Time-series forecasting with deep learning: a survey

Group D

Karan Jeswani, Yuanting Li, Tianhong Gao, Yiqing Zhu

October 14, 2023

Outline

Introduction

Deep Learning Use Cases

- image classification
- natural language processing
- reinforcement learning

Why Deep Learning

- DNN are able to learn complex data representations, which alleviates the need for manual feature engineering
- availability of open-source backpropagation frameworks [16,17] has also simplified the network training

Objectives

Deep learning architectures for time-series forecasting

- Convolutional neural networks
- Recurrent neural networks
- Attention mechanisms
- Outputs and loss functions

Multi-horizon forecasting models & uncertainty estimation

- Iterative methods
- Direct methods

Objectives

Hybrid Models

- Non-probabilistic hybrid models
- Probabilistic hybrid models

facilitate decision support

- Interpretability with time-series data
- Counterfactual predictions and causal inference over time

Architecture

The basic architecture of DNN

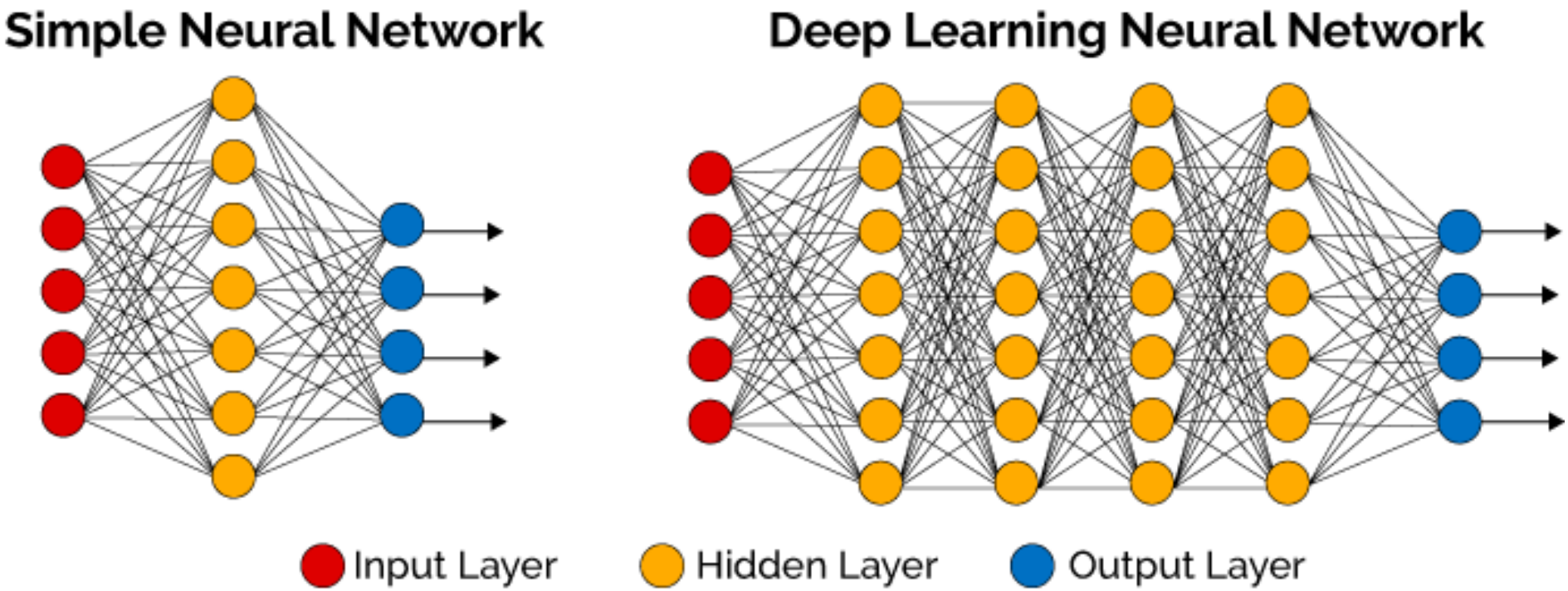


Figure: Architecture of DNN

Perceptron

Graphical Representation of working of a single node in a DNN

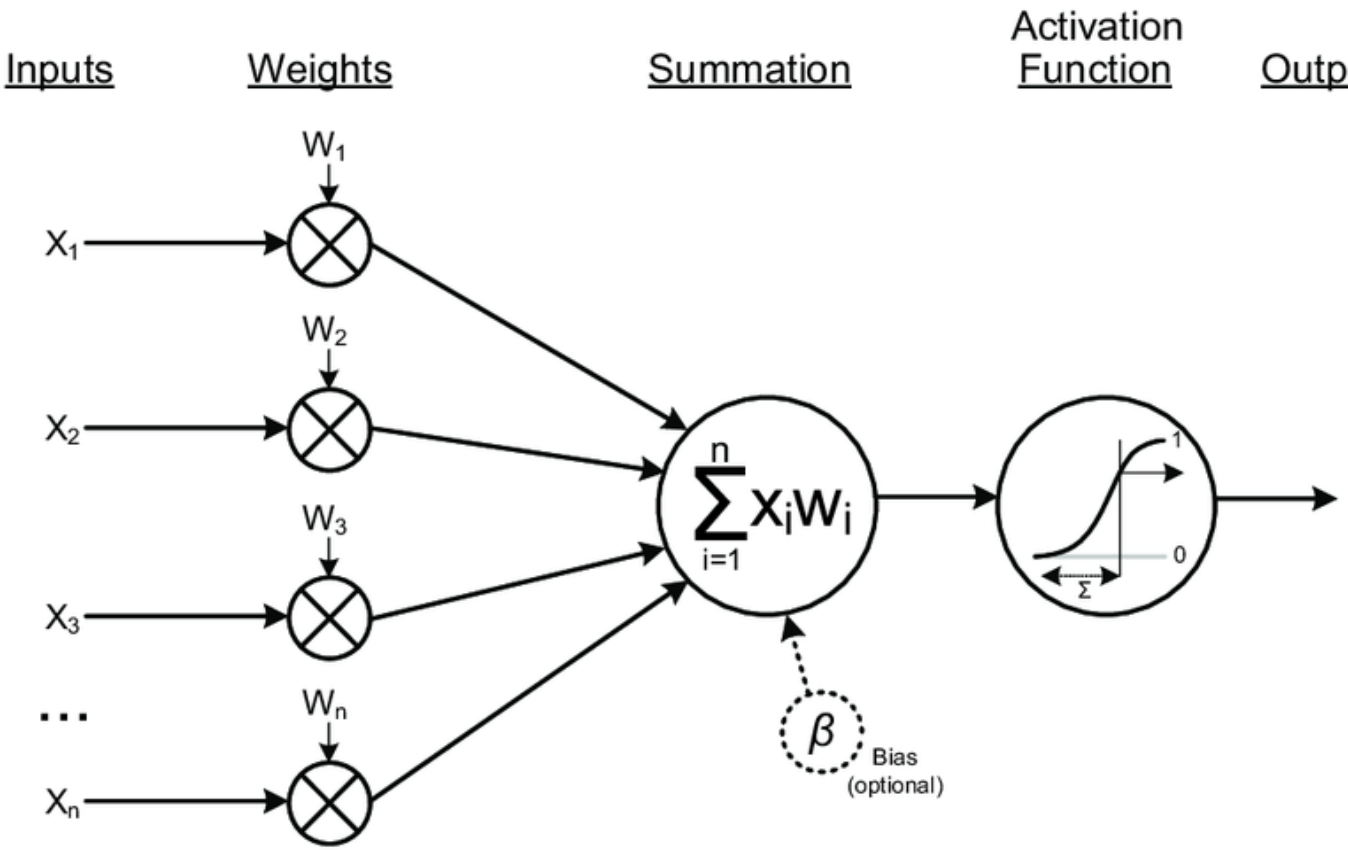


Figure: Artificial Neural Network (ANN), with a sigmoidal activation function

Multi-layer Perceptron

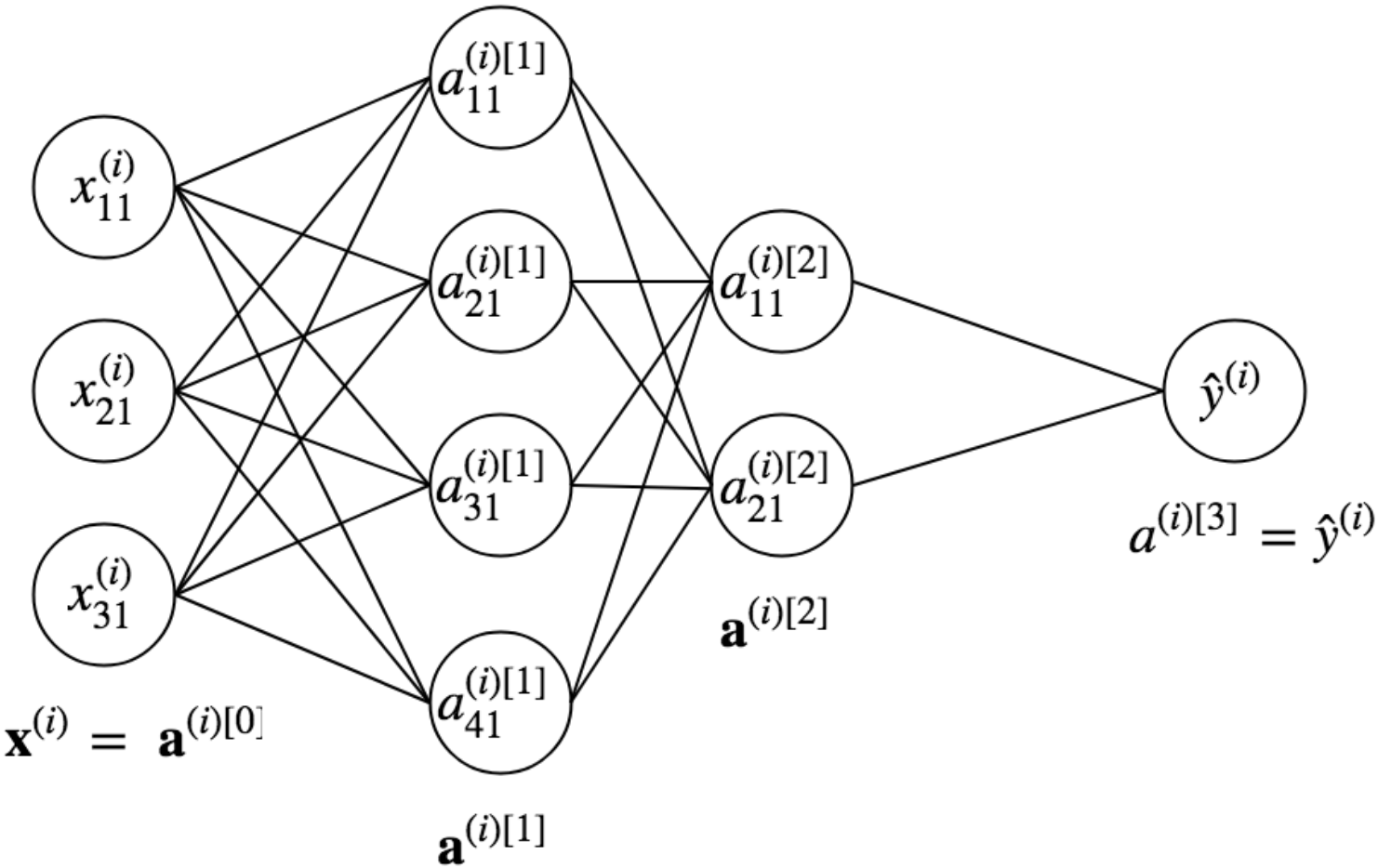


Figure: Forward Propogation in DNN , We have hidden layers $a^{(i)}[1]$ and $a^{(i)}[2]$, and output layer $a^{(i)}[3] = \hat{y}^{(i)}$

