Lecturer Introduction

Arthur Szlam

Arthur Szlam is a research scientist at Facebook AI Research in New York. He was previously on the mathematics faculty of the City College of New York, and more previously, earned his PhD in mathematics from Yale University.

Softmax and Preview of Attention

FACEBOOK AI Georgia Tech

- “Attention”: weighting or probability distribution over inputs that depends on computational state and inputs.

- Attention allows information to propagate directly between “distant” computational nodes while making minimal structural assumptions.

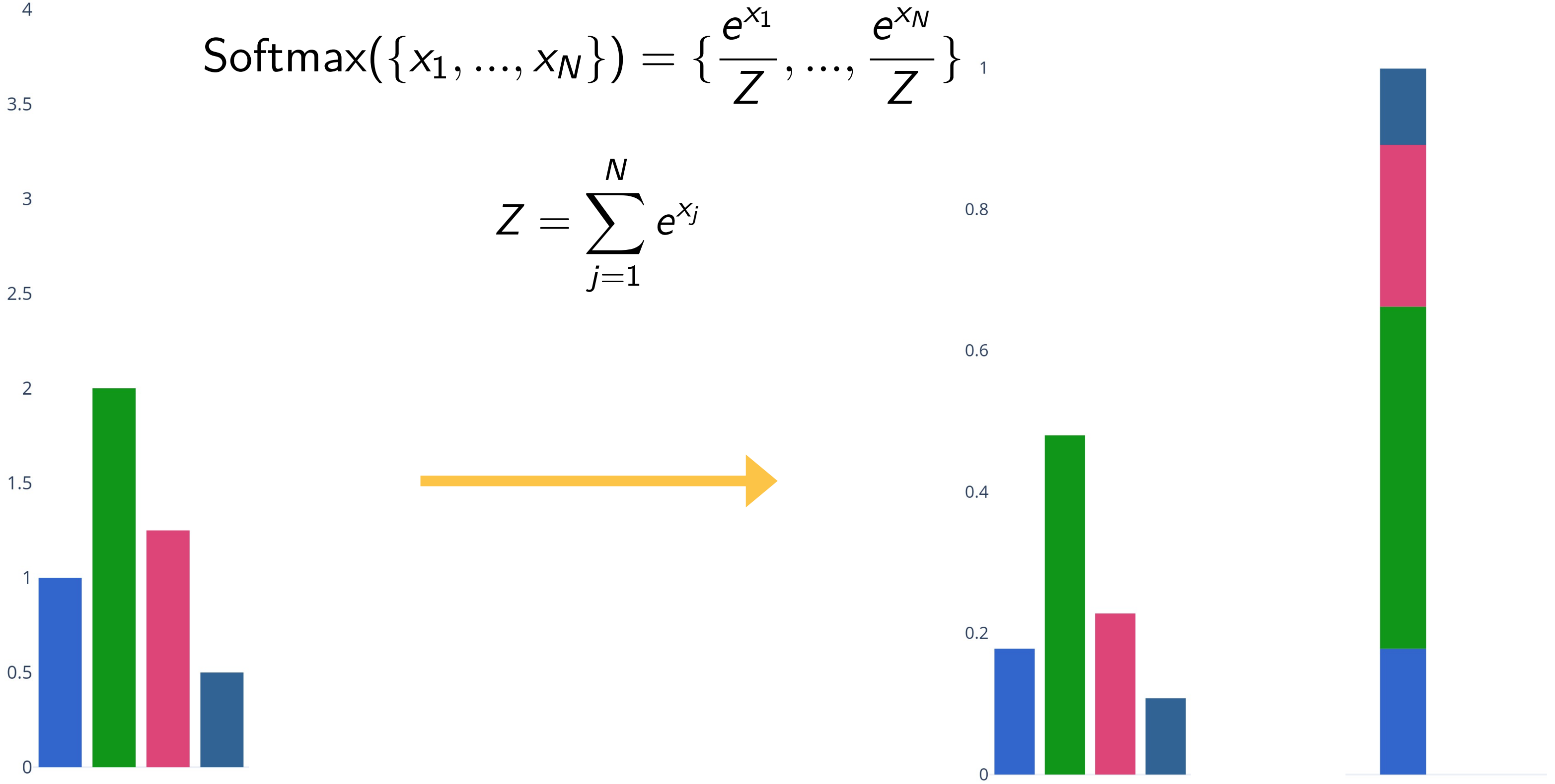
- The most standard form of attention in current neural networks is implemented with the Softmax, which we will now review.

- Given $\{x_1, \ldots, x_N\}$, each $x_j \in \mathbb{R}$,
- $\text{Softmax}(\{x_1, \ldots, x_N\}) = \{\frac{e^{x_1}}{Z}, \ldots, \frac{e^{x_N}}{Z}\}$, with $Z = \sum_{j=1}^{N} e^{x_j}$.
- For any inputs the softmax returns a probability distribution
- Softmax is permutation equivariant (a permutation of the input leads to the same permutation of the output)

Review of Softmax Function

$$\text{Softmax}(\{x_1, ..., x_N\}) = \{\frac{e^{x_1}}{Z}, ..., \frac{e^{x_N}}{Z}\}$$

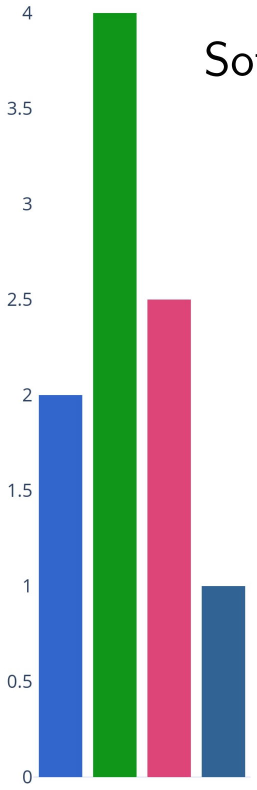
$$Z = \sum_{j=1}^{N} e^{x_j}$$

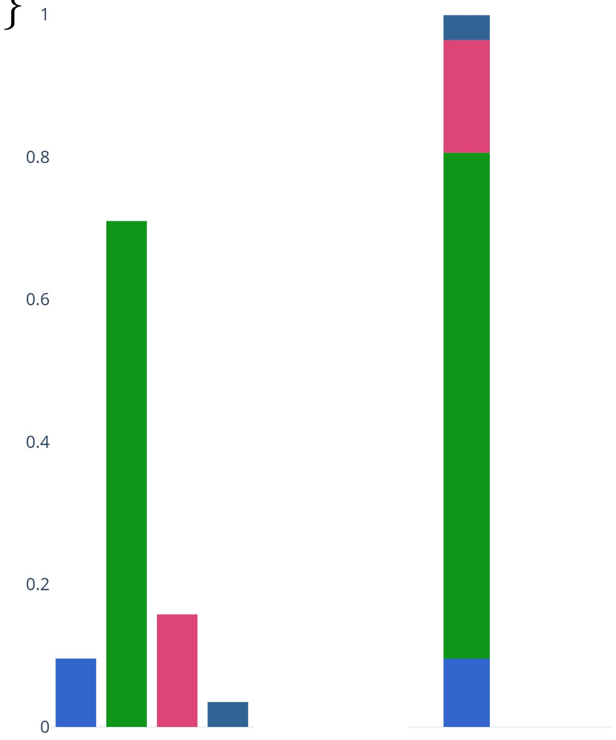


Review of Softmax Function

$$\text{Softmax}(\{x_1, \ldots, x_N\}) = \{\frac{e^{x_1}}{Z}, \ldots, \frac{e^{x_N}}{Z}\}$$

$$Z = \sum_{j=1}^{N} e^{x_j}$$

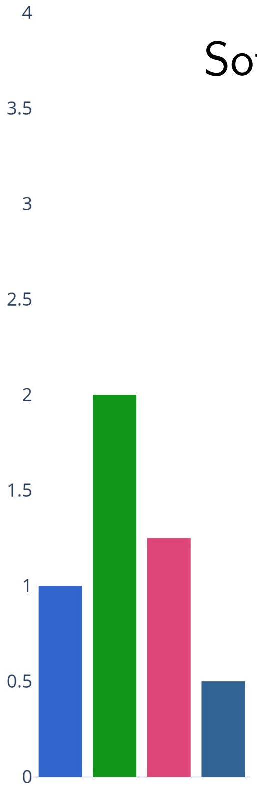


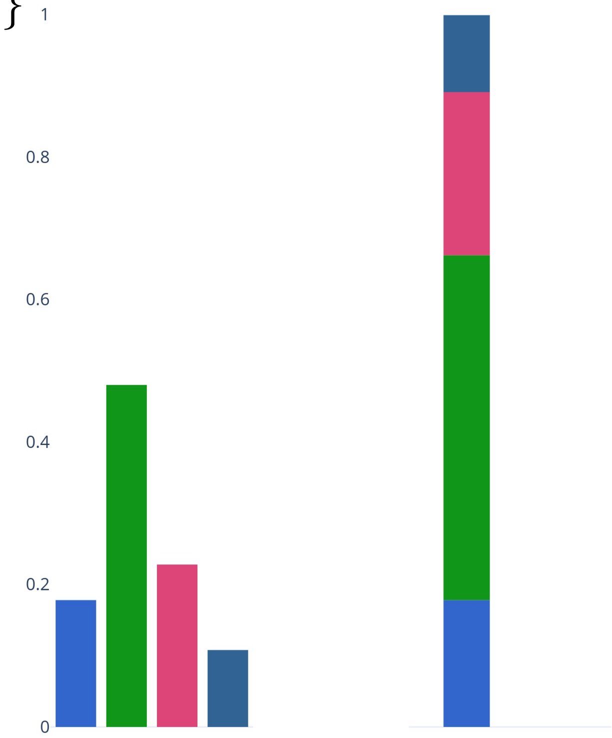


Review of Softmax Function

$$\text{Softmax}(\{x_1, \ldots, x_N\}) = \{\frac{e^{x_1}}{Z}, \ldots, \frac{e^{x_N}}{Z}\}$$

$$Z = \sum_{j=1}^{N} e^{x_j}$$

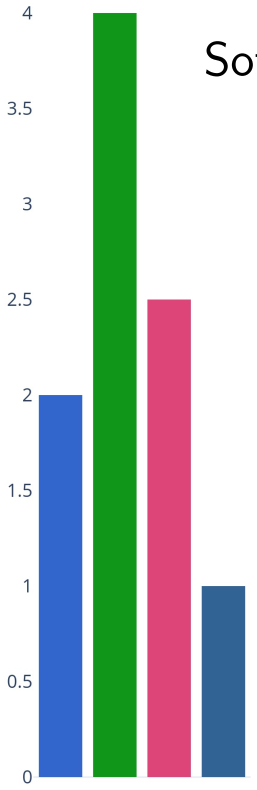


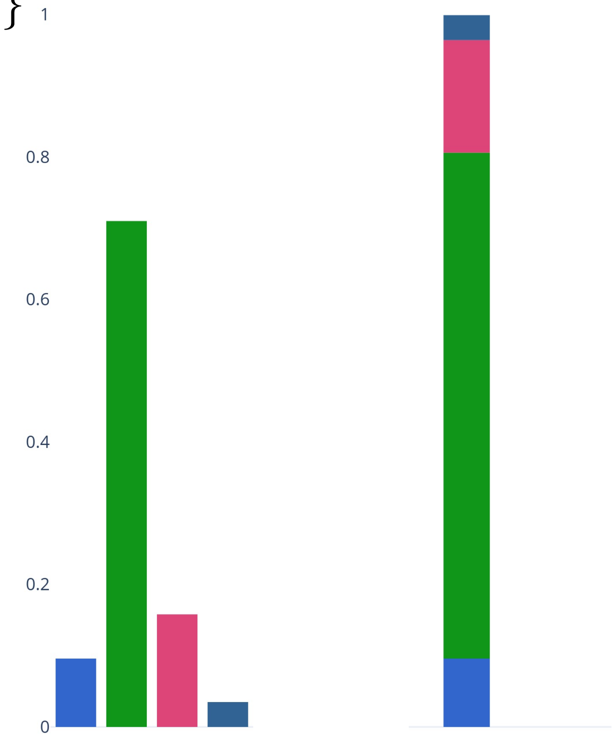


Review of Softmax Function

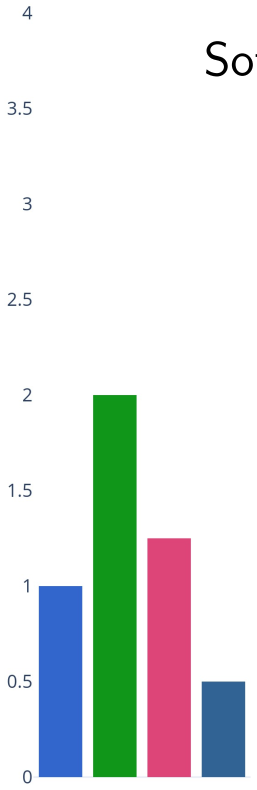
$$\text{Softmax}(\{x_1, \ldots, x_N\}) = \{\frac{e^{x_1}}{Z}, \ldots, \frac{e^{x_N}}{Z}\}$$

