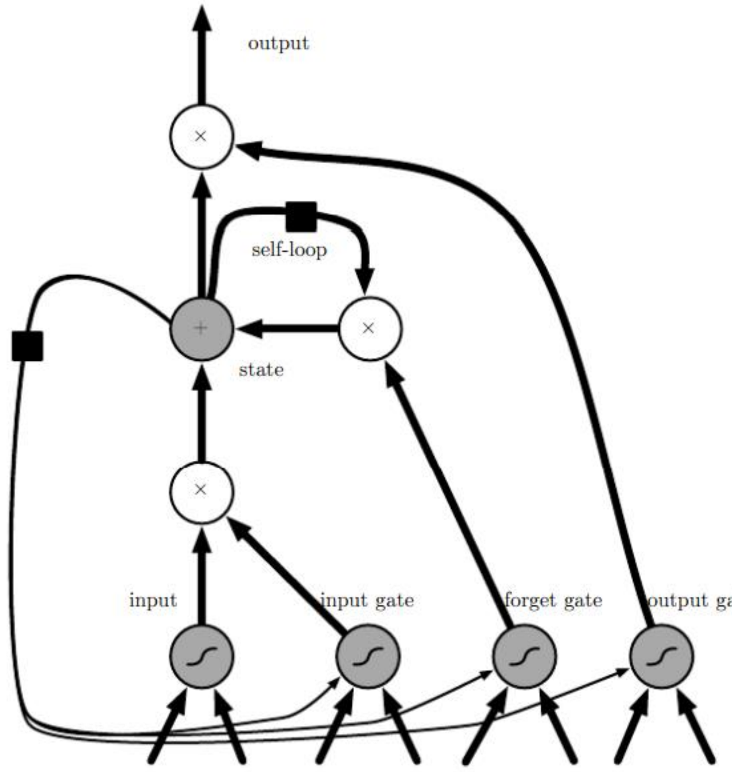
ECE 4252/8803: Fundamentals of Machine Learning (FunML)
Fall 2024

Lecture 20: Sequence Modeling



Overview

In this Lecture..

- Introduction and Examples of Sequence Modeling
- Limitations of MLPs while Modeling Sequences
- The Recurrent Neural Network: architecture, loss function, and Backpropagation
- The Vanishing Gradient Problem
- LSTM and GRU Design to Overcome the Vanishing Gradient Problem
- Introduction to the Temporal Convolutional Network

Sequence Modeling

Sequences

- Feature data used with many machine learning models including traditional MLPs does not account for ordering of features present

Age	Sex	Occupation	Income
24	F	Engineer	$70K
35	M	Doctor	$120K

- Many real-life situations involve data with time-dependence e.g., temperature, stock exchange, language etc.

- Inter-relationships between individual data points are not modeled by classical ML models and MLPs during training and testing phases

Sequence Modeling

Sequences

- Some examples of sequence modeling problems include:
  - Sequence prediction and classification

    This kind of problem involves predicting a single output (e.g., label, real number etc.) given a sequence. The input sequence may consist either of discrete or real values

  - Sequence generation

    Involves generating a new output sequence that has the same general characteristics as other sequences in the corpus

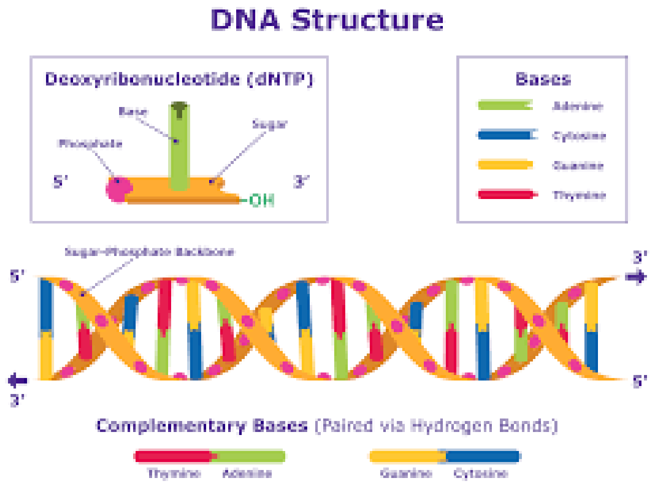
  - Sequence to sequence prediction

    Involves predicting an output sequence given an input sequence. Unlike the previous class of sequence modeling problems, this involves an explicit mapping from an input to output domain

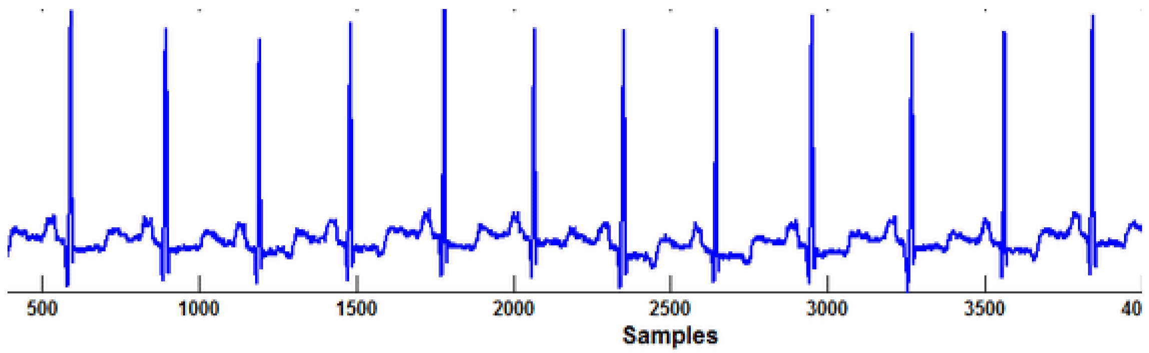
[1] https://machinelearningmastery.com/sequence-prediction/

Sequence Modeling

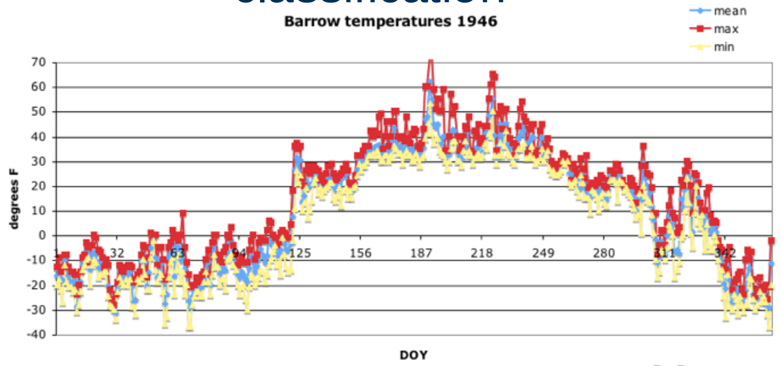
Sequences and Classification



DNA sequence classification[1]



Anomaly detection in ECG signals



Temperature forecasting[2]

“I have been using this product for 3 yrs and absolutely love this!”

Sentiment analysis

[1] https://blog.praxilabs.com/2021/02/08/dna-sequencing-definition-importance-methods-facts-and-more/
[2] https://serc.carleton.edu/eet/gsod/part_2.html

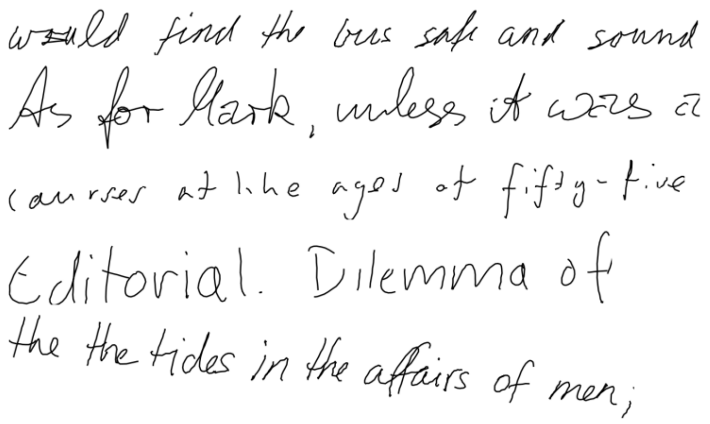
Sequence Modeling

Sequence Generation

“To be, or not to be: that is the question:
Whether ‘tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
And by opposing end them? To die: to sleep;”

—Hamlet, Shakespeare

Text generation



would find the bus safe and sound
As for Mark, unless it was a
caursers at lihe ages of fifty-five
Editorial. Dilemma of
the tides in the affairs of men;

Handwriting generation[1]

The man in grey swings a bat while the man in black looks on.

A big bus sitting next to a person.

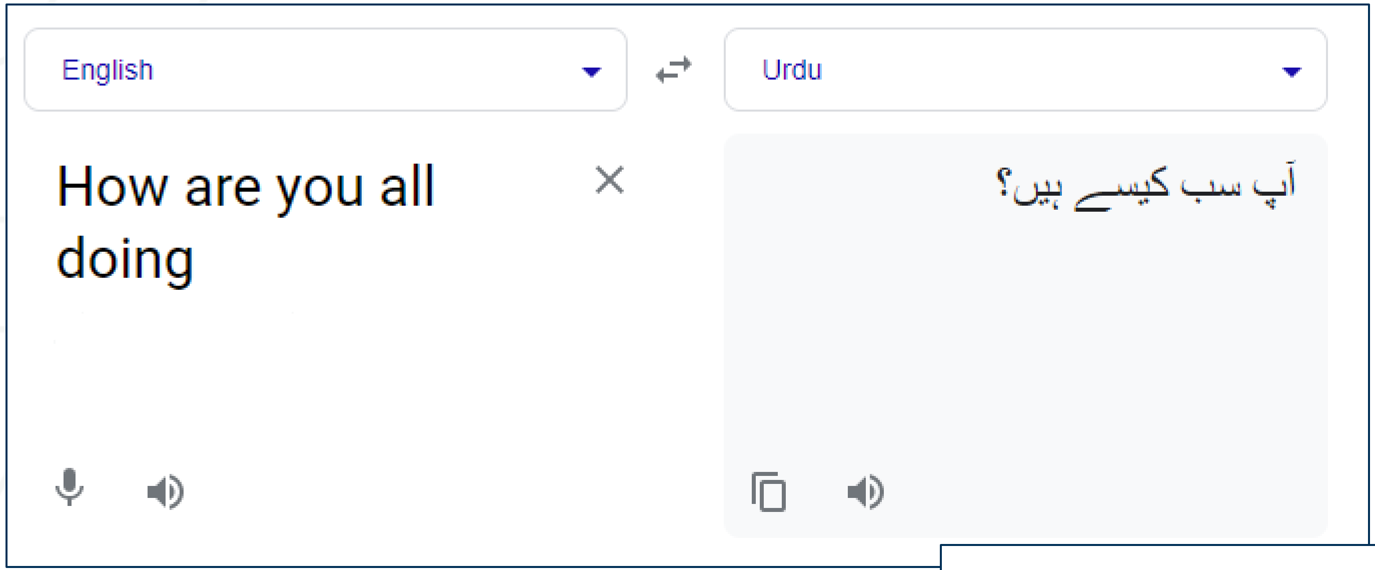
Image captioning[2]

