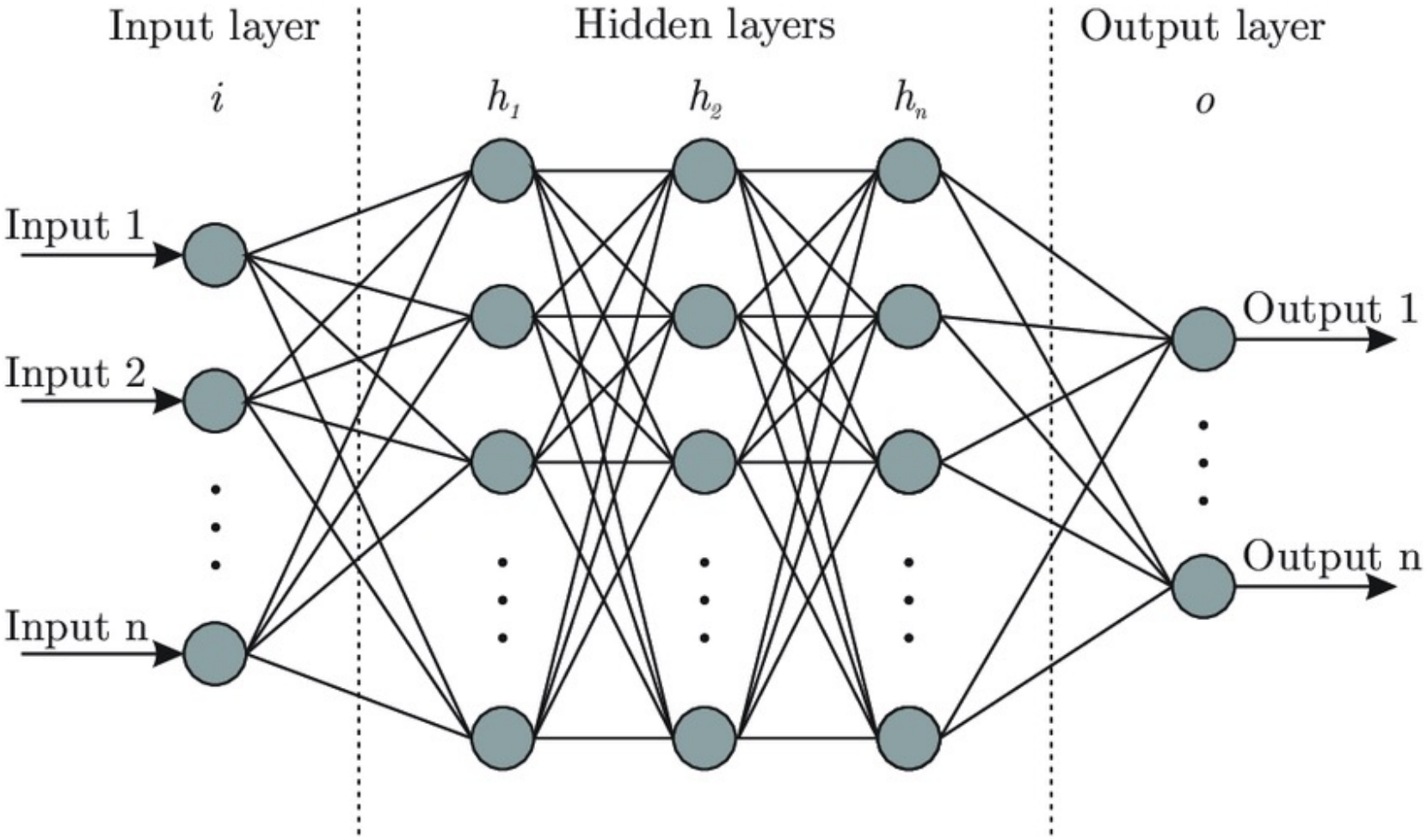
2022 Fall
IE313 Time Series Analysis

A3. Artificial Neural Network Models for Time Series Analysis

Yongjae Lee
Department of Industrial Engineering
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Artificial neural networks



Artificial neural networks

- Big difference from time series models we learned so far
  - Huge flexibility
    - According to universal approximation theorems, feed forward networks can closely approximate almost any kind of continuous functions (even nonlinear and very complicated functions)
    - Of course, this is just theoretical guarantee. In practice, it is sometimes quite hard to find the right parameters that can closely appreciate the target function
    - AI researchers want to identify some special structures of ANNs (or inductive bias) that are appropriate for certain type of problems
      - E.g., CNNs for image problems, RNNs for sequence problems

Artificial neural networks

- Big difference from time series models we learned so far
  - Almost purely data-driven
    - ANN models mostly do not require any kind of assumption on the dynamics of time series data
      - AR models assume that the current and past observations have a linear relationship
      - DLMs assume that its state process follows a linear model and observations have a linear relationship with the current state
      - HMMs assume that its state process follows a Markov chain
    - ANN models just learn the dynamics from the data

Artificial neural networks

- Big difference from time series models we learned so far
  - Previous models have their own dynamics, and we want to find an appropriate model that has the most similar dynamics with the data.
  - However, for ANN models, we do not choose models to fit the dynamics of the data. Instead, we choose models that have “the ability to learn” from the data.

Artificial neural networks for time series analysis

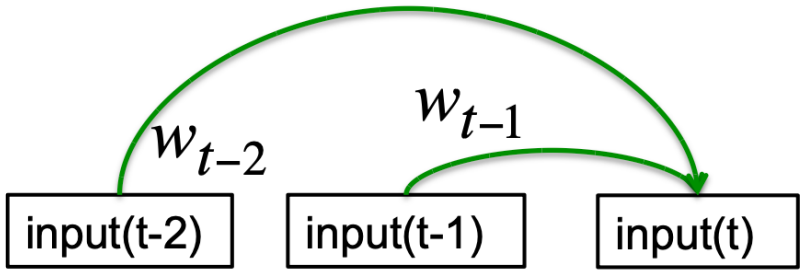
- Recurrent Neural Network (RNN) and its variants
  - Long Short Term Memory (LSTM)
  - Gated Recurrent Unit (GRU)
  - Neural ODEs
- Generative models
  - Generative Adversarial Networks (GANs)
  - Diffusion models
- Attention-based models
  - Transformers

A2.1

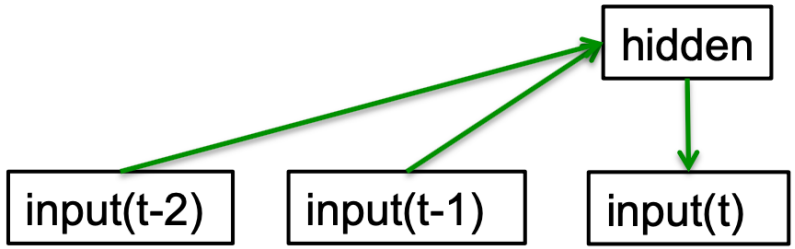
Recurrent Neural Network (RNN) and Its Variants

Memoryless models for sequences

- Autoregressive models
  Predict the next term in a sequence from a fixed number of previous terms using “delay taps”.



- Feed-forward neural nets
  These generalize autoregressive models by using one or more layers of non-linear hidden units.



Source: Geoffrey Hinton’s lecture note
https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf

Beyond memoryless models

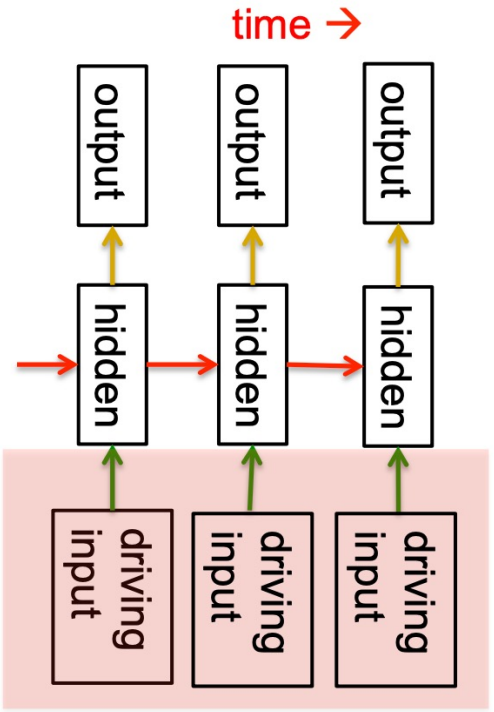
- If we give our generative model some hidden state, and if we give this hidden state its own internal dynamics, we get a much more interesting kind of model.
  - It can store information in its hidden state for a long time.
  - If the dynamics is noisy and the way it generates outputs from its hidden state is noisy, we can never know its exact hidden state.
  - The best we can do is to infer a probability distribution over the space of hidden state vectors.
- This inference is only tractable for two types of hidden state model.

Source: Geoffrey Hinton's lecture note
https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf

Beyond memoryless models

Linear Dynamical Systems (engineers love them!)

- These are generative models. They have a real-valued hidden state that cannot be observed directly.
  - The hidden state has linear dynamics with Gaussian noise and produces the observations using a linear model with Gaussian noise.
  - There may also be driving inputs.
- To predict the next output (so that we can shoot down the missile) we need to infer the hidden state.
  - A linearly transformed Gaussian is a Gaussian. So the distribution over the hidden state given the data so far is Gaussian. It can be computed using "Kalman filtering".

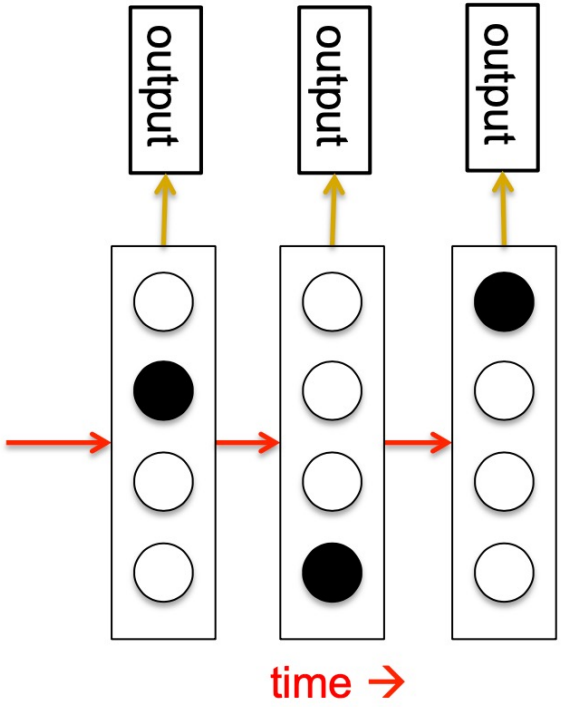


Source: Geoffrey Hinton's lecture note
https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf

Beyond memoryless models

Hidden Markov Models (computer scientists love them!)

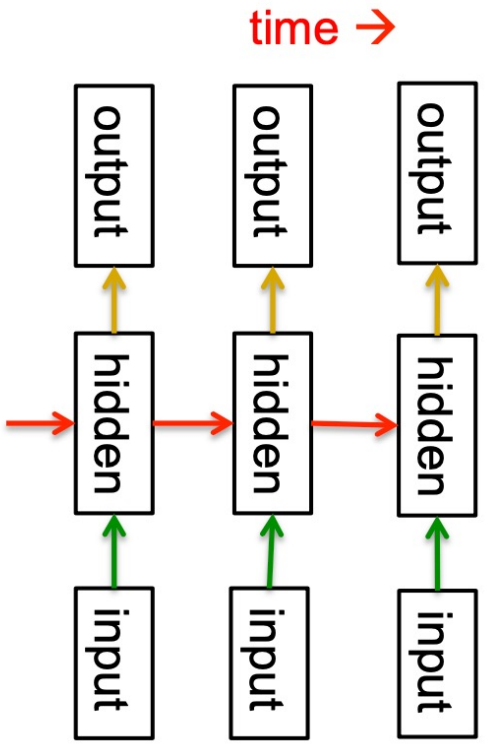
- Hidden Markov Models have a discrete one-of-N hidden state. Transitions between states are stochastic and controlled by a transition matrix. The outputs produced by a state are stochastic.
  - We cannot be sure which state produced a given output. So the state is "hidden".
  - It is easy to represent a probability distribution across N states with N numbers.
- To predict the next output we need to infer the probability distribution over hidden states.
  - HMMs have efficient algorithms for inference and learning.



Source: Geoffrey Hinton's lecture note
https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf

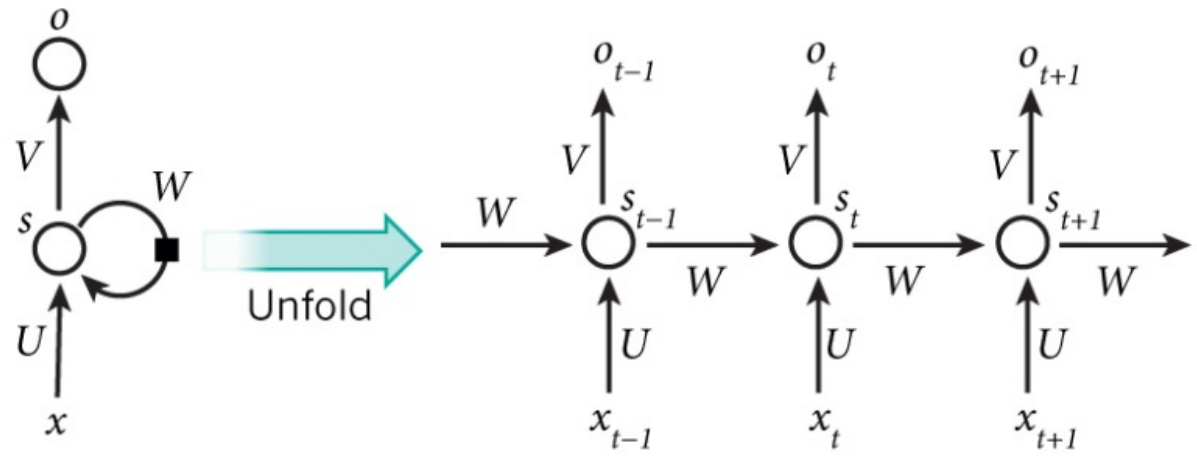
Recurrent neural networks (RNN)

- RNNs are very powerful, because they combine two properties:
  - Distributed hidden state that allows them to store a lot of information about the past efficiently.
  - Non-linear dynamics that allows them to update their hidden state in complicated ways.
- With enough neurons and time, RNNs can compute anything that can be computed by your computer.



