Neural Networks

Eduard I. STAN
stan.ionelеduard@spes.uniud.it

Department of Mathematics, Computer Science, and Physics
University of Udine

November 21, 2018

Introduction

Simple traditional neural networks work fine when the problem is **linearly separable**.

If we deal with more complex problems such as the **logical XOR** (which is not linearly separable), we need to add **more layers**.

Deep Learning (DL) learns high level features from low level feature previously learned.

Data preparation and classification interleaved.

Backpropagation

Backpropagation

Calculus on Computational Graphs

Backpropagation is the **key algorithm** for the learning process that makes training deep models computationally tractable.

Fundamentally, it's a technique for calculating **derivatives** quickly.

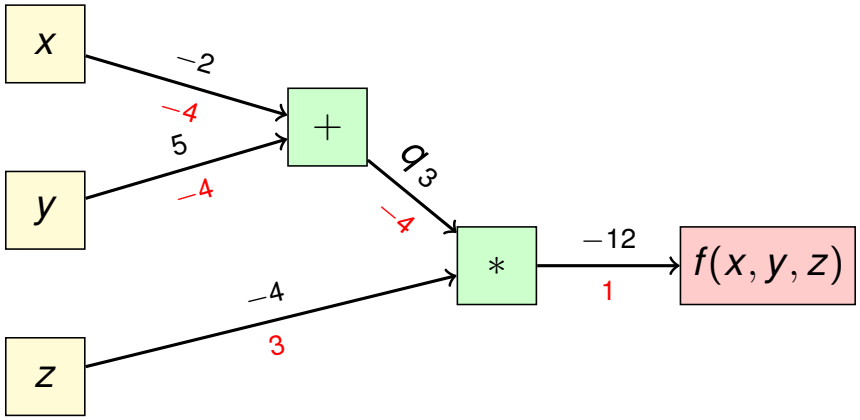
Computational graphs are a nice way to think about mathematical expressions.

We can evaluate the derivatives on the computational graph.

Computational Graph

Derivatives

If we want to represent $f(x, y, z) = (x + y)z$, the resulting graph is:



To evaluate the partial derivatives on this graph we need the sum rule and the product rule:

- $\frac{\partial}{\partial a}(a+b) = \frac{\partial a}{\partial a} + \frac{\partial b}{\partial a} = 1$
- $\frac{\partial}{\partial u}uv = u\frac{\partial v}{\partial u} + v\frac{\partial u}{\partial u} = v$

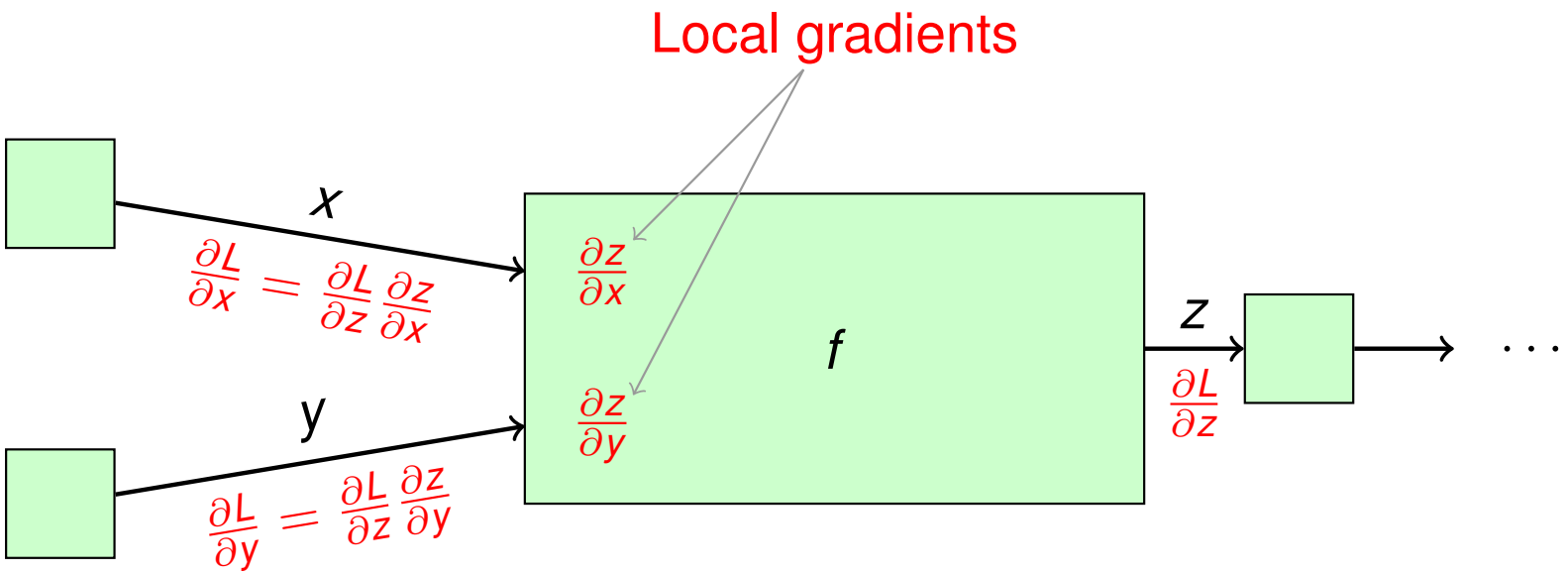
The key idea is to apply the **chain rule**

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x},$$

obtaining $\frac{\partial f}{\partial f} = 1$, $\frac{\partial f}{\partial z} = 3$, $\frac{\partial f}{\partial q} = -4$, $\frac{\partial f}{\partial x} = -4$, $\frac{\partial f}{\partial x} = -4$.

