CSCI-567 Machine Learning (Spring 2021) Special Topics: Representation Learning and Time-series Processing with Neural Networks

Nitin Kamra

University of Southern California

April 21, 2021

Acknowledgements

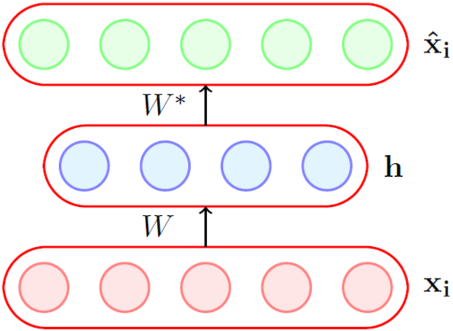
The materials borrow *heavily* from the following sources:

- Chris Olah's blog post on LSTMs: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`
- Dr. Nasim Zolaktaf's UBC lecture on recurrent neural networks: `https://www.cs.ubc.ca/labs/lci/mlrg/slides/rnn.pdf`
- Dr. Mitesh M. Khapra's lectures on Deep Learning: `https://www.cse.iitm.ac.in/~miteshk/CS7015/Slides/Teaching/pdf/Lecture7.pdf`

Outline

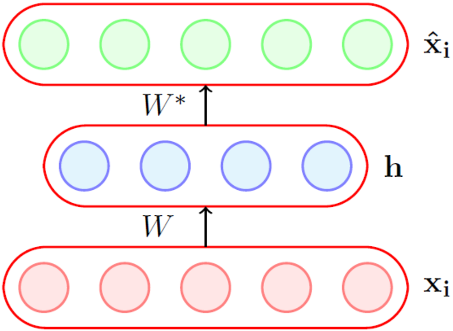
1. Autoencoders
2. Recurrent Neural Networks

Introduction to Autoencoder



An autoencoder is a special type of feedforward network which does the following:

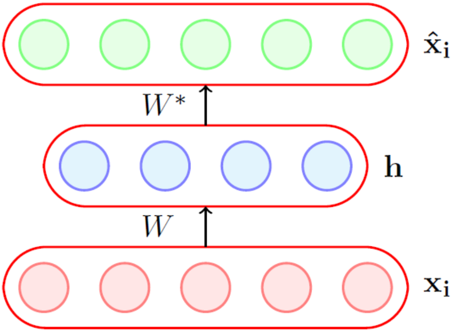
Introduction to Autoencoder



An autoencoder is a special type of feedforward network which does the following:

- Encodes its input $\boldsymbol{x_i}$ into a hidden representation h

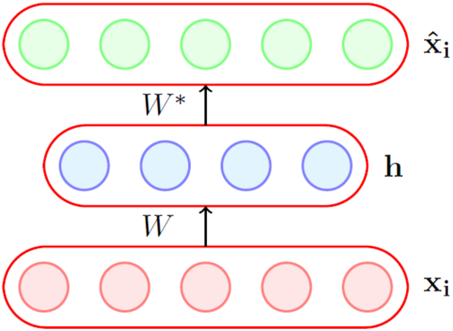
Introduction to Autoencoder



An autoencoder is a special type of feedforward network which does the following:

- Encodes its input $\boldsymbol{x_i}$ into a hidden representation h
- Decodes the input again from this hidden representation h

Introduction to Autoencoder

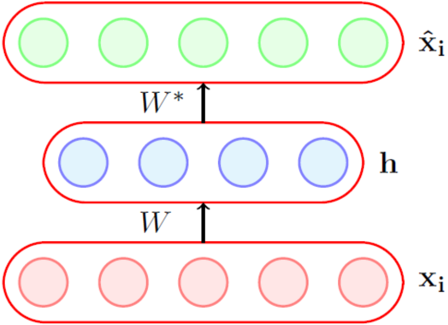


An autoencoder is a special type of feedforward network which does the following:

- Encodes its input $\boldsymbol{x_i}$ into a hidden representation h
- Decodes the input again from this hidden representation h

The model is trained to minimize a certain loss function which ensures that $\boldsymbol{\hat{x}_i}$ is close to $\boldsymbol{x_i}$.

Introduction to Autoencoder



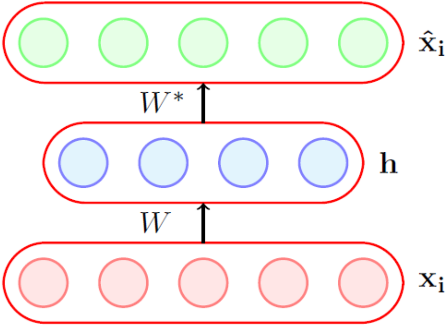
$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

An autoencoder is a special type of feedforward network which does the following:

- Encodes its input $\boldsymbol{x_i}$ into a hidden representation h
- Decodes the input again from this hidden representation h

The model is trained to minimize a certain loss function which ensures that $\hat{\boldsymbol{x}}_{\boldsymbol{i}}$ is close to $\boldsymbol{x_i}$.

