Time Series Forecasting: Advanced

Jaemin Yoo

School of Electrical Engineering

Kim Jaechul Graduate School of AI

Outline

1. **State space models**
2. From linear to deep models
3. Convolutional neural networks
4. Encoder-decoder structure
5. Summary

Error-Correction Function

- **Q:** What are alternatives to linear regression models?
- **Idea:** Create a prediction by improving the previous prediction:

$$\hat{z}_{t+1} = \underbrace{\hat{z}_t}_{\text{previous forecast}} + \alpha \underbrace{(z_t - \hat{z}_t)}_{\text{error in previous forecast}},$$

  - Also known as an **error-correction function**.
  - Since it corrects the error caused from the previous forecast.
  - α is called a smoothing parameter.

Simple Exponential Smoothing

- We can rewrite the error-correction function as follows:

$$
\begin{aligned}
\hat{z}_3 &= \hat{z}_2 + \alpha(z_2 - \hat{z}_2) \\
&= \alpha z_2 + (1-\alpha)\hat{z}_2 \\
&= \alpha z_2 + \alpha(1-\alpha)z_1 + (1-\alpha)^2\hat{z}_1
\end{aligned}
$$

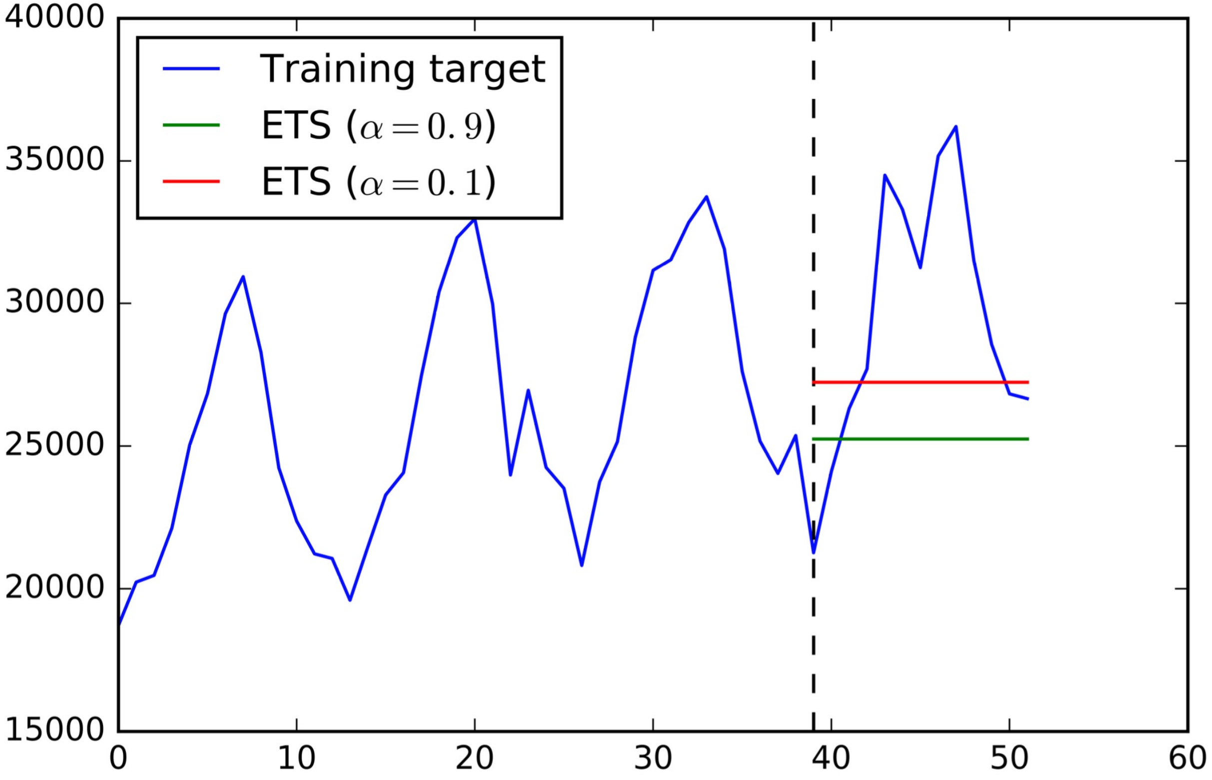
…

- The model is called **ETS** (Simple **E**xponen**T**ial **S**moothing):

$$\widehat{z}_{T+h} = \alpha z_T + \alpha(1-\alpha)z_{T-1} + \alpha(1-\alpha)^2 z_{T-2} + \cdots + (1-\alpha)^T \widehat{z}_1$$

Simple Exponential Smoothing

- Larger α puts more emphasis on the recent observations.

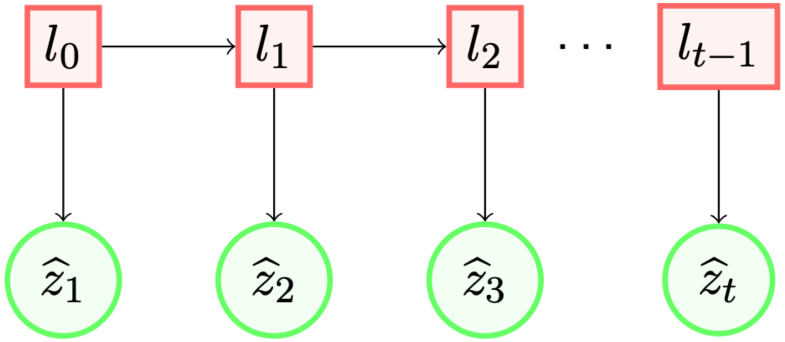


State and Prediction

- Let’s separate ETS into two parts: **state** and **prediction**.
  - l_t is a state at time t.
  - l_t is used to create the prediction $\hat{z}_t$ at time t.
  - l_t is updated to l_{t+1} for the next time step.

Forecast equation $\quad \widehat{z}_t = \ell_{t-1}$

Error Correction $\quad \ell_t = \ell_{t-1} + \alpha(z_t - \widehat{z}_t),$



General Exponential Smoothing

- **Idea:** Let's make the state l_t contain more information.

Forecast equation	$\widehat{z}_t = \boldsymbol{a}_t^T \boldsymbol{l}_{t-1}$
Error Correction	$\boldsymbol{l}_t = \boldsymbol{F}_t \boldsymbol{l}_{t-1} + \boldsymbol{g}_t (z_t - \widehat{z}_t),$

  - $\boldsymbol{l}_t$ is now a vector, and is not equivalent to the prediction $\hat{z}_t$.
  - Parameter $\boldsymbol{a}_t$ maps $\boldsymbol{l}_t$ to $\hat{z}_t$ through the dot product.
  - Parameters $\boldsymbol{F}_t$ and $\boldsymbol{g}_t$ are used to update $\boldsymbol{l}_t$ to $\boldsymbol{l}_{t+1}$.

State Space Models

- **Q:** Do we really need the error-correction part $z_t - \hat{z}_t$?
  - Maybe not. Let's model the error $z_t - \hat{z}_t$ as a random variable ϵ_t.
- **State space model (SSM)** simplifies the previous model.
  - Consist of the measurements and state transition parts.
  - Add white noise to both parts, replacing the error-correction function.

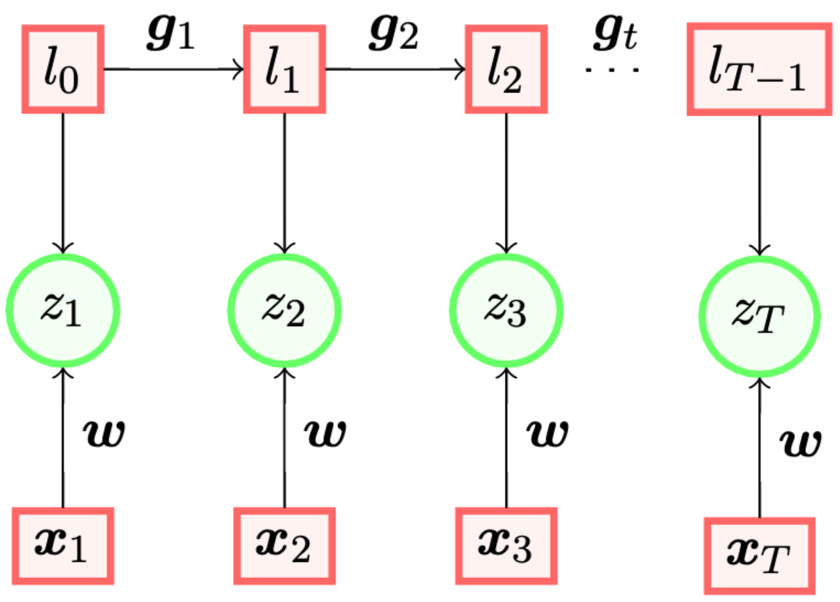
Measurements $\quad z_t = \boldsymbol{a}_t^T \boldsymbol{l}_{t-1} + \epsilon_t, \quad \epsilon_t \sim N(0, \sigma^2)$

State transition $\quad \boldsymbol{l}_t = \boldsymbol{F}_t \boldsymbol{l}_{t-1} + \boldsymbol{g}_t \epsilon_t, \quad \boldsymbol{l}_0 \sim N(\boldsymbol{\mu}_0, \mathrm{diag}(\boldsymbol{\sigma}_0^2)).$

Linear State Space Models

- Let's combine the SSM with (feature-based) linear regression:

$$z_t \sim P(z_t | y_t)$$
$$y_t = \boldsymbol{a}_t^T \boldsymbol{l}_{t-1} + \boldsymbol{w}^T \boldsymbol{x}_t$$
$$\boldsymbol{l}_t = \boldsymbol{F}_t \boldsymbol{l}_{t-1} + \boldsymbol{g}_t \epsilon_t$$



- **Pros:** Show the strength of both models.
- **Cons:** More parameters to learn.

