Deep Learning

Lecture 8: Deep Learning Architectures (Part I)

University of Agder,
Kristiansand, Norway

Prepared by: Hadi Ghauch*, Hossein S. Ghadikolaei†

* Telecom ParisTech

† Royal Institute of Technology, KTH

https://sites.google.com/view/fundl/home

April 2019

Outline

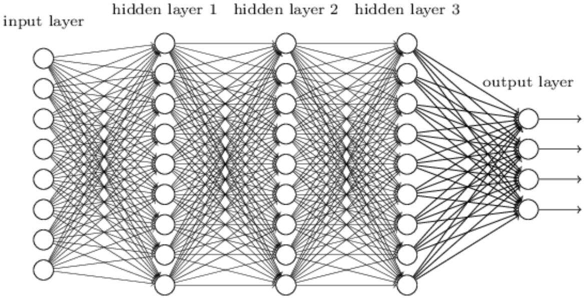
1. Recurrent Neural Networks

2. Long Short Term Memory Networks

Recap: Deep Neural Networks (DNNs)

Multi-layer Percepteron:

- Output from some perceptrons are used in the inputs to other perceptrons



$\boldsymbol{W}_1$ $\boldsymbol{W}_2$ $\boldsymbol{W}_3$ Output

Feedforward Deep Neural Network (DNN):

- Multi-layer Percepteron with 'many' hidden layers
- Only forward connections: from input to hidden layer or output

Recap: Deep Neural Networks (DNNs)

DNN with J-layers: cascade of J filters + activation

$$\boldsymbol{z}_0 \to \underbrace{\boxed{\boxed{\boldsymbol{W}_1, \boldsymbol{b}_1} \to \boxed{\boldsymbol{\sigma}_1}}}_{\text{Layer 1}} \to \boldsymbol{z}_1 \to \underbrace{\boxed{\boxed{\boldsymbol{W}_2, \boldsymbol{b}_2} \to \boxed{\boldsymbol{\sigma}_2}}}_{\text{Layer 2}} \to \boldsymbol{z}_2 \to \cdots \to \boldsymbol{z}_J$$

In compact form: $\boldsymbol{z}_j = \boldsymbol{\sigma}_j(\boldsymbol{W}_j \boldsymbol{z}_{j-1} + \boldsymbol{b}_j)$

$\boldsymbol{z}_j \in \mathbb{R}^{d_j}$: output to layer j

$\boldsymbol{W}_j \in \mathbb{R}^{d_j \times d_{j-1}}$: **Weights** for layer j (drop bias for simplicity)

$\boldsymbol{\sigma}_j : \mathbb{R}^{d_j} \mapsto \mathbb{R}^{d_j}$: **Activation func** for layer j

Outline

1. Recurrent Neural Networks

2. Long Short Term Memory Networks

Motivation and Setting

Feedforward NNs assume input/output have **fixed dimension**:
e.g. image classification, regression

assume **samples of training set are i.i.d.**

these assumptions restrictive when dealing with sequential data

Feedforward NN ill-equipped when **data is sequential**: temporal/causal dependence in data: e.g., classifying frames of a movie, or words in sentence

knowing previous samples critical to predict next sample

Recurrent NNs: family of NNs designed to learn sequential data

training set **time-dependent samples** (not i.i.d.): $\{ (\boldsymbol{x}_t, \boldsymbol{y}_t) \}_{t=1}^{T}$
sample $(\boldsymbol{x}_t, \boldsymbol{y}_t)$ depends on previous ones
classifying images/frames of a movie: time index t is frame number
consecutive frames correlated (i.i.d. assumption does not hold)
learning such relation critical to classify other frames

Intuition: Share parameters from different parts of model
enables learning sequences not seen in training and improve generalization

Recurrent NNs implement parameter sharing by design:
each output depends on previous outputs

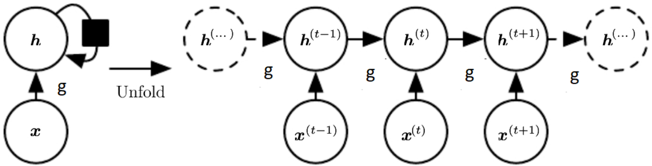
Recurrent NN

Motivation: **dynamical system**, $\boldsymbol{s}_t = g(\boldsymbol{s}_{t-1}, \boldsymbol{x}_t)$, $\boldsymbol{x}_t$ external input
states are **recursively generated by** $g()$
Goal: Use recurrent NN learn states of dynamical system
not suitable for feedforward NNs

