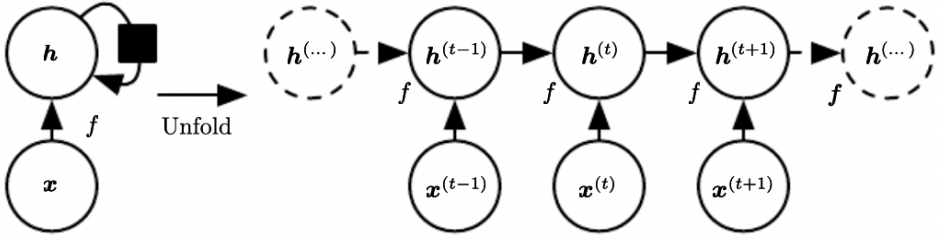
Models for sequence data

Brooks Paige

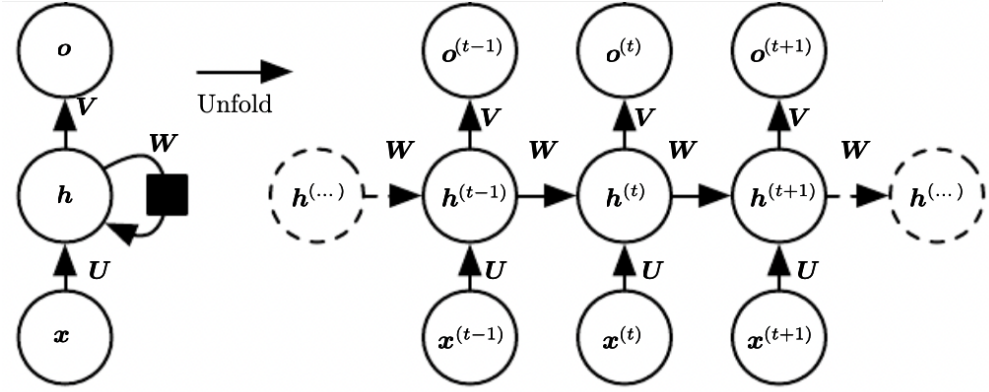
COMP0171

Recurrent neural networks



$$\mathbf{h}_t = f_\theta(\mathbf{x}_t, \mathbf{h}_{t-1})$$

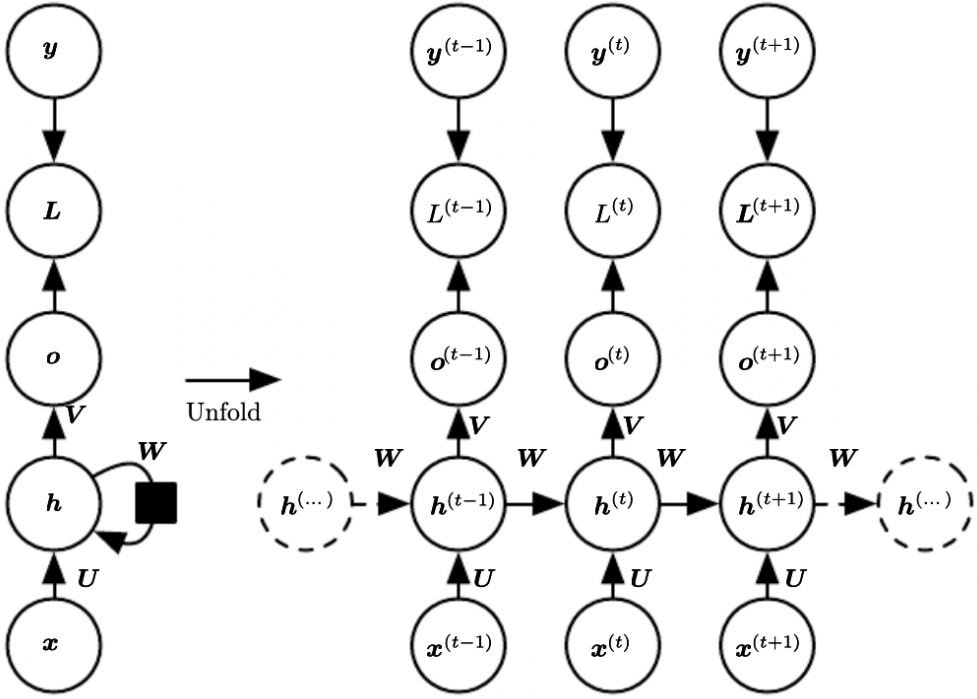
Recurrent neural networks



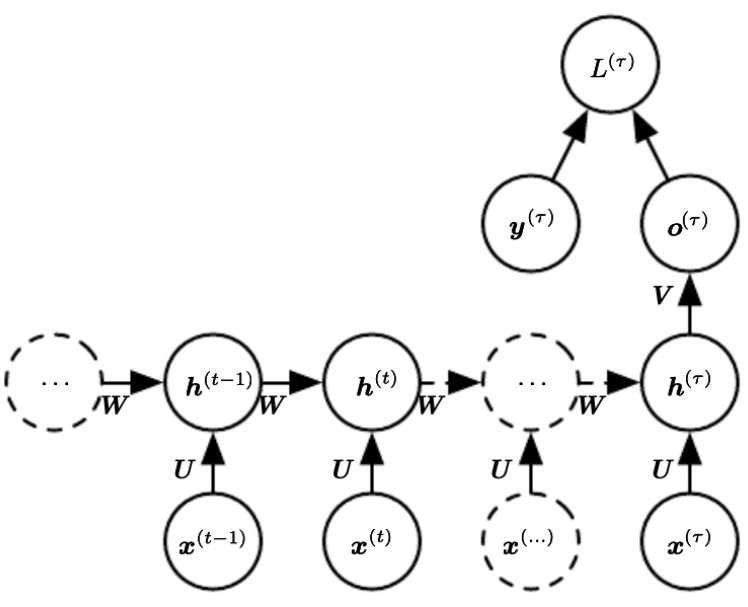
$$\mathbf{a}_t = \mathbf{b} + \mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t \qquad \mathbf{h}_t = \tanh(\mathbf{a}_t) \qquad \mathbf{o}_t = \mathbf{c} + \mathbf{V}\mathbf{h}_t$$

Recurrent neural networks

$$\mathbf{a}_t = \mathbf{b} + \mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t$$
$$\mathbf{h}_t = \tanh(\mathbf{a}_t)$$
$$\mathbf{o}_t = \mathbf{c} + \mathbf{V}\mathbf{h}_t$$
$$\hat{y}_t = \mathrm{softmax}(\mathbf{o}_t)$$