Computational Graph

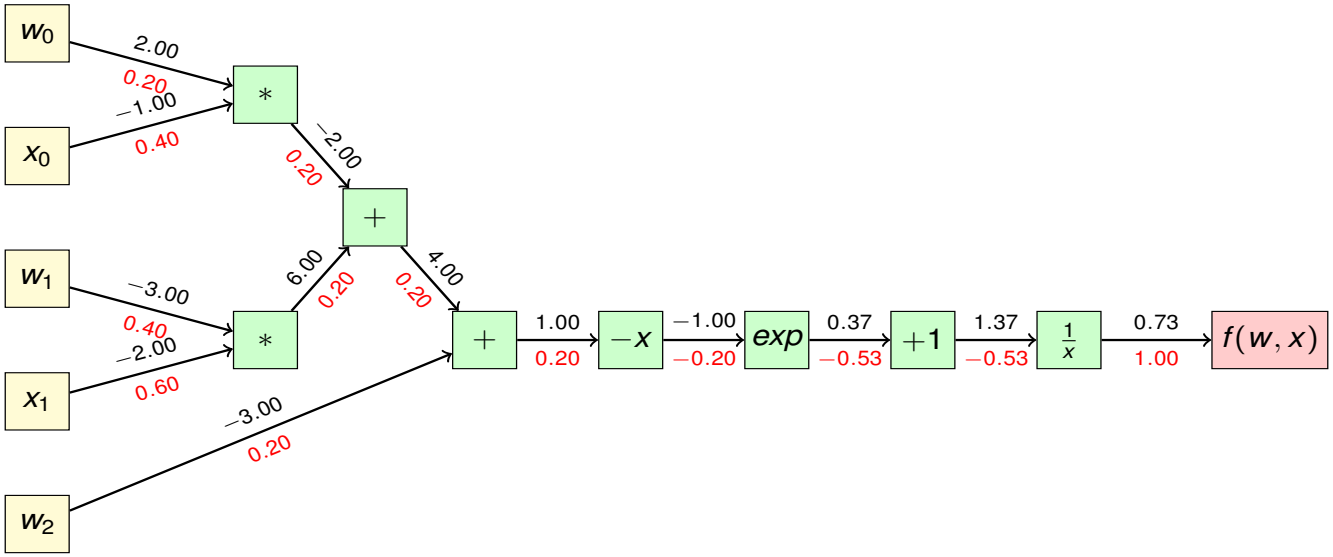
Single gate



Computational Graph

Another example

Let $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$, the resulting graph is

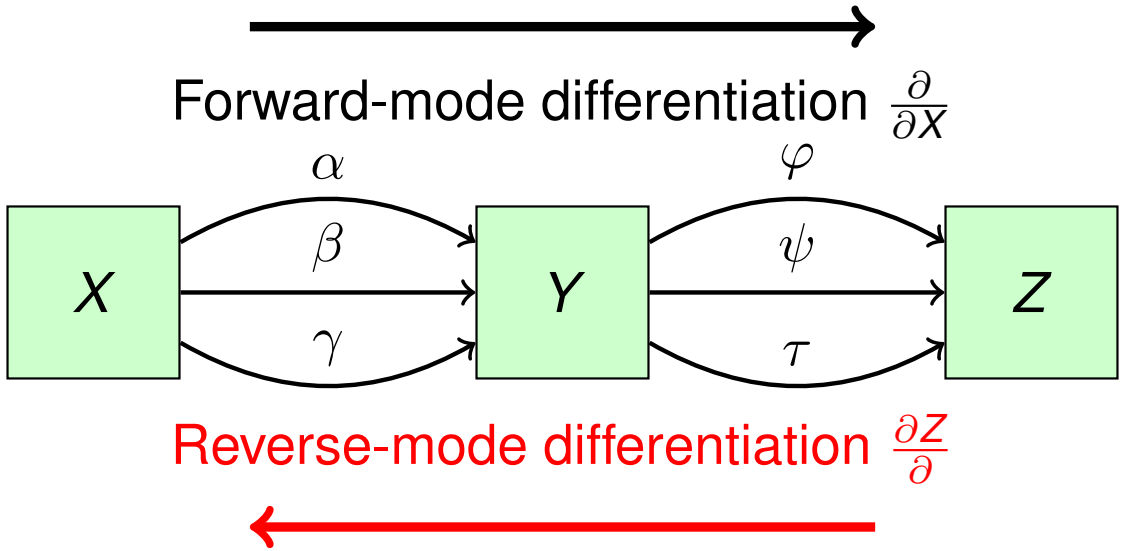


$$f(x) = e^x \longrightarrow \quad \frac{\partial f}{\partial x} = e^x \qquad f_a(x) = ax \longrightarrow \quad \frac{\partial f}{\partial x} = a$$

$$f(x) = \frac{1}{x} \longrightarrow \quad \frac{\partial f}{\partial x} = -\frac{1}{x^2} \qquad f_c(x) = c + x \longrightarrow \quad \frac{\partial f}{\partial x} = 1$$

