Deep Learning Architectures for Sequence Processing

May 18, 2020

ddebarr@uw.edu

http://cross-entropy.net/ML530/Architectures_for_Sequences.pdf

Speech and Language Processing 3rd ed draft

Chapter 9: Deep Learning Architectures for Sequence Processing

https://web.stanford.edu/~jurafsky/slp3/

1. Language Models “Revisited”
2. Recurrent Neural Networks
3. Managing Context in RNNs: LSTMs and GRUs
4. Self-Attention Networks: Transformers
5. Potential Harms from Language Models

Language Models

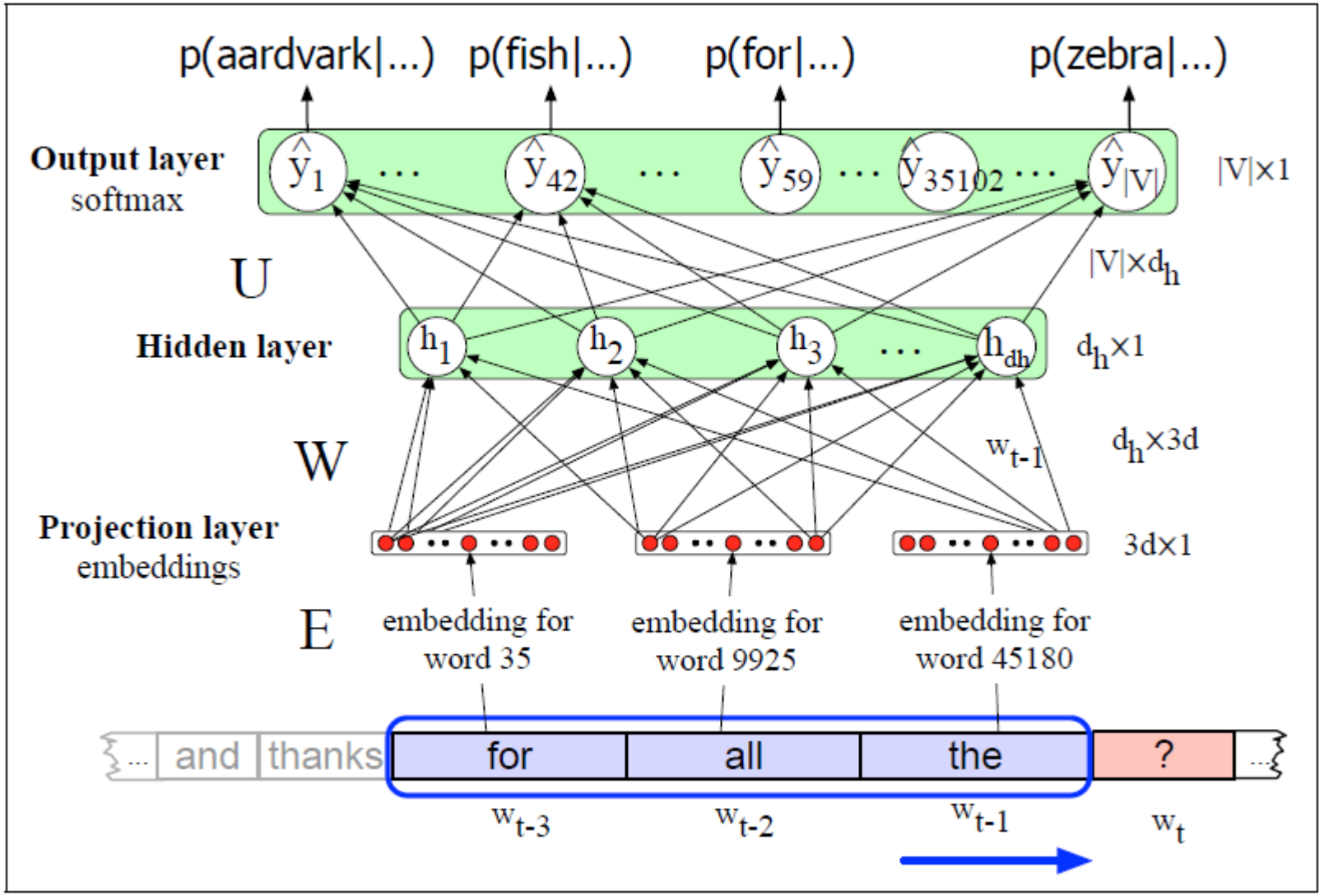
- Can be used to …
  - evaluate the probability of observing a string for a language
  - generate text
- Example: $P(w_{1:n}) = \prod_{i=1}^{n} P(w_i|w_{<i})$

  P(Thanks for all the fish) = P(Thanks) * P(for | Thanks) * P(all | Thanks for) * P(the | Thanks for all) * P(fish | Thanks for all the)
- Markov assumption: $P(w_n|w_{1:n-1}) \approx P(w_n|w_{(n-N+1):(n-1)})$

  P(fish | Thanks for all the) ≈ P(fish | for all the)
- Autoregression:

  Previous values in the sequence used to predict the next value

Language Model Example: Embedding + 2 Dense Layers



Using a sliding window for the last 3 words to predict the next word

Note:
Instead of X * W, where each row of X is an input sequence and each column of W is a neuron, this author transposes the X and W matrices for W * X

We're going to assume that hidden layer has an activation function ☺

Perplexity of a Language Model

- Geometric mean of inverted predicted probabilities
- Measures how well a model predicts words
  - 1 is perfect
  - vocab_size is terrible: achieved by always predicting P(word) = 1/vocab_size
- Compare language models using the *same* vocabulary

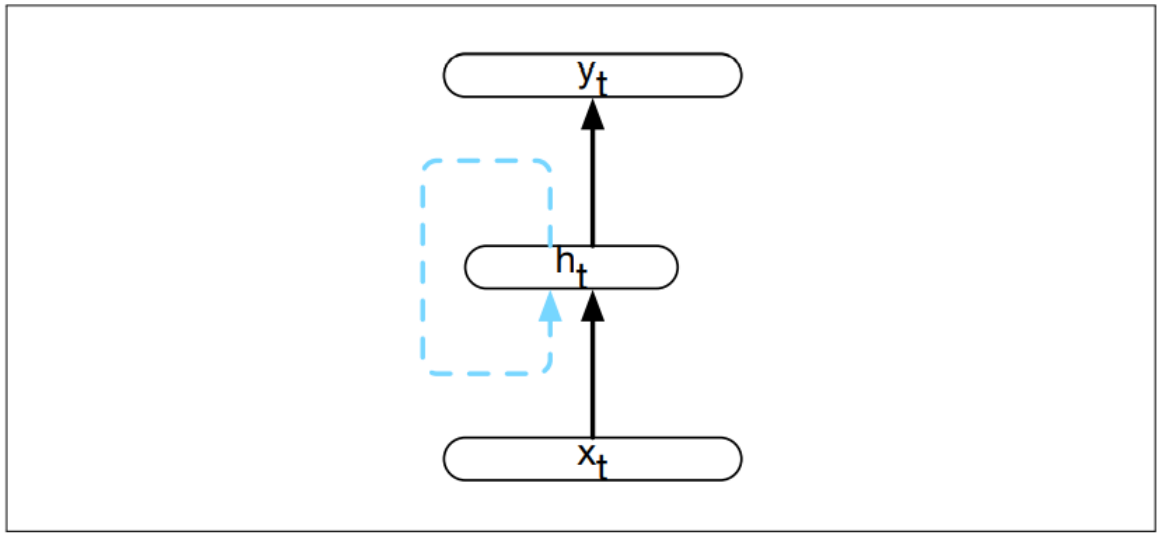
$$\left(\prod_{i=1}^{N}\frac{1}{p_i}\right)^{\frac{1}{N}} = exp\left(log\left(\left(\prod_{i=1}^{N}\frac{1}{p_i}\right)^{\frac{1}{N}}\right)\right) = exp\left(\frac{1}{N}log\left(\prod_{i=1}^{N}\frac{1}{p_i}\right)\right) = exp\left(\frac{1}{N}\sum_{i=1}^{N}log\left(\frac{1}{p_i}\right)\right)$$

$$= exp\left(\frac{1}{N}\sum_{i=1}^{N}\big(log(1) - log(p_i)\big)\right) = exp\left(\frac{1}{N}\sum_{i=1}^{N}\big(0 - log(p_i)\big)\right) = exp\left(\frac{1}{N}\sum_{i=1}^{N}\big(-log(p_i)\big)\right)$$

$$= exp\left(\frac{-\sum_{i=1}^{N}\big(log(p_i)\big)}{N}\right)$$

Recurrent Neural Network Cell

- "Elman Network" or "Simple Recurrent Network (SRN)"
- At each time step, we're computing features for the new input and adding it to our memory
  - x_i is often a word embedding
  - g() is typically tanh()
  - f() is softmax() for LM



