released with the paper Leveraging Pre-trained Checkpoints for Sequence Generation Tasks by Sascha Rothe, Shashi Narayan, Aliaksei Severyn.
5. CamemBERT (from Inria/Facebook/Sorbonne) released with the paper CamemBERT: a Tasty French Language Model by Louis Martin*, Benjamin Muller*, Pedro Javier Ortiz Suárez*, Yoann Dupont, Laurent Romary, Éric Villemonte de la Clergerie, Djamé Seddah and Benoît Sagot.
6. CTRL (from Salesforce) released with the paper CTRL: A Conditional Transformer Language Model for Controllable Generation by Nitish Shirish Keskar*, Bryan McCann*, Lav R. Varshney, Caiming Xiong and Richard Socher.
7. DeBERTa (from Microsoft Research) released with the paper DeBERTa: Decoding-enhanced BERT with Disentangled Attention by Pengcheng He, Xiaodong Liu, Jianfeng Gao, Weizhu Chen.
8. DialoGPT (from Microsoft Research) released with the paper DialoGPT: Large-Scale Generative Pre-training for Conversational Response Generation by Yizhe Zhang, Siqi Sun, Michel Galley, Yen-Chun Chen, Chris Brockett, Xiang Gao, Jianfeng Gao, Jingjing Liu, Bill Dolan.
9. DistilBERT (from HuggingFace), released together with the paper DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter by Victor Sanh, Lysandre Debut and Thomas Wolf. The same method has been applied to compress GPT2 into DistilGPT2, RoBERTa into DistilRoBERTa, Multilingual BERT into DistilmBERT and a German version of DistilBERT.
10. DPR (from Facebook) released with the paper Dense Passage Retrieval for Open-Domain Question Answering by Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih.
11. ELECTRA (from Google Research/Stanford University) released with the paper ELECTRA: Pre-training text encoders as discriminators rather than generators by Kevin Clark, Minh-Thang Luong, Quoc V. Le, Christopher D. Manning.
12. FlauBERT (from CNRS) released with the paper FlauBERT: Unsupervised Language Model Pre-training for French by Hang Le, Loïc Vial, Jibril Frej, Vincent Segonne, Maximin Coavoux, Benjamin Lecouteux, Alexandre Allauzen, Benoît Crabbé, Laurent Besacier, Didier Schwab.
13. Funnel Transformer (from CMU/Google Brain) released with the paper Funnel-Transformer: Filtering out Sequential Redundancy for Efficient Language Processing by Zihang Dai, Guokun Lai, Yiming Yang, Quoc V. Le.
14. GPT (from OpenAI) released with the paper Improving Language Understanding by Generative Pre-Training by Alec Radford, Karthik Narasimhan, Tim Salimans and Ilya Sutskever.
15. GPT-2 (from OpenAI) released with the paper Language Models are Unsupervised Multitask Learners by Alec Radford*, Jeffrey Wu*, Rewon Child, David Luan, Dario Amodei** and Ilya Sutskever**.
16. LayoutLM (from Microsoft Research Asia) released with the paper LayoutLM: Pre-training of Text and Layout for Document Image Understanding by Yiheng Xu, Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, Ming Zhou.
17. Longformer (from AllenAI) released with the paper Longformer: The Long-Document Transformer by Iz Beltagy, Matthew E. Peters, Arman Cohan.
18. LXMERT (from UNC Chapel Hill) released with the paper LXMERT: Learning Cross-Modality Encoder Representations from Transformers for Open-Domain Question Answering by Hao Tan and Mohit Bansal.
19. MarianMT Machine translation models trained using OPUS data by Jörg Tiedemann. The Marian Framework is being developed by the Microsoft Translator Team.
20. MBart (from Facebook) released with the paper Multilingual Denoising Pre-training for Neural Machine Translation by Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, Luke Zettlemoyer.
21. MMBT (from Facebook), released together with the paper a Supervised Multimodal Bitransformers for Classifying Images and Text by Douwe Kiela, Suvrat Bhooshan, Hamed Firooz, Davide Testuggine.
22. Pegasus (from Google) released with the paper PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization> by Jingqing Zhang, Yao Zhao, Mohammad Saleh and Peter J. Liu.
23. Reformer (from Google Research) released with the paper Reformer: The Efficient Transformer by Nikita Kitaev, Łukasz Kaiser, Anselm Levskaya.
24. RoBERTa (from Facebook), released together with the paper a Robustly Optimized BERT Pretraining Approach by Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov. ultilingual BERT into DistilmBERT and a German version of DistilBERT.
25. SqueezeBert released with the paper SqueezeBERT: What can computer vision teach NLP about efficient neural networks? by Forrest N. Iandola, Albert E. Shaw, Ravi Krishna, and Kurt W. Keutzer.
26. T5 (from Google AI) released with the paper Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer by Colin Raffel and Noam Shazeer and Adam Roberts and Katherine Lee and Sharan Narang and Michael Matena and Yanqi Zhou and Wei Li and Peter J. Liu.
27. Transformer-XL (from Google/CMU) released with the paper Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context by Zihang Dai*, Zhilin Yang*, Yiming Yang, Jaime Carbonell, Quoc V. Le, Ruslan Salakhutdinov.
28. XLM (from Facebook) released together with the paper Cross-lingual Language Model Pretraining by Guillaume Lample and Alexis Conneau.
29. XLM-RoBERTa (from Facebook AI), released together with the paper Unsupervised Cross-lingual Representation Learning at Scale by Alexis Conneau*, Kartikay Khandelwal*, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer and Veselin Stoyanov.
30. XLNet (from Google/CMU) released with the paper XLNet: Generalized Autoregressive Pretraining for Language Understanding by Zhilin Yang*, Zihang Dai*, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, Quoc V. Le.

