Outline

The two approaches to time series modeling

Self-attention and Transformers

Transformers beyond time series (very briefly)
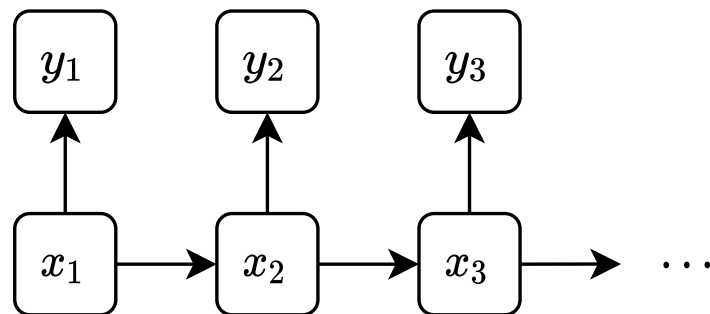
Outline

The two approaches to time series modeling

Self-attention and Transformers

Transformers beyond time series (very briefly)

Time series prediction

Let's recall our basic time series prediction task from the previous lectures

More fundamentally, a time series prediction task is the task of predicting

$$y_{1:T} = f_\theta(x_{1:T})$$

where y_t can depend only on $x_{1:t}$

There are multiple methods for doing so, which may or may not involve the latent state representation of RNNs

The RNN "latent state" approach

We have already seen the RNN approach
to time series: maintain "latent state" h_t
that summarizes *all* information up until
that point

Pros: Potentially "infinite" history,
compact representation

Cons: Long "compute path" between
history and current time $\implies$ vanishing /
exploding gradients, hard to learn
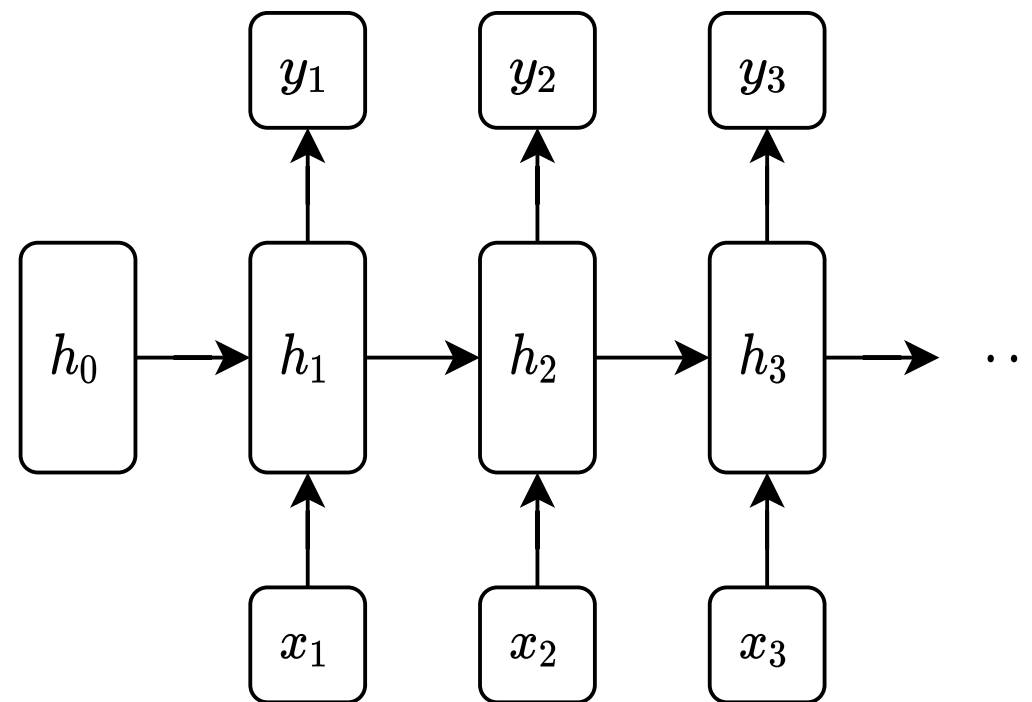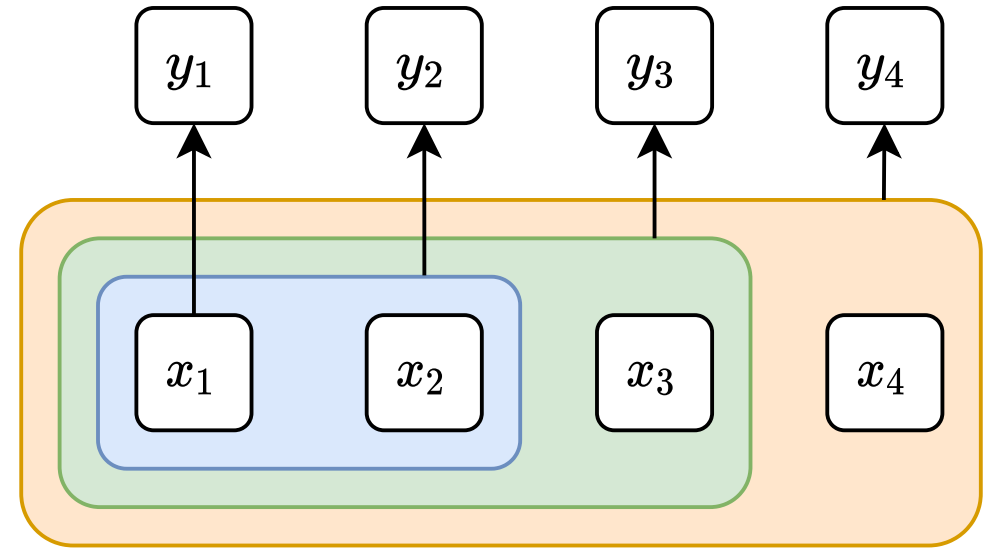
The "direct prediction" approach