Recurrent NNs: designed specifically to learn states in a recursive setting
similar to feed-forward NNs used to learn non-recursive func

several designs used. start with **plain recurrent NNs**: $\boldsymbol{h}_t = g(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$
set state the dynamical system equal to hidden unit, $\boldsymbol{h}_t$

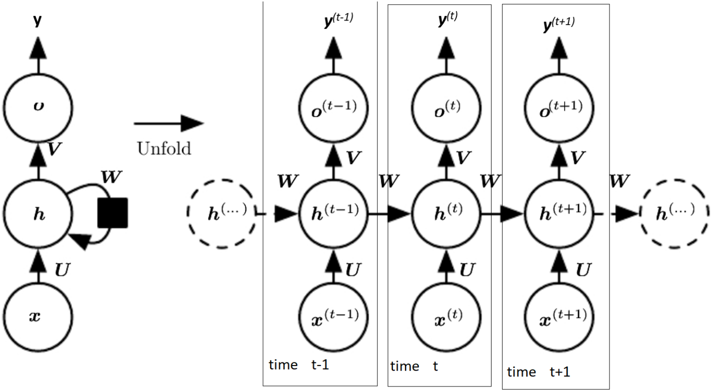
Model system by unfolding the states: **start with unfolded model**



Simple **toy architecture**. Next, move more realistic architecture.

Prevalent design Recurrent NNs

So far no learning: now lets add some **parameters to learn**



Single input single output at each time: at time t, one input-output pair $(\boldsymbol{x}_t, \boldsymbol{y}_t)$.
Goal: Learn a **same set parameters/weights**, $\boldsymbol{V}, \boldsymbol{W}, \boldsymbol{U}$, that approximates **all samples**, $\{\boldsymbol{x}_t, \boldsymbol{y}_t\}_{t=1}^{T}$
parameters/weights shared across all samples: notion of **parameter sharing**

Mathematical Model of Recurrent NNs

Model for sample t:

$\boldsymbol{x}_t \in \mathbb{R}^n$: Input vector at time t

$\boldsymbol{h}_t \in \mathbb{R}^n$: Hidden layer/unit at time t

Inputs for $\boldsymbol{h}_t$:

linear combination of prev hidden layer: $\boldsymbol{W}\boldsymbol{h}_t$, where $\boldsymbol{W} \in \mathbb{R}^{n\times n}$
linear combination of input: $\boldsymbol{U}\boldsymbol{x}_t$, where $\boldsymbol{U} \in \mathbb{R}^{n\times n}$
what happens in $\boldsymbol{h}_t$? sum them apply non-linear activation: $\psi(\boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{U}\boldsymbol{x}_t)$
$\boldsymbol{\psi} : \mathbb{R}^n \mapsto \mathbb{R}^n$: vector-valued vector func (applied element-wise)

Output for $\boldsymbol{h}_t$:

linearly combine output of $\boldsymbol{h}_t$: $\boldsymbol{V}\boldsymbol{h}_t$, where $\boldsymbol{V} \in \mathbb{R}^{n\times n}$
then apply non-linear activation: $\phi(\boldsymbol{V}\boldsymbol{h}_t)$
$\boldsymbol{\phi} : \mathbb{R}^n \mapsto \mathbb{R}^n$: vector-valued vector func (applied element-wise)

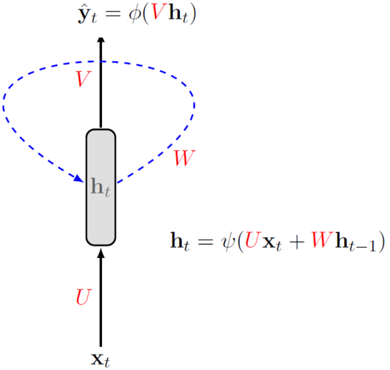
$\hat{\boldsymbol{y}}_t \in \mathbb{R}^n$: **predicted output vector** (by RNN) at time t

$\boldsymbol{y}_t \in \mathbb{R}^n$: **true output vector** at time t

$\{\boldsymbol{x}_t, \boldsymbol{y}_t\}_{t=1}^{T}$: training set of T samples (recall the time index t)
assume all dimensions are same $(= n)$ for simplicity