https://github.com/huggingface/transformers

Evaluating Language Models





Credit:

- https://www.ruder.io/word-embeddings-1/

Evaluation – Language Model

- **Intrinsic evaluation**: measure the quality of a model independent of any application
- **Extrinsic evaluation**: situate model in an application and evaluate the whole application for improvement. Also called in-vivo evaluation

Perplexity

$$\text{Average NLL} = -\frac{1}{N}\sum_{i=1}^{N}\log P(w_i|w_1, w_2, ..., w_{i-1})$$

where N is the total number of words in the test dataset, and $P(w_i|w_1, w_2, ..., w_{i-}$ is the probability of word w_i given the previous words $w_1, w_2, ..., w_{i-1}$.

$$\text{Perplexity} = e^{\text{Average NLL}}$$

Value range: best: 1, **worst**: positive infinite;
practical upper bound: number of words in vocabulary

1. P(John) = 0.1
2. P(bought | John) = 0.4
3. P(apples | bought) = 0.3
4. P(from | apples) = 0.5
5. P(the | from) = 0.6
6. P(market | the) = 0.7

Now, let's compute the probability of the generated sequence:

$P(\text{"John bought apples from the market"}) = P(John) \times P(bought|John) \times P(apples|bought) \times P(from|apples) \times P(the|from) \times P(market|the)$

$P(\text{"John bought apples from the market"}) = 0.1 \times 0.4 \times 0.3 \times 0.5 \times 0.6 \times 0.7$

Hence, P("John bought apples from the market") = 0.00252

Average NLL = – log(0.00252) / 6. [N = 6 as the model generated six words]

Hence, Average NLL = 0.99725

Perplexity = Exp(Average NLL) = 2.71

Credit:

- https://blog.uptrain.ai/decoding-perplexity-and-its-significance-in-llms/
- https://medium.com/@priyankads/perplexity-of-language-models-41160427ed72

Perplexity Comments

1. A model with a vocabulary of 10,000 words and a perplexity of 2.71 is much better than a model with a vocabulary of 100 words and the same perplexity score of 2.71.
2. Lower perplexity results in higher consistency. As we know, LLMs are non-deterministic, i.e., the same inputs can result in two different outputs; a lower perplexity means that the model is more likely to produce the same output over multiple runs.
3. Perplexity is the inverse of the geometric mean of the probability of each word. Hence, the inverse of average probability (2.3077 in the previous case) can be considered a good proxy for quick calculations.
4. This calculation happens in the tokens space (compared to the words space), but the core principle remains the same.

Credit:

- https://blog.uptrain.ai/decoding-perplexity-and-its-significance-in-llms/

Perplexity

- Suppose
  - P('X') = 0.25
  - P('Y') = 0.5
  - P('Z') = 0.25
- Perplexity
  - ('XXX') = - exp (log (0.25 x 0.25 x 0.25) * (1/3)) = 3.94
  - Perplexity ('XYX') = - exp(log(0.25 x 0.5 x 0.25) * (1/3)) =
- Lower the number, the better is the model

Perplexity (Approx Calculation)

- Intrinsic evaluation
- **Definition**: perplexity of a language model on a test set is the inverse probability of the test set, normalized by the number of words

$$\begin{aligned} \text{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}} \end{aligned}$$

Of bi-grams

$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

Example: digits – 0 ..9, assuming equal probab. of 0.1

$$\begin{aligned} \text{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \left(\frac{1}{10}^{N}\right)^{-\frac{1}{N}} \\ &= \frac{1}{10}^{-1} \\ &= 10 \end{aligned}$$

From Jurafsky & Martin

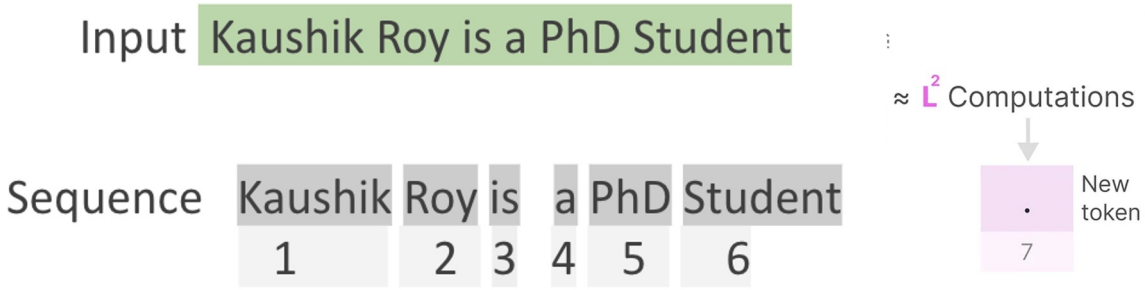
Evaluation – Extrinsic

- **Extrinsic evaluation**: situate model in an application and evaluate the whole application for improvement. Also called in-vivo evaluation

Credit:
- https://medium.com/@sthanikamsanthosh1994/understanding-bleu-and-rouge-score-for-nlp-evaluation-1ab334ecadcb

Mamba

Revisiting (Recurrent Neural Networks) RNNs and Family



Generating tokens for a sequence of length L needs roughly L^2 computations which can be costly if the sequence length increases.