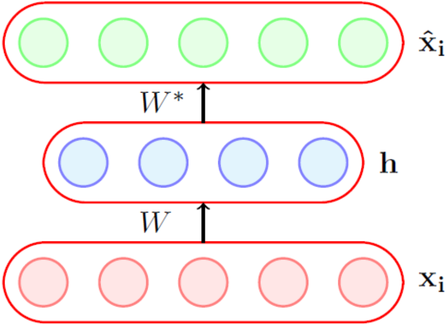
Why autoencoders?



$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

What should be the dimension of $\boldsymbol{h}$?

Why autoencoders?

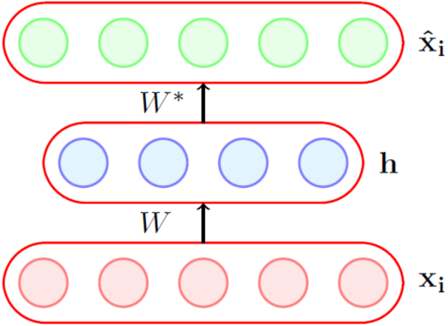


$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

What should be the dimension of $\boldsymbol{h}$?

- If $\dim(\boldsymbol{h}) \geq \dim(\boldsymbol{x_i})$, then the neural network can always perfectly recover $\boldsymbol{x_i}$ from $\boldsymbol{h}$ by just copying its elements → Not interesting!

Why autoencoders?

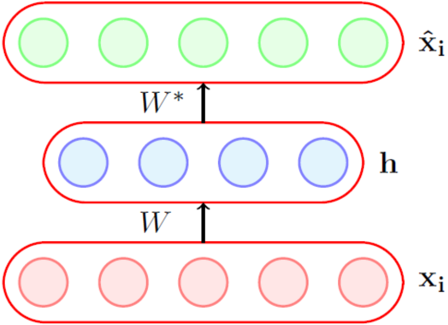


$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

What should be the dimension of $\boldsymbol{h}$?

- If dim$(\boldsymbol{h}) \geq$ dim$(\boldsymbol{x_i})$, then the neural network can always perfectly recover $\boldsymbol{x_i}$ from $\boldsymbol{h}$ by just copying its elements → Not interesting!
- If dim$(\boldsymbol{h}) <$ dim$(\boldsymbol{x_i})$, then the neural network will have to encode maximum information from $\boldsymbol{x_i}$ into $\boldsymbol{h}$ for an accurate reconstruction!

Why autoencoders?

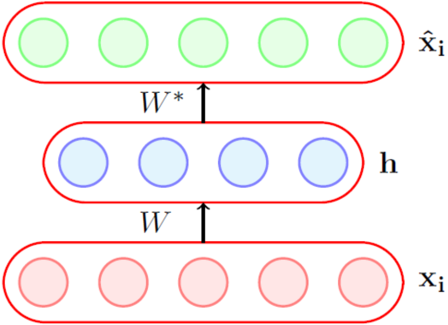


$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

Hence, one can use autoencoders for:

- Representation learning
- Dimensionality reduction
- Finding hidden structure in data
- Data compression
- Clustering
- Anomaly detection

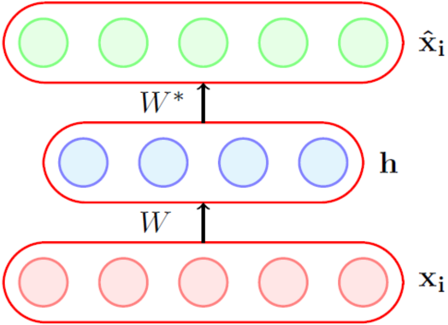
Choice of f and g



$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

What should be the choice for decoder activation f?

Choice of f and g

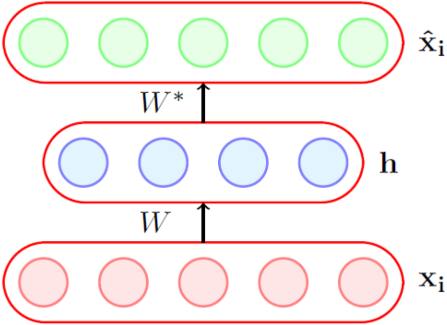


$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

What should be the choice for decoder activation f?

- What if inputs are binary?

Choice of f and g

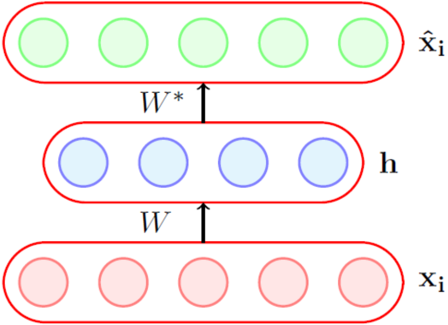


$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

What should be the choice for decoder activation f?

- What if inputs are binary?
- Answer: sigmoid

Choice of f and g

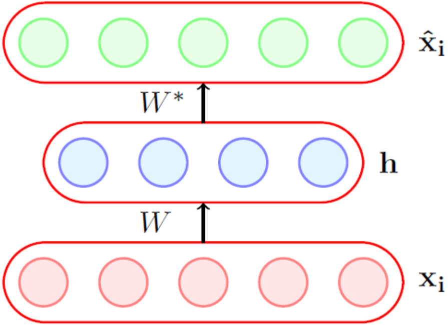


$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

What should be the choice for decoder activation f?

- What if inputs are binary?
- Answer: sigmoid
- What if inputs are real-valued?