function FORWARDRNN(x, *network*) **returns** output sequence y

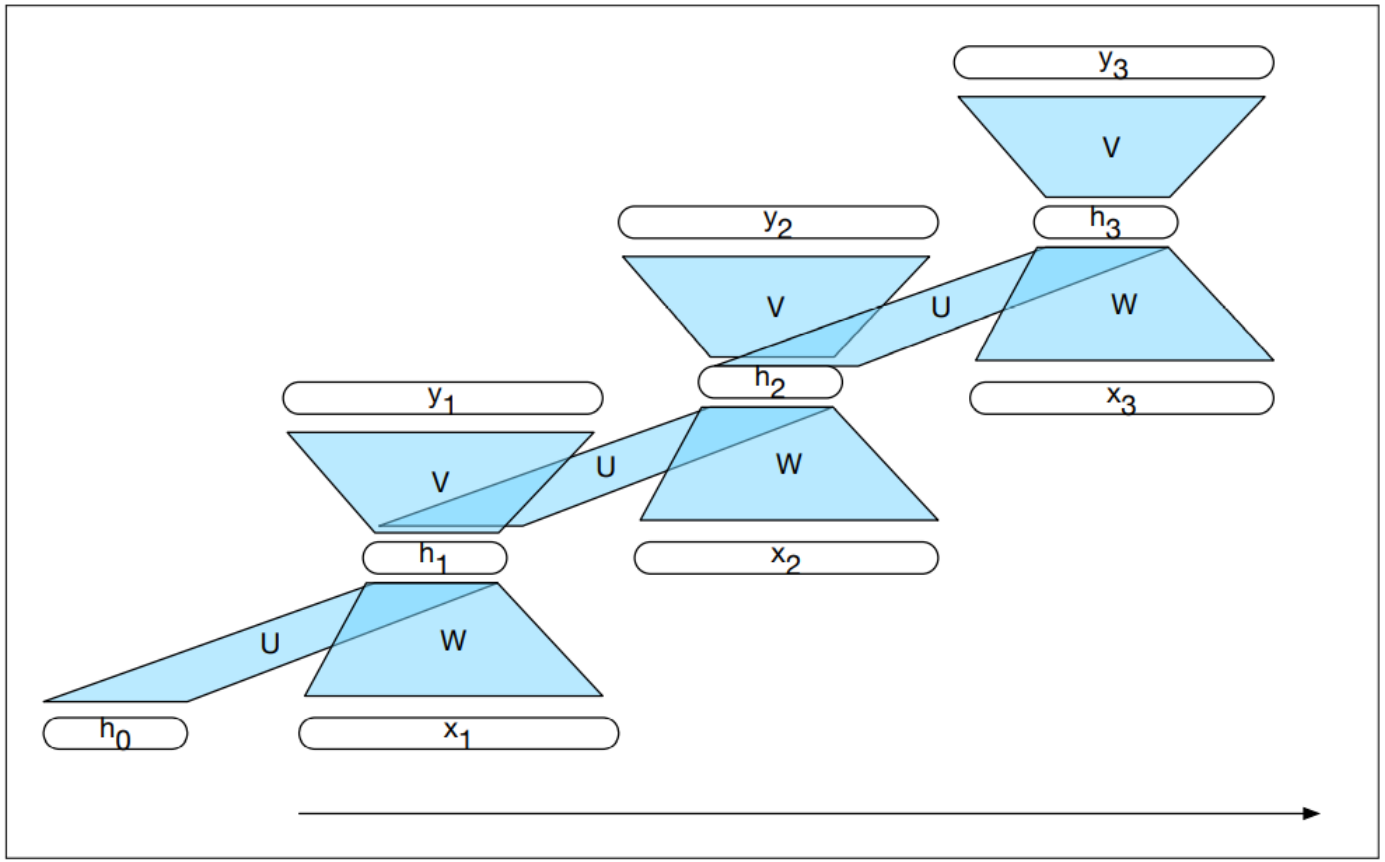
$h_0 \leftarrow 0$
for $i \leftarrow 1$ **to** LENGTH(x) **do**
  $h_i \leftarrow g(U\ h_{i-1} + W\ x_i)$
  $y_i \leftarrow f(V\ h_i)$
return y

Unrolled Across Timesteps

- Same U, W, and V matrices used across timesteps
- U and W used to derive features for the recurrent cell
- V is for the output layer
- Weight tying: E=V is possible
  - Improved perplexity
  - Reduced parameters

$$P(w_{t+1} = i | w_{1:t}) = y_t^i$$

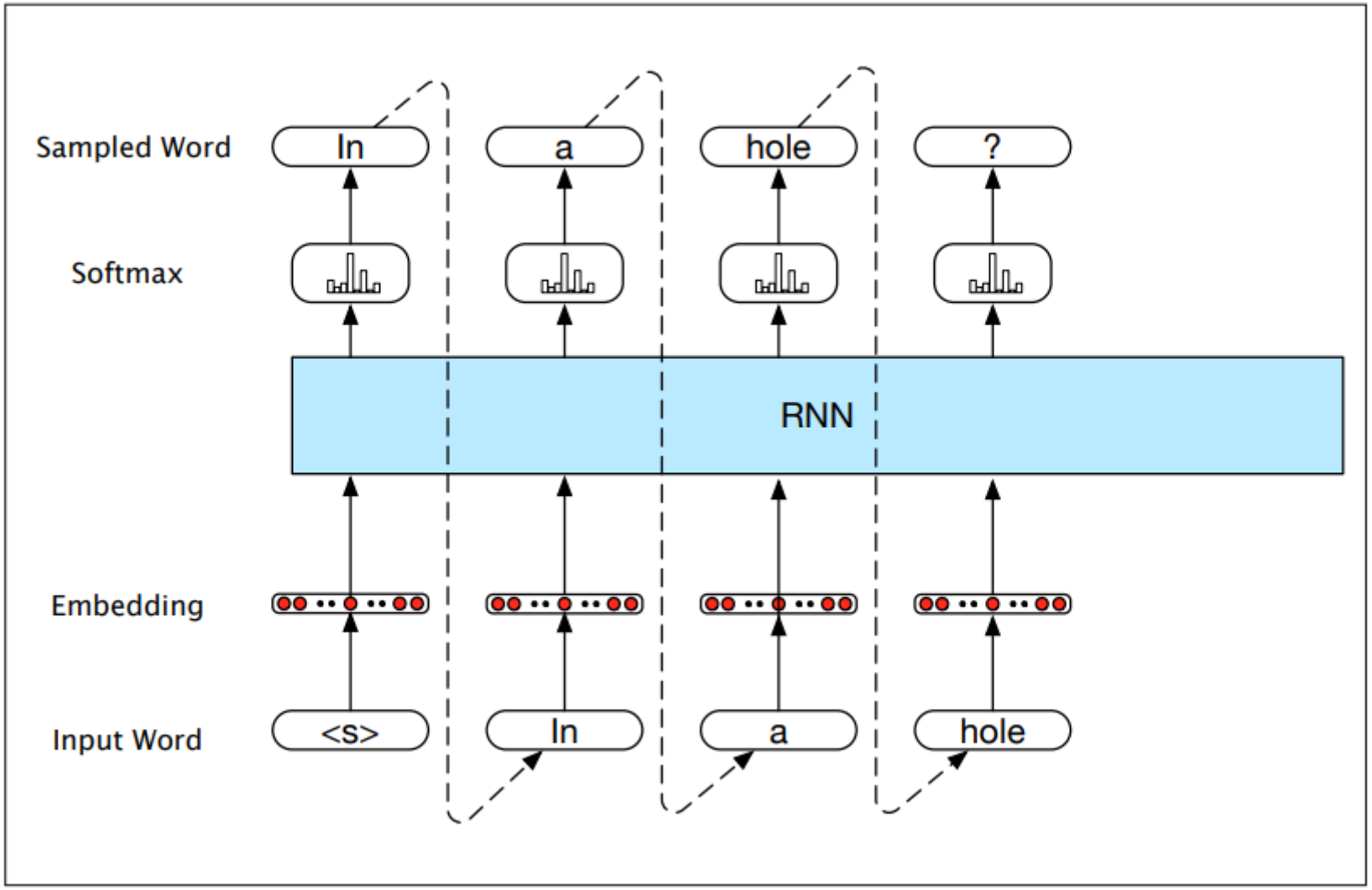
That superscript 'i' is a word index
for output 'y' at timestep 't'



RNN Notes

- Teacher forcing: we use the ground truth as input to predict the next output [instead of using the previously predicted output]
- End-to-end training: using the loss for the output to adjust the weights throughout the network
- Backpropagation Through Time: gradients are based on all contributions to the loss [summed over the sequence]
- Vanishing gradients: repeated multiplications across time steps during back propagation can result in gradients being driven to zero
- Dropout: dropping out the same elements across time steps for a sequence is preferred