$$Z = \sum_{j=1}^{N} e^{x_j}$$



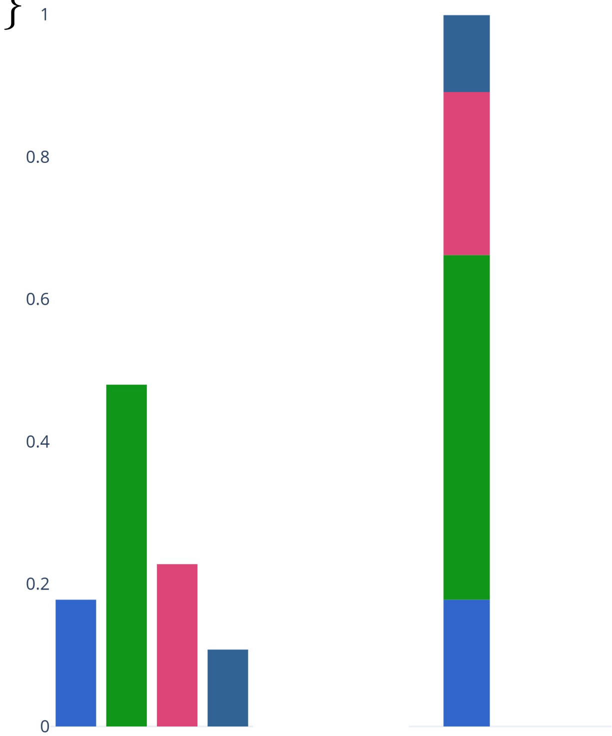


Review of Softmax Function

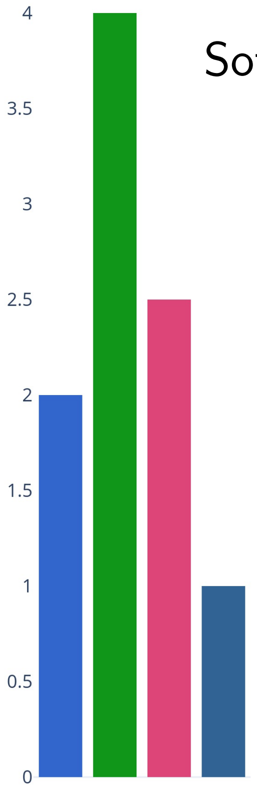


$$\text{Softmax}(\{x_1, \ldots, x_N\}) = \{\frac{e^{x_1}}{Z}, \ldots, \frac{e^{x_N}}{Z}\}$$

$$Z = \sum_{j=1}^{N} e^{x_j}$$

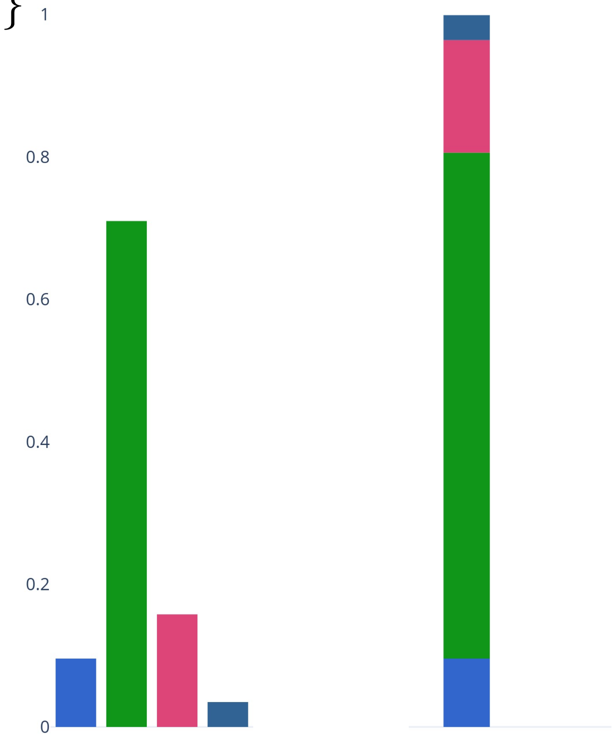


Review of Softmax Function

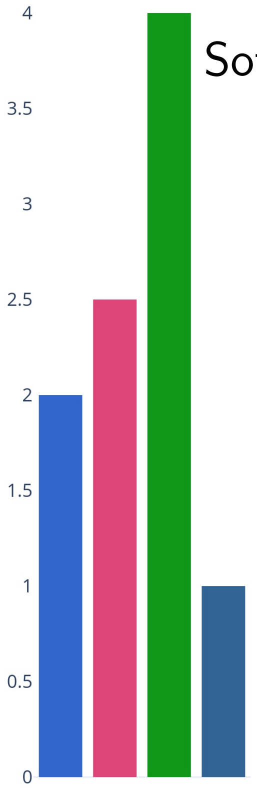


$$\text{Softmax}(\{x_1, \ldots, x_N\}) = \{\frac{e^{x_1}}{Z}, \ldots, \frac{e^{x_N}}{Z}\}$$

$$Z = \sum_{j=1}^{N} e^{x_j}$$

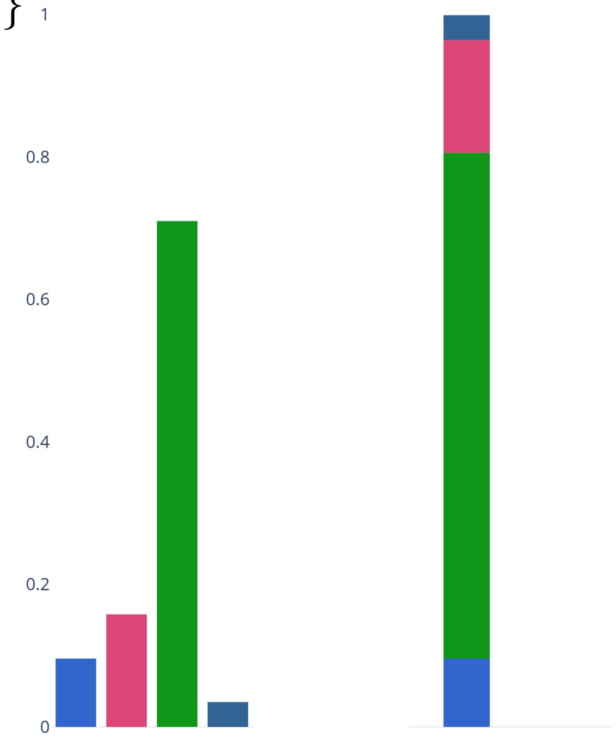


Review of Softmax Function



$$\text{Softmax}(\{x_1, \ldots, x_N\}) = \{\frac{e^{x_1}}{Z}, \ldots, \frac{e^{x_N}}{Z}\}$$

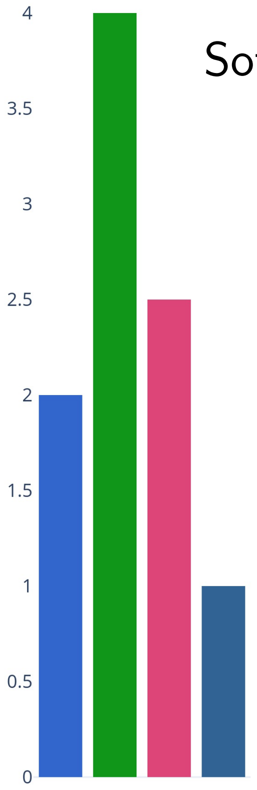
$$Z = \sum_{j=1}^{N} e^{x_j}$$

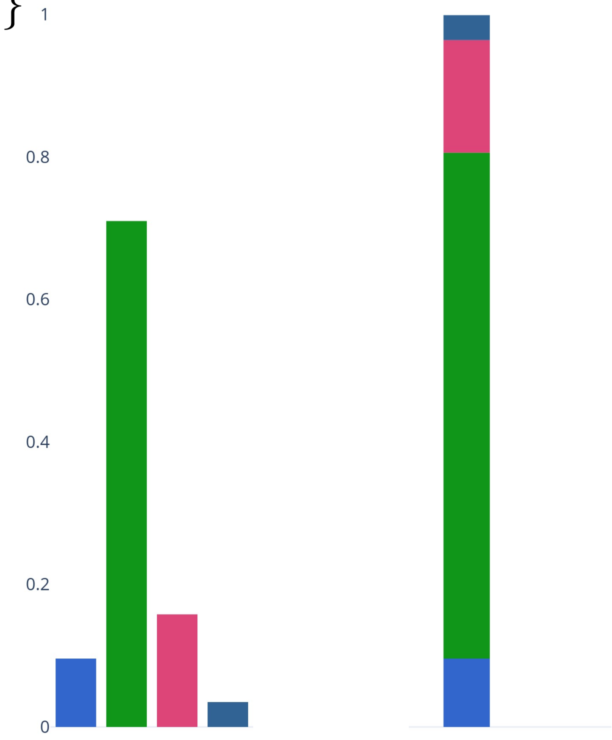


Review of Softmax Function

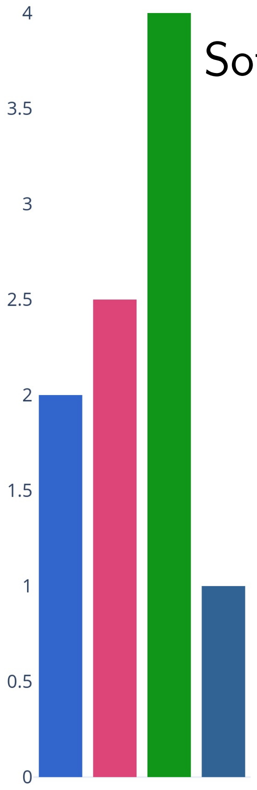
$$\text{Softmax}(\{x_1, \ldots, x_N\}) = \{\frac{e^{x_1}}{Z}, \ldots, \frac{e^{x_N}}{Z}\}$$

$$Z = \sum_{j=1}^{N} e^{x_j}$$



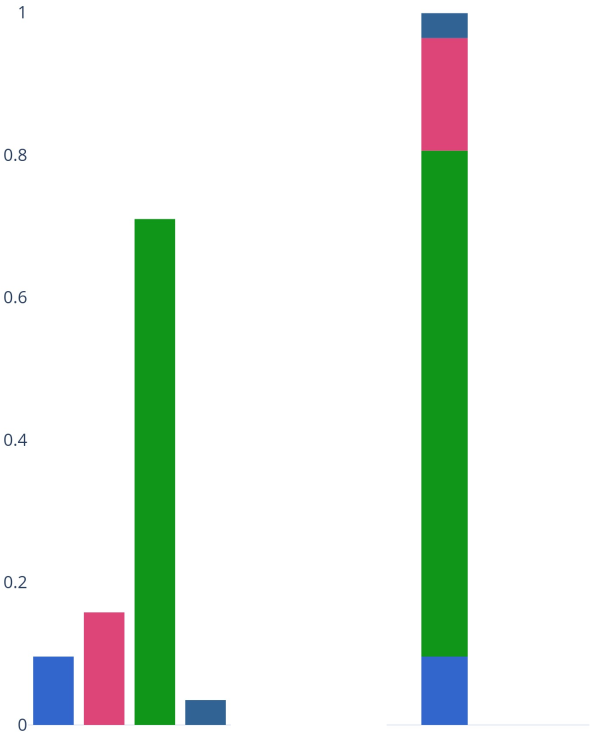


Review of Softmax Function

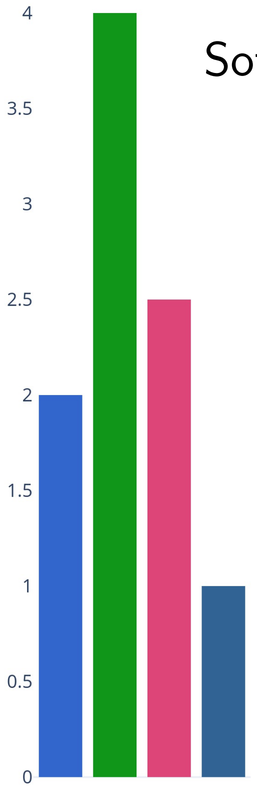


$$\text{Softmax}(\{x_1, \ldots, x_N\}) = \{\frac{e^{x_1}}{Z}, \ldots, \frac{e^{x_N}}{Z}\}$$

$$Z = \sum_{j=1}^{N} e^{x_j}$$

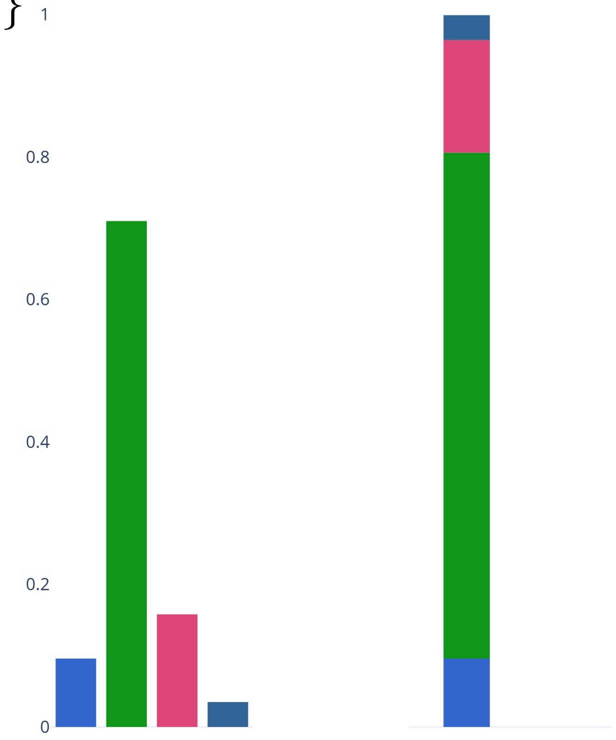


Review of Softmax Function



$$\text{Softmax}(\{x_1, \ldots, x_N\}) = \{\frac{e^{x_1}}{Z}, \ldots, \frac{e^{x_N}}{Z}\}$$

$$Z = \sum_{j=1}^{N} e^{x_j}$$



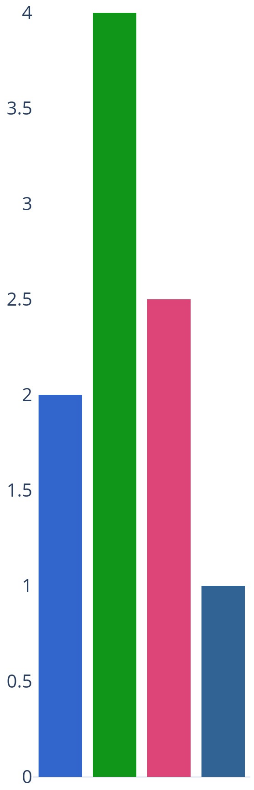
Review of Softmax Function

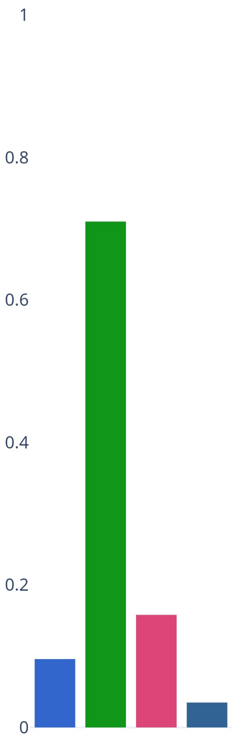
- Softmax should be really be ArgSoftmax:

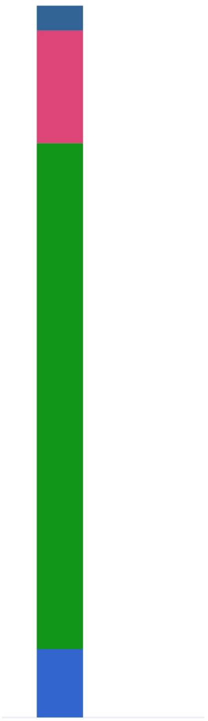
- Softmax interpolates between a distribution that selects an index uniformly at random

- And a distribution that selects the Argmax index with probability 1

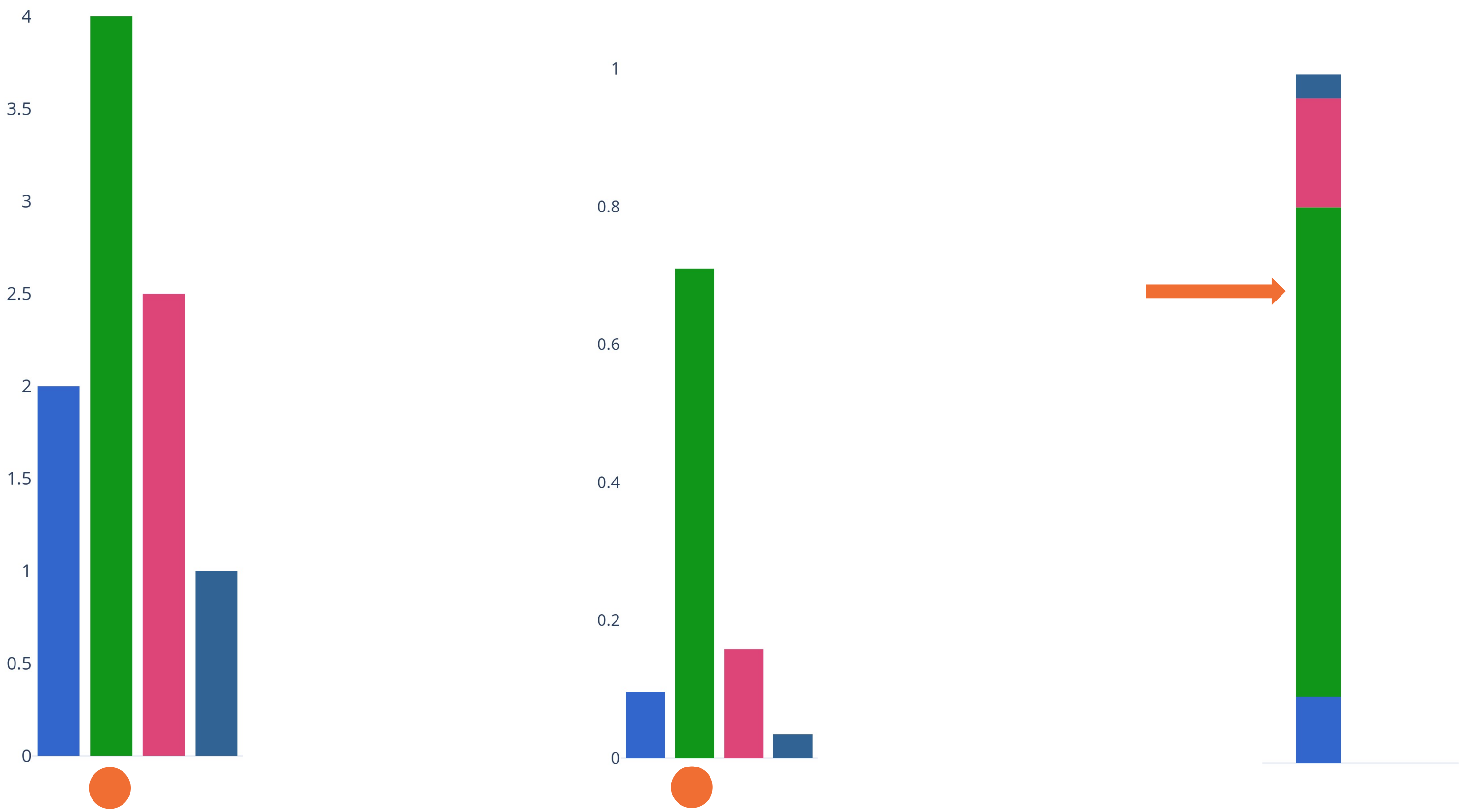
- Softmax is differentiable…



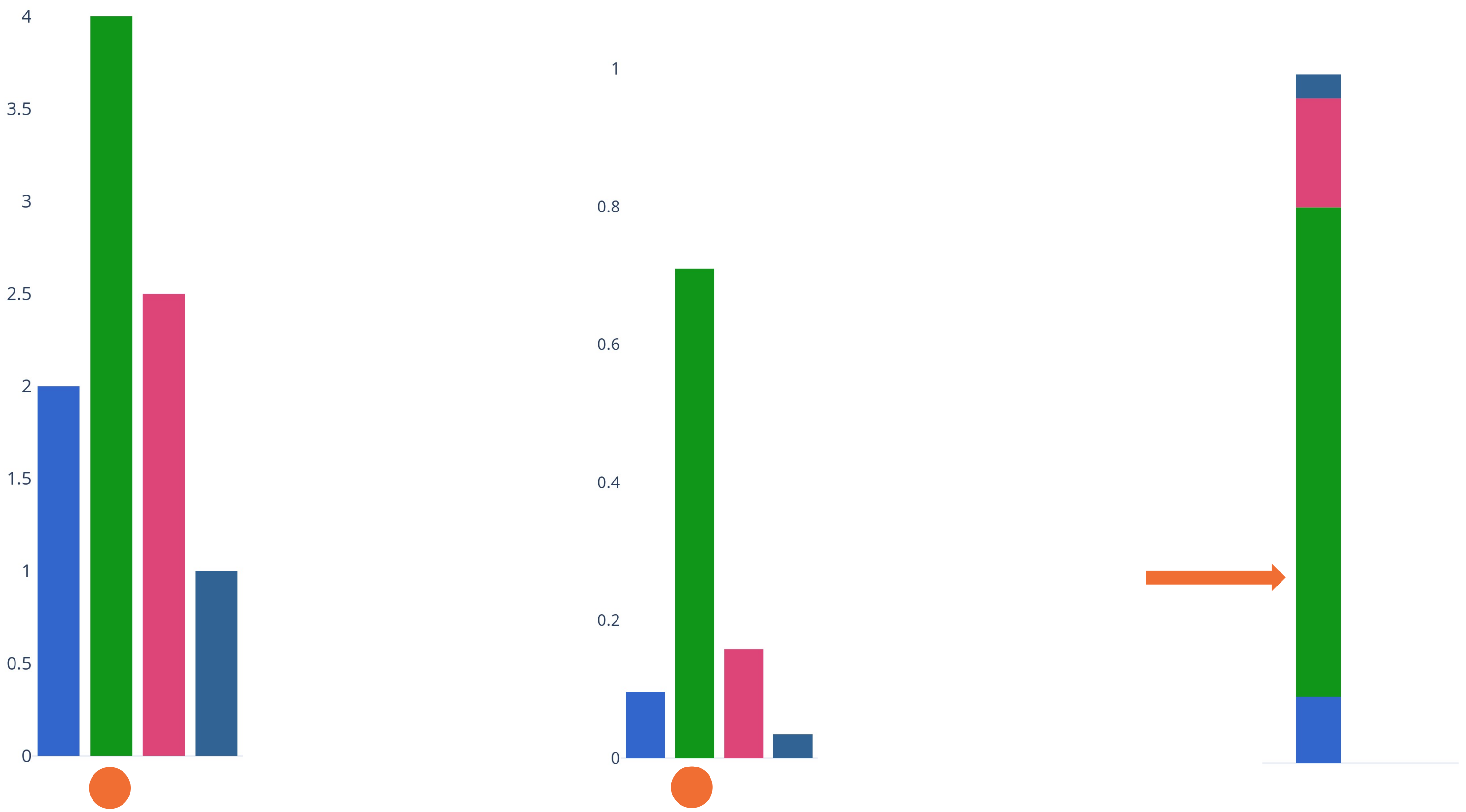




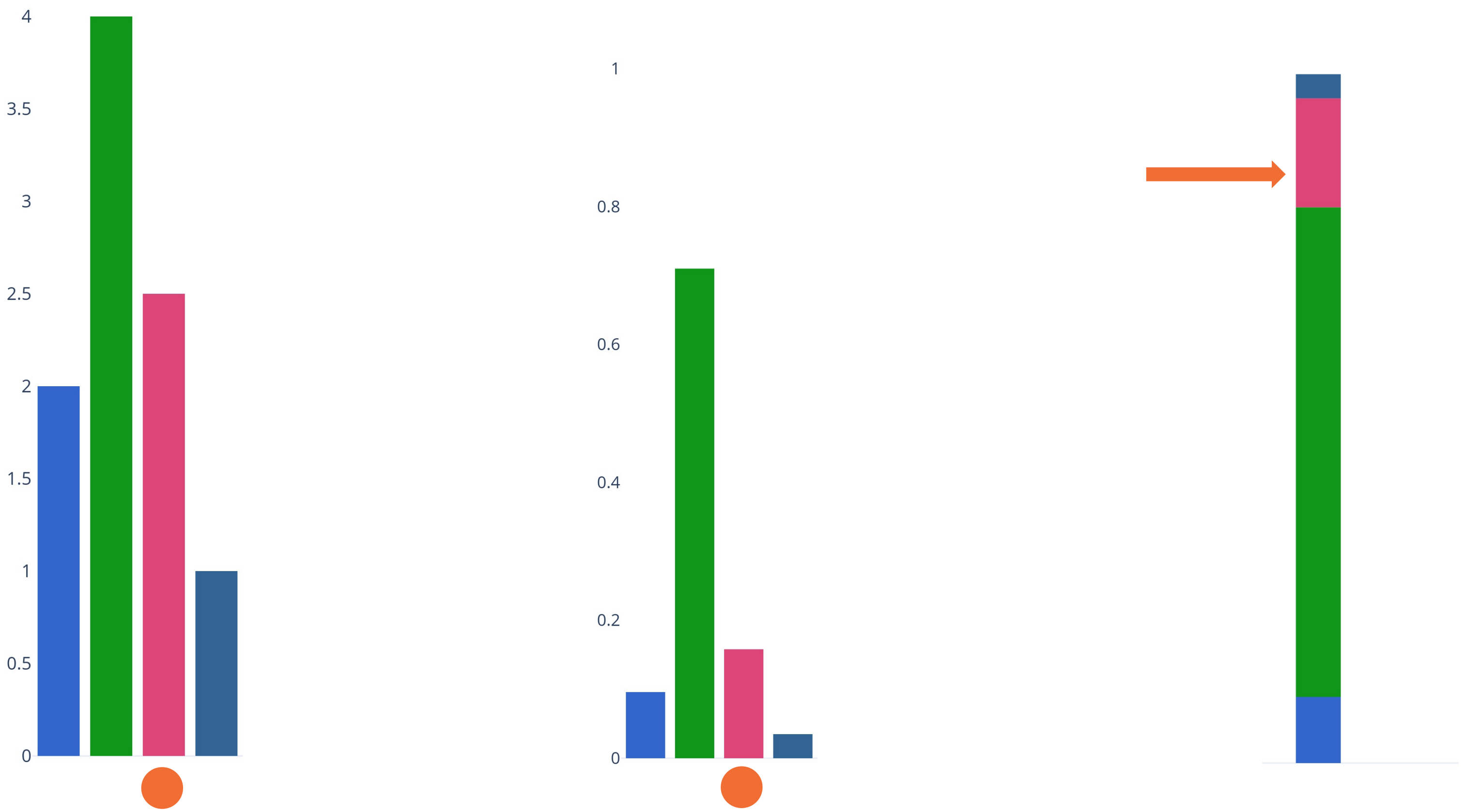
Softmax and Index Selection



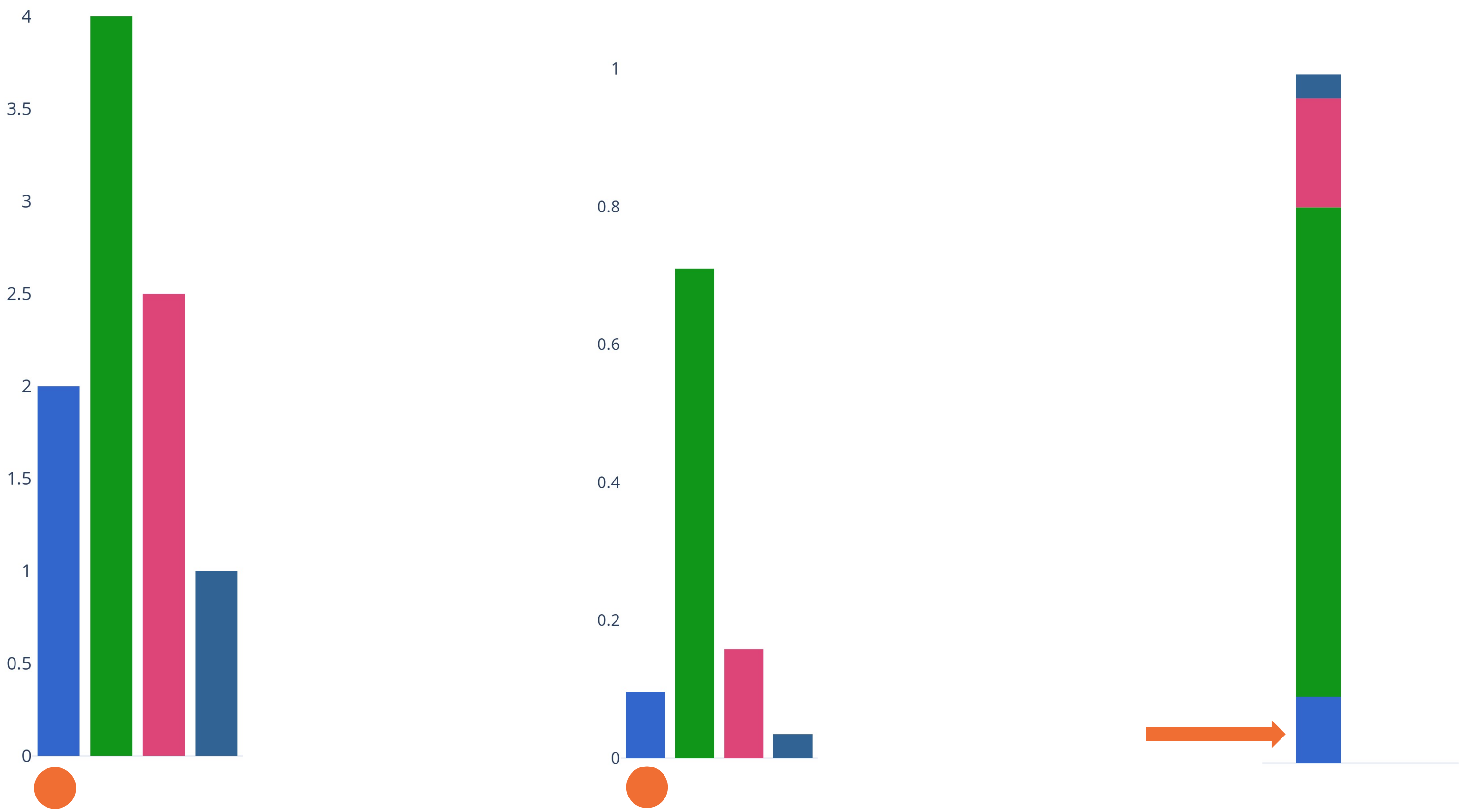
Softmax and Index Selection



Softmax and Index Selection



Softmax and Index Selection



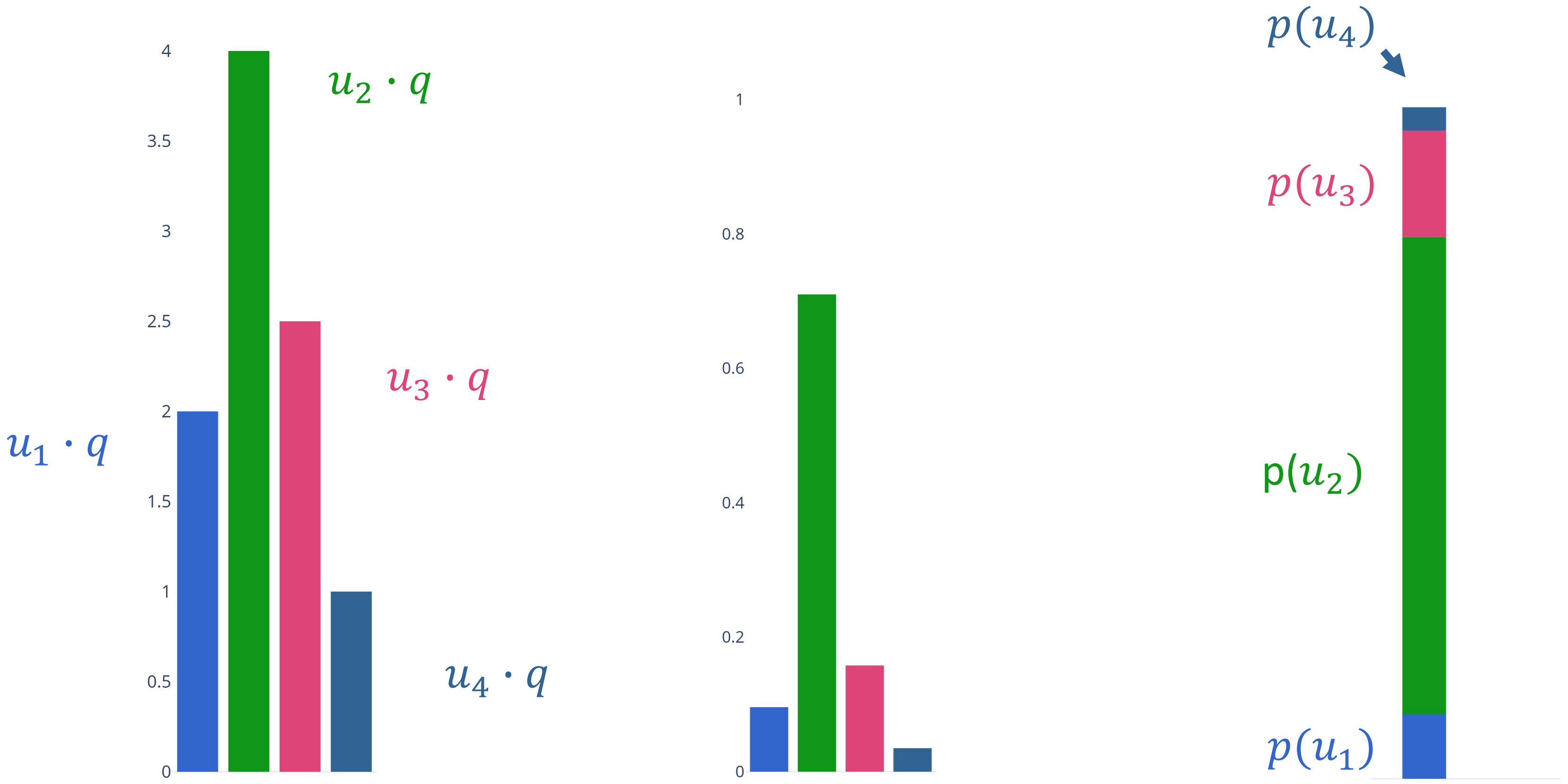
Softmax and Index Selection

- Given a set of vectors $\{u_1, ..., u_N\}$ and a "query" vector q
- We can select the most similar vector to q via $\hat{j} = \arg\max_j u_j \cdot q$
- This is the distribution $p = p_{\text{hard}}$ which has mass 0 on all indices except $\hat{j}$

Selecting a Vector from a Set

- Given a set of vectors $\{u_1, \dots, u_N\}$ and a "query" vector q
- We can select the most similar vector to q via $p = Softmax(Uq)$
- U is the vectors arranged as rows in a matrix

- Given a set of vectors $\{u_1, \ldots, u_N\}$ and a "query" vector q
- We can select the most similar vector to q via $p = Softmax(Uq)$
- This is "Softmax attention"



Softmax Attention Over Vectors

FACEBOOK AI Georgia Tech

When Softmax is applied at the final layer of a MLP:

- q is the last hidden state, $\{u_1, \dots, u_N\}$ is the embeddings of the class labels
- Samples from the distribution correspond to labelings (outputs)