Source: Geoffrey Hinton's lecture note
https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf

Recurrent neural networks (RNN)

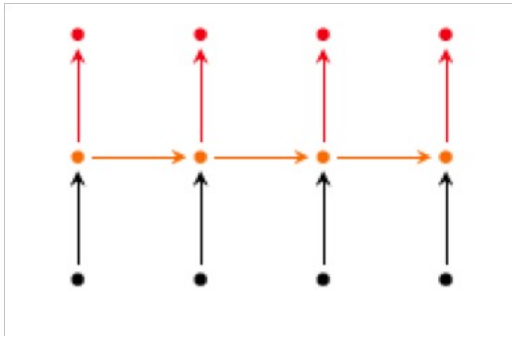


A recurrent neural network and the unfolding in time of the computation involved in its forward computation. Source: Nature

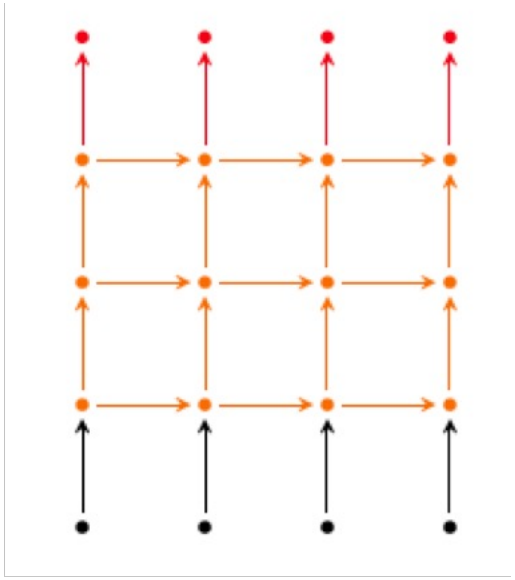
- x_t: input at time step t
- s_t: hidden state at time step t
  - It is the memory of the network
  - Calculated by $s_t = f(Ux_t + Ws_{t-1})$ (typical choice for *f* would be tanh or ReLU)
- v_t: output at step t
  - Calculated by $v_t = g(Vs_t)$ (typical choice for *g* would be softmax)

Recurrent neural networks (RNN)

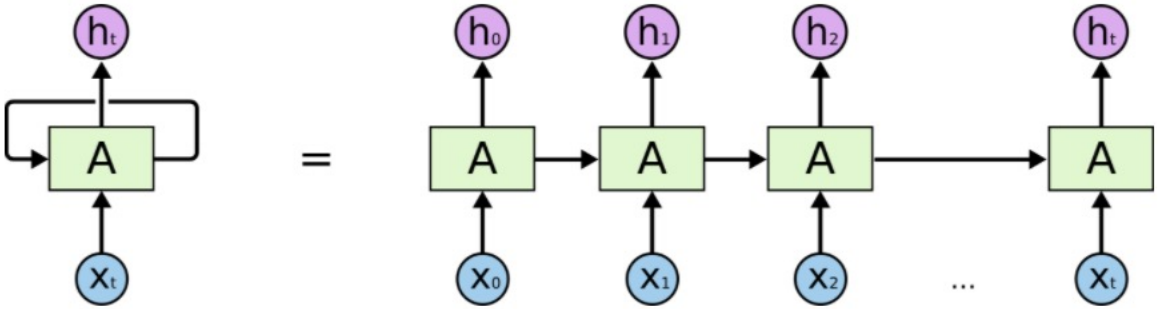
Single-layer



Multi-layer



Recurrent neural networks (RNN)

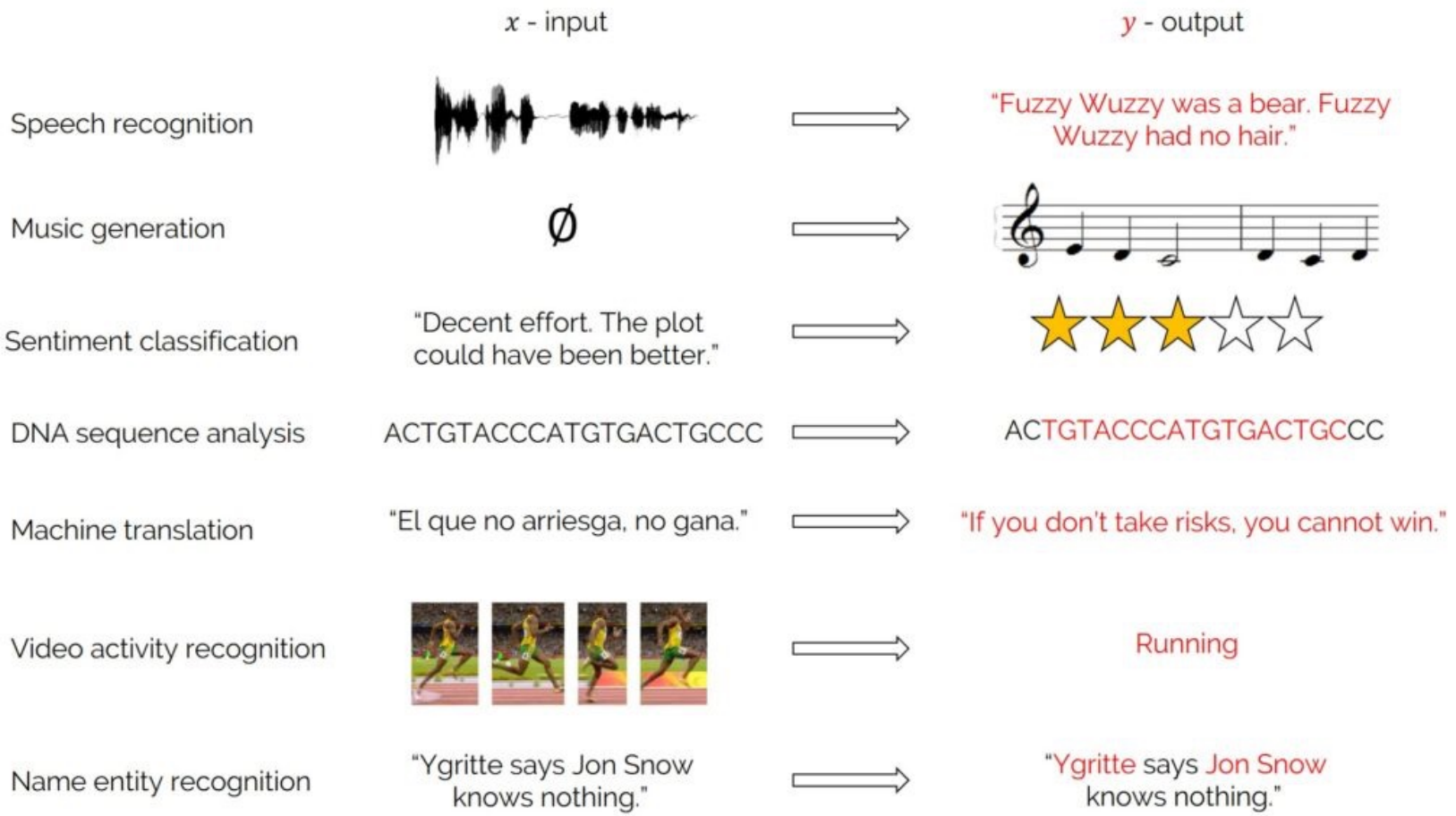


An unrolled recurrent neural network.

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent neural networks (RNN)

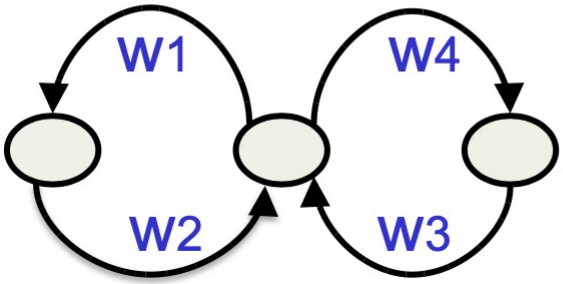
- RNNs are widely used in



http://datahacker.rs/003-rnn-architectural-types-of-different-recurrent-neural-networks/

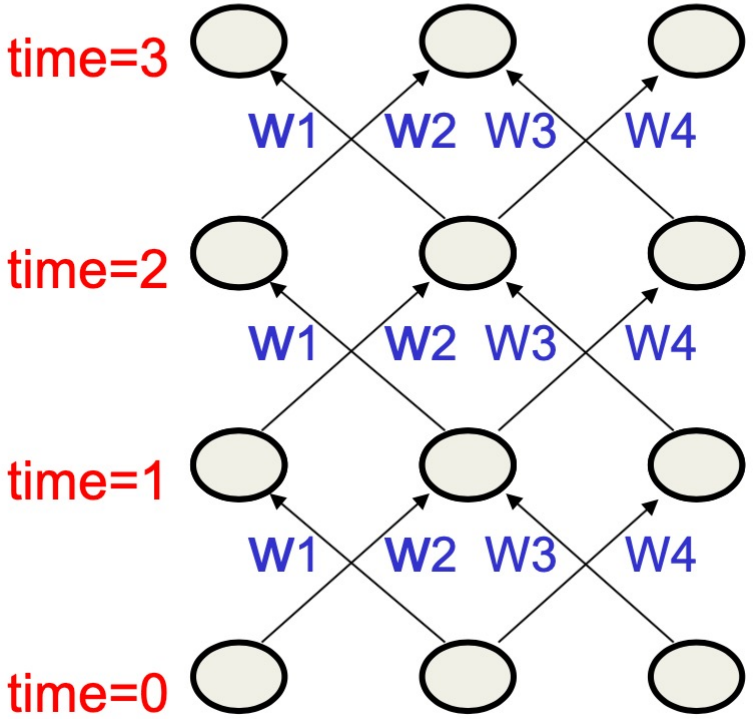
Recurrent neural networks (RNN)

The equivalence between feedforward nets and recurrent nets



Assume that there is a time delay of 1 in using each connection.

The recurrent net is just a layered net that keeps reusing the same weights.



Source: Geoffrey Hinton's lecture note
https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf

Recurrent neural networks (RNN)

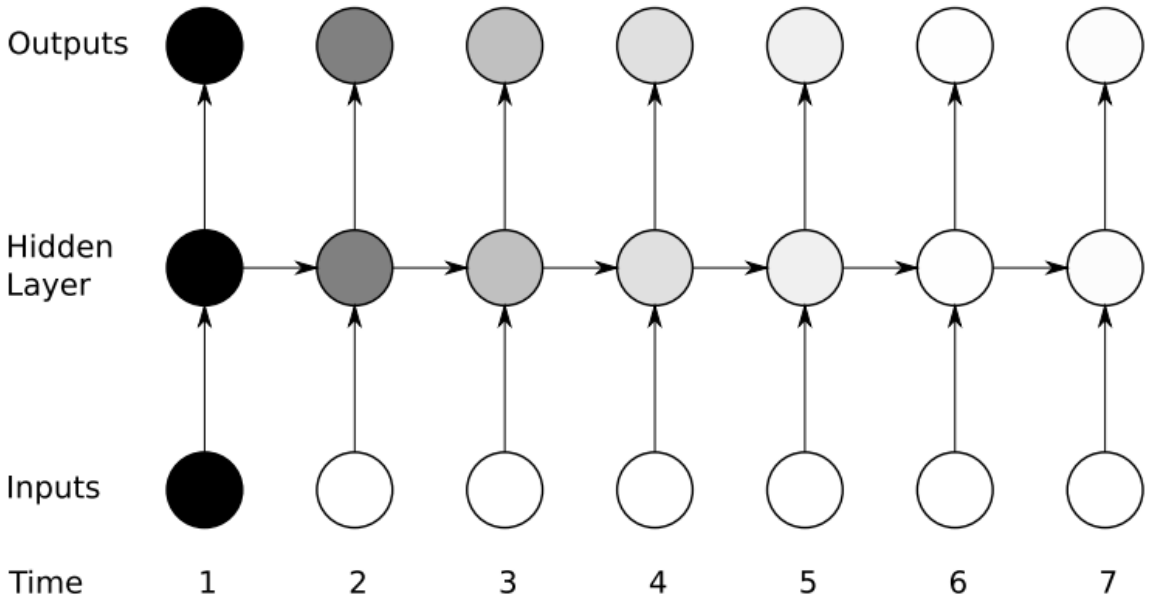
Do generative models need to be stochastic?

- Linear dynamical systems and hidden Markov models are stochastic models.
  - But the posterior probability distribution over their hidden states given the observed data so far is a deterministic function of the data.
- Recurrent neural networks are deterministic.
  - So think of the hidden state of an RNN as the equivalent of the deterministic probability distribution over hidden states in a linear dynamical system or hidden Markov model.

Source: Geoffrey Hinton's lecture note
https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf

Vanishing gradient of RNNs

- There is a crucial limitation in the vanilla RNN architecture
  - The influence of an input either decays or blows up exponentially as it cycles around the recurrent connections
  - Roughly speaking, RNNs ***forget*** about early inputs



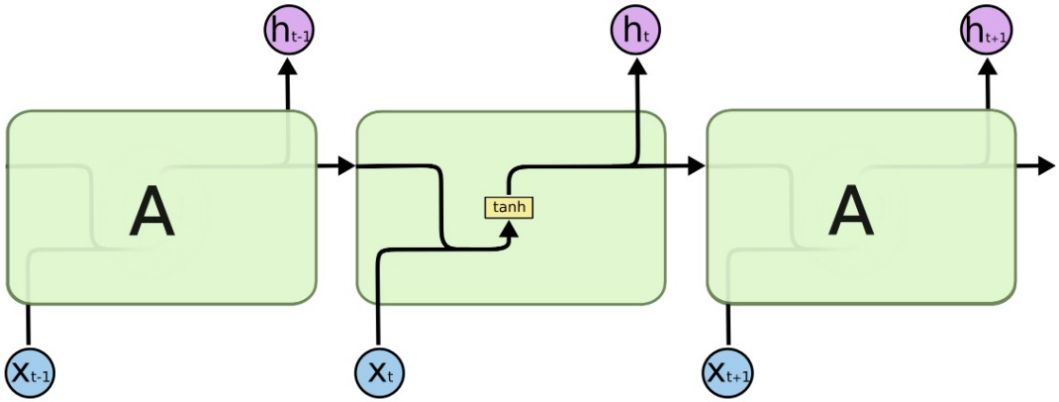
Long short term memory (LSTM)

- Hochreiter & Schmidhuber (1997) solved the problem of getting an RNN to remember things for a long time (like hundreds of time steps).
- They designed a memory cell using logistic and linear units with multiplicative interactions.

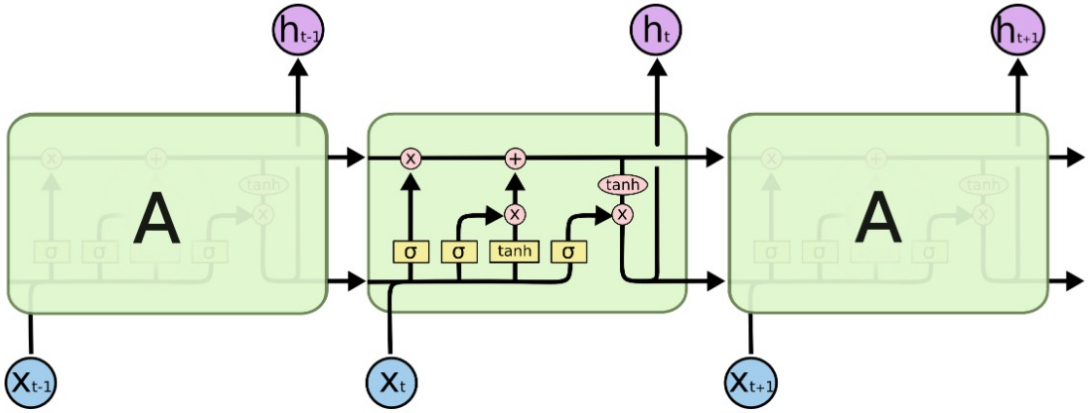
- Information gets into the cell whenever its “write” gate is on.
- The information stays in the cell so long as its “keep” gate is on.
- Information can be read from the cell by turning on its “read” gate.

Source: Geoffrey Hinton’s lecture note
https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf

Long short term memory (LSTM)



The repeating module in a standard RNN contains a single layer.

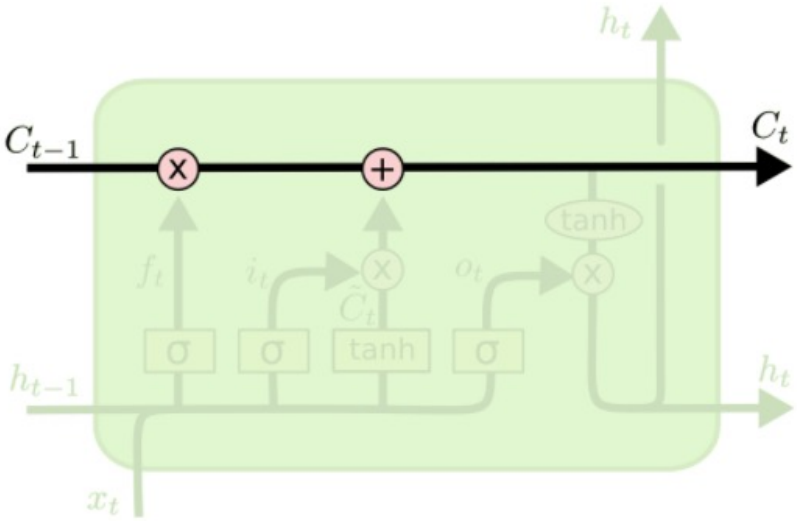


The repeating module in an LSTM contains four interacting layers.