Backpropagation

Forward-mode vs. reverse-mode differentiation

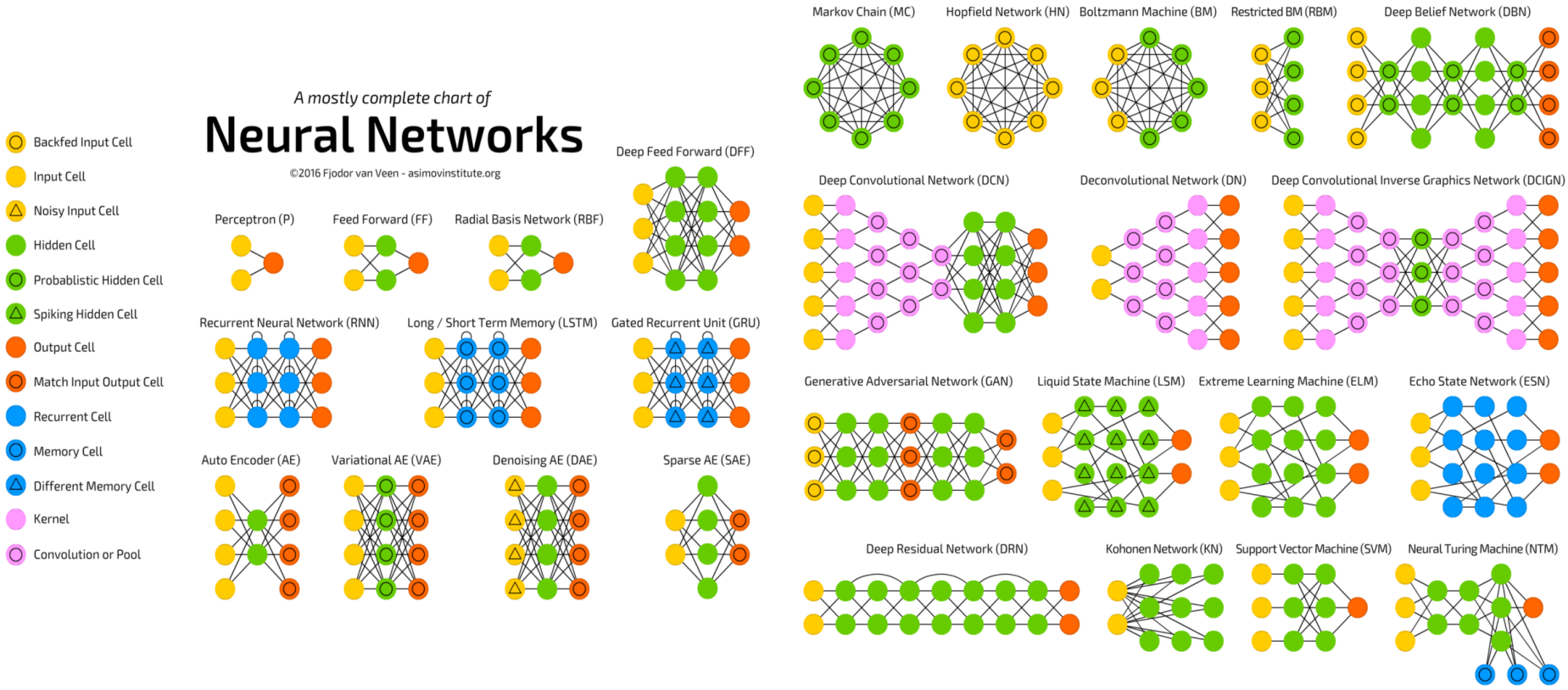


When training neural networks, we think at the cost as a function of the parameters (numbers describing how the network behaves). We want to calculate the derivatives of the cost w.r.t all the parameters in the net. So, backpropagation gives us a massive speed up!