Choice of f and g



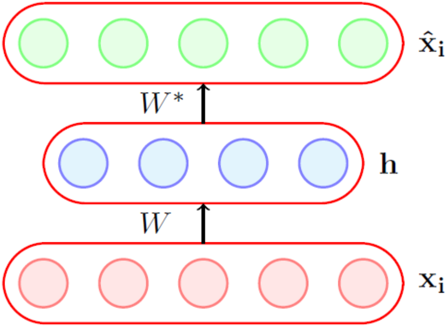
$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

What should be the choice for decoder activation f?

- What if inputs are binary?
- Answer: sigmoid
- What if inputs are real-valued?
- Answer: identity

What should be the choice for encoder activation g?

Choice of f and g



$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

What should be the choice for

decoder activation f?

- What if inputs are binary?
- Answer: sigmoid
- What if inputs are real-valued?
- Answer: identity

What should be the choice for encoder activation g?

Answer: Typically, g is chosen as the sigmoid or tanh function to keep embedding values ($\boldsymbol{h}$) bounded and introduce non-linearity.

Choice of loss function

What should be the loss function to train an autoencoder?

Choice of loss function

What should be the loss function to train an autoencoder?

Consider the case when inputs are real-valued:

Choice of loss function

What should be the loss function to train an autoencoder?

Consider the case when inputs are real-valued:

- The objective of the autoencoder is to reconstruct $\hat{x}_i$ to be as close to x_i as possible.

Choice of loss function

What should be the loss function to train an autoencoder?

Consider the case when inputs are real-valued:

- The objective of the autoencoder is to reconstruct $\boldsymbol{\hat{x}_i}$ to be as close to $\boldsymbol{x_i}$ as possible.
- This can be formalized using the following objective function:

$$\min_{W,W*,\boldsymbol{c},\boldsymbol{b}} \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}(\hat{x}_{ij}-x_{ij})^2$$

$$=\min_{W,W*,\boldsymbol{c},\boldsymbol{b}} \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{\hat{x}_i}-\boldsymbol{x_i})^T(\boldsymbol{\hat{x}_i}-\boldsymbol{x_i})$$

Choice of loss function

What should be the loss function to train an autoencoder?

Consider the case when inputs are real-valued:

- The objective of the autoencoder is to reconstruct $\hat{\boldsymbol{x}}_{\boldsymbol{i}}$ to be as close to $\boldsymbol{x}_{\boldsymbol{i}}$ as possible.
- This can be formalized using the following objective function:

$$\min_{W, W*, \boldsymbol{c}, \boldsymbol{b}} \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} (\hat{x}_{ij} - x_{ij})^2$$

$$= \min_{W, W*, \boldsymbol{c}, \boldsymbol{b}} \frac{1}{m} \sum_{i=1}^{m} (\hat{\boldsymbol{x}}_{\boldsymbol{i}} - \boldsymbol{x}_{\boldsymbol{i}})^T (\hat{\boldsymbol{x}}_{\boldsymbol{i}} - \boldsymbol{x}_{\boldsymbol{i}})$$

- We can then train the autoencoder using back-propagation.

Choice of loss function

What should be the loss function to train an autoencoder?

Consider the case when inputs are real-valued:

- The objective of the autoencoder is to reconstruct $\hat{\boldsymbol{x}}_{\boldsymbol{i}}$ to be as close to $\boldsymbol{x}_{\boldsymbol{i}}$ as possible.
- This can be formalized using the following objective function:

$$\min_{W,W*,\boldsymbol{c},\boldsymbol{b}} \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}(\hat{x}_{ij}-x_{ij})^2$$

$$=\min_{W,W*,\boldsymbol{c},\boldsymbol{b}} \frac{1}{m}\sum_{i=1}^{m}(\hat{\boldsymbol{x}}_{\boldsymbol{i}}-\boldsymbol{x}_{\boldsymbol{i}})^T(\hat{\boldsymbol{x}}_{\boldsymbol{i}}-\boldsymbol{x}_{\boldsymbol{i}})$$

- We can then train the autoencoder using back-propagation.

Homework question: What if the inputs were binary-valued?

Connection to Principal Components Analysis

Connection to Principal Components Analysis

An autoencoder boils down to performing PCA on real-valued data when we:

Connection to Principal Components Analysis

An autoencoder boils down to performing PCA on real-valued data when we:

- use a linear encoder ($g =$ identity)

Connection to Principal Components Analysis

An autoencoder boils down to performing PCA on real-valued data when we:

- use a linear encoder ($g =$ identity)
- use a linear decoder ($f =$ identity)

Connection to Principal Components Analysis

An autoencoder boils down to performing PCA on real-valued data when we:

- use a linear encoder ($g =$ identity)
- use a linear decoder ($f =$ identity)
- use squared error loss function

Connection to Principal Components Analysis

An autoencoder boils down to performing PCA on real-valued data when we:

- use a linear encoder ($g =$ identity)
- use a linear decoder ($f =$ identity)
- use squared error loss function
- center the input by subtracting column-wise means

Autoencoders beyond PCA

Autoencoders can be much more flexible than doing PCA:

Autoencoders beyond PCA

Autoencoders can be much more flexible than doing PCA:

- **Feasibility on binary-valued data**: PCA works for real-valued data but autoencoders can apply to binary-valued data with sigmoid activation for f and a suitable loss function.