Linear State Space Models

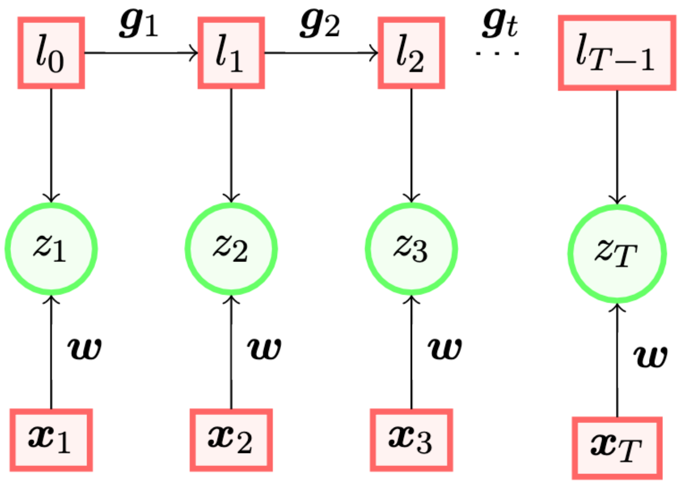
Linear State Space Model part:

$$u_t = \boldsymbol{a}_t^T \boldsymbol{l}_{t-1}$$

Feature-based part:

$$b_t = \boldsymbol{w}^T \boldsymbol{x}_t$$

Probabilistic model for data (likelihood): $z_t \sim P(z_t | u_t + b_t)$

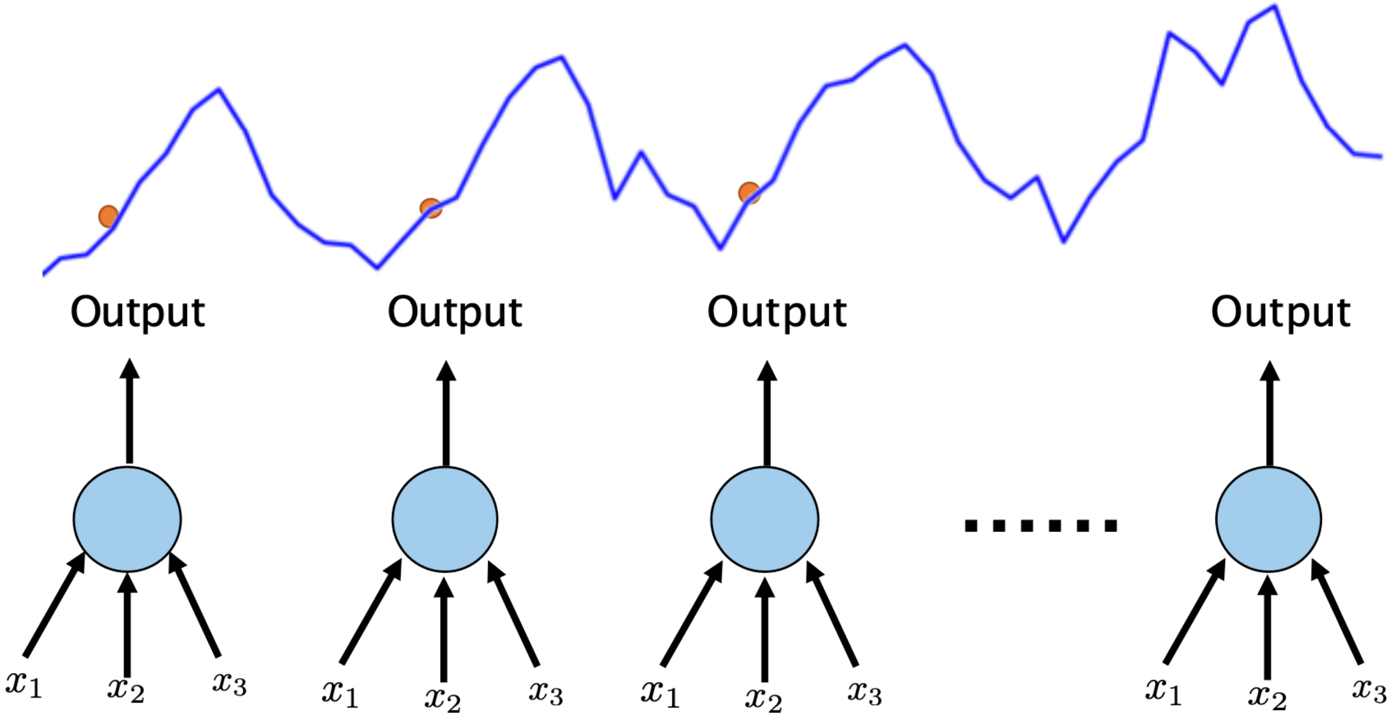


Outline

1. State space models
2. **From linear to deep models**
3. Convolutional neural networks
4. Encoder-decoder structure
5. Summary

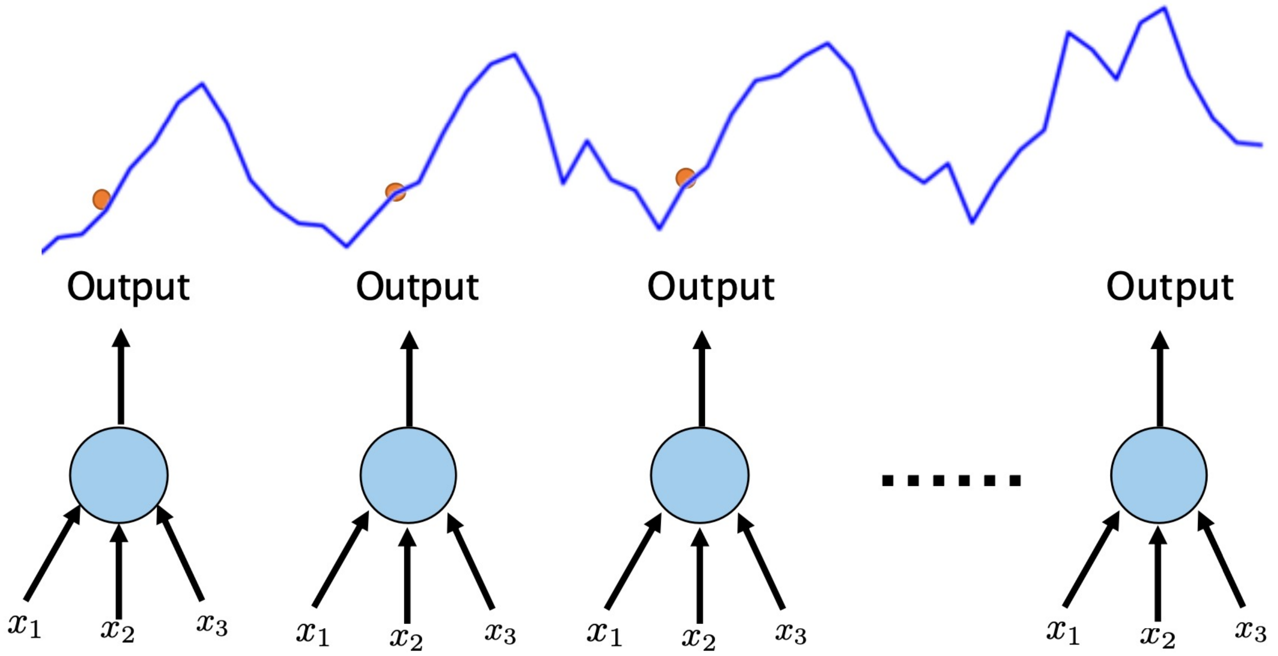
Generalizing Linear Regression

- Recall that **linear regression** works as $z_t = w^\top x_t$ for forecasting.



Generalizing Linear Regression

- We can generalize the linear mapping using a **deep neural network.**



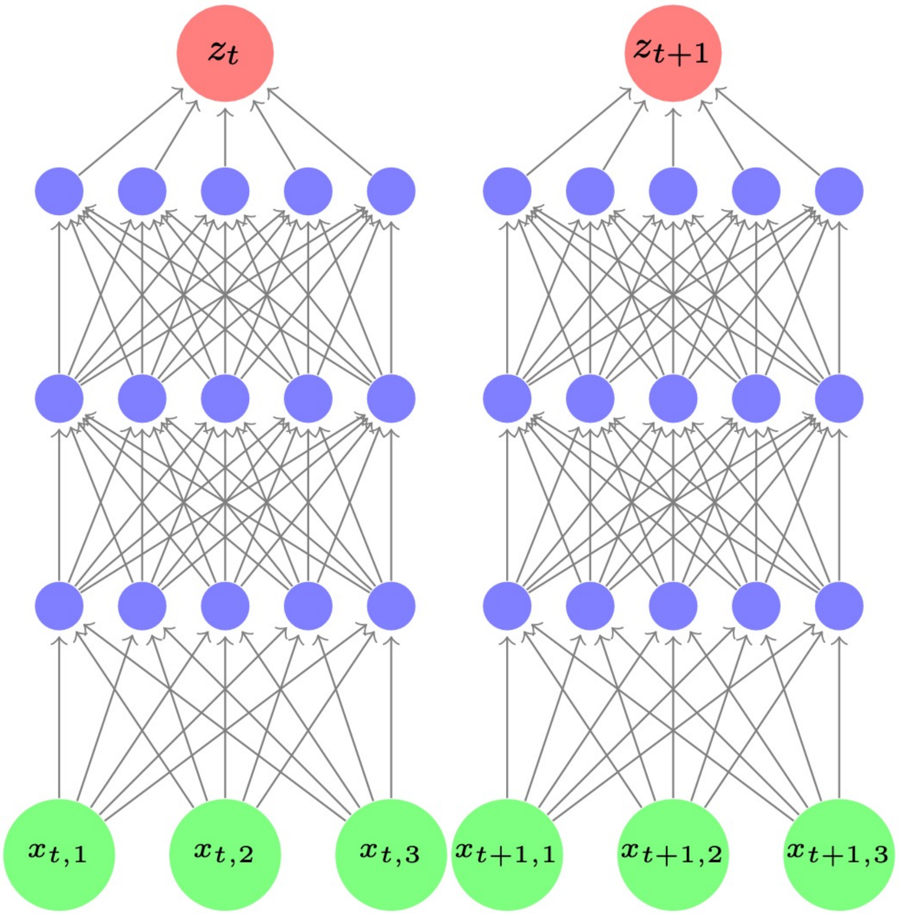
$$z_t = \sigma(\boldsymbol{w}_l^T(\sigma(W_{l-1}^T(\sigma(W_{l-2}^T(\cdots W_0^T \boldsymbol{x}_t))))) := \text{DEEP-NET}(\boldsymbol{x}_t)$$

