ECE 587 – Hardware/Software Co-Design
Lecture 03
Neural Networks and Language Models

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

January 21, 2026

Outline

Reading Assignment

- This lecture: Neural Networks and Language Models
  - Attention Is All You Need, Vaswani et al.
    `https://arxiv.org/abs/1706.03762`
- Next lecture (Fri. 1/23): General Matrix Multiplication (GEMM)

Outline

Neural Networks

Natural Language Processing

Large Language Models

(Artificial) Neural Networks

- A model of computation inspired by biological neurons.
  - Still, we don't know how biological neural networks work.
  - Dated back to 1940's but with a few AI winters.
- Substantial progress since the last decade.
  - Availability of large amount of data.
  - Availability of GPUs for general-purpose computing.
- Dominate current computational resource consumption.

Nodes and Layers

- Most neural networks are dataflow graphs consisting of many nodes.
  - Each node computes its output as a simple function of its inputs, e.g. weighted summation, activation, and softmax.
  - Feedforward/uni-directional/without cycles or loops.
- Layers: tremendous number of nodes are organized into layers to facilitate reasoning and implementation.
  - Layers are ordered so that outputs from previous layers are used as inputs to next layers.
- Together, any vector-valued function can be approximated.

A Typical Layer

$$\boldsymbol{h} = g(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b})$$

- $\boldsymbol{x}$: input vector of this layer
- $\boldsymbol{h}$: output vector of this layer
- $\boldsymbol{W}$, $\boldsymbol{b}$: weight matrix and bias vector
  - Could be fixed paramaters or inputs to the layer.
- g: activation function
  - A fixed nonlinear function applied element-wise to a vector.
- Learning by approximating known input/output relations.
  - Find a good number of layers and then $\boldsymbol{W}$ and $\boldsymbol{b}$ for each layer.
  - Challenge: generalization – the learned model should also perform nicely on unseen inputs.
  - Deep learning: models with more layers tend to generalize better as they require less dimension in $\boldsymbol{W}$ and $\boldsymbol{b}$.

Outline

Natural Language Processing (NLP)

- Use natural language as interface between computers and human beings.
- Applications
  - Voice command
  - Machine translation
  - Text summarization
  - Image and video captioning
  - Question answering
  - Story, image, and video generation
  - Many more to come
- Turing test: what is intelligence?

Tokenization

- Convert texts in natural language into tokens that may have meanings to facilitate further processing.
- Character-based tokenization
  - Simple and effective to digitalize texts, e.g. ASCII and Unicode
  - Need extra effort when characters don't carry meanings by themselves, e.g. English.
- Word-based tokenization
  - Encode individual words and punctuations using a vocabulary.
  - How to handle out-of-vocabulary and misspelled words?
  - A very difficult task by itself for languages without word separators, e.g. Chinese.
- Subword tokenization
  - Learn common patterns from character sequences as subword that usually carry meanings and fall back to characters.
  - Handle rare, new, or misspelled words by breaking them into known subword (and characters).

Embedding

- If there are M different tokens, a token can be represented as a $M \times 1$ vector via one-hot encoding.
  - One element is 1 while the rest are 0.
- However, one-hot encoding doesn't capture any meanings.
- Embedding: represent tokens as vectors (usually shorter) to capture semantic relationships and similarities.
  - Tokens are then points in the embedding space.
  - Tokens with similar meanings like 'I' and 'me' are mapped to points that are close in a subspace.
- Assume each vector is of the size $d \times 1$, embedding is learnt during the training process as a $d \times M$ matrix.
- For now on, we will not distinguish between the token and its vector after embedding.

Encoder-Decoder Models

- Most NLP tasks can be formulated as to generate an output sequence of tokens from an input sequence of tokens.
- Since both input and output sequences can have arbitrary lengths, two models are introduced for the NLP task.
  - Encoder $C' = E(C, x)$: process the input sequence of arbitrary length by consuming one token x at a time and transforming the context vector C of fixed size into the next one C'.
  - Decoder $(x, C') = D(C)$: generate the output sequence one token at a time by computing a token x from the context vector C and transforming C into the next one C'.