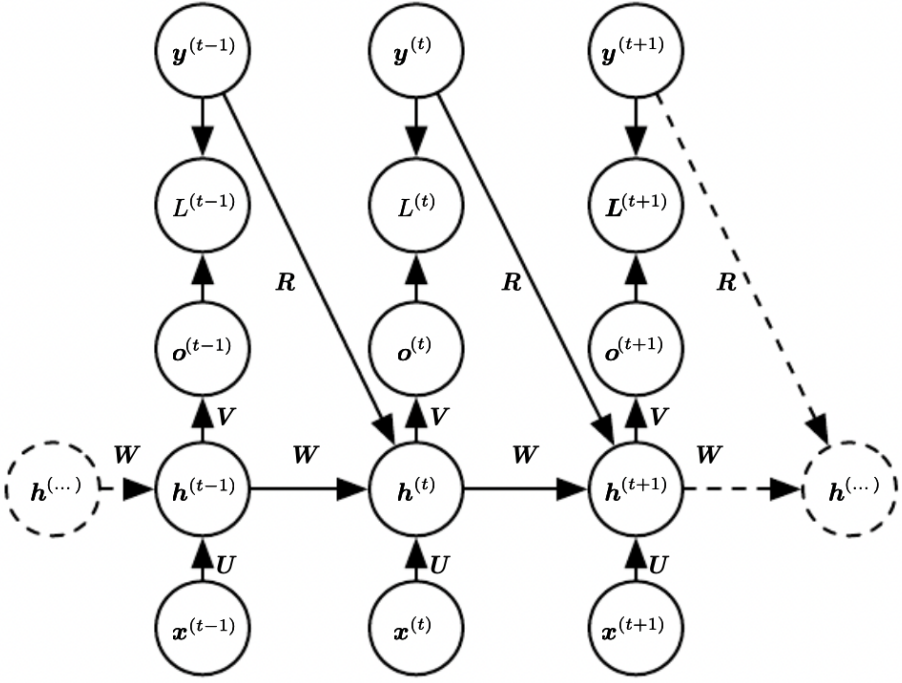


Summarizing an entire sequence

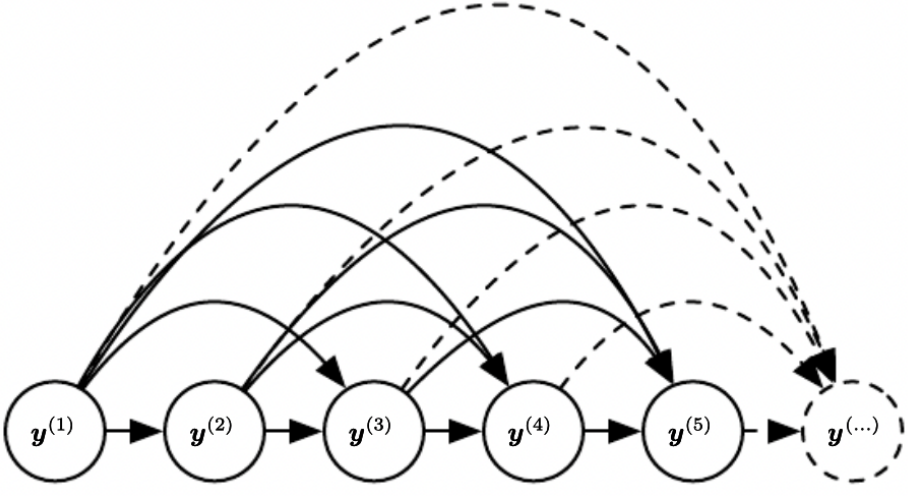


$$\mathbf{h}_\tau = f_\theta(\mathbf{x}_1, \ldots, \mathbf{x}_\tau) \qquad \mathbf{y} = g_\theta(\mathbf{h}_\tau)$$

