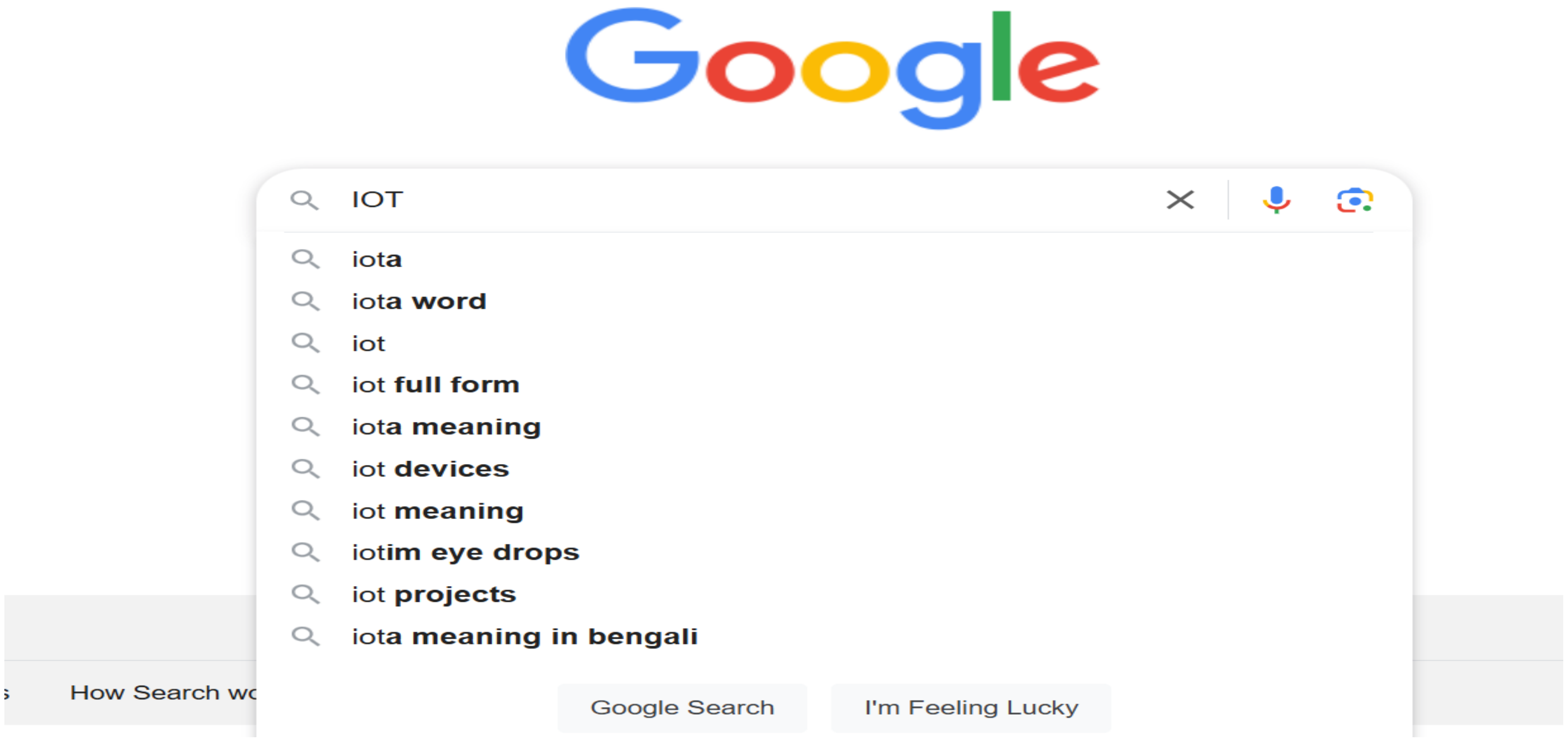
Lecture 2: Generative AI - Shaping the Future of Creativity and Innovation

We use language models every day



IOT

iota

iota word

iot

iot full form

iota meaning

iot devices

iot meaning

iotim eye drops

iot projects

iota meaning in bengali

How do we learn a language model?

Estimate probabilities using text data

● Collect a textual corpus

● Find a distribution that maximizes the probability of the corpus – maximum likelihood estimation

A naive solution: count and divide

Assume we have N training sentences

● Let x1 , x2 , …, xn be a sentence, and c(x1 , x2 , …, xn) be the number of times it appeared in the training data.