Autoregressive Generation

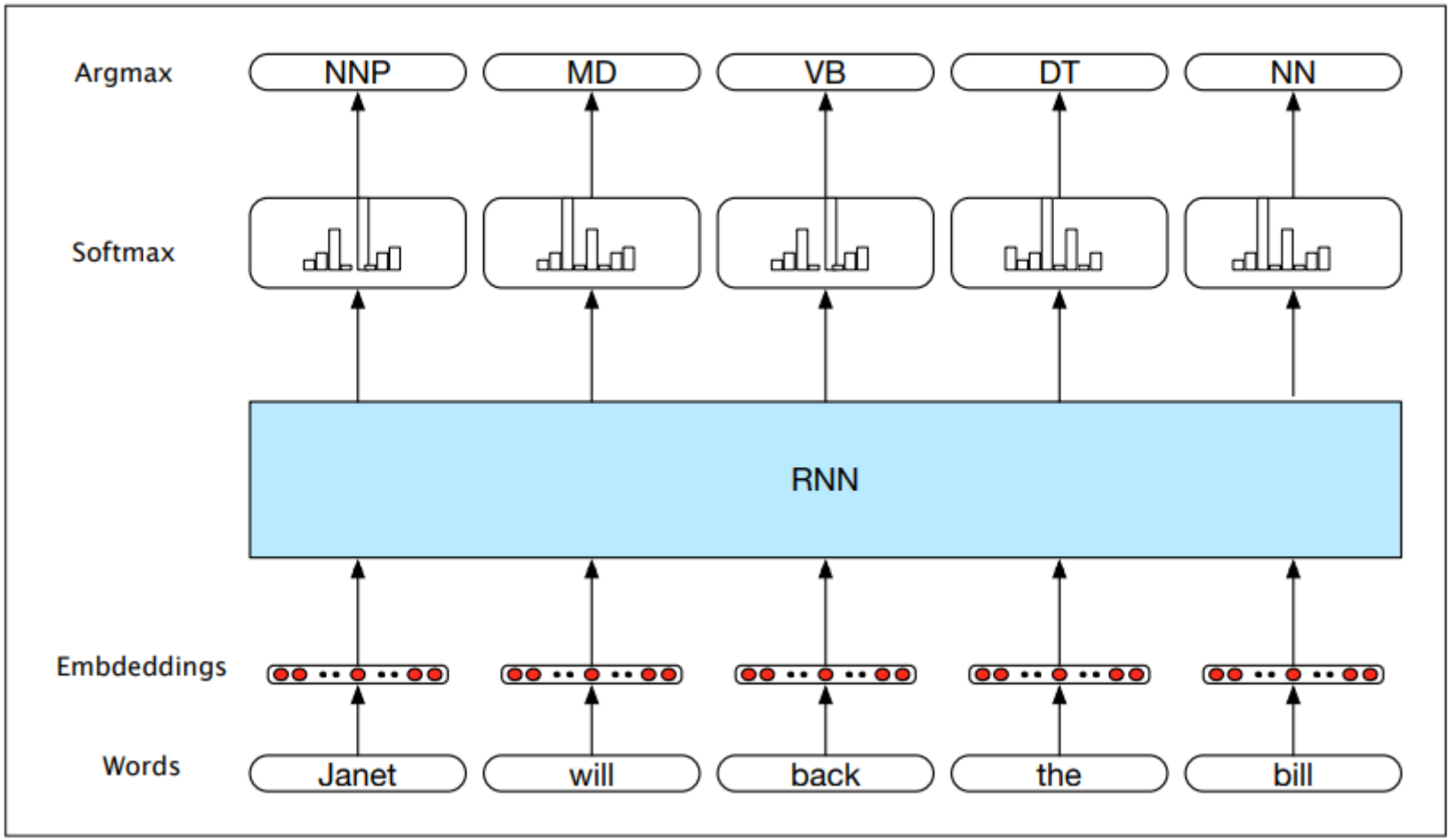


Start with a start symbol (e.g. <s>); stop with a stop symbol (e.g. </s>) or max length

Other Applications of RNNs

- Sequence labeling; e.g. part of speech tagging, Named Entity Recognition (NER)
- Sequence classification; e.g. sentiment analysis, topic classification
- Sequence-to-Sequence tasks; e.g. summarization, machine translation, question answering

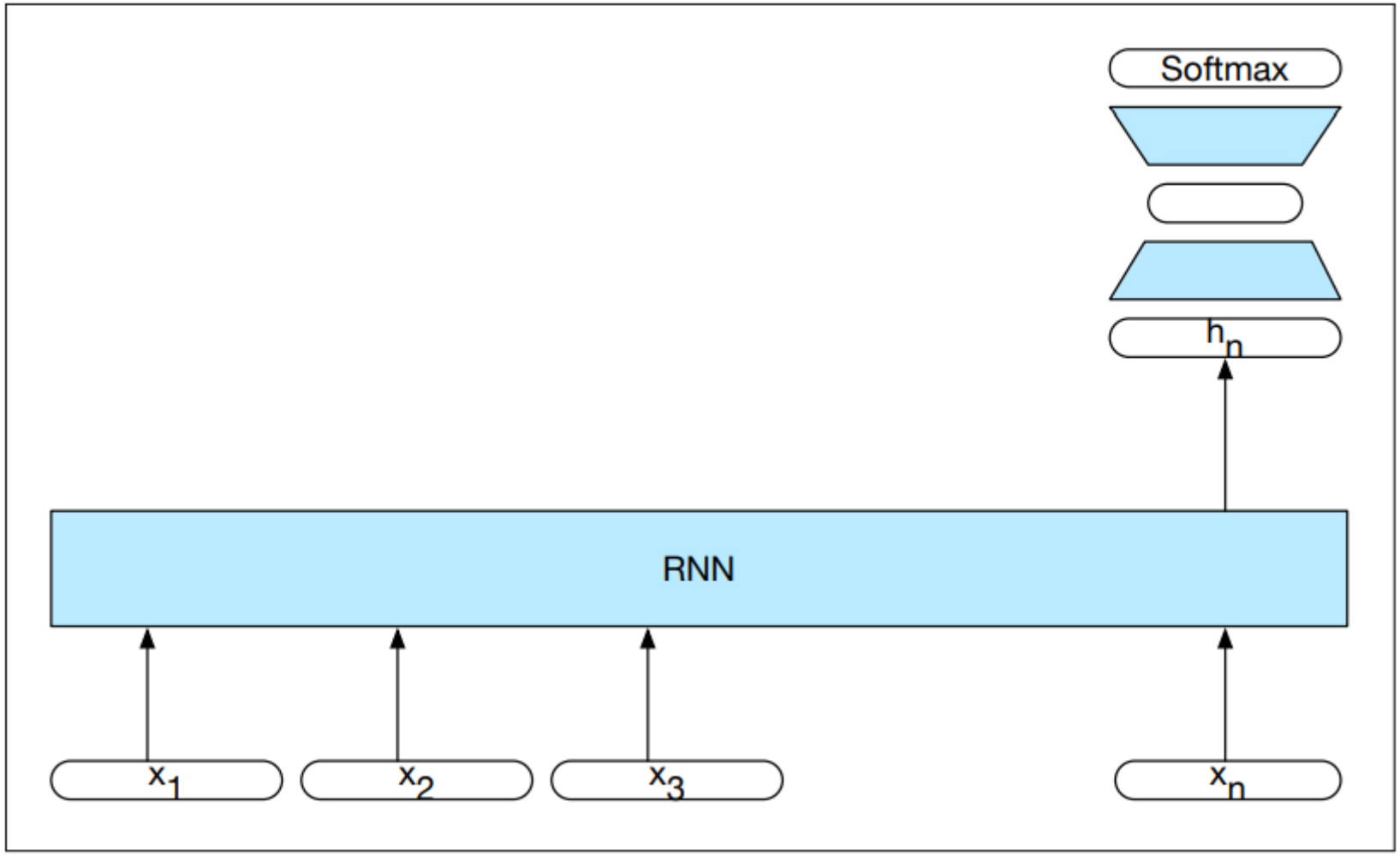
Part Of Speech Tagging



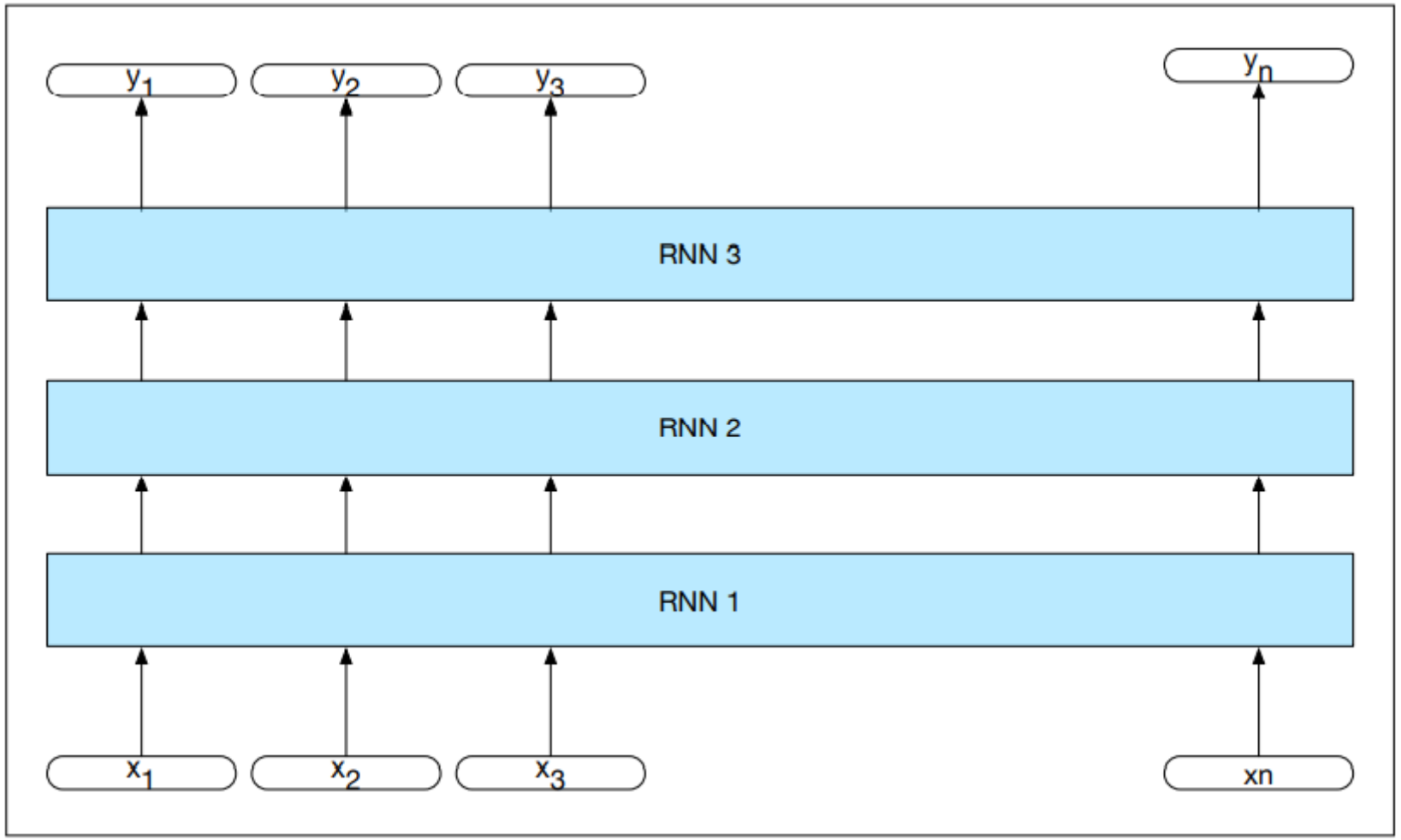
NNP: proper noun, singular; MD: modal; VB: verb, base form; DT: determiner; NN: noun, singular