Conditioning on output values

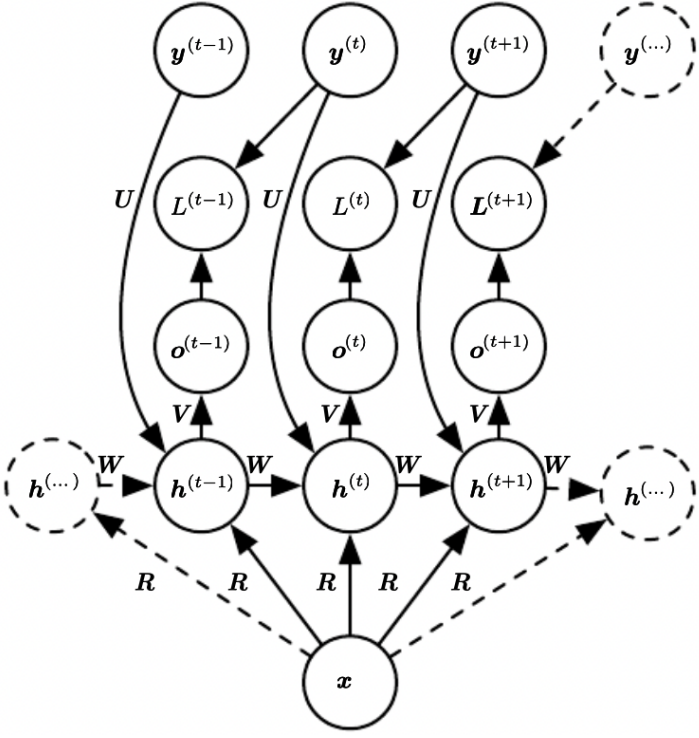


As a probability distribution over outputs

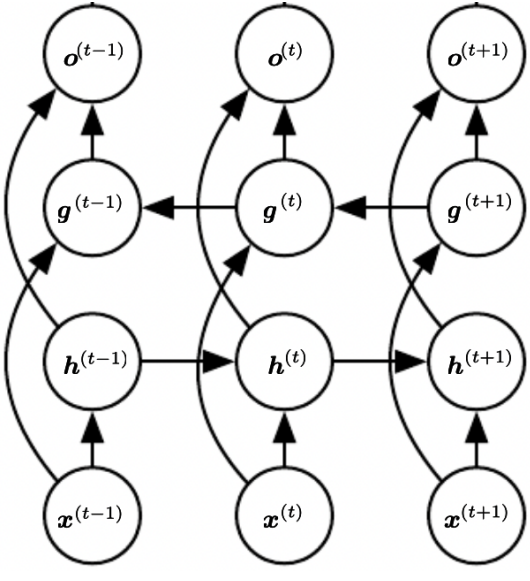


$$p(y_1, \ldots, y_\tau) = p(y_1)p(y_2|y_1)p(y_3|y_1, y_2) \ldots p(y_\tau|y_1, \ldots, y_{\tau-1})$$

Conditioning on "context"