● Define a language model:

$$p(x_1, \ldots, x_n) = \frac{c(x_1, \ldots, x_n)}{N}$$

Unigram probability

“I have a dog whose name is Alpha. I have two cats, they like playing with Alpha”

corpus size m = 17

- P(Alpha) = 2/17; P(cats) = 1/17
- Unigram probability: $P(w) = \frac{\text{count}(w)}{m} = \frac{C(w)}{m}$

Bigram probability

"I have a dog whose name is Alpha. I have two cats, they like playing with Alpha"

$$P(A \mid B) = \frac{P(A,B)}{P(B)}$$

$$P(\text{have} \mid \text{I}) = \frac{P(\text{I have})}{P(\text{I})} = \frac{2}{2} = 1$$

$$P(\text{two} \mid \text{have}) = \frac{P(\text{have two})}{P(\text{have})} = \frac{1}{2} = 0.5$$

$$P(\text{eating} \mid \text{have}) = \frac{P(\text{have eating})}{P(\text{have})} = \frac{0}{2} = 0$$

$$P(w_2|w_1) = \frac{C(w_1,w_2)}{\sum_w C(w_1,w)} = \frac{C(w_1,w_2)}{C(w_1)}$$

Trigram probability/n-gram probability

- "I have a dog whose name is Alpha. I have two cats, they like playing with Alpha"

$$P(A \mid B) = \frac{P(A,B)}{P(B)}$$

$$P(a \mid I\ have) = \frac{C(I\ have\ a)}{C(I\ have)} = \frac{1}{2} = 0.5$$

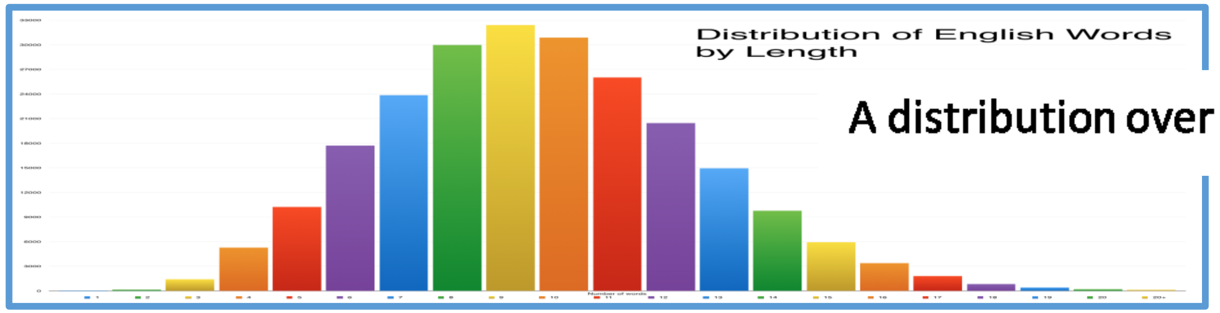
$$P(several \mid I\ have) = \frac{C(I\ have\ several)}{C(I\ have)} = \frac{0}{2} = 0$$

$$P(w_3 \mid w_1\, w_2) = \frac{C(w_1, w_2, w_3)}{\sum_w C(w_1, w_2, w)} = \frac{C(w_1, w_2, w_3)}{C(w_1, w_2)}$$

$$P(A \mid B) = \frac{P(A,B)}{P(B)}$$

$$P(w_i \mid w_1, w_2, \ldots, w_{i-1}) = \frac{C(w_1, w_2, \ldots, w_{i-1}, w_i)}{C(w_1, w_2, \ldots, w_{i-1})}$$

Neural language models



A distribution over the vocabulary

$f(\Theta)$

A differentiable function (e.g. a neural network)

I am a student of UEM

Sigmoid Function (e.g.☺
hyperbolic tangent function
ReLU (Rectified Linear Unit) Function
Softmax Function
SoftPlus Function

How do we maximize the likelihood?