In contrast, can also *directly* predict output y_t

$$y_t = f_\theta(x_{1:t})$$

(just need a function that can make predictions of differently-sized inputs)

Pros: Often can map from past to current state with shorter compute path

Cons: No compact state representation, finite history in practice

CNNs for direct prediction

One of the most straightforward ways to specify the function f_θ: (fully) convolutional networks, a.k.a. temporal convolutional networks (TCNs)

The main constraint is that the convolutions be *causal*: $z_t^{(i+1)}$ can only depend on $z_{t-k:t}^{(i)}$

Many successful applications: e.g. WaveNet for speech generation (van den Oord et al., 2016)
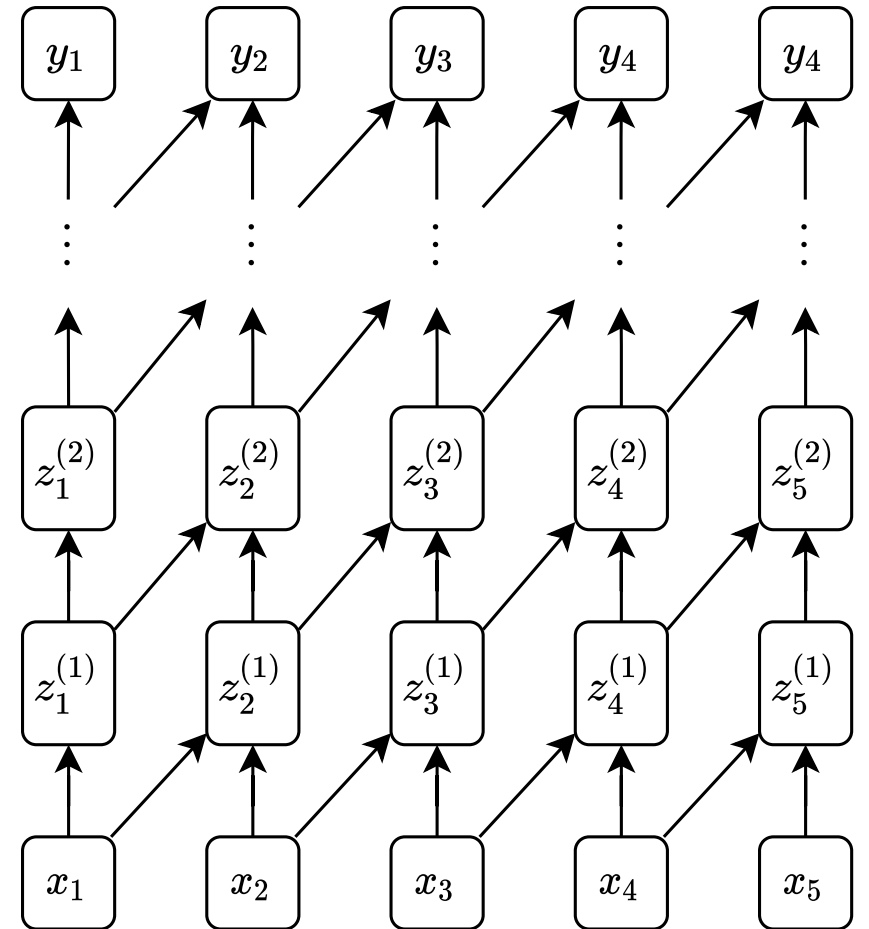
Challenges with CNNs for dense prediction