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long short term memory (LSTM)

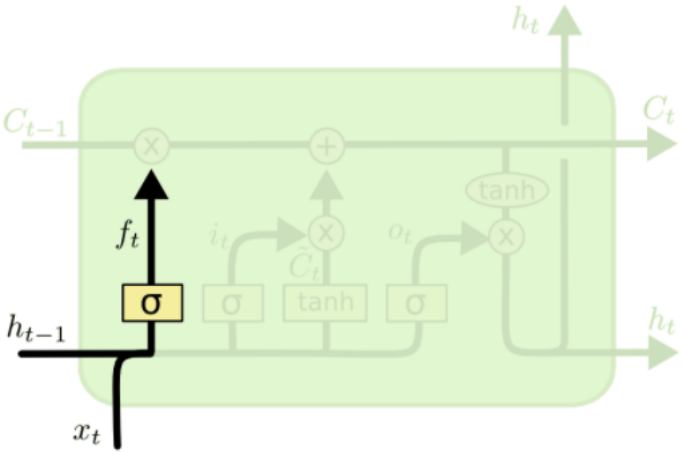
- **Cell state $\{C_t\}$**
  - It runs straight down the entire chain with only some minor linear interactions (like a conveyor belt)
  - LSTM has the ability to remove or add information to the cell state, carefully regulated by gates



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long short term memory (LSTM)

- **Forget gate layer: what to keep and what to forget**
  - This gate looks at the previous hidden state h_{t-1} and the current input x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}
    - 1: completely keep this
    - 0: completely get rid of this

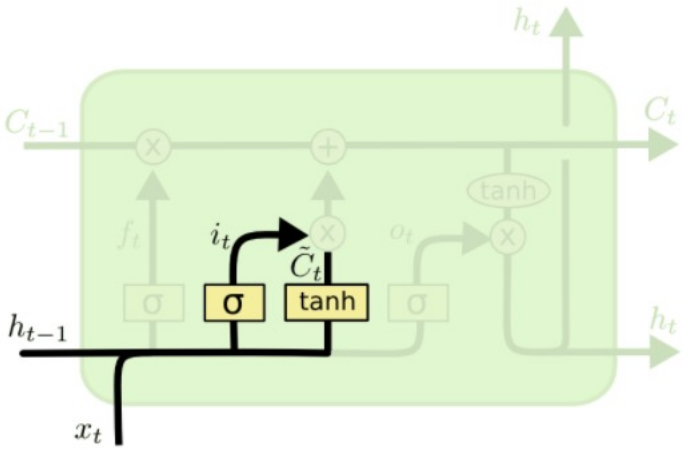


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long short term memory (LSTM)

- **What new information to store in the cell state**
  - **Input gate layer (sigmoid layer)**
    - Decides which values we'll update
  - **tanh layer**
    - Creates a vector of new candidate values $\tilde{C}_t$



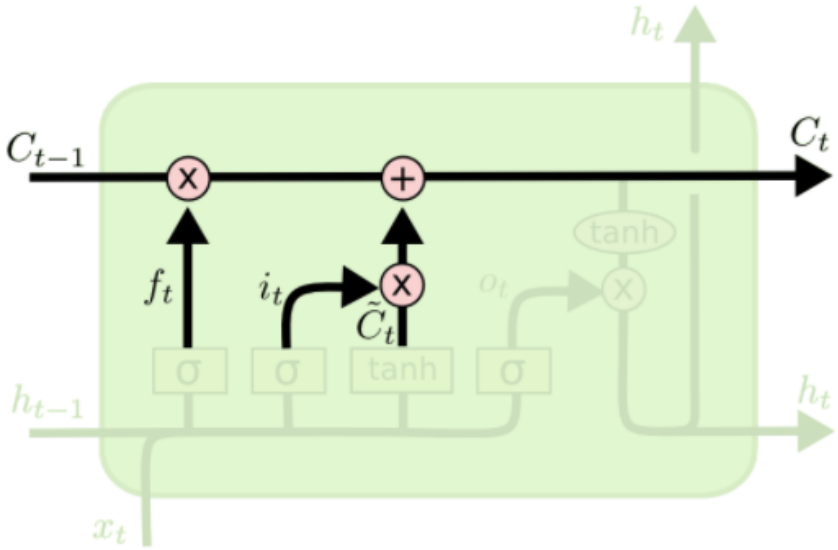
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long short term memory (LSTM)

- **Update the cell state**

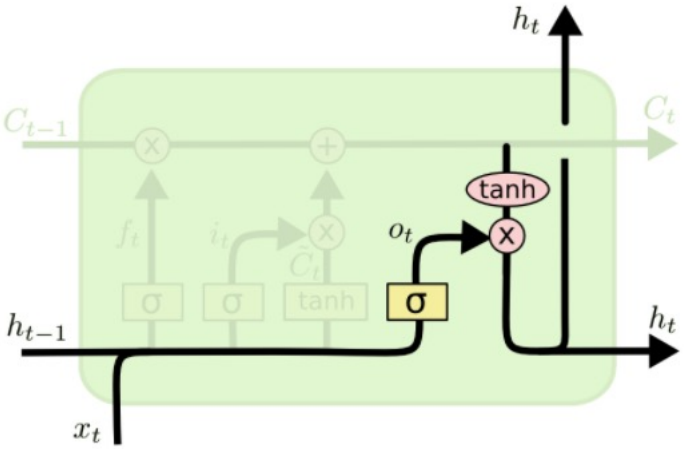


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long short term memory (LSTM)

- **What we're going to output**
  - **Sigmoid layer (read gate)**
    - Decides what parts of the cell state we're going to output
  - **tanh layer**
    - The cell state goes through this layer and multiplied by the out put of the sigmoid gate



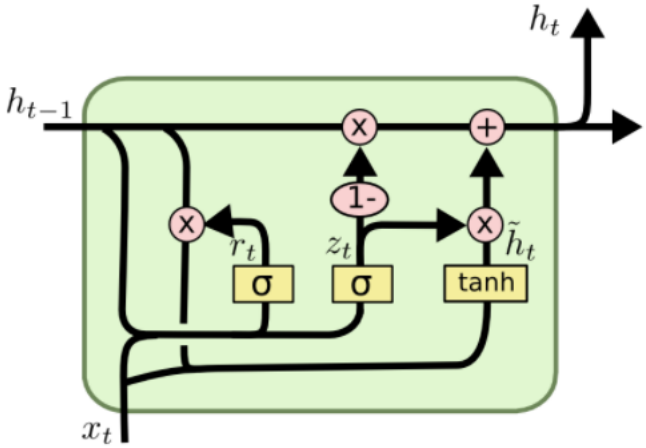
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Gated Recurrent Unit (GRU)

- It is a variation on the LSTM proposed by Cho et al. (2014)
  - It combines the forget and input gates into a single "update gate"
  - It also merges the cell state and hidden state, and makes some other changes
  - The resulting model is **simpler** than standard LSTM models, and has been growing increasingly popular



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Prof. Cho Kyunghyun
NYU CS

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Neural ODE

- RNN-based models including LSTM and GRU assume a discrete-time process for hidden states
  - This may have problems when observations are not evenly spaced
- NeuralODE (Chen et al., 2018) assumes a continuous-time process for hidden states
  - ANN learns the **derivative** of hidden state

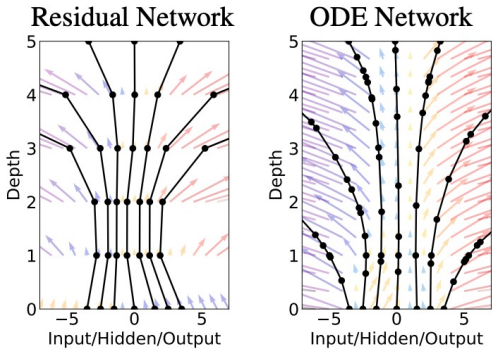


Figure 1: *Left:* A Residual network defines a discrete sequence of finite transformations. *Right:* A ODE network defines a vector field, which continuously transforms the state. *Both:* Circles represent evaluation locations.

$$\mathbf{h}_{t+1} = \mathbf{h}_t + f(\mathbf{h}_t, \theta_t)$$

$$\frac{d\mathbf{h}(t)}{dt} = f(\mathbf{h}(t), t, \theta)$$

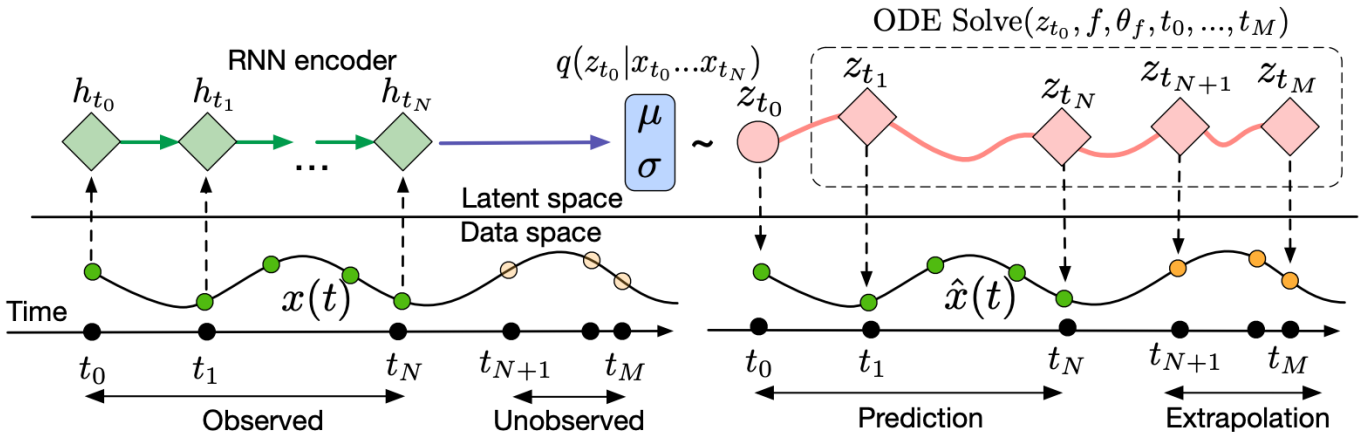


Figure 6: Computation graph of the latent ODE model.

RNN-based models

- Example: DeepAR (Salinas et al., 2020) by Amazon

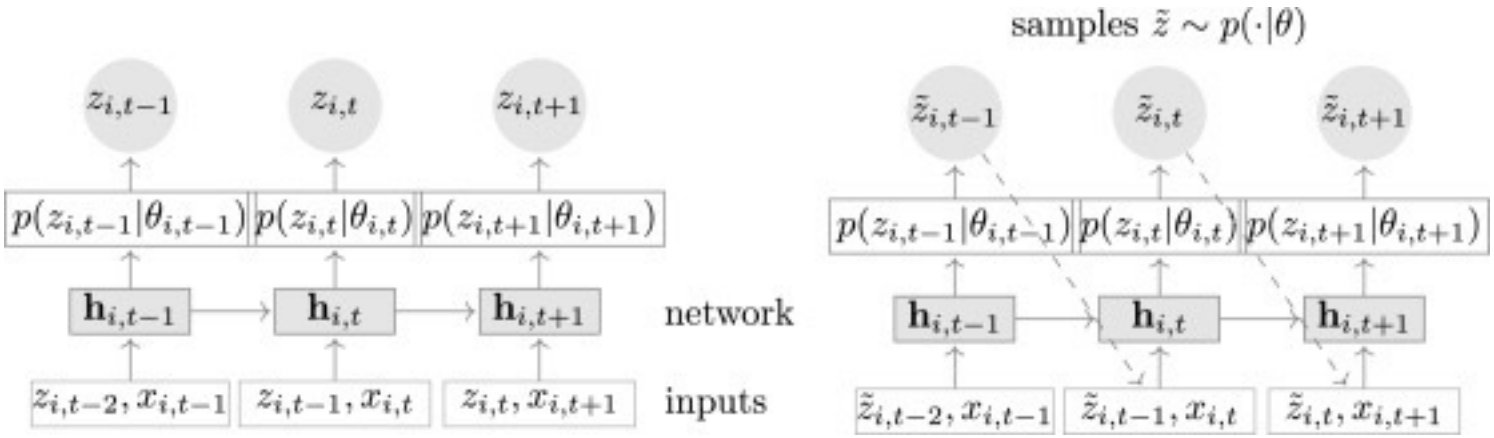


Fig. 3. Summary of the model. Training (left): At each time step t, the inputs to the network are the covariates $x_{i,t}$, the target value at the previous time step $z_{i,t-1}$, and the previous network output $\mathbf{h}_{i,t-1}$. The network output $\mathbf{h}_{i,t} = h\left(\mathbf{h}_{i,t-1}, z_{i,t-1}, \mathbf{x}_{i,t}, \Theta\right)$ is then used to compute the parameters $\theta_{i,t} = \theta\left(\mathbf{h}_{i,t}, \Theta\right)$ of the likelihood $p\left(z|\theta\right)$, which is used for training the model parameters. For prediction, the history of the time series $z_{i,t}$ is fed in for $t < t_0$, then in the prediction range (right) for $t \geq t_0$ a sample $\hat{z}_{i,t} \sim p\left(\cdot|\theta_{i,t}\right)$ is drawn and fed back for the next point until the end of the prediction range $t = t_0 + T$, generating one sample trace. Repeating this prediction process yields many traces that represent the joint predicted distribution.

https://doi.org/10.1016/j.ijforecast.2019.07.001

RNN-based models

- Example: DeepAR (Salinas et al., 2020) by Amazon

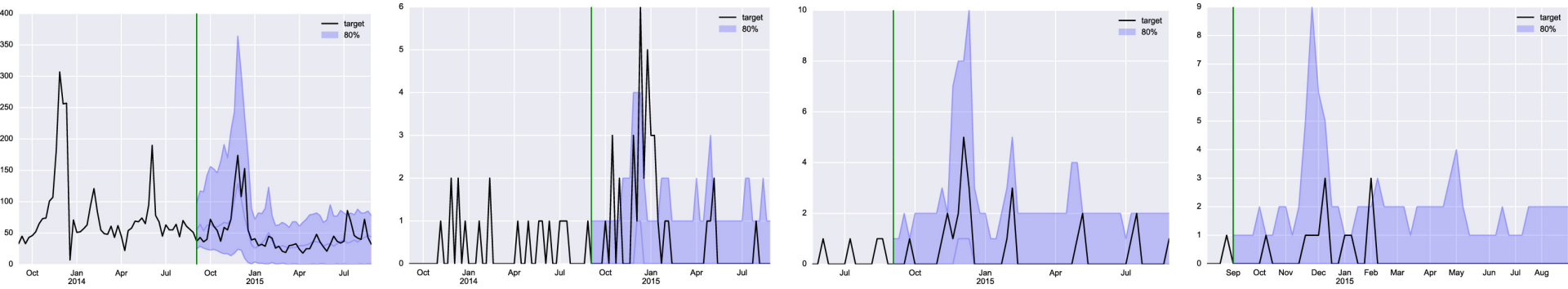


Fig. 5. Example time series of ec. The vertical line separates the conditioning period from the prediction period. The black line shows the true target. In the prediction range, we plot the p50 as a blue line (mostly zero for the three slow items), along with 80% confidence intervals (shaded). The model learns accurate seasonality patterns and uncertainty estimates for items of different velocities and ages . (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

https://doi.org/10.1016/j.ijforecast.2019.07.001

A2.2

Generative models

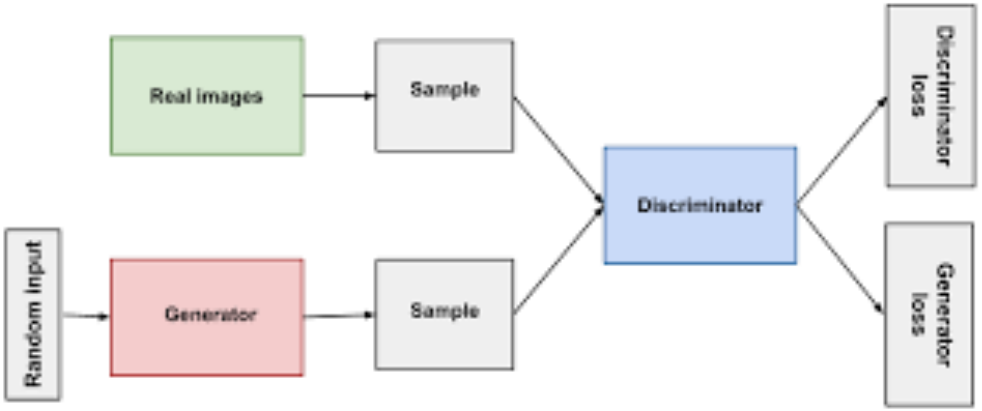
Generative models

- **Generative** models vs. **discriminative** models
  - Informally..
    - **Discriminative** models discriminate between different kinds of data instances
      - For example, given a photo, determine if it contains a cat or a dog
    - **Generative** models can generate new data instances (unsupervised learning)
      - For example, generate a new photo that looks like other photos in the data set
  - More formally,
    given a set of data instances X and a set of labels Y,
    - **Discriminative** models capture the conditional probability $\mathrm{P}(\mathrm{Y} \mid X)$
    - **Generative** models capture the joint probability $\mathrm{P}(X, Y)$, or just $\mathrm{P}(X)$ if there are no labels