Music generation

[1] Training samples from the IAM online handwriting database
[2] https://www.oreilly.com/content/caption-this-with-tensorflow/

Sequence Modeling

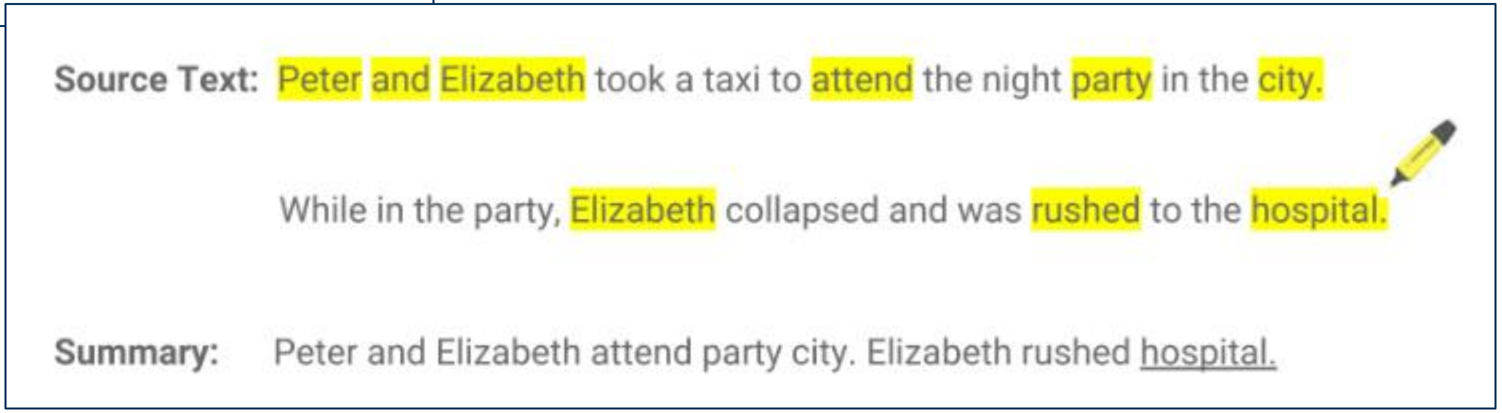
Sequence to Sequence Generation


English
Urdu

How are you all doing

آپ سب کیسے ہیں؟

Machine translation


Source Text: Peter and Elizabeth took a taxi to attend the night party in the city. While in the party, Elizabeth collapsed and was rushed to the hospital.

Summary: Peter and Elizabeth attend party city. Elizabeth rushed hospital.

Text summarization[1]

[1] https://blog.floydhub.com/gentle-introduction-to-text-summarization-in-machine-learning/

MLP Sequence Modeling
Limitations

- MLPs input and output fixed length vectors decided at training time. They cannot scale to variable length sequences afterwards

- For long input/output sequences, the number of weights in an MLP would be extremely large → *overfitting in presence of limited data*

- Words can change positions in a sentence without change in meaning. The MLP would have to be trained to explicitly recognize all such input combinations

Sequence Modeling

Overcoming Limitations: State History

- An ideal sequence model should also have a state summarizing the history of all past inputs to help in the prediction of present and future outputs.

- Combining the idea of parameter sharing with state history gives rise to the following computational model

$$\underbrace{h^{(t)}}_{\text{State vector at present time}} = f(\underbrace{h^{(t-1)}}_{\text{State vector at past time}}, \underbrace{x^{(t)}}_{\text{Input at present time}})$$

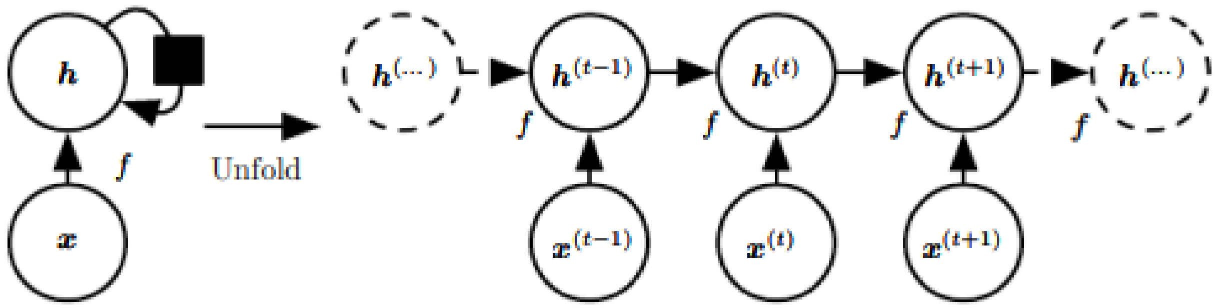


Image source: Chapter 10, *Deep Learning,* Ian Goodfellow

Recurrent Neural Networks (RNN)

Overcoming Limitations: State History

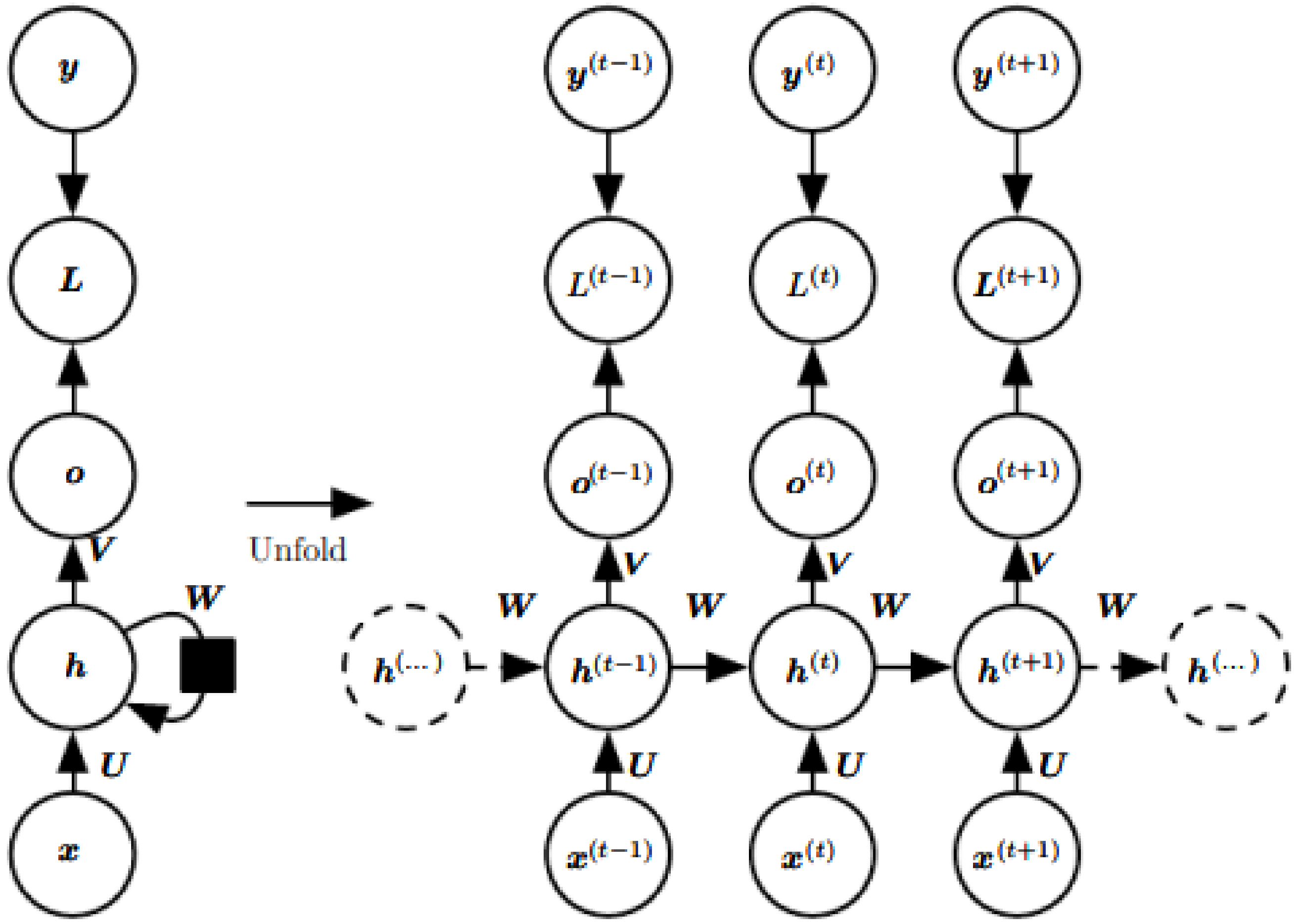
- Using the basic ideas discussed in the slides before, one can design a variety of recurrent neural network (RNN) architectures suited to one or more of the various sequence modeling problems discussed earlier

- A popular choice for sequence-to-sequence problems involves training an RNN to predict an output of the same length as the corresponding input sequence

- The input sequences themselves can be of varying lengths

Recurrent Neural Networks

RNN Architecture

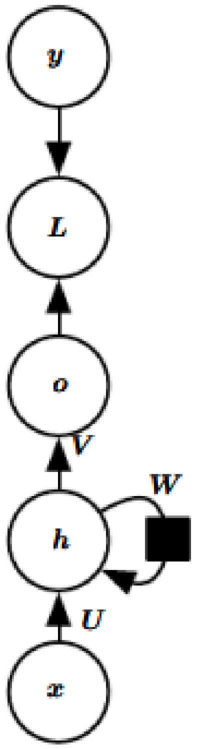


$\boldsymbol{x}$: Input at time t
$\boldsymbol{h}$: Hidden state at time t
$\boldsymbol{o}$: Output at time t
$\boldsymbol{L}$: Loss value at time t
$\boldsymbol{y}$: Target corresponding to input at time t
$\boldsymbol{U}$: Mapping from input to hidden state
$\boldsymbol{W}$: Mapping from present hidden state to future hidden state
$\boldsymbol{V}$: Mapping from present hidden state to present output

Image source: Chapter 10, *Deep Learning,* Ian Goodfellow

Recurrent Neural Networks

RNN Update Equations



$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}$ (Computing the neuron activations)

$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{a}^{(t)})$ (Applying non-linear activation)

$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}$ (Computing output vector as a function of present hidden state vector)