The dominant strategy from the past decade:

1. The randomly initialized differentiable function (neural network) takes the context as input

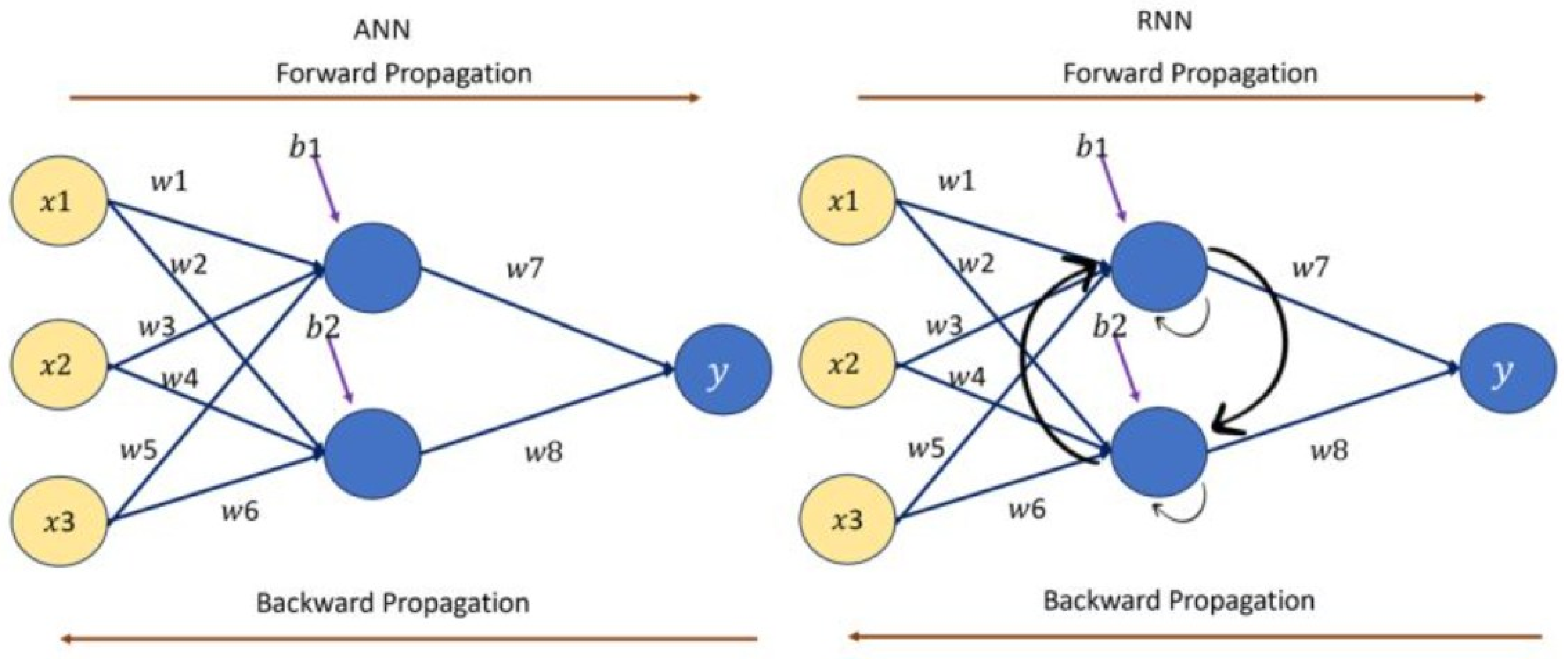
2. Have that function output a probability distribution over the vocabulary