Neural Networks

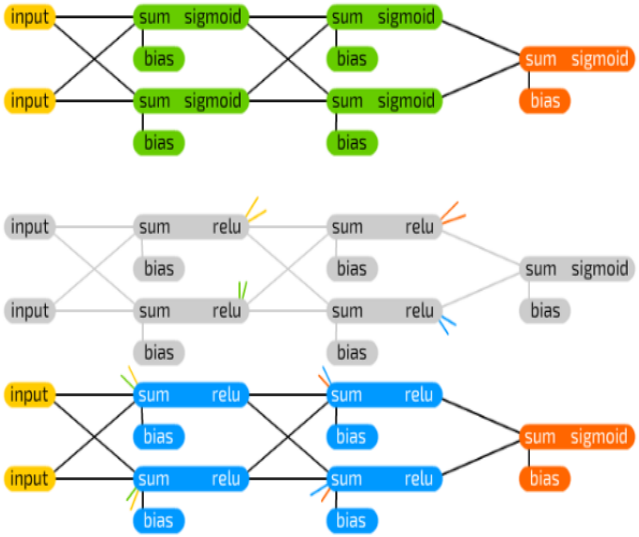
Neural Networks

The Zoo

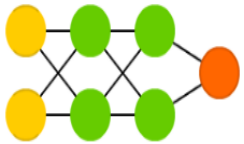


Neural Networks

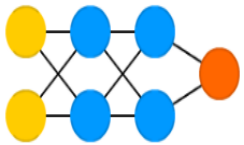
Graphs: Feed Forward - RNN



Deep Feed Forward Example



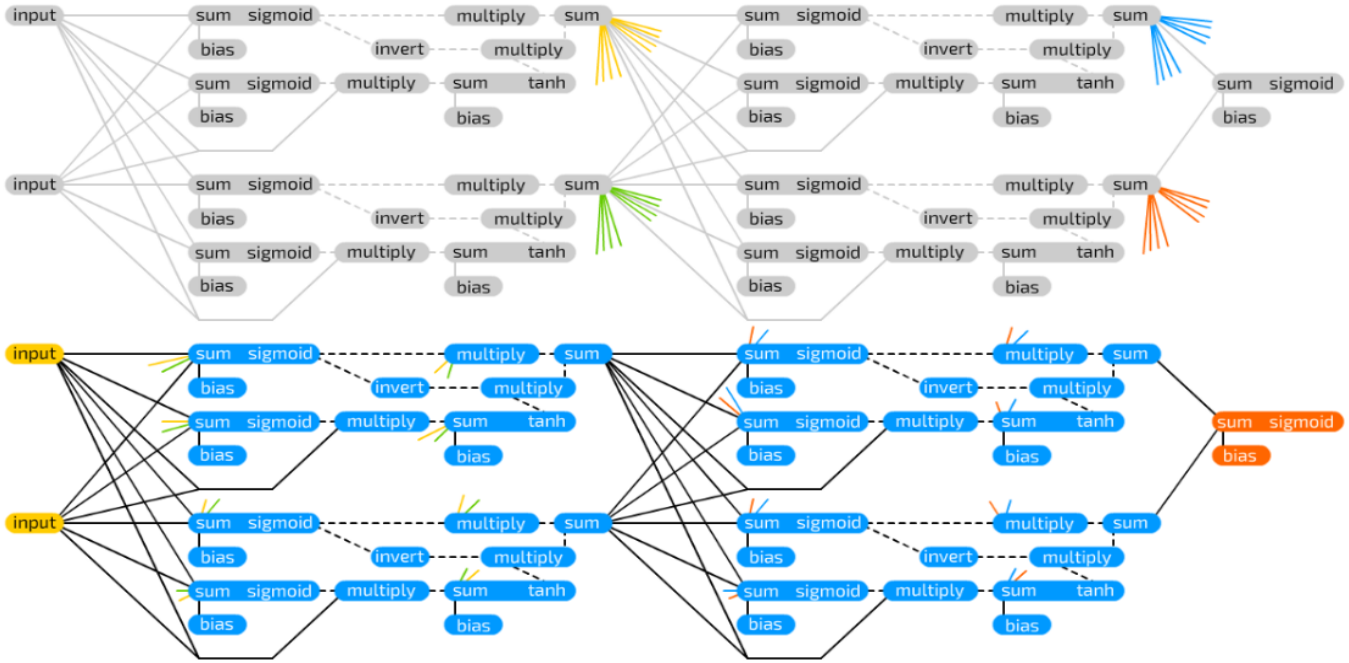
Deep Recurrent Example
(previous iteration)



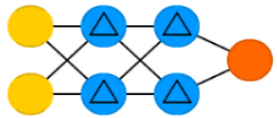
Deep Recurrent Example

Neural Networks

Graphs: Gated Recurrent Unit (GRU)



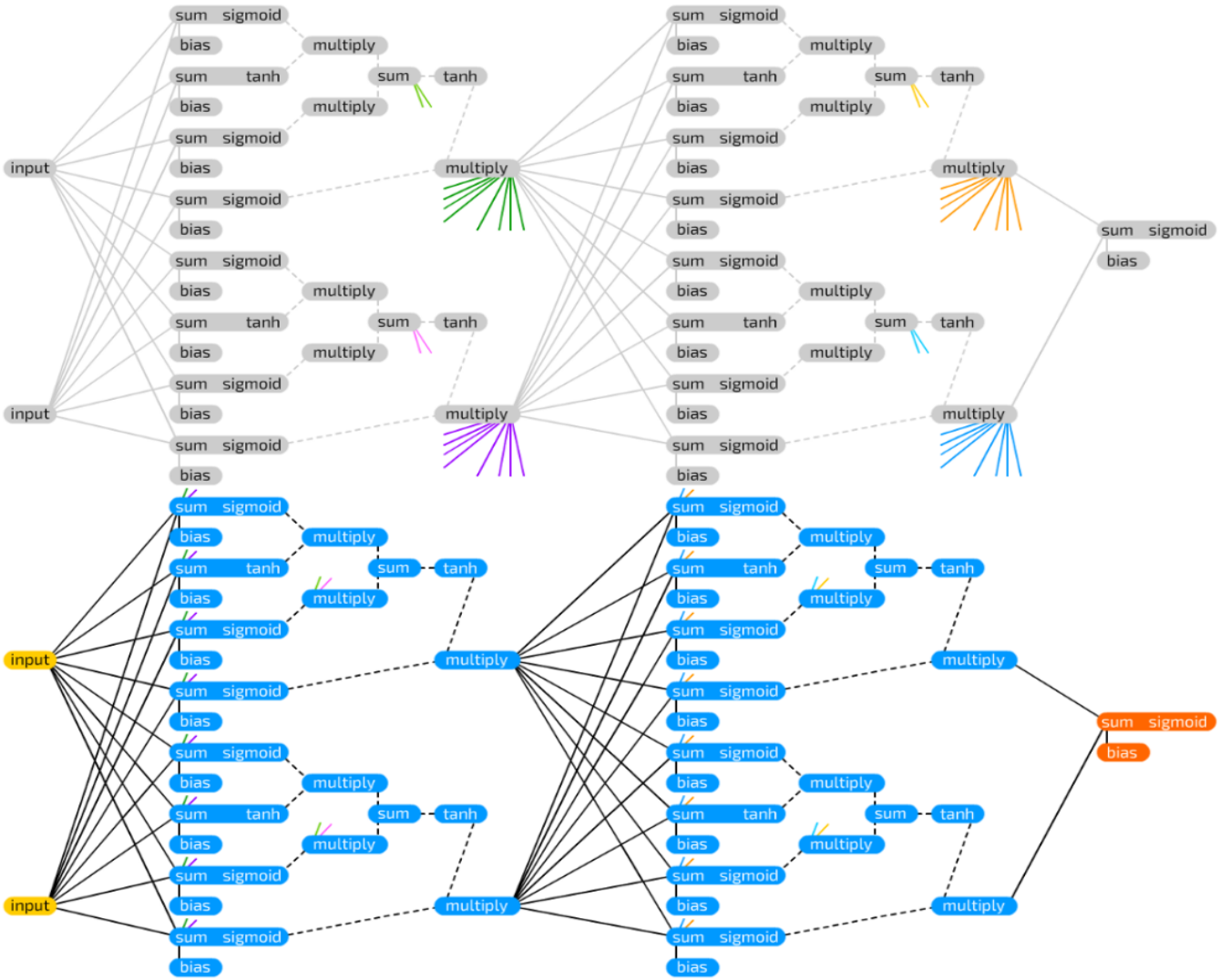
Deep GRU Example
(previous iteration)