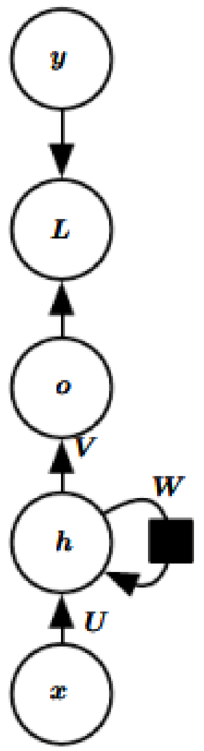
$\boldsymbol{y}^{(t)} = \mathrm{softmax}(\boldsymbol{o}^{(t)})$ (Applying softmax to obtain pseudo-probabilities)

More on Design Options:

- W, U, and V matrices characterize linear layers as in MLPs. b and c characterize bias vectors
- It is possible to choose different non-linear activation functions to the one described here
- The transition from hidden state to output can happen in multiple fully connected layers as opposed to only one as shown here (at the cost of increased network complexity)
- One may omit softmax if dealing with real-valued outputs

Recurrent Neural Networks

RNN Update Equations



$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}$ (Computing the neuron activations)

$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{a}^{(t)})$ (Applying non-linear activation)

$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}$ (Computing output vector as a function of present hidden state vector)

$\boldsymbol{y}^{(t)} = \mathrm{softmax}(\boldsymbol{o}^{(t)})$ (Applying softmax to obtain pseudo-probabilities)

Things to Keep in mind:

- Decide on dimensions of input, hidden state, and output vectors prior to training.
- Different options available to encode input vectors: one-hot encoding versus word embeddings
- Hidden state vector should be large enough to store condensed sequence information
- Output vector dimensions decided by the number of words it is possible to output

Recurrent Neural Networks
Loss Evaluation and Backpropagation

- The total loss is computed by summing the individual losses of all time steps

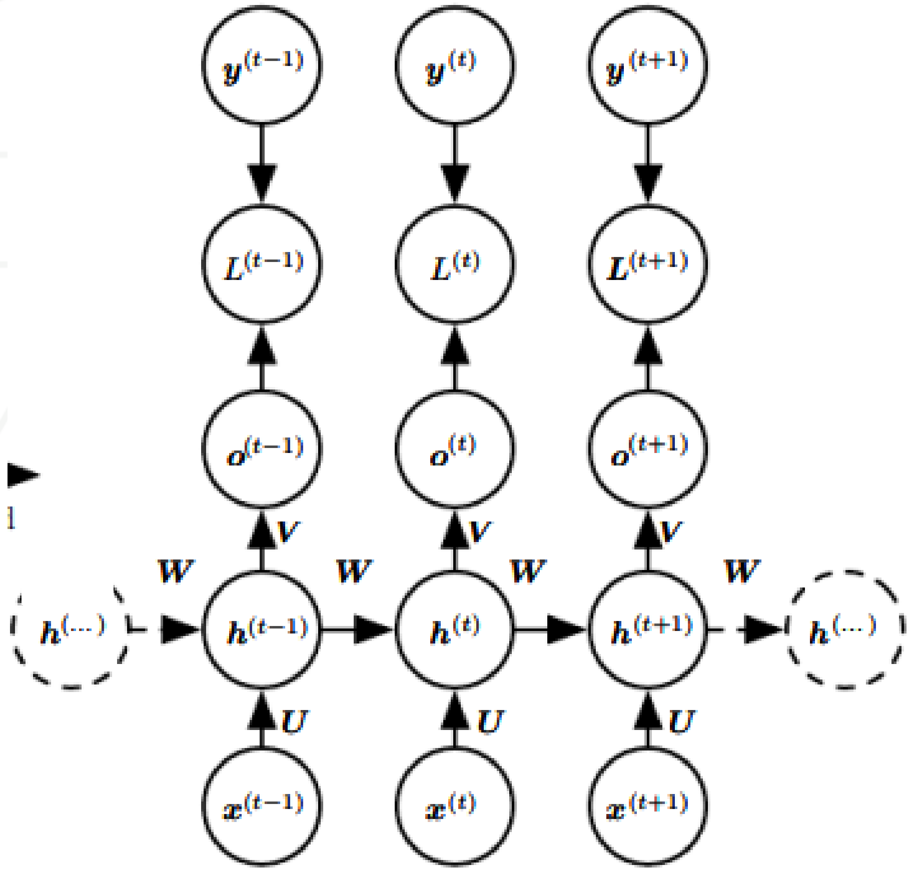
$$\text{loss} = \sum_t L^{(t)}$$

- Given the unique time-dependent nature of RNNs, backpropagation in such networks has certain peculiarities associated with it, and is called *backpropagation through time (BPTT)*

- Loss on each time step is contributing to the hidden state vector at that timestep and previous time steps. Work backwards through the last time step to obtain the full gradients

Recurrent Neural Networks

Loss Evaluation and Backpropagation



$$L = \sum_t L^{(t)}$$

$$\frac{\partial L^{(t)}}{\partial \boldsymbol{W}} = \frac{\partial L^{(t)}}{\partial \boldsymbol{o}^{(t)}} \times \frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \times \sum_{\tau=1}^{t} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(\tau)}} \times \frac{\partial \boldsymbol{h}^{(\tau)}}{\partial \boldsymbol{W}}$$

Repeat for $L^{(t)}$ for all times t and examples i and sum to obtain $\frac{\partial L}{\partial \boldsymbol{W}}$:

$$\frac{\partial L}{\partial \boldsymbol{W}} = \sum_i \sum_t \frac{\partial L_i^{(t)}}{\partial \boldsymbol{W}}$$

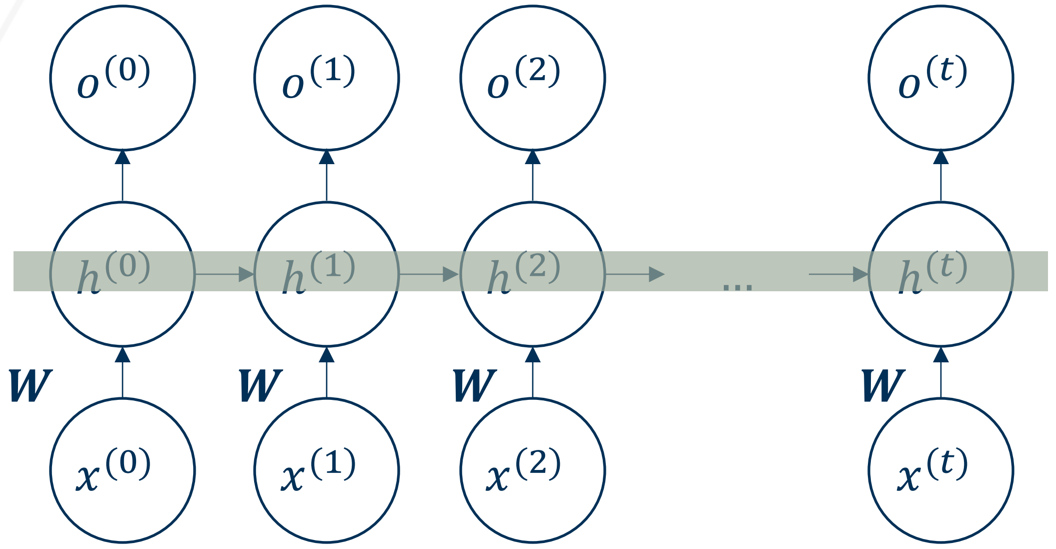
Recurrent Neural Networks

Vanishing Gradient Problem

$$\frac{\partial L^{(t)}}{\partial \boldsymbol{W}} = \frac{\partial L^{(t)}}{\partial \boldsymbol{o}^{(t)}} \times \frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \times \sum_{\tau=1}^{t} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(\tau)}} \times \frac{\partial \boldsymbol{h}^{(\tau)}}{\partial \boldsymbol{W}}$$

For a large value of t, this term will involve a long chain of products e.g.,

$$\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(0)}} = \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} \times \cdots \times \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial \boldsymbol{h}^{(1)}}{\partial \boldsymbol{h}^{(0)}}$$



As the sequence becomes longer, this product gets bigger and longer as well!

Recurrent Neural Networks

Vanishing Gradient Problem

$$\frac{\partial L^{(t)}}{\partial \boldsymbol{W}} = \frac{\partial L^{(t)}}{\partial \boldsymbol{o}^{(t)}} \times \frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \times \sum_{\tau=1}^{t} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(\tau)}} \times \frac{\partial \boldsymbol{h}^{(\tau)}}{\partial \boldsymbol{W}}$$

For a large value of t, this term will involve a long chain of products e.g.,

$$\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(0)}} = \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} \times \cdots \times \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial \boldsymbol{h}^{(1)}}{\partial \boldsymbol{h}^{(0)}}$$

From the RNN update equation,

$$\boldsymbol{h}^{(t)} = \boldsymbol{\sigma}\left(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t-1)}\right).$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{\sigma}\left(\boldsymbol{a}^{(t)}\right).$$

From this, “it is easy” to see[1] that

$$\begin{aligned}\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} &= \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{a}^{(t)}} \times \frac{\partial \boldsymbol{a}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} \\ &= \begin{bmatrix} h_{\boldsymbol{1}}^{(t)}(1-h_{\boldsymbol{1}}^{(t)}) & \cdots & -h_{\boldsymbol{1}}^{(t)}h_{\boldsymbol{N}}^{(t)} \\ \vdots & \ddots & \vdots \\ -h_{\boldsymbol{N}}^{(t)}h_{\boldsymbol{1}}^{(t)} & \cdots & h_{\boldsymbol{N}}^{(t)}(1-h_{\boldsymbol{N}}^{(t)}) \end{bmatrix} \times \boldsymbol{W}^{T} \\ &= \boldsymbol{J}^{(t)} \times \boldsymbol{W}^{\boldsymbol{T}}\end{aligned}$$

As the sequence becomes longer, this product gets bigger and longer as well!

$$\begin{aligned}\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(0)}} &= \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} \times \cdots \times \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial \boldsymbol{h}^{(1)}}{\partial \boldsymbol{h}^{(0)}} \\ &= \boldsymbol{J}^{(\boldsymbol{t})} \times \boldsymbol{W}^{\boldsymbol{T}} \times \boldsymbol{J}^{(\boldsymbol{t-1})} \times \boldsymbol{W}^{\boldsymbol{T}} \times \cdots \times \\ \boldsymbol{J}^{(\boldsymbol{0})} \times \boldsymbol{W}^{\boldsymbol{T}}\end{aligned}$$

[1] eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/

Recurrent Neural Networks

Vanishing Gradient Problem

$$\frac{\partial L^{(t)}}{\partial \boldsymbol{W}} = \frac{\partial L^{(t)}}{\partial \boldsymbol{o}^{(t)}} \times \frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \times \sum_{\tau=1}^{t} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(\tau)}} \times \frac{\partial \boldsymbol{h}^{(\tau)}}{\partial \boldsymbol{W}}$$