Sequence Classification

Only the last h_n exposed to the next layer

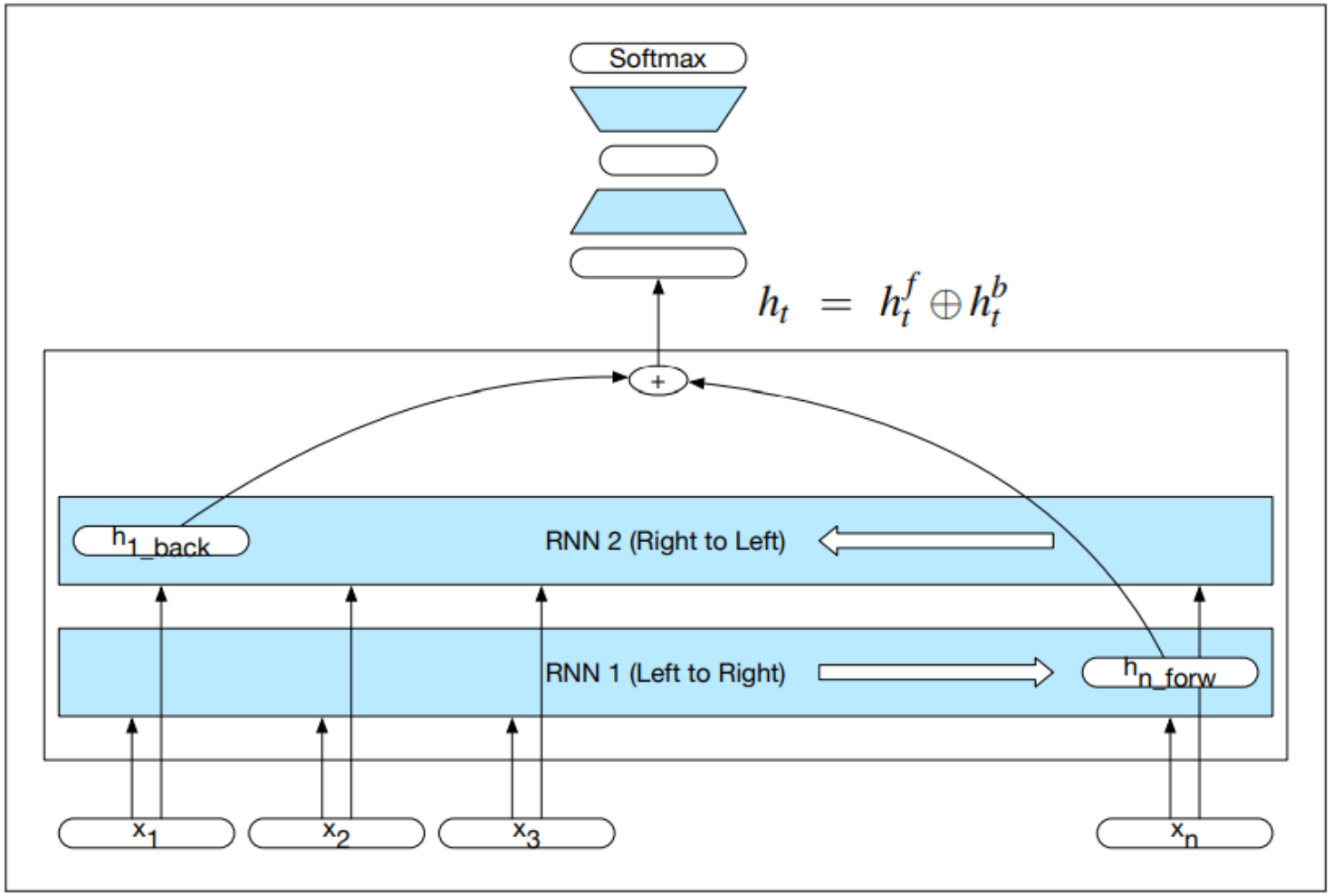


Stacked RNNs



return_sequences = True

Bidirectional RNNs



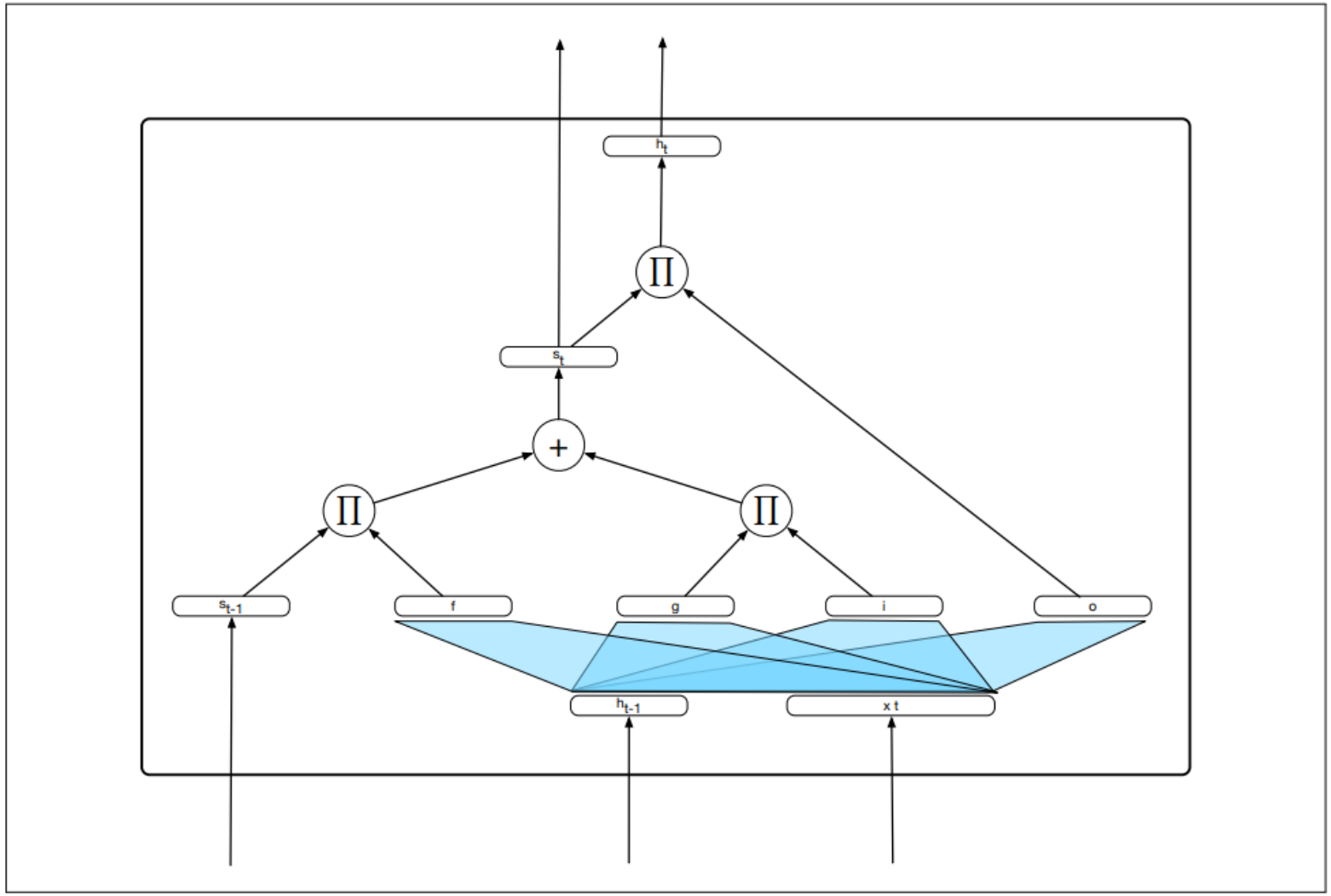
h_{1_back} vector is concatenated to h_{n_forw} vector