Autoencoders beyond PCA

Autoencoders can be much more flexible than doing PCA:

- **Feasibility on binary-valued data**: PCA works for real-valued data but autoencoders can apply to binary-valued data with sigmoid activation for f and a suitable loss function.
- **Non-linear Dimensionality Reduction**: PCA only uses linear transformation on data. Autoencoders permit non-linearity via activations.

Autoencoders beyond PCA

Autoencoders can be much more flexible than doing PCA:

- **Feasibility on binary-valued data**: PCA works for real-valued data but autoencoders can apply to binary-valued data with sigmoid activation for f and a suitable loss function.
- **Non-linear Dimensionality Reduction**: PCA only uses linear transformation on data. Autoencoders permit non-linearity via activations.
- **Deep autoencoders**: One may also use multiple layers in both encoder and decoder to allow learning more flexible hidden representations of the data.

Autoencoders beyond PCA

Autoencoders can be much more flexible than doing PCA:

- **Feasibility on binary-valued data**: PCA works for real-valued data but autoencoders can apply to binary-valued data with sigmoid activation for f and a suitable loss function.
- **Non-linear Dimensionality Reduction**: PCA only uses linear transformation on data. Autoencoders permit non-linearity via activations.
- **Deep autoencoders**: One may also use multiple layers in both encoder and decoder to allow learning more flexible hidden representations of the data.
- **Denoising autoencoders**: Corrupt the input with probabilistic noise before sending it into the autoencoder. Trains the autoencoder to de-noise input.

Autoencoders beyond PCA

Autoencoders can be much more flexible than doing PCA:

- **Feasibility on binary-valued data**: PCA works for real-valued data but autoencoders can apply to binary-valued data with sigmoid activation for f and a suitable loss function.
- **Non-linear Dimensionality Reduction**: PCA only uses linear transformation on data. Autoencoders permit non-linearity via activations.
- **Deep autoencoders**: One may also use multiple layers in both encoder and decoder to allow learning more flexible hidden representations of the data.
- **Denoising autoencoders**: Corrupt the input with probabilistic noise before sending it into the autoencoder. Trains the autoencoder to de-noise input.
- **Sparse autoencoders**: Use a sigmoid function for g and include a sparsity penalty on the hidden representations in the loss function.

Outline

1 Autoencoders

2 Recurrent Neural Networks

Sequential Data

- Sometimes the sequence of data matters.
  - Text generation
  - Stock price prediction

Sequential Data

- Sometimes the sequence of data matters.
  - Text generation
  - Stock price prediction
- The clouds are in the .... ?

Sequential Data

- Sometimes the sequence of data matters.
  - Text generation
  - Stock price prediction
- The clouds are in the .... ?
  - sky

Sequential Data

- Sometimes the sequence of data matters.
  - Text generation
  - Stock price prediction
- The clouds are in the .... ?
  - sky
- Simple solution: N-grams?

Sequential Data

- Sometimes the sequence of data matters.
  - Text generation
  - Stock price prediction
- The clouds are in the .... ?
  - sky
- Simple solution: N-grams?
  - Hard to represent patterns with more than a few words (possible patterns increases exponentially)

Sequential Data

- Sometimes the sequence of data matters.
  - Text generation
  - Stock price prediction
- The clouds are in the .... ?
  - sky
- Simple solution: N-grams?
  - Hard to represent patterns with more than a few words (possible patterns increases exponentially)
- Simple solution: Neural networks?
  - Fixed input/output size
  - Fixed number of steps

Recurrent Neural Networks

- **Recurrent neural networks (RNNs)** are networks with loops, allowing information to persist [Rumelhart et al., 1986].

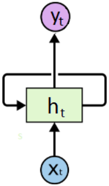
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

input vector at some time step

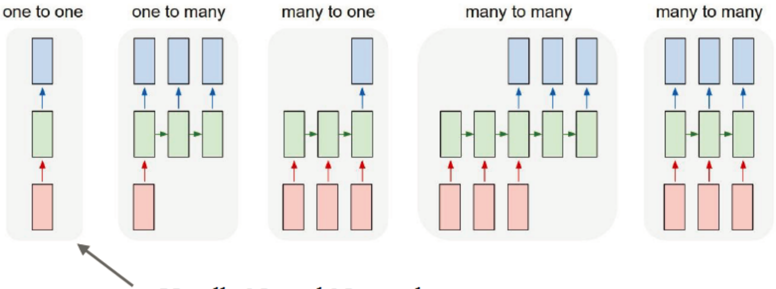
some function with parameters W



- Have **memory** that keeps track of information observed so far
- Maps from the entire history of previous inputs to each output
- Handle sequential data