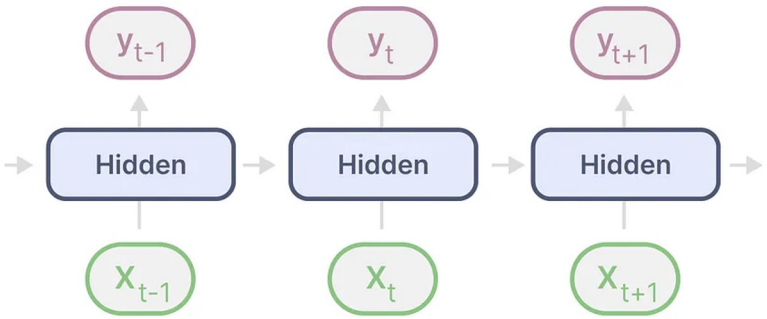
Credit:

- https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

Credit:

- https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

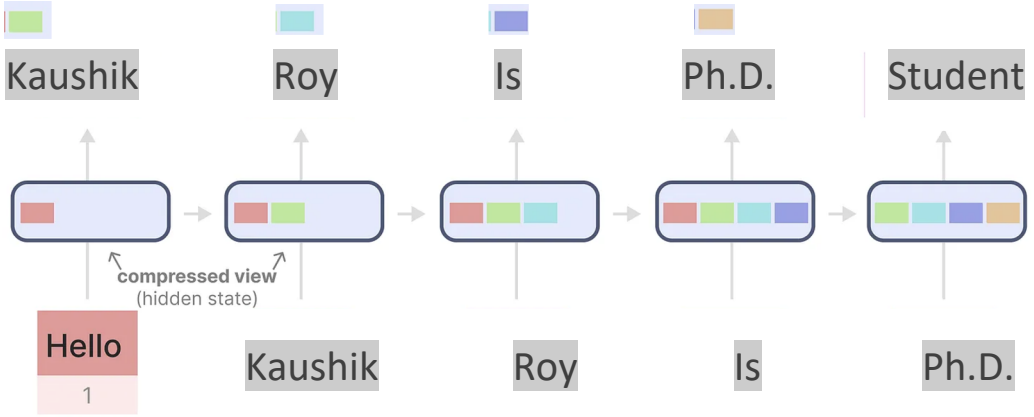
Long Context Issues



- Notice that there is forgetting when maintaining long context!
- Ad-hoc improvements – (Long term Short Term) LSTM, (Gated Recurrent Units) GRUs, etc.

In other words, RNNs can do inference fast as it scales linearly with the sequence length! In theory, it can even have an *infinite context length*.

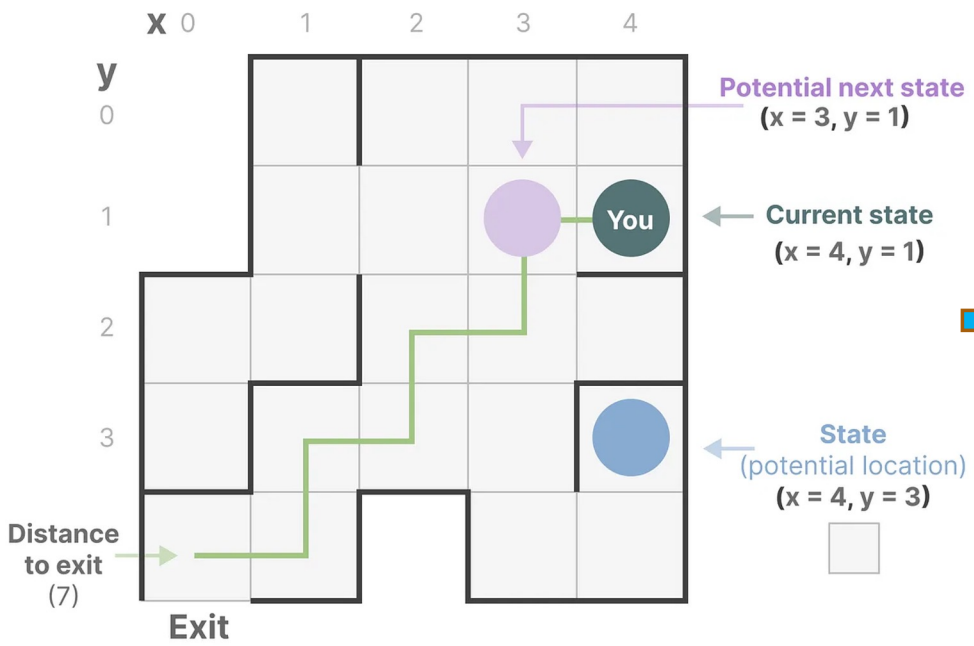
To illustrate, let's apply the RNN to the input text we have used before.

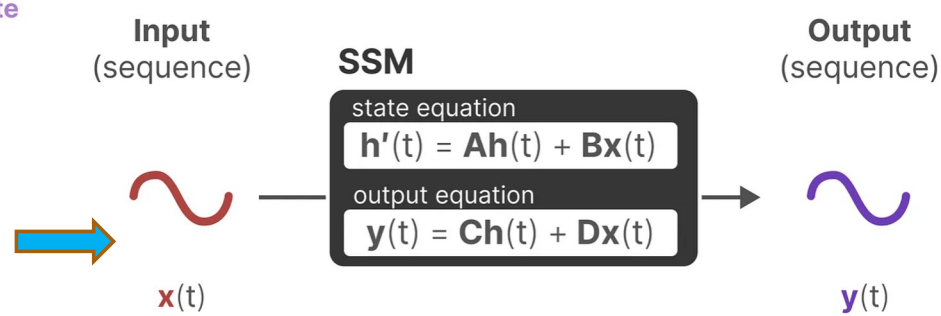


Each hidden state is the aggregation of all previous hidden states and is typically a compressed view.