Long Short-Term Memory (LSTM) Cell

- LSTM cell maintains a separate context (memory) across time steps
- 8 weight matrices instead of 2 (or 4 if previous time step activations are concatenated to previous layer activations):
  - Forget gate
  - Features
  - Input gate
  - Output gate
  - Biases used by default (though omitted by author)
- Steps:
  - Forget gate applied to previous context (memory)
  - Features generated
  - Input gate applied to features
  - Context updated
  - Output gate applied to context activations (output)

$$
\begin{aligned}
f_t &= \sigma(U_f h_{t-1} + W_f x_t) \\
k_t &= c_{t-1} \odot f_t \\
g_t &= tanh(U_g h_{t-1} + W_g x_t) \\
i_t &= \sigma(U_i h_{t-1} + W_i x_t) \\
j_t &= g_t \odot i_t \\
c_t &= j_t + k_t \\
o_t &= \sigma(U_o h_{t-1} + W_o x_t) \\
h_t &= o_t \odot tanh(c_t)
\end{aligned}
$$

LSTM Cell as a Computation Graph



Author appears to be using s_t in place of c_t

Gated Recurrent Unit (GRU) Cell

- 6 weight matrices instead of 2 (or 3 if previous time step activations are concatenated to previous layer activations):
  - Reset gate
  - Update gate
  - Features
  - Biases used by default (though omitted by author)
- Steps:
  - Compute weights for reset and update gates
  - Apply reset gate to previous time step activations, and generate features
  - Apply update gate to generate output activations as the weighted mean of the current features and the previous time step activations

$$
\begin{aligned}
r_t &= \sigma(U_r h_{t-1} + W_r x_t) \\
z_t &= \sigma(U_z h_{t-1} + W_z x_t) \\
\tilde{h}_t &= tanh(U(r_t \odot h_{t-1}) + W x_t) \\
h_t &= (1 - z_t) h_{t-1} + z_t \tilde{h}_t
\end{aligned}
$$

Dense and Recurrent Layers

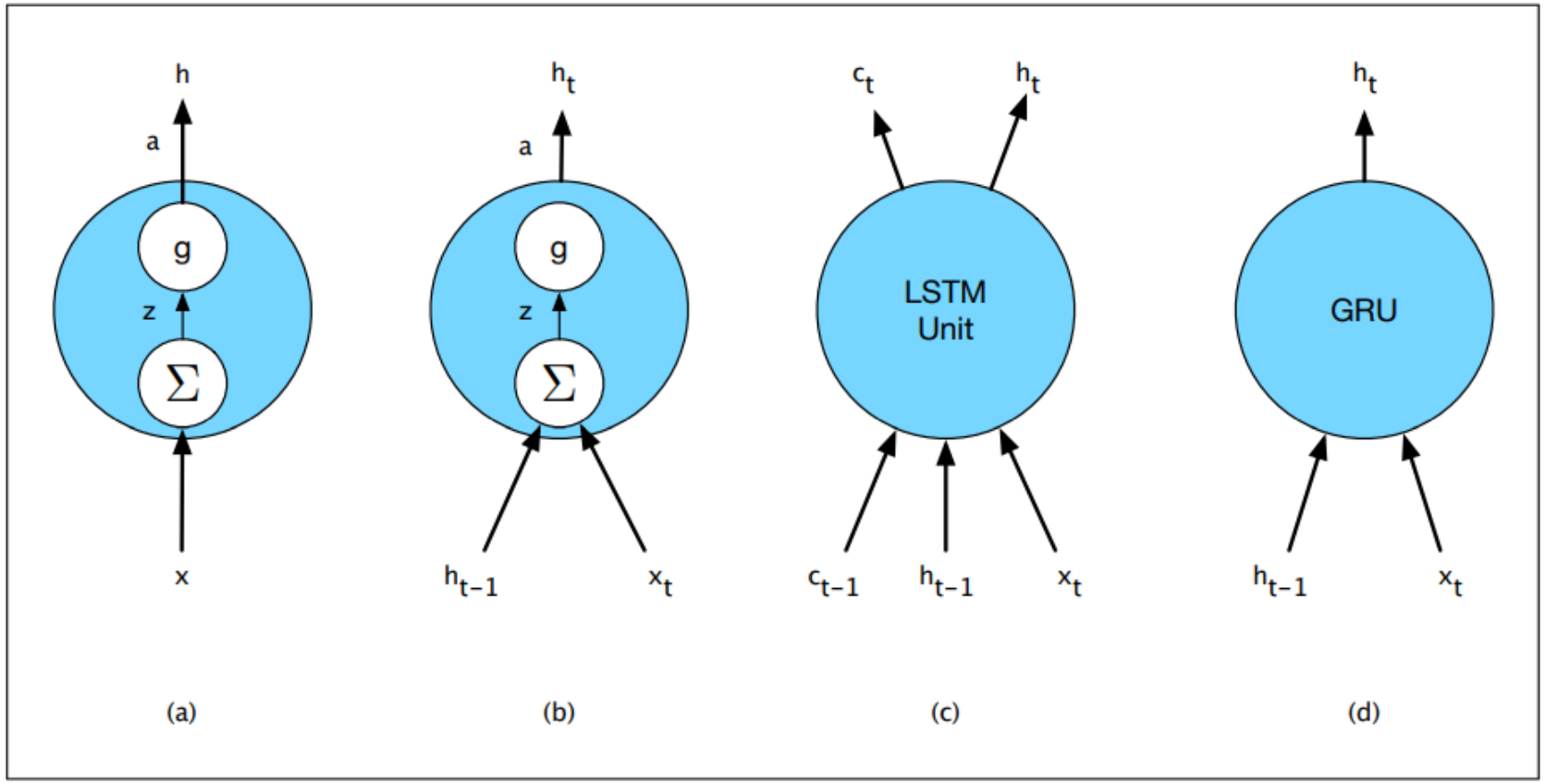
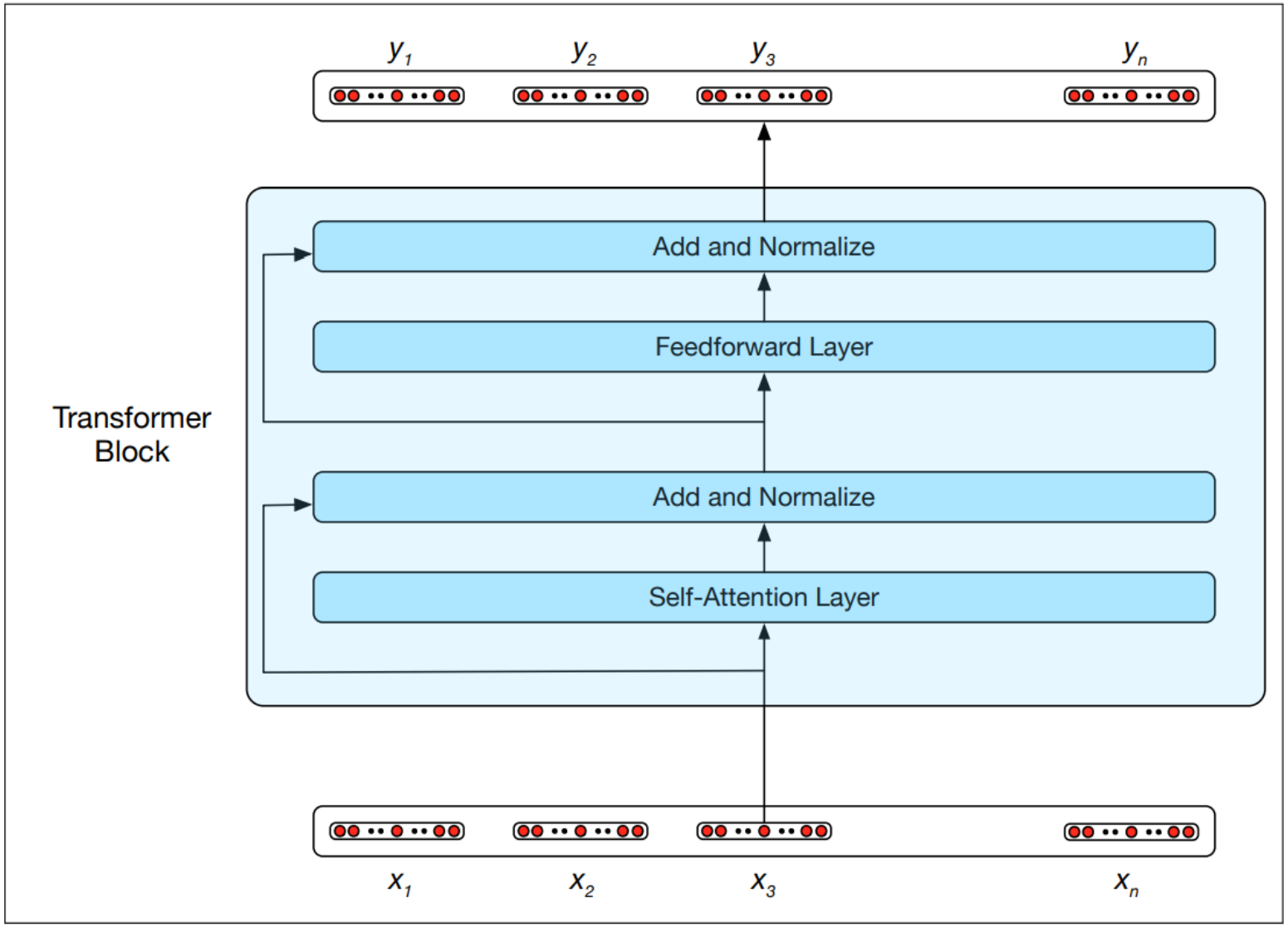