Despite their simplicity, CNNs have a notable disadvantage for time series prediction: the *receptive field* of each convolution is usually relatively small $\implies$ need deep networks to actually incorporate past information

Potential solutions:

- Increase kernel size: also increases the parameters of the network
- Pooling layers: not as well suited to dense prediction, where we want to predict all of $y_{1:T}$
- Dilated convolutions: "Skips over" some past state / inputs

Outline

The two approaches to time series modeling

Self-attention and Transformers

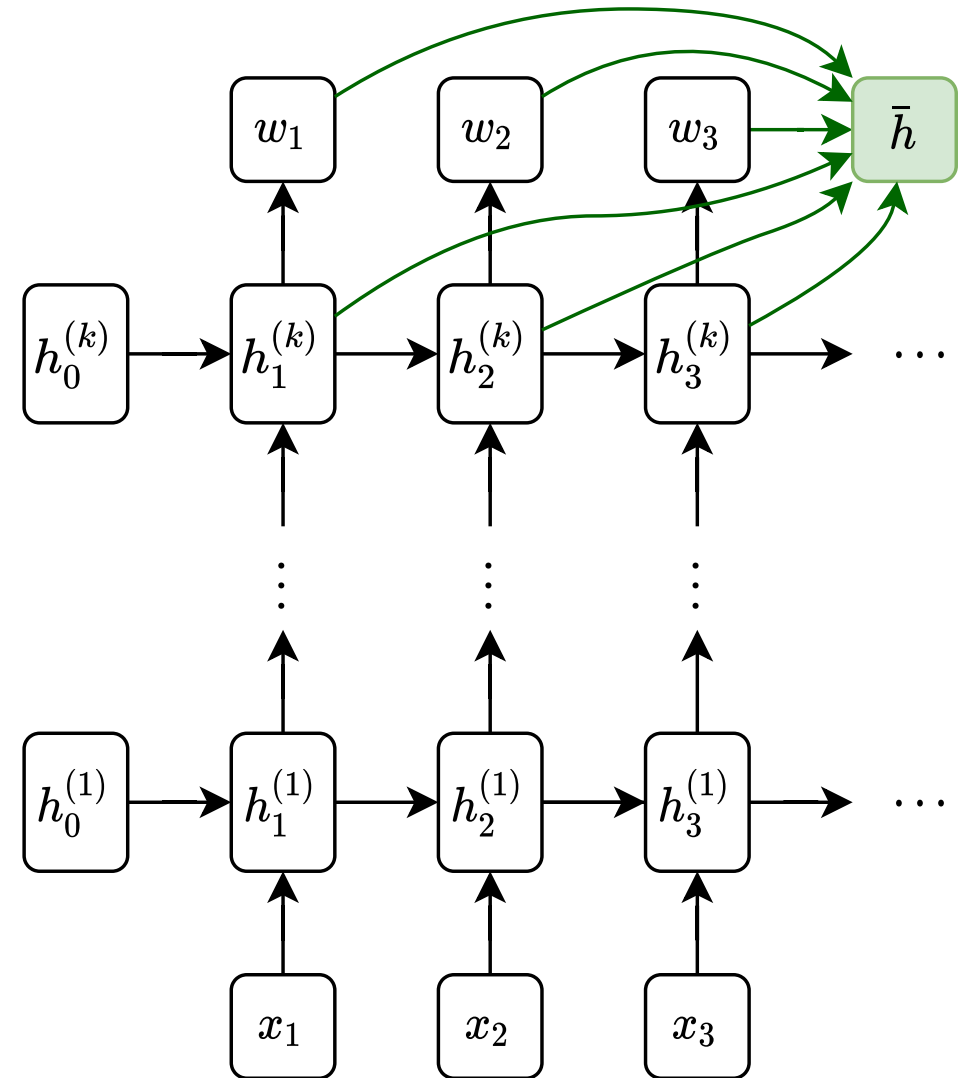Transformers beyond time series (very briefly)