Credit:
- https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

State Space Models



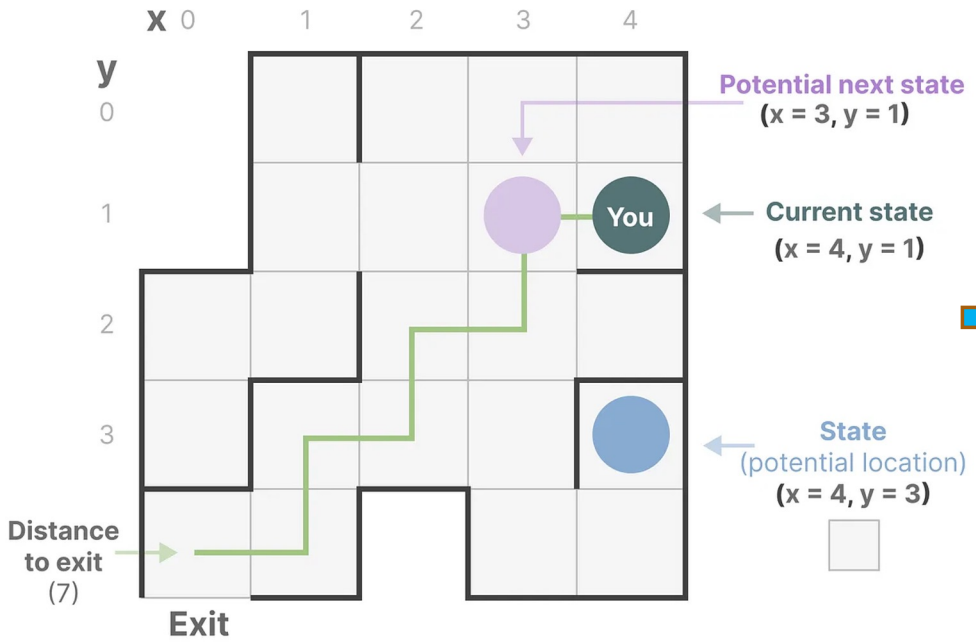


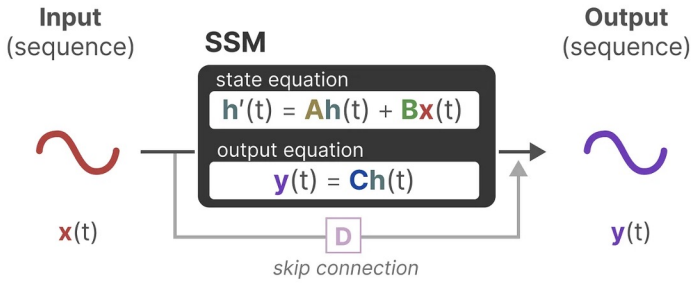
- The theoretical generalization of RNNs, where a system state is maintained to track essential elements, as required for prediction.

Credit:

- https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

RNN Family - Mamba

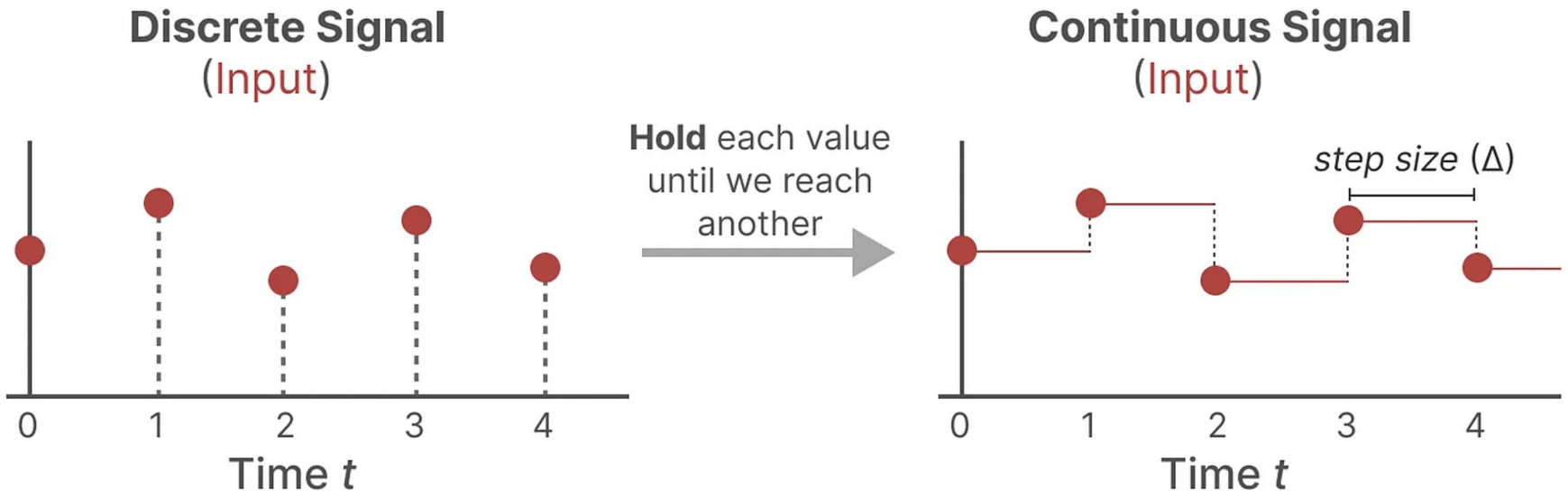




- Mamba is a special type of SSM that makes specific and informed design choices towards (i) Long context tracking (ii) Expressive context representations (iii) Efficient training/inference

Credit:
- https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

(i) RNN Family - Mamba (Discretization)



- SSM theory is defined for continuously evolving systems, and in the finite data regime, we require discretization that achieves a kind of "prolong holding of signal", i.e., long context state tracking

