GPT in Probabilistic Programming

Separating Inference from Model

Daniel Lee
daniel@bayesianops.com

August 10, 2023

Preamble

- Lots of credit due.
  (References on last slide.)
- All mistakes are mine.
- Please, do not try to build production GPTs in PPLs.

The Problem

- Given some text, we want new text that is a continuation of the text.

- Given some text, we want new text that is a continuation of the text.

Example

Input:
"O Romeo, Romeo, wherefore art"

Output:
" thou Romeo?"

- Given some text, we want new text that is a continuation of the text.

plausible

Example

Input:
"O Romeo, Romeo, wherefore art"

Output:
" thou Romeo?"

Generative Pre-trained Transformer

Generative Pre-trained Transformer (GPT)

- **Generative**
  The model is capable of generating text

- **Pre-trained**
  The model is trained on a (large) dataset before use

- **Transformer**
  The model uses parallel multi-head attention mechanism

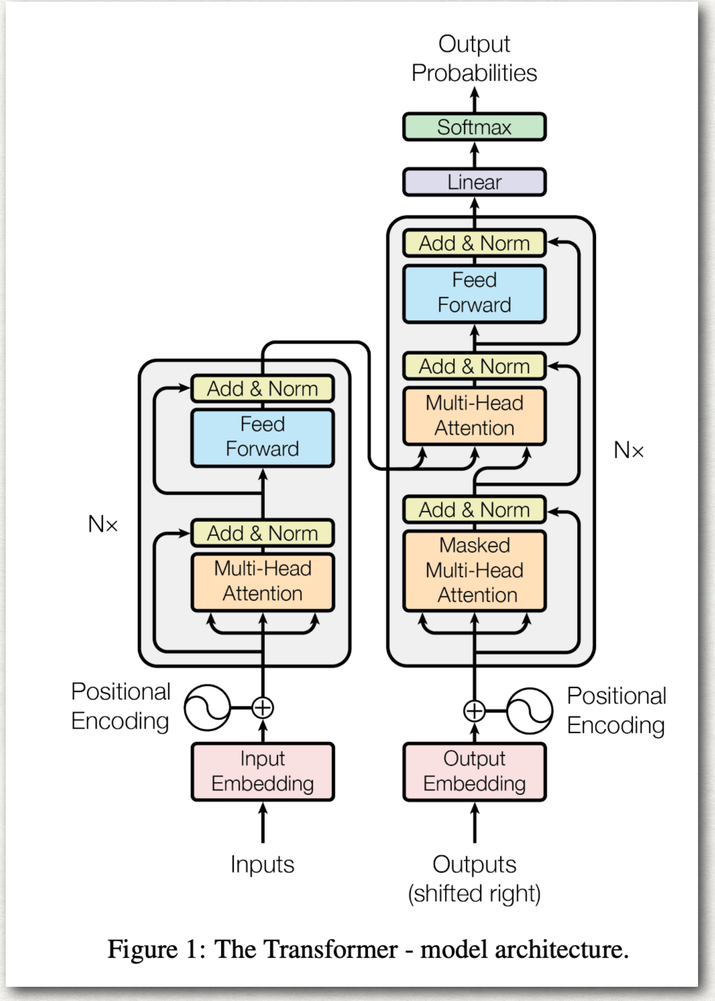


Figure 1: The Transformer - model architecture.

arXiv:1706.03762 [cs.CL]
"Attention is All You Need"

Building a GPT in Stan

1. Bigram model
2. Embedding size
3. Positional encoding
4. Self-attention
5. Multi-headed self-attention
6. (We'll skip)
   Feed Forward NN, Skip Connection,
   Larger Feed Forward NN, Blocks, Dropout