Autoregression

- Decoder needs to be statistical: $(Pr, C') = D(C)$
  - Have to learn from natural languages, which are ambiguous and have a lot of variability.
  - Instead of the actual token x, decoder computes Pr as the vector of the probability of each token to be the output.
  - A sampling process then samples Pr to obtain x.
  - But then C' has no knowledge of x – how could the decoder ensure the whole output sequence to be coherent?
- Autoregression: $(Pr_{N+1}, C') = D(C, x_1, x_2, \ldots, x_N)$
  - The decoder takes a window of N previously generated output tokens as additional inputs to make better predictions.
- Challenges
  - How can we design encoders and decoders as neural networks?
  - How to define loss functions to train models?
  - How to obtain data for training?

Outline

Decoder-only Models

$$Pr_{N+1} = D(x_1, x_2, \ldots, x_N)$$

- When the window size N is large enough, the whole input sequence can be included as if they are generated first.
- Introduce special tokens to indicate end of input.
  - Prompt the decoder to generate actual output tokens.
- No need to use encoder and context any more.

Considerations for Training

$$(Pr_2, Pr_3, \ldots, Pr_{N+1}) = D(x_1, x_2, \ldots, x_N)$$

- The decoder model actually predict probability Pr_2, Pr_3, $\ldots$ for known tokens x_2,x_3,$\ldots$ in addition to the next token.
  - A model architecture matching lengths of input and output.
- A loss function can be defined between actual tokens $(x_2, \ldots, x_{N+1})$ and predictions $(Pr_2, \ldots, Pr_{N+1})$.
  - Masking: ensure that probabilites are only computed from previous tokens, like how we read a sentence word by word.
  - For example, Pr_2 should only depend on x_1, and Pr_N should only depend on $(x_1, \ldots, x_{N-1})$ but not x_N.
- Learn D from vast amount of text via unsupervised learning, without the need to label data by human beings.
- How to build neural networks for D?

Attention: Query

- Attention: a neural network layer that allows to extract data from a sequence of arbitrary length.
- Query q: a vector representing a pattern of interests.
  - Assume q to have the same size as x_i, i.e. both are $d \times 1$ vectors. Then the inner product $q^T x_i$ is a scalar representing how similar q and x_i are.
- Use inner products to score tokens: $(q^T x_1, q^T x_2, \ldots, q^T x_N)$
  - Token with higher score will contribute more to extracted data.
  - Use softmax to calculate weights for each token and extracted data as a weighted summation of all tokens.
- Attention with query: softmax$(q^T \boldsymbol{X}^T)\boldsymbol{X}$
  - $\boldsymbol{X}$ is a matrix with N rows $x_1^T, \ldots, x_N^T$, and d columns.
  - $q^T \boldsymbol{X}^T$ gives a $1 \times N$ row vector and so does softmax.
  - softmax$(q^T \boldsymbol{X}^T)\boldsymbol{X}$ extracts a $1 \times d$ row vector from the input sequence of arbitrary length with the given query q.

Attention: Keys and Values

- What if we would like to have more flexibility so both query and output could have a different size?
- Keys: $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^K$ where $\boldsymbol{W}^K$ are the weights
  - Query with the key instead of the tokens.
  - Assume $\boldsymbol{W}^K$ is a $d \times d_k$ matrix.
  - $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^K$ is a $N \times d_k$ matrix.
- The scores and weights become softmax$(q^T \boldsymbol{K}^T)$
  - q will have a matching size of $d_k \times 1$.
  - $q^T \boldsymbol{K}^T$ gives a $1 \times N$ row vector and so does softmax.
- Values: $\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^V$ where $\boldsymbol{W}^V$ are the weights
  - Extract data as weighted summation of value instead of tokens.
  - Assume $\boldsymbol{W}^V$ is a $d \times d_v$ matrix.
  - $\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^V$ is a $N \times d_v$ matrix.
- Attention: softmax$(q^T \boldsymbol{K}^T)\boldsymbol{V}$, a $1 \times d_v$ row vector