Time Series Forecasting using DNN

Time-series forecasting models predict future values of a target yi,t for a given entity i at time t. In the simplest case, one-step-ahead forecasting models take the form

one-step-ahead forecasting

$$\hat{y}_{i,t+1} = f(y_{i,t-k:t}, x_{i,t-k:t}, s_i)$$

where,
$\hat{y}_{i,t+1}$ is the model forecast,
$y_{i,t-k:t} = \{y_{i,t-k}, \ldots, y_{i,t}\}$,
$x_{i,t-k:t} = \{x_{i,t-k}, \ldots, x_{i,t}\}$, are observations of the target and exogenous inputs respectively over a look-back window k,
s_i is static metadata associated with the entity (e.g. sensor location),
$f(.)$ is the prediction function learnt by the model.

Latent Variable

Deep Neural Networks encode relevant historical information into a latent variable $\mathbf{z}_t$, with the final forecast produced using $\mathbf{z}_t$ alone

Prediction Using Latent Variable

$$f(y_{t-k:t}, x_{t-k:t}, s) = g_{\text{dec}}(z_t)$$

$$z_t = g_{\text{enc}}(y_{t-k:t}, x_{t-k:t}, s)$$

where $g_{\text{enc}}(.), g_{\text{dec}}(.)$ are encoder and decoder functions respectively **Encoders** and **Decoders** hence form the basic building blocks of deep learning architectures

Convolutional neural networks

- Traditionally designed for image datasets
- Extract local relationships that are invariant across spatial dimensions

Causal convolutional filter, for intermediate feature at hidden layer l

$$h_t^{l+1} = A\left(W * h\right)^{(l,t)}$$

$$\left(W * h\right)^{(l,t)} = \sum_{\tau=0}^{k} W^{(l,\tau)} h_{t-\tau}^{l}$$

Where $h_t^l \in \mathbb{R}^{H_{\text{in}}}$ is an intermediate state at layer l at time t, $*$ is the convolution operator, $W^{(l,\tau)} \in \mathbb{R}^{H_{\text{out}} \times H_{\text{in}}}$ is a fixed filter weight at layer l, and $A(.)$ is an activation function, such as a sigmoid function.
For CNNs that use a total of L convolutional layers, we note that the encoder output is then $z_t = h_t^L$.

CNN:Example

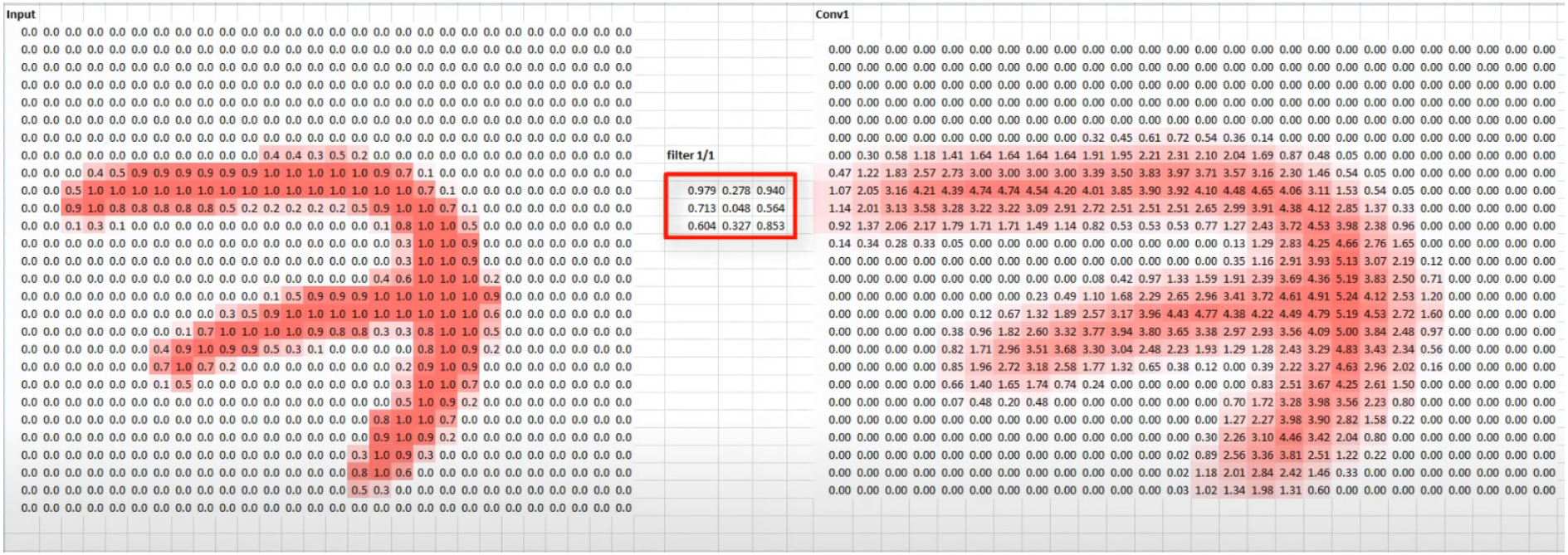


Figure: Convolution of Input Image with Filter

CNN:Example

Figure: Each filter detects an edge

Connection with Time Series Analysis

Considering the 1-D case, we can see that CNN equation bears a strong resemblance to finite impulse response (FIR) filters in digital signal processing

FIR Model

$$y[k] = \begin{cases} \sum_{n=0}^{k} g[n]u[k-n] + v[k], & \text{if } k \leq M-1 \\ \sum_{n=0}^{M} g[n]u[k-n] + v[k], & \text{if } k > M \end{cases}$$

- Highly useful in giving estimates of time-delay, system dynamics, and orders.
- It also belongs to the family of parametric models, as we shall shortly learn.
- **Estimation:** FIR models are estimated using (i) correlation analysis (ii) least squares methods (same as correlation analysis) or (iii) subspace methods.
- **Demerits:** Does not model the noise component.

Dilated convolutions

Using standard convolutional layers can be computationally challenging where long-term dependencies are significant, as the number of parameters scales directly with the size of the receptive field. Modern architectures alleviate this by using **dilated convolutional layers**.

Dilated Convolution Equation

$$(W * h)(l, t, d_l) = \sum_{\tau=0}^{\lfloor k/d_l \rfloor} W(l, \tau) h_l^{t-d_l\tau}$$

Where $\lfloor . \rfloor$ is the floor operator and d_l is a layer-specific dilation rate. By increasing the dilation rate with each layer, dilated convolutions efficiently aggregate historical information across different time blocks.

WaveNet Architecture

Dilation rates increase in powers of 2, allowing 2^l time steps to be used at layer l.

Dilated convolutions: Example

Dilated Convolutions with Dilation Rate of $2^1 = 2$

Input Sequence:

$$[a, b, c, d, e, f, g]$$

Filter:

$$[x, y, z]$$

Applying Filter with Dilation Rate of 2:

Input Segments:

$$[a, \underline{c}, e], [b, \underline{d}, f], [c, \underline{e}, g]$$

Resulting Sequence:

$$[ax + cy + ez], [bx + dy + fz], [cx + ey + gz]$$

Recurrent neural networks

Issues in the feed forward neural network

- Can't handle sequential data.
- Consider only current input.
- Can't memorize the previous input.