Multi-layer Perceptrons

- **Multi-layer perceptrons (MLPs):**
  - The most basic deep learning architecture.
  - Each neuron in a hidden layer computes an affine function of the previous layer.
  - It is then followed by an *activation function*:

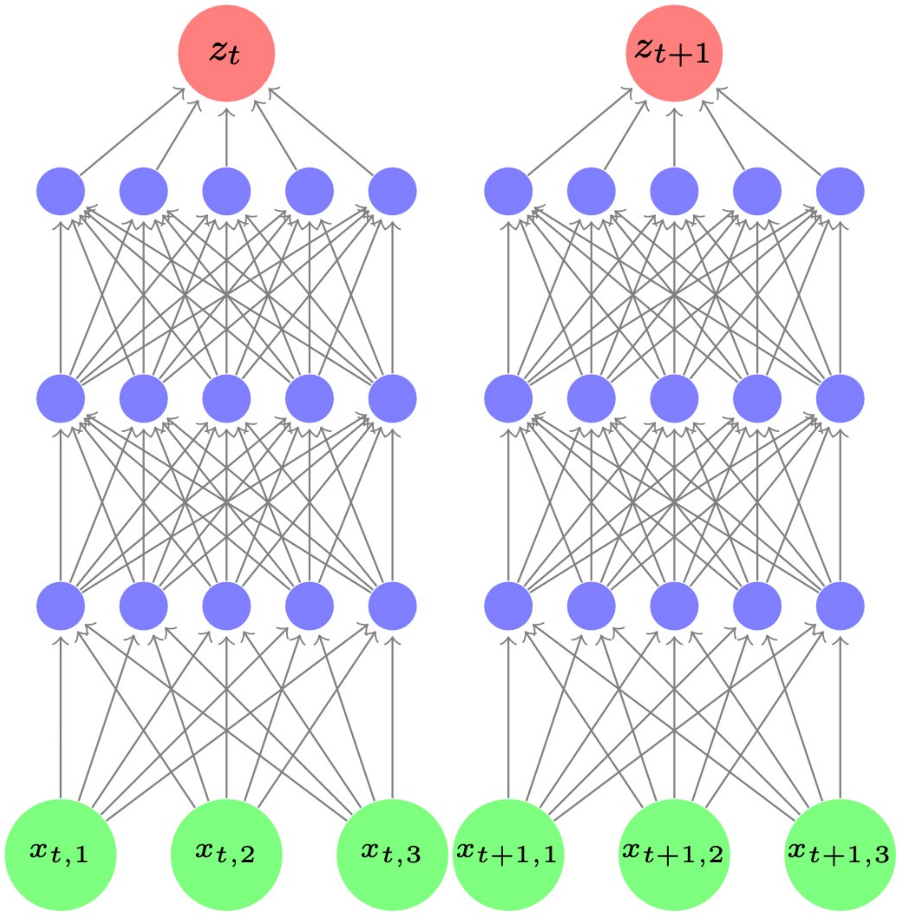
$$h_{l,j} = \sigma(\mathbf{w}_{l,j}^{\top}\mathbf{h}_{l-1} + b_{l,j}).$$

- MLPs are flexible function estimators.
  - More layers → more complex functions.



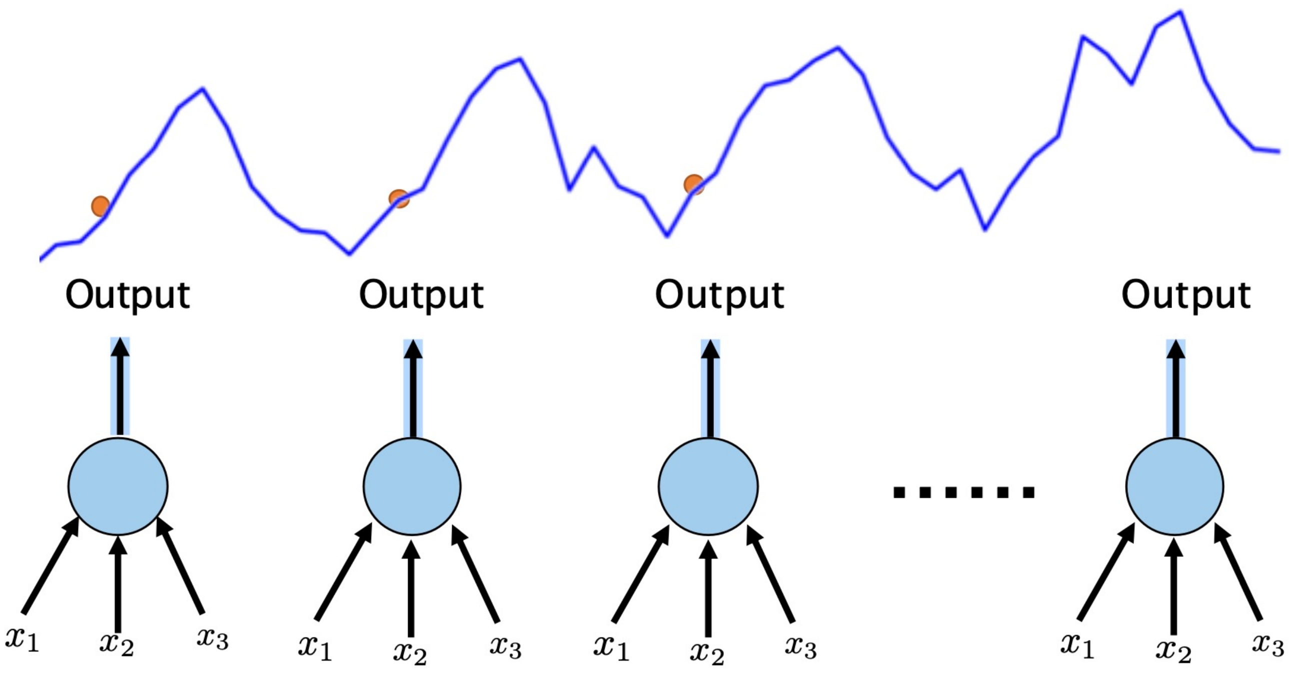
Multi-layer Perceptrons

- **Advantages:** MLPs can learn complex input-output relationships.
  → Less manual feature engineering.
- **Disadvantages:** More data are needed.
  - Careful tuning (e.g., regularization, learning rate, etc.) is necessary.
  - The model is sensitive to scaling of inputs.



Recap: MLPs for Forecasting

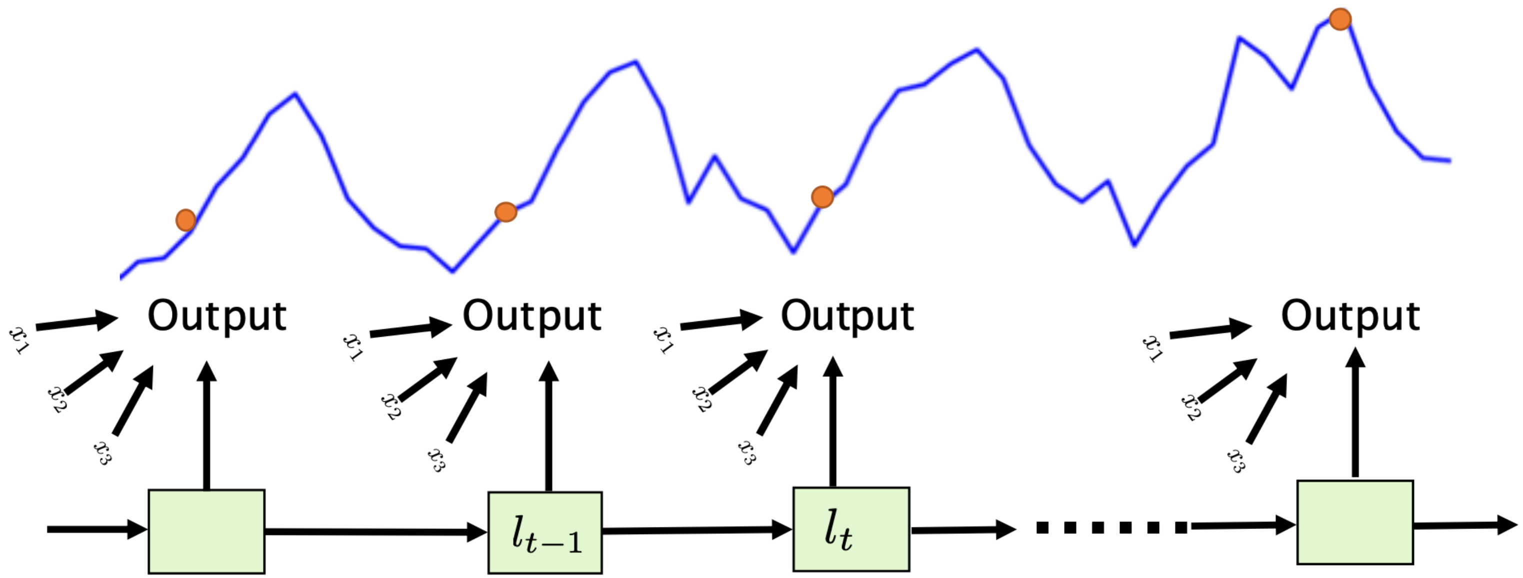
- **Question:** How can we model the sequential relationship?