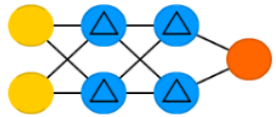
Deep GRU Example

Neural Networks

Graphs: Long Short Term Memory (LSTM)



Deep LSTM Example
(previous iteration)



Deep LSTM Example

Convolutional Neural Networks

Convolutional Neural Networks (CNNs)

Introduction

Specialized kind of neural networks for processing data that has a known **grid-like topology** (e.g., 1D grid for time series data, 2D grid for an image data).

Convolution networks are simply neural networks that use (a mathematical operation called) convolution in place of general matrix multiplication in at least one of their layers.

CNNs

Convolution

The operation

$$s(t) = \int x(a)w(t-a)da$$

is called the **convolution**. Typically denoted with an asterisk as

$$s(t) = (x * w)(t),$$

where x is often referred to as the **input**, w as the **kernel**, and the output is sometimes referred to as the **feature map**.

When we work with data on a computer, time will be discretized.

If we now assume that x and w are defined only on integer t, we can define the discrete convolution as:

$$s(t) = (x * w)(t) = \sum x(a)w(t-a).$$

CNNs

Motivation

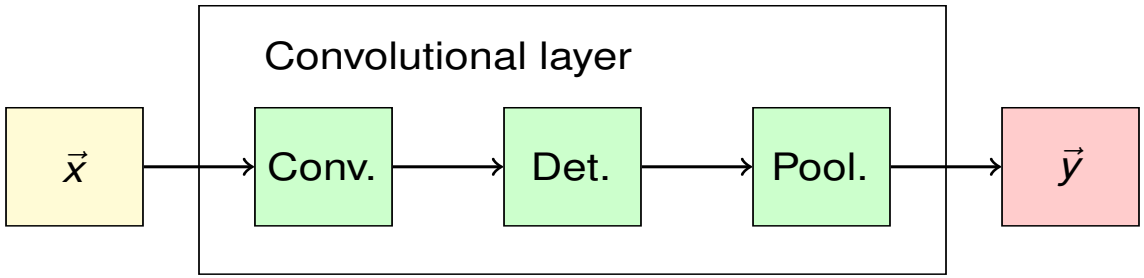
Convolution leverages on at least two important ideas that can help improve a machine learning system:

- **Sparse interaction.** Every output unit may not interact with every input unit. We need to store fewer parameters, from $\mathcal{M}^{m \times n}$ parameters to $\mathcal{M}^{k \times n}$ parameters, where k is the limit on the number of connections each output may have (i.e. **sparse weights vs full matrix multiplication**).
- **Parameter sharing.** Using the same parameter for more than one function model. Each element of the weight matrix is used exactly once when computing the output of a layer. One can say that the network has **tied weights**, because the value of the weight applied to one input is tied to the value of a weight applied elsewhere.

CNNs

Pooling

A typical layer of a CNN consists of three stages:



In the second stage, each presynaptic activation is run through a nonlinear activation function (e.g., rectified linear activation function). This stage is also called the **Detector** stage.

In the third stage, we use a **Pooling function** to modify the output of the layer further. This function replaces the output of the net at a certain location with a summary statistic of the nearby outputs (e.g., **maxpooling function**).

Generative Adversarial Networks

Generative Adversarial Networks (GANs)

Introduction

Two models play two distinct (literally, **adversarial**) roles.

The goal is to generate data that **resemble the data that was in the training set**; performing a form of **unsupervised learning** on the original dataset.

An **adversarial training** is defined by means of the worst case input for a player is produced by the other (i.e. trained to do the best on the worst-case scenario).

GANs

Generative Modeling

We may want two things from a generative model:

1. **Density estimation.** Given a large collection of examples, we want to find the probability of the density function that describes the examples.
2. **Sample generation.** Try to learn a function or a program that can generate more samples from the training distribution.

GANs

Framework

We have two agents playing a game each other:

1. **Generator.** Tries to generate data.
2. **Discriminator**. Examines data and estimates whether is real or fake.

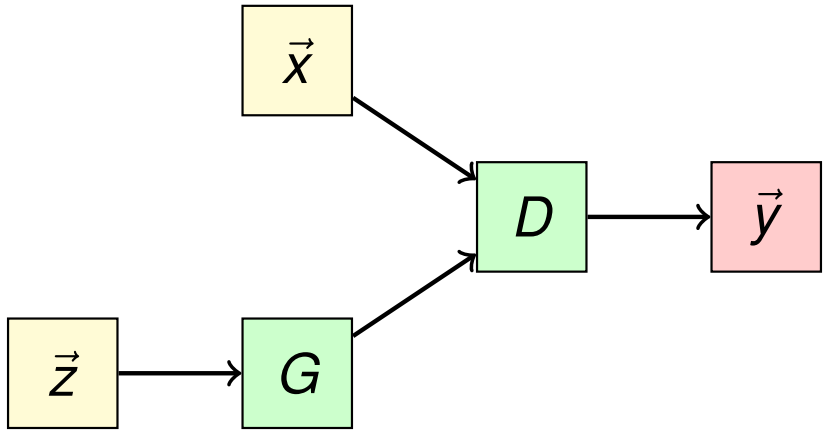
The goal of the Generator is to fool Discriminator and, as both players get better and better at the time, eventually the Generator is forced to generate data as realistic as possible.