Using formula from above:

$$\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(0)}} = \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} \times \cdots \times \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial \boldsymbol{h}^{(1)}}{\partial \boldsymbol{h}^{(0)}}$$
$$= \boldsymbol{J}^{(t)} \times \boldsymbol{W}^{\boldsymbol{T}} \times \boldsymbol{J}^{(\boldsymbol{t}-\boldsymbol{1})} \times \boldsymbol{W}^{\boldsymbol{T}} \times \cdots \times \boldsymbol{J}^{(\boldsymbol{0})} \times \boldsymbol{W}^{\boldsymbol{T}}$$

- Network weights are usually initialized from a standard normal distribution
- Gradients from the sigmoid will (most of the time) be zero
- *Result: a* long product chain means gradients contributed to from earlier time steps can quickly become 0 or close to 0.
- *RNN only captures short-term dependencies!*

Recurrent Neural Networks

Mitigating the Vanishing Gradient Problem

1. Use activation functions with well-behaved derivatives:
   - Sigmoid and Tanh activations have high gradient values for only a small subset of their total input domain.
   - For many inputs, one can expect the gradients to be zero or very close to zero
   - Use ReLU for better behaved gradients

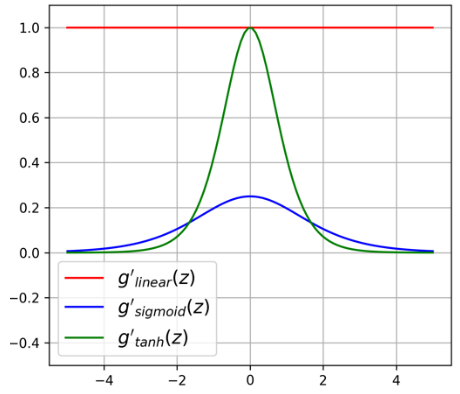
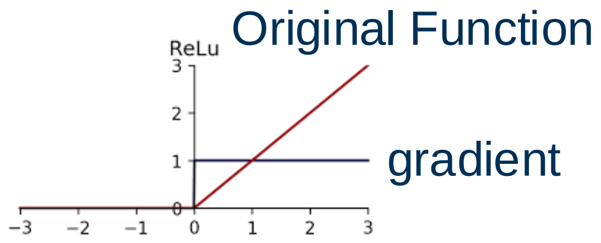


Image Source: [1]



2. Better weight initialization strategies
   - Initializing weight matrices from the standard normal distribution results in small parameter values
   - Initialize instead as identity matrices

$$W = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

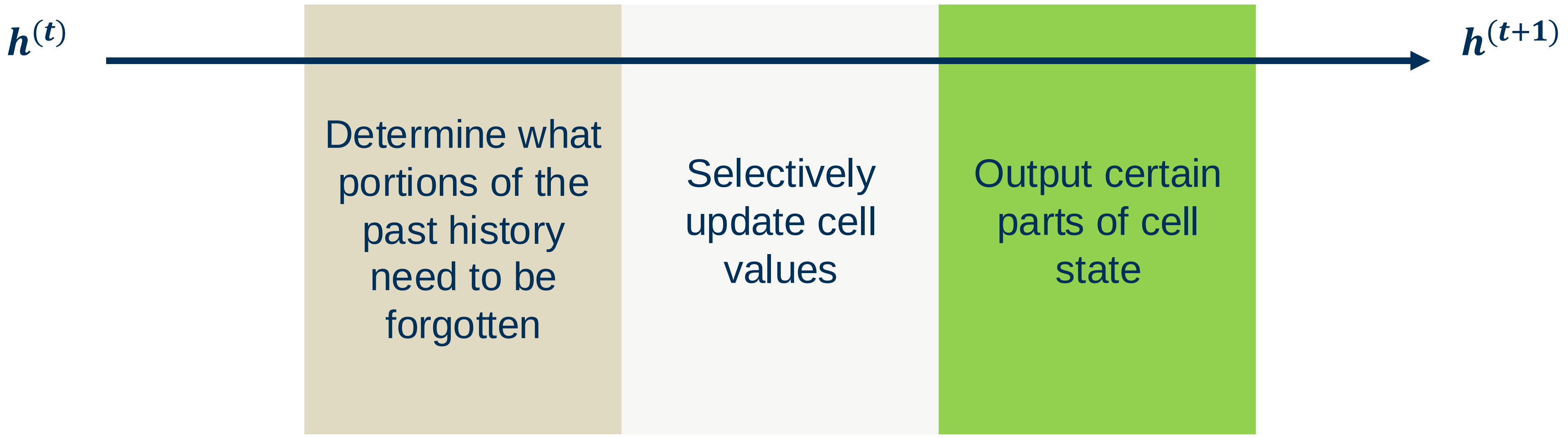
[1] https://dustinstansbury.github.io/theclevermachine/derivation-common-neural-network-activation-functions

Recurrent Neural Networks
Designing better Recurrent Units

- Recurrent unit with gates and self connections to allow for better flow of information
- Gates help determine what portions, if any, of the past history need to be retained vs updated
- Self connections improve the flow of gradients

Long-Short Term Memory (LSTMs) Networks

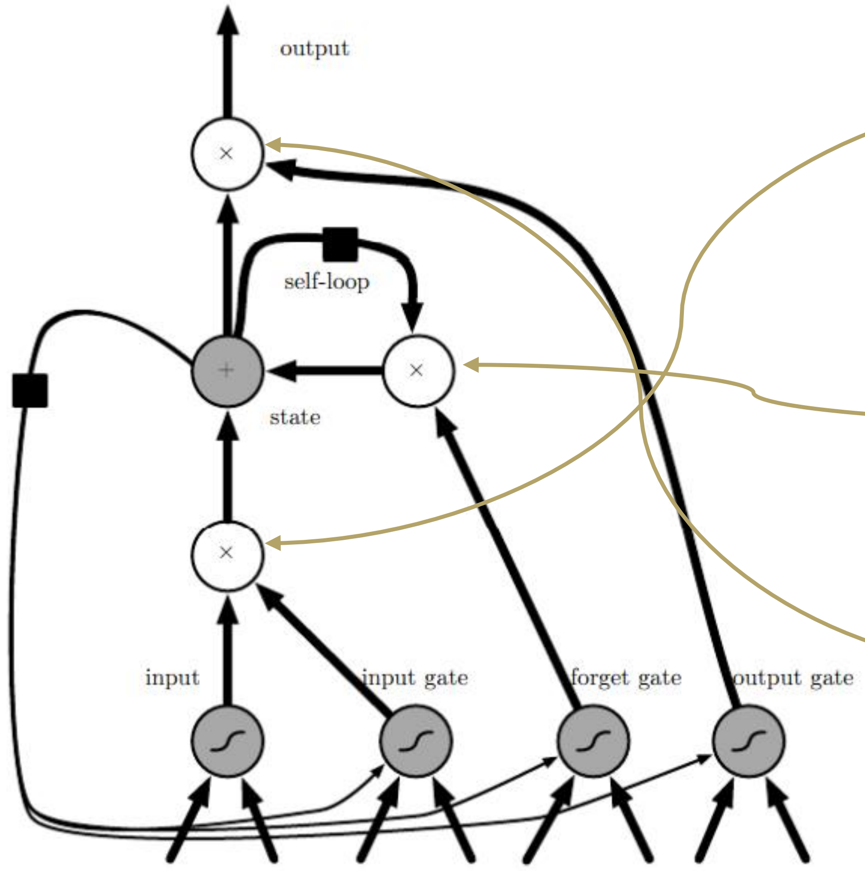
Motivation



$\boldsymbol{h}^{(t)}$
$\boldsymbol{h}^{(t+1)}$
Determine what portions of the past history need to be forgotten
Selectively update cell values
Output certain parts of cell state

Long-Short Term Memory (LSTMs) Networks

LSTM Cell



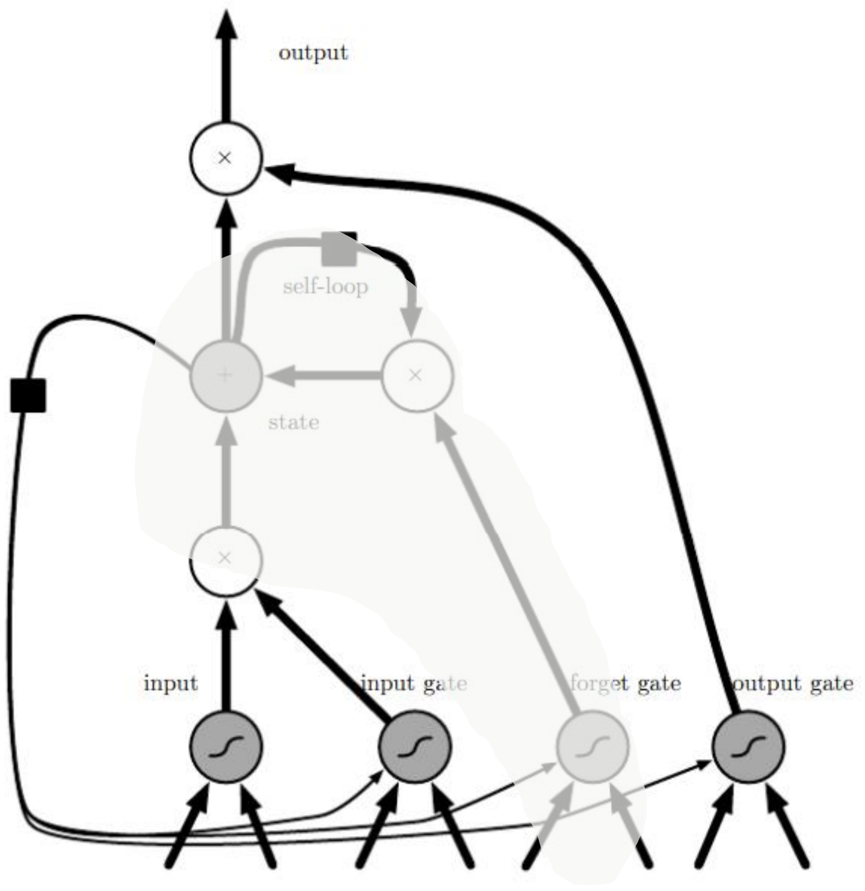
An LSTM cell for one time step [1]

- **Input Gate:** Controls how much of the present input is allowed to factor into the computation for present state
- **Forget Gate:** Controls how much of the old state is forgotten versus retained in the computation of the present state
- **Output Gate:** Controls how much of the present output is influenced by the present state versus that by the past state