Adapted from: *N. Cinbis and C. Olah*

Recurrent Networks Offer a Lot of Flexibility



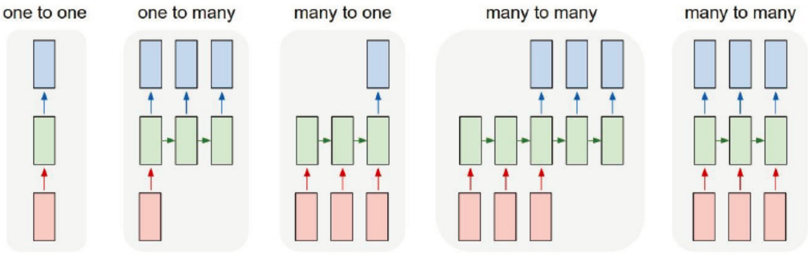
Vanilla Neural Network

fixed-sized input -> fixed-size output

e.g. image classification

Adapted from: *A. Karpathy*

Recurrent Networks Offer a Lot of Flexibility



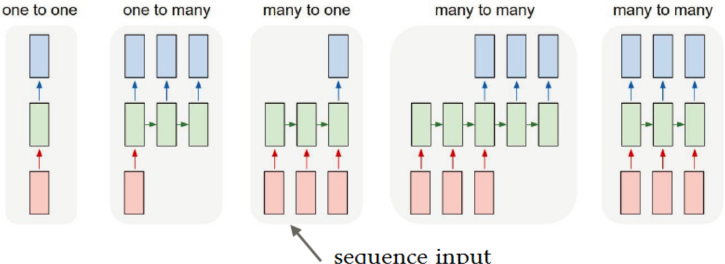
sequence output

e.g. image captioning

image -> sequence of words

Adapted from: *A. Karpathy*

Recurrent Networks Offer a Lot of Flexibility



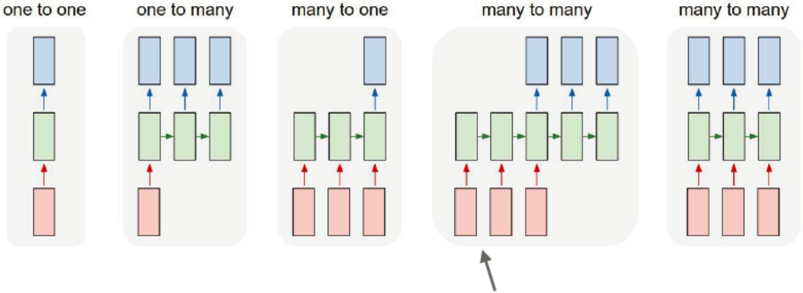
sequence input

e.g. sentiment classification

sequence of words -> sentiment

Adapted from: *A. Karpathy*

Recurrent Networks Offer a Lot of Flexibility

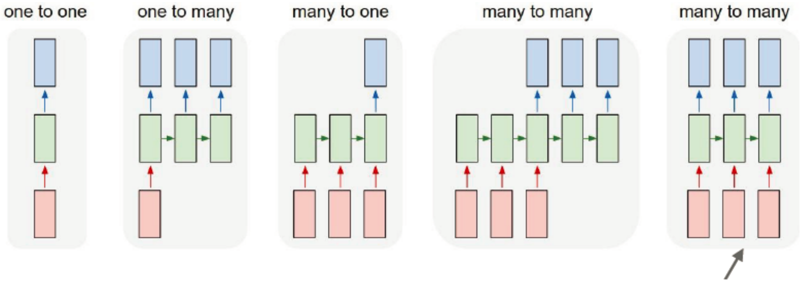


sequence input and sequence output

e.g. machine translation

seq of words -> seq of words

Recurrent Networks Offer a Lot of Flexibility



synced sequence input and output

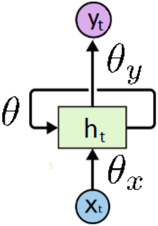
e.g. video classification on frame level

Adapted from: *A. Karpathy*

Recurrent Neural Networks

$$\mathbf{h}_t = \theta\phi(\mathbf{h}_{t-1}) + \theta_x \mathbf{x}_t$$

$$\mathbf{y}_t = \theta_y \phi(\mathbf{h}_t)$$

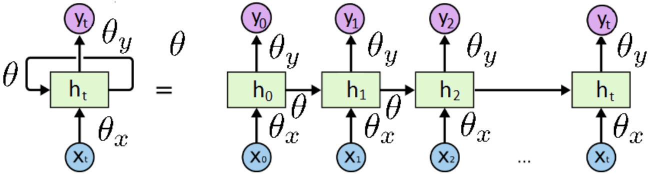


- $\mathbf{x}_t$ is the **input** at time t.
- $\mathbf{h}_t$ is the **hidden state** (memory) at time t.
- $\mathbf{y}_t$ is the **output** at time t.
- θ, θ_x, θ_y are distinct **weights**.
  - weights are the same at all time steps.

Adapted from: *C. Olah*

Recurrent Neural Networks

- RNNs can be thought of as multiple copies of the same network, each passing a message to a successor.

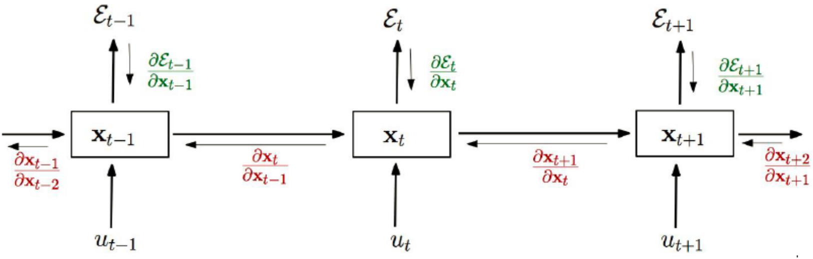


- The same function and the same set of parameters are used at every time step.
    - Are called recurrent because they perform the same task for each input.