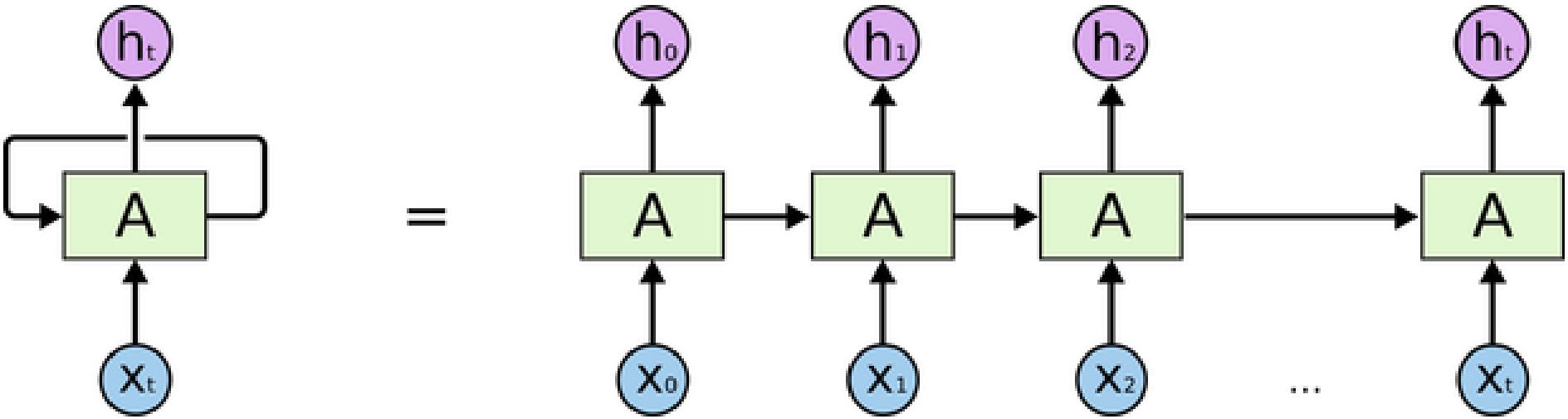


Figure: Basic Architecture of RNN

Recurrent neural networks

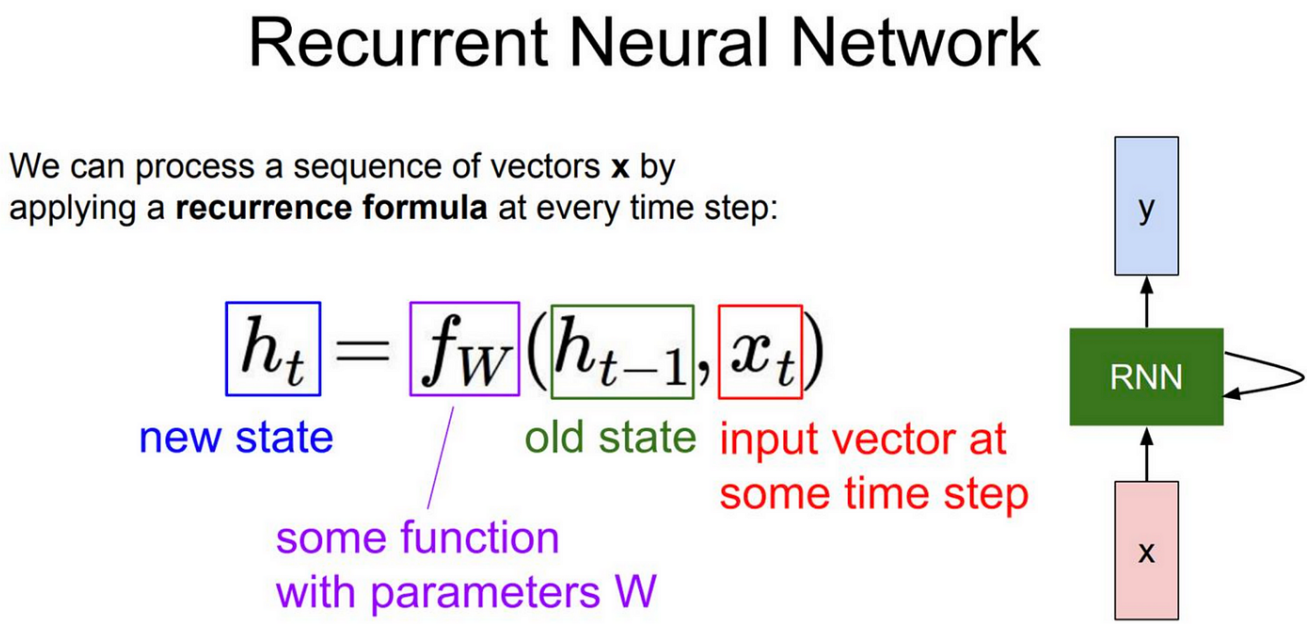


Figure: Basic Equation of RNN

As explained by the diagram, in RNN, the output of any layer not only depends on the current input but also on the set of inputs that have came before. This special feature provides it a significant advantage over other neural networks by taking help of inputs obtained before to predict outputs at the later stage.

Exploding and Vanishing Gradients

As we know, each layer computer a function of the current input and the previous hidden activations i.e

$$h(t) = f\left(x(t), h(t-1)\right)$$

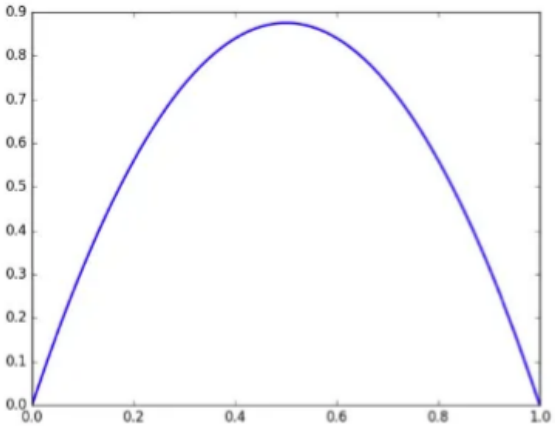
by expanding it recursively, we get

$$h(4) = f\left(f\left(f\left(h(1), x(2)\right), x(3)\right), x(4)\right)$$

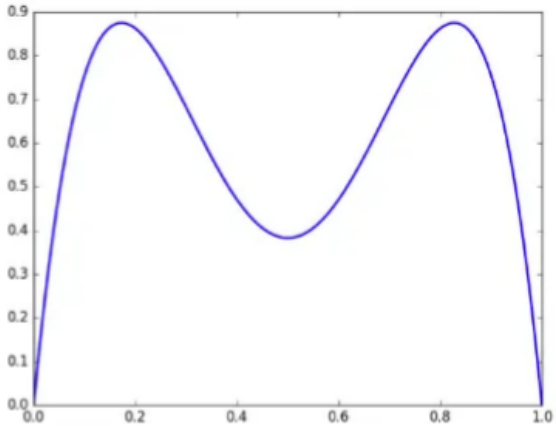
Let's now understand the above concept by taking an example of simple quadratic function

$$f(x) = 3.5x(1-x)$$

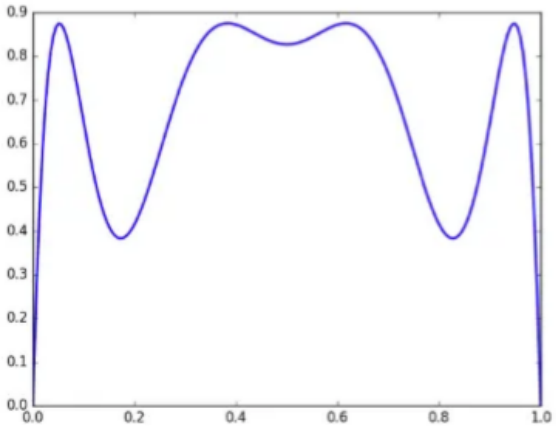
Exploding and Vanishing Gradients



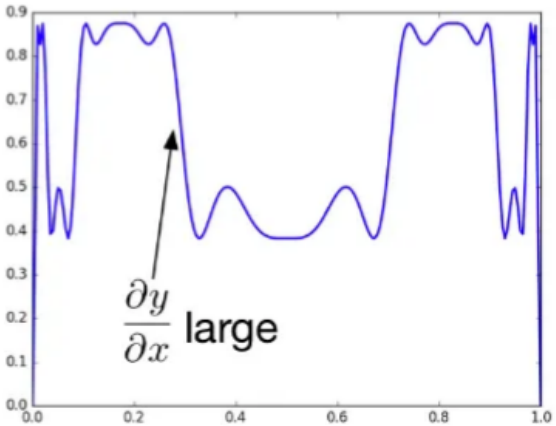
$y = f(x)$



$y = f(f(x))$



$y = f(f(f(x)))$



$y = \underbrace{f \circ \cdots \circ f}_{\text{6 times}}(x)$

Long Short-Term Memory Model

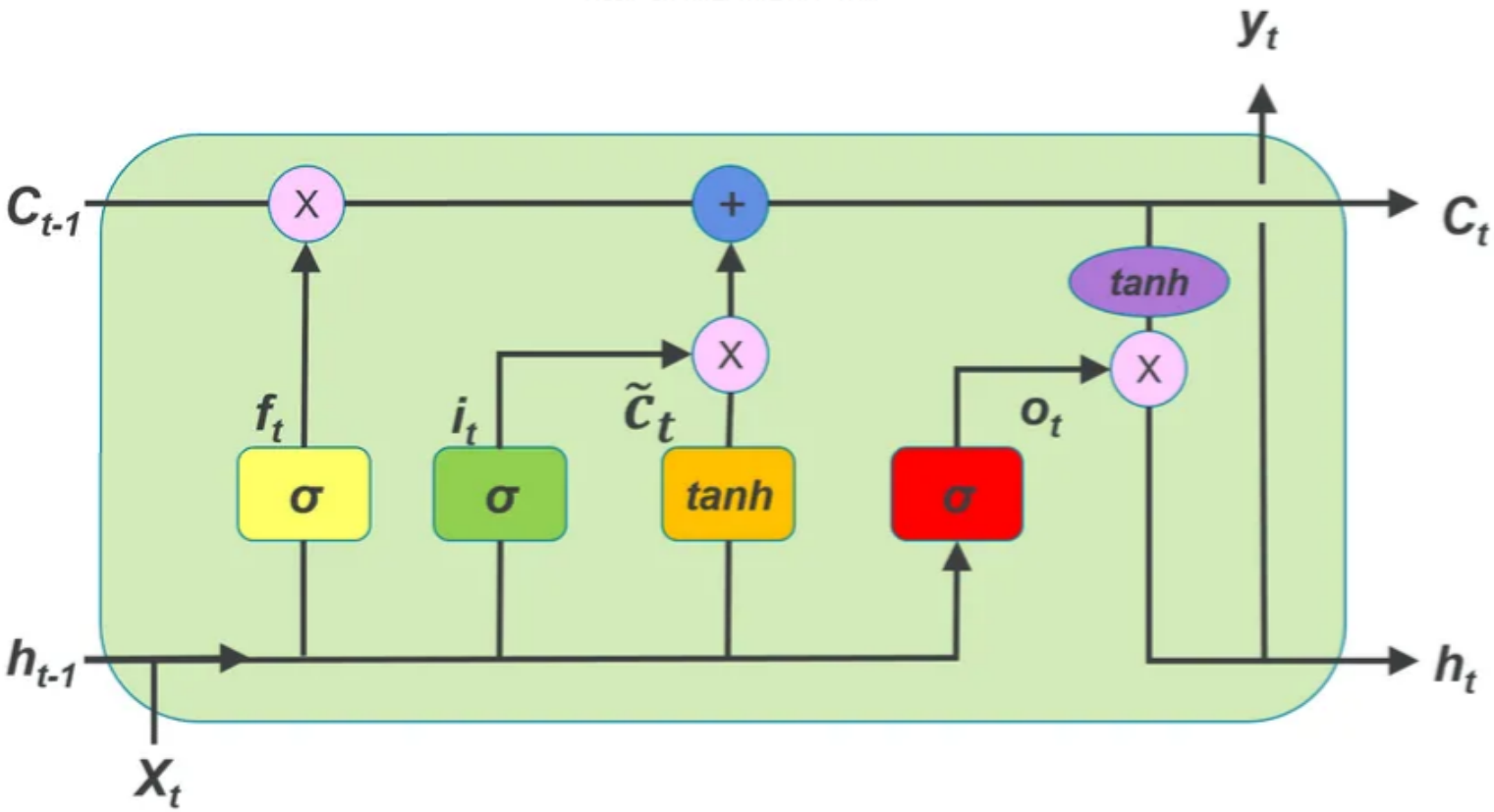


Figure: Basic Architecture of LSTM

Deciding how much of the past it should remember

First step in the LSTM is to decide which information to be omitted in from the cell in that particular time step. It is decided by the sigmoid function. It looks at the previous state (h_{t-1}) and the current input x_t and computes the function.