"Attention" in deep learning

"Attention" in deep networks generally refers to any mechanism where individual states are *weighted and then combined*

$$z_t = \theta^T h_t^{(k)}$$
$$w = \text{softmax}(z)$$
$$\bar{h} = \sum_{t=1}^{T} w_t h_t^{(k)}$$

Used originally in RNNs when one wanted to combine latent states over all times in a more general manner than "just" looking at the last state

The self-attention operation

Self-attention refers to a particular form of attention mechanism

Given three inputs $Q, K, V \in \mathbb{R}^{T \times d}$ ("queries", "keys", "values", in one of the least-meaningful semantic designations we have in deep learning)

$$
Q = \begin{bmatrix} - & q_1^\top & - \\ - & q_2^\top & - \\ & \vdots & \\ - & q_T^\top & - \end{bmatrix}, \quad
K = \begin{bmatrix} - & k_1^\top & - \\ - & k_2^\top & - \\ & \vdots & \\ - & k_T^\top & - \end{bmatrix}, \quad
V = \begin{bmatrix} - & v_1^\top & - \\ - & v_2^\top & - \\ & \vdots & \\ - & v_T^\top & - \end{bmatrix}
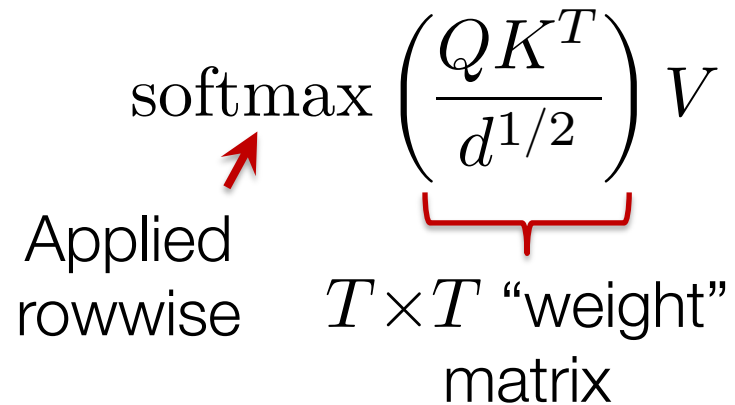$$

we define the self attention operation as

$$
\mathrm{SelfAttention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{d^{1/2}}\right) V
$$

Self-attention in more detail

$$\mathrm{softmax} \left(\frac{QK^T}{d^{1/2}} \right) V$$

Applied
rowwise

$T \times T$ "weight"
matrix

Properties of self-attention:

- Invariant (really, equivariant) to permutations of the Q, K, V matrices
- Allows influence between q_t, k_t, v_t over *all* times
- Compute cost is $O(T^2 + Td)$ (cannot be easily reduced due to nonlinearity applied to full $T \times T$ matrix)
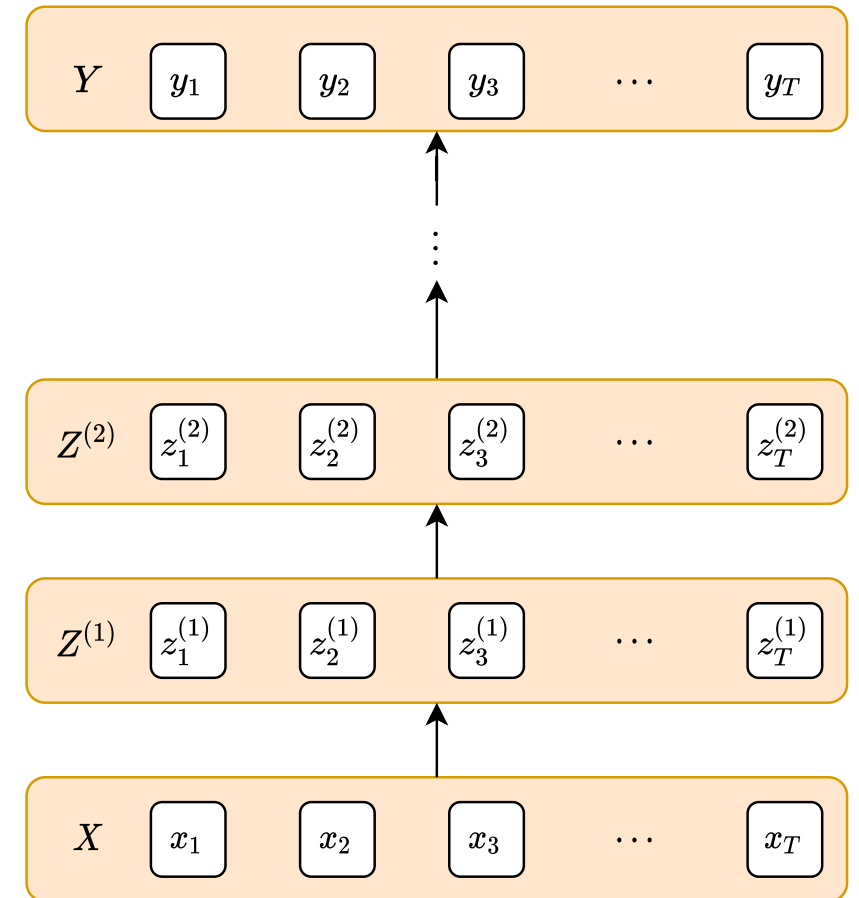
Transformers for time series

The Transformer architecture uses a series of attention mechanisms (and feedfoward layers) to process a time series

$$Z^{(i+1)} = \text{Transformer}(Z^{(i)})$$

(described in detail on next slide)

All time steps (in practice, within a given time slice) are processed in parallel, avoids the need for sequential processing as in RNNs

| Y | y_1 | y_2 | y_3 | $\cdots$ | y_T |

| $Z^{(2)}$ | $z_1^{(2)}$ | $z_2^{(2)}$ | $z_3^{(2)}$ | $\cdots$ | $z_T^{(2)}$ |

| $Z^{(1)}$ | $z_1^{(1)}$ | $z_2^{(1)}$ | $z_3^{(1)}$ | $\cdots$ | $z_T^{(1)}$ |

| X | x_1 | x_2 | x_3 | $\cdots$ | x_T |

Transformer block

In more detail, the Transformer block has the following form:

$$\tilde{Z} := \text{SelfAttention}(Z^{(i)}W_Q, Z^{(i)}W_K, Z^{(i)}W_V)$$

$$= \text{softmax}\left(\frac{Z^{(i)}W_Q W_K^T Z^{(i)T}}{d^{1/2}}\right) Z^{(i)}W_V$$