3. Treat the probability of the correct token as your objective to maximize.

4. Or negative log (probability) as your objective to minimize

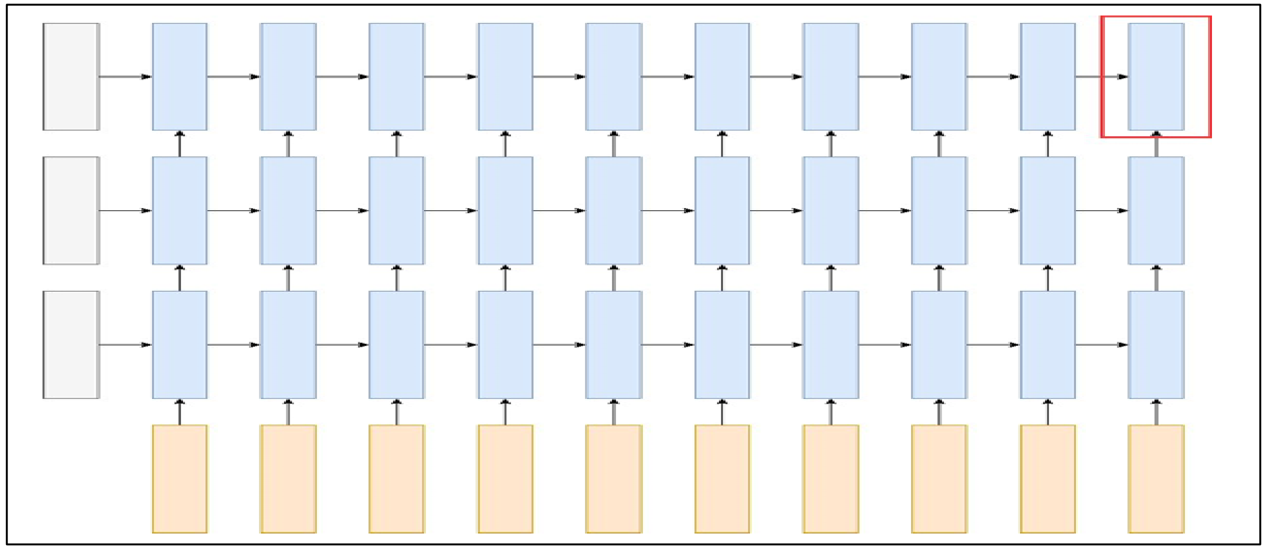
5. Differentiate with respect to the parameters, and perform **gradient descent, or Stochastic Gradient Descent**

What optimization!!!!



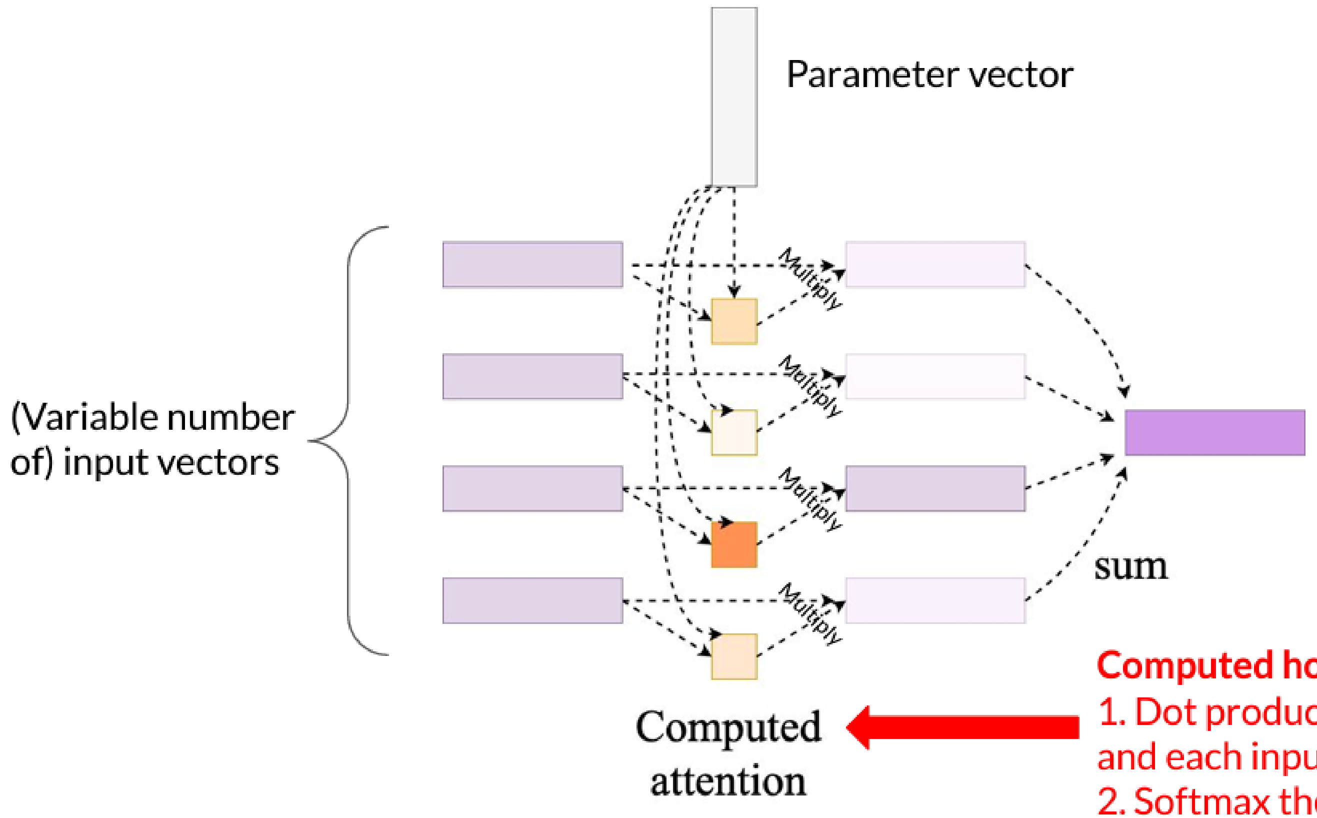
Why did the transformer make such a big difference for language modeling?