Source: Google Developers

Generative Adversarial Networks (GAN)

- Two networks compete with each other like a game. Proposed by Goodfellow et al. (2014)
  - Generator (generative network)
    - Tries to learn the data distribution of interest
    - Generates candidates to fool the discriminator
  - Discriminator (discriminative network)
    - Tries to distinguish candidates produced by the generator from the true data distribution



Generative Adversarial Networks (GAN)

- To be more specific,
  - From a random noise z,
    - Generator G tries to generate a sample that looks like one of data instances x

$$G(z) \approx x$$

    - Discriminator D tried to determine if the input is generated by the generator or one of real data instances
      - $D(\cdot)$ is close to 1 if the input seems to be from the real dataset
      - $D(\cdot)$ is close to 0 if the input seems to be a fake generated by G
    - Hence, we can find D by

$$\max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right]$$

    - Finally, we want G to fool D, so

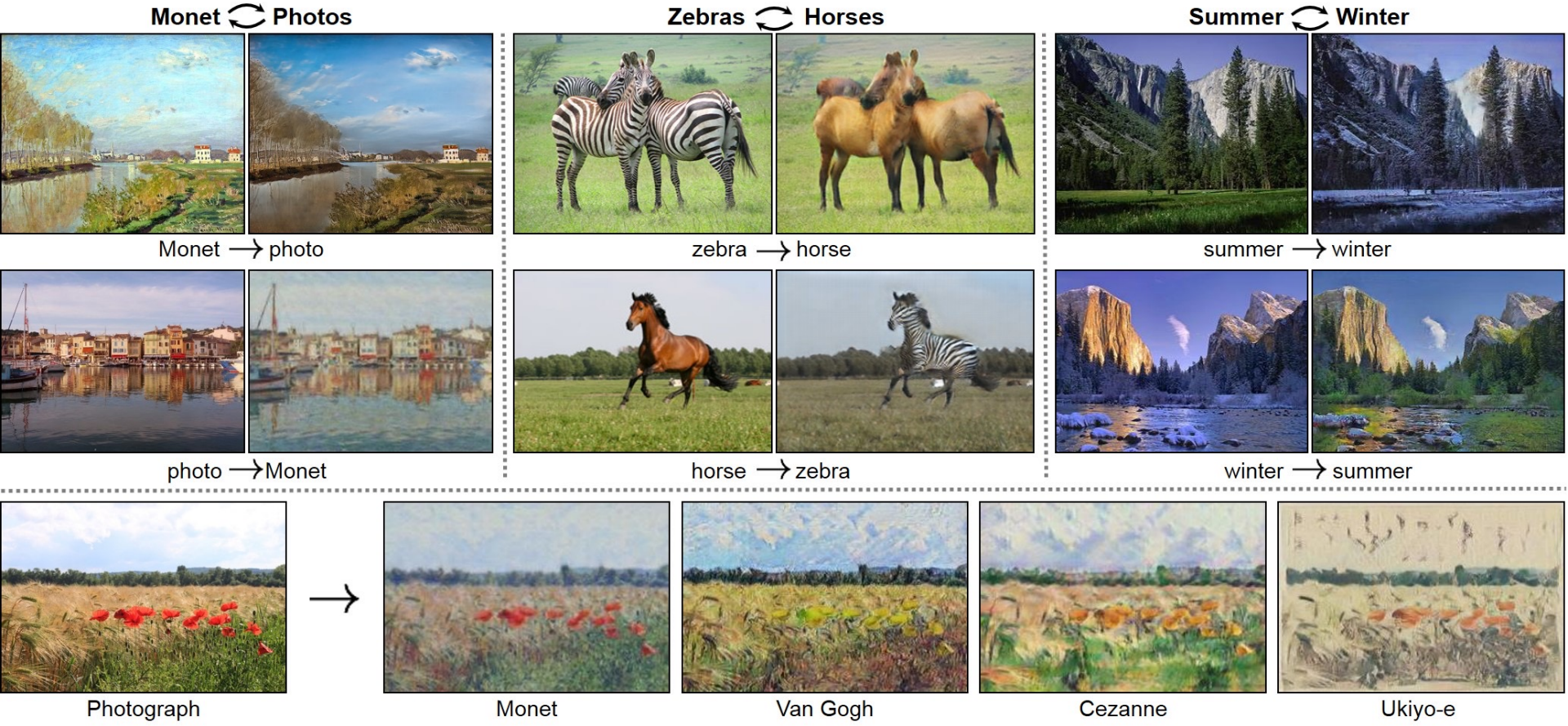
$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right]$$

Generative Adversarial Networks (GAN)

- Examples: StyleGAN (Karras et al., 2019) by NVIDIA

Generative Adversarial Networks (GAN)

- Examples: CycleGAN (Zhu et al., 2017)



Generative Adversarial Networks (GAN)

- Examples: TimeGAN (Yoon et al., 2019)

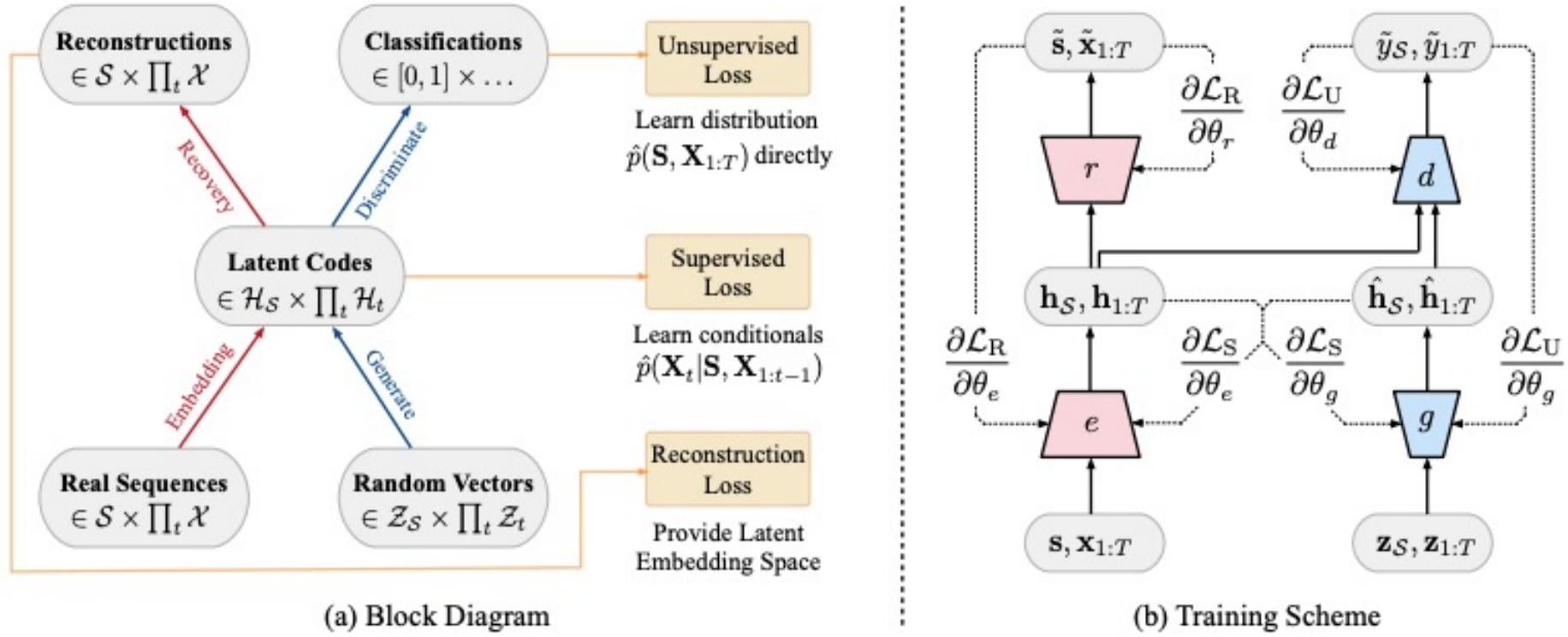
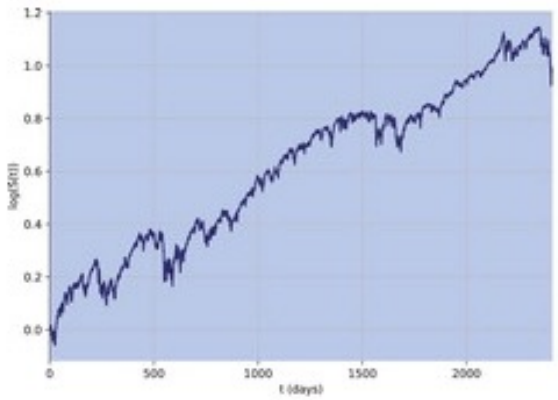


Figure 1: (a) Block diagram of component functions and objectives. (b) Training scheme; solid lines indicate forward propagation of data, and dashed lines indicate backpropagation of gradients.

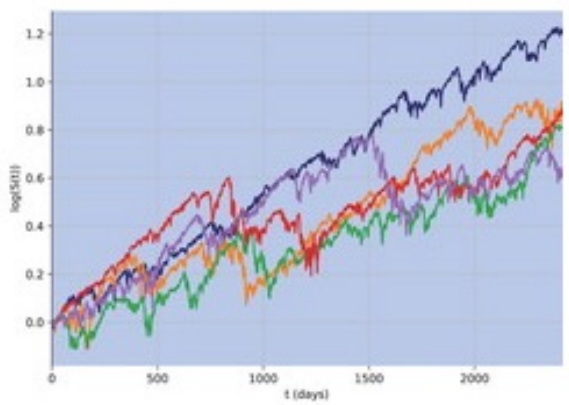
https://papers.nips.cc/paper/2019/hash/c9efe5f26cd17ba6216bbe2a7d26d490-Abstract.html

Generative Adversarial Networks (GAN)

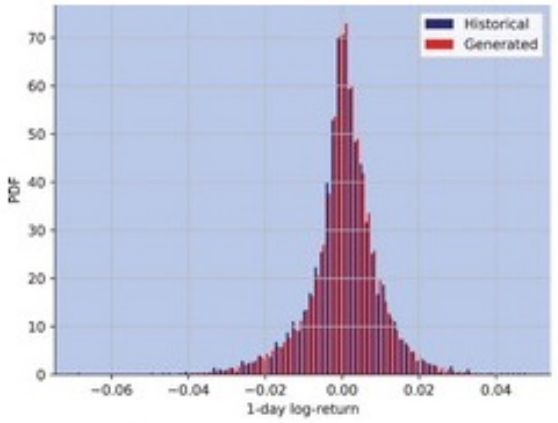
- Examples: QuantGANs (Wiese et al., 2020)



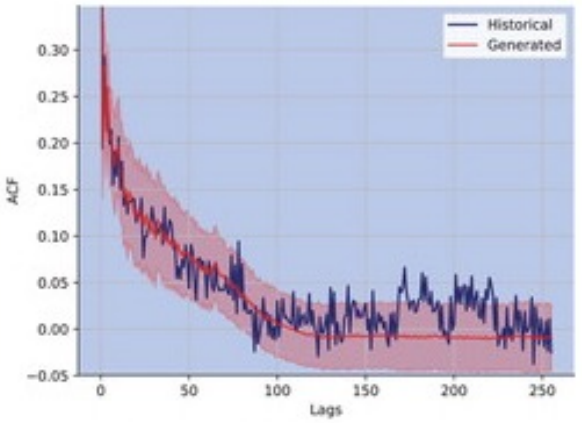
(a) S&P 500 log-path



(b) Generated log-paths of a Quant GAN



(a) Histogram of daily log-returns

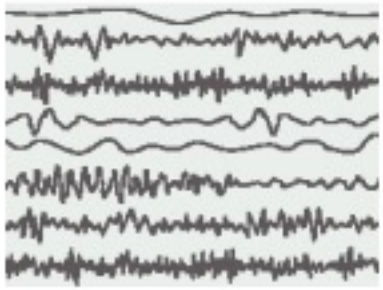


(b) ACF of absolute log-returns

•https://doi.org/10.1080/14697688.2020.1730426

Generative Adversarial Networks (GAN)

- Examples: TadGAN (Geiger et al., 2020)



Unsupervised ML Model

Anomalies	t_{start}	t_{stop}
1	Jan 10th, 2019 - 8:16 am	Jan 10th, 2019 - 3:34 pm
2	Jan 16th, 2019 - 11:16 am	Jan 17th, 2019 - 2:34 am
...	...	...
18	Mar 24th, 2019 - 2:12 pm	Mar 28th, 2019 - 3:19 pm

Fig. 1. An illustration of time series anomaly detection using unsupervised learning. Given a multivariate time series, the goal is to find out a set of anomalous time segments that have unusual values and do not follow the expected temporal patterns.