$$\tilde{Z} := \text{LayerNorm}\left(Z^{(i)} + \tilde{Z}\right)$$

$$Z^{(i+1)} := \text{LayerNorm}(\text{ReLU}(\tilde{Z}W_1)W_2 + \tilde{Z})$$

A bit complex, but really just self-attention, followed by a linear layer + ReLU, with residual connections and normalization thrown in somewhat arbitrarily (and precisely where these go are often adjusted)
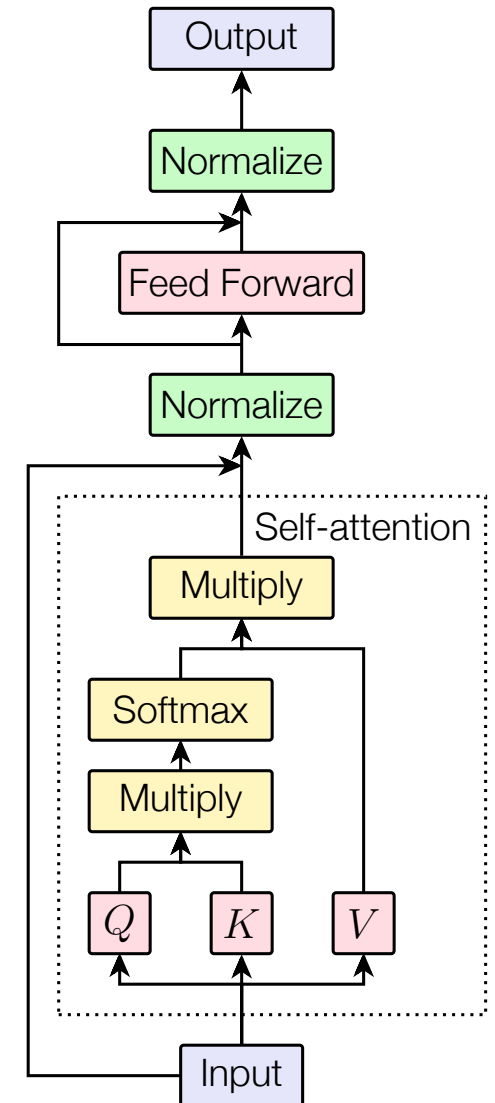
Transformers applied to time series

We can apply the Transformer block to the "direct" prediction method for time series, instead of using a convolutional block
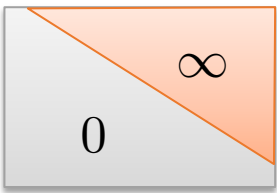
Pros:

- Full receptive field within a single layer (i.e., can immediately use past data)
- Mixing over time doesn't increase parameter count (unlike convolutions)

Cons:

- All outputs depend on all inputs (no good e.g., for autoregressive tasks)
- No ordering of data (remember that transformers are equivariant to permutations of the sequence)

Masked self-attention

To solve the problem of "acausal" dependencies, we can *mask* the softmax operator to assign zero weight to any "future" time steps

$$\text{softmax}\left(\frac{QK^T}{d^{1/2}} - M\right)V, \qquad M = \begin{bmatrix} \begin{array}{c} \infty \\ 0 \end{array} \end{bmatrix}$$

Note that even though technically this means we can "avoid" creating those entries in the attention matrix to being with, in practice it's often faster to just form them then mask them out (more on Monday)

Positional encodings

To solve the problem of "order invariance", we can add a **positional encoding** to the input, which associates each input with its position in the sequence

$$X \in \mathbb{R}^n = \begin{bmatrix} - & x_1^\top & - \\ - & x_2^\top & - \\ & \vdots & \\ - & x_T^\top & - \end{bmatrix} + \begin{bmatrix} \sin(\omega_1 \cdot 1) & \cdots & \sin(\omega_n \cdot 1) \\ \sin(\omega_1 \cdot 2) & \cdots & \sin(\omega_n \cdot 2) \\ \vdots & \ddots & \vdots \\ \sin(\omega_1 \cdot T) & \cdots & \sin(\omega_n \cdot T) \end{bmatrix}$$

and where $\omega_i, i = 1, \ldots n$ is typically chosen according to a logarithmic schedule

(Really, add positional encoding to d-dimensional projection of X)

Outline

The two approaches to time series modeling

Self-attention and Transformers

Transformers beyond time series (very briefly)

Transformers beyond time series

Recent work has observed that transformer blocks are extremely powerful beyond just time series

- Vision Transformers: Apply transformer to image (represented by a collection of patch embeddings), works better than CNNs for large data sets
- Graph Transformers: Capture graph structure in the attention matrix

In all cases, some challenges are:

- How to represent data such that $O(T^2)$ operations are feasible
- How to form positional embeddings
- How to form the mask matrix