A 3-layer LSTM's calculations for an input of 10 tokens



Tokenization: The input text is divided into smaller units called tokens, which can be words, subwords, or characters. This process allows the model to process the text more efficiently.

A simple form of attention



Computed how?
1. Dot product between param vector and each input vector
2. Softmax the set of resulting scalars.

Attention mechanisms: The attention mechanism used in the original transformer architecture is called scaled dot-product attention. The input consists of queries and keys of dimension d_k and values of dimension dv. We compute the dot products of the query with all keys, divide each by $\sqrt{d_k}$, and apply a softmax function to obtain the weights on the values.

Pros and cons

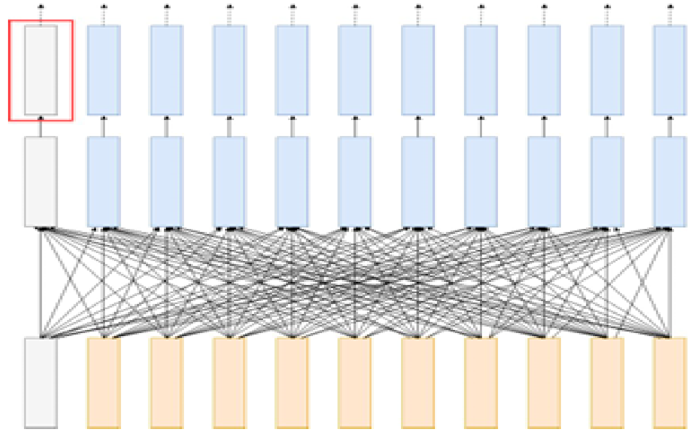
Pros:

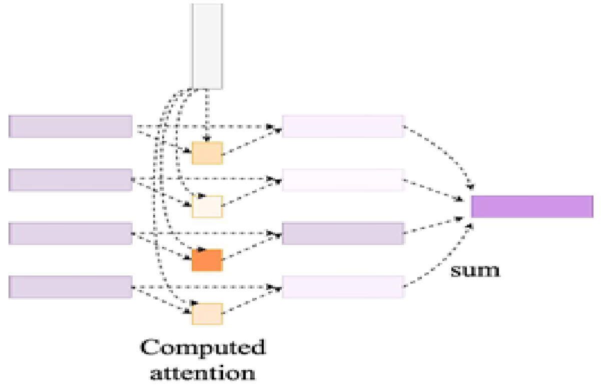
● We have a function that can compute a weighted average (largely) in parallel of an arbitrary number of vectors!

● The parameters determining what makes it into our output representation are learned