$$z_t = \text{DEEP-NET}(\boldsymbol{x}_t)$$

Recap: State Space Models

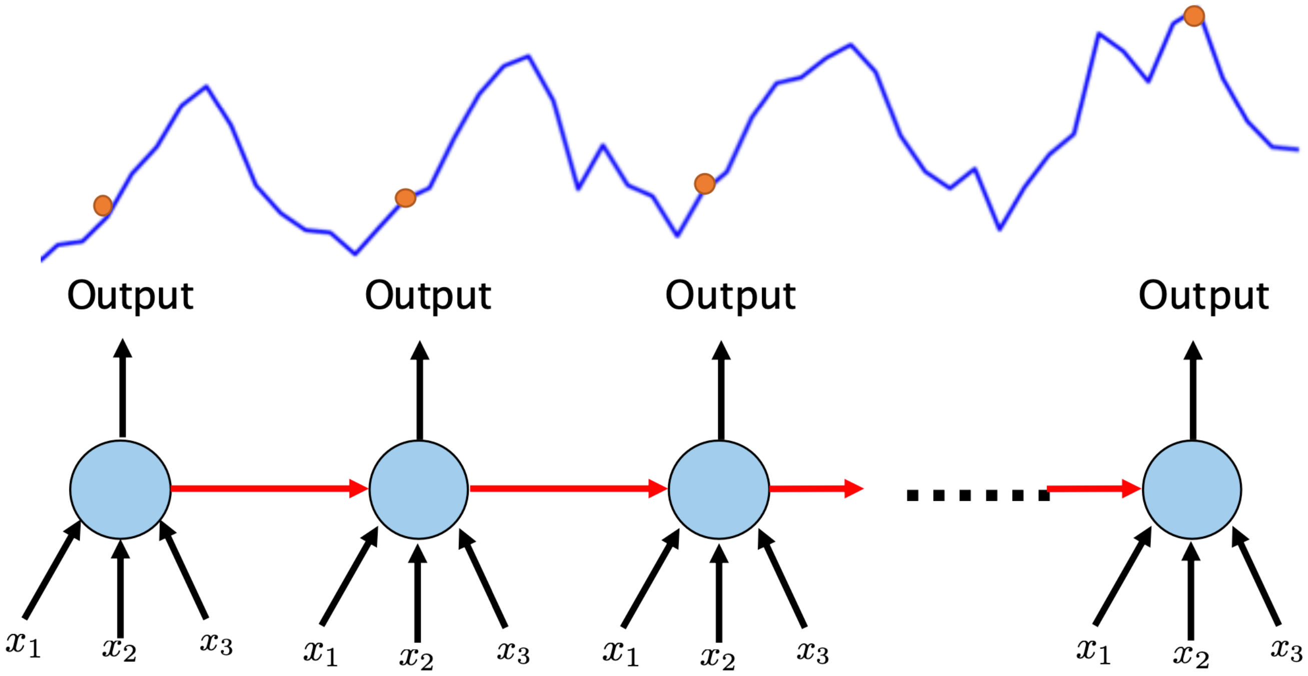
- **Question:** Can we do the same with neural networks?



$$l_t = l_{t-1} + \alpha \cdot \epsilon_t$$
$$z_t = \boldsymbol{v}^T \boldsymbol{l}_t + \boldsymbol{w}^T \boldsymbol{x}_t + \epsilon_t$$

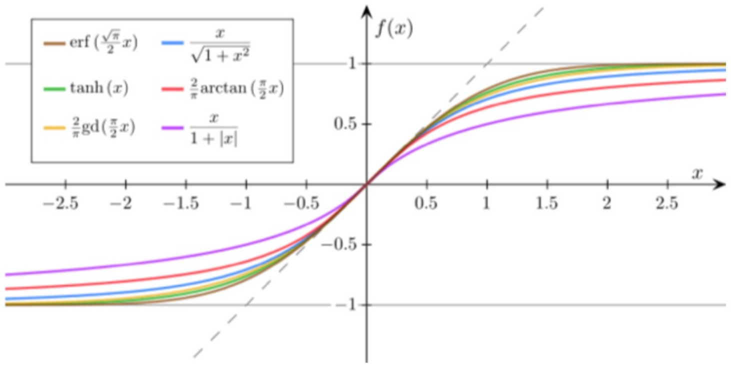
From Feed-forward to Recurrent Models

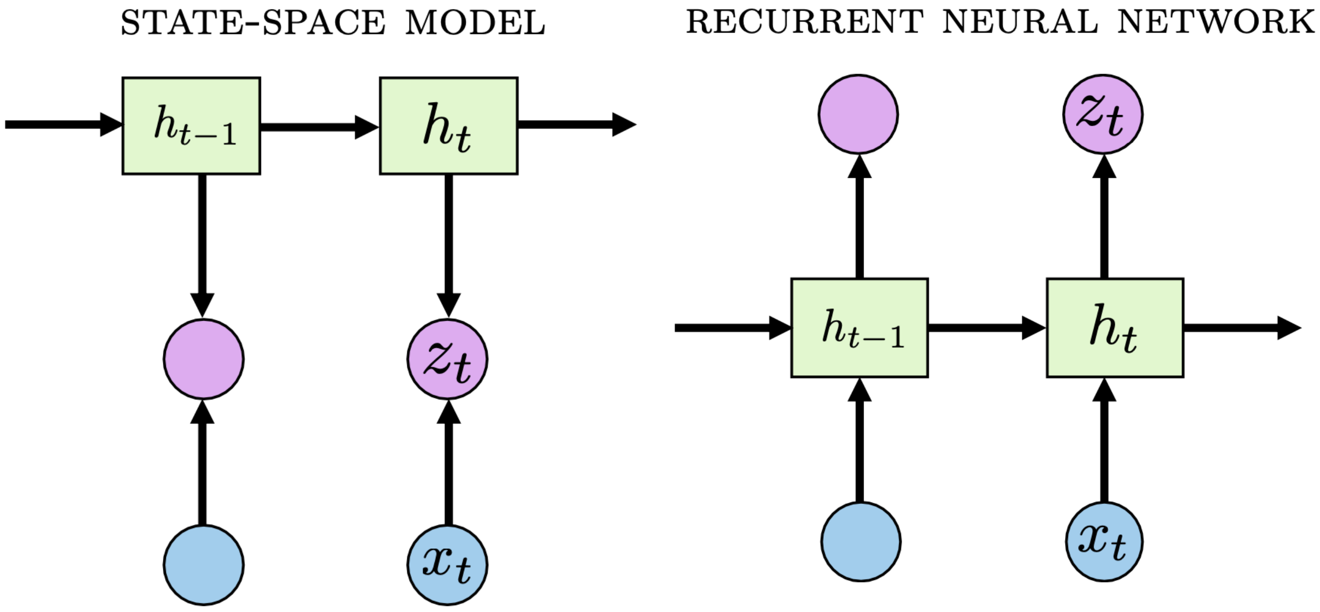
- We add the concept of **state** to the deep forecasting model.
  - The previous predictions affect the current one.



Toward Recurrent Neural Networks

- **Recurrent neural networks:**
  - Current state h_t combines
    - Previous state h_{t-1}.
    - Input features x_t.
    - Activation function σ.



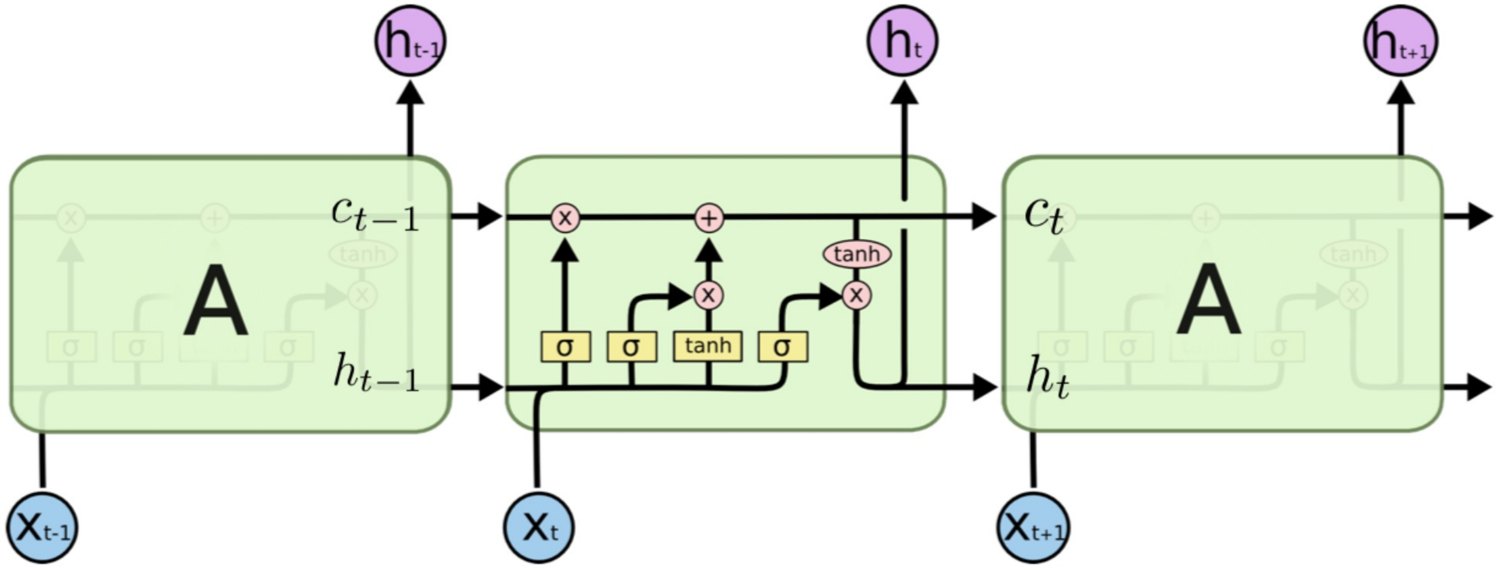


$$\boldsymbol{l}_t = \boldsymbol{l}_{t-1} + \alpha \cdot \epsilon_t$$
$$z_t = \boldsymbol{v}^T \boldsymbol{l}_t + \boldsymbol{w}^T \boldsymbol{x}_t + \epsilon_t$$

$$\boldsymbol{h}_t = \sigma(\theta_0 h_{t-1} + \theta_1 x_t)$$
$$z_t = \sigma(\theta h_t)$$

Long Short-Term Memory (LSTM)

- **LSTM** uses two states C_t and h_t with a **forget gate**:
  - The forget gate is similar to the exponential smoothing from ETS.