Adapted from: *C. Olah*

Back-Propagation Through Time (BPTT)

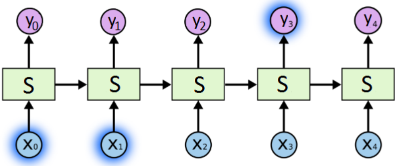
- Using the generalized back-propagation algorithm one can obtain the so-called **Back-Propagation Through Time** algorithm.
- The recurrent model is represented as a multi-layer one (with an unbounded number of layers) and backpropagation is applied on the unrolled model.



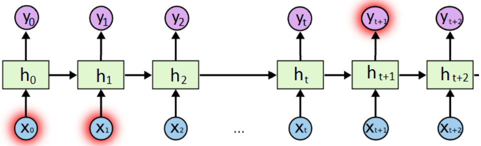
$\mathcal{E}_{t-1}$
$\frac{\partial \mathcal{E}_{t-1}}{\partial \mathbf{x}_{t-1}}$
$\mathcal{E}_t$
$\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t}$
$\mathcal{E}_{t+1}$
$\frac{\partial \mathcal{E}_{t+1}}{\partial \mathbf{x}_{t+1}}$
$\mathbf{x}_{t-1}$
$\mathbf{x}_t$
$\mathbf{x}_{t+1}$
$\frac{\partial \mathbf{x}_{t-1}}{\partial \mathbf{x}_{t-2}}$
$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_{t-1}}$
$\frac{\partial \mathbf{x}_{t+1}}{\partial \mathbf{x}_t}$
$\frac{\partial \mathbf{x}_{t+2}}{\partial \mathbf{x}_{t+1}}$
u_{t-1}
u_t
u_{t+1}

The Problem of Long-term Dependencies

- RNNs connect previous information to present task:
  - may be enough for predicting the next word for ”the clouds are in the sky”



  - may not be enough when more context is needed: ”I grew up in France ... I speak fluent French”



Adapted from: *C. Olah*

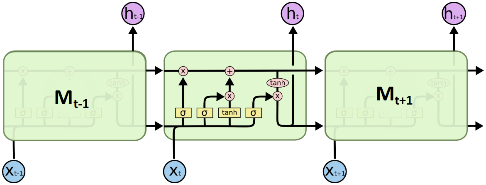
Vanishing/Exploding Gradients

- In RNNs, during the gradient back propagation phase, the gradient signal can end up being multiplied many times.
- If the gradients are large
    - Exploding gradients, learning diverges
    - Solution: clip the gradients to a certain max value.
- If the gradients are small
    - Vanishing gradients, learning very slow or stops
    - Solution: introducing memory via LSTM, GRU, etc.

Adapted from: *N. Cinbis*

Long Short-Term Memory Networks

- **Long Short-Term Memory (LSTM) networks** are RNNs capable of learning **long-term dependencies** [Hochreiter and Schmidhuber, 1997].



- A **memory** cell using logistic and linear units with multiplicative interactions:
  - Information **gets** into the cell whenever its **input gate** is on.
  - Information is **thrown away** from the cell whenever its **forget gate** is off.
  - Information can be **read** from the cell by turning on its **output gate**.

Adapted from: *C. Olah*

LSTM Overview

- We define the LSTM unit at each time step t to be a collection of vectors in $\mathbb{R}^d$:
  - **Memory cell** $\mathbf{c}_t$

    $\widetilde{\mathbf{c}}_t = \text{Tanh}(W_c.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c)$ **vector of new candidate values**

    $$\mathbf{c}_t = \mathbf{f}_t * \mathbf{c}_{t-1} + \mathbf{i}_t * \widetilde{\mathbf{c}}_t$$
  - **Forget gate** $\mathbf{f}_t$ in [0, 1]: scales old memory cell value (**reset**)

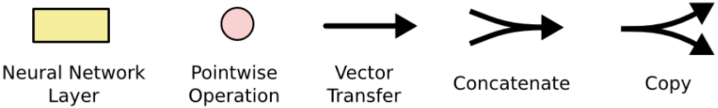
    $$\mathbf{f}_t = \sigma(W_f.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f)$$
  - **Input gate** $\mathbf{i}_t$ in [0, 1]: scales input to memory cell (**write**)