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

f_t = forget gate
Decides which information to delete that is not important from previous time step

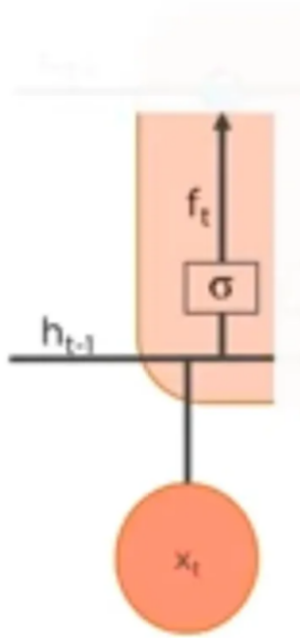


Figure: The forget gate takes the information from the current input and previous hidden state and decides what information should be discarded or retained. Since $\sigma(x)$ interpolates between 0 and 1, the values of f_t ranges between these values. The closer f_t is to 0, the smaller is the contribution of c_{t1} to c_t. A value close to 1 means keeping c_{t1}.

Decide how much should this unit add to the current state

In the second layer, there are 2 parts. One is the sigmoid function and the other is the tanh. In the *sigmoid* function, it decides which values to let through(0 or 1). *tanh* function gives the weightage to the values which are passed deciding their level of importance(-1 to 1).

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

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

i_t = input gate
Determines which information to let through based on its significance in the current time step

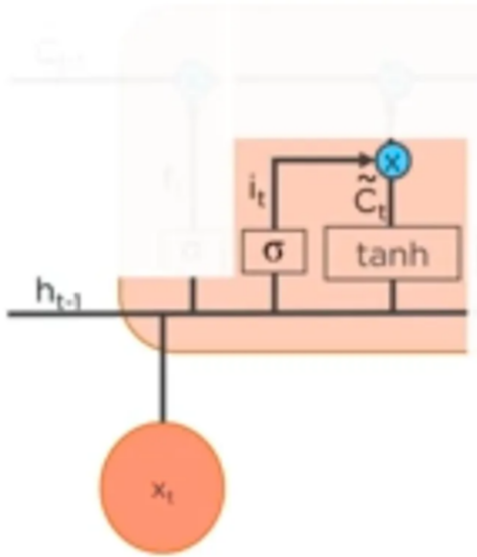


Figure: The input gate updates the cell state it two steps: (i) We pass the previous hidden state and current input into the sigmoid activation. (ii) We also pass the previous hidden state and current input into the tanh activation. The result is multiplied by the output of the sigmoid. This determines the extent to which the output of the tanh function will impact the update of the cell state (depending on how close i_t is to 0 or 1).

Decide what part of the current cell makes it to the output

The third step is to decide what will be our output. First, we run a sigmoid layer which decides what parts of the cell state make it to the output. Then, we put the cell state through tanh to push the values to be between -1 and 1 and multiply it by the output of the sigmoid gate.

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

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

o_t = output gate
Allows the passed in information to impact the output in the current time step

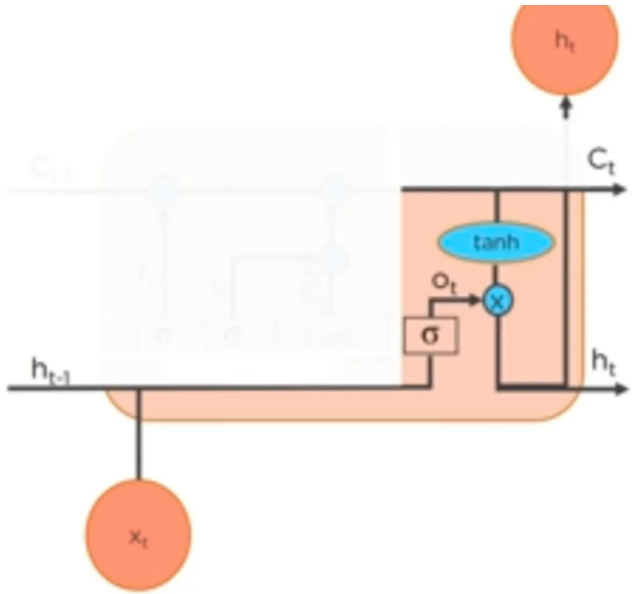


Figure: We pass the previous hidden state and the current input to the sigmoid activation. Then we pass the updated cell state and the previous hidden state to the tanh activation. We multiply the outputs of the two activation functions to decide what information the hidden state should carry.

Encoder-Decoder Neural Network

Before we introduce the attention mechanism, we need to know the Encoder-Decoder model. Let's see an example in machine translation.

- Origin English Sentence: Let's go. $\Rightarrow$ Spanish sentence: Vamos.
- We need to use Encoder to translate English words into vectors. And use Decoder to translate vectos into Spanish.

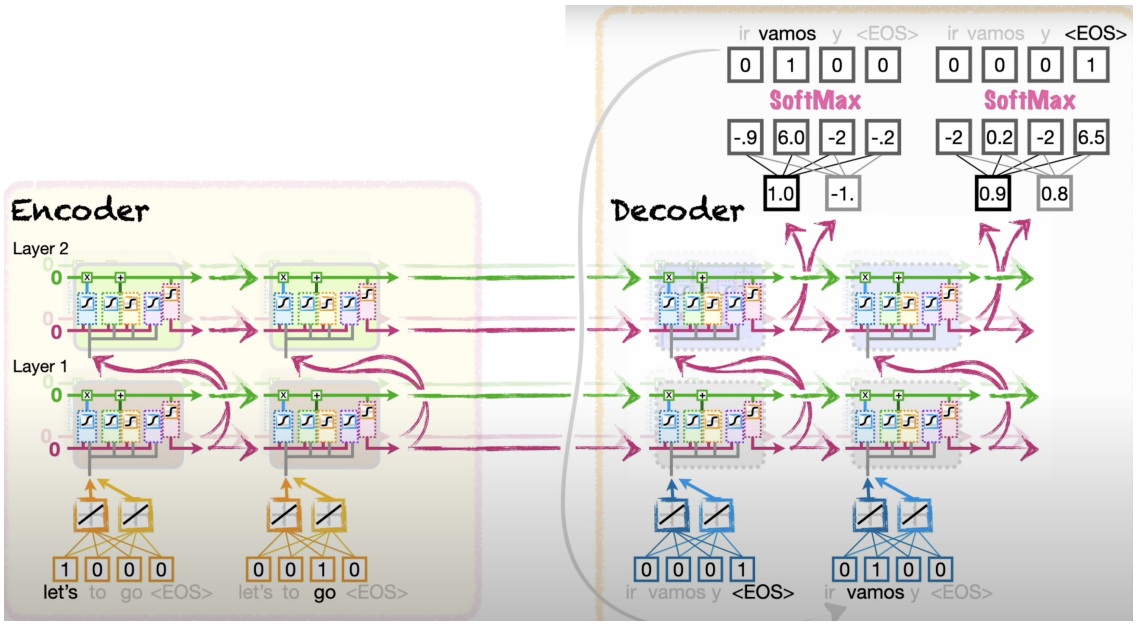
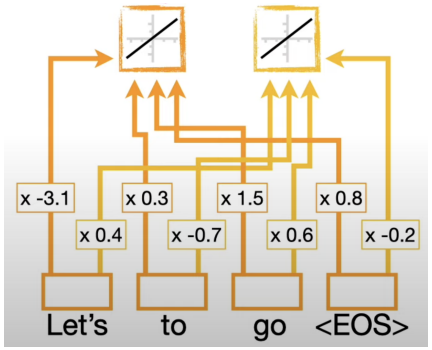


Figure: An example of Encoder-Decoder NN

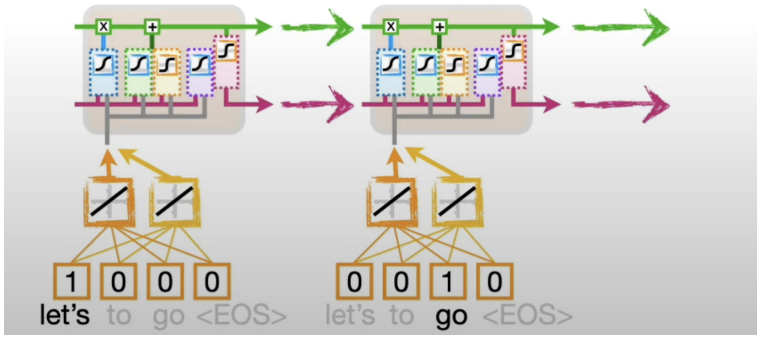
Encoder Part

It looks much more complicated than the simple LSTM. But actually it's based on LSTM.

- We use embedding layers to read the input words. Then pass the word vector in their order in the sentence.