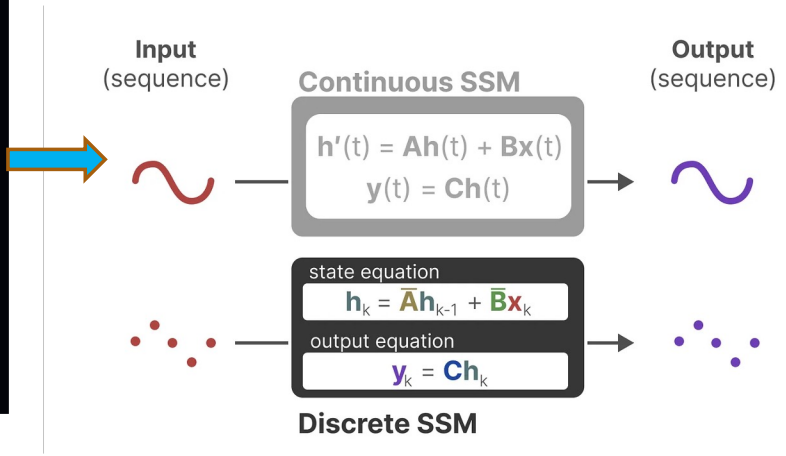
Credit:
- https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

(i) RNN Family - Discrete Mamba

- The exponentiation-based discretization utilized by Mamba is particularly simple and intuitive to understand, as exponentiation simply amplifies the signal until the next discrete time point

Discretized matrix **A** $\bar{\mathbf{A}} = \exp(\Delta\mathbf{A})$

```
    self.A = nn.Parameter(torch.randn(d_model, state_size))
    nn.init.xavier_uniform_(self.A)

def forward(self, x):
    batch_size, seq_len, _ = x.shape  # Dynamically infer the sequence length from input

    # Initialize dynamic buffers
    B = torch.zeros(batch_size, seq_len, self.state_size, device=x.device)
    C = torch.zeros(batch_size, seq_len, self.state_size, device=x.device)
    delta = torch.zeros(batch_size, seq_len, self.d_model, device=x.device)
    dA = torch.zeros(batch_size, seq_len, self.d_model, self.state_size, device=x.device)
    h = torch.zeros(batch_size, seq_len, self.d_model, self.state_size, device=x.device)
```



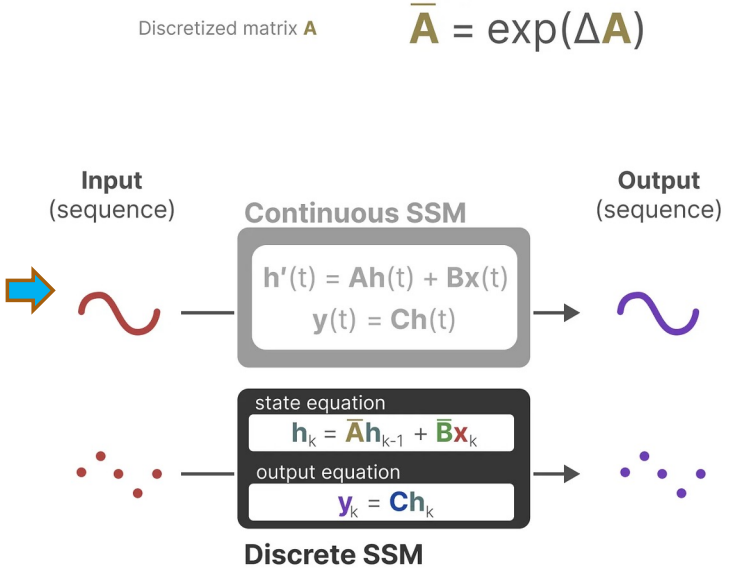
Credit:

- https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

(i) Mamba (S6: Structured State Space for Sequence Modeling with Selective Scanning)

- Thus Mamba in it's entirety is a discretized state space model
- as It is called an S6 model because it is first, an S4 model which is an abbreviation for "The Structured State Space for Sequence Modeling"
- The S6 comes from it's selective scanning procedure, where the discretization matrix effective **selects** how much of the context to **hold**

```python
# Step 3: Define the Mamba Model
class S6(nn.Module):
    def __init__(self, d_model, state_size):
        super(S6, self).__init__()
        self.fc1 = nn.Linear(d_model, d_model)
        self.fc2 = nn.Linear(d_model, state_size)
        self.fc3 = nn.Linear(d_model, state_size)
        self.d_model = d_model
        self.state_size = state_size

        self.A = nn.Parameter(torch.randn(d_model, state_size))
        nn.init.xavier_uniform_(self.A)

    def forward(self, x):
        batch_size, seq_len, _ = x.shape  # Dynamically infer the sequence length from input

        # Initialize dynamic buffers
        B = torch.zeros(batch_size, seq_len, self.state_size, device=x.device)
        C = torch.zeros(batch_size, seq_len, self.state_size, device=x.device)
        delta = torch.zeros(batch_size, seq_len, self.d_model, device=x.device)
        dA = torch.zeros(batch_size, seq_len, self.d_model, self.state_size, device=x.device)
        h = torch.zeros(batch_size, seq_len, self.d_model, self.state_size, device=x.device)

        # Apply linear transformations
        B = self.fc2(x)
        C = self.fc3(x)
        delta = F.softplus(self.fc1(x))

        # Discretization operation
        dA = torch.exp(torch.einsum("bld,dn->bldn", delta, self.A))

        # Compute h and y (output)
        h = torch.einsum('bldn,bldn->bldn', dA, h) + torch.unsqueeze(x, dim=-1)
        y = torch.einsum('bln,bldn->bld', C, h)

        return y
```