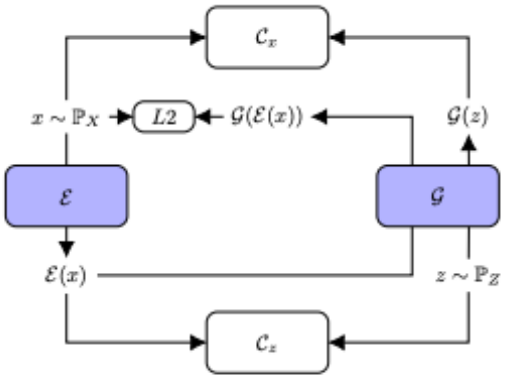
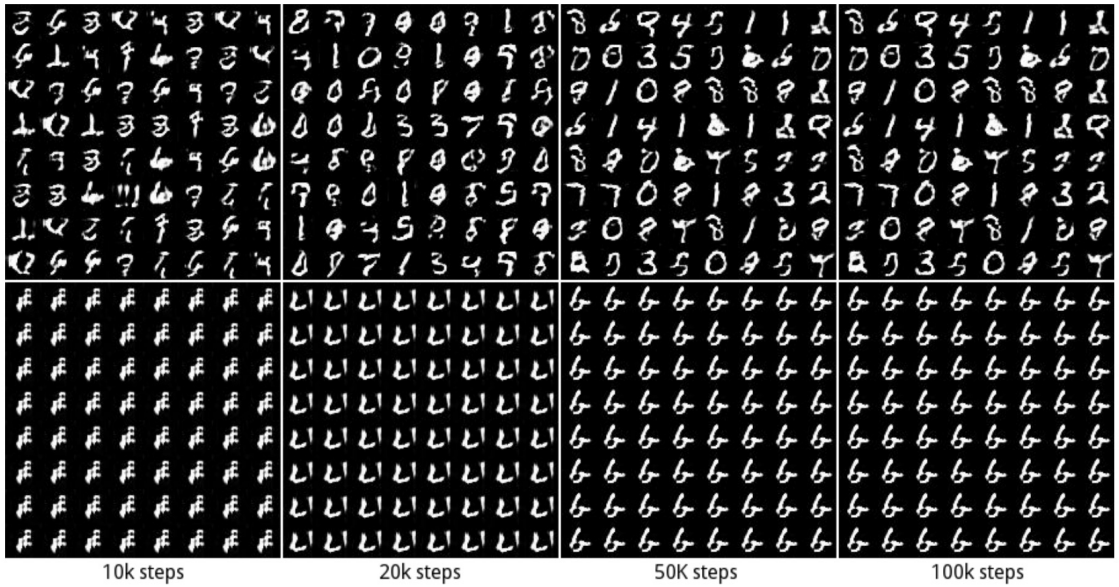


Fig. 2. Model architecture: *Generator* $\mathcal{E}$ is serving as an Encoder which maps the time series sequences into the latent space, while *Generator* $\mathcal{G}$ is serving as a Decoder that transforms the latent space into the reconstructed time series. *Critic* C_x is to distinguish between real time series sequences from X and the generated time series sequences from $\mathcal{G}(z)$, whereas *Critic* C_z measures the goodness of the mapping into the latent space.

https://ieeexplore.ieee.org/abstract/document/9378139

Generative Adversarial Networks (GAN)

- While GANs have brought many developments in generative models, there are a few tricky points to actually use GANs
  - Mode collapse
    - The generator may be able to find a very strange output that can always fool the discriminator



    - This may be resolved by using Wasserstein loss (WGAN, Arjovsky et al., 2017), but still a quite tricky point

Generative Adversarial Networks (GAN)

- While GANs have brought many developments in generative models, there are a few tricky points to actually use GANs
  - Difficult to train
    - It is quite hard to find the right hyperparameters that would make the right balance between the generator and discriminator

Diffusion models

- By gradually adding noise to the data instances, we can see how a data instance gradually turns into a pure noise
- If we can learn the reverse of this process, we may generate new data instances from pure noises

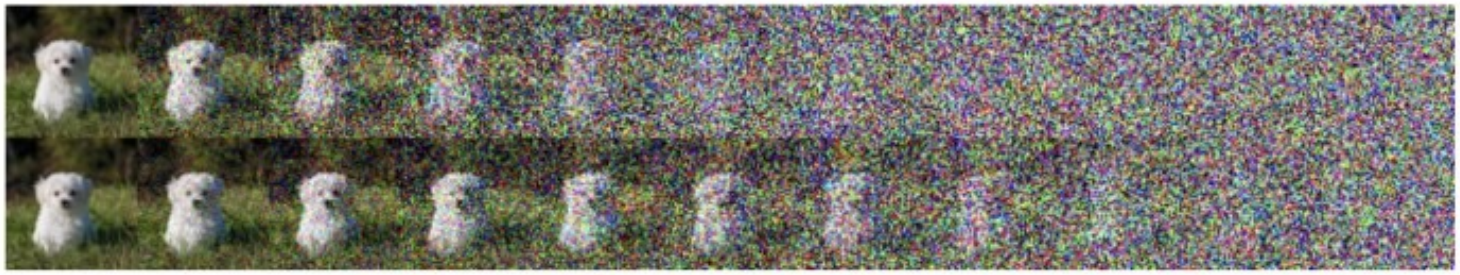


Image source: S. Orbell's blog

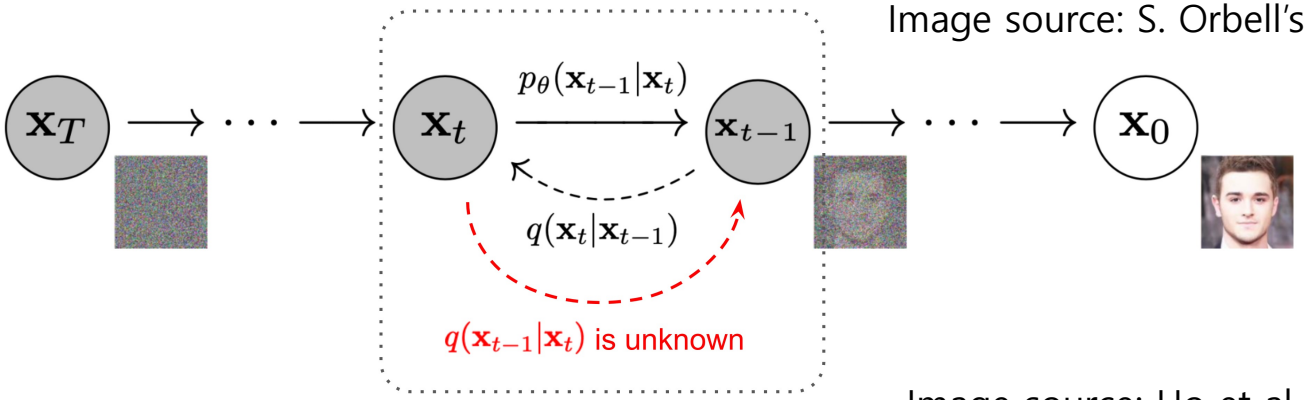


Image source: Ho et al. (2020)

Diffusion models

- Example: TimeGrad (Rasul et al., 2021)

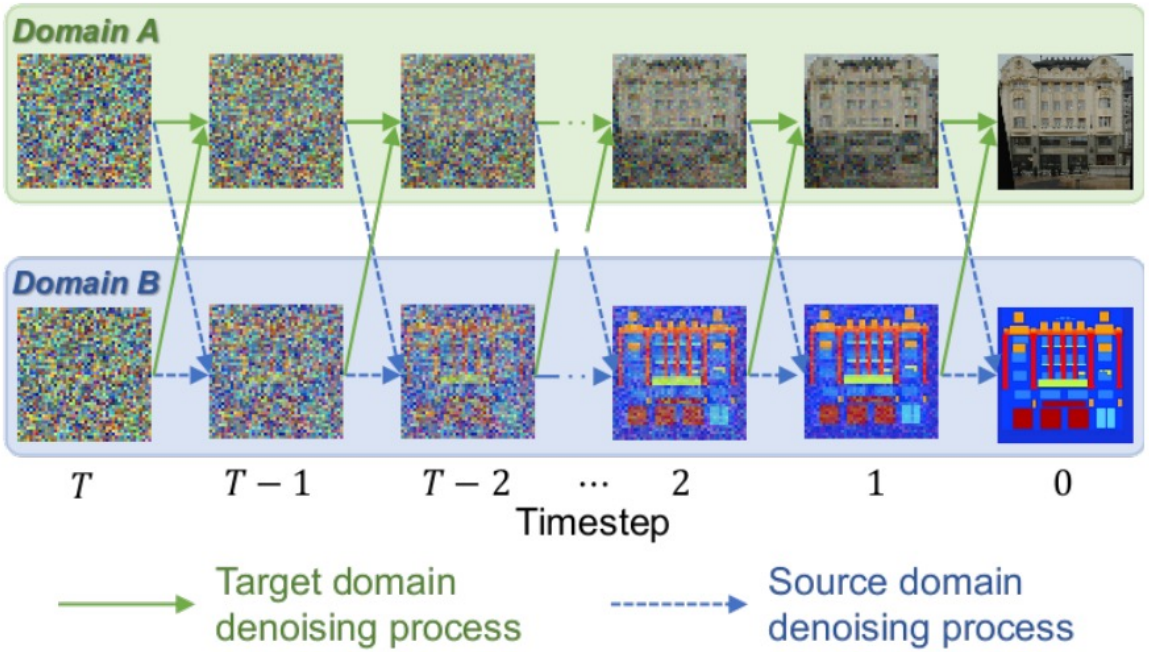


Figure 1: Conceptual illustration of our novel image-to-image translation approach using denoising diffusion probabilistic models.

https://proceedings.mlr.press/v139/rasul21a.html

Diffusion models

- Example: TimeGrad (Rasul et al., 2021)

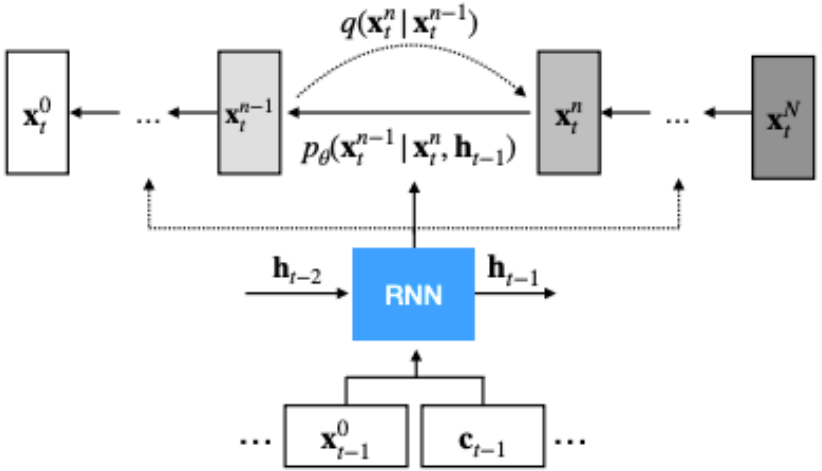


Figure 1. `TimeGrad` schematic: an RNN *conditioned* diffusion probabilistic model at some time $t-1$ depicting the fixed forward process that adds Gaussian noise and the learned reverse processes.

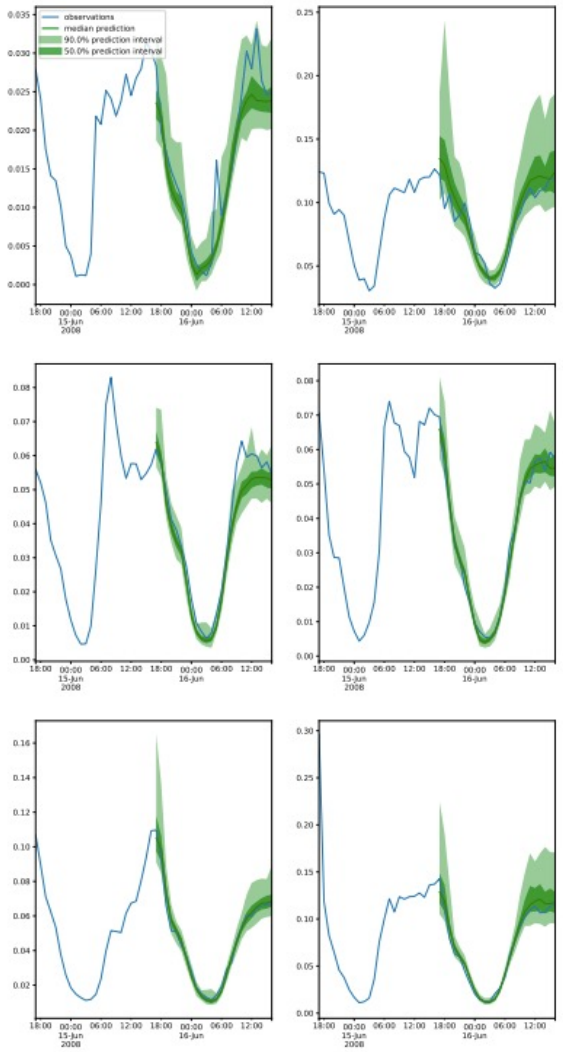


Figure 4. `TimeGrad` prediction intervals and test set ground-truth for `Traffic` data of the first 6 of 963 dimensions from first rolling-window. Note that neighboring entities have an order of magnitude difference in scales.

https://proceedings.mlr.press/v139/rasul21a.html

A2.3

Attention-based models

Sequence-to-sequence models

- For sequence-to-sequence (seq2seq) models, a sequence of inputs go through models like RNNs and the hidden state is updated

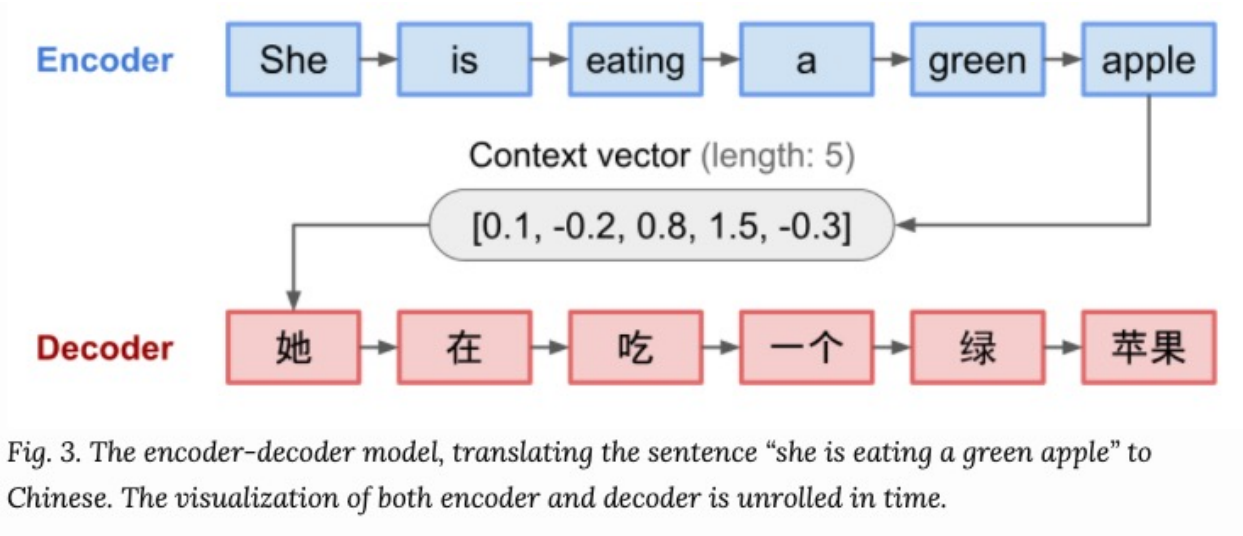


Fig. 3. The encoder-decoder model, translating the sentence "she is eating a green apple" to Chinese. The visualization of both encoder and decoder is unrolled in time.

Image source: Lil'Log

- Then, the hidden state is used to the task (e.g. forecasting, translation, …)

Sequence-to-sequence models

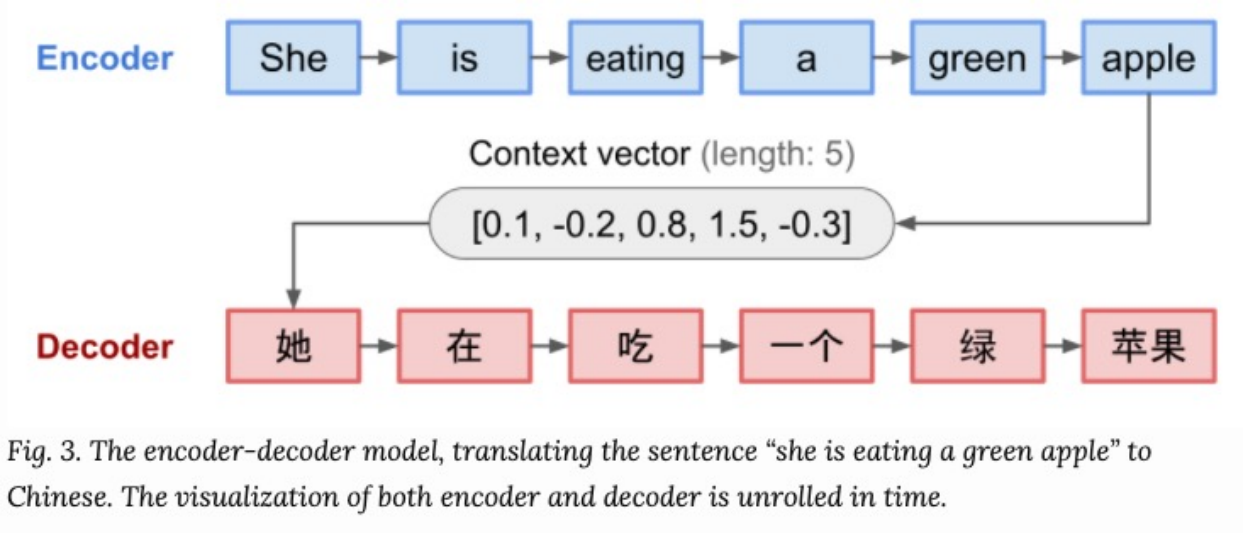


Fig. 3. The encoder-decoder model, translating the sentence "she is eating a green apple" to Chinese. The visualization of both encoder and decoder is unrolled in time.