https://github.com/bayesianops/gpt-tutorial

0. Data

- Subset of Shakespeare (see: https://github.com/karpathy/char-rnn)
- Examples:

```
First Citizen:
Before we proceed any further, hear me speak.

All:
Speak, speak.

First Citizen:
You are all resolved rather to die than to famish?

All:
Resolved. resolved.

First Citizen:
First, you know Caius Marcius is chief enemy to the people.

All:
We know't, we know't.

First Citizen:
Let us kill him, and we'll have corn at our own price.
```

```
Dropping upon thy head.

SEBASTIAN:
What, art thou waking?

ANTONIO:
Do you not hear me speak?

SEBASTIAN:
I do; and surely
It is a sleepy language and thou speak'st
Out of thy sleep. What is it thou didst say?
This is a strange repose, to be asleep
With eyes wide open; standing, speaking, moving,
And yet so fast asleep.

ANTONIO:
Noble Sebastian,
Thou let'st thy fortune sleep--die, rather; wink'st
Whiles thou art waking.
```

0. Data

- Convert sequence of text to sequence of numbers: 1-65.
- Example (first 10 characters):

```
"First Citi" -> [19, 48, 57, 58, 59, 2, 16, 48, 59, 48]
```

- Data structured as: batch_size x block_size

xb:

```
[[26, 40, 50, 44, 2, 64, 54, 60],
 [53, 11, 1, 15, 60, 59, 2, 64],
 [54, 58, 59, 2, 41, 44, 40, 60],
 [62, 44, 51, 42, 54, 52, 44, 2],
 [58, 2, 50, 44, 44, 55, 2, 48],
 [57, 2, 22, 2, 52, 40, 64, 2],
 [7, 1, 20, 48, 61, 44, 58, 2],
 [40, 57, 44, 2, 59, 47, 44, 64],
 [45, 2, 52, 44, 7, 1, 14, 53],
 [48, 59, 1, 52, 40, 50, 44, 58]]
```

yb:

```
[[40, 50, 44, 2, 64, 54, 60, 57],
 [11, 1, 15, 60, 59, 2, 64, 54],
 [58, 59, 2, 41, 44, 40, 60, 59],
 [44, 51, 42, 54, 52, 44, 2, 64],
 [2, 50, 44, 44, 55, 2, 48, 53],
 [2, 22, 2, 52, 40, 64, 2, 53],
 [1, 20, 48, 61, 44, 58, 2, 58],
 [57, 44, 2, 59, 47, 44, 64, 13],
 [2, 52, 44, 7, 1, 14, 53, 43],
 [59, 1, 52, 40, 50, 44, 58, 2]]
```

0. Data

- Convert sequence of text to sequence of numbers: 1-65.
- Example (first 10 characters):

```
"First Citi" -> [19, 48, 57, 58, 59, 2, 16, 48, 59, 48]
```

- Data structured as:

xb:

```
[[26, 40, 50, 44, 2, 64, 54, 60],
 [53, 11, 1, 15, 60, 59, 2, 64],
 [54, 58, 59, 2, 41, 44, 40, 60],
 [62, 44, 51, 42, 54, 52, 44, 2],
 [58, 2, 50, 44, 44, 55, 2, 48],
 [57, 2, 22, 2, 52, 40, 64, 2],
 [7, 1, 20, 48, 61, 44, 58, 2],
 [40, 57, 44, 2, 59, 47, 44, 64],
 [45, 2, 52, 44, 7, 1, 14, 53],
 [48, 59, 1, 52, 40, 50, 44, 58]]
```

yb:

```
[[40, 50, 44, 2, 64, 54, 60, 57],
 [11, 1, 15, 60, 59, 2, 64, 54],
 [58, 59, 2, 41, 44, 40, 60, 59],
 [44, 51, 42, 54, 52, 44, 2, 64],
 [2, 50, 44, 44, 55, 2, 48, 53],
 [2, 22, 2, 52, 40, 64, 2, 53],
 [1, 20, 48, 61, 44, 58, 2, 58],
 [57, 44, 2, 59, 47, 44, 64, 13],
 [2, 52, 44, 7, 1, 14, 53, 43],
 [59, 1, 52, 40, 50, 44, 58, 2]]
```

0. Data

- Convert sequence of text to sequence of numbers: 1-65.
- Example (first 10 characters):

```
"First Citi" -> [19, 48, 57, 58, 59, 2, 16, 48, 59, 48]
```