In Softmax attention:

- q is an internal hidden state, $\{u_1, \dots, u_N\}$ is the embeddings of an “input” (e.g. previous layer)
- The distribution correspond to a summary of $\{u_1, \dots, u_N\}$

Attention

Attention!

What is to the left of the table?

FACEBOOK AI Georgia Tech

Attention!

What is to the left of the **table**?

FACEBOOK AI Georgia Tech

Attention!

What is to the left of the **table**?

Attention!

What is to the **left** of the table?

Attention!

What is to the left of the table? Tent

"Attention": distribution over inputs that depends on computational state and the inputs themselves.

Attention can be

- "hard", where samples are drawn from the distribution over the input,
- "soft", where the distribution is used directly as a weighted average

Currently, the most standard form of attention in neural networks is implemented with the Softmax.

Long history in computer vision, classically inspired by saccades

- [P. N. Rajesh et. al. 1996; Butko et. al 2009; Larochelle et. al 2010; Mnih et. al. 2014] (lots more)

In these works, attention is over the set of spatial locations in an image.

- given current state/history of glimpses, where and at what scale should we look next?

Alignment in machine translation: for each word in the target, get a distribution over words in the source [Brown et. al. 1993], (lots more)

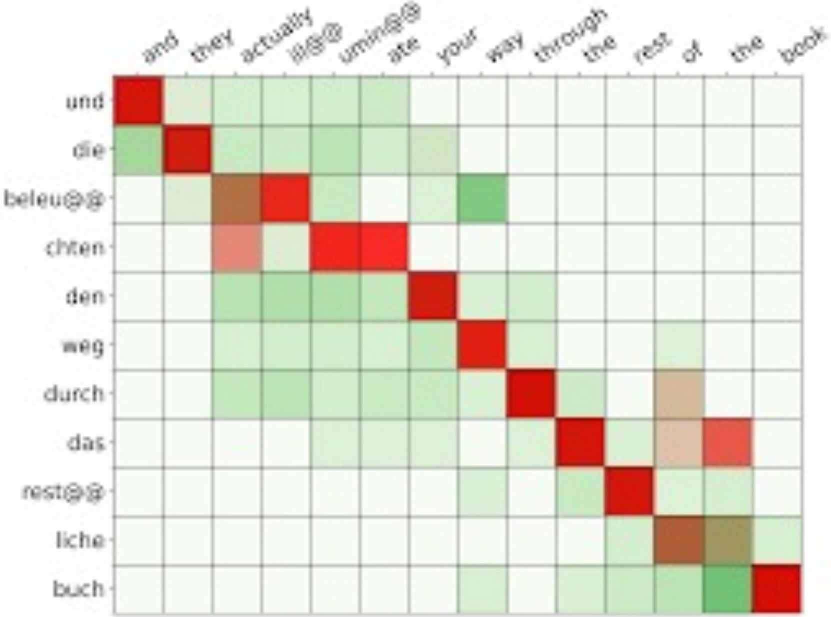
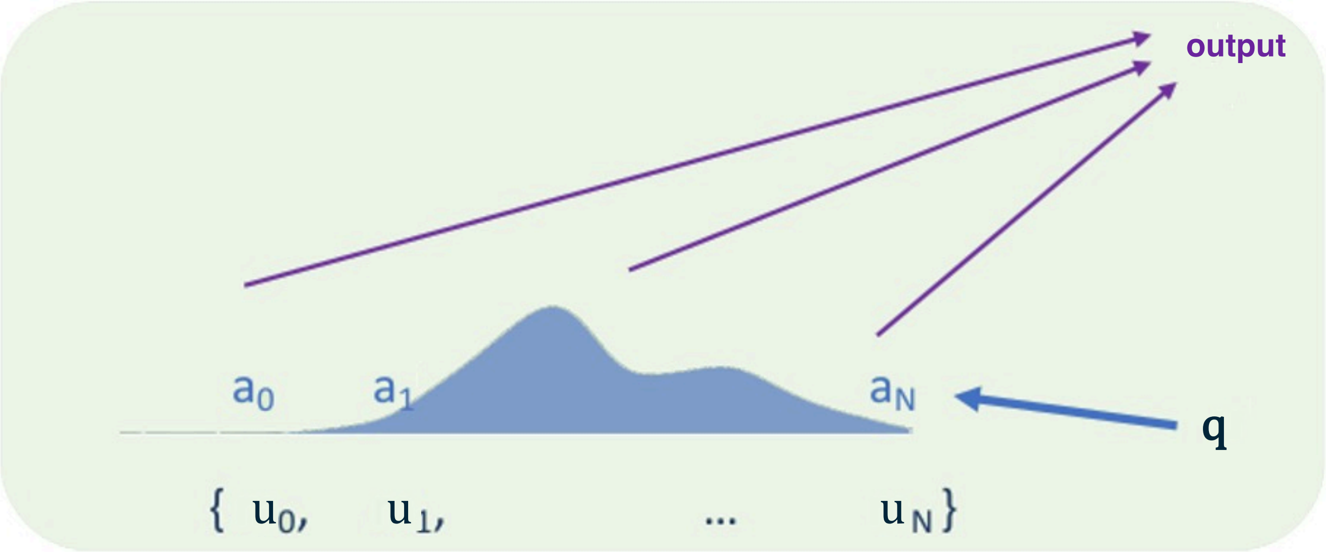


Figure from Latent Alignment and Variational Attention by Deng et. al.