GANs

Architecture

Let $\vec{x}$ be an input from the sample data and $\vec{z}$ a noise input vector[1]:

- The network for the Discriminator is a differentiable function D (i.e. a neural network) that is applied to $\vec{x}$ and the goal is to have $D(\vec{x})$ near to 1 (i.e. $\vec{x}$ is a real example).
- The network for the Generator is a differentiable function G that is applied to $\vec{z}$ (sampled from the model). We apply the D to G and in this case:
  - Discriminator tries to make $D(G(\vec{z}))$ near 0 (i.e. the input is fake), and
  - Generator tries to make $D(G(\vec{z}))$ near to 1 (i.e. tries fool the Discriminator).



[1] It is a source of randomness that allows the Generator to output many different images instead of one realistic image

Recurrent Neural Networks

Recurrent Neural Networks (RNNs)

Introduction

One of the early ideas found in machine learning and statistical models of the 80's, is that of **sharing parameters** across different parts of the model, allowing to **extend and to apply the model** to examples of different forms and generalize across them, e.g. with examples of different lengths, in the case of sequential data.

This can be found in Hidden Markov Models (HMMs) (Rabiner and Juang, 1986), which where the dominant technique for speech recognition for about 30 years. These models of sequences involve parameters such as the state-to-state transition matrix

$$P(\mathbf{s}_t \mid \mathbf{s}_{t-1}),$$

which are reused for each time step t, i.e. depends only on $\mathbf{s}_t$ and $\mathbf{s}_{t-1}$ but not on t as such.

RNNs

Preliminaries

A RNN consists of a **hidden state**

$$\vec{h} := (h_1, \ldots, h_k)$$

and an optional **output**

$$\vec{y} := (y_1, \ldots, y_m)$$

which operates on a **variable-length sequence**

$$\vec{x} := (x_1, \ldots, x_n).$$

RNNS

Preliminaries

At each time step t, the hidden state and the output are

$$h_t = f_{\mathbf{W}}(h_{t-1}, x_t),$$
$$y_t = g_{\mathbf{W}}(h_t),$$

where $f_{\mathbf{W}}$ is an **activation function** (e.g., sigmoid, ReLU, tanh) and $g_{\mathbf{W}}$ is a **softmax function** (which produces valid probabilities), w.r.t. some parameters $\mathbf{W}$, also called **weights**.

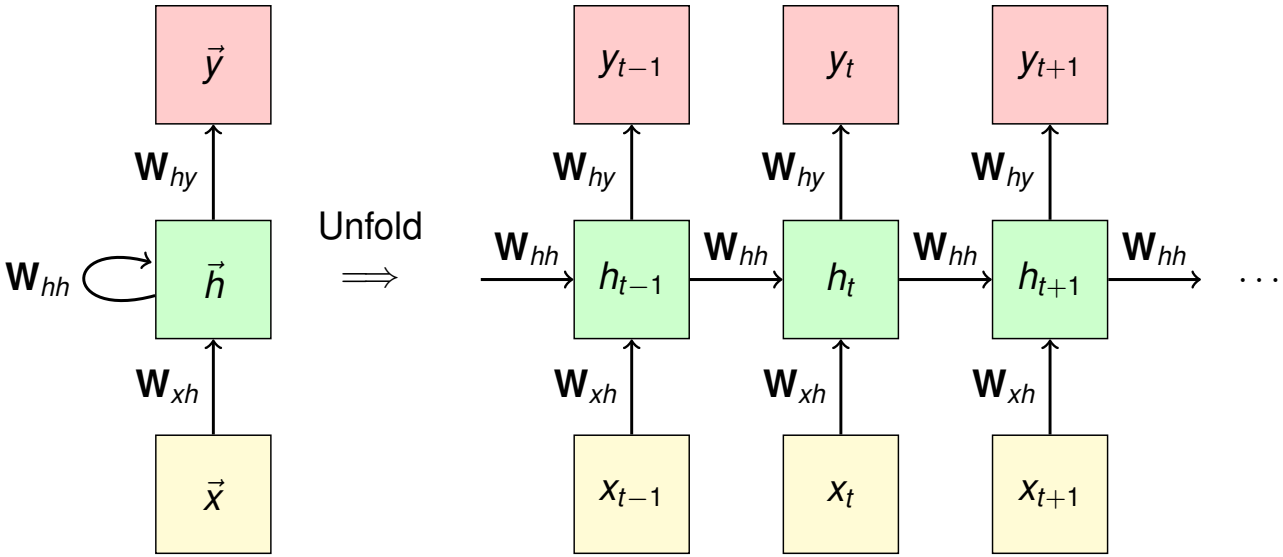
For each time step t, we obtain the following hidden state and output,

$$h_t = \tanh(\mathbf{W}_{hh} h_{t-1} + \mathbf{W}_{xh} x_t), \tag{1}$$

$$y_t = \operatorname{softmax}(\mathbf{W}_{hy} h_t) = \frac{e^{\mathbf{W}_{hy} h_t}}{\sum_{i=1}^{m} e^{\mathbf{W}_{hy} h_i}} \text{ for } t \in \{1, 2, \ldots, m\}. \tag{2}$$

RNNs

Architecture



Sequence to Sequence Deep Learning

Sequence to Sequence (Seq2Seq)

Introduction

Seq2Seq is an **end-to-end deep learning algorithm**.