- Data structured as:

xb:

```
[[26, 40, 50, 44, 2, 64, 54, 60],
 [53, 11, 1, 15, 60, 59, 2, 64],
 [54, 58, 59, 2, 41, 44, 40, 60],
 [62, 44, 51, 42, 54, 52, 44, 2],
 [58, 2, 50, 44, 44, 55, 2, 48],
 [57, 2, 22, 2, 52, 40, 64, 2],
 [7, 1, 20, 48, 61, 44, 58, 2],
 [40, 57, 44, 2, 59, 47, 44, 64],
 [45, 2, 52, 44, 7, 1, 14, 53],
 [48, 59, 1, 52, 40, 50, 44, 58]]
```

yb:

```
[[40, 50, 44, 2, 64, 54, 60, 57],
 [11, 1, 15, 60, 59, 2, 64, 54],
 [58, 59, 2, 41, 44, 40, 60, 59],
 [44, 51, 42, 54, 52, 44, 2, 64],
 [2, 50, 44, 44, 55, 2, 48, 53],
 [2, 22, 2, 52, 40, 64, 2, 53],
 [1, 20, 48, 61, 44, 58, 2, 58],
 [57, 44, 2, 59, 47, 44, 64, 13],
 [2, 52, 44, 7, 1, 14, 53, 43],
 [59, 1, 52, 40, 50, 44, 58, 2]]
```

0. Data

- Convert sequence of text to sequence of numbers: 1-65.
- Example (first 10 characters):

```
"First Citi" -> [19, 48, 57, 58, 59, 2, 16, 48, 59, 48]
```

- Data structured as:

xb:

```
[[26, 40, 50, 44, 2, 64, 54, 60],
 [53, 11, 1, 15, 60, 59, 2, 64],
 [54, 58, 59, 2, 41, 44, 40, 60],
 [62, 44, 51, 42, 54, 52, 44, 2],
 [58, 2, 50, 44, 44, 55, 2, 48],
 [57, 2, 22, 2, 52, 40, 64, 2],
 [7, 1, 20, 48, 61, 44, 58, 2],
 [40, 57, 44, 2, 59, 47, 44, 64],
 [45, 2, 52, 44, 7, 1, 14, 53],
 [48, 59, 1, 52, 40, 50, 44, 58]]
```

yb:

```
[[40, 50, 44, 2, 64, 54, 60, 57],
 [11, 1, 15, 60, 59, 2, 64, 54],
 [58, 59, 2, 41, 44, 40, 60, 59],
 [44, 51, 42, 54, 52, 44, 2, 64],
 [2, 50, 44, 44, 55, 2, 48, 53],
 [2, 22, 2, 52, 40, 64, 2, 53],
 [1, 20, 48, 61, 44, 58, 2, 58],
 [57, 44, 2, 59, 47, 44, 64, 13],
 [2, 52, 44, 7, 1, 14, 53, 43],
 [59, 1, 52, 40, 50, 44, 58, 2]]
```

0. Data

```
data {
  int<lower = 1> vocab_size;            // e.g. 65
  int<lower = 1> batch_size;            // e.g. 32
  int<lower = 1> block_size;            // e.g. 8

  array[batch_size, block_size] int<lower = 1, upper = vocab_size> xb;
  array[batch_size, block_size] int<lower = 1, upper = vocab_size> yb;

  array[batch_size, block_size] int<lower = 1, upper = vocab_size> xb_val;
  array[batch_size, block_size] int<lower = 1, upper = vocab_size> yb_val;

  int<lower = 0> max_new_tokens;
}
```

1. Bigram Model

- Only look at last character to predict the next.
- Each token has contains a 65-length unnormalized logit parameter

```
parameters {
  array[vocab_size] vector[vocab_size] token_embedding;
}
transformed parameters {
  real loss = 0;
  for (b in 1:batch_size) {
    for (t in 1:block_size) {
      loss += categorical_logit_lpmf(yb[b, t] | token_embedding[xb[b, t]]);
    }
  }
  loss /= batch_size * block_size;
}
model {
  target += loss;
}
```