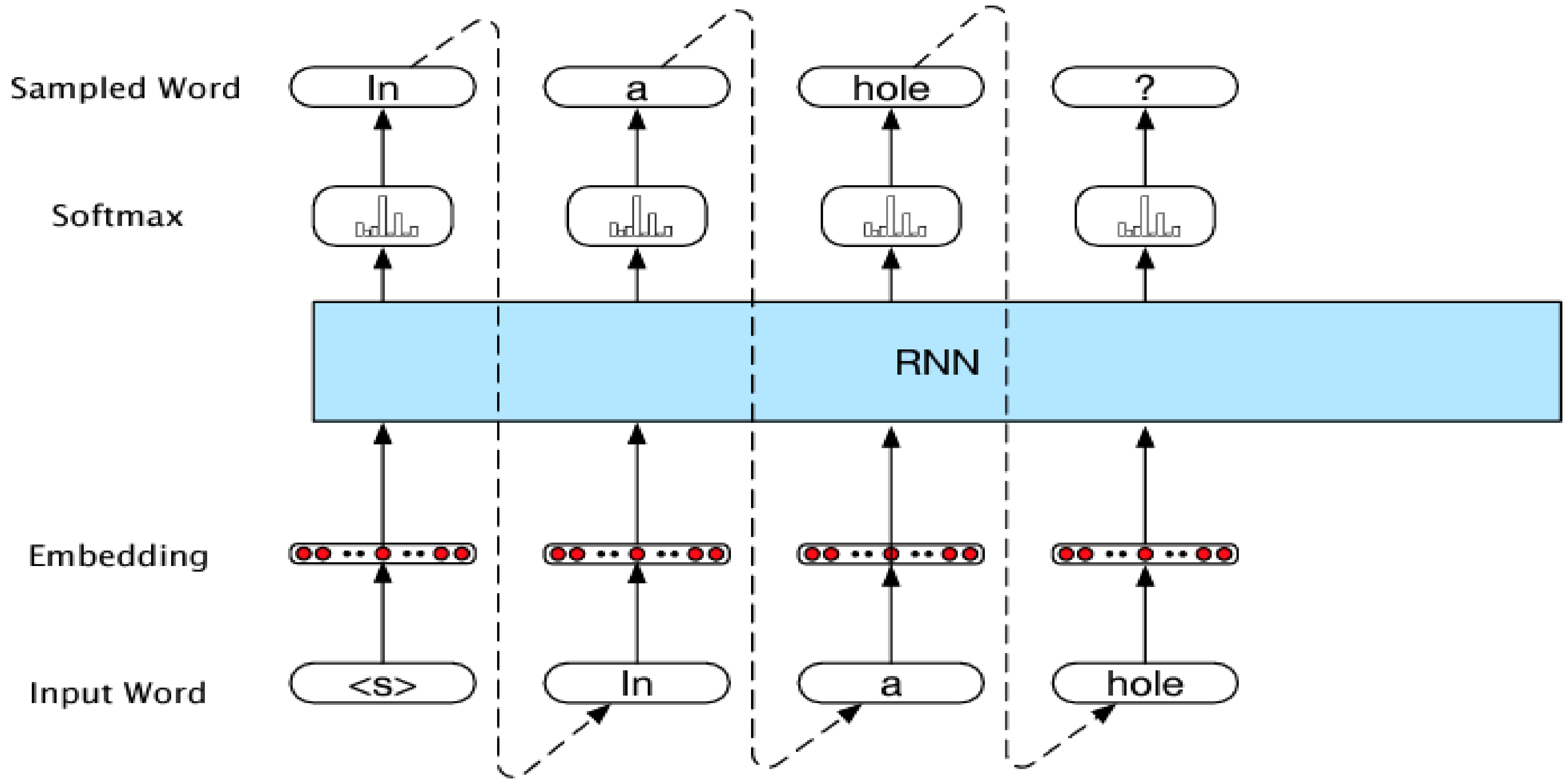
Cons:

● We're also hoping to produce n different output token representations… and this just produces one…