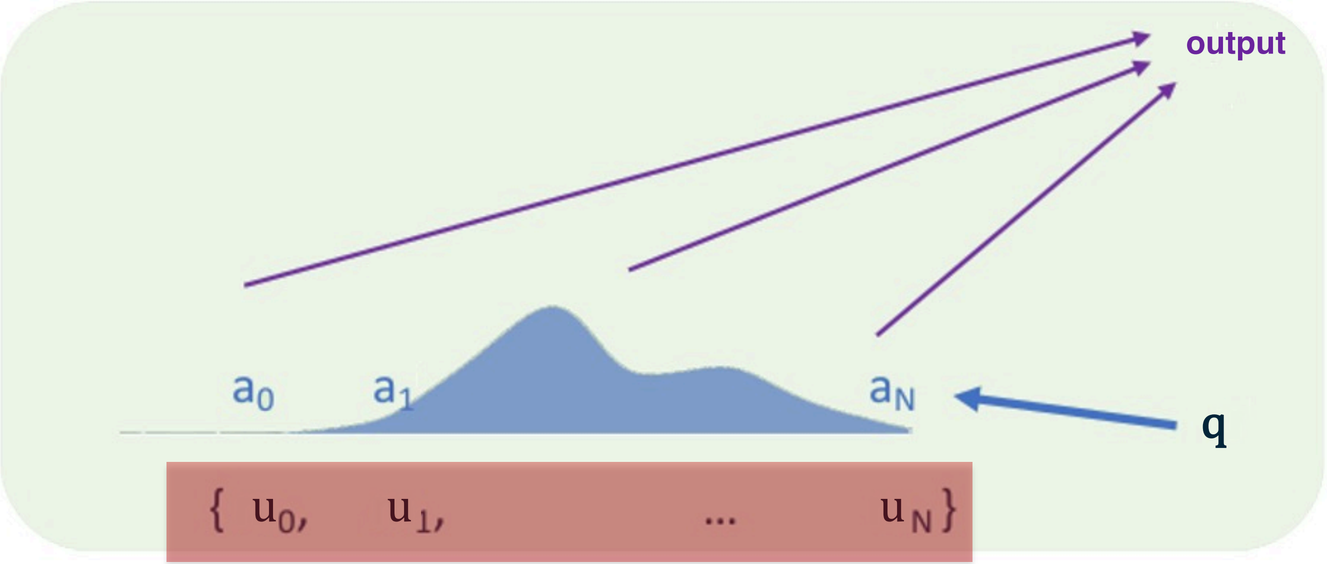
Attention as a neural network layer seems to be a surprisingly new development

- Location based, for handwriting generation: [Graves 2013]
- In machine translation: “content based”, [Bahdanau et. al. 2014], inspired by alignment



$$a_j = \frac{e^{u_j \cdot q}}{\sum_k e^{u_k \cdot q}}$$

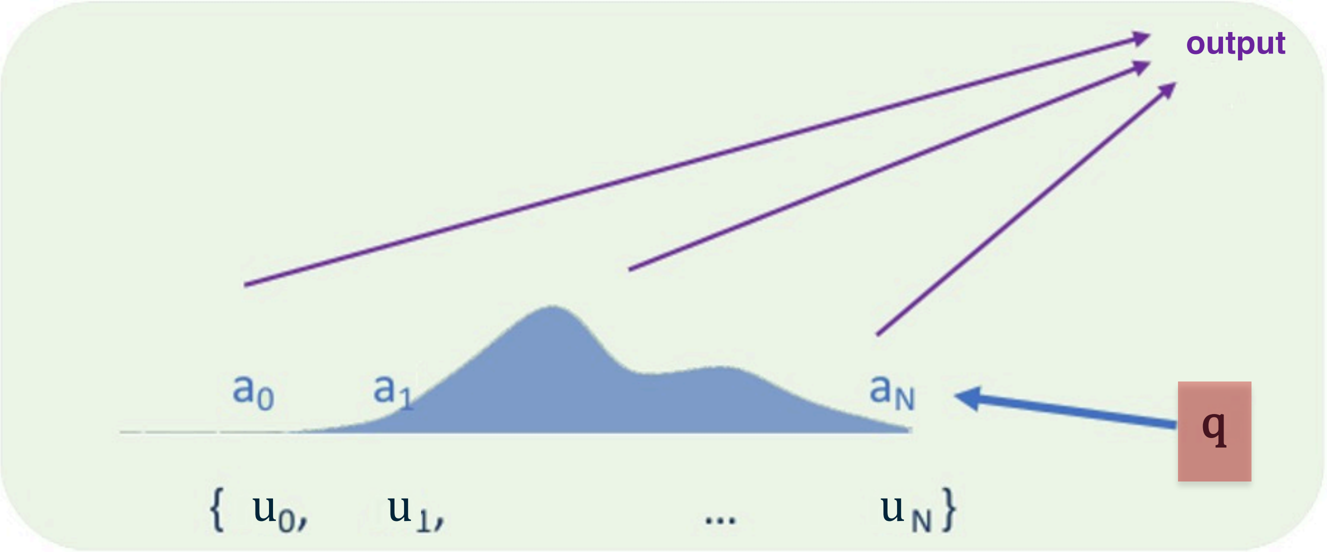
$$\text{output} = \sum_k a_k u_k$$



$$a_j = \frac{e^{u_j \cdot q}}{\sum_k e^{u_k \cdot q}}$$

$$\text{output} = \sum_k a_k u_k$$

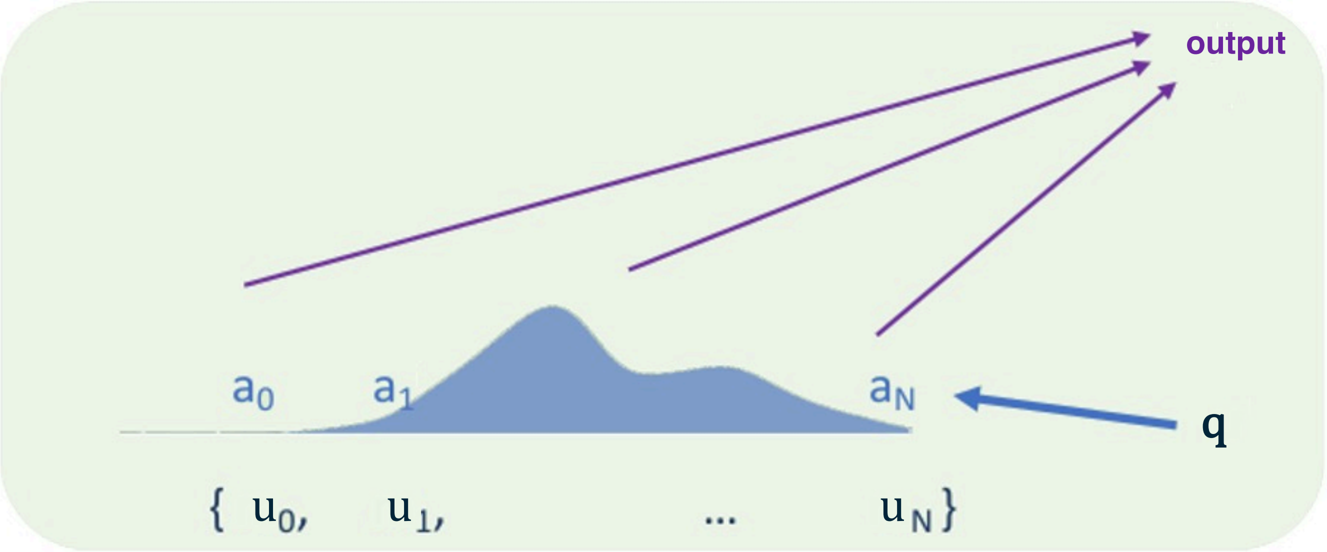
Current (Standard) Approach to (Soft) Attention



$$a_j = \frac{e^{u_j \cdot q}}{\sum_k e^{u_k \cdot q}}$$

$$\text{output} = \sum_k a_k u_k$$

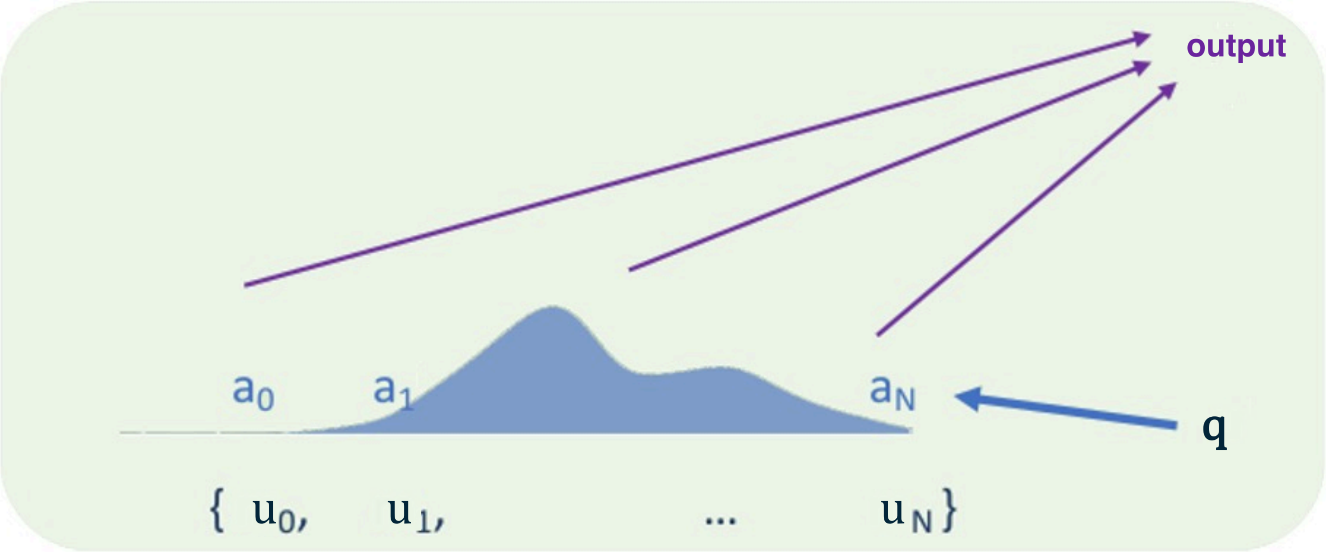
Current (Standard) Approach to (Soft) Attention



$$a_j = \frac{e^{u_j \cdot q}}{\sum_k e^{u_k \cdot q}}$$

$$\text{output} = \sum_k a_k u_k$$

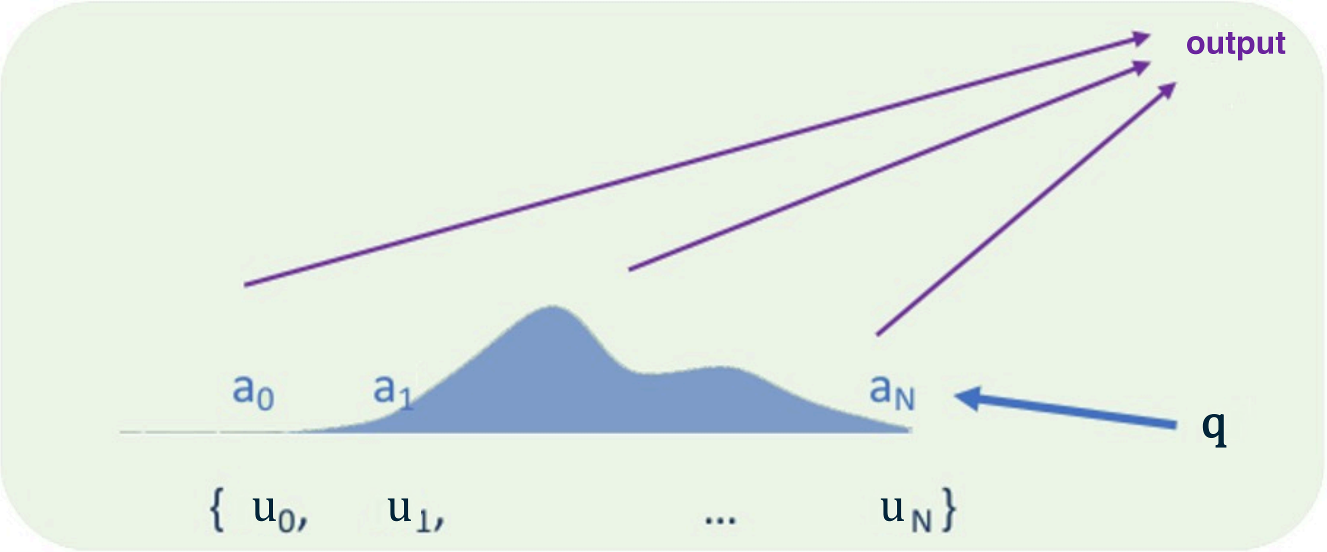
Current (Standard) Approach to (Soft) Attention