Image source: Lil'Log

- The problem is that it is hard to remember long history

Attention mechanism

- Therefore, instead of summarizing all the inputs into a single hidden state, attention mechanism decides which input to "attend" more

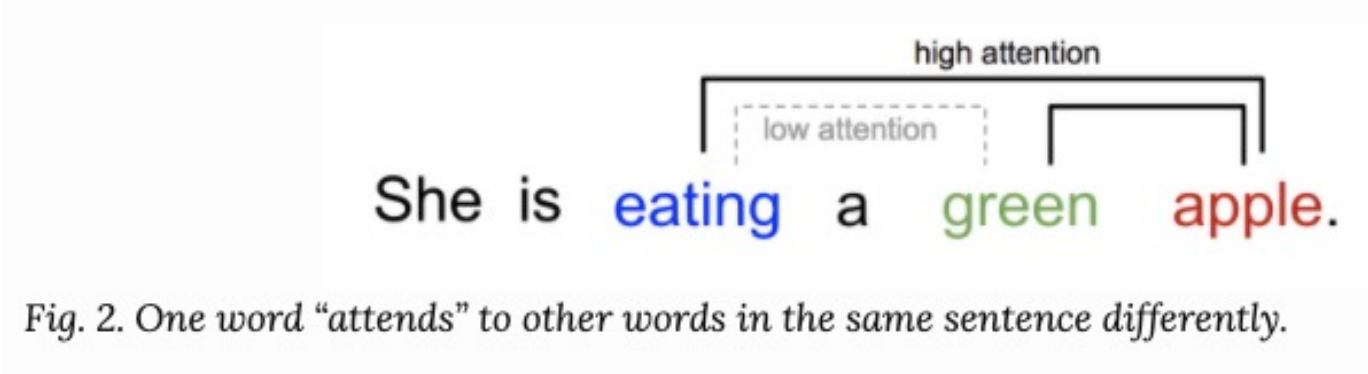


Fig. 2. One word "attends" to other words in the same sentence differently.

Image source: Lil'Log

Attention mechanism

- Therefore, instead of summarizing all the inputs into a single hidden state, attention mechanism decides which input to "attend" more

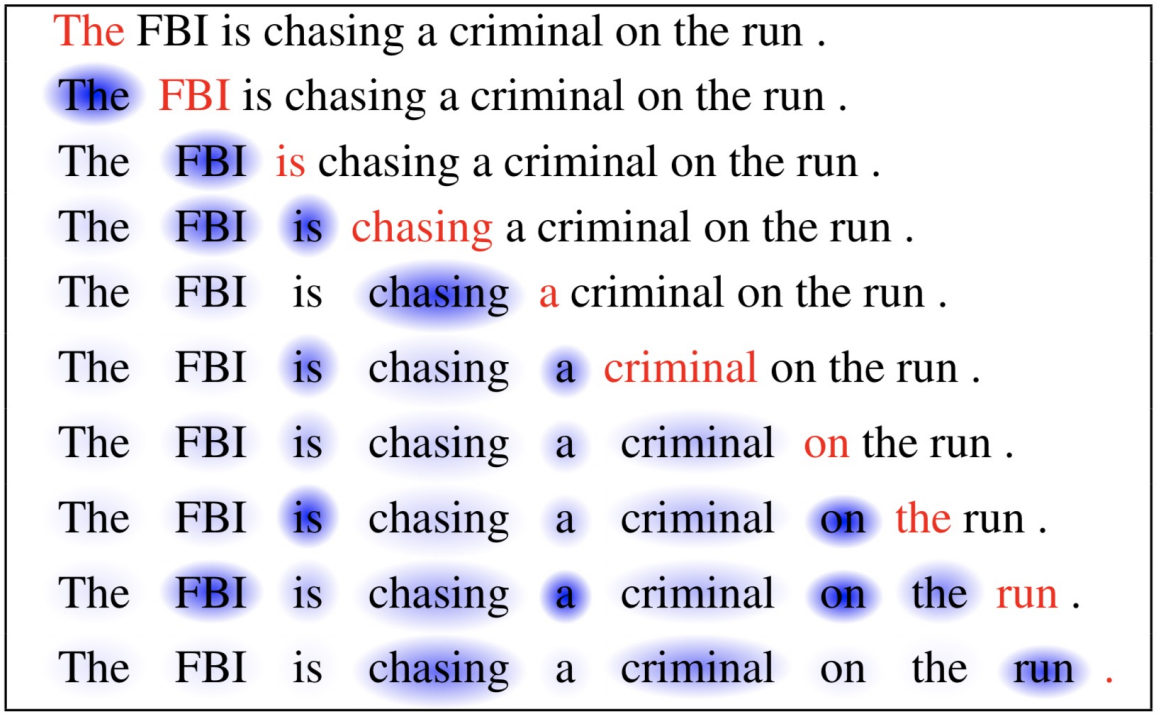


Image source: Cheng et al. (2016)

Attention mechanism

- Attention mechanism is mostly based on three things
  - Query (Q): what kind of information we are looking for
  - Key (K): relevance to the query
  - Value (V): actual contents of the input
  - For example,
    - X: embeddings of some words (each row represents an embedding vector of one token (word))
    - X is multiplied by three weights W^Q, W^K, W^V to get Q, K, V

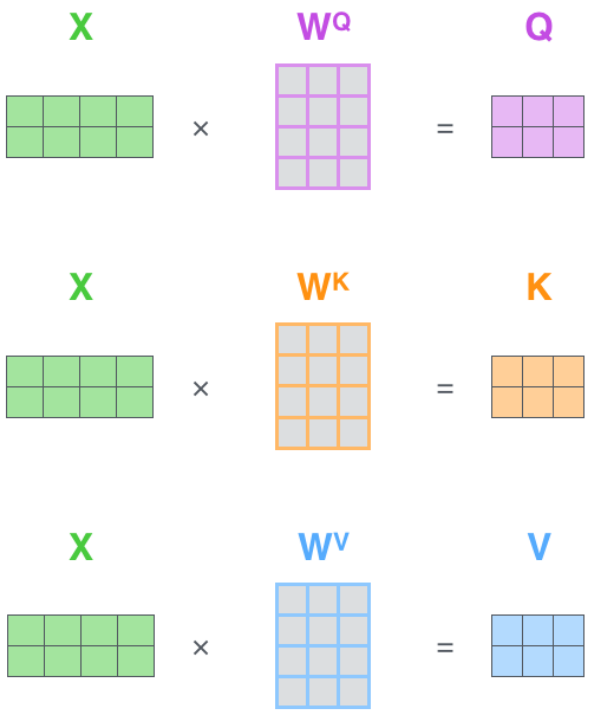


Image source: towarddatascience.com

Attention mechanism

- Attention mechanism is mostly based on three things
  - Query (Q): what kind of information we are looking for
  - Key (K): relevance to the query
  - Value (V): actual contents of the input
  - Then, a dot product between query (Q) and key (K) to get the similarity (attention score) among different tokens
  - For example, if we have 6 tokens, QK^T would be

$$
\begin{array}{c}
 \\
Hello \\
, \\
how \\
are \\
you \\
? \\
 \\
\end{array}
\begin{array}{c}
\begin{array}{cccccc} Hello & , & how & are & you & ? \end{array} \\
\begin{pmatrix}
72.40 * 10^{-06} & 1.23 * 10^{-21} & 6.51 * 10^{-40} & 2.62 * 10^{-22} & 9.99 * 10^{-01} & 4.30 * 10^{-08} \\
1.00 * 10^{+00} & 7.51 * 10^{-30} & 1.54 * 10^{-17} & 9.91 * 10^{-13} & 8.15 * 10^{-69} & 1.09 * 10^{-30} \\
3.12 * 10^{-70} & 2.51 * 10^{-51} & 2.72 * 10^{-21} & 8.03 * 10^{-09} & 1.29 * 10^{-07} & 9.99 * 10^{-01} \\
2.47 * 10^{-72} & 5.54 * 10^{-05} & 9.80 * 10^{-01} & 1.98 * 10^{-02} & 2.77 * 10^{-82} & 2.58 * 10^{-08} \\
2.67 * 10^{-05} & 1.21 * 10^{-09} & 9.75 * 10^{-07} & 3.17 * 10^{-76} & 9.99 * 10^{-01} & 3.64 * 10^{-28} \\
8.59 * 10^{-47} & 1.05 * 10^{-35} & 9.99 * 10^{-01} & 2.38 * 10^{-15} & 4.21 * 10^{-27} & 4.07 * 10^{-06}
\end{pmatrix}
\end{array}
\begin{array}{c}
 \\
= 1 \\
= 1 \\
= 1 \\
= 1 \\
= 1 \\
= 1
\end{array}
$$

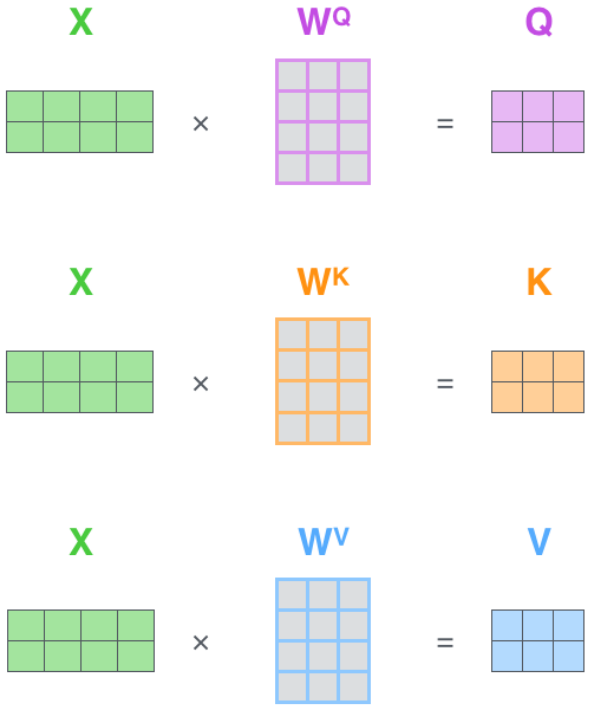
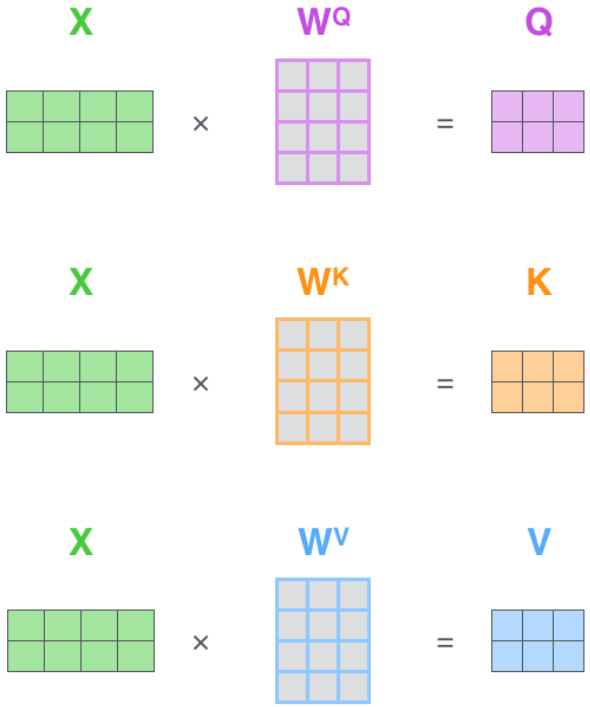


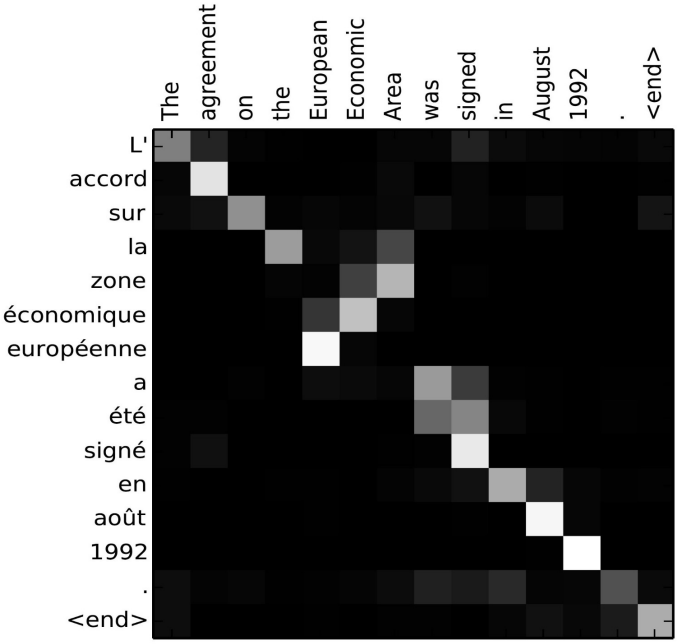
Image source: towarddatascience.com

Note:
In this example, query and key are the same. Hence, it is a self-attention

Attention mechanism

- Attention mechanism is mostly based on three things
  - Query (Q): what kind of information we are looking for
  - Key (K): relevance to the query
  - Value (V): actual contents of the input





Note:
In this figure, query and key are different

Image source: Bahdanau et al. (2015)

Attention mechanism

- Attention mechanism is mostly based on three things
  - Query (Q): what kind of information we are looking for
  - Key (K): relevance to the query
  - Value (V): actual contents of the input
  - Then, QK^T go through a scaled softmax and multiplied by value V

$$\text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

  - Hence, the result can be regarded as a weighted sum of V

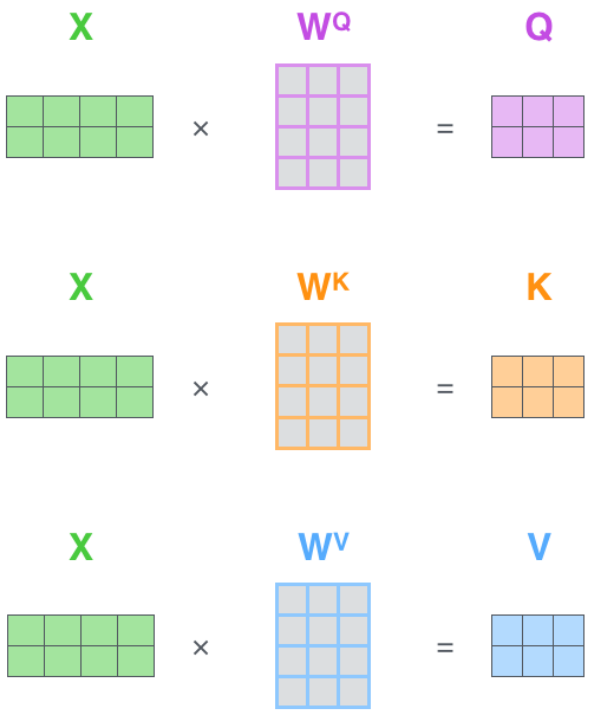
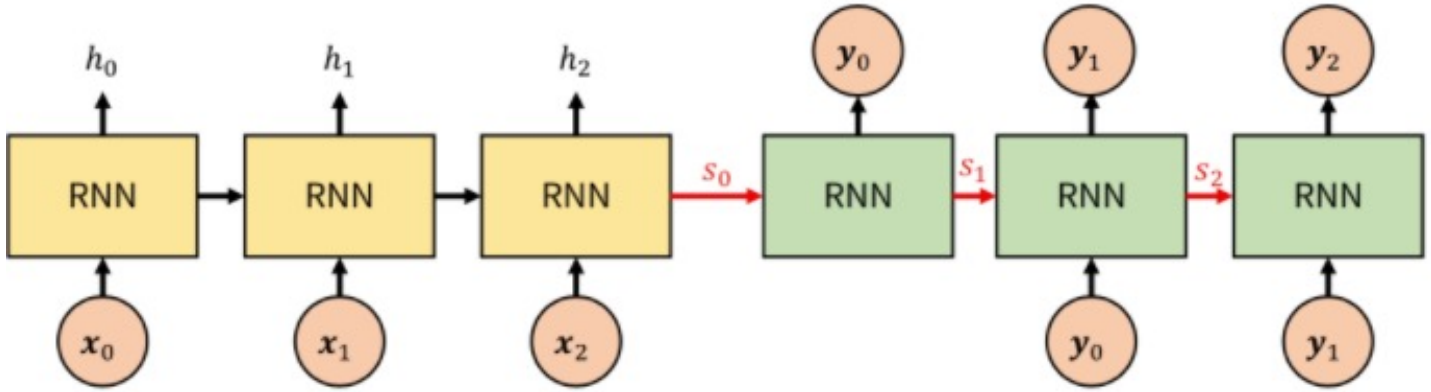