It learns to **encode** a variable-length sequence $\vec{x} = (x_1, \ldots, x_n)$ into a fixed-length vector representation and to **decode** a given fixed-length vector representation into a variable-length sequence $\vec{y} = (y_1, \ldots, y_m)$.

From a probabilistic perspective, this new model is a general method to learn the conditional distribution over a variable-length sequence conditioned on yet another variable-length sequence, e.g.,

$$p(y_1, \ldots, y_m \mid x_1, \ldots, x_n),$$

where potentially $n \neq m$.

Seq2Seq

Encoder-Decoder

The **Encoder** is an RNN that reads each symbol of the input sequence $\vec{x}$ sequentially. As it reads each symbol x_t, the hidden state of the RNN changes accordingly to

$$h_t = \tanh(\mathbf{W}_{hh} h_{t-1} + \mathbf{W}_{xh} x_t).$$

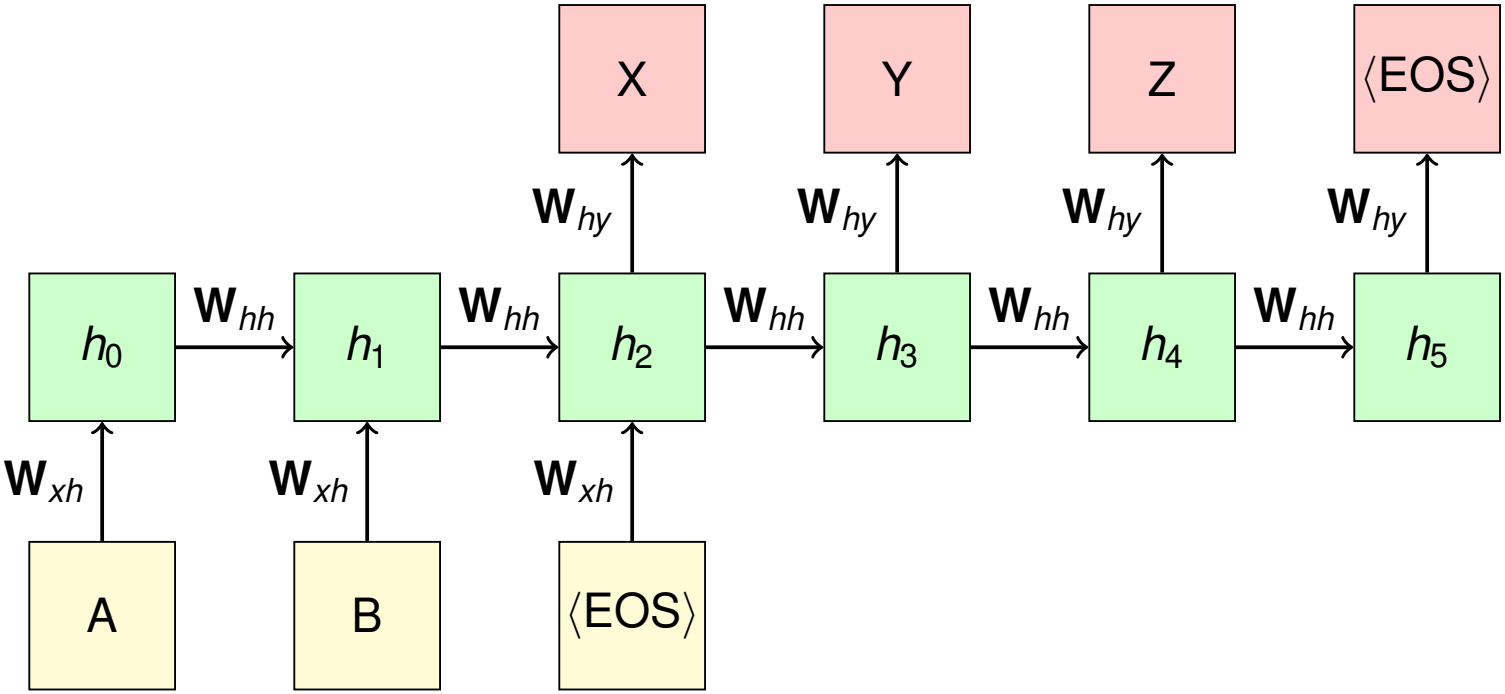
After reading the end of the sequence (marked by the **end-of-sequence** symbol), the hidden state of the RNN is a summary $\vec{s}$ of the all input sequence.

The **Decoder** is another RNN which is trained to **generate** the output sequence $\vec{y}$ by predicting the next symbol y_t given the hidden state h_t accordingly to

$$y_t = \text{softmax}(\mathbf{W}_{hy} h_t) = \frac{e^{\mathbf{W}_{hy} h_t}}{\sum_{i=1}^{m} e^{\mathbf{W}_{hy} h_i}} \text{ for } t \in \{1, 2, \ldots, m\}.$$

Seq2Seq

Example



Seq2Seq

Experiments

The dataset consists into translated pairs between Italian and English, and has 295357 sentence pairs.