(a) Weights and Bias

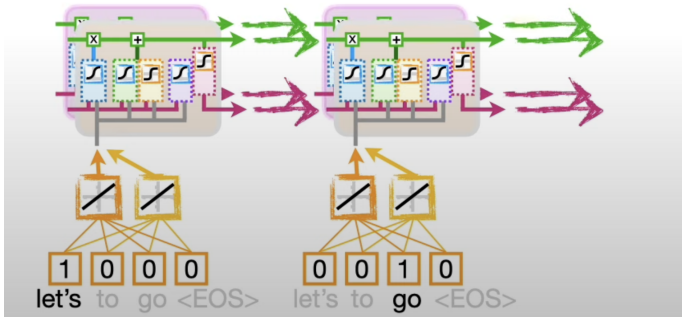


(b) Simple Encoder

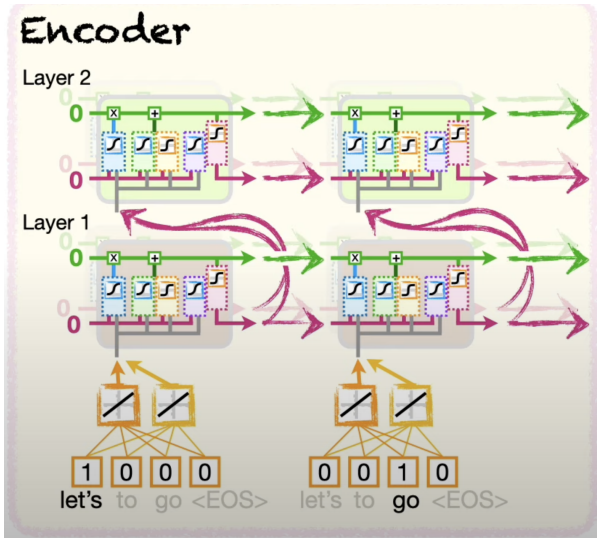
Encoder part

Then we improve our simple model, it outputs a Context Vector.

- To get better prediction, we often use more cells and more layers.



(a) More cells



(b) More layers

Decoder part

The Decoder is also a combination of LSTM.

- It uses the Context Vector to initialized the LSTM cells. And EOS as the first input.
- Use SoftMax function to output the word vector corresponding in the Spanish vocabulary.

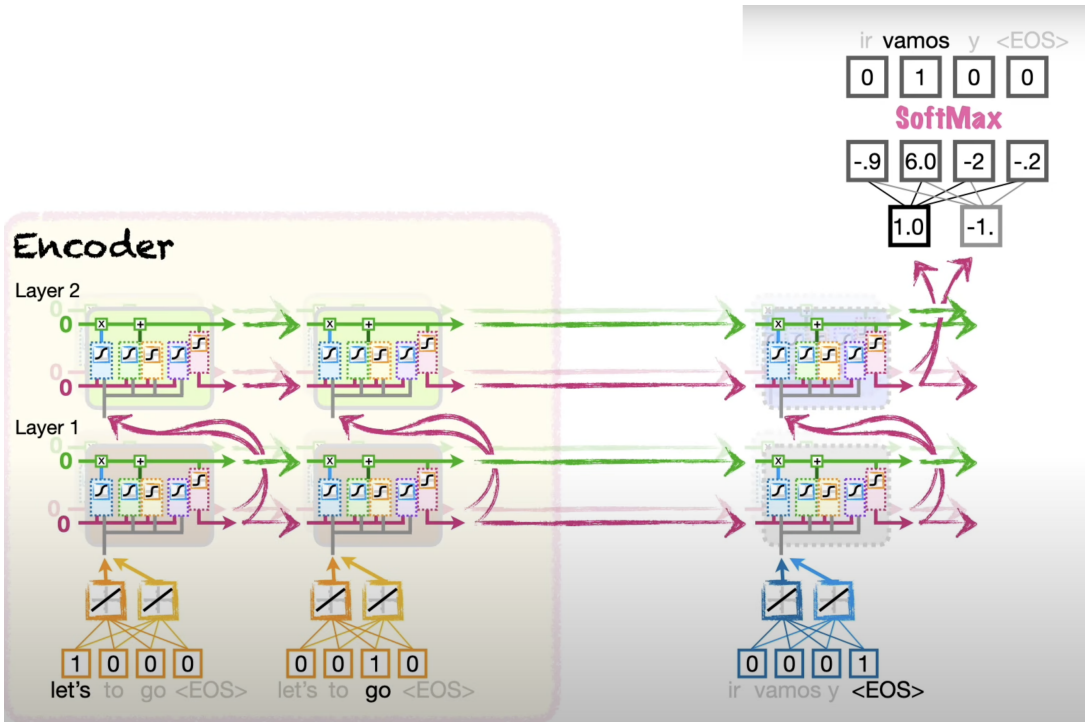


Figure: Decoder architecture

Decoder part

- Then the decoder uses the first output as input, to find the next word.
- When the next word is EOS , the translation stops.

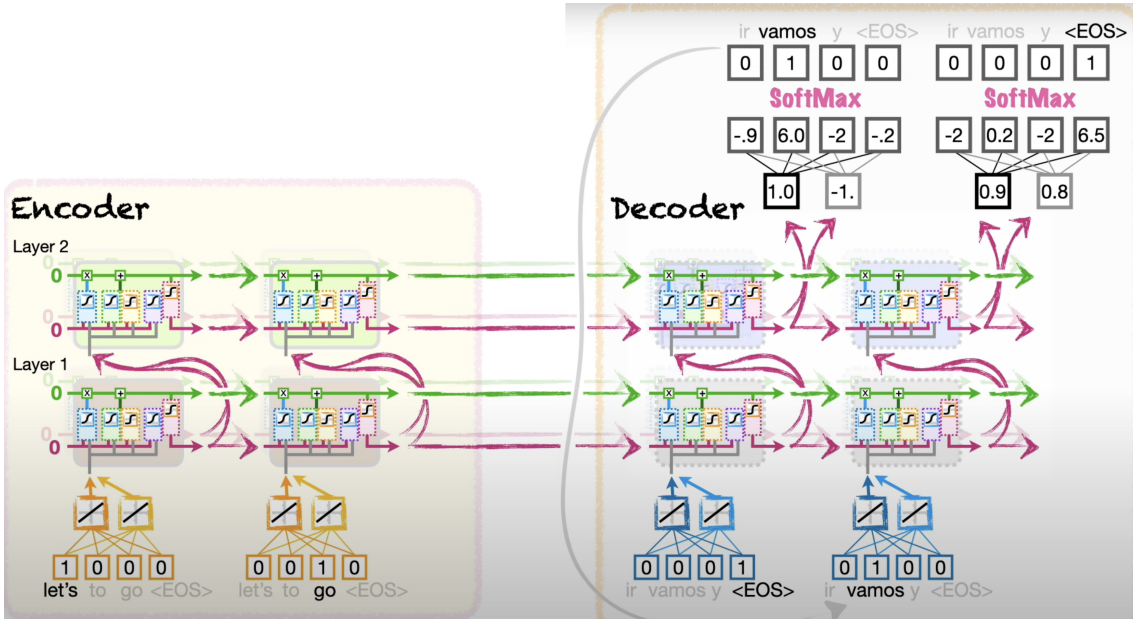


Figure: An example of Encoder-Decoder NN

Attention!! Please consider what will happen if the sentence is very long.

Back to Attention Mechanism

Let's see this sentence.

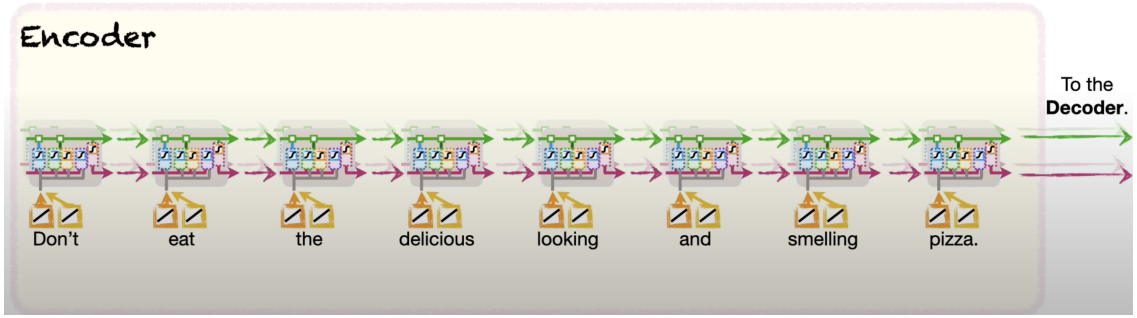


Figure: Very long sentence!

The meaning is totally different if the first word "don't" is not translated!

Figure: The origin Encoder-Decoder NN

Back to Attention Mechanism

Just as the previous slides, when you see the Context Vector and the Attention!! keywords. You notice them! We use the same idea.

- For each words, we can give it a weight calculated from the data in Encoder and Decoder.

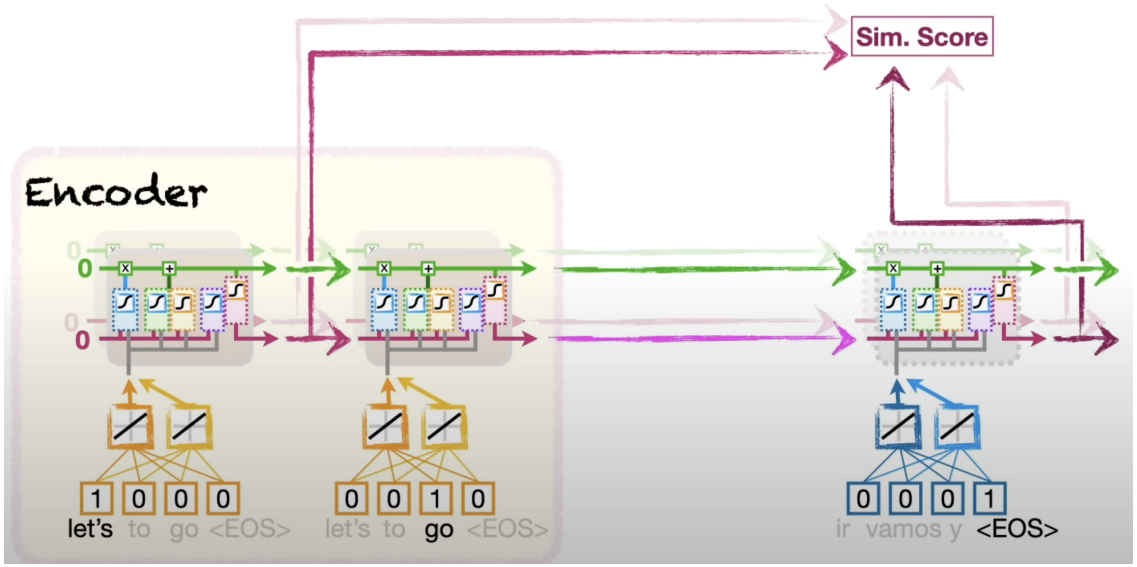


Figure: Calculate the weight

Back to Attention Mechanism

Here is an example of how the weight is calculated.