HTTP://COLAH.GITHUB.IO/POSTS/2015-08-UNDERSTANDING-LSTMS/

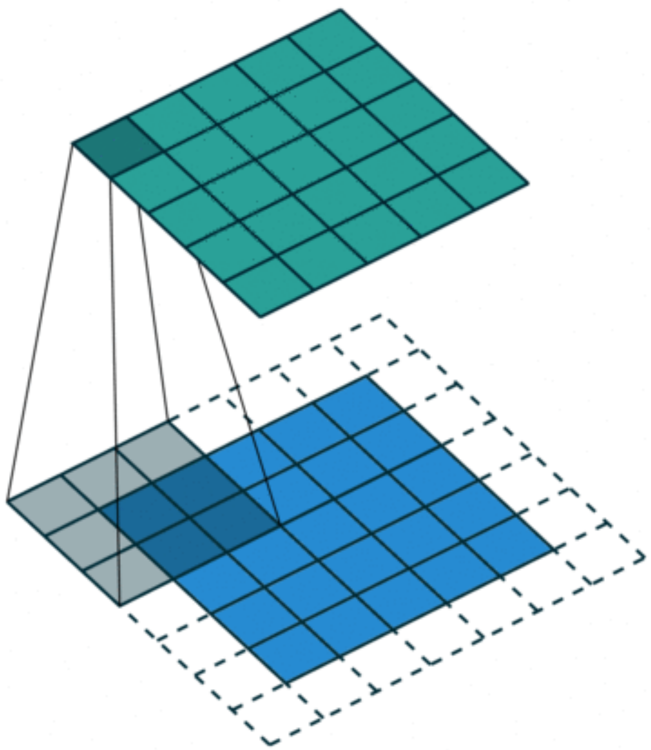
$$C_t = \alpha_t \cdot C_{t-1} + \beta_t \times \sigma(\theta_0 h_{t-1} + \theta_1 x_t)$$

Outline

1. State space models
2. From linear to deep models
3. **Convolutional neural networks**
4. Encoder-decoder structure
5. Summary

Convolutional Neural Networks

- **Convolutional neural networks (CNNs):**
  - Neural networks that use convolutional layers.
  - CNNs with 2D convolutions are successful in CV.
    - The idea is to encode *spatial invariance*.
  - 1D convolutions are a promising alternative to RNNs for sequential data.
    - Encode temporal invariance, e.g., stationarity.
    - Often more lightweight and effective than RNNs.

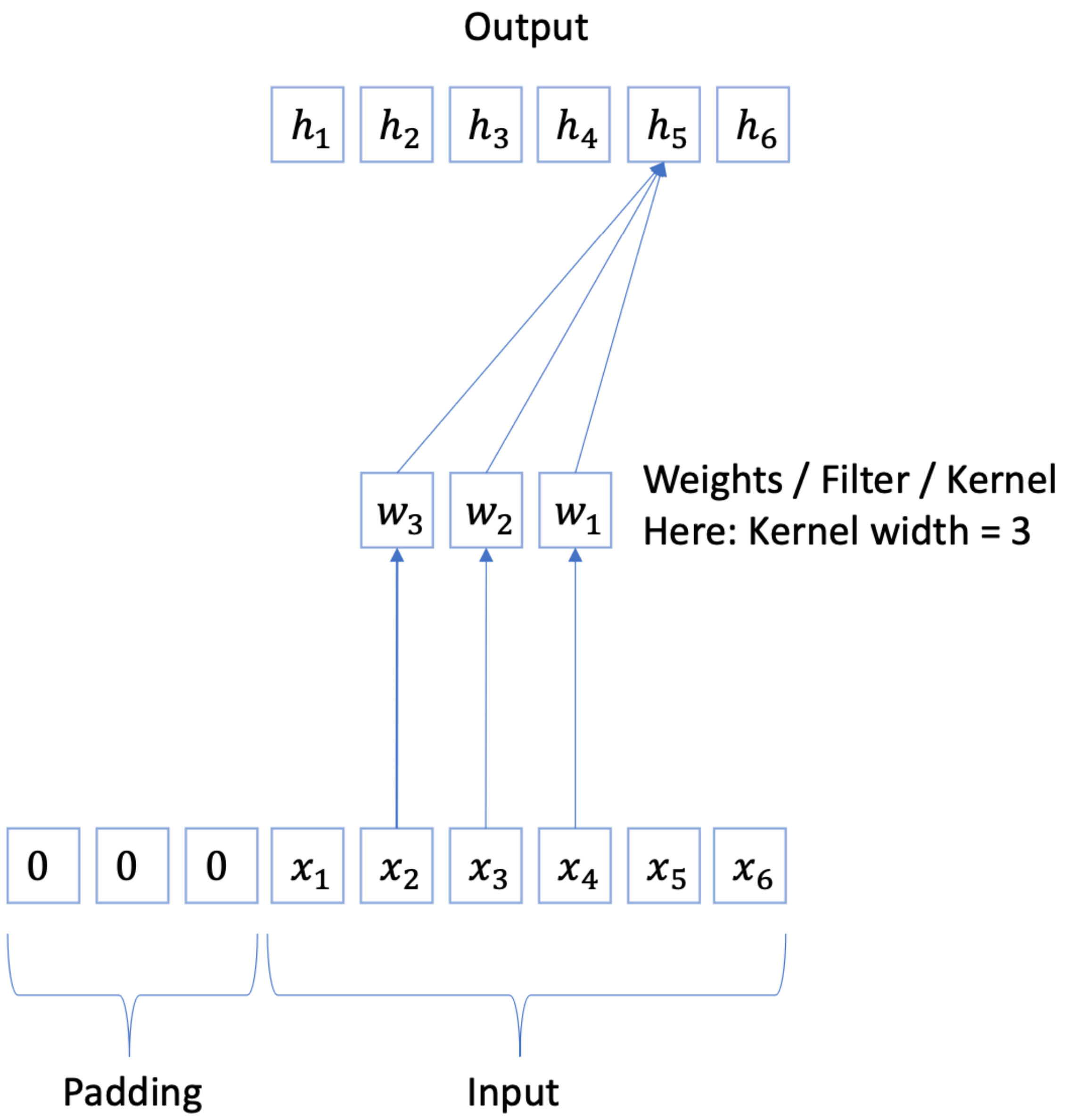


Convolutional Layers

- The output h_j in a convolution layer is a discrete convolution of the inputs $\mathbf{x}$ with weights $\mathbf{w}$.
- For a 1-dimensional convolution with a kernel with width D:

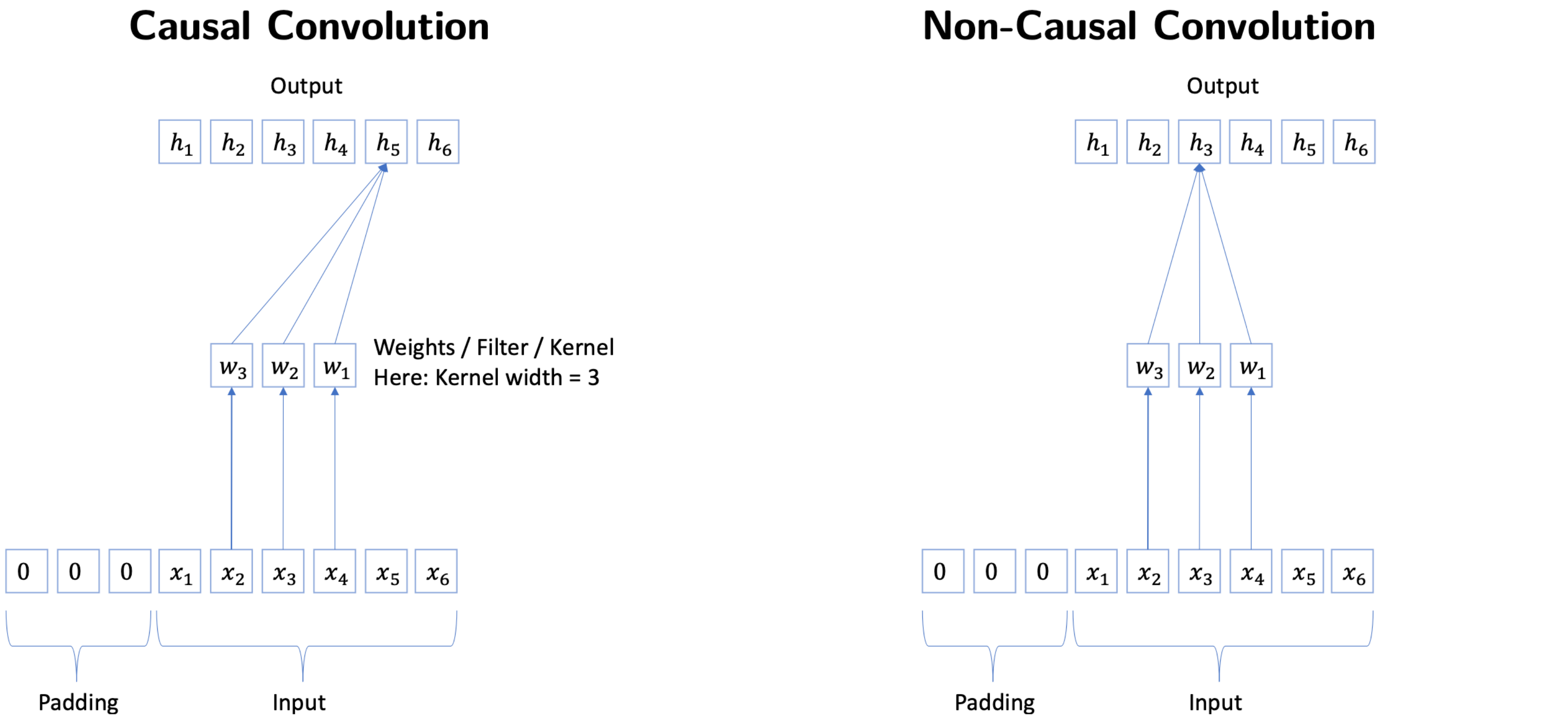
$$h_j = \sum_{d=1}^{D} w_d x_{j-d}.$$

- *Padding* is usually added to the first part of the sequence.