Goal: Learn RNN parameters, $\boldsymbol{W}, \boldsymbol{V}, \boldsymbol{U}$ to minimize the loss b/w predicted output by RNN, $\hat{\boldsymbol{y}}_t$, and true output, $\boldsymbol{y}_t$

Plain Recurrent NNs (folded form)



snapshot of input-output at time t
Source: Trivedi, Kodor, 2017

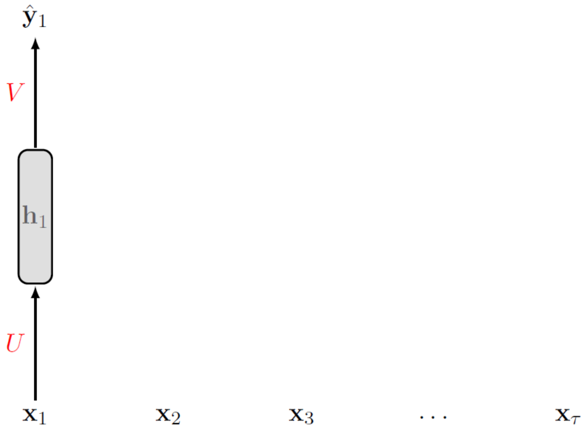
Plain Recurrent NNs (folded form):

$\phi()$, $\psi()$: vector-valued vector function (non-linear activation)

$\boldsymbol{U}$, $\boldsymbol{W}$, $\boldsymbol{V}$: weight matrices of appropriate size

Plain Recurrent NNs (unfolded form)

Previous Recurrent NN in unfolded form

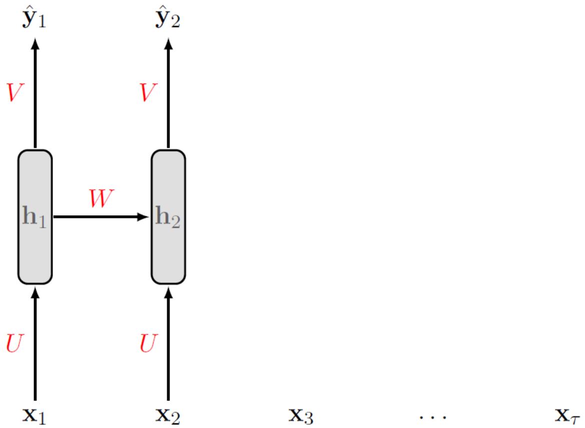


Source: Trivedi, Kodor, 2017

snapshot of input-output relation at time $t = 1$

Plain Recurrent NNs (unfolded form)

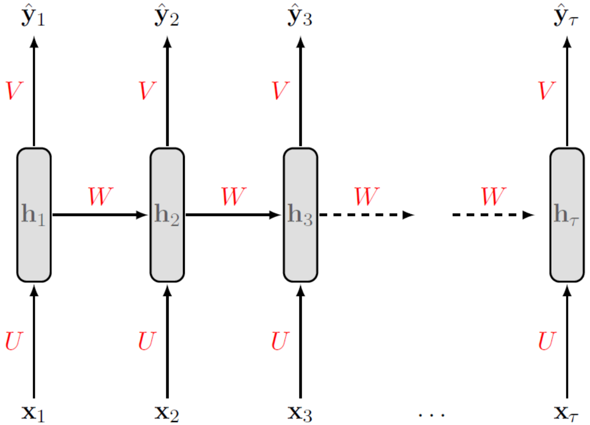
Previous Recurrent NN in unfolded form



Source: Trivedi, Kodor, 2017

snapshot of input-output relation at time $t = 2$

Plain Recurrent NNs (unfolded form)



Parameters (weights) the same or all time samples (weights indep of t):
learn set of **parameters (weights) shared for all time samples**

By learning $\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{V}$ **uncover time-dependence/correlation** among **time-dependent samples** of training set

Training Recurrent NN

Express mathematically input-output relation at time t:

$\boldsymbol{a}_t = \boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{U}\boldsymbol{x}_t + b$:
linearly combine $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$

$\boldsymbol{h}_t = \boldsymbol{\psi}(\boldsymbol{a}_t)$:
apply non-linearity at input of hidden layer

$\boldsymbol{o}_t = \boldsymbol{V}\boldsymbol{h}_t + c$:
linearly combine output of hidden layer

$\hat{\boldsymbol{y}}_t = \boldsymbol{\phi}(\boldsymbol{o}_t)$:
apply non-linearity just before output