Weight calculation(FYI)

Here we often use the similarity of the vectors. For example, the Cosine similarity/inner product similarity.

	Cell #1	Cell #2
E: Let's	-0.76	0.75
D: EOS	0.91	0.38

$$\textit{Cosine Similarity} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} = -0.39$$

Back to Attention Mechanism

Use it, we can make predictions with different weights. In other words, the model is given Attention!.

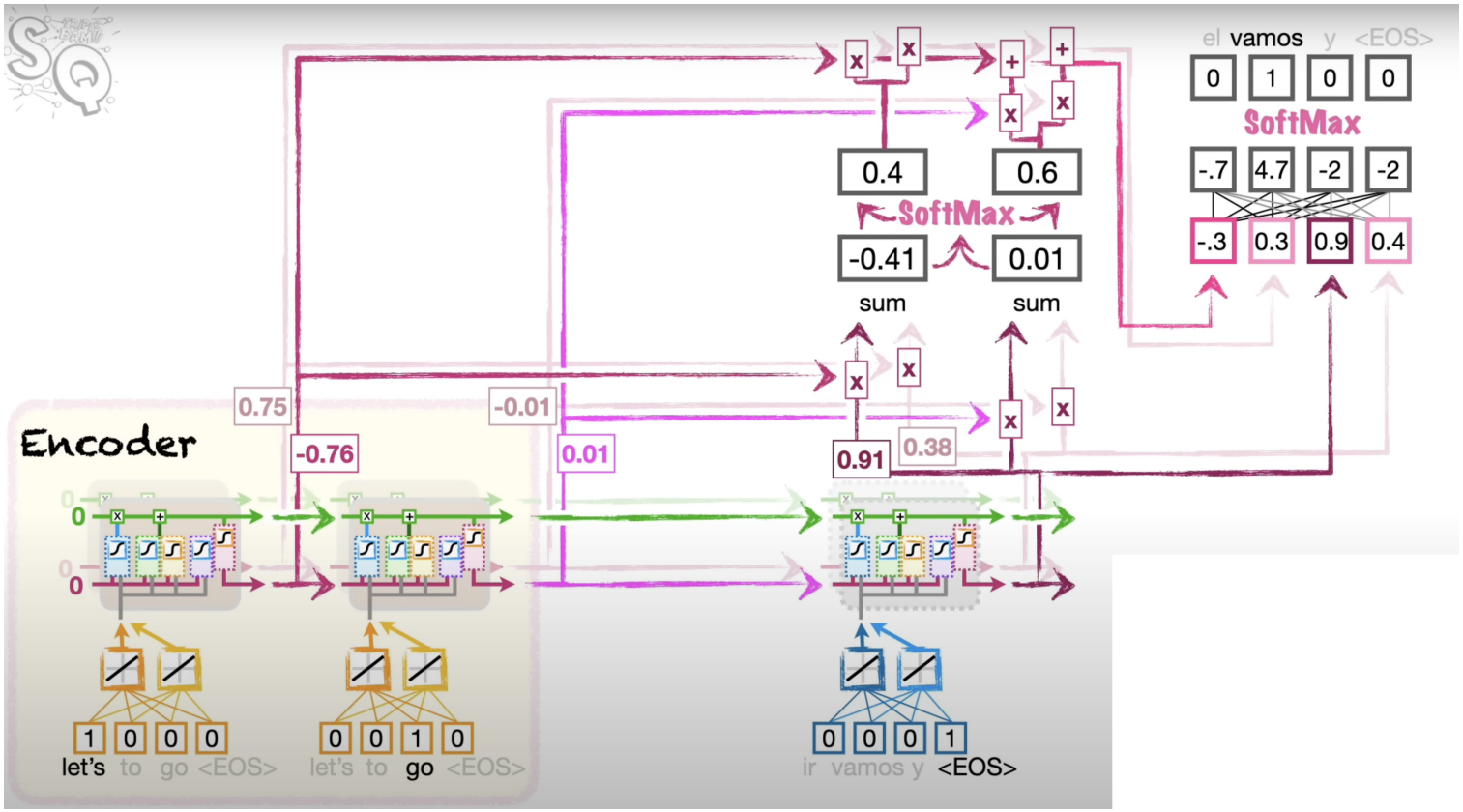


Figure: Calculate the weight

Answer to What is Attention Mechanism?

Actually attention is just weight. Here is the general model of it:

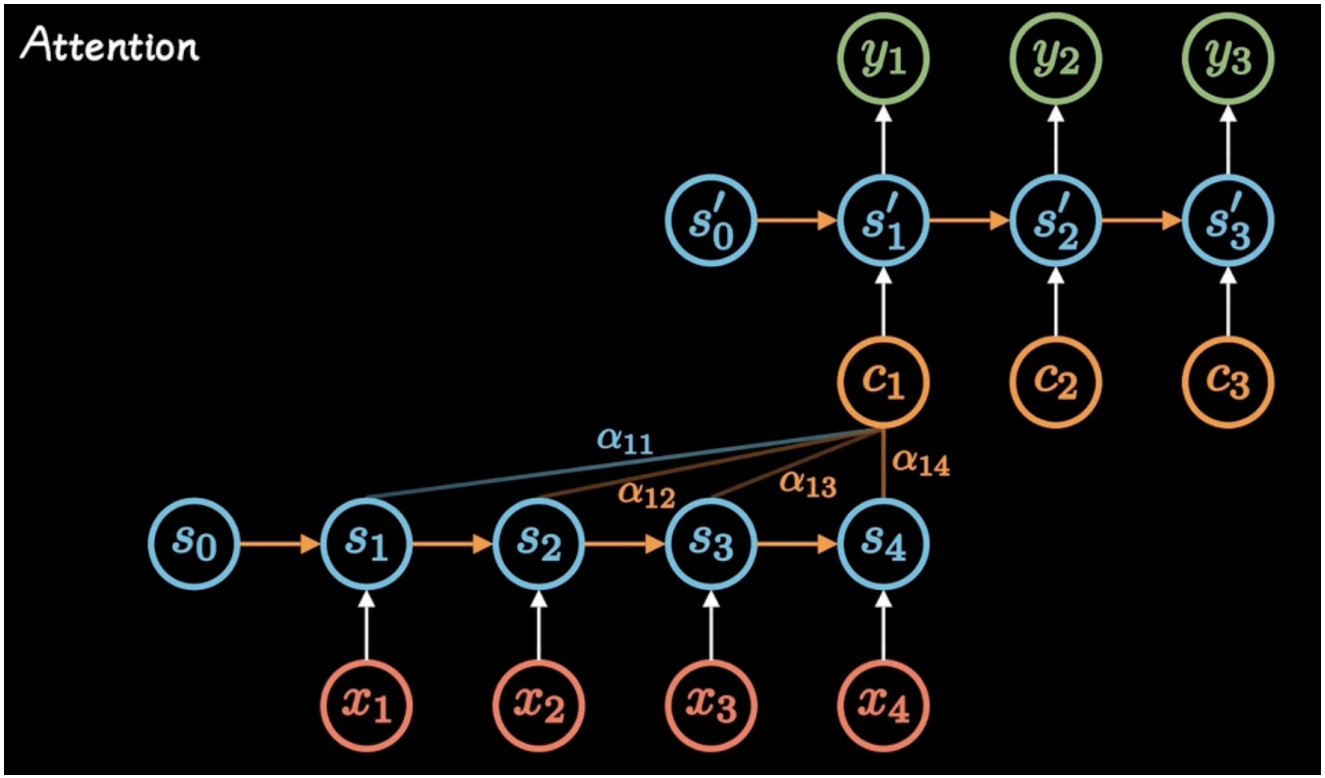


Figure: Attention model

It's not restricted to Encoder-Decoder method. Actually the main idea is to assign different weights to the data in different time stage.

Self Attention(Just FYI)

As you can see, we can assign different weights to each state. Actually we don't need the hidden state! Then it becomes

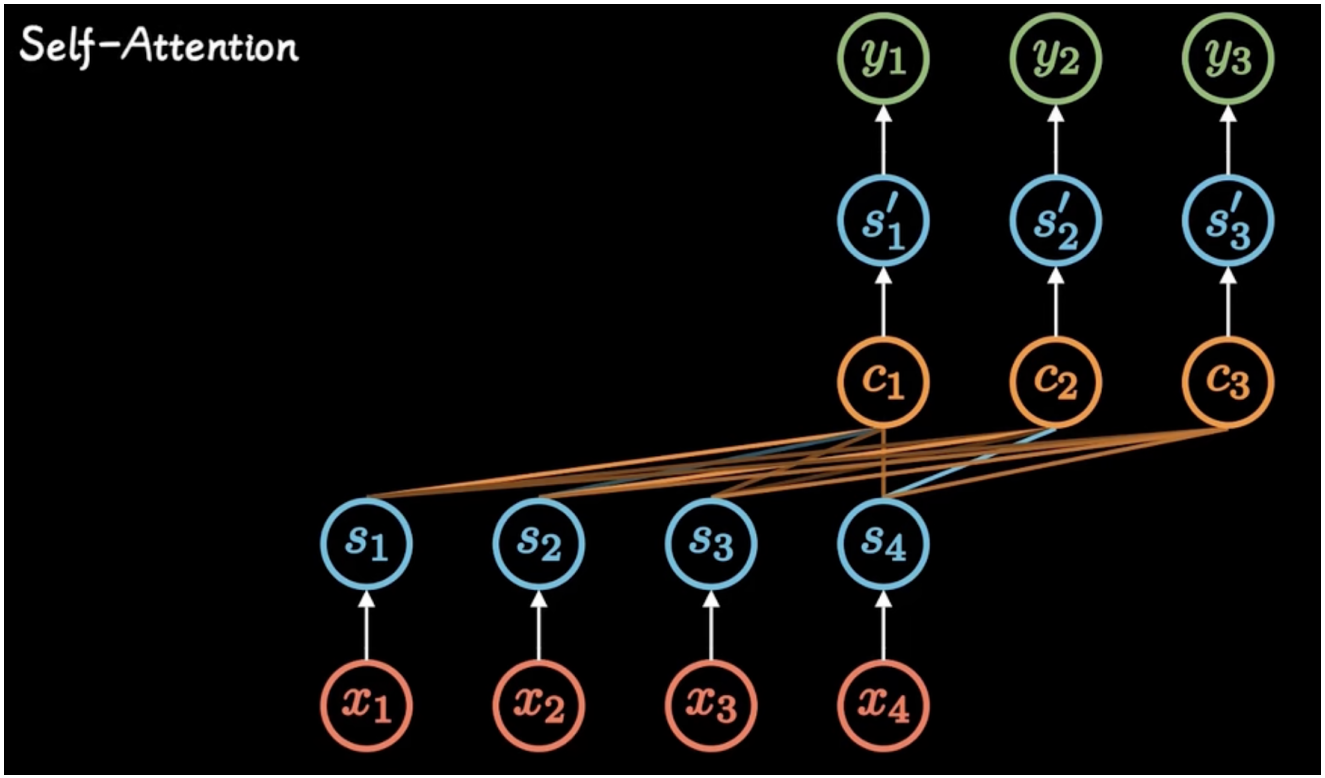


Figure: Self Attention model

Now it's not based on RNN/LSTM at all!