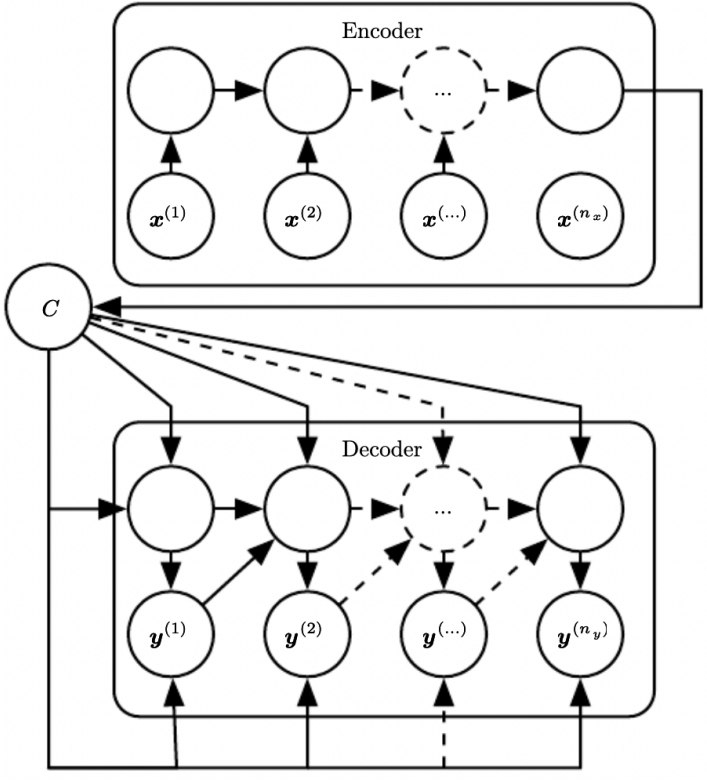


Bidirectional RNNs

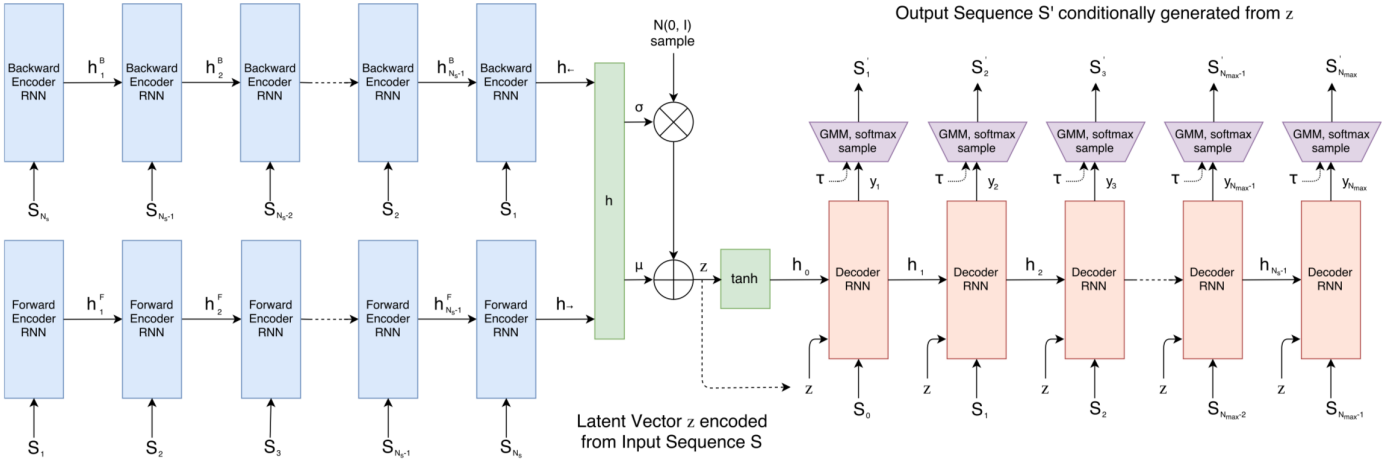


When conditioning on the whole sequence, reading it "both ways" to have features that depend on the "future" (e.g. for transcribing pre-recorded speech)

Input and output sequences of different lengths



Encoder / decoder architecture



Using a bidirectional encoder to extract "context" and a unidirectional decoder means that it is still possible to treat the decoder generatively.

Figure: Ha & Eck

Dealing with long sequences

Vanishing and exploding gradients

When trying to compute derivatives backward through time, we run again into the common problem of having derivatives which can vanish (or even explode!) as the number of sequential operations increases.

Additionally, while RNNs in principal can model long-range dependencies, in practice when training them they tend to only capture shorter-term behaviour.

One easy way of stabilizing things is to include skip connections through time. But there is a more common solution for RNNs in particular...

Gated networks (LSTM and GRU)

The equations for the recurrent network earlier were the most simple type of "recurrent unit", i.e. $f_\theta(\mathbf{x}_t, \mathbf{h}_{t-1})$.

In general two more complicated functional forms are used:

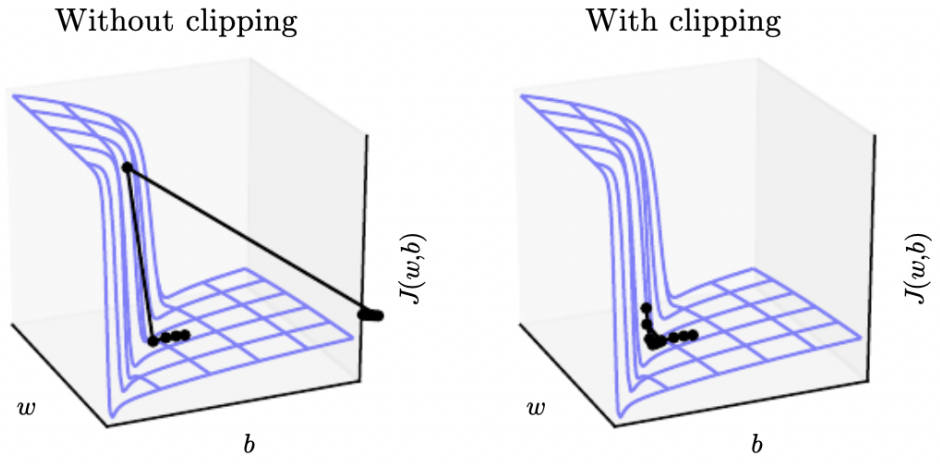
- **Long short-term memory** (LSTM), and
- **Gated recurrent unit** (GRU).