Parameters: 65*65 = 4225

Cross-entropy loss
aka log loss

2. Embedding Size

- Different embedding size: e.g. 32

```
functions {
  vector lm_head(vector x, matrix A, vector b) {
    return A * x + b;
  }
}
data {
  int<lower = 1> n_embed;               // e.g. 32
}
parameters {
  array[vocab_size] vector[n_embed] token_embedding;
  matrix[vocab_size, n_embed] lm_head_multiplier;
  vector[vocab_size] lm_head_offset;
}
transformed parameters {
  real loss = 0;
  for (b in 1:batch_size) {
    for (t in 1:block_size) {
      vector[vocab_size] logits = lm_head(token_embedding[xb[b, t]],
                                          lm_head_multiplier, lm_head_offset);
      loss += categorical_logit_lpmf(yb[b, t] | logits);
    }
  }
  loss /= batch_size * block_size;
}
model {
  target += loss;
}
```

Language modeling head

token_embedding: 65*32 = 2080
lm_head_multiplier: 2080
lm_head_offset: 65

Unnormalized logit vector
length: 65

3. Positional Embedding

- Start using position

```
parameters {
  array[vocab_size] vector[n_embed] token_embedding;
  matrix[vocab_size, n_embed] lm_head_multiplier;
  vector[vocab_size] lm_head_offset;

  array[block_size] vector[n_embed] position_embedding;
}
transformed parameters {
  array[batch_size, block_size] vector[n_embed] x;
  for (b in 1:batch_size) {
    for (t in 1:block_size) {
      x[b, t] = token_embedding[xb[b, t]] + position_embedding[t];
    }
  }

  real loss = 0;
  for (b in 1:batch_size) {
    for (t in 1:block_size) {
      vector[vocab_size] logits = lm_head(x[b, t], lm_head_multiplier, lm_head_offset);

      loss += categorical_logit_lpmf(yb[b, t] | logits);
    }
  }
  loss /= batch_size * block_size;
}
model {
  target += loss;
}
```