$$a_j = \frac{e^{u_j \cdot q}}{\sum_k e^{u_k \cdot q}}$$

$$\boxed{\text{output} = \sum_k a_k u_k}$$

Current (Standard) Approach to (Soft) Attention

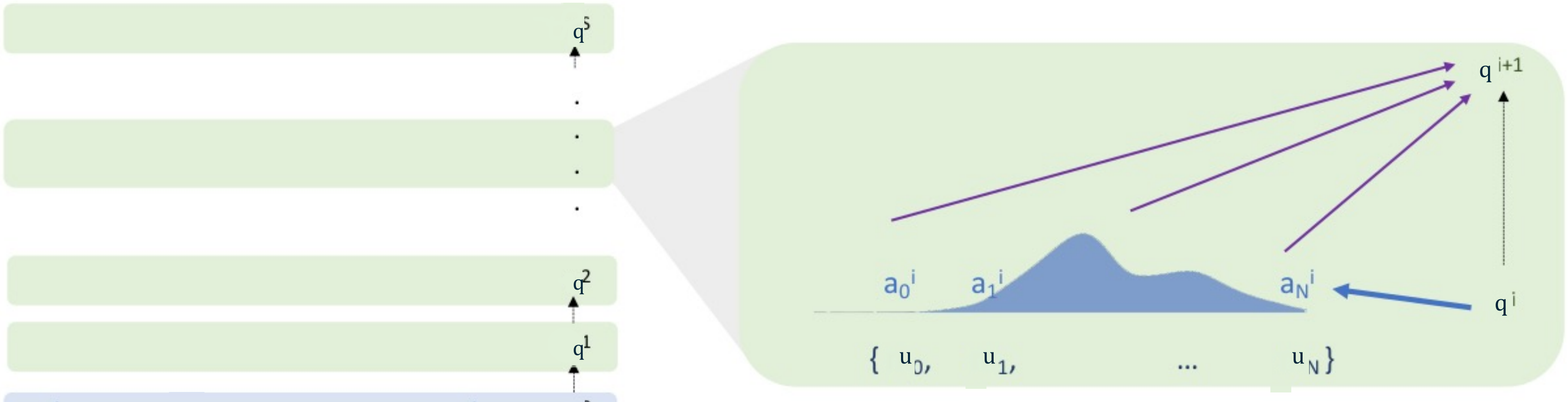


$$a_j = \frac{e^{u_j \cdot q}}{\sum_k e^{u_k \cdot q}}$$

$$\text{output} = \sum_k a_k u_k$$

Examples of controllers:

- In [Bahdanau et. al. 2014], $\mathbf{q}$ is the hidden state at a given token in an LSTM, and $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_N\}$ are the word embeddings of the last N tokens
- In memory networks [Weston et. al. 2014, Sukhbaatar et. al. 2015], the controller state q is updated by multiple layers of attention over inputs

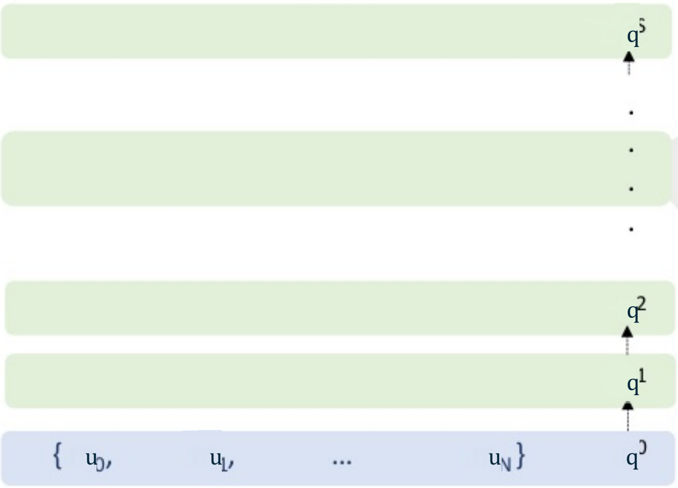


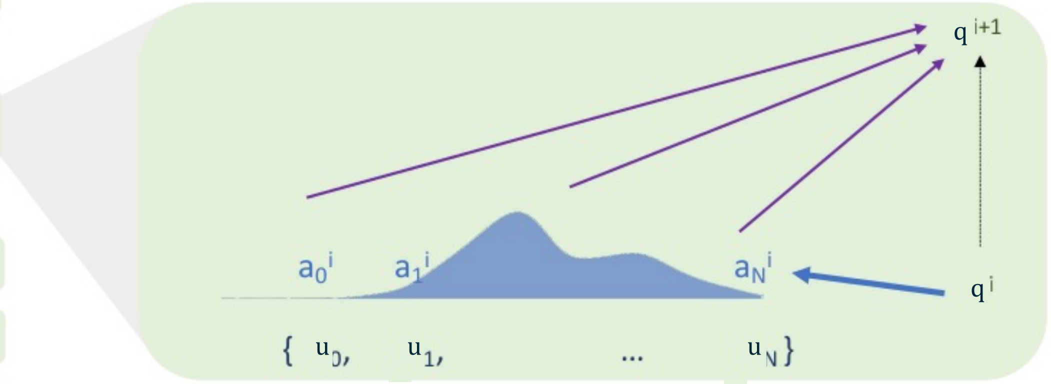
$$a^i = \mathrm{Softmax}(\{u_0 \cdot q^i, \dots, u_N \cdot q^i\})$$

$$q^{i+1} = \sum_k a_k^i u_k$$

Multi-Layer Soft Attention

Story (16: basic induction)	Support
Brian is a frog.	yes
Lily is gray.	
Brian is yellow.	yes
Julius is green.	
Greg is a frog.	yes
What color is Greg? Answer: yellow	



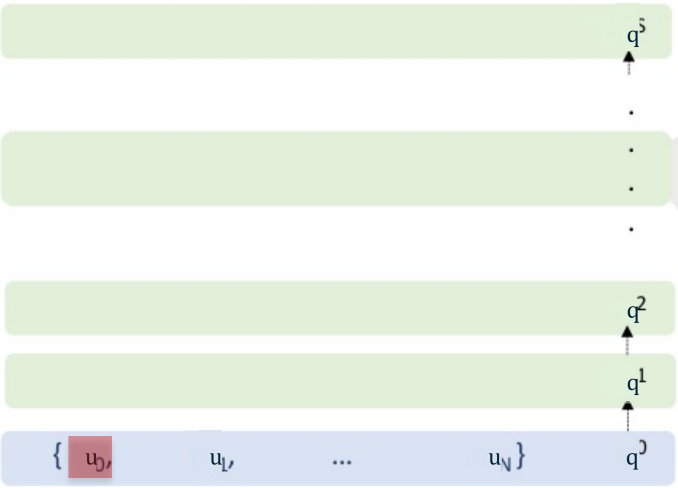


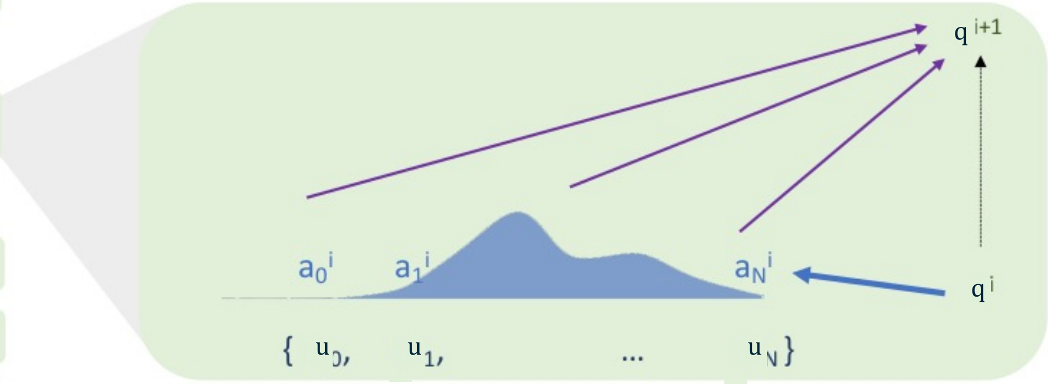
$$a^i = \mathrm{Softmax}(\{u_0 \cdot q^i, \ldots, u_N \cdot q^i\})$$

$$q^{i+1} = \sum_k a_k^i u_k$$

Multi-Layer Soft Attention

Story (16: basic induction)	Support
Brian is a frog.	yes
Lily is gray.	
Brian is yellow.	yes
Julius is green.	
Greg is a frog.	yes
What color is Greg? Answer: yellow	



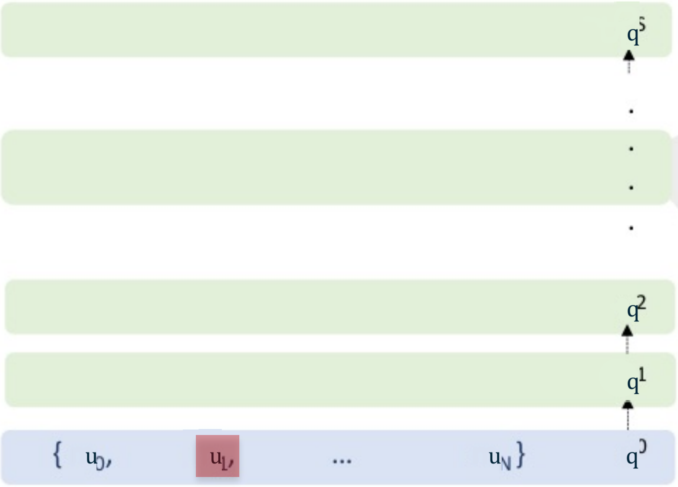


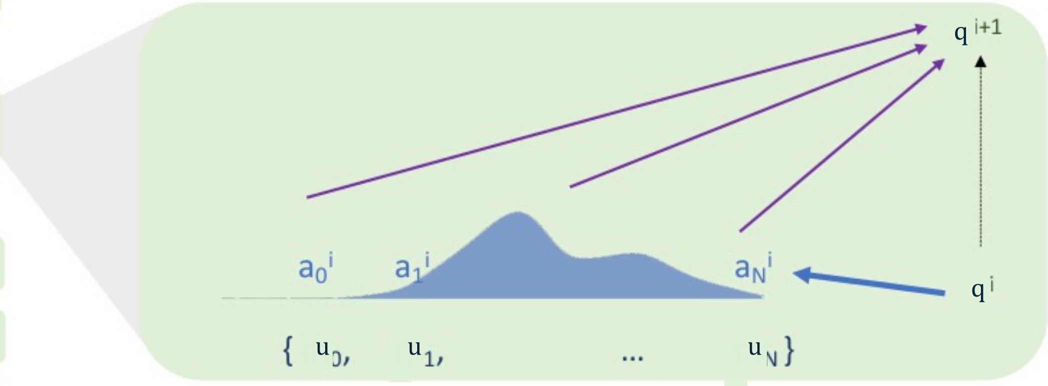
$$a^i = \mathrm{Softmax}(\{u_0 \cdot q^i, \ldots, u_N \cdot q^i\})$$

$$q^{i+1} = \sum_k a_k^i u_k$$

Multi-Layer Soft Attention

Story (16: basic induction)	Support
Brian is a frog.	yes
Lily is gray.	
Brian is yellow.	yes
Julius is green.	
Greg is a frog.	yes
What color is Greg? Answer: yellow	



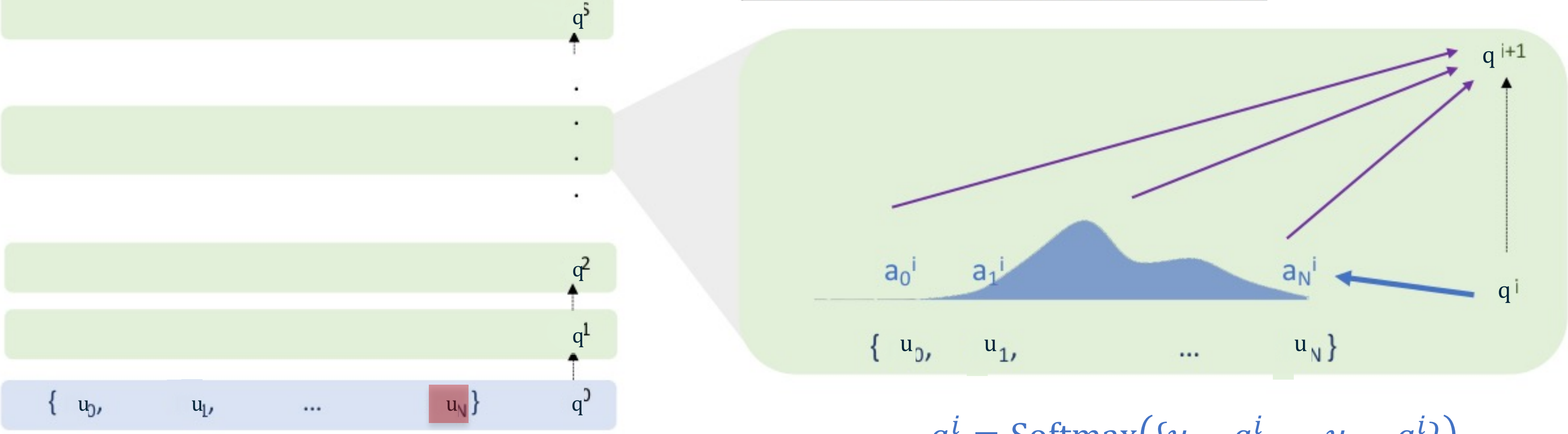


$$a^i = \text{Softmax}(\{u_0 \cdot q^i, \ldots, u_N \cdot q^i\})$$

$$q^{i+1} = \sum_k a_k^i u_k$$

Multi-Layer Soft Attention

Story (16: basic induction)	Support
Brian is a frog.	yes
Lily is gray.	
Brian is yellow.	yes
Julius is green.	
Greg is a frog.	yes
What color is Greg? Answer: yellow	