Both of these introduce additional weights, and a "gating" behaviour which controls the flow of the hidden state through time. The GRU update is

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{h}_{t-1} + \mathbf{V}_z \mathbf{x}_t) \qquad \mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{h}_{t-1} + \mathbf{V}_r \mathbf{x}_t)$$
$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{V}\mathbf{x}_t) \qquad \mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t$$

These models are additionally better at capturing long-term dependencies than "vanilla" RNNs.

Gradient clipping



If $\|\mathbf{g}\| > v$ then $\mathbf{g} \leftarrow \frac{\mathbf{g}v}{\|\mathbf{g}\|}$

A note on training objectives and algorithm

In the deep learning book, they spend a while talking about different training mechanisms: "teacher forcing", and "backpropagation through time".

In my opinion these are not too important to know about. Both of them correspond to **reverse-mode automatic differentiation of a maximum likelihood objective**, but under different choices of model structure / connectivity.

RNN Summary

- Like convolutional networks, recurrent neural networks have a particular structure well-suited to sequence data with mostly-local dependencies
- "Weight sharing" means these are quite parameter efficient
- Different high-level structures are motivated by different sorts of data, and different use-cases (e.g. sequence to sequence vs. sequence summarization)

Other language models

Transformers and Attention

On a lot of NLP tasks these days, these recurrent models are now outperformed by models based on "attention".

The most famous family of these models are **transformer** networks. These networks use "positional embeddings" for each location in the sequence to allow learning what (potentially long-range) dependencies are important.

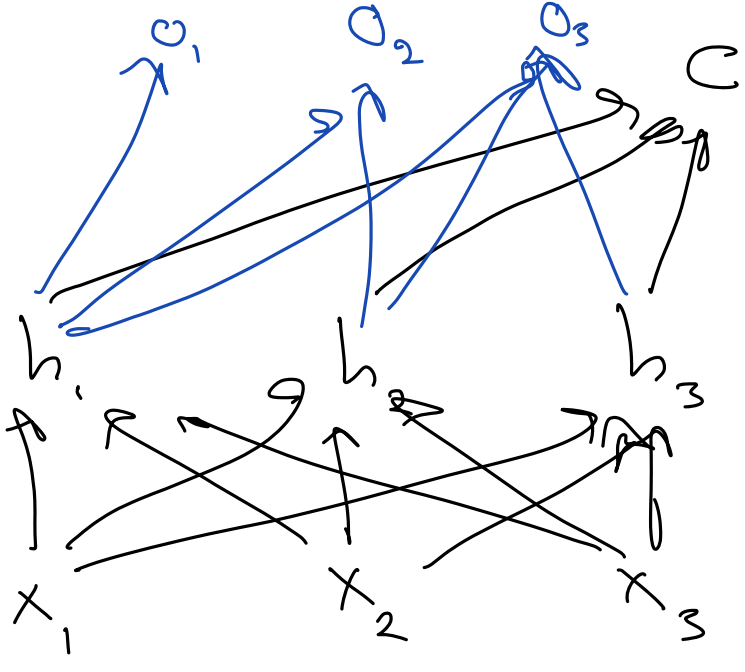
If you are interested in these, there are a lot of nice demos (and pre-trained networks, and instructional videos) at `https://huggingface.co/`.

Attention.

$$w^T h(x) = \sum_i w_i h_i(x)$$

$$w(x)^T h(x) = \sum_i w_i(x) h_i(x)$$

Sequence



. . .