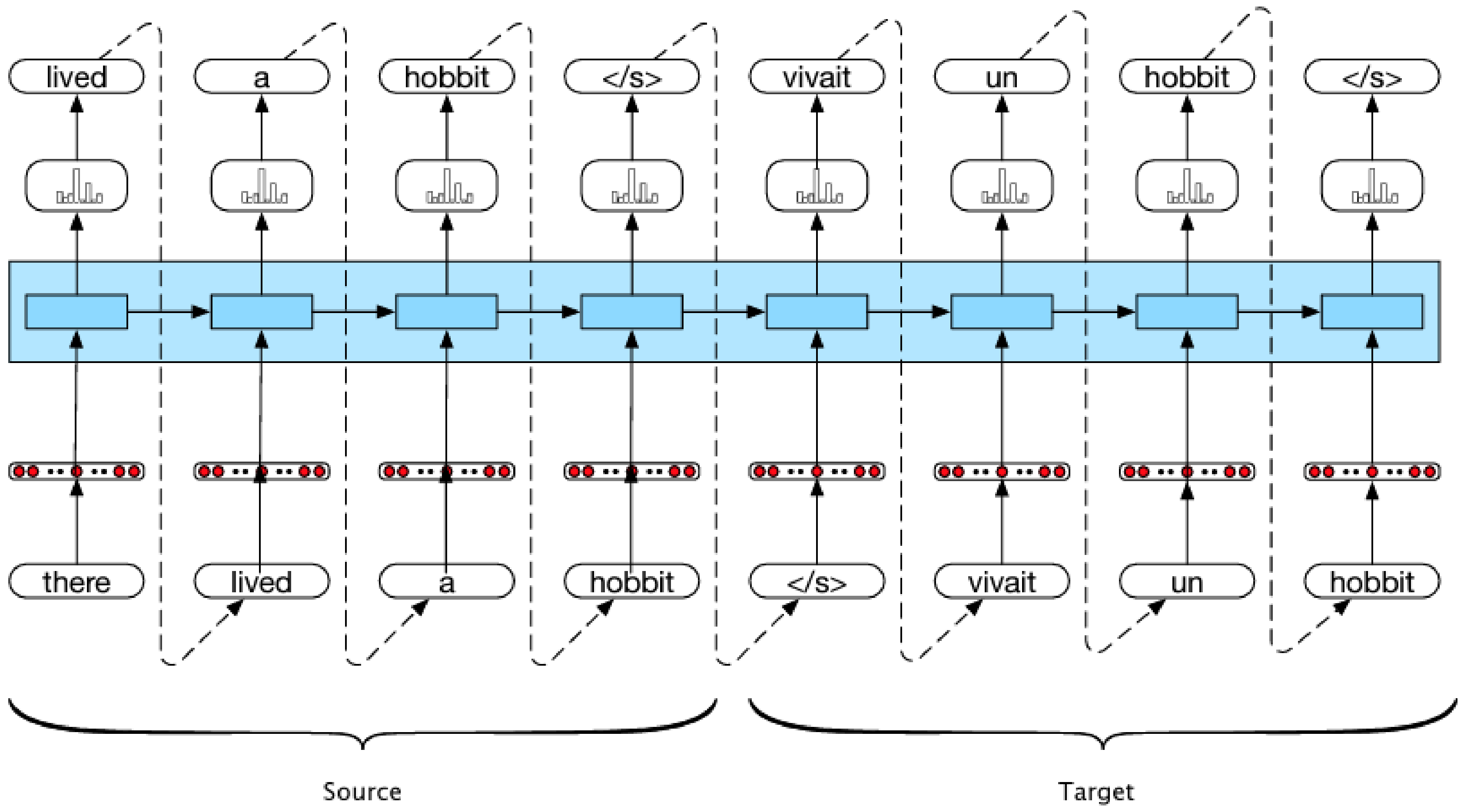




RNNs for language generation -Autoregressive generation

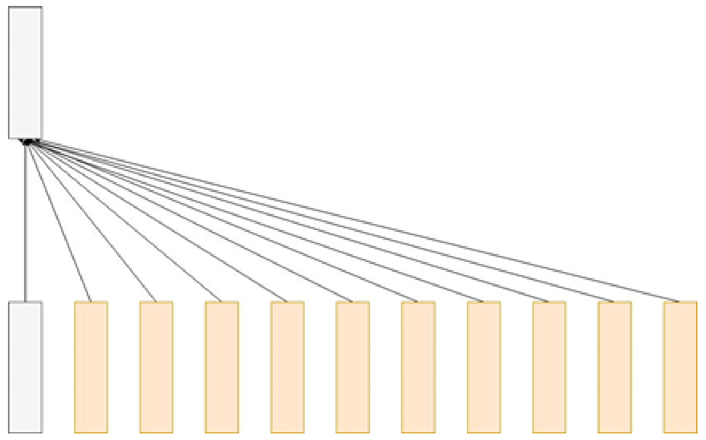


RNNs for language generation - Machine Translation



Building up to the attention mechanism

What about an average?
‘But we probably don’t want to weight all input vectors equally… How about a weighted average? Great idea! How can we automatically decide the weights for a weighted average of the input vectors?