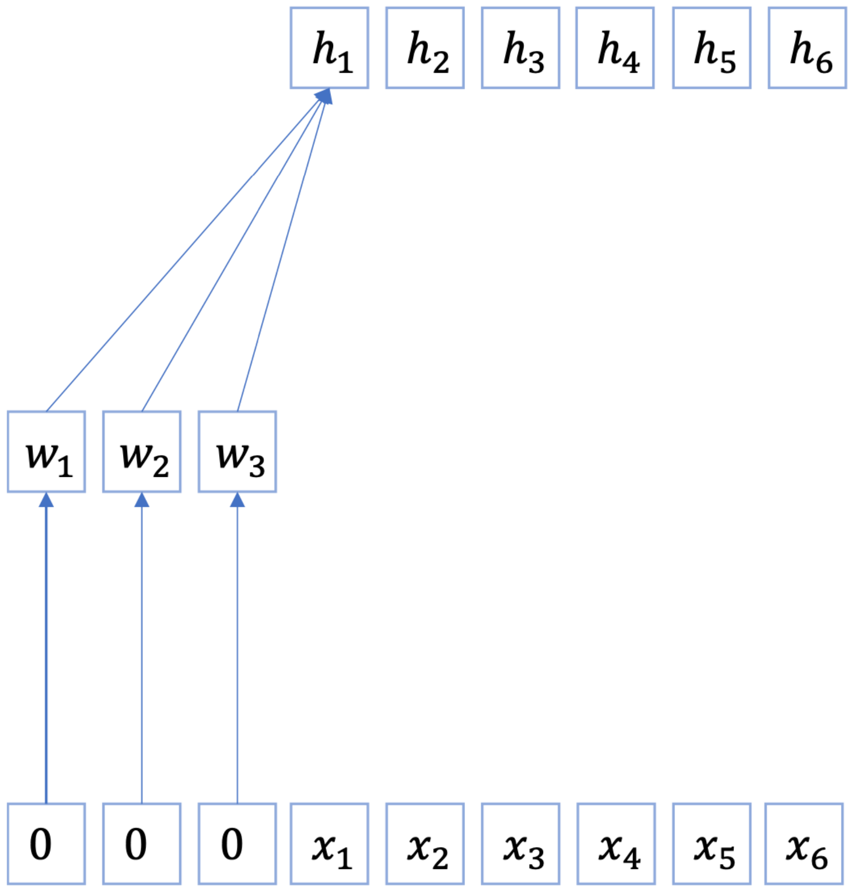


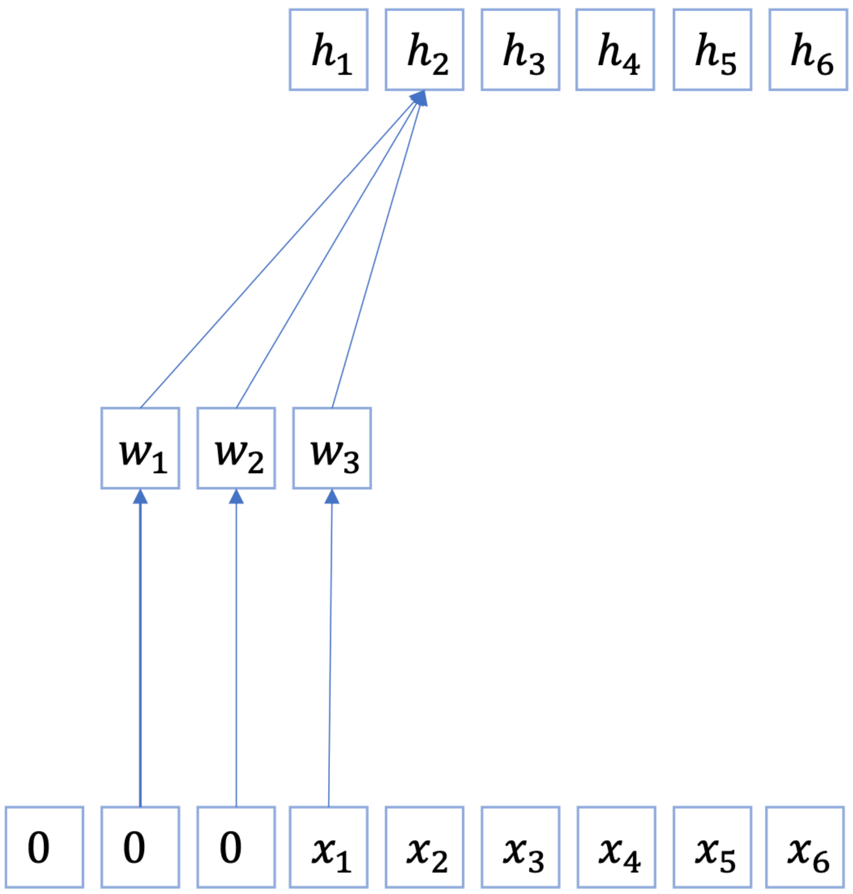
Causal vs. Non-causal Convolution

- Non-causal convolution is used mostly for timeseries-level tasks.



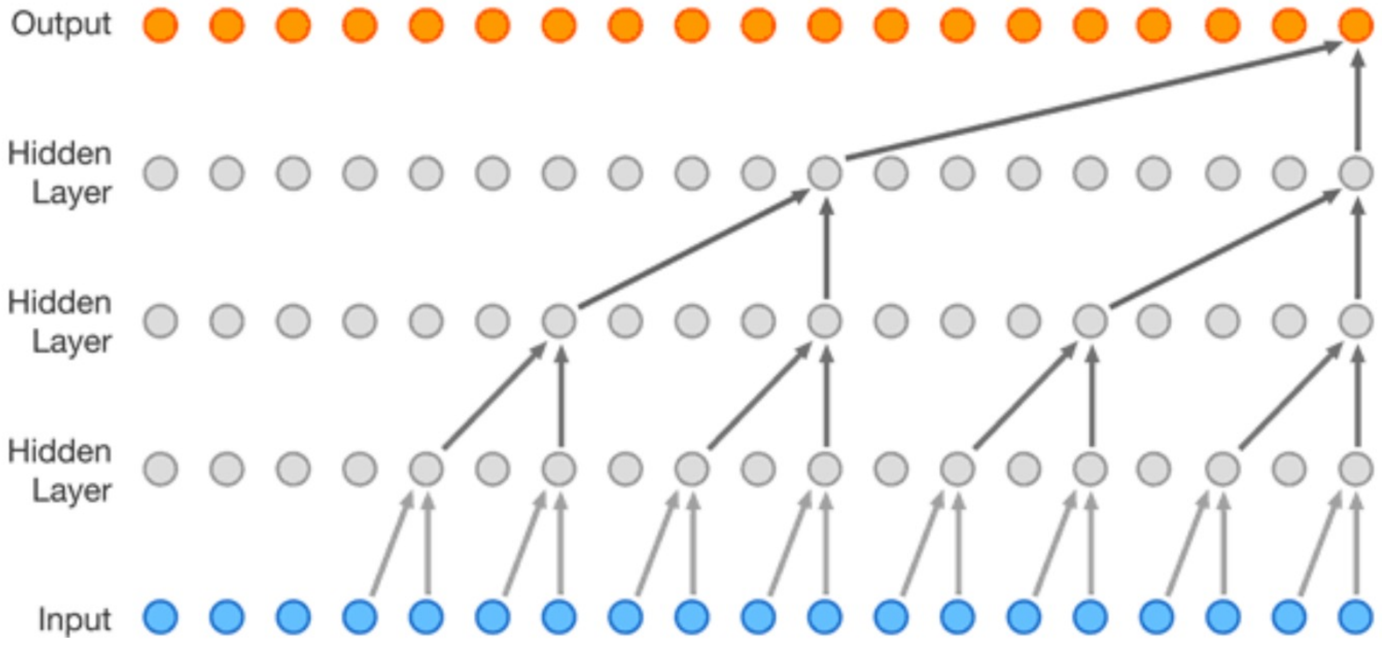
1D Causal Convolution





Canonical Models: Dilated Convolution

- **Dilation** quickly increases receptive field through multiple layers.
  - Forecast is generated in an autoregressive fashion.