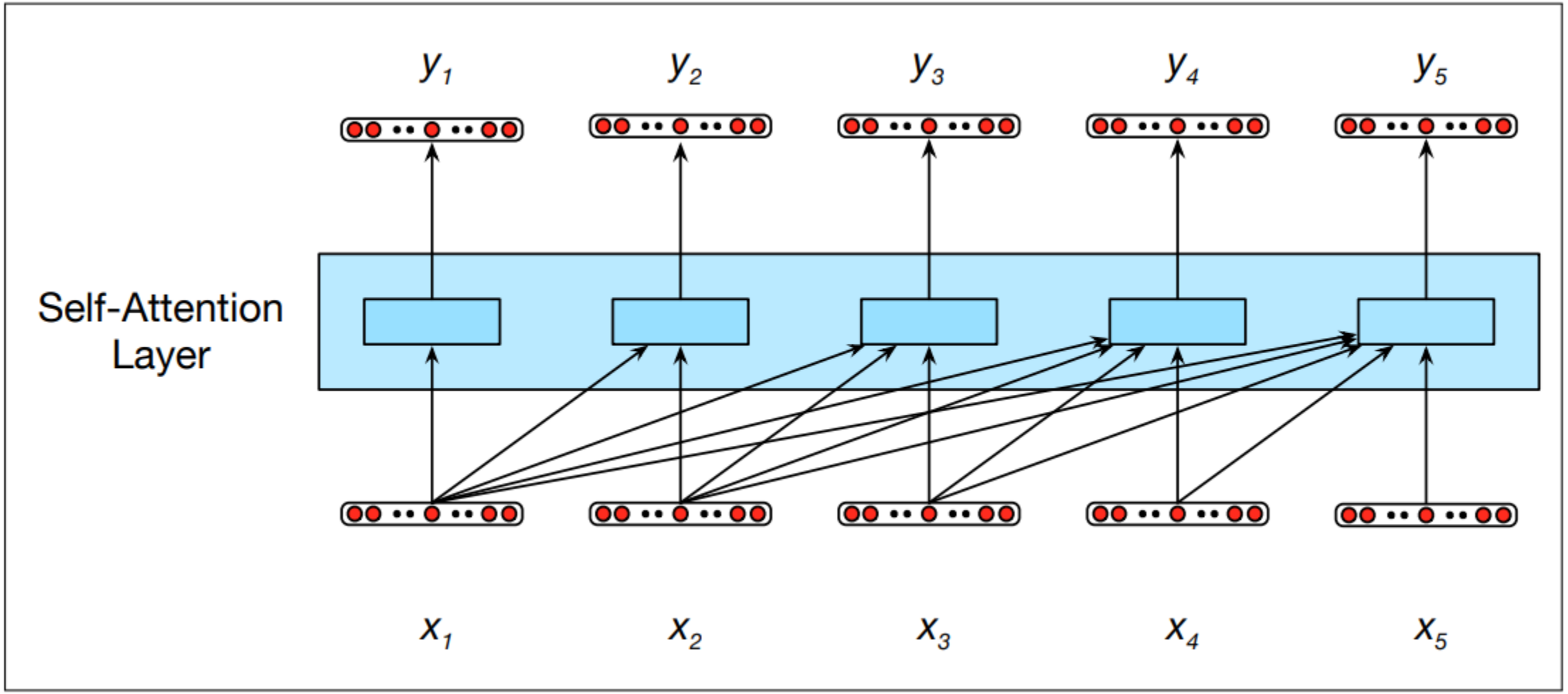
Figure 9.14 Basic neural units used in feedforward, simple recurrent networks (SRN), long short-term memory (LSTM) and gate recurrent units.

Transformer Block

- Self-Attention and Feedforward "layers" form residuals
- Self-Attention covered on the next few slides
- Layer normalization also described later
- Feedforward block has 2 matrices: one to generate d_{ff} features (a large number of features), and another to project them back to d_m (e.g. $d_{ff} = 4*d_m$, where d_m is the number of features in each position's embedding)



“Causal” Self-Attention



Lookahead not allowed for language models ☺;
but bidirectional embeddings are fine for other applications (such as sequence classification)

Self Attention: Queries, Keys, and Values

- Terms “query“, “key“, and “value“ are from information retrieval
- Roles played by the input embeddings during the attention process
  - Each query embedding is matched against the key embeddings to produce a weighted sum of value embeddings
  - Weight matrices are used to derive Query, Keyword, and Value features for the input embeddings, then …

$$SelfAttention(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Example values for a self-attention block:
h = number of heads = 16
$d_q = d_k = d_v = 64$
$d_m = h * d_v = 1024$ [this is the size of each position’s embedding]

d for dimensions (or depth)
h for heads
k for key
m for model
q for query
v for value

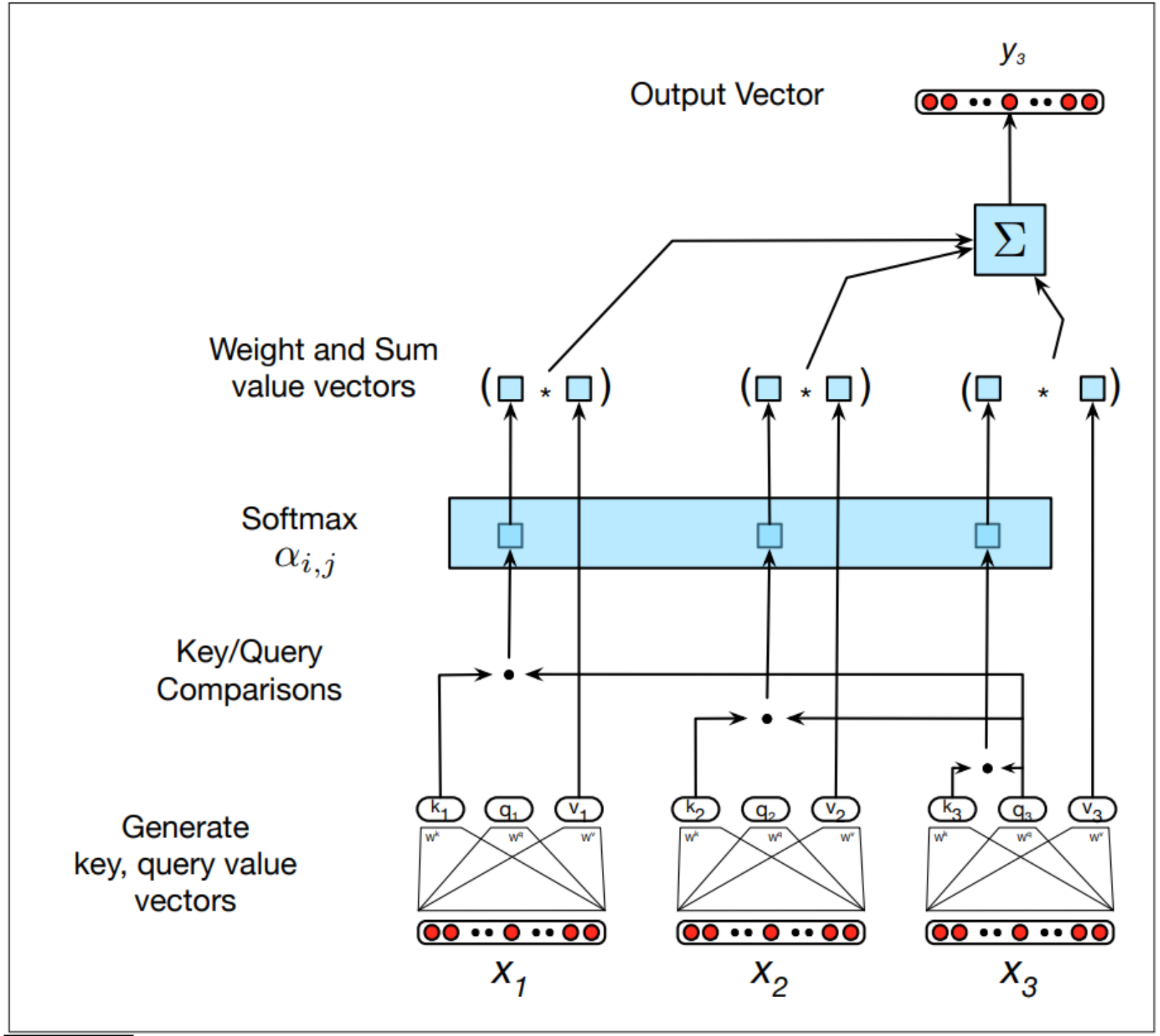
https://www.cross-entropy.net/ML530/transformer.py

Please note that we've switched back to $Q = X * W_Q$; $K = X * W_K$; $V = X * W_V$ for this slide