What kind of function can take in a variable number of inputs?

First AI generated Image

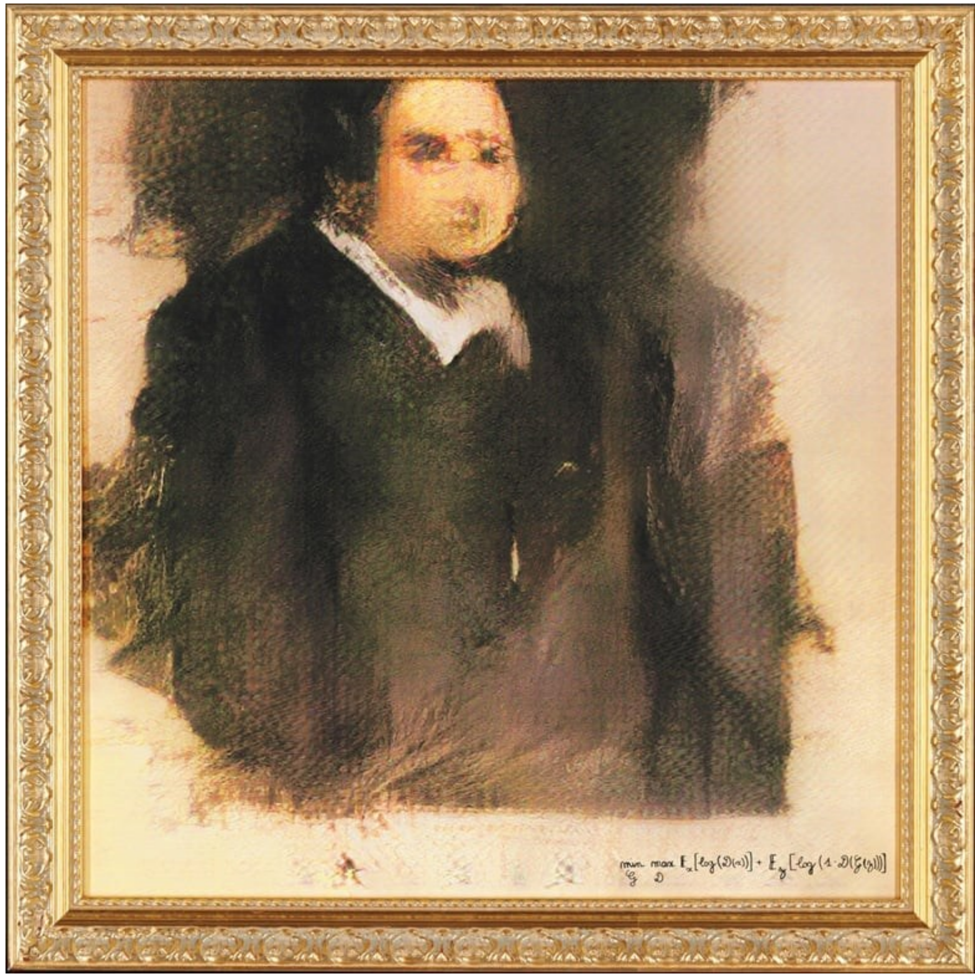
```
from PIL import Image

# Image dimensions
width, height = 512, 512

image = Image.new('RGB', (width, height))

# Generate a gradient
for x in range(width):
    for y in range(height):
        # Set pixel color based on position
        red = int(255 * (x / width))      # Gradient for red
        green = int(255 * (y / height))  # Gradient for green
        blue = 128                        # Constant blue value
        image.putpixel((x, y), (red, green, blue))


image.save("C:/Users/45919/Desktop/UEM_classs/gradient_image.png")
 print("Image saved as 'gradient_image.png'")
```



Edmond de Belamy Fetching $432,500

For attendance

```
import openai

# Set your OpenAI API key
openai.api_key = “TTTTTTTTTTT"
######################################################


prompt = "An of a cat, standing in a roof on a sunny day, smiling."


response = openai.Image.create(
    prompt=prompt,
    n=1,  # Number of images
    size="512x512"  )

# Get the URL of the generated image
image_url = response['data'][0]['url']
print(f"Generated Image URL: {image_url}")
```

Thanks for listening