Attention: Advantages and applications

From this example in machine translation, you can see Attention mechanism really improves the model. In general, it can help the NN

- Use less parameters
- predict with faster speed
- Get better performance

And more applications:

- Predict holiday or promotional periods in retail forecasting.
- Learn some previous strange signals in financial time series.

Output

Forget all the specific models!

- Our output from the deep learning model can be discrete(classification) or continuous(regression).
- More precisely, consider the results, we have the expected value of the target(the value with the highest probability). We call it Point estimates.
- Consider the uncertainty, we can also find the distribution of the results. We can call it probabilistic outputs. In deep learning models, it's very important to know the uncertainty!!! e.g. financial risk management.

An example of GMM

$$y_{t+\tau} \sim \mathcal{N}(\mu(t,\tau), \sigma(t,\tau)^2)$$

$$\mu(t,\tau) = W_\mu + h_t^L + b_\mu$$

$$\sigma(t,\tau) = softplus(W_\Sigma h_t^L + b_\Sigma)$$

Loss function

Loss function is very important when we train the DL model using backpropagation. For one step ahead forecasts of point estimates. We have these loss functions.

Classification: Cross Entropy

$$\mathcal{L}_{classification} = -\frac{1}{T}\sum_{t=1}^{T}[y_t \log_2(\hat{y}_t) + (1-y_t)\log_2(1-\hat{y}_t)]$$

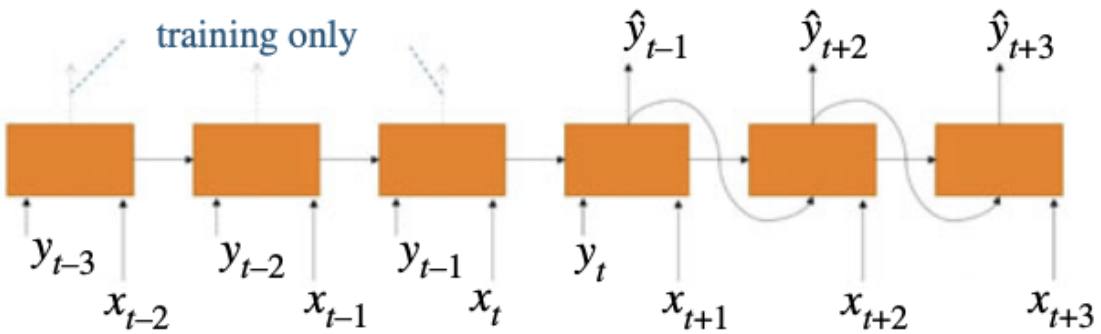
Regression: MSE

$$\mathcal{L}_{regression} = \frac{1}{T}\sum_{t=1}^{T}(y_t - \hat{y}_t)^2$$

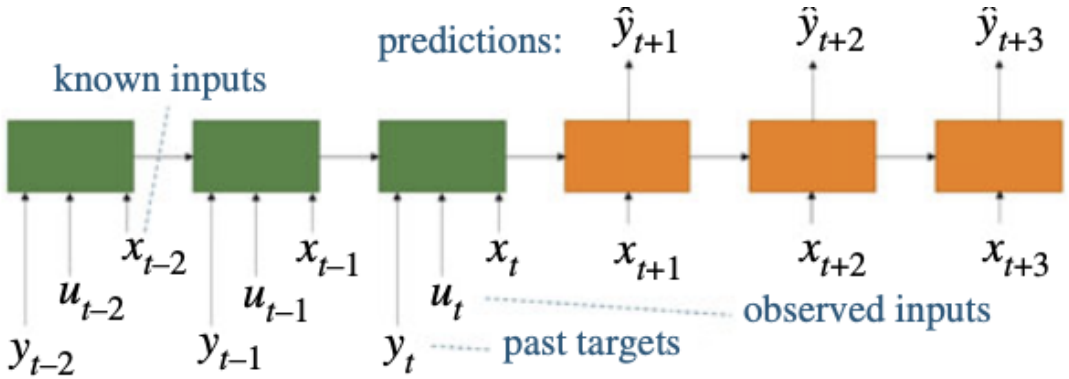
Multi-horizon forecasting models

In practice, we often need to make multiple time periods prediction. We have two different methods:

- Iterative methods: Predict one in the one step ahead$(x, y_{1:t-1} \Rightarrow \hat{y}_t)$, update it to the model then predict two steps$(\hat{y}_{t+1})$...



- Direct methods: Predict all we want by historical data$(x, y_{1:t-1} \Rightarrow \hat{y}_{t+n})$



Incorporating domain knowledge with hybrid models

Key reasons for developing hybrid models:

- The flexibility of machine learning methods make them prone to overfitting.
- machine learning models can be sensitive to how inputs are pre-processed.

In general, hybrid models use deep neural networks in two manners:

- To encode time-varying parameters for non-probabilistic parametric models.
- To produce distribution parameters for probabilistic models

Non-probabilistic hybrid models

Example: ES-RNN. Use the update equations of the Holt-Winters expobebtial smoothing model - combining multiplicative level and seasonality components with deep learning outputs as below.

$$\hat{y}_{i,t+\tau} = \exp(\mathbf{W}_{ES}\mathbf{h}^{L}_{i,t+\tau} + \mathbf{b}_{ES}) \times l_{i,t} \times \gamma_{i,t+\tau},$$

$$l_{i,t} = \frac{\beta_1^{(i)} y_{i,t}}{\gamma_{i,t}} + (1-\beta_1^{(i)}) l_{i,t-1}, \quad \gamma_{i,t} = \frac{\beta_2^{(i)} y_{i,t}}{l_{i,t}} + (1-\beta_2^{(i)}) \gamma_{i,t-\kappa},$$

where $\mathbf{h}^{L}_{i,t+\tau}$ is the final layer of the network for the τth-step-ahead forecast, $l_{i,t}$ is a level component, $\gamma_{i,t}$ is a seasonality component with period κ, and $\beta_1^{(i)}, \beta_2^{(i)}$ are entity specific static coefficients.

Key advantages of the RNN

- Leverage information across different time series
- Allow for exogenous variables to promote predictive power

Probabilistic hybrid models

Example: Deep State Space Models. Use neural networks to encode time-varying parameters for linear state space models as below.

$$y_t = \mathbf{a}(\mathbf{h}^L_{i,t+\tau})^T \mathbf{l}_t + \phi(\mathbf{h}^L_{i,t+\tau})\epsilon_t$$

and

$$\mathbf{l}_t = \mathbf{F}(\mathbf{h}^L_{i,t+\tau})\mathbf{l}_{t-1} + \mathbf{q}(\mathbf{h}^L_{i,t+\tau}) + \Sigma(\mathbf{h}^L_{i,t+\tau})\Sigma_t,$$

where $\mathbf{l}_t$ is the hidden latent state, $\mathbf{a}(.), \mathbf{F}(.), \mathbf{q}(.)$ are linear transformations of $\mathbf{h}^L_{i,t+\tau}$, $\phi(.), \Sigma(.)$ are linear transformations with softmax activations, and $\epsilon_t \sim N(0,1), \Sigma_t \sim N(0, \mathbb{I})$ are standard normal random variables.

Probabilistic hybrid models

State Space Models. SSMs model the temporal structure of the data via a latent state $\mathbf{l}_t \in \mathbb{R}^L$ that can be used to encode time series components such as level, trend, and seasonality patterns. A general SSM is described by the state-transition equation, definiting the stochastic transition dynamics $p(\mathbf{l}_t|\mathbf{l}_{t-1})$ by which the latent state evolves over time, and an observation model specifying the conditional probability $p(z_t|\mathbf{l}_t)$ of observations given the latent state. - state and observation equation

$$\mathbf{l}_t = \mathbf{F}_t\mathbf{l}_{t-1} + \mathbf{g}_t\epsilon_t, \quad z_t = y_t + \sigma_t\epsilon_t, \qquad y_t = \mathbf{a}^T\mathbf{l}_t + b_t, \quad \epsilon_t \sim \mathcal{N}(0,1)$$

Parameter learning

The state space model is fully specified by the parameters $\Theta_t = (\mu_0, \Sigma_0, \mathbf{F}_t, \mathbf{g}_t, \mathbf{a}_t, b_t, \sigma_t)$, where $\mathbf{l}_0 \sim N(\mu_0, \Sigma_0)$. One generic way of estimating them is by maximizing the marginal likelihood.

Note that in the classical setting, if there is more than one time series, a separate set of parameters $\Theta^{(i)}$ is learned for each time series $z_{1:T}^{(i)}$ independently. This has the disadvantage that no information is shared across different time series.