Self-Attention Code

```
def scaled_dot_product_attention(q, k, v):
    matmul_qk = tf.matmul(q, k, transpose_b = True)    # (..., seq_len, seq_len)

    dk = tf.cast(tf.shape(k)[-1], tf.float32)
    scaled_attention_logits = matmul_qk / tf.math.sqrt(dk)    # scale matmul_qk

    # softmax is normalized on the last axis (seq_len) so that the scores add up to 1
    attention_weights = tf.nn.softmax(scaled_attention_logits, axis=-1)    # (..., seq_len, seq_len)

    output = tf.matmul(attention_weights, v)    # (..., seq_len, depth)

    return output, attention_weights
```

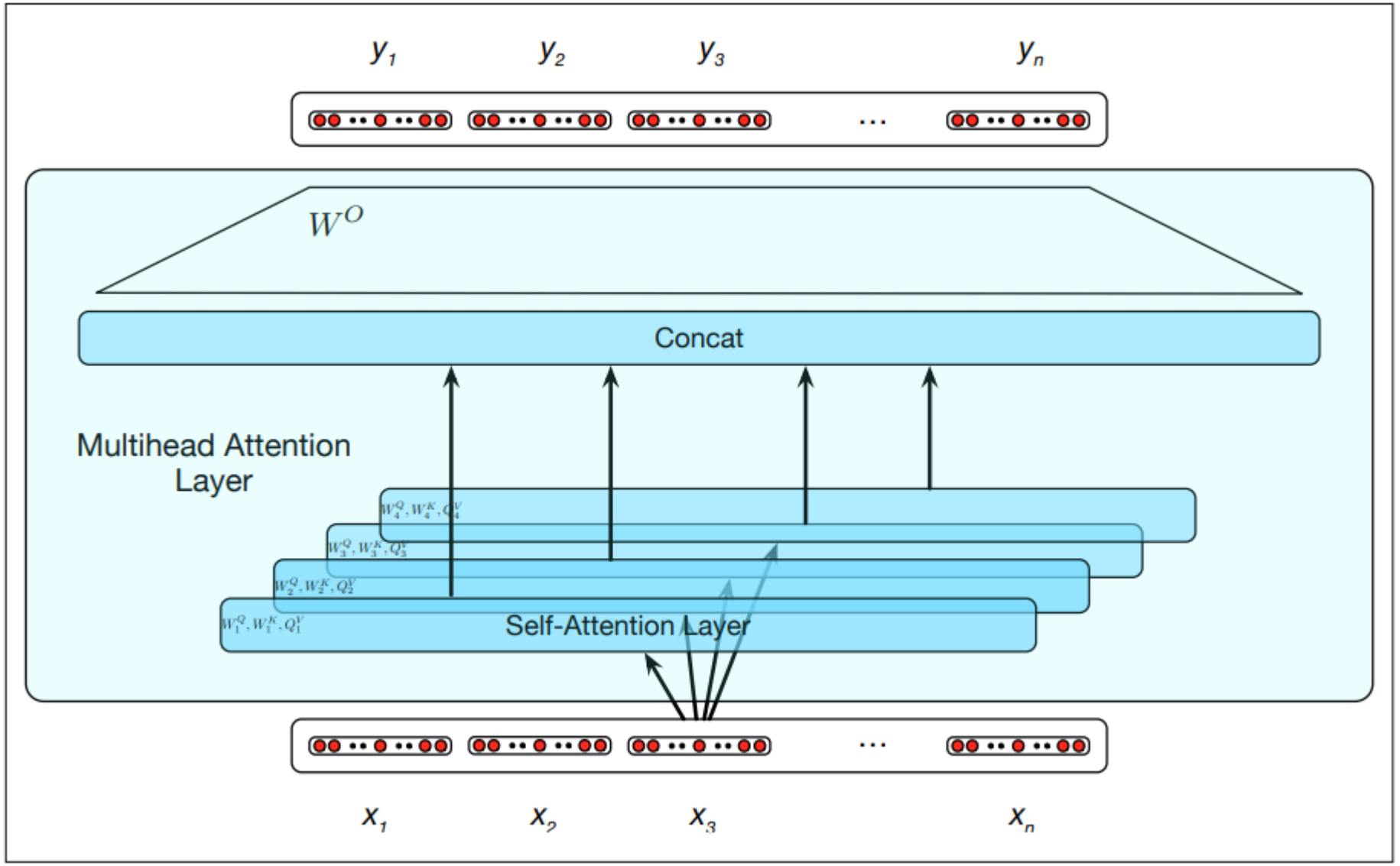
Computing the weighted value embedding for position 3 …



The Multihead Self-Attention Layer

Author says “projected down to d_m”

Note that the output “projection” matrix has access to features from all heads



Position Embeddings

- “if you scramble the order of inputs in the attention computation illustrated earlier you get exactly the same answer”
- Position embeddings typically trained as part of the network; i.e. there is an embedding matrix for the Word embeddings and a second embedding matrix for the Position embeddings
- Position indices are literally used to retrieve Position embeddings
- Position embeddings are added to the Word embeddings, then layer normalization is performed

Layer Normalization

For each sample `x_i` in `inputs` with `k` features, we compute the mean and variance of the sample:

```
mean_i = sum(x_i[j] for j in range(k)) / k
var_i = sum((x_i[j] - mean_i) ** 2 for j in range(k)) / k
```

and then compute a normalized `x_i_normalized`, including a small factor `epsilon` for numerical stability.

```
x_i_normalized = (x_i - mean_i) / sqrt(var_i + epsilon)
```

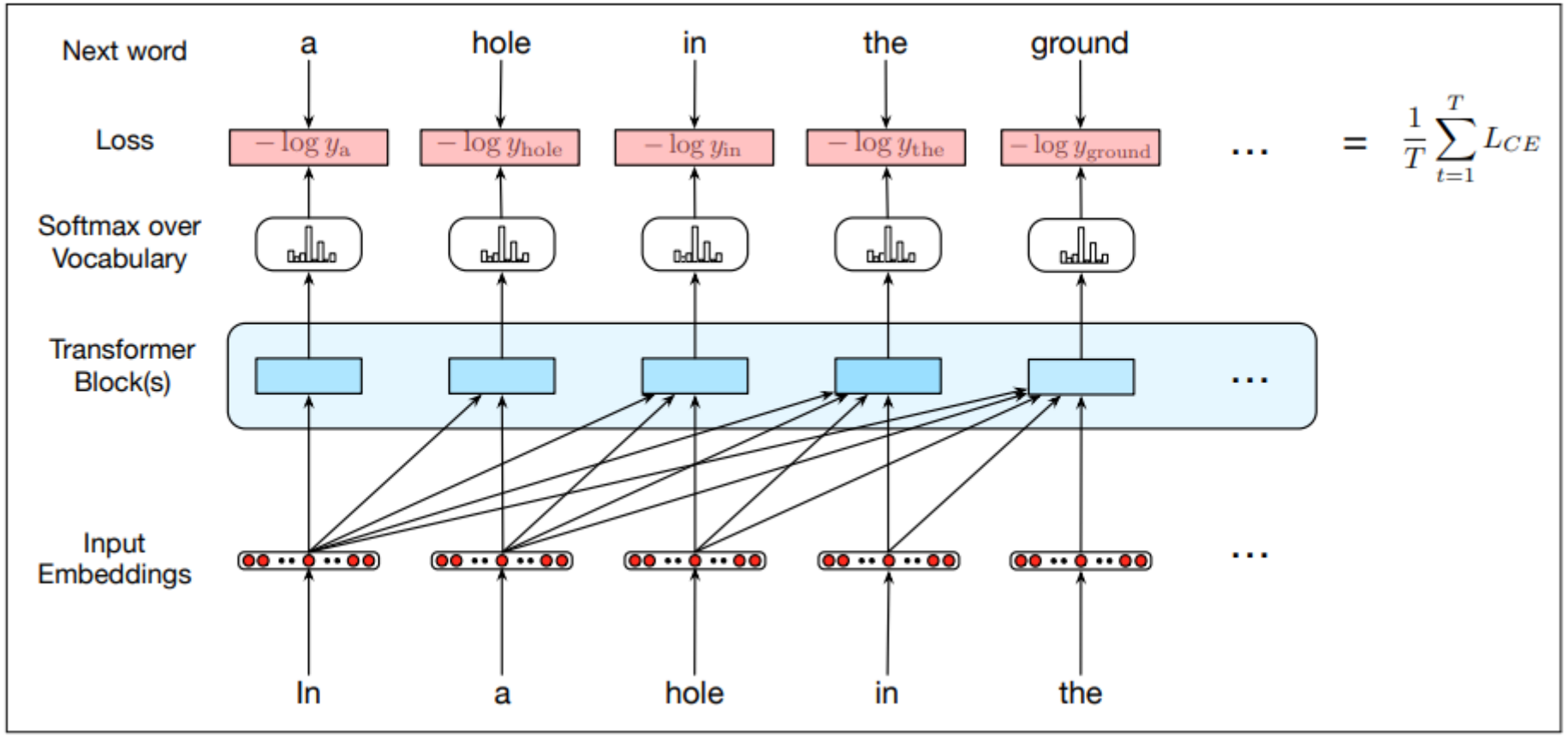
And finally `x_i_normalized` is linearly transformed by `gamma` and `beta`, which are learned parameters:

```
output_i = x_i_normalized * gamma + beta
```