Outline

1. State space models
2. From linear to deep models
3. Convolutional neural networks
4. **<u>Encoder-decoder structure</u>**
5. Summary

Encoder-Decoder Structure

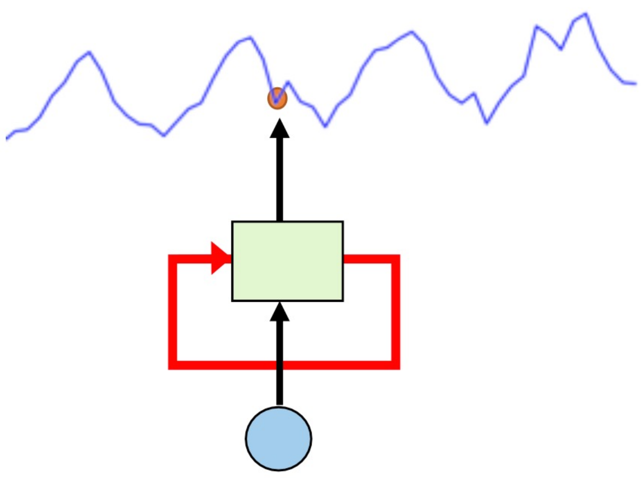
- Many deep forecasting models rely on additional features $\boldsymbol{X}$.
- **Q:** What if $\boldsymbol{X}$ is given only for the current T, not for the future?
  - Autoregressive prediction is no longer possible.
- **Solution:** Generalize the model into **encoder-decoder** structure.

Encoder-Decoder Structure

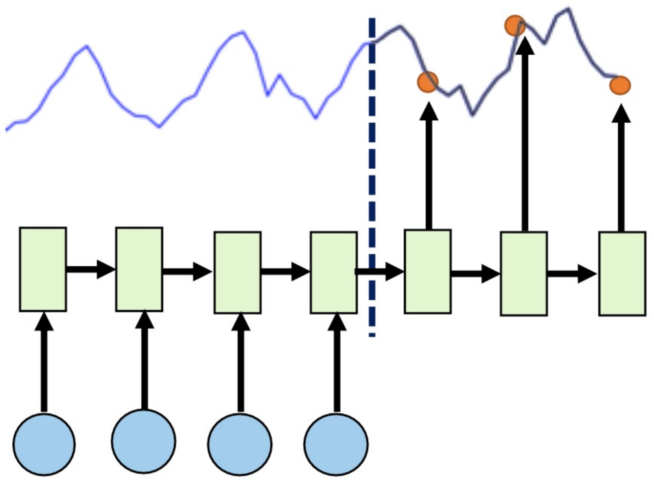
- **Idea:** Our prediction model consists of an **encoder** and a **decoder**.
  - **Encoder** takes $\boldsymbol{z}_{1:T}$ and $\boldsymbol{X}_{1:T}$ and summarizes them as a state $\boldsymbol{h}_T$.
  - **Decoder** takes $\boldsymbol{h}_T$ and generates predictions without more input.
- We choose a linear mapping $\hat{y} = \boldsymbol{W}\boldsymbol{h}_T$ as a decoder in many cases.
  - Meaning that we map a representation $\boldsymbol{h}_T$ into a scalar prediction.

Encoder-Decoder Structure

- What if we use a sequential model (e.g., an RNN) as a decoder?
- We can generate a sequence, from $\boldsymbol{h}_T$, without further input!



Canonical (One-to-One)



Seq2Seq (Many-to-Many)

Encoder Part

- Encoder is a general representation learner of time series.
  - That is, the encoded output $\boldsymbol{h}_T$ can be used for other tasks as well.
- Encoder is the same in one-to-one and many-to-many cases.
  - **One-to-one:** The output $\boldsymbol{h}_T = f_{\text{encoder}}(\cdot)$ is used to predict z_{T+1}.
  - **Seq2seq:** The output $\boldsymbol{h}_T = f_{\text{encoder}}(\cdot)$ is used to predict z_{T+1}: z_{T+h}.

$$f_{encoder} : \{z_1, \cdots, z_{T_e}\} \mapsto \boldsymbol{h}_{T_e}$$

$$f_{decoder} : \boldsymbol{h}_{T_e} \mapsto \{z_{T_e+1}, \cdots, z_{T_e+T_d}\}$$

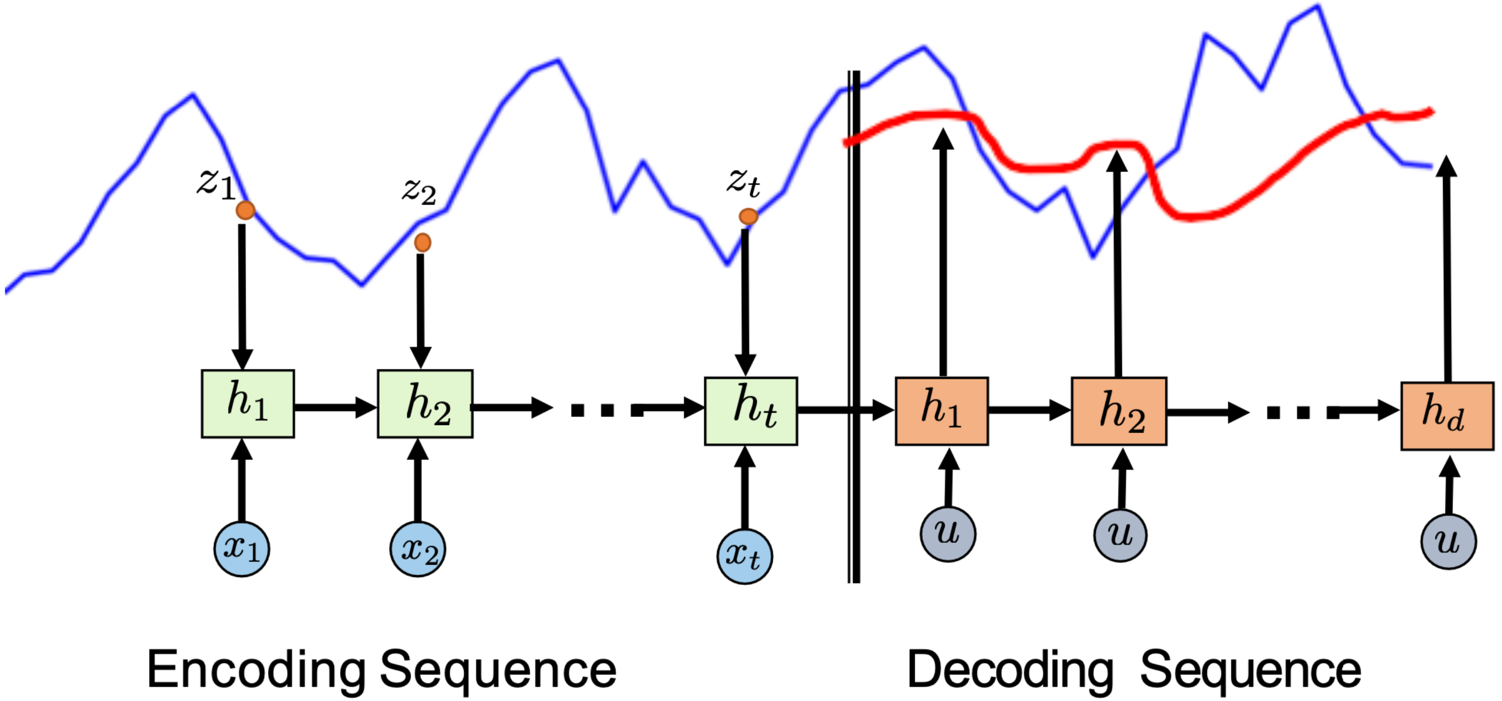
Decoder Part

- Decoder should be able to work in an autoregressive way.
  - That is, it should create a sequence without any input.
- Possible decoder models:
  - MLP with h output neurons; it creates h outputs at the same time.
  - RNN that assumes dummy (= meaningless) inputs.

$$f_{encoder} : \{z_1, \cdots, z_{T_e}\} \mapsto \boldsymbol{h}_{T_e}$$
$$f_{decoder} : \boldsymbol{h}_{T_e} \mapsto \{z_{T_e+1}, \cdots, z_{T_e+T_d}\}$$

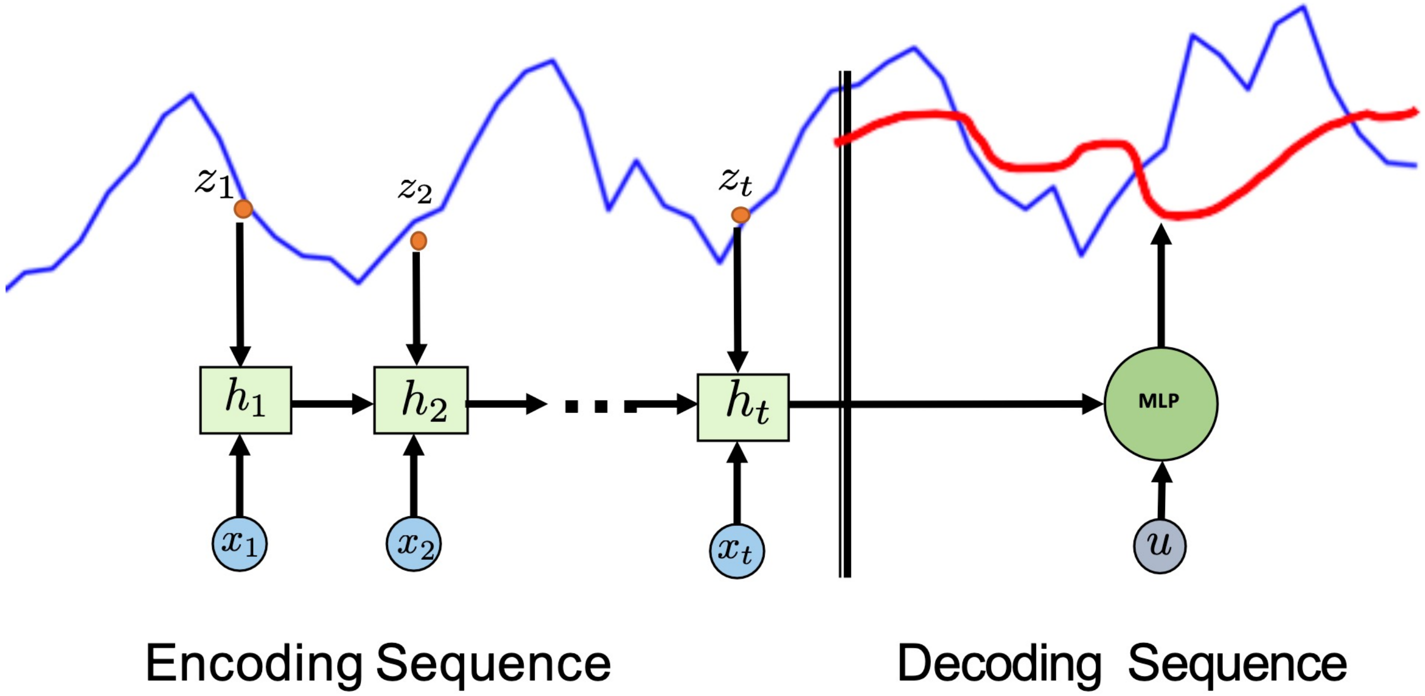
Example: RNN-RNN

- We can use an RNN as both an encoder and a decoder.
  - Decoder RNN uses a dummy input u.