[1] Chapter 10, *Deep Learning*, Yoshua Bengio and Ian Goodfellow

Long-Short Term Memory (LSTMs) Networks

LSTM Cell



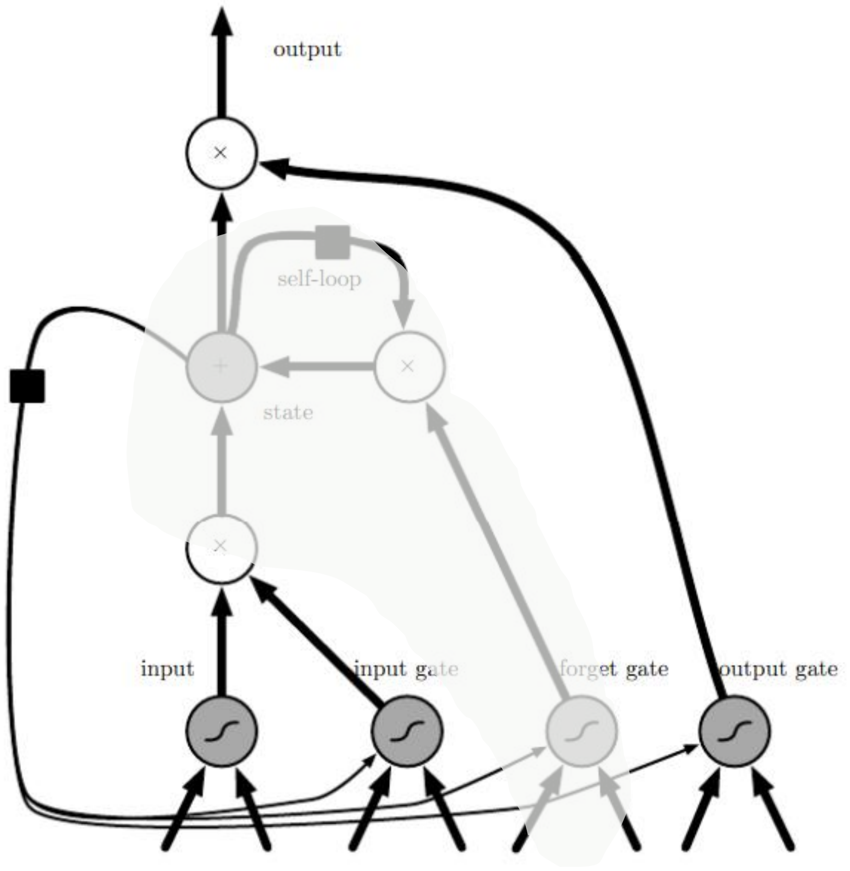
An LSTM cell for one time step [1]

- State variable at current timestep $s^{(t)}$ a function of state at previous time step $s^{(t-1)}$ via an **internal self loop**
  - For RNNs, state variable from past time can only contribute via the recurrent cell update equation
  - Helps improve gradient flow since gradient through addition can compensate for poor gradients through recurrent computations

[1] Chapter 10, *Deep Learning*, Yoshua Bengio and Ian Goodfellow

Long-Short Term Memory (LSTMs) Networks

LSTM Cell



An LSTM cell for one time step [1]

- State variable at current timestep $s^{(t)}$ a function of state at previous time step $s^{(t-1)}$ via an internal self loop
  - For RNNs, state variable from past time can only contribute via the recurrent cell update equation
  - Helps improve gradient flow since gradient through addition can compensate for poor gradients through recurrent computations
- Element wise multiplication with forget state vector to decide what parts of the old state to keep

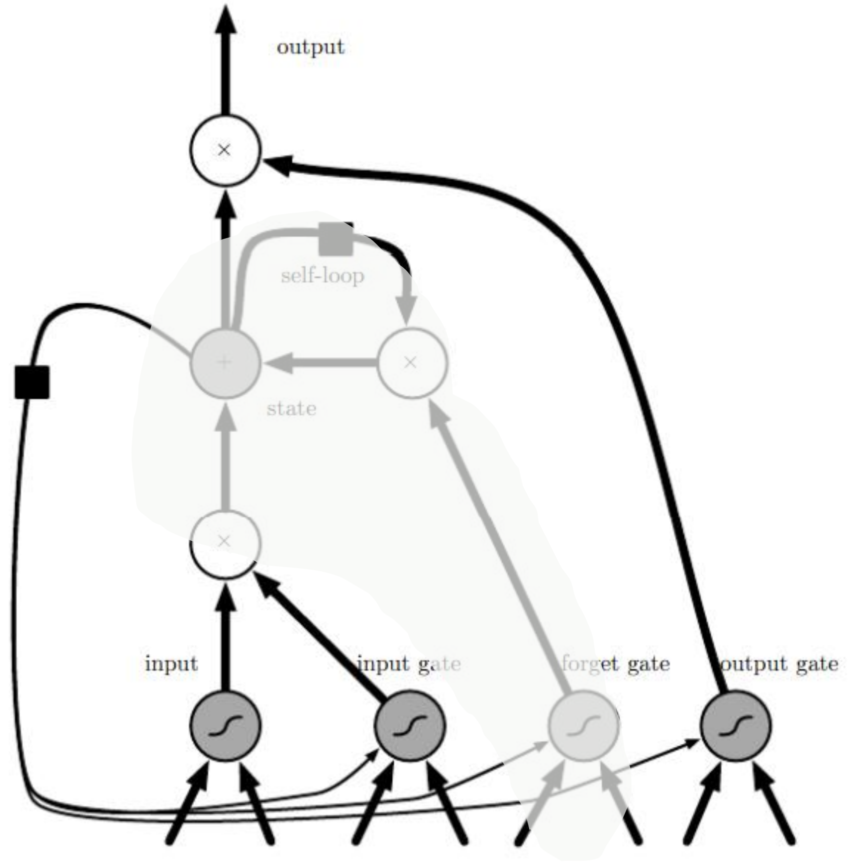
$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \circ \begin{pmatrix} m \\ n \\ o \\ p \end{pmatrix} = \begin{pmatrix} am \\ bn \\ co \\ dp \end{pmatrix}$$

- Parameters in the forget vector learned via gradient descent

[1] Chapter 10, *Deep Learning*, Yoshua Bengio and Ian Goodfellow

Long-Short Term Memory (LSTMs) Networks

LSTM Cell



An LSTM cell for one time step [1]

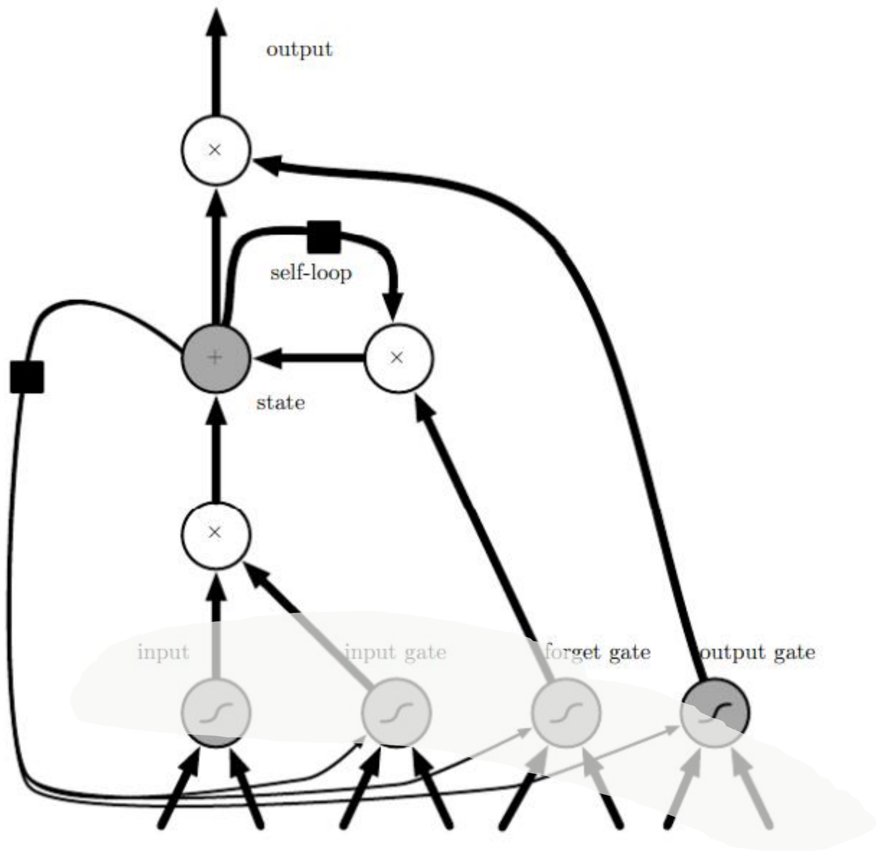
- State update equation:
  $$\underbrace{\boldsymbol{s}^{(t)}}_{\text{State variable for present time}} = \underbrace{\boldsymbol{f}^{(t)} \circ \boldsymbol{s}^{(t-1)}}_{\text{Forget vector controlling contribution from past state vector via internal self loop}} + \underbrace{\boldsymbol{g}^{(t)} \circ \boldsymbol{\sigma}\left(\boldsymbol{b} + \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{W}\boldsymbol{h}^{(t-1)}\right)}_{\text{External input gate vector controlling contribution from external input and past state at present time}}$$

- Since the present state is contributed to by the old state both directly (through forget gate) and indirectly (via external input gate), the vanishing gradient problem is mitigated
  - Weak gradients from the latter can be compensated by strong gradients from the former

[1] Chapter 10, *Deep Learning*, Yoshua Bengio and Ian Goodfellow

Long-Short Term Memory (LSTMs) Networks

LSTM Cell



An LSTM cell for one time step [1]

- The gate update equations are:

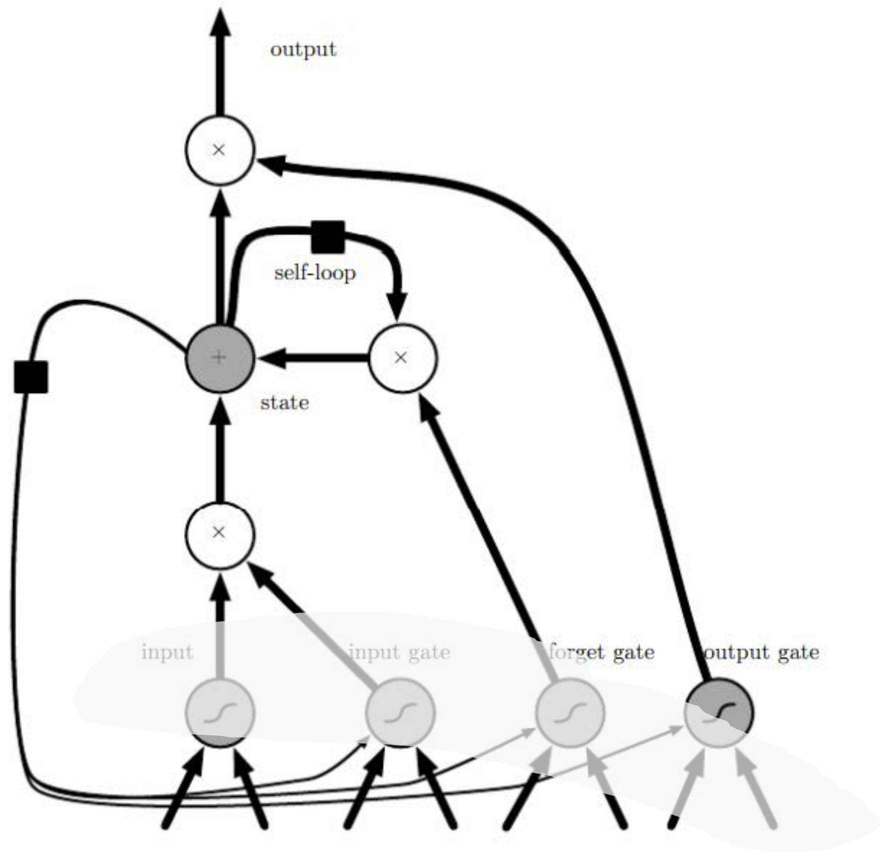
$$\boldsymbol{f}^{(t)} = \boldsymbol{\sigma}\left(\boldsymbol{b}^{f} + \boldsymbol{U}^{f}\boldsymbol{x}^{(t)} + \boldsymbol{W}^{f}\boldsymbol{h}^{(t-1)}\right)$$
$$\boldsymbol{g}^{(t)} = \boldsymbol{\sigma}\left(\boldsymbol{b}^{g} + \boldsymbol{U}^{g}\boldsymbol{x}^{(t)} + \boldsymbol{W}^{g}\boldsymbol{h}^{(t-1)}\right)$$

Hence the gates themselves are learned parameters conditioned on the context!