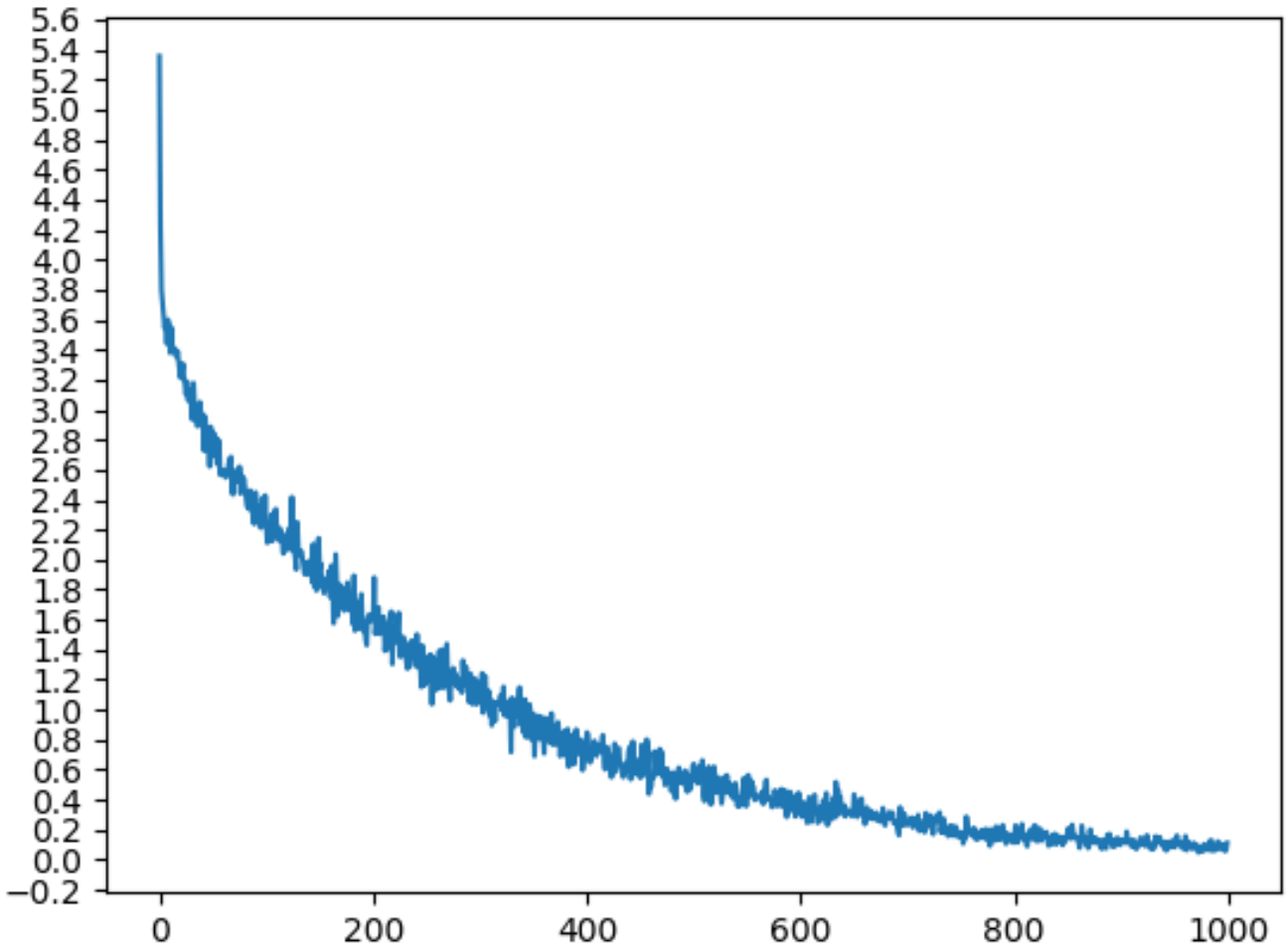
We tokenized the pairs and filtered them to obtain a reduced dataset with 5672 observations, where the vocabularies had 2114 and 1499 words, respectively for the Italian and the English language.

We used GRU layers for both Encoder and Decoder. Each hidden state (i.e., each layer) is compound of 128 single GRU units, and there are 5 of such hidden states.

Unfortunately, we used 1 GPU (the only available), occupying 487MB of memory, and, yet, even without no kind of parallelization, the training took 49 minutes and 27 seconds.

Seq2Seq

Results



Results

Wrongly Predicted Sentences

Some of the wrongly predicted sentences are:

1. 1. > lei e bella .
   2. = you are beautiful .
   3. < she is beautiful .
2. 1. > lei e una cantante famosa .
   2. = she is a famous singer .
   3. < she is a well singer singer .
3. 1. > sono riconoscente per le vacanze .
   2. = i am thankful for vacations .
   3. < i am thankful for the holidays .
4. 1. > resto per qualche altra settimana .
   2. = i am staying for another few weeks .
   3. < i am going for the few days .
5. 1. > non e giovane .
   2. = he is not young .
   3. < he aren t young .

References


Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.


Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Montreal, Canada: MIT Press, 2014, pp. 3104–3112.


Oriol Vinyals and Quoc V. Le. "A Neural Conversational Model". In: *CoRR* abs/1506.05869 (2015).


Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *CoRR* abs/1406.1078 (2014).


Andrej Karpathy. *The unreasonable effectiveness of Recurrent Neural Networks*. May 2015. URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness/.


Andrej Karpathy. *Convolutional Neural Networks for Visual Recognition*. Stanford Course. 2016.


Fjodor van Veen. *The Neural Network Zoo*. Mar. 2017. URL: http://www.asimovinstitute.org/neural-network-zoo/.


Christopher Olah. *Calculus on Computational Graphs: Backpropagation*. Aug. 2015. URL: http://colah.github.io/posts/2015-08-Backprop/.