Image source: towarddatascience.com

Attention mechanism to seq2seq

- At first, the attention mechanism was used for seq2seq models



  - For example, the hidden states of the input (or encoder) part are used as **keys** and **values** and the hidden states of the output (or decoder) part are used as **queries**
  - In this regard, we can decide how much to attend the previous hidden state values to make an output for the current RNN cell

Attention is all you need (Transformers)

- Vaswani et al. (2017) proposed the Transformer model consists of only attention layers without RNN cells
  - The Transformer model showed very promising performances in various sequence related tasks
  - Also, one of the best part of the Transformers is that it is easy to parallelize the training of the model
  - Hence, we can easily build an extremely large models

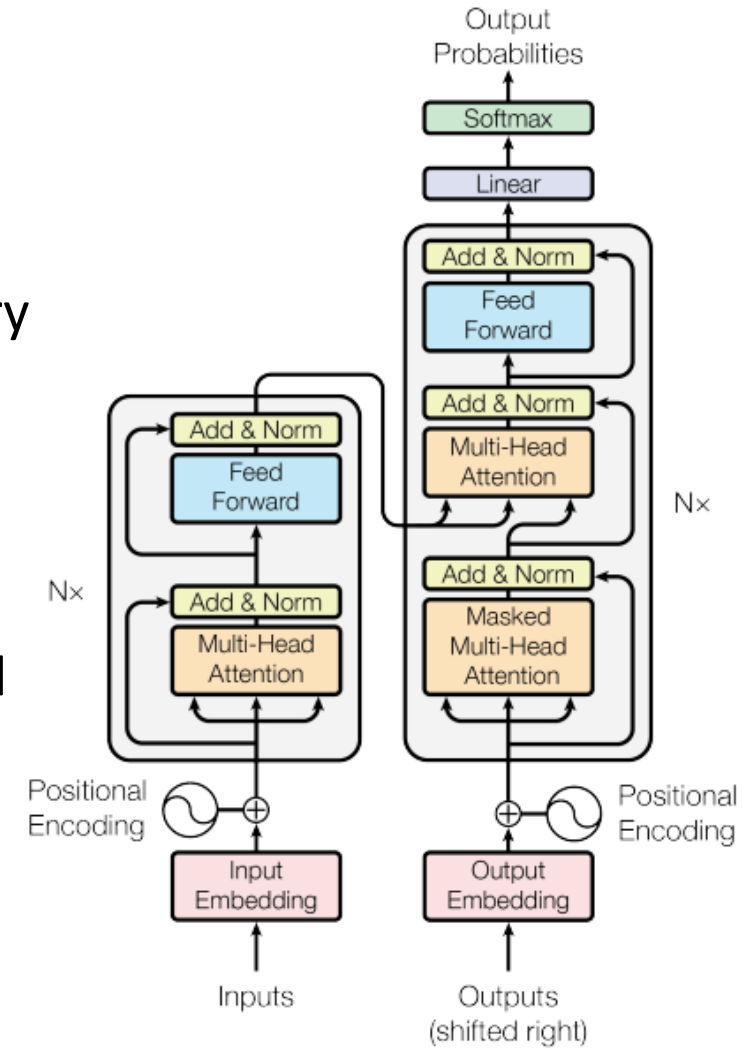
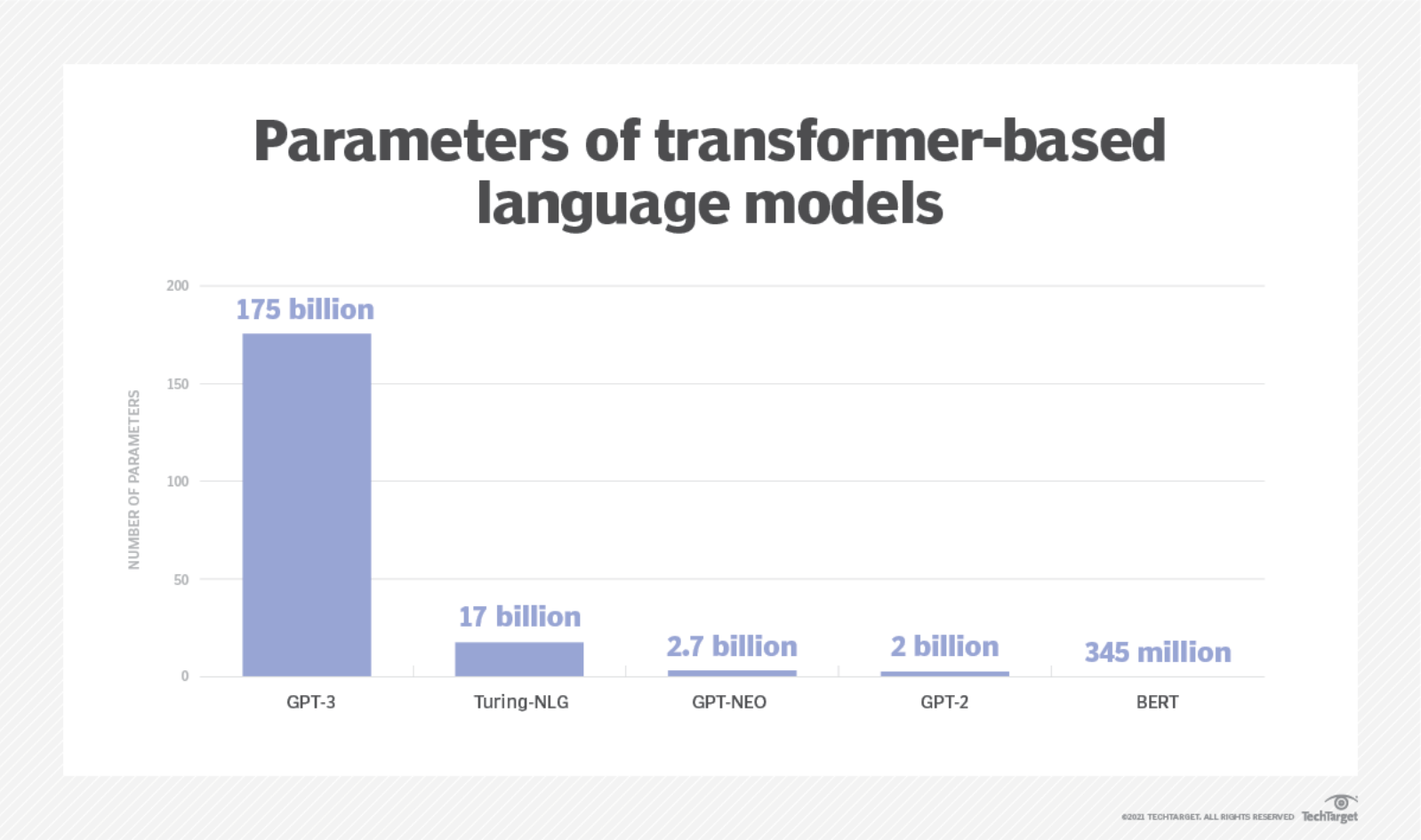


Figure 1: The Transformer - model architecture.

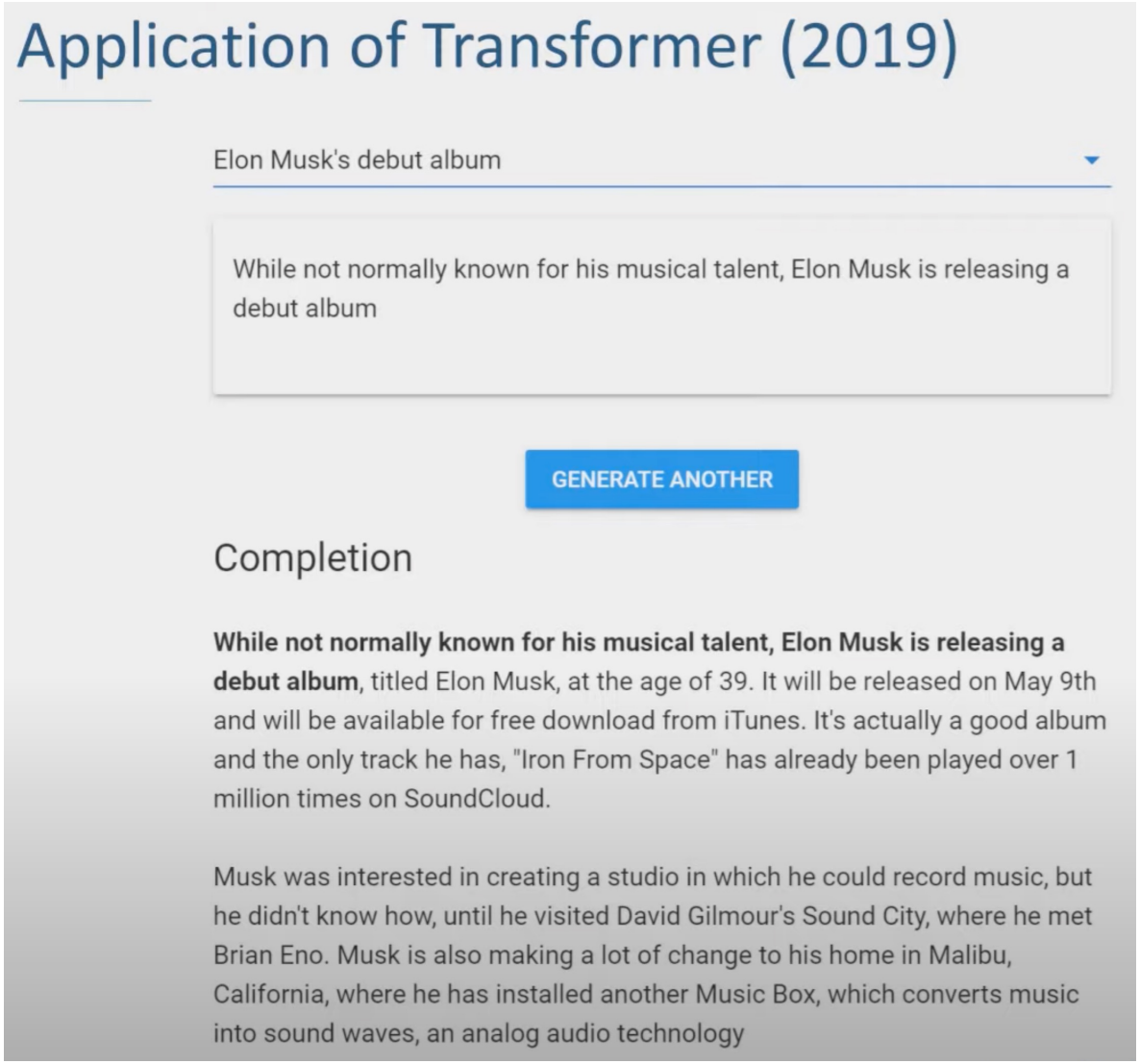
Attention is all you need (Transformers)

- Example:



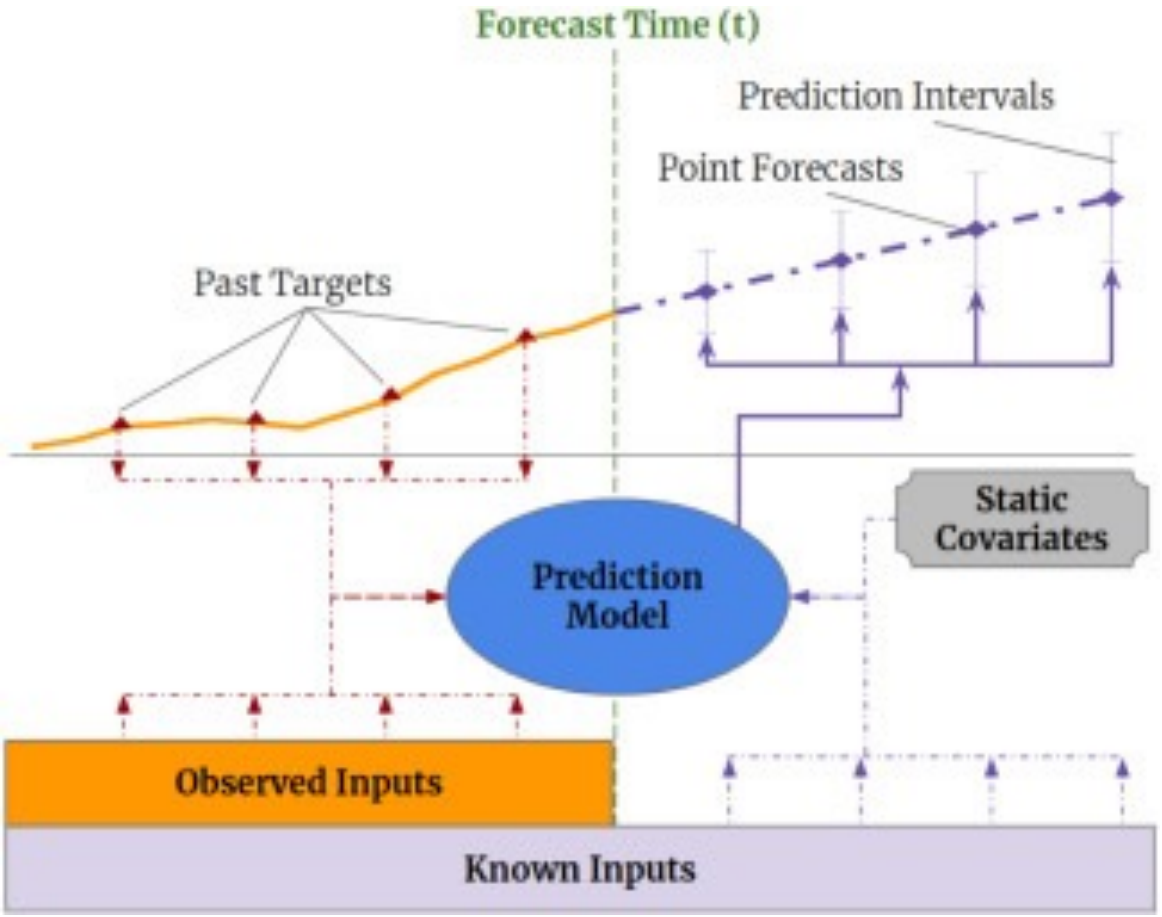
Attention is all you need (Transformers)

- Example:



Attention is all you need (Transformers)