new parameter

Use position embedding

Unnormalized logit vector length: 65

4. Self-Attention

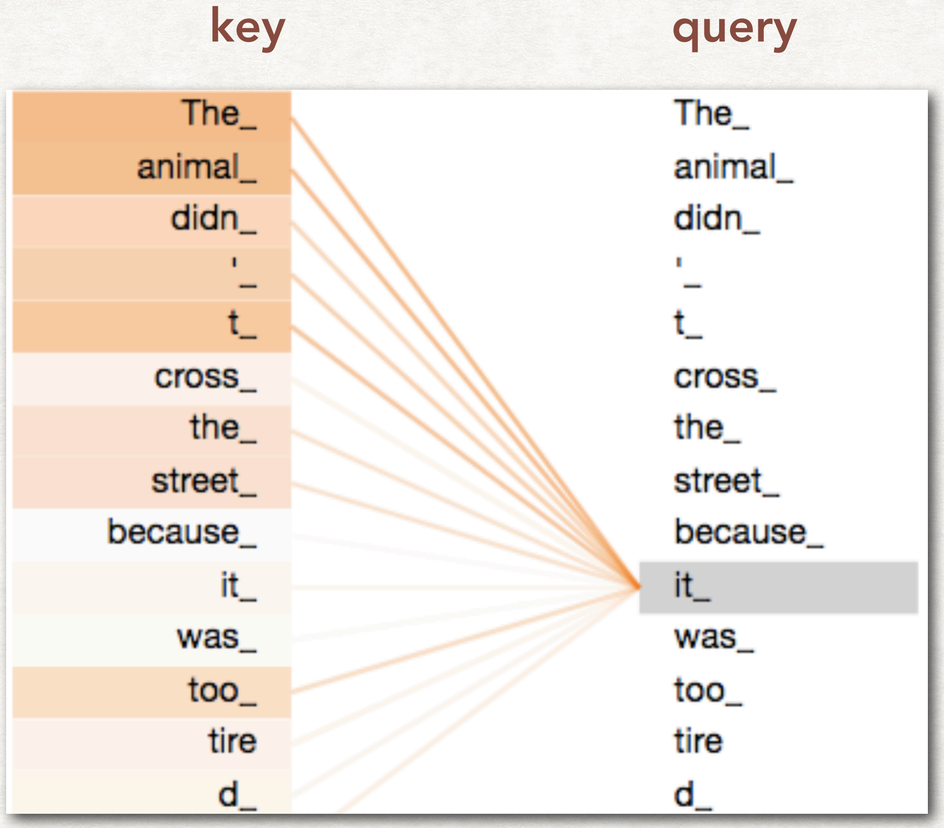
```
parameters {
  array[vocab_size] vector[n_embed] token_embedding;
  matrix[vocab_size, n_embed] lm_head_multiplier;
  vector[vocab_size] lm_head_offset;

  array[block_size] vector[n_embed] position_embedding;

  // Self Attention
  matrix[n_embed, n_embed] key;
  matrix[n_embed, n_embed] query;
  matrix[n_embed, n_embed] value;
}
transformed parameters {
  array[batch_size, block_size] vector[n_embed] x;
  for (b in 1:batch_size) {
    for (t in 1:block_size) {
      x[b, t] = token_embedding[xb[b, t]] + position_embedding[t];
    }
  }

  x = self_attention(x, key, query, value);

  real loss = 0;
  for (b in 1:batch_size) {
    for (t in 1:block_size) {
      vector[vocab_size] logits = lm_head(x[b, t], lm_head_multiplier, lm_head_offset);
      loss += categorical_logit_lpmf(yb[b, t] | logits);
    }
  }
  loss /= batch_size * block_size;
}
model {
  target += loss;
}
```

Self-Attention parameters:
key, query, value

Self-attention

4. Self-Attention (example)

- Example: "The animal didn't cross the street because it was too tired."



The Illustrated Transformer. http://jalammar.github.io/illustrated-transformer/

4. Self-Attention (function)

```
functions {
  // self_attention for a single block
  array[] vector self_attention(array[] vector x,  // block_size, vector[n_embed]
                                matrix key,        // n_embed x head_size
                                matrix query,      // n_embed x head_size
                                matrix value) {    // n_embed x head_size
    // -> block_size, head_size
    int block_size = dims(x)[1];
    int n_embed = rows(key);
    int head_size = cols(key);


    matrix[block_size, n_embed] x_matrix;
    for (t in 1:block_size) {
      x_matrix[t, :] = x[t]';
    }
    matrix[block_size, head_size] k = x_matrix * key;
    matrix[block_size, head_size] q = x_matrix * query;
    matrix[block_size, head_size] v = x_matrix * value;

    // decoder block: wei is lower triangular matrix
    //   allows communication from different parts of the visible context
    //   e.g. 1st output only allows 1st x, 2nd output allows 1st and second, etc.
    matrix[block_size, block_size] wei = rep_matrix(0, block_size, block_size);
    matrix[block_size, block_size] tmp_wei = q * k' / sqrt(head_size);
    for (t in 1:block_size) {
      wei[t, 1:t] = softmax(tmp_wei[t, 1:t]')';
    }

    matrix[block_size, head_size] weighted_value = wei * v;
    array[block_size] vector[head_size] out;

    for (t in 1:block_size) {
      out[t] = weighted_value[t, :]';
    }
    return out;
  }
}
```