Self-Attention

- Is it possible to use multiple queries and how to obtain them?
  - Yes and we can obtain them from the input sequence itself.
- Queries: $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^Q$ where $\boldsymbol{W}^Q$ are the weights
  - Query the input sequence with itself.
  - $\boldsymbol{W}^Q$ is a $d \times d_k$ matrix and $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^Q$ is a $N \times d_k$ matrix.
  - Each row of $\boldsymbol{Q}$ is a query and there are N queries.
- $\boldsymbol{Q}\boldsymbol{K}^T$ computes scores between the N queries and N keys.
  - Each row contains scores for a single query with all keys.
  - We can apply softmax row by row to obtain weights.
- Self-Attention$(\boldsymbol{X}) = \text{softmax}(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}})\boldsymbol{V}$, a $N \times d_v$ matrix.
  - $\boldsymbol{Q}\boldsymbol{K}^T$ is scaled by $\sqrt{d_k}$ as its elements get larger when each query and key becomes longer.
  - Learn all the weights $\boldsymbol{W}^Q$,$\boldsymbol{W}^K$,$\boldsymbol{W}^V$ during training.
- We ignore the details of masking and positional encoding here.

Multi-Head Attention

$$\text{head}_i = \text{Self-Attention}_i(\boldsymbol{X})$$
$$\text{MultiHead}(\boldsymbol{X}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\boldsymbol{W}^O$$

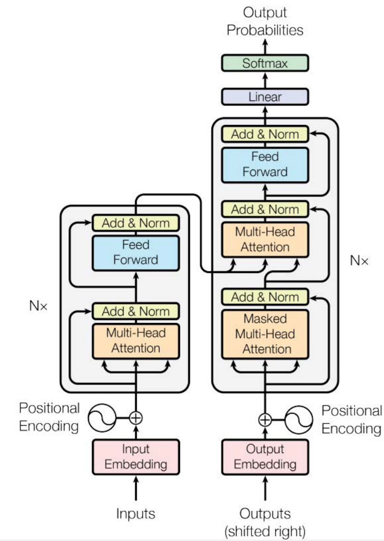
- Learn multiple (h) sets of $(\boldsymbol{W}^Q, \boldsymbol{W}^K, \boldsymbol{W}^V)$
- Each generate a $N \times d_v$ matrix as output using self-attention.
- Concatenate the outputs into a $N \times hd_v$ matrix.
- Learn the matrix $\boldsymbol{W}^O$ of size $hd_v \times d$ as the output weights so the overall output has the same size $N \times d$ as the input.
- Multi-head attention provide a lot of opportunities for parallelization.
- When input and output are of the same size, we can stack many of the same layers for a deeper and larger model.

Position-wise Feed-Forward Networks (FFN)

$$\mathsf{MultiHead}(\boldsymbol{X}) = \mathsf{Concat}(\mathsf{head}_1, \ldots, \mathsf{head}_h)\boldsymbol{W}^O$$

- The output of MultiHead($\boldsymbol{X}$) as a $N \times d$ matrix can be viewed as a sequence of N row vectors.
- Introduce additional non-linearity and capacity by transforming individual output vectors identically.
- Make use of multiple fully connected (MLP) layers, e.g.
  $$\mathsf{FFN}(\boldsymbol{y}) = ReLU(\boldsymbol{y}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2$$
  - $\boldsymbol{y}$ is a row vector from the output of multi-head attention.
  - Learn weights and bias's $\boldsymbol{W}_1$, $\boldsymbol{b}_1$, $\boldsymbol{W}_2$, $\boldsymbol{b}_2$ during training.
  - The same set of $\boldsymbol{W}_1$, $\boldsymbol{b}_1$, $\boldsymbol{W}_2$, $\boldsymbol{b}_2$ are used for all rows.

Transformer



(Figure 1, Attention Is All You Need,

Vaswani et al., 2017)

- The original transformer model contains both encoder and decoder.
- Stack of FFN and attention layers.
  - With layer normalizations and residual connections.
- Probabilites are generated at each output position identically.
  - First, a linear layer transform the output vector of size d into a vector of size M.
  - Then, apply softmax to obtain the probabilities at this position.
- Remove encoder related parts to obtain a decoder-only transformer.

Summary

- Matrix multiplications are essential to neural networks.
- How can we implement General Matrix Multiplication (GEMM) efficiently?
  - As a unified HW/SW design problem.
  - Toward a HW/SW co-design methodology for any computations.