[1] Chapter 10, *Deep Learning*, Yoshua Bengio and Ian Goodfellow

Long-Short Term Memory (LSTMs) Networks

LSTM Cell



An LSTM cell for one time step [1]

- The output equations are:

$$\boldsymbol{h}^{(t)} = \tanh\left(\boldsymbol{s}^{(t)}\right) \circ \boldsymbol{q}^{(t)}$$

$$\boldsymbol{q}^{(t)} = \boldsymbol{\sigma}\left(\boldsymbol{b}^{o} + \boldsymbol{U}^{o}\boldsymbol{x}^{(t)} + \boldsymbol{W}^{o}\boldsymbol{h}^{(t-1)}\right)$$

- $\boldsymbol{q}^{(t)}$ is output gate also determined via learned parameters through context.

[1] Chapter 10, *Deep Learning*, Yoshua Bengio and Ian Goodfellow

Gated Recurrent Units (GRUs)

LSTM Inefficiency

- Other variants of RNNs have been proposed to similarly enable the use of self connections and gates to control information flow, retain context, and incorporate new information where needed

- One such variant, called the GRU minimizes the redundancy found in LSTMs to have only a single gate to control the influx of information from past and present

- The update equation is

$$\boldsymbol{h}^{(t)} = \boldsymbol{u}^{(t)} \circ \boldsymbol{h}^{(t-1)} + (\boldsymbol{1} - \boldsymbol{u})^{(t)} \circ \boldsymbol{\sigma}\left(\boldsymbol{b} + \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{W}\boldsymbol{h}^{(t-1)}\right)$$

Gated Recurrent Units (GRUs)

Update Equation

- The gate equations are given as

$$\boldsymbol{u}^{(t)} = \boldsymbol{\sigma}\left(\boldsymbol{b}^{u} + \boldsymbol{U}^{u}\boldsymbol{x}^{(t)} + \boldsymbol{W}^{u}\boldsymbol{h}^{(t-1)}\right)$$
$$\boldsymbol{r}^{(t)} = \boldsymbol{\sigma}\left(\boldsymbol{b}^{r} + \boldsymbol{U}^{r}\boldsymbol{x}^{(t)} + \boldsymbol{W}^{r}\boldsymbol{h}^{(t-1)}\right)$$

- The update gate can on one extreme choose to only use past state (when the sigmoid outputs 1) and on the other extreme choose to only use present information

- The reset gate controls what portions of past state get used to compute the next state

Other Recurrent Configurations

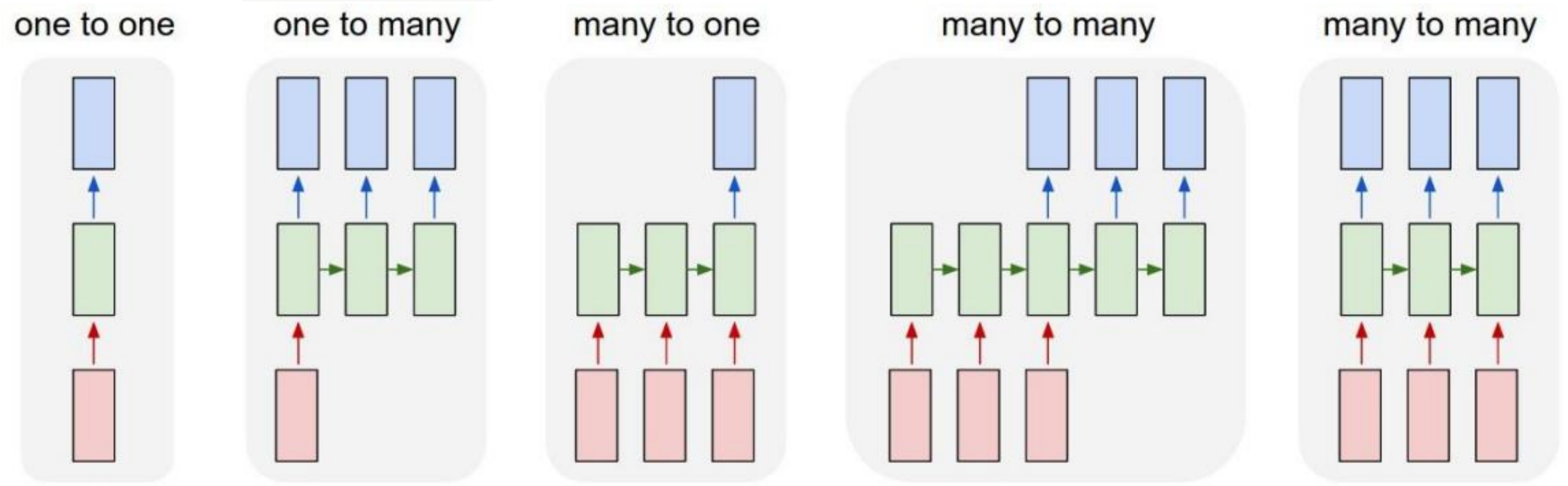
Update Equation

An MLP for non-sequence related tasks

For sequence generation tasks e.g., image captioning, music generation etc.

For sequence classification e.g., sentiment recognition

For text summarization and machine translation



Various configurations for RNNs and other associated models. Image courtesy [1]. Blue boxes denote outputs, red boxes the inputs, and green boxes the hidden states.

[1] Stanford CS231n Lecture Slides

Sequence Modeling

Overcoming Limitations: Parameter Sharing

- A convolutional kernel can be thought of as a learned function applied to various portions of the image to look for specific features (e.g., edges, shapes etc.)
  - Results in spatial invariance to object positions
  - Not limited by fixed image size since kernel slides over the whole image regardless of dimensions
  - Can think of this as multiple identical functions with shared parameters acting over individual input windows
  - Fixed parameter size only dependent on the convolutional kernel and not on image size

- Words can happen at different positions in sentence sequences; sequences can be of different lengths.
  - Need a single common function scanning a sequence (regardless of length/ordering) to look for pertinent information

I went to Japan in 2009

My name is John. In 2009, I went to Japan

Temporal Convolution Networks

Motivation

- Recurrent networks and associated variants suffer from following limitations:

Parallelism	Receptive Field
Computations for future time steps must wait for earlier time steps since the former need state vectors from the latter	How much the network can look back in the input sequence for context is an indirect function of architecture and other design parameters—not easily controlled
Training Time Memory	**Gradient Stability**
Storing partial results through a long sequence can use up significant memory resources while training recurrent networks	Long input sequences can lead to unstable gradients in the network that either vanish or explode

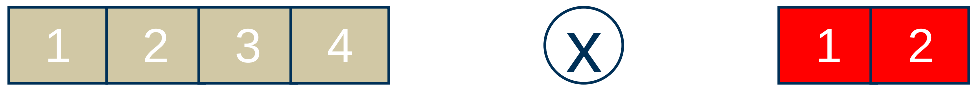
Possible Solution: sequence models built using 1D CNNs

[1] Bai, Shaojie, J. Zico Kolter, and Vladlen Koltun. "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling." *arXiv preprint arXiv:1803.01271* (2018).

Temporal Convolution Networks

Computation

1D CNNs work by sliding a fixed set of kernels through the sequence. This operation can be actualized via multiplication of a Toeplitz matrix with a vector, and hence parallelizable, leading to faster computations.



Convolution of the above sequence (blue) with the kernel (red) is equivalent to below:

$$\begin{bmatrix} 1 & 2 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

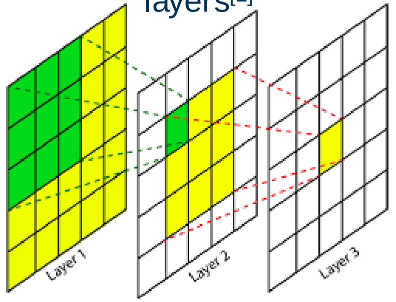
Temporal Convolution Networks
Dilated Convolution

- However, CNNs require very deep architectures to cover the input receptive field
- Solution: Dilated convolution

- Dilated convolution expands kernels by adding 'holes'
- Allows larger receptive field in non-deep layers

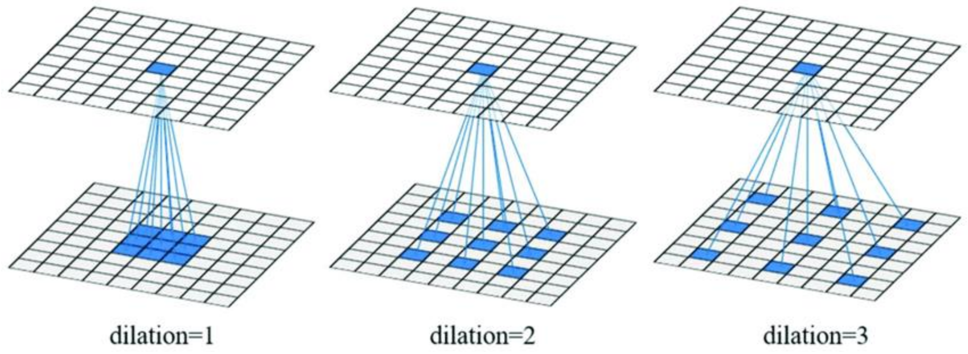
- Used in image processing and audio processing applications

Stacking multiple layers[1]



Multiresolution kernels

Dilated Convolutions[2]