Use key, query, value

Scaled dot-product self-attention

Encoder: lower triangular matrix

Output

4. Self-Attention

```
parameters {
  array[vocab_size] vector[n_embed] token_embedding;
  matrix[vocab_size, n_embed] lm_head_multiplier;
  vector[vocab_size] lm_head_offset;

  array[block_size] vector[n_embed] position_embedding;

  // Self Attention
  matrix[n_embed, n_embed] key;
  matrix[n_embed, n_embed] query;
  matrix[n_embed, n_embed] value;
}
transformed parameters {
  array[batch_size, block_size] vector[n_embed] x;
  for (b in 1:batch_size) {
    for (t in 1:block_size) {
      x[b, t] = token_embedding[xb[b, t]] + position_embedding[t];
    }
  }

  x = self_attention(x, key, query, value);

  real loss = 0;
  for (b in 1:batch_size) {
    for (t in 1:block_size) {
      vector[vocab_size] logits = lm_head(x[b, t], lm_head_multiplier, lm_head_offset);
      loss += categorical_logit_lpmf(yb[b, t] | logits);
    }
  }
  loss /= batch_size * block_size;
}
model {
  target += loss;
}
```

Self-attention

5. Multi-Headed Self-Attention

- Introduce: n_head

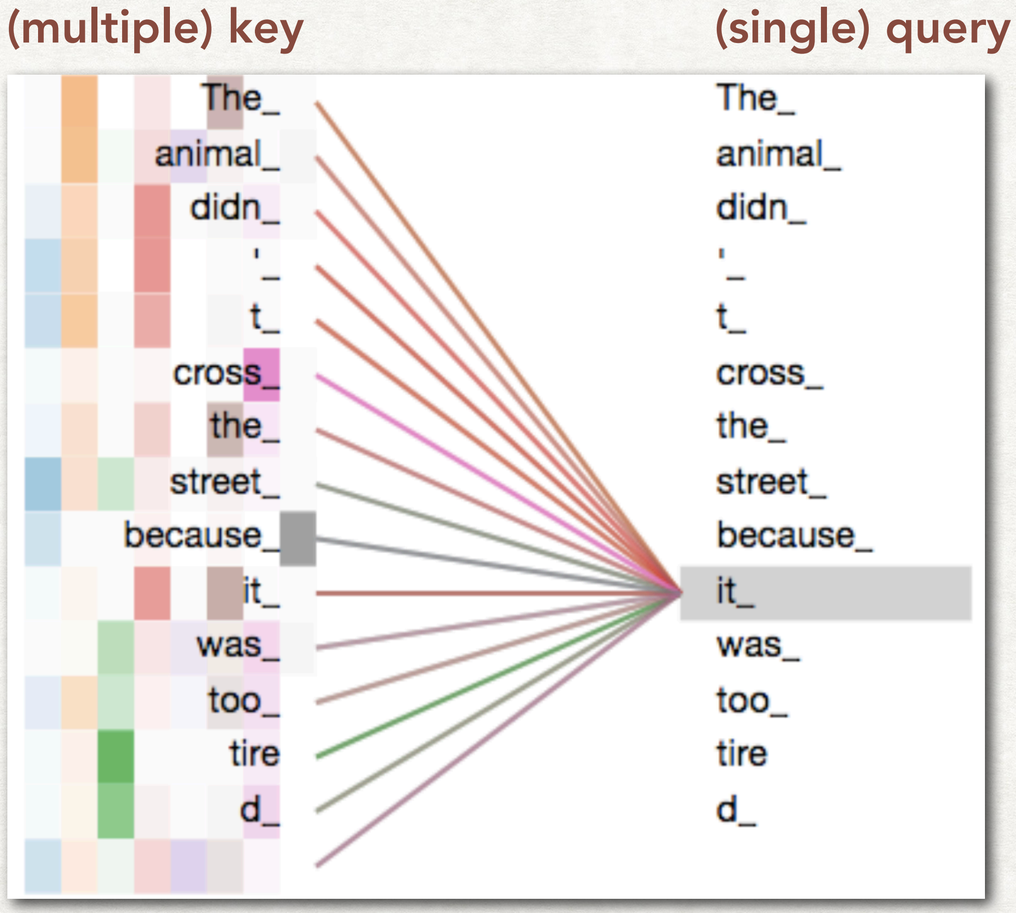
```
parameters {
  array[vocab_size] vector[n_embed] token_embedding;
  matrix[vocab_size, n_embed] lm_head_multiplier;
  vector[vocab_size] lm_head_offset;

  array[block_size] vector[n_embed] position_embedding;

  // Multi-head self attention
  array[n_head] matrix[n_embed, head_size] key;
  array[n_head] matrix[n_embed, head_size] query;
  array[n_head] matrix[n_embed, head_size] value;
}
transformed parameters {
  array[batch_size, block_size] vector[n_embed] x;
  for (b in 1:batch_size) {
    for (t in 1:block_size) {
      x[b, t] = token_embedding[xb[b, t]] + position_embedding[t];
    }
  }

  x = multi_head_self_attention(x, key, query, value);

  real loss = 0;
  for (b in 1:batch_size) {
    for (t in 1:block_size) {
      vector[vocab_size] logits = lm_head(x[b, t], lm_head_multiplier, lm_head_offset);
      loss += categorical_logit_lpmf(yb[b, t] | logits);
    }
  }
  loss /= batch_size * block_size;
}
model {
  target += loss;
}
```