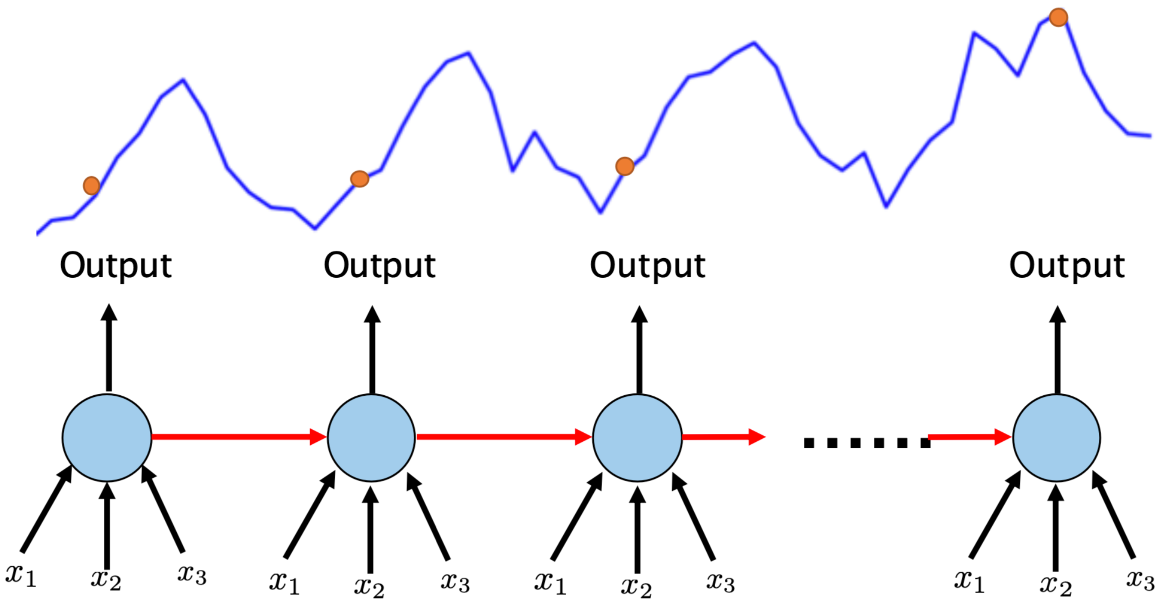
Example: RNN-MLP

- We can combine different model structures as well.
  - Decoder MLP generates a sequence altogether.



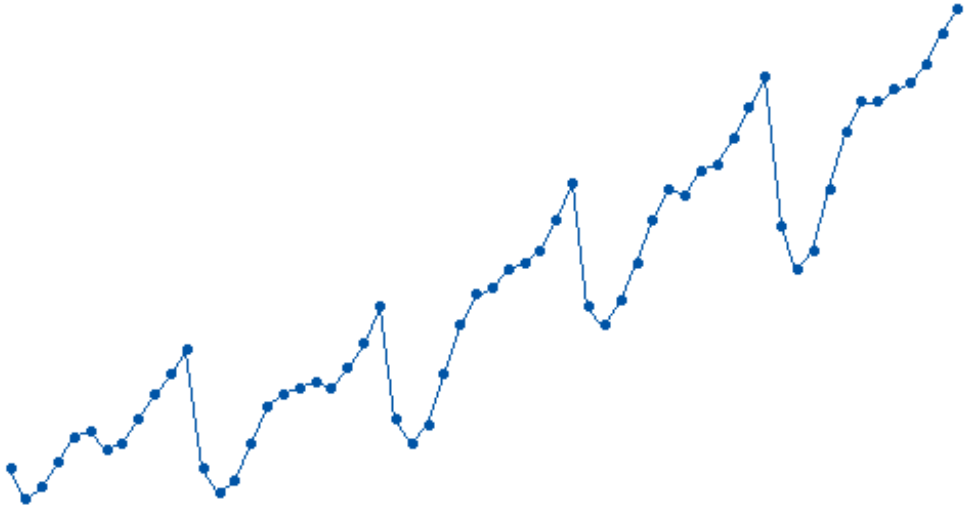
Attention RNN

- Suppose we use an RNN encoder in the Seq2Seq structure.
- **Natural approach:** We pass the last state $\boldsymbol{h}_T$ to the decoder.
  - Why? The last cell creates a good summary of all observations.



Attention RNN

- **Limitation 1:** The encoder is likely to forget early observations.
  - Especially when the window size is large.
- **Limitation 2:** Single state $\boldsymbol{h}_T$ may not be enough for the decoder.
  - Decoder needs different information at different locations.



Attention RNN

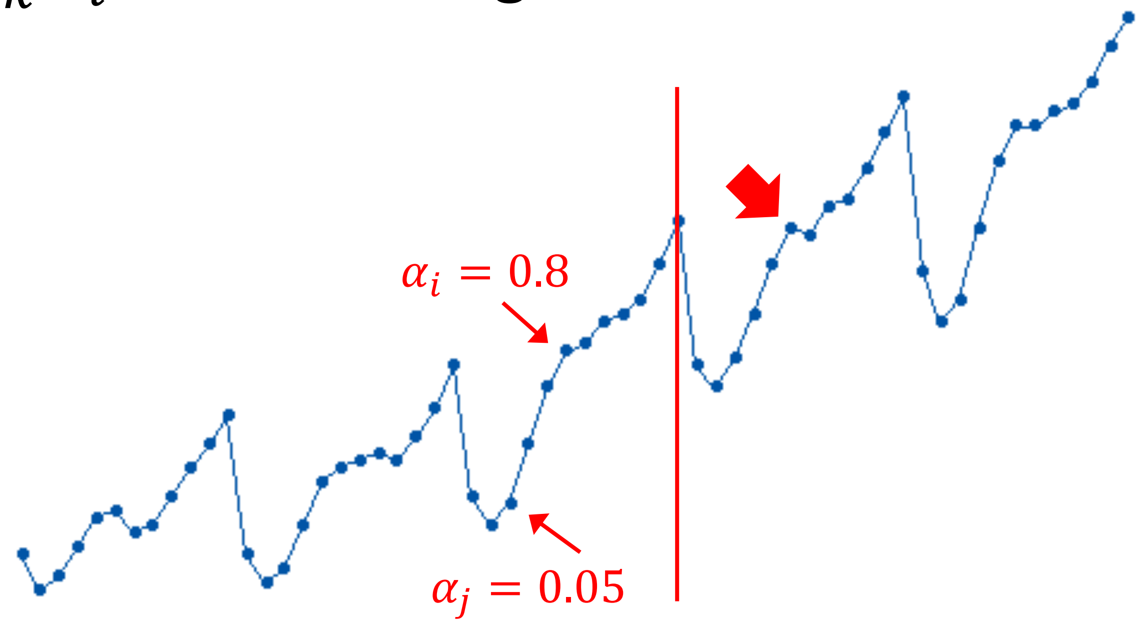
- **Solution:** Attention mechanism.
  - Let $\boldsymbol{h}_1, \cdots, \boldsymbol{h}_T$ be the state vectors created from RNN cells.
  - Let $\boldsymbol{h}_{T+k}$ be the state vector for the k-th decoding step.
  - Create a weighting function f such that

$$f(\boldsymbol{h}_{T+k}, \boldsymbol{h}_1, \cdots, \boldsymbol{h}_T) = \frac{\exp\left(\boldsymbol{h}_{T+k}^\top \boldsymbol{h}_i\right)}{\sum_{i=1}^{T} \exp\left(\boldsymbol{h}_{T+k}^\top \boldsymbol{h}_i\right)}.$$

    - $\boldsymbol{h}_{T+k}$ is the query of attention; it determines which i is more important.
    - $\boldsymbol{h}_i$ is a key of attention; it contains the property of time step i.

Properties of Attention

- **Property 1:** All scores are positive (thanks to $\exp(\cdot)$).
- **Property 2:** The sum of scores is always one.
- **Property 3:** Higher $\boldsymbol{h}_{T+k}^{\top}\boldsymbol{h}_i$ leads to a higher score.
  - It means more related.



Attention RNN

- How an attention RNN works:
  - Compute the scores of all encoder states using the scoring function.
  - Compute the weighted average of these states:

$$\vec{\boldsymbol{h}}_{T+k} = \sum_{i=1}^{T} \alpha_i \boldsymbol{h}_i.$$

  - Pass $\vec{\boldsymbol{h}}_{T+k}$ as an input to the k-th decoding step.
  - **Note:** The state changes at every decoding step by the query $\boldsymbol{h}_{T+k}$.

Outline

1. State space models
2. From linear to deep models
3. Convolutional neural networks
4. Encoder-decoder structure
5. **<u>Summary</u>**

Summary

- MLPs are robust baseline methods.
- RNNs were the de facto standard model for sequence modeling.
- Later research shows that CNNs are more effective than RNNs.
  - Although some people favor advanced versions of RNNs.
- Transformer methods show SOTA performance in some cases.
  - They still have various limitations, e.g., quadratic complexity.
  - Currently an active area of research in time series analysis.