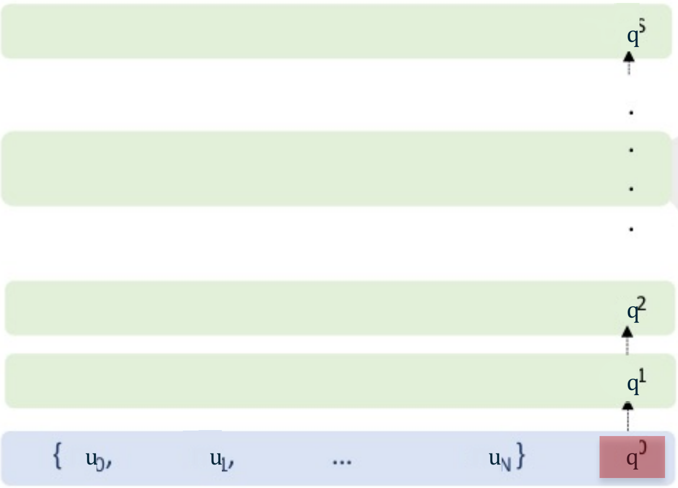


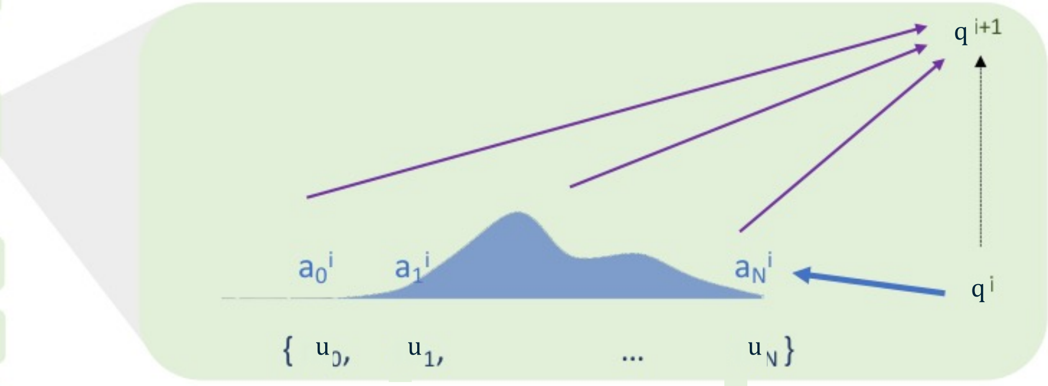
$$a^i = \text{Softmax}(\{u_0 \cdot q^i, \dots, u_N \cdot q^i\})$$

$$q^{i+1} = \sum_k a_k^i u_k$$

Multi-Layer Soft Attention

Story (16: basic induction)	Support
Brian is a frog.	yes
Lily is gray.	
Brian is yellow.	yes
Julius is green.	
Greg is a frog.	yes
What color is Greg? **Answer: yellow**	





$$a^i = \text{Softmax}(\{u_0 \cdot q^i, \ldots, u_N \cdot q^i\})$$

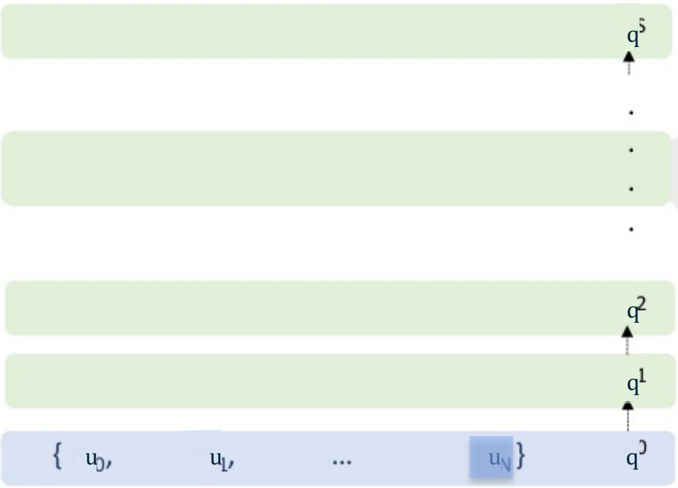
$$q^{i+1} = \sum_k a_k^i u_k$$

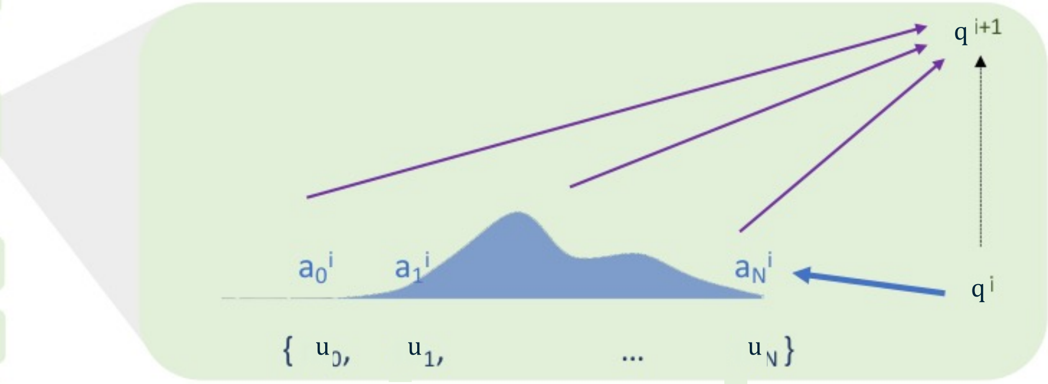
Multi-Layer Soft Attention

FACEBOOK AI

Georgia Tech

Story (16: basic induction)	Support	Hop 1
Brian is a frog.	yes	0.00
Lily is gray.		0.07
Brian is yellow.	yes	0.07
Julius is green.		0.06
Greg is a frog.	yes	0.76
What color is Greg? Answer: yellow		



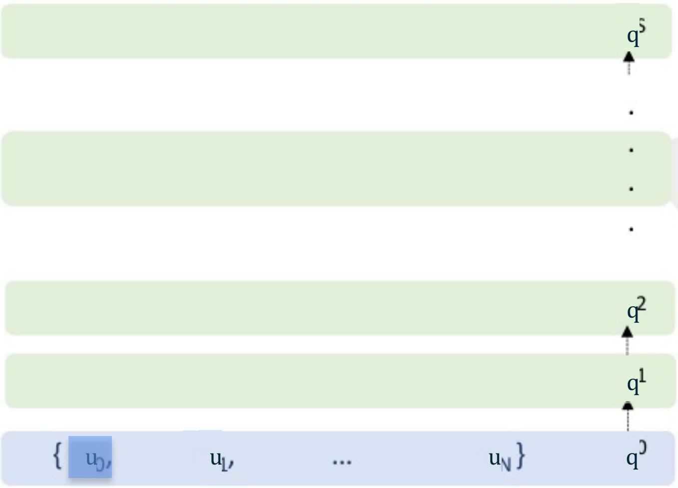


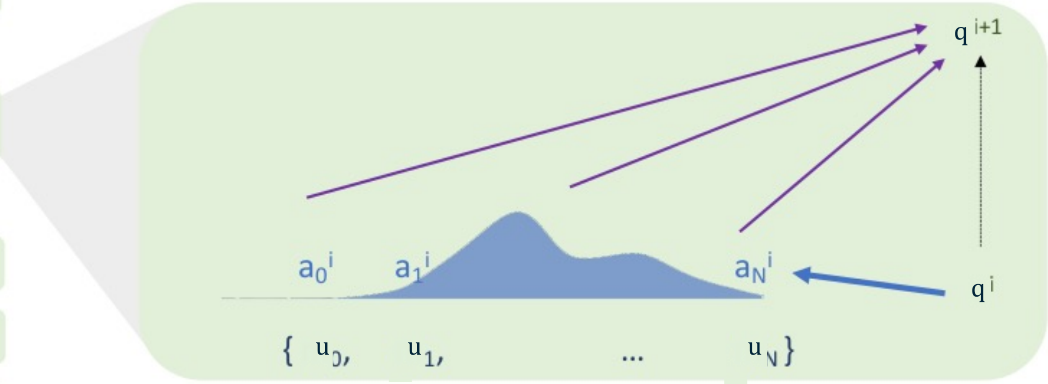
$$a^i = \mathrm{Softmax}(\{u_0 \cdot q^i, \ldots, u_N \cdot q^i\})$$

$$q^{i+1} = \sum_k a_k^i u_k$$

Multi-Layer Soft Attention

Story (16: basic induction)	Support	Hop 1	Hop 2
Brian is a frog.	yes	0.00	0.98
Lily is gray.		0.07	0.00
Brian is yellow.	yes	0.07	0.00
Julius is green.		0.06	0.00
Greg is a frog.	yes	0.76	0.02
What color is Greg? Answer: yellow			

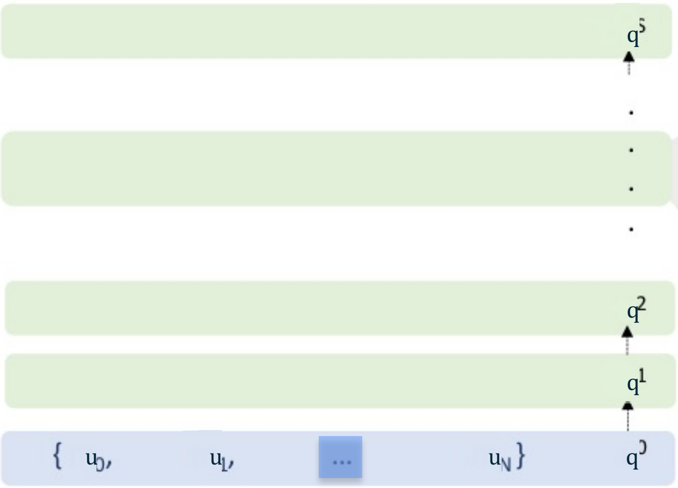




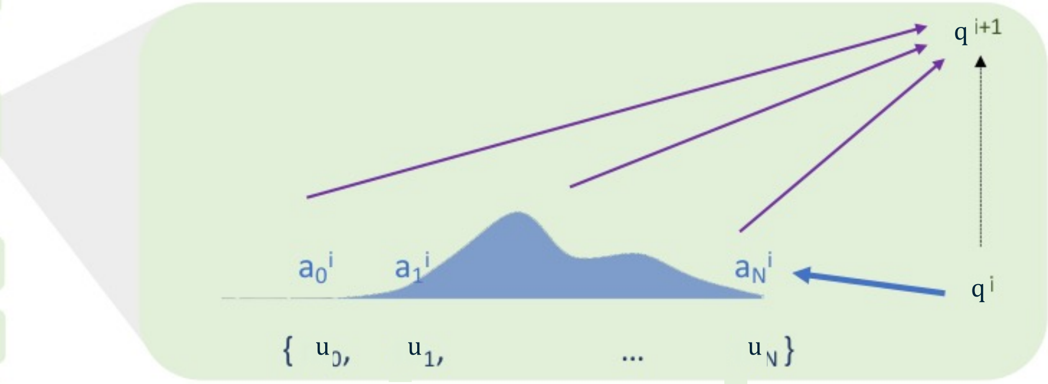
$$a^i = \text{Softmax}(\{u_0 \cdot q^i, \ldots, u_N \cdot q^i\})$$

$$q^{i+1} = \sum_k a_k^i u_k$$

Multi-Layer Soft Attention



Story (16: basic induction)	Support	Hop 1	Hop 2	Hop 3
Brian is a frog.	yes	0.00	0.98	0.00
Lily is gray.		0.07	0.00	0.00
Brian is yellow.	yes	0.07	0.00	1.00
Julius is green.		0.06	0.00	0.00
Greg is a frog.	yes	0.76	0.02	0.00
What color is Greg? Answer: yellow				



$$a^i = \mathrm{Softmax}(\{u_0 \cdot q^i, \ldots, u_N \cdot q^i\})$$

$$q^{i+1} = \sum_k a_k^i u_k$$

Multi-Layer Soft Attention

If inputs have an underlying geometry, can include geometric information in the weighted “bags”

Important example: for sequential data, use position encoding, giving the “location in the sequence of that input