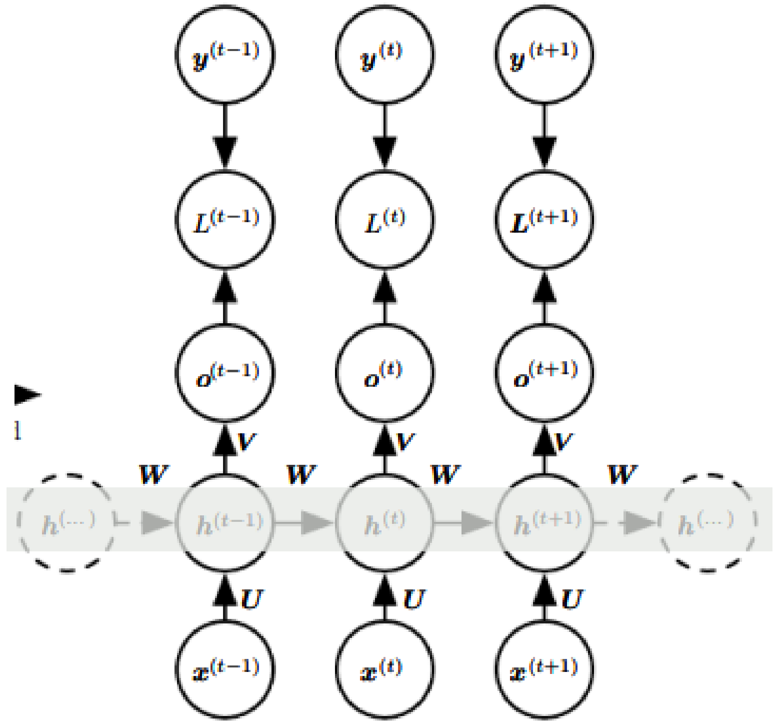


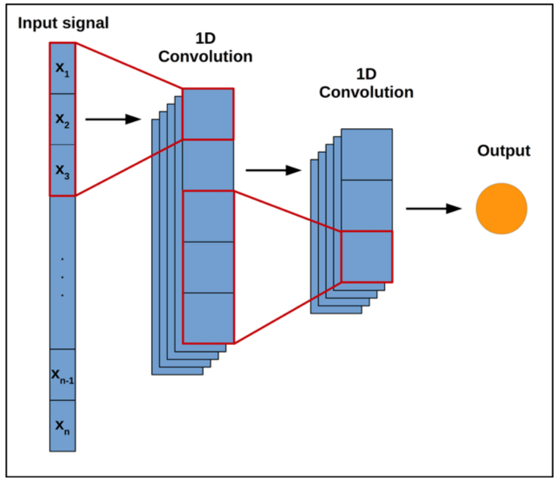
[1] https://theaisummer.com/receptive-field/
[2] Cui, Ximin, et al. "Multiscale spatial-spectral convolutional network with image-based framework for hyperspectral imagery classification." *Remote Sensing* 11.19 (2019): 2220.

Temporal Convolution Networks

Computational Complexity

- CNNs only use a fixed set of kernels per layer. RNNs store many intermediate state variables increasing the computational bottleneck

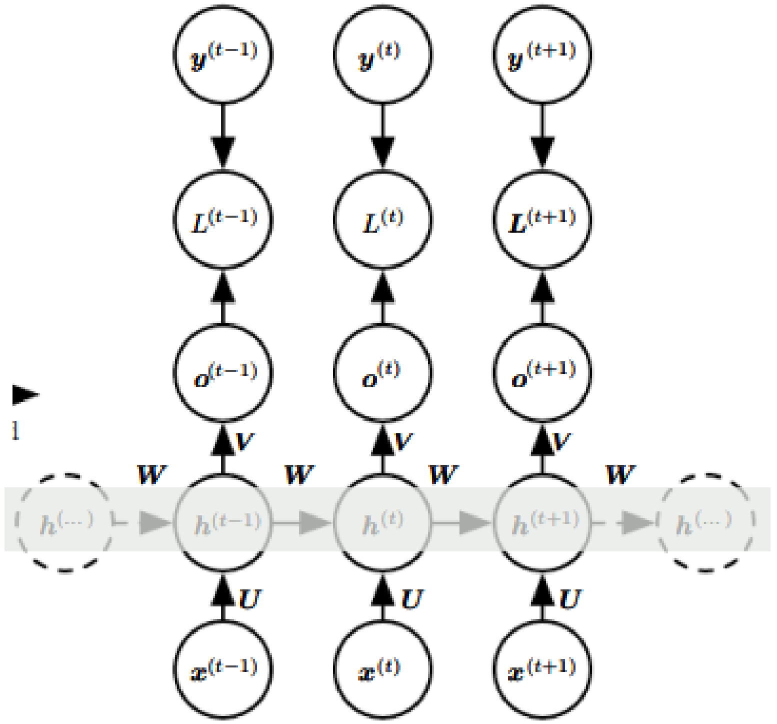


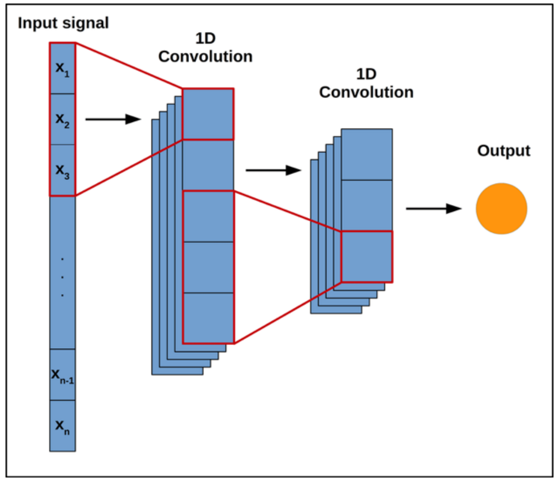


Temporal Convolution Networks

Computational Complexity

- CNNs only use a fixed set of kernels per layer. RNNs store many intermediate state variables increasing the computational bottleneck
- Gradient in an RNN involve long product chains of intermediate state variables, leading to vanishing/exploding gradient problem for long sequences
    - In a CNN, gradients are backpropagated laterally, rather than vertically through the input sequence length [1]





[1] Shenfield, Alex, and Martin Howarth. "A novel deep learning model for the detection and identification of rolling element-bearing faults." *Sensors* 20, no. 18 (2020): 5112.

Temporal Convolution Networks

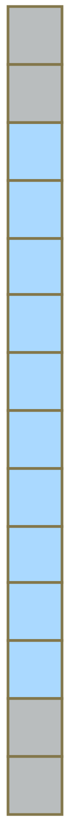
Advantages

- **Input sequence invariability:** The TCN operates on a sequence regardless of its length to produce another sequence of the same length (similar to an RNN)
- **Causal sequence modeling:** Each term in the output sequence is a function of only past samples in the input sequence. This is achieved by specifying certain combinations of dilation and padding hyperparameters (more on this later)
- **Residual units for ease of optimization:** The TCN is composed of multiple, stacked units mimicking residual units in 2D CNNs for vision tasks. This allows for a deeper network that can be efficiently optimized without running into gradient-related issues
- **Optimal receptive field design:** The network uses dilated causal convolutions over multiple layers to sufficiently cover the entire context of past input samples needed to predict a given sample for the output sequence

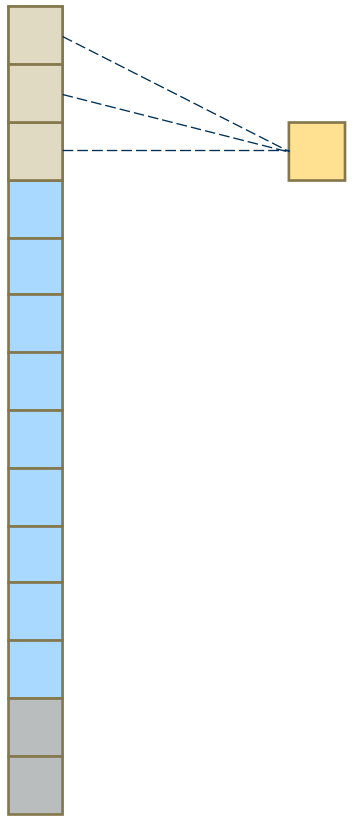
Temporal Convolution Networks

Causal Convolutions

$$y^{(t)} = f(x^{(t)}, x^{(t-1)}, \dots, x^{(1)})$$



For a kernel of length k, take the input sequence and append on both sides with $k-1$ zeros.

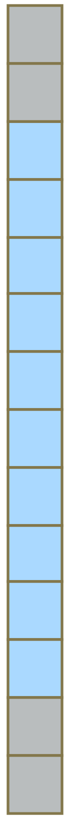


As the kernel slides along the padded input sequence, each output sample is produced as function of only past input samples

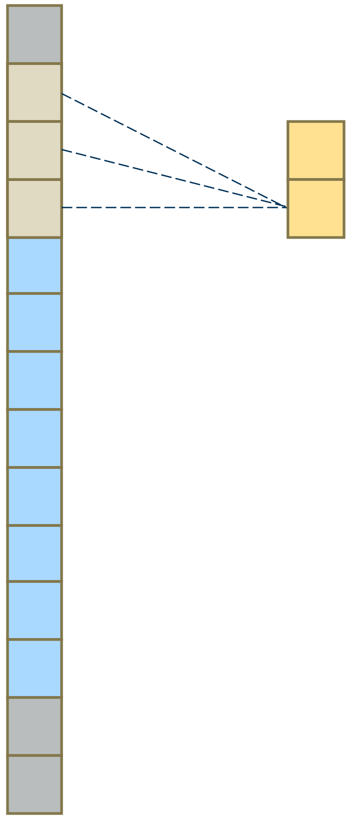
Temporal Convolution Networks

Causal Convolutions

$$y^{(t)} = f(x^{(t)}, x^{(t-1)}, \dots, x^{(1)})$$



For a kernel of length k, take the input sequence and append on both sides with $k-1$ zeros.

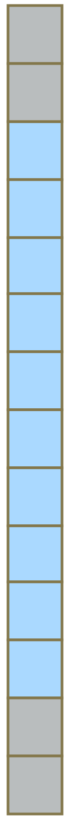


As the kernel slides along the padded input sequence, each output sample is produced as function of only past input samples

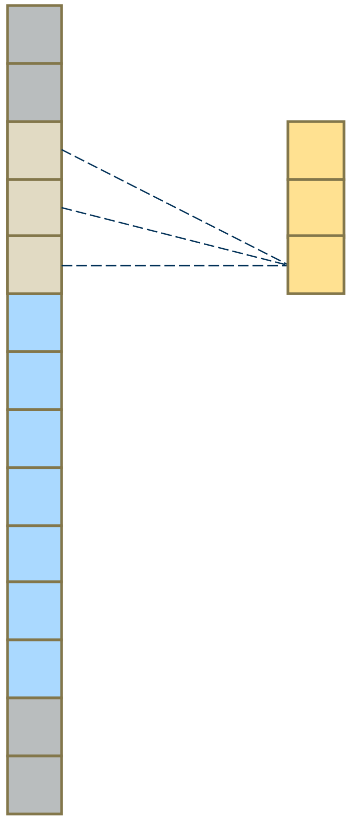
Temporal Convolution Networks

Causal Convolutions

$$y^{(t)} = f\left(x^{(t)}, x^{(t-1)}, \ldots, x^{(1)}\right)$$



For a kernel of length k, take the input sequence and append on both sides with $k - 1$ zeros.