Probabilistic hybrid models

Deep State Space Models. Instead of learning the state parameters $\Theta^{(i)}$ for each time series independently, our forecasting model instead learns a globally shared mapping from the covariate vectors $\mathbf{x}_{1:T}^{(i)}$ associated with each target time series $z_{1:T}^{(i)}$, to the (time-varing) parameters $\Theta_t^{(i)}$ of a linear state space model for the i-th time series as follows.

$$\Theta_t^{(i)} = \Psi(\mathbf{x}_{1:T}^{(i)}, \Phi).$$

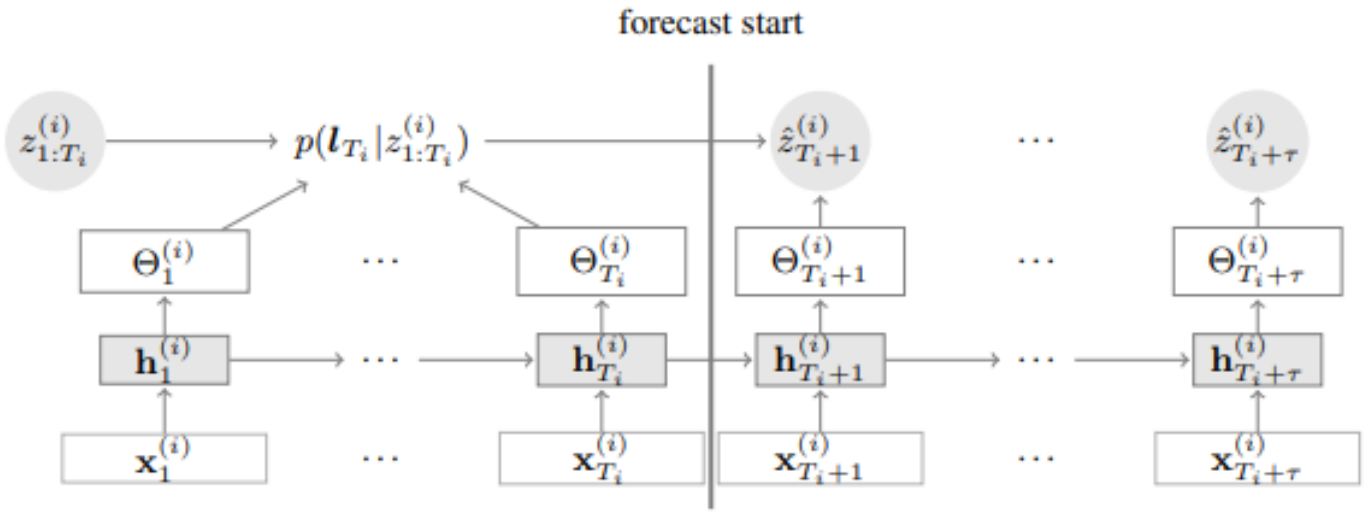
Then, given the features $\mathbf{x}_{1:T}^{(i)}$ and the parameters Φ, under our model, the data $z_{1:T_i}^{(i)}$ is distributed according to

$$p(z_{1:T_i}^{(i)} | \mathbf{x}_{1:T}^{(i)}, \Phi) = p_{SS}(z_{1:T_i}^{(i)} | \Theta_{1:T_i}^{(i)}),$$

where p_{SS} denotes the marginal likelihood under a linear state space model, which can be used for parameter learning.

Deep State Space Models(Continue)

Parameter learning. Then we parameterize the mapping Ψ from covariates to state space model parameters using a deep recurrent neural network (RNN) and train the model based on the likelihood function. Figure below shows a sketch of the overall model structure.



Facilitating decision support using deep neural networks

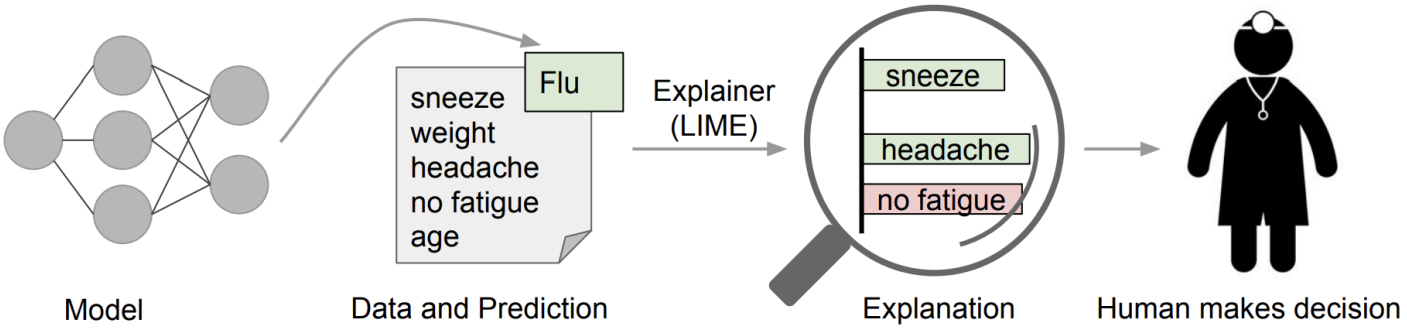


Figure: Understanding and interpreting the predictions made by deep neural networks is as important as accurate predictions. In real-world scenarios, like in the medical field, professionals use these predictions to make vital decisions. This is especially important in time series data scenario.

Interpretability with Time-Series Data

This deals with understanding how and why a model makes a specific prediction. Given the complex nature of neural networks, research has focused on methods to interpret these models. Two categories exist:

- Techniques for post hoc interpretability
- Inherent interpretability with attention weights

Techniques for Post Hoc Interpretability

- These methods don't change the original model but provide interpretations after the fact.
- They may overlook sequential dependencies in time-series data.
- Surrogate Models (LIME, SHAP):
  - They provide approximated explanations by creating simpler interpretable models between inputs and outputs of the neural network to provide explanations.
  - They identify and rank features based on their importance.
- Gradient-based Methods (Saliency Maps, Influence Functions):
  - Analyze network gradients to determine input features that most impact the loss functions.

LIME Overview and Fidelity-Interpretability Trade-off

- Goal: Provide interpretable model representations that are locally faithful to the classifier.The primary goal is to bridge the gap between machine understanding and human interpretability.
- Explanation produced by LIME:

$$\xi(x) = argmin_{g \in G} L(f, g, \pi_x) + \Omega(g)$$

Where:

- g is a model in class G of interpretable models.
- $\Omega(g)$ measures the explanation's complexity.
- $L(f, g, \pi_x)$ measures how unfaithful g is in approximating f in a locality defined by π_x.

Sampling for Local Exploration

Objective: Learn local behavior of f by drawing samples around the instance being explained without making any assumption about f.

- Approximate $L(f, g, \pi_x)$ using samples weighted by π_x.
- Sample instances around x_0 by drawing non-zero elements of x_0 uniformly at random.
- Given a perturbed sample z_0, recover the sample in the original representation z and obtain $f(z)$ for the explanation model.

LIME Intuition

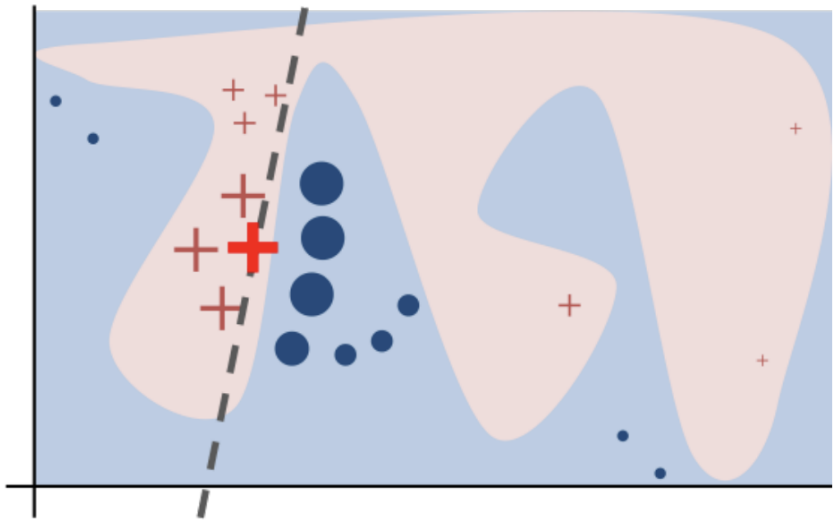


Figure: The black-box model's complex decision function f (unknown to LIME) is represented by the blue/pink background, which cannot be approximated well by a linear model. The bold red cross is the instance being explained. LIME samples instances, gets predictions using f, and weighs them by the proximity to the instance being explained (represented here by size). The dashed line is the learned explanation that is locally (but not globally) faithful.

Influence Functions

- Consider training points $z_1, \ldots, z_n$ where $z_i = (x_i, y_i) \in X \times Y$.
- Goal: Understand the effect of training points on model's predictions.
- Empirical risk minimizer:

$$\hat{\theta} = \arg\min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} L(z_i, \theta)$$

.

Key Ideas and Formulas

- Influence functions provide an efficient approximation to understand the effect of a point on the model.
- Influence of upweighting z:

$$I_{\text{up,params}}(z) = -H_{\hat{\theta}}^{-1} \nabla_\theta L(z, \hat{\theta})$$

- Effect on a test point z_{test}:

$$I_{\text{up,loss}}(z, z_{\text{test}}) = -\nabla_\theta L(z_{\text{test}}, \hat{\theta})^T H_{\hat{\theta}}^{-1} \nabla_\theta L(z, \hat{\theta})$$

- Counterfactual approach: Understand prediction changes if training input is modified.

Comparison and Computation

- Compare influence functions with nearest neighbors in Euclidean space.
- Influence functions capture training effects more accurately.
- Techniques for efficient computation: Conjugate gradients (CG) and Stochastic estimation.

Inherent Interpretability with Attention Weights

- These models are designed with interpretable components from the start, often using attention layers.
- Attention mechanisms can highlight which time points in the data are crucial for predictions by assigning weights to input features. item As attention weights are produced as outputs from a softmax layer, the weights are constrained to sum to 1. Thus, the outputs can be interpreted as a weighted average over temporal features.
- Analyzing these weights can reveal the importance of different time steps in the data and even persistent patterns like seasonality.

Counterfactual predictions and causal inference over time

This section discusses how deep learning can produce predictions outside of their observational datasets, helping in scenario analysis. The challenge is handling time-dependent confounding effects. Several recent methods adjust for these confounders, drawing from statistical techniques and designing new loss functions.

Challenges and Methods

Challenges:

- Time series datasets have time-dependent confounding effects, where actions affecting the target also depend on observations of the target.
- This can lead to biased results if not addressed.

Methods:

- Techniques like extensions of the inverse-probability-of-treatment-weighting (IPTW) approach and G-computation framework use deep learning to model distributions and learn unbiased predictions.
- Other methods, like domain adversarial training, create balanced representations of data histories to mitigate biases.

Conclusion

- Explores advancements of deep neural networks in forecasting due to increased data and computational power.
- Highlights primary time-series forecasting architectures.
- Emphasizes adaptability, hybrid deep learning integration, and interpretability's significance.
- Points out enduring challenges: need for regular time intervals and addressing hierarchical structures.

References

Lim, B, Zohren S (2021)

Time-series forecasting with deep learning: a survey

Phil. Trans. R. Soc. A 379: 20200209.

Thanks for your attention!