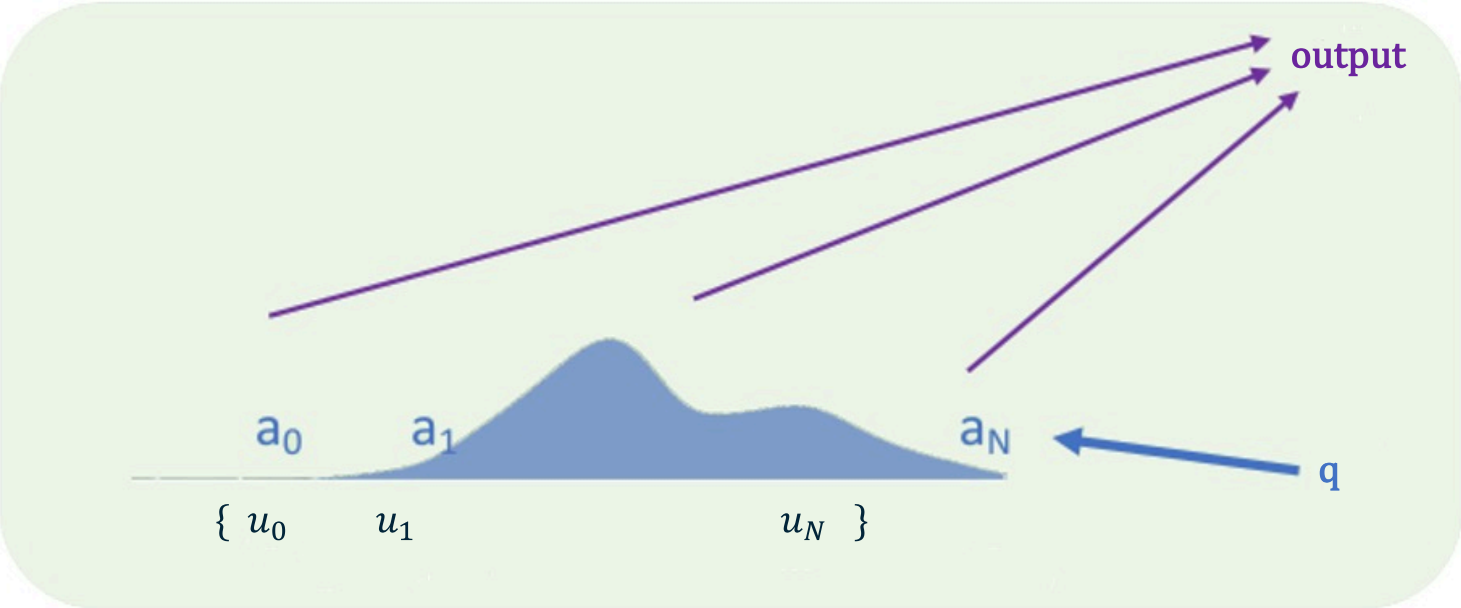
- For each input $\mathbf{m}_j$ add to it a vector $\mathbf{l}(j)$
- $\mathbf{l}(j)$ can be fixed during training or learned
- For example, a sinusoid of varying frequency in each coordinate

Transformers

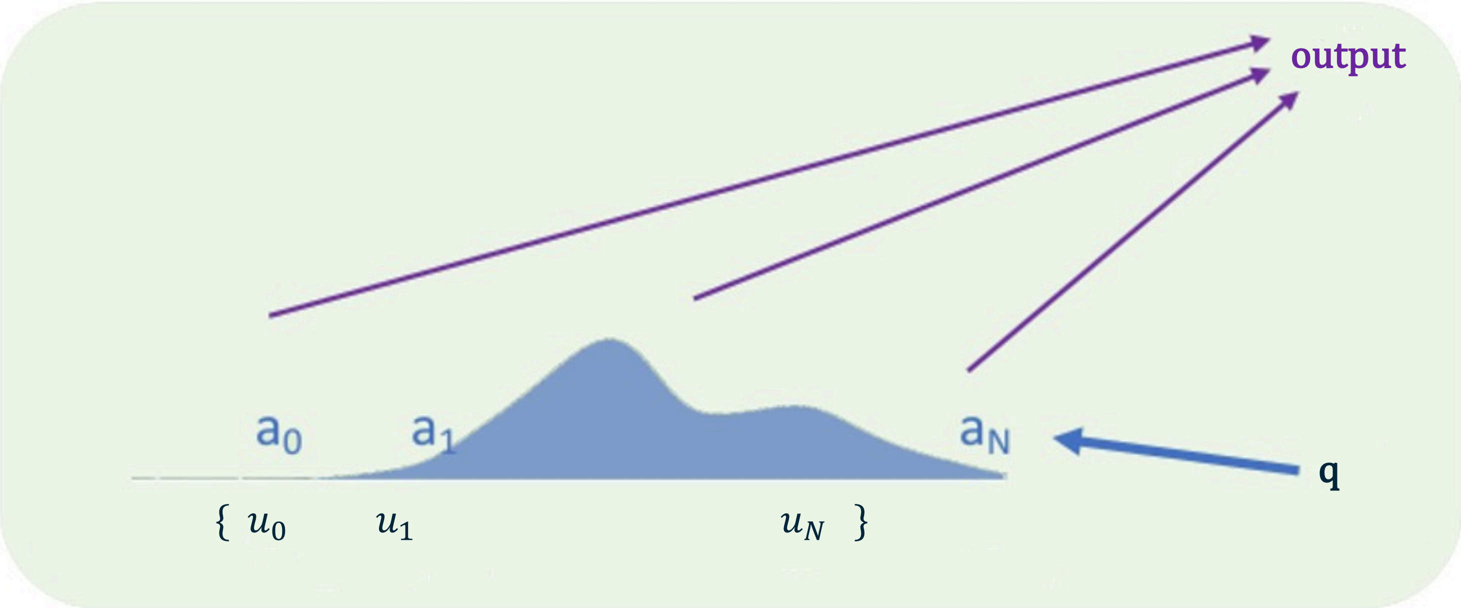
Transformer [Vaswani et. al. 2017] is a multi-layer attention model that is currently state of the art in most language tasks (and in many other things!)

Has superior performance compared to previous attention-based architectures via

- Multi-query hidden-state propagation ("self-attention")
- Multi-head attention
- Residual connections, LayerNorm



$$\mathbf{a} = \text{Softmax}(\{u_0 \cdot q, \ldots, u_N \cdot q\})$$
$$\text{output} = \sum_k a_k u_k$$

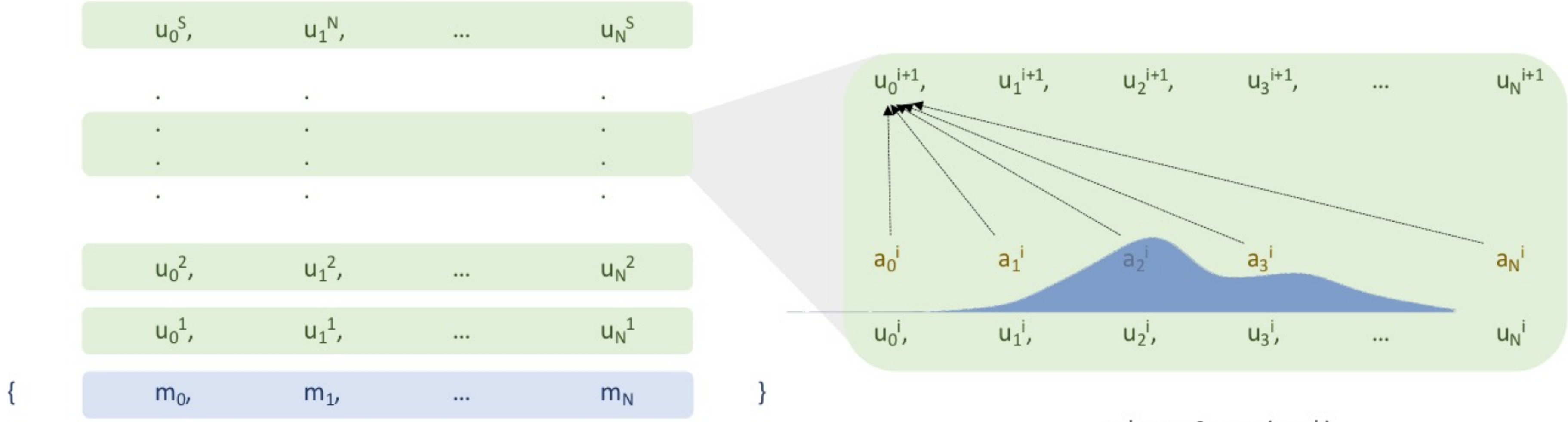


$$\mathbf{a} = \mathrm{Softmax}(\{u_0 \cdot q, \ldots, u_N \cdot q\})$$
$$\mathrm{output} = \sum_k a_k v_k$$

Transformer [Vaswani et. al. 2017] is a multi-layer attention model that is currently state of the art in most language tasks (and in many other things!)

Has superior performance compared to previous attention-based architectures via

- **Multi-query hidden-state propagation ("self-attention")**
- Multi-head attention
- Residual connections, LayerNorm



$a_j^i = \text{Softmax}(U u_j^i)$

all u_j^{i+1} are updated $u_j^{i+1} \leftarrow \boldsymbol{\Sigma}_k\, a_{kj}^i\, u_k^i$

Self-Attention

Transformer [Vaswani et. al. 2017] is a multi-layer attention model that is currently state of the art in most language tasks (and in many other things!)

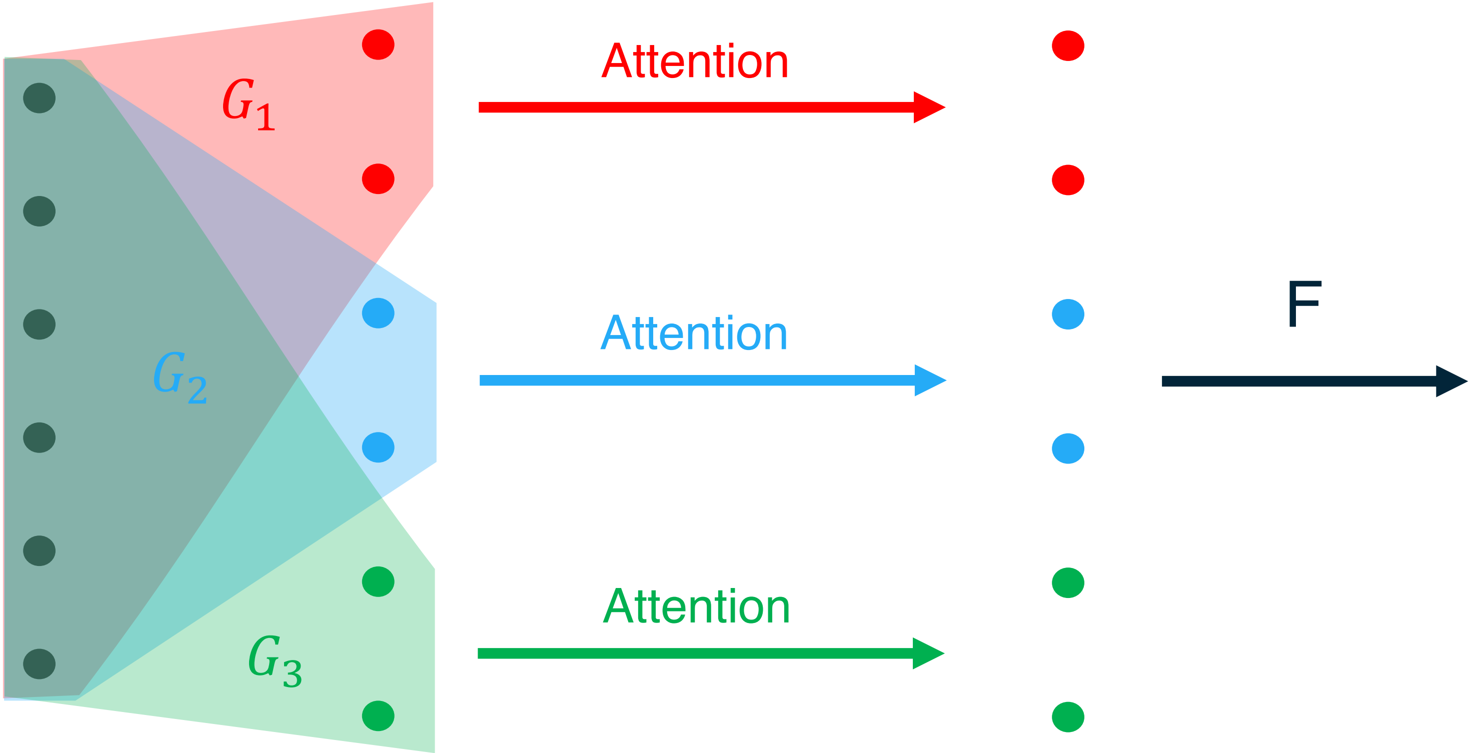
Has superior performance compared to previous attention-based architectures via

- Multi-query hidden-state propagation ("self-attention")
- **Multi-head attention**
- Residual connections, LayerNorm

Enter the Transformer

- Multi-head attention combines multiple attention ‘heads’ being trained in the same way on the same data - but with different weight matrices, and yielding different values

- Each of the L attention heads yields values for each token - these values are then multiplied by trained parameters and added



G are projections, F is fully connected. Dots are in feature dim.

Multi-Head attention

	Single Head	Multi-Head
Layer Output	$u_j \leftarrow \sum_k a_{jk} u_k$	$u_j \leftarrow F\left(\begin{bmatrix} \sum_k a^1_{jk} G_1(u_k) \\ \sum_k a^2_{jk} G_2(u_k) \\ \vdots \\ \sum_k a^L_{jk} G_L(u_k) \end{bmatrix}\right)$
Attn. Weights	$a_{jk} = \frac{e^{u_j^T u_k}}{\sum_s e^{u_j^T u_s}}$	$a^L_{jk} = \frac{e^{u_j \cdot G_L(u_k)}}{\sum_s e^{u_j \cdot G_L(u_s)}},$

Multi-Head attention

Transformer [Vaswani et. al. 2017] is a multi-layer attention model that is currently state of the art in most language tasks (and in many other things!)

Has superior performance compared to previous attention-based architectures via

- Multi-query hidden-state propagation (“self-attention”)
- Multi-head attention
- **Residual connections, LayerNorm**

While Transformers operate on *sets* of vectors (or graphs), they were introduced in the setting of text.

Some standard specializations of Transformers for text:

- Position encodings depending on the location of a token in the text
- For language models: “causal attention”
- Training code outputs a prediction at each token simultaneously (and takes a gradient at teach token simultaneously)