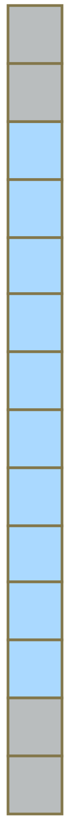


As the kernel slides along the padded input sequence, each output sample is produced as function of only past input samples

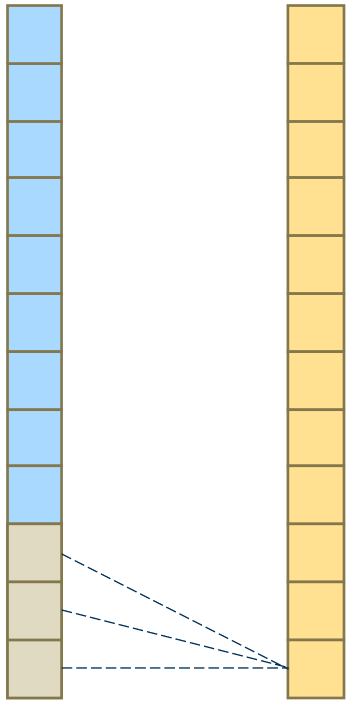
Temporal Convolution Networks

Causal Convolutions

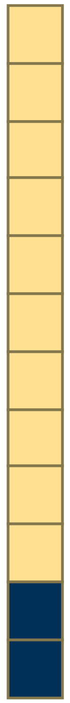
$$y^{(t)} = f(x^{(t)}, x^{(t-1)}, \dots, x^{(1)})$$



For a kernel of length k, take the input sequence and append on both sides with $k-1$ zeros.



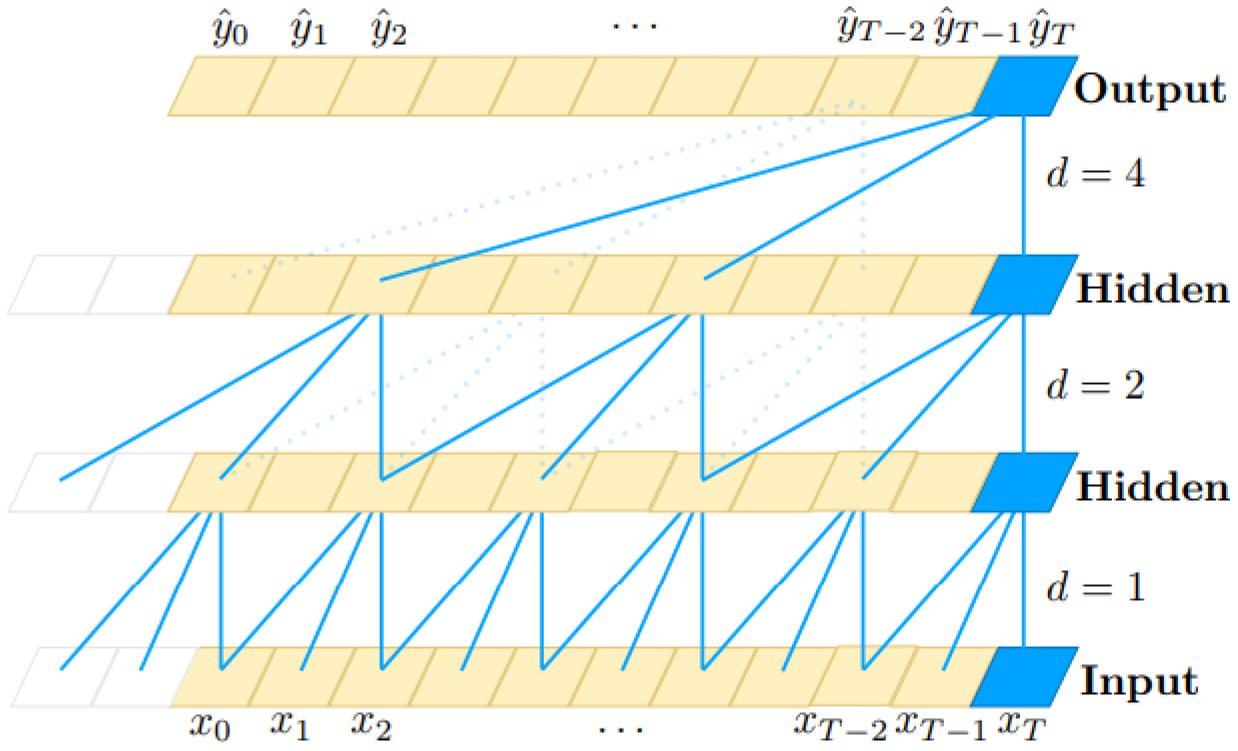
As the kernel slides along the padded input sequence, each output sample is produced as function of only past input samples



Remove the last $k-1$ samples from the output to obtain a sequence equal in length to the input and a causal function of the input samples

Temporal Convolution Networks

Dilated Convolutions



$\hat{y}_0$ $\hat{y}_1$ $\hat{y}_2$ ... $\hat{y}_{T-2}$ $\hat{y}_{T-1}$ $\hat{y}_T$
Output
$d = 4$
Hidden
$d = 2$
Hidden
$d = 1$
Input
x_0 x_1 x_2 ... x_{T-2} x_{T-1} x_T

Multiple, stacked dilated convolutions [1]

Network should have sufficient depth to cover the entire context of input length to predict the last output sample

- Increasing the dilation factor helps expand the effective receptive field of the kernel while maintaining the number of learnable parameters constant.
- TCN used exponentially increasing dilation factors to increase the receptive field at an exponential rate through the network layers.
- Number of zeros padded to each side of the input sequence then becomes $d(k-1)$, where d and k refer to the dilation factor and kernel size respectively. The convolution still occurs causally as before.

[1] Bai, Shaojie, J. Zico Kolter, and Vladlen Koltun. "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling." *arXiv preprint arXiv:1803.01271* (2018).

Temporal Convolution Networks

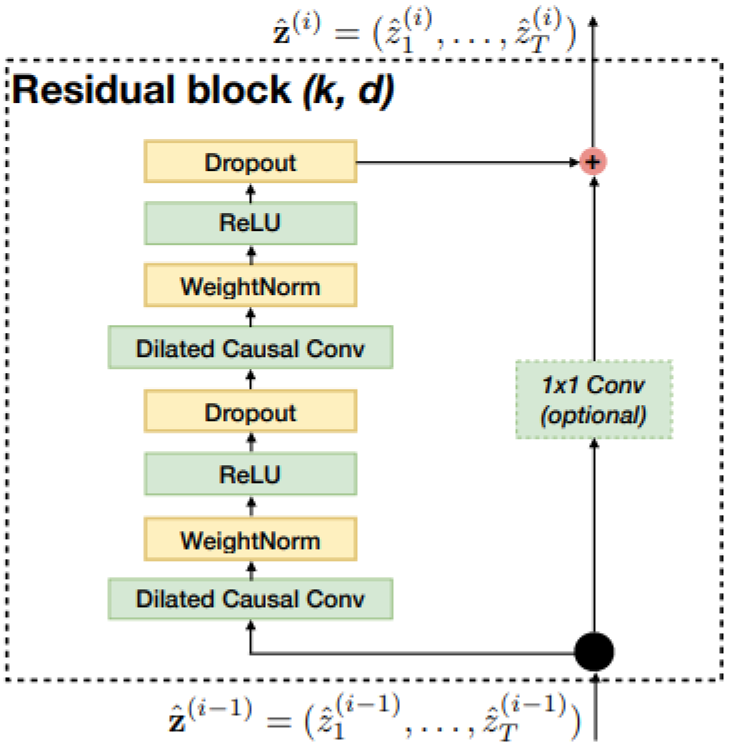
Weight Normalization

- Batch Norm: Requires a large batch size for stable calculations
  - Not good for RNNs/LSTMs
- Alternative is Weight Normalization $w = \frac{\mathrm{g}\, v}{||v||}$
- Directly optimize for g and v.
  - g = weight vector
  - $v/||v||$ = direction of weight vector
- Stochastic gradient descent convergence is sped up
- Weight norm is more robust to learning rates

[1] Salimans, Tim, and Durk P. Kingma. "Weight normalization: A simple reparameterization to accelerate training of deep neural networks." *Advances in neural information processing systems* 29 (2016)..

Temporal Convolution Networks

Residual Units



$\hat{\mathbf{z}}^{(i)} = (\hat{z}_1^{(i)}, \ldots, \hat{z}_T^{(i)})$
Residual block (k, d)
Dropout
ReLU
WeightNorm
Dilated Causal Conv
Dropout
ReLU
WeightNorm
Dilated Causal Conv
1x1 Conv (optional)
$\hat{\mathbf{z}}^{(i-1)} = (\hat{z}_1^{(i-1)}, \ldots, \hat{z}_T^{(i-1)})$

Structure of a single residual block in a TCN [1]

- The network architecture consists of **multiple residual blocks** laterally stacked together. Each block has the same general structure.
- **Dilated causal convolution** layers extract features from the input sequence as discussed earlier
- **Weight normalization** layers decouple gradient magnitudes from their direction to help speed convergence
- **ReLU layers** introduce non-linearity into the learning process
- **Dropout layers** reduce the risk of overfitting by randomly turning a certain prespecified portion of activations off for every minibatch of examples
- **Skip connection** through a 1×1 conv layer maps the input sequence to have the same number of channels as the output of the last dropout layer in the block. This allows the input and output sequences to be added together via a skip connection

[1] Bai, Shaojie, J. Zico Kolter, and Vladlen Koltun. "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling." *arXiv preprint arXiv:1803.01271* (2018).

Appendix
Notations

- x_i: a single feature
- $\boldsymbol{x}_i$: feature vector (a data sample)
- $\boldsymbol{x}_{:,i}$: feature vector of all data samples
- $\boldsymbol{X}$: matrix of feature vectors (dataset)
- N: number of data samples
- $\boldsymbol{W}$: weight matrix
- $\boldsymbol{b}$: bias vector
- $\boldsymbol{v}(t)$: first moment at time t
- $\boldsymbol{G}(t)$: second moment at time t
- $\boldsymbol{H}(\boldsymbol{\theta})$: Hessian matrix
- P: number of features in a feature vector
- α: learning rate
- Bold letter/symbol: vector
- Bold capital letters/symbol: matrix