Credit:
- https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

(ii) Non-Linear Mamba and HIPPO

- So far we have seen Mamba handle linear matrices
- But of course real world, and especially language data must have non-linear dependencies
- Mamba achieves this using **HIPPO**: High-order Polynomial Projection Operators, specifically by leveraging **Legendre Polynomials?!**

```python
# Legendre Polynomial Function
def legendre_polynomials(x, order=5):
    """
    Compute the first few Legendre polynomials (up to a given order) for each input in x.
    Args:
    x (torch.Tensor): Input tensor of shape (batch_size, seq_len, d_model).
    order (int): The maximum order of the Legendre polynomial to compute.

    Returns:
    torch.Tensor: Transformed input with additional Legendre polynomial features.
    """
    batch_size, seq_len, d_model = x.shape

    # Initialize a list to store polynomials P_0(x), P_1(x), ..., P_order(x)
    polynomials = []

    # P_0(x) = 1 (constant)
    P0 = torch.ones_like(x)
    polynomials.append(P0)

    # P_1(x) = x
    P1 = x
    polynomials.append(P1)

    # Recursively compute P_n(x) for n = 2, 3, ..., order
    for n in range(2, order + 1):
        Pn = ((2 * n - 1) * x * polynomials[n - 1] - (n - 1) * polynomials[n - 2]) / n
        polynomials.append(Pn)

    # Stack all polynomials together along the last dimension (d_model)
    # This will create additional features for each input based on Legendre polynomials
    return torch.cat(polynomials, dim=-1)
```

Credit:

- https://en.wikipedia.org/wiki/Polynomial_regression#:~:text=In%20statistics%2C%20polynomial%20regression%20is,nth%20degree%20polynomial%20in%20x.

(ii) Recall Polynomial Regression

Matrix form and calculation of estimates [edit]

The polynomial regression model

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_m x_i^m + \varepsilon_i \ (i = 1, 2, \ldots, n)$$

can be expressed in matrix form in terms of a design matrix $\mathbf{X}$, a response vector $\vec{y}$, a parameter vector $\vec{\beta}$, and a vector $\vec{\varepsilon}$ of random errors. The *i*-th row of $\mathbf{X}$ and $\vec{y}$ will contain the *x* and *y* value for the *i*-th data sample. Then the model can be written as a system of linear equations:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^m \\ 1 & x_2 & x_2^2 & \cdots & x_2^m \\ 1 & x_3 & x_3^2 & \cdots & x_3^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^m \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_m \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \vdots \\ \varepsilon_n \end{bmatrix},$$

which when using pure matrix notation is written as

$$\vec{y} = \mathbf{X}\vec{\beta} + \vec{\varepsilon}.$$

The vector of estimated polynomial regression coefficients (using ordinary least squares estimation) is

$$\hat{\vec{\beta}} = (\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}\,\mathbf{X}^\mathsf{T}\vec{y},$$

- Polynomials provide a way to expand a linear model to capture non-linear dependencies
- Polynomials are also **universal approximators** (roughly), i.e., they can approximate any non-linear function upto an arbitrary degree of precision (see Taylor series)
- But successive polynomials are clearly correlated with previous ones, **(e.g., x^2 correlated with x), and in learning, we like uncorrelated features**

Credit:
- https://en.wikipedia.org/wiki/Gram%E2%80%93Schmidt_process

(ii) Recall Gram Schmidt Orthogonalization

Gram–Schmidt process

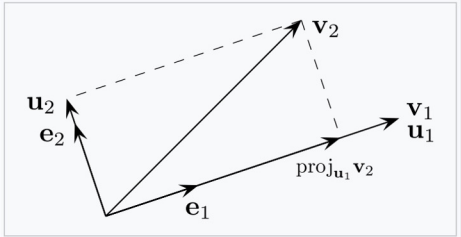
35 languages

Article Talk | Read Edit View history Tools

From Wikipedia, the free encyclopedia

In mathematics, particularly linear algebra and numerical analysis, the **Gram–Schmidt process** or Gram-Schmidt algorithm is a way of finding a set of two or more vectors that are perpendicular to each other.

By technical definition, it is a method of constructing an orthonormal basis from a set of vectors in an inner product space, most commonly the Euclidean space $\mathbb{R}^n$ equipped with the standard inner product. The Gram–Schmidt process takes a finite, linearly independent set of vectors $S = \{\mathbf{v}_1, \dots, \mathbf{v}_k\}$ for $k \leq n$ and generates an orthogonal set $S' = \{\mathbf{u}_1, \dots, \mathbf{u}_k\}$ that spans the same k-dimensional subspace of $\mathbb{R}^n$ as S.



The first two steps of the Gram–Schmidt process

The method is named after Jørgen Pedersen Gram and Erhard Schmidt, but Pierre-Simon Laplace had been familiar with it before Gram and Schmidt.[1] In the theory of Lie group decompositions, it is generalized by the Iwasawa decomposition.

The application of the Gram–Schmidt process to the column vectors of a full column rank matrix yields the QR decomposition (it is decomposed into an orthogonal and a triangular matrix).

- Gram schmidt orthogonalization is a procedure by which a set of correlated vectors can be converted into an orthogonal set of vectors
- When a polynomial basis is **orthogonalized, i.e., made uncorrelated** using the Gram schmidt orthogonalization process, we **get legendre polynomials**

Credit:
- https://en.wikipedia.org/wiki/Legendre_polynomials

(ii) Legendre Polynomials

The first few Legendre polynomials are:

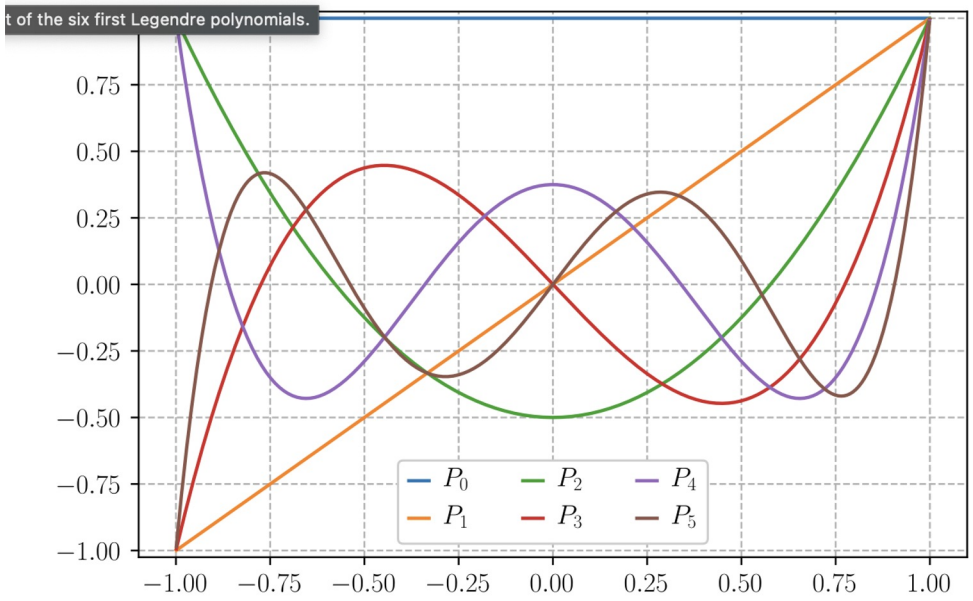
n	$P_n(x)$
0	1
1	x
2	$\frac{1}{2}\left(3x^2-1\right)$
3	$\frac{1}{2}\left(5x^3-3x\right)$
4	$\frac{1}{8}\left(35x^4-30x^2+3\right)$
5	$\frac{1}{8}\left(63x^5-70x^3+15x\right)$
6	$\frac{1}{16}\left(231x^6-315x^4+105x^2-5\right)$
7	$\frac{1}{16}\left(429x^7-693x^5+315x^3-35x\right)$
8	$\frac{1}{128}\left(6435x^8-12012x^6+6930x^4-1260x^2+35\right)$
9	$\frac{1}{128}\left(12155x^9-25740x^7+18018x^5-4620x^3+315x\right)$
10	$\frac{1}{256}\left(46189x^{10}-109395x^8+90090x^6-30030x^4+3465x^2-63\right)$

- Gram schmidt orthogonalization is a procedure by which a set of correlated vectors can be converted into an orthogonal set of vectors
- When a polynomial basis is **orthogonalized, i.e., made uncorrelated** using the Gram schmidt orthogonalization process, we **get legendre polynomials**

Credit:
- https://en.wikipedia.org/wiki/Legendre_polynomials

(ii) Legendre Polynomials

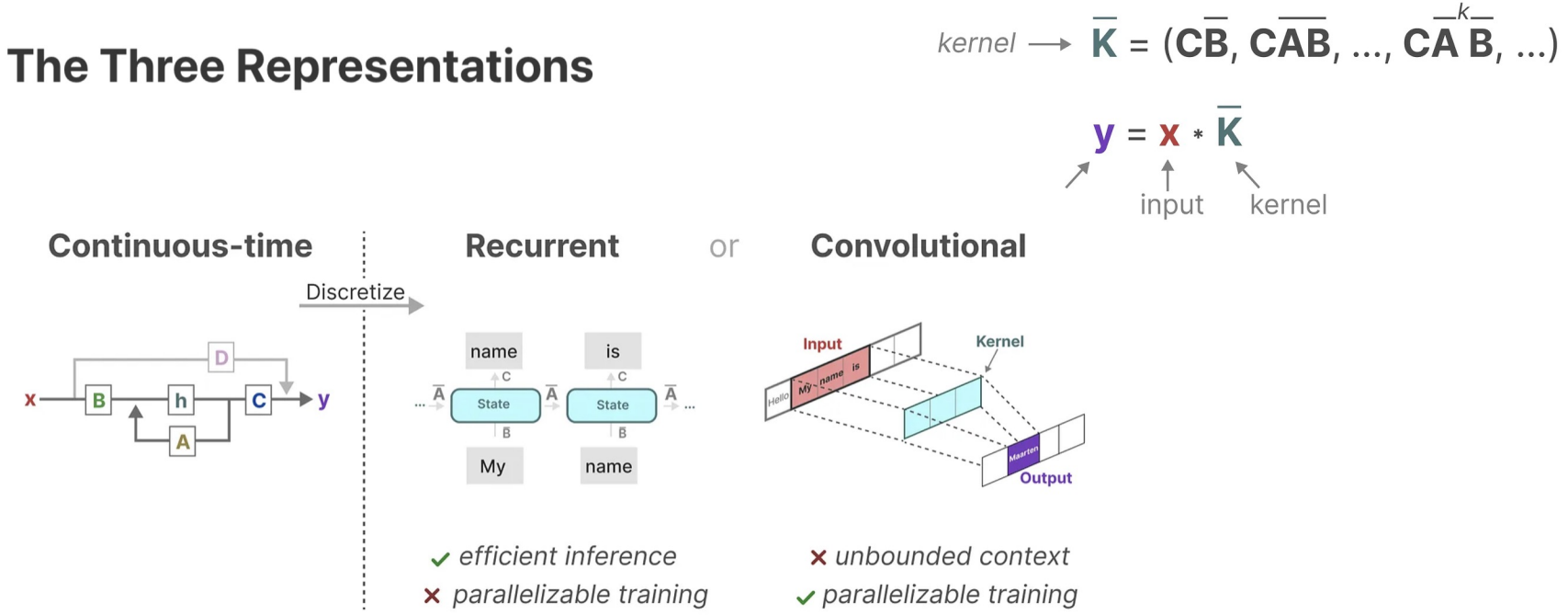
The graphs of these polynomials (up to $n = 5$) are shown below:



- The graph helps visualize intuitively how any non linear function can be captured by some combination of legendre polynomials
- As we can see, even upto just degree five they already create a very expressive basis