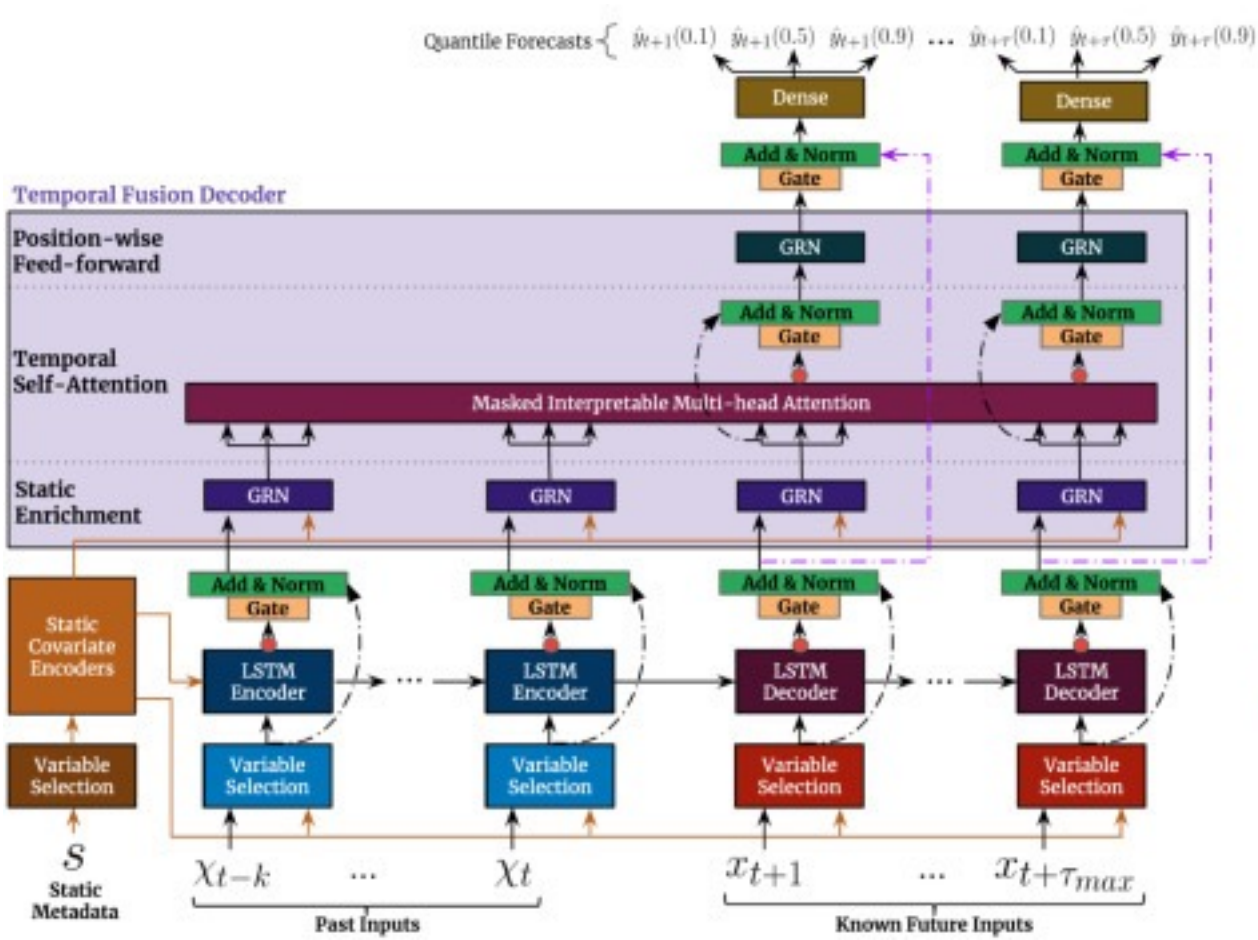
- Example: Temporal Fusion Transformers (Lim et al., 2021)

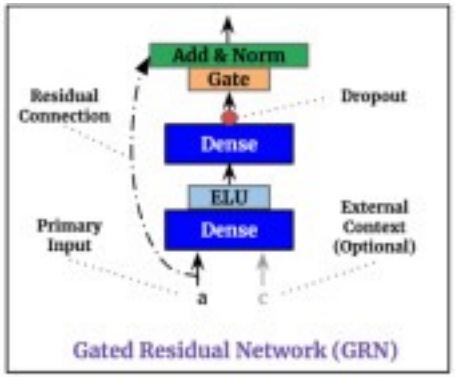


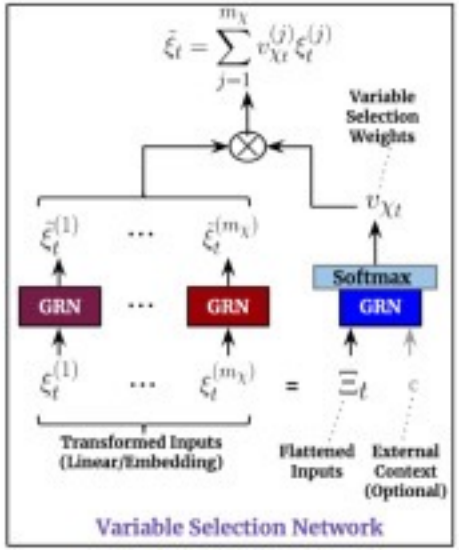
https://doi.org/10.1016/j.ijforecast.2021.03.012

Attention is all you need (Transformers)

- Example: Temporal Fusion Transformers (Lim et al., 2021)







https://doi.org/10.1016/j.ijforecast.2021.03.012

Attention is all you need (Transformers)

- Example: Temporal Fusion Transformers (Lim et al., 2021)

(a) S&P 500

(b) FTSE 100

(c) Nikkei 225

Fig. E.7. TFT one-step-ahead forecasts for realized volatility across major stock indices following the incidence of COVID-19. From the target values in purple, we note the regime shift following the sharp increase in index volatilities in March 2020, which later decay to elevated baseline levels over 2020. While volatilities in March 2020 do appear at or around the 0.975 quantile forecast, the TFT maintains reasonable uncertainty estimates over the regime shift – with most values lying within the 95% credible interval – demonstrating its ability to adapt to changing temporal dynamics. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

https://doi.org/10.1016/j.ijforecast.2021.03.012

Attention is all you need (Transformers)

- Example: Informer (Zhou et al., 2021)

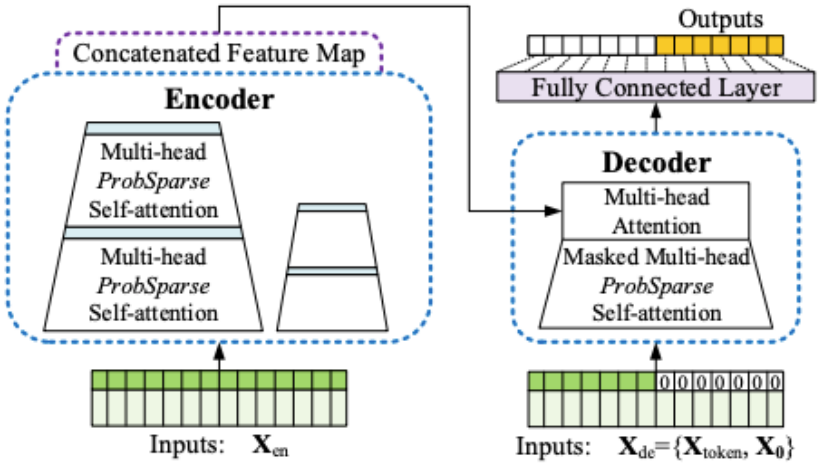


Figure 2: Informer model overview. Left: The encoder receives massive long sequence inputs (green series). We replace canonical self-attention with the proposed *ProbSparse* self-attention. The blue trapezoid is the self-attention distilling operation to extract dominating attention, reducing the network size sharply. The layer stacking replicas increase robustness. Right: The decoder receives long sequence inputs, pads the target elements into zero, measures the weighted attention composition of the feature map, and instantly predicts output elements (orange series) in a generative style.

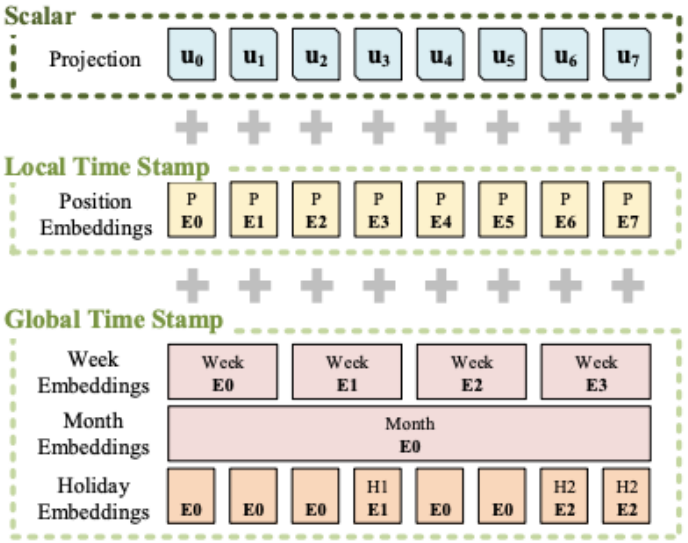


Figure 6: The input representation of Informer. The inputs's embedding consists of three separate parts, a scalar projection, the local time stamp (Position) and global time stamp embeddings (Minutes, Hours, Week, Month, Holiday etc.).

https://doi.org/10.1609/aaai.v35i12.17325

Attention is all you need (Transformers)

- Example: Autoformer (Wu et al., 2021)

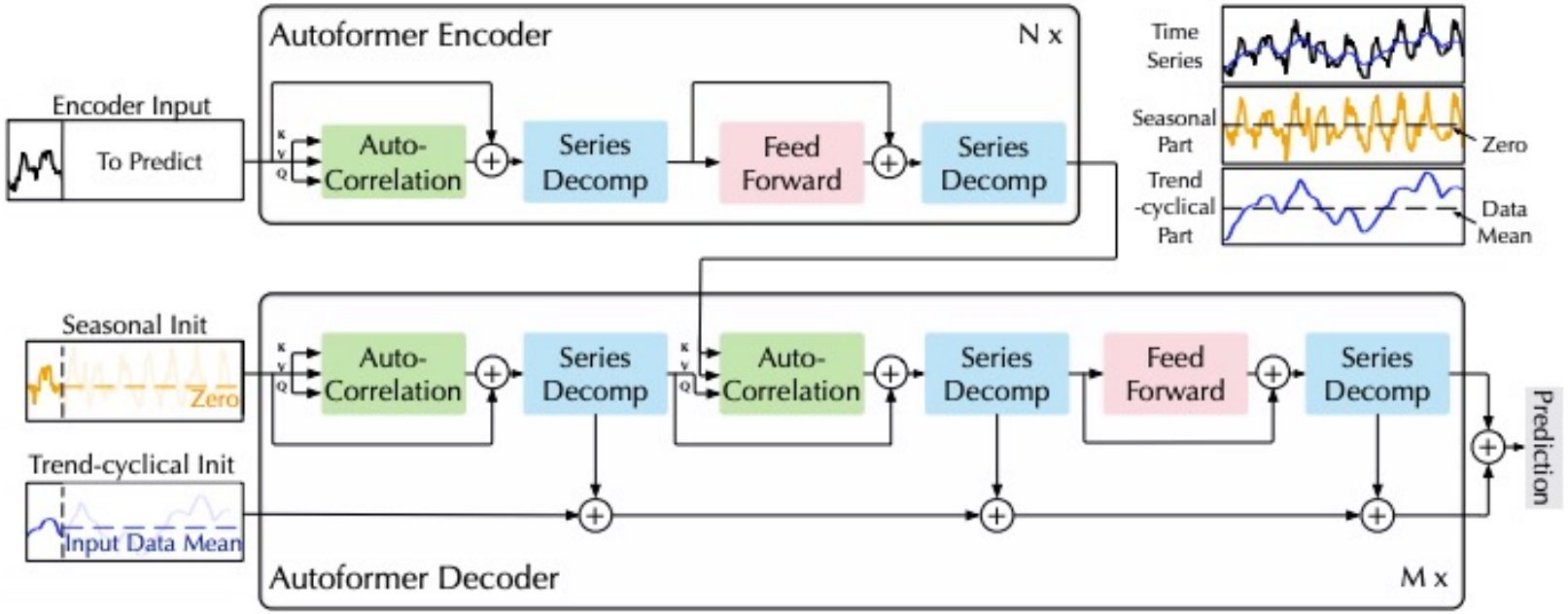


Figure 1: Autoformer architecture. The encoder eliminates the long-term trend-cyclical part by series decomposition blocks (blue blocks) and focuses on seasonal patterns modeling. The decoder accumulates the trend part extracted from hidden variables progressively. The past seasonal information from encoder is utilized by the encoder-decoder Auto-Correlation (center green block in decoder).

https://proceedings.neurips.cc/paper/2021/hash/bcc0d400288793e8bdcd7c19a8ac0c2b-Abstract.html

Attention is all you need (Transformers)

- Example: Autoformer (Wu et al., 2021)

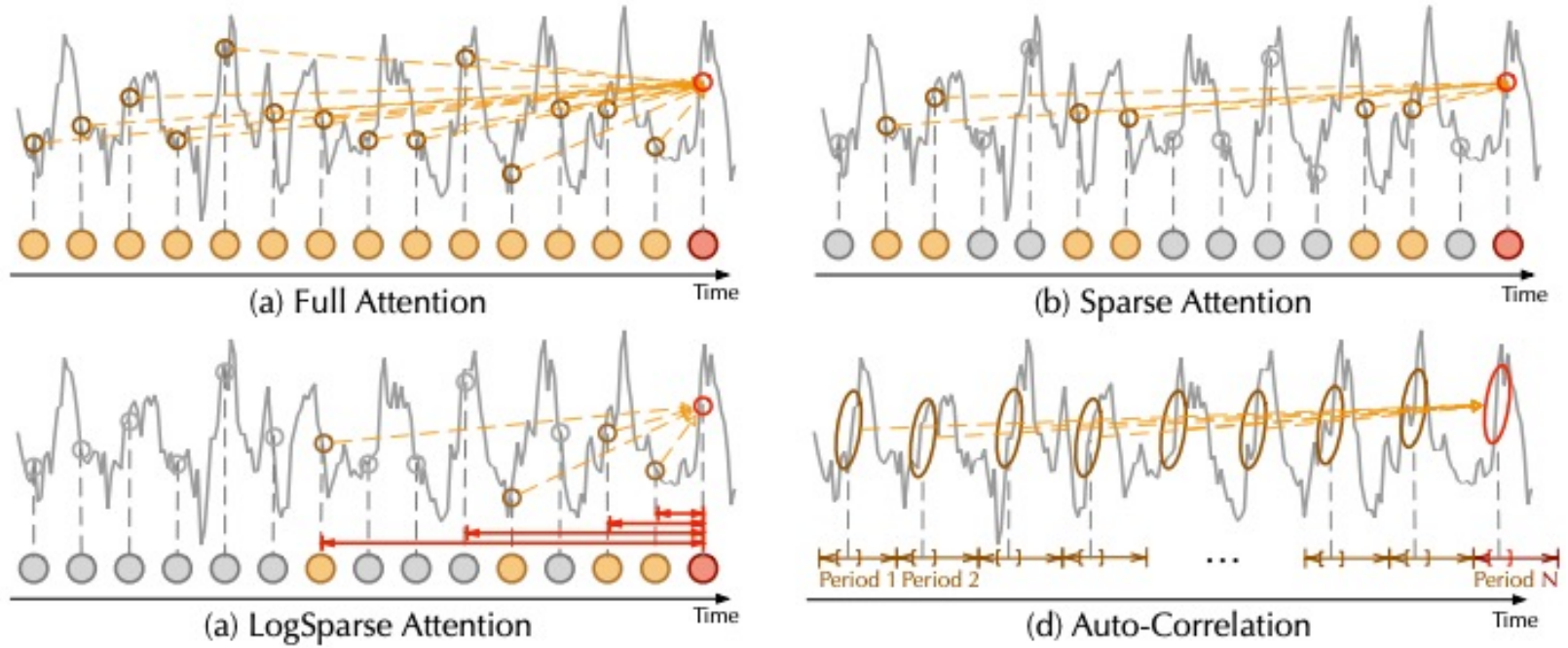


Figure 3: Auto-Correlation vs. self-attention family. Full Attention [41] (a) adapts the fully connection among all time points. Sparse Attention [23, 48] (b) selects points based on the proposed similarity metrics. LogSparse Attention [26] (c) chooses points following the exponentially increasing intervals. Auto-Correlation (d) focuses on the connections of sub-series among underlying periods.

https://proceedings.neurips.cc/paper/2021/hash/bcc0d400288793e8bdcd7c19a8ac0c2b-Abstract.html