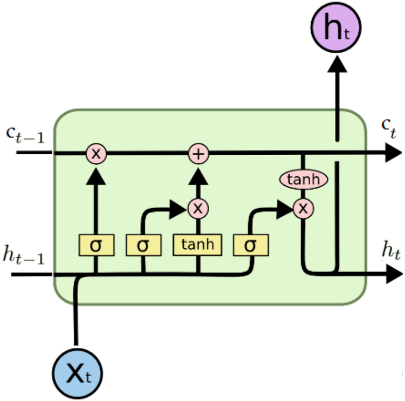
    $$\mathbf{i}_t = \sigma(W_i.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i)$$
  - **Output gate** $\mathbf{o}_t$ in [0, 1]: scales output from memory cell (**read**)

    $$\mathbf{o}_t = \sigma(W_o.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o)$$
  - **Output** $\mathbf{h}_t$

    $$\mathbf{h}_t = \mathbf{o}_t * \text{Tanh}(\mathbf{c}_t)$$

Notation

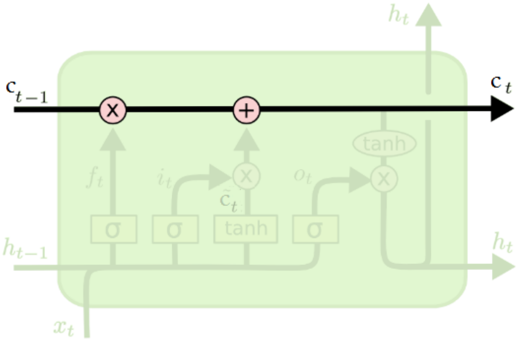




Adapted from: *C. Olah*

The Core Idea Behind LSTMs: Cell State (Memory Cell)

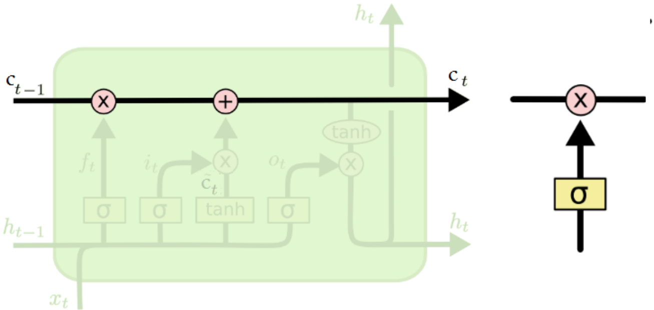
- Information can flow along the **memory cell unchanged**.
- Information can be **removed** or **written** to the **memory cell**, regulated by gates.



Adapted from: *C. Olah*

Gates

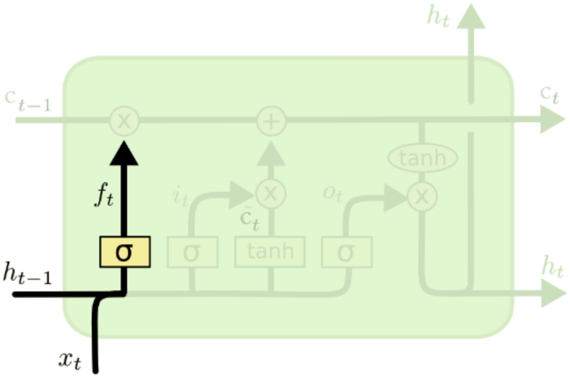
- **Gates** are a way to optionally let information through.
  - A **sigmoid layer** outputs number between 0 and 1, **deciding** how much of each component should be let through.
  - A pointwise multiplication operation applies the decision.



Adapted from: *C. Olah*

Forget Gate

- A **sigmoid** layer, **forget gate**, **decides** which values of the **memory cell** to **reset**.

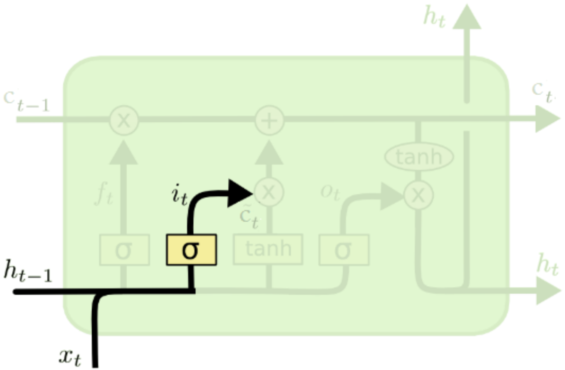


$$\mathbf{f}_t = \sigma(W_f.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f)$$

Adapted from: *C. Olah*

Input Gate

- A **sigmoid** layer, **input gate**, **decides** which values of the **memory cell** to **write** to.

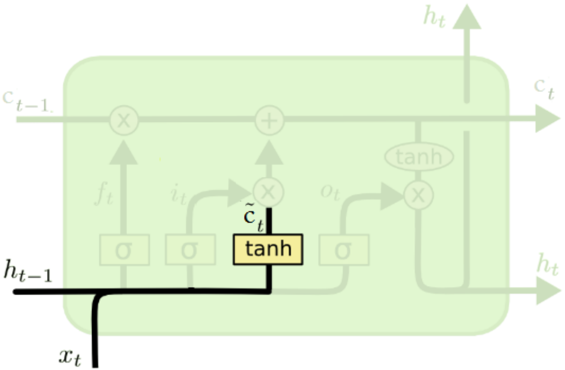


$$\mathbf{i}_t = \sigma(W_i.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i)$$

Adapted from: *C. Olah*

Vector of New Candidate Values

- A **Tanh** layer creates a **vector of new candidate values** $\widetilde{c}_t$ to **write** to the **memory cell**.