https://keras.io/api/layers/normalization_layers/layer_normalization/

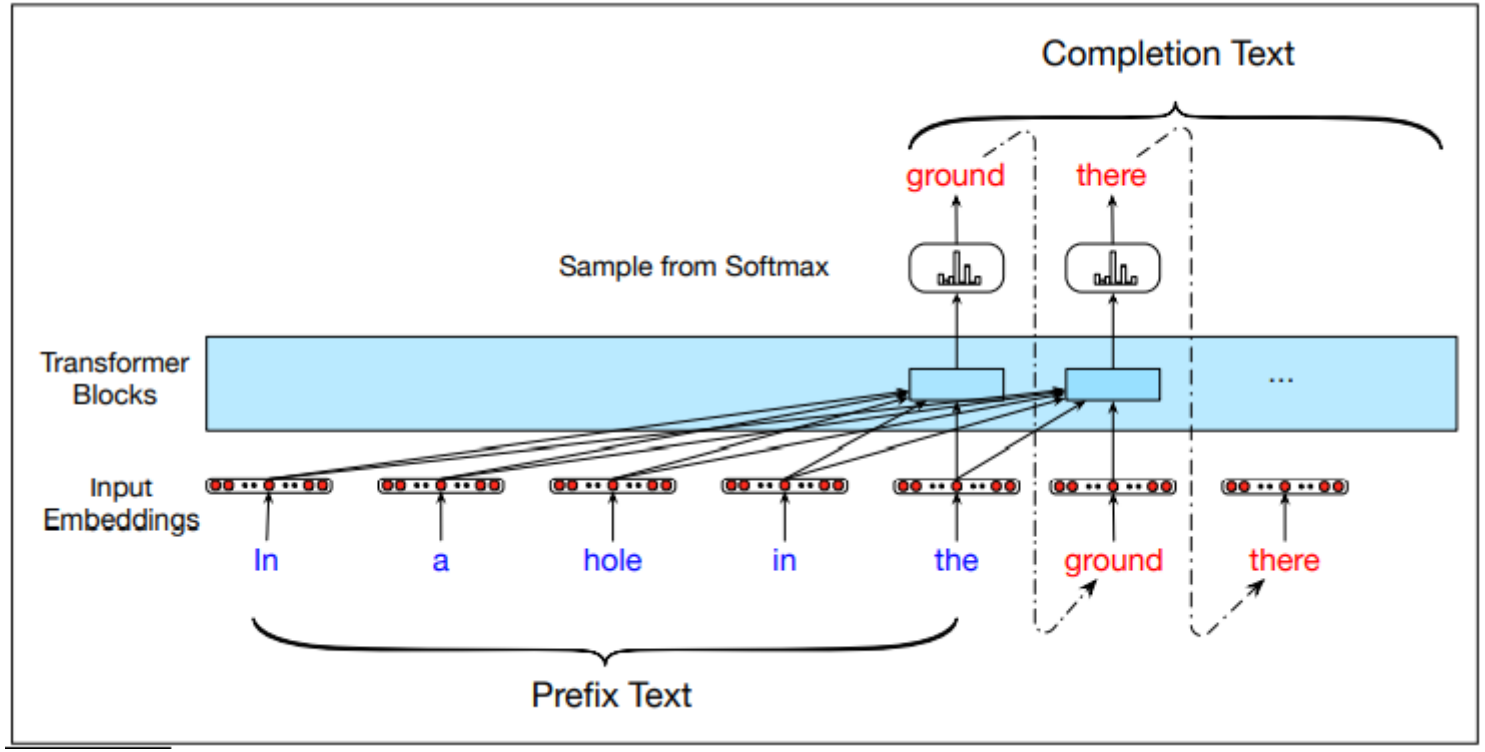
Training a Transformer as a Language Model

Same 6 matrix transformer block used across positions



"the elements in the upper-triangular portion of the attention matrix are zeroed out, thus eliminating any knowledge of words that follow in the sequence" [masking]

Autoregressive Text Completion



Cable News Network Daily Mail Example

Original Article

The only thing crazier than a guy in snowbound Massachusetts boxing up the powdery white stuff and offering it for sale online? People are actually buying it. For $89, self-styled entrepreneur Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says.

But not if you live in New England or surrounding states. "We will not ship snow to any states in the northeast!" says Waring's website, ShipSnowYo.com. "We're in the business of expunging snow!"

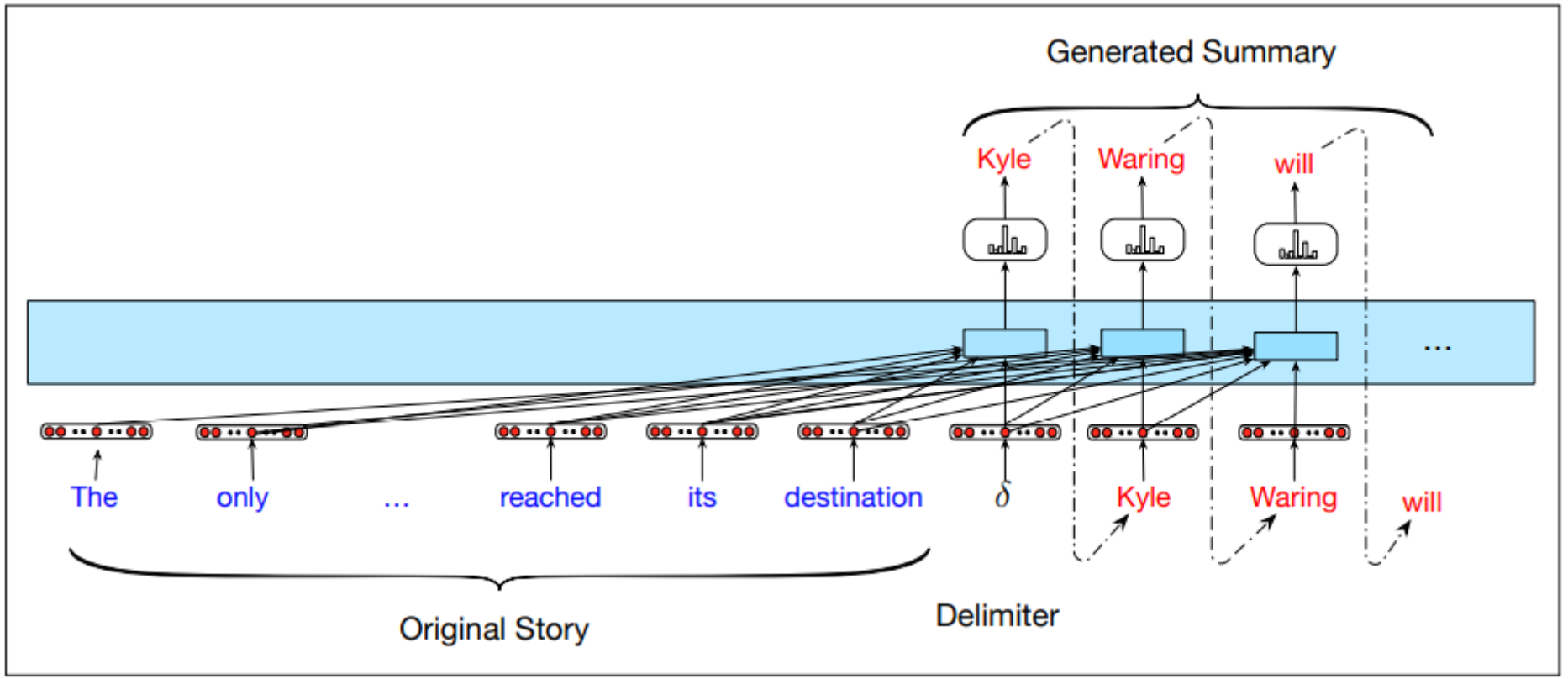
His website and social media accounts claim to have filled more than 133 orders for snow – more than 30 on Tuesday alone, his busiest day yet. With more than 45 total inches, Boston has set a record this winter for the snowiest month in its history. Most residents see the huge piles of snow choking their yards and sidewalks as a nuisance, but Waring saw an opportunity.

According to Boston.com, it all started a few weeks ago, when Waring and his wife were shoveling deep snow from their yard in Manchester-by-the-Sea, a coastal suburb north of Boston. He joked about shipping the stuff to friends and family in warmer states, and an idea was born. His business slogan: "Our nightmare is your dream!" At first, ShipSnowYo sold snow packed into empty 16.9-ounce water bottles for $19.99, but the snow usually melted before it reached its destination...

Summary

Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says. But not if you live in New England or surrounding states.

Summarization with Transformers



Potential Harms from Language Models

- Biases from the training data can cause trouble, as can training on sensitive data
- Beware that bad things can happen during query completion or predictive typing …
  - Toxic language
    - Negative connotations/vibes based on gender, race, religion, sexual orientation, etc
    - Hate speech
    - Misinformation
    - Radicalization
  - Information leaks
    - Sensitive data from the training corpus

Speech and Language Processing 3rd ed draft

Chapter 9: Deep Learning Architectures for Sequence Processing

https://web.stanford.edu/~jurafsky/slp3/

1. Language Models “Revisited”
2. Recurrent Neural Networks
3. Managing Context in RNNs: LSTMs and GRUs
4. Self-Attention Networks: Transformers
5. Potential Harms from Language Models