Multiple self-attention parameters: key, query, value

Multi-headed self-attention

5. Multi-Headed Self-Attention (example)

- Example: "The animal didn't cross the street because it was too tired."



The Illustrated Transformer. http://jalammar.github.io/illustrated-transformer/

5. Multi-Headed Self-Attention (function)

```
functions {
  // multi head self attention
  array[,] vector multi_head_self_attention(array[,] vector x,          // batch_size, block_size, n_embed
                                            array[] matrix key,         // n_heads, n_embed x head_size
                                            array[] matrix query,       // n_heads, n_embed x head_size
                                            array[] matrix value) {     // n_heads, n_embed x head_size
    // -> n_embed, block_size, n_embed
    int batch_size = dims(x)[1];
    int block_size = dims(x)[2];
    int n_head = dims(key)[1];
    int n_embed = rows(key[1]);
    int head_size = cols(key[1]);

    array[batch_size, block_size] vector[n_embed] out;
    for (b in 1: batch_size) {
      for (n in 1:n_head) {
        array[block_size] vector[head_size] x_self = self_attention(x[b], key[n], query[n], value[n]);
        for (t in 1:block_size) {
          out[b, t, (((n-1) * head_size) + 1):(n * head_size)] = x_self[t];
        }
      }
    }
    return out;
  }
}
```

Repeat self-attention multiple times
concatenate results

6+. Feed Forward, Skip Connection, Larger Feed Forward, Blocks, Dropout

- More to make this work
- Skip for time
- See: https://github.com/bayesianops/gpt-tutorial