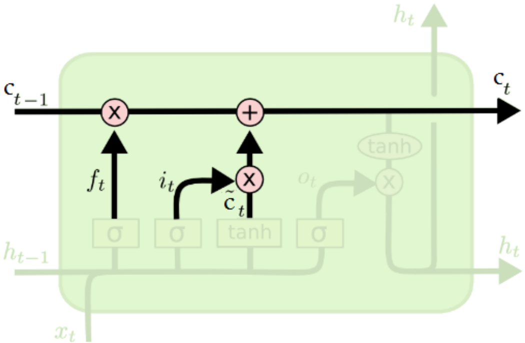


$$\widetilde{\mathbf{c}}_t = \text{Tanh}(W_c.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c)$$

Adapted from: *C. Olah*

Memory Cell Update

- The previous steps decided which values of the **memory cell** to **reset** and **overwrite**.
- Now the LSTM **applies the decisions** to the **memory cell**.

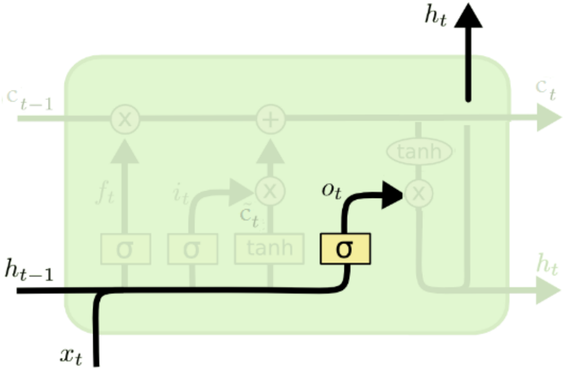


$$\mathbf{c}_t = \mathbf{f}_t * \mathbf{c}_{t-1} + \mathbf{i}_t * \widetilde{\mathbf{c}}_t$$

Adapted from: *C. Olah*

Output Gate

- A **sigmoid** layer, **output gate**, **decides** which values of the **memory cell** to **output**.

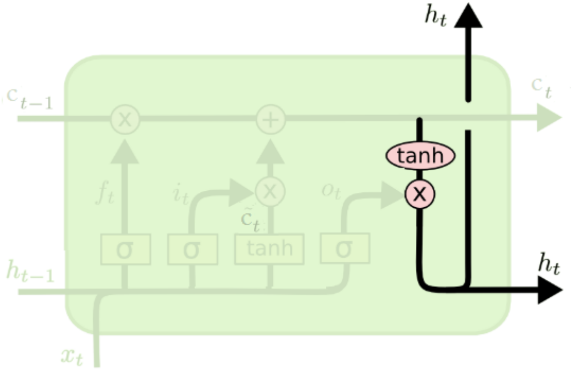


$$\mathbf{o}_t = \sigma(W_o.[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o)$$

Adapted from: *C. Olah*

Output Update

- The **memory cell** goes through **Tanh** and is multiplied by the **output gate**.



$$\mathbf{h}_t = \mathbf{o}_t * \text{Tanh}(\mathbf{c}_t)$$

Adapted from: *C. Olah*

Variants on LSTM

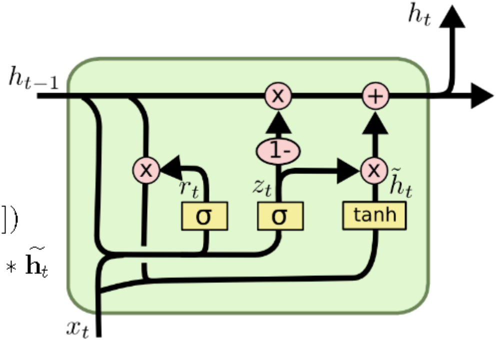
- Gated Recurrent Unit (GRU) [Cho et al., 2014]:
  - Combine the **forget** and **input** gates into a single **update** gate.
  - **Merge the memory cell and the hidden state**.
  - ...

$$z_t = \sigma(W_z.[\mathbf{h}_{t-1}, \mathbf{x}_t])$$

$$r_t = \sigma(W_r.[\mathbf{h}_{t-1}, \mathbf{x}_t])$$

$$\widetilde{\mathbf{h}_t} = \text{Tanh}(W.[r_t * \mathbf{h}_{t-1}, \mathbf{x}_t])$$

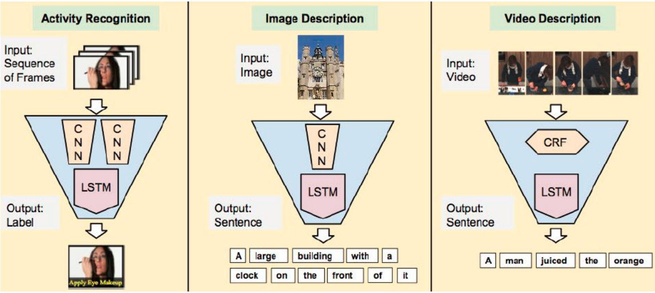
$$\mathbf{h}_t = (1 - z_t) * \mathbf{h}_{t-1} + (z_t) * \widetilde{\mathbf{h}_t}$$



Adapted from: *C. Olah*

Applications

- Cursive handwriting recognition
  - https://www.youtube.com/watch?v=mLxsbWAYIpw
- Translation
  - Translate any signal to another signal, e.g., translate English to French, translate image to image caption, and songs to lyrics.
- Visual sequence tasks



Adapted from: *Jeff Donahue et al. CVPR15*