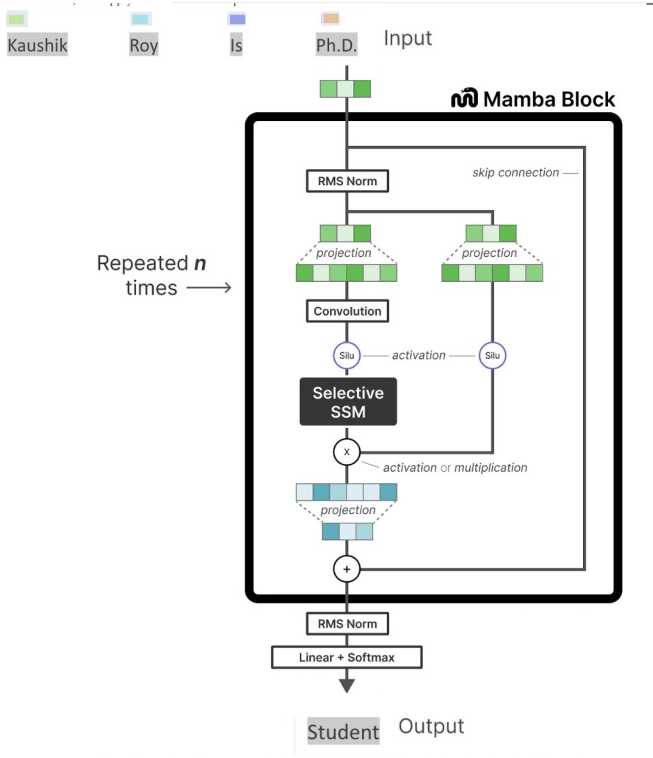
Credit:
- https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

(iii) Representation of the Operations for Efficiency



[Image Source]

Putting it All together - Mamba



Simplified for Pedagogical Purpose

```python
class MambaBlock(nn.Module):
    def __init__(self, d_model, state_size, legendre_order=5):
        super(MambaBlock, self).__init__()
        self.legendre_order = legendre_order  # Order of Legendre polynomials
        self.inp_proj = nn.Linear((legendre_order + 1) * d_model, 2 * d_model)
        self.out_proj = nn.Linear(2 * d_model, d_model)
        self.s6 = S6(2 * d_model, state_size)
        self.norm = nn.LayerNorm(2 * d_model)  # LayerNorm matches the d_model a

    def forward(self, x):
        # Apply Legendre polynomials to the input
        x_legendre = legendre_polynomials(x, order=self.legendre_order)  # Non-

        # Project input to 2*d_model after Legendre expansion
        x_proj = self.inp_proj(x_legendre)
        x_proj = self.norm(x_proj)  # Apply normalization
        x_ssm = self.s6(x_proj)  # Pass through S6 module
        x_out = self.out_proj(x_ssm)  # Project back to d_model dimension
        return x_out
```

Live Coding

Mamba-based CV processing

Do in parallel live coding with your resume.

```python
# Mamba Block with Legendre Polynomial Transformation
class MambaBlock(nn.Module):
    def __init__(self, d_model, state_size, legendre_order=5):
        super(MambaBlock, self).__init__()
        self.legendre_order = legendre_order  # Order of Legendre polynomials
        self.inp_proj = nn.Linear((legendre_order + 1) * d_model, 2 * d_model)  # Adjust input size after Legendre expansion
        self.out_proj = nn.Linear(2 * d_model, d_model)
        self.s6 = S6(2 * d_model, state_size)
        self.norm = nn.LayerNorm(2 * d_model)  # LayerNorm matches the d_model after projection

    def forward(self, x):
        # Apply Legendre polynomials to the input
        x_legendre = legendre_polynomials(x, order=self.legendre_order)  # Non-linear transform

        # Project input to 2*d_model after Legendre expansion
        x_proj = self.inp_proj(x_legendre)
        x_proj = self.norm(x_proj)  # Apply normalization
        x_ssm = self.s6(x_proj)  # Pass through S6 module
        x_out = self.out_proj(x_ssm)  # Project back to d_model dimension
        return x_out

# Full Mamba Model
class Mamba(nn.Module):
    def __init__(self, d_model, state_size, vocab_size, legendre_order=5):
        super(Mamba, self).__init__()
        self.embedding = nn.Embedding(vocab_size, d_model)  # Embedding layer
        self.mamba_block1 = MambaBlock(d_model, state_size, legendre_order)
        self.mamba_block2 = MambaBlock(d_model, state_size, legendre_order)
        self.mamba_block3 = MambaBlock(d_model, state_size, legendre_order)
        self.fc_out = nn.Linear(d_model, vocab_size)  # Final output layer for MLM

    def forward(self, x):
        x = self.embedding(x)  # Embed the input tokens to shape (batch_size, seq_len, d_model)
        x = self.mamba_block1(x)
        x = self.mamba_block2(x)
        x = self.mamba_block3(x)
        return self.fc_out(x)  # Return logits for each token

# Step 4: Training the Mamba Model
# Hyperparameters
d_model = 128  # Dimensionality of the model
state_size = 256  # Size of the hidden state
batch_size = 32
num_epochs = 5
vocab_size = tokenizer.vocab_size  # Vocabulary size from the tokenizer
```

Concluding Segment

Image Source

Concluding Comments

- We looked at the Mamba model
  - learned about the design choices that went into the Mamba model that make it superior for long context sequential modeling compared to transformer models
  - Did CV processing using it
  - Discussed tradeoffs

Image Source

About Next Lecture – Language Modeling

- Playground
- Comparisons on a suite of tasks

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 13

Lecture 13 Outline

- Comparing architectures
- Comparing finetuning
- Applying a new domain/ task – elections?

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