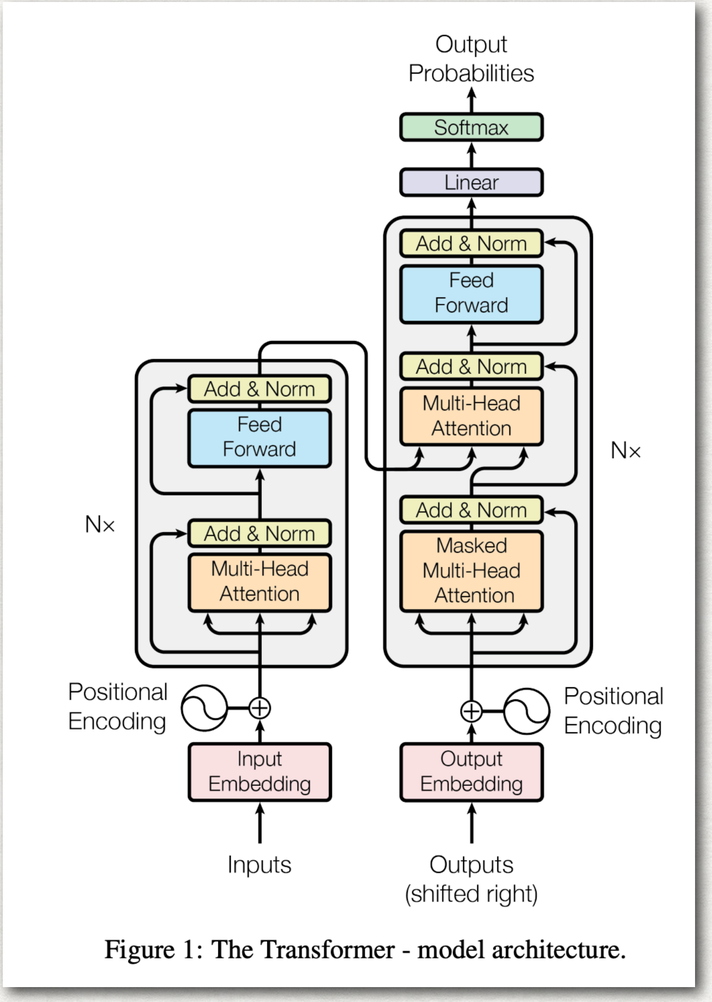


Figure 1: The Transformer - model architecture.

arXiv:1706.03762 [cs.CL]
"Attention is All You Need"

There's a statistical model

Statistical Model

- Data: X
- Parameters: θ
- Statistical Model: $p(\theta, X)$

Statistical Model

- Data: X
- Parameters: θ
- Statistical Model: $p(\theta, X)$
  - $p(\theta, X)$ is a function. It returns a single number.
  - $p : \mathbb{R}^{n+m} \to \mathbb{R}_{\geq 0}$ where $\theta \in \mathbb{R}^n$ and $X \in \mathbb{R}^m$
  - Model links data and parameters. Higher is better.

Inference is separate

Three Types of Inference

- 3 principled types with different goals.
  (and lots of ad-hoc methods that *work)

  1. Optimization.
     Find the set of parameters that give highest value.

  2. Approximate inference.
     The problem is hard. Pick an easier problem to solve.
     Find the parameters of the easier problem that give the best answer.

  3. Bayesian inference.
     Find the distribution of the parameter values that make sense given the data.

Three Types of Inference

- 3 principled types with different goals.
  (and lots of ad-hoc methods that *work)

1. Optimization.

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\ p(\theta, X)$$

2. Approximate inference. Example: ADVI.

$$\hat{p}(\theta|X) \approx q(\hat{\phi}) \qquad \text{where } \hat{\phi} = \underset{\phi}{\operatorname{argmin}}\ D_{\mathrm{KL}}\left(q(\theta|\,\phi)\ ||\ p(\theta, X)\right)$$

3. Bayesian inference. Example: MCMC

$$p(\theta \mid X) \qquad \text{approximated with} \qquad \{\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(N)}\}$$

Inference on GPT

- Use stochastic optimization.
- Break data into small chunks
  - Update subset of parameters
  - Repeat
- Why?

When To Use / Not Use
(Opinion)

- Optimization
  - Use: Usable estimates without uncertainty, speed.
  - Don't use: Not converging; hierarchical models. (Try optimizing 8 schools)
- Bayesian Inference
  - Use: Parameter inference, uncertainty important. Complex models.
  - Don't use: Speed, multi-modal posterior.
- Approximate Inference
  - Use: Complex models without need for true uncertainty.
  - Don't use: Need uncertainty but don't know how good approximate algorithm is.

Takeaways

Recap

1. GPT can be written in PPL

2. Statistical models can be separated from inference

3. Choose inference technique wisely

References

- Andrej Karparthy. "Let's build GPT: from scratch, in code, spelled out." (YouTube) https://youtu.be/kCc8FmEb1nY
- Vaswani, Ashish, et al. "Attention is all you need." (Paper) https://arxiv.org/abs/1706.03762
- Jay Alammar. "The Illustrated Transformer." (Blog) http://jalammar.github.io/illustrated-transformer/
- Daniel Lee. StanCon 2023 GPT Tutorial. (Code) https://github.com/bayesianops/gpt-tutorial
- Stan. (Code) https://github.com/stan-dev/

Thank you!

Daniel Lee
daniel@bayesianops.com
https://www.linkedin.com/in/syclik/