Scaled dot-product attention

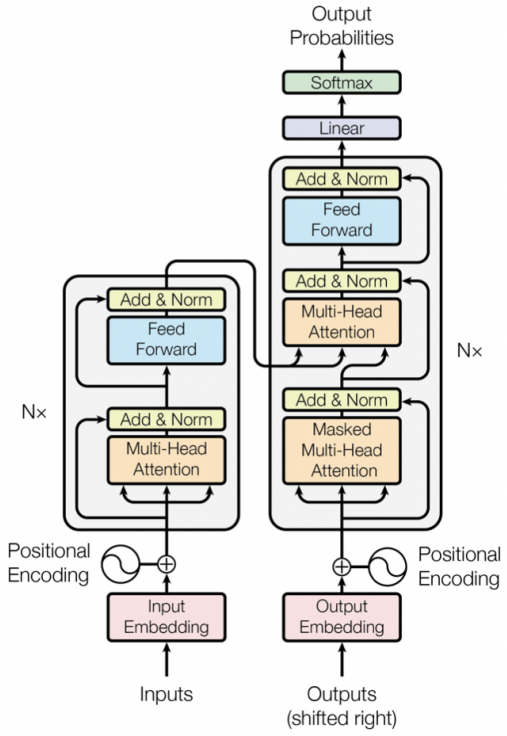
$$q : \mathbb{R}^D \to Q : \mathbb{R}^{D \times L}$$

$$K : \mathbb{R}^{M \times D}$$

$$V : \mathbb{R}^{M \times H}$$

$$\text{Att}(q, K, V) = \underbrace{\text{softmax}\left(\frac{Kq}{\sqrt{D}}\right)}_{w \in \mathbb{R}^M} V \in \mathbb{R}^H$$

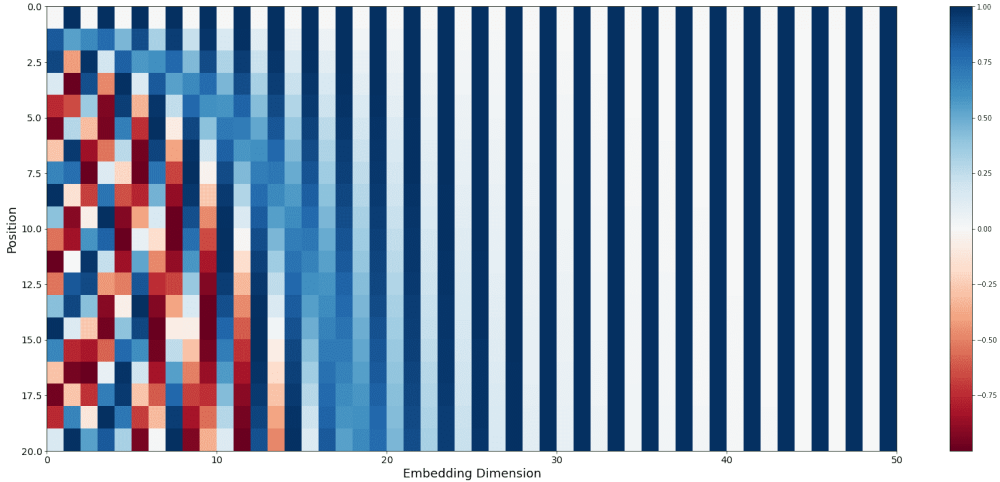
Transformers



[Figure: Vaswani et al., 2017]

Positional encoding

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$



Alternative: train positional **embeddings**

[Figure: https://erdem.pl/2021/05/understanding-positional-encoding-in-transformers]

General families of models

- Models called **GPT** or similar: stands for "Generative Pre-trained Transformer". Autoregressive (left-to-right).
  - For **generating sequences**, like a unidirectional RNN
- Models like **BERT**: stands for "Bidirectional Encoder Representations from Transformers". Trained on a "masked language model" objective, where e.g. 20% of the tokens are randomly removed, and the network aims to recover them given the rest of the sequence.
  - For **extracting features**, like a bidirectional RNN