$\hat{\boldsymbol{y}}_t$ is prediction that RNN outputs for sample $\boldsymbol{x}_t$ (drop b, c wlog):

$$\hat{\boldsymbol{y}}_t := \boldsymbol{\phi}[\ \boldsymbol{V}\boldsymbol{\psi}(\boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{U}\boldsymbol{x}_t)\ ]$$

Training Recurrent NN

Training Recurrent NNs:

Loss for sample t: $f_t(\boldsymbol{y}_t, \hat{\boldsymbol{y}}_t)$ (diff b/w true and predicted value)
  negative log-likelihood loss is a common choice
  total loss function, f is sum of individual losses: $f := \sum_t f_t$

Empirical risk minimization:

$$\underset{\boldsymbol{U},\boldsymbol{W},\boldsymbol{V}}{\arg\min} \sum_{t=1}^{T} f_t\left(\underbrace{\boldsymbol{y}_t}_{\text{true}}, \underbrace{\boldsymbol{\phi}[\ \boldsymbol{V}\boldsymbol{\psi}(\boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{U}\boldsymbol{x}_t)\ ]}_{\text{RNN prediction}}\right) := f \tag{1}$$

How to optimize $\boldsymbol{U}, \boldsymbol{W}, \boldsymbol{V}$?

Compute $\nabla_{\boldsymbol{W}}\ f$, $\nabla_{\boldsymbol{U}}\ f$, $\nabla_{\boldsymbol{V}}\ f$ using same 'tricks' as backpropagation
  called **backpropagation trough time (BPTT**) for RNN training

BPTT

1. gradient of $\boldsymbol{V}$:

$$\nabla_{\boldsymbol{V}} f = \sum_t (\nabla_{\boldsymbol{o}_t} f)(\boldsymbol{h}_t)^T$$

2. gradient of $\boldsymbol{W}$:

$$\nabla_{\boldsymbol{W}} f = \sum_t [\ \text{diag}(\mathbf{1} - \boldsymbol{h}_t \circ \boldsymbol{h}_t)\ (\nabla_{\boldsymbol{h}_t} f)\ (\boldsymbol{h}_{t-1})^T\ ]$$

$$(\nabla_{\boldsymbol{h}_t} f) = \begin{cases} (\boldsymbol{V})^T (\nabla_{\boldsymbol{o}_t} f), & \text{for } t = T \\ (\boldsymbol{W})^T\ \text{diag}(\mathbf{1} - \boldsymbol{h}_{t+1} \circ \boldsymbol{h}_{t+1})(\nabla_{\boldsymbol{h}_{t+1}} f) + (\boldsymbol{V})^T (\nabla_{\boldsymbol{o}_t} f), & \text{for } t < T \end{cases}$$

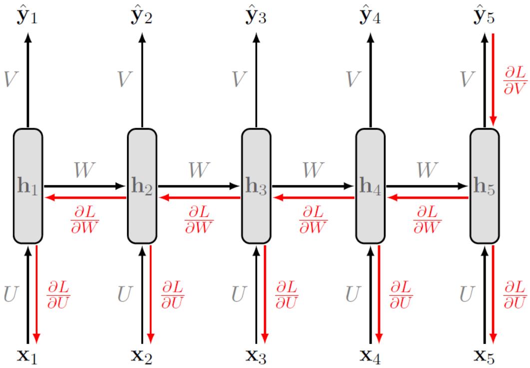
$(\nabla_{\boldsymbol{h}_t} f)$ depends on $(\nabla_{\boldsymbol{h}_{t+1}} f)$:
start hidden layers with largest index, $\boldsymbol{h}_T$, and work backwards (like BackProp)

3. gradient of $\boldsymbol{U}$:

$$\nabla_{\boldsymbol{U}} f = \sum_t [\ \text{diag}(\mathbf{1} - \boldsymbol{h}_t \circ \boldsymbol{h}_t)(\nabla_{\boldsymbol{h}_t} f)(\boldsymbol{x}_t)^T$$

see [Goodfellow, 2016], Chap 10, for detailed derivations of BPTT.

BPTT



Source: Trivedi, Kodor, 2017

BPTT= BP in the unrolled RNN

sample $(\boldsymbol{x}_5, \hat{\boldsymbol{y}}_5)$: compute $\nabla_{\boldsymbol{V}}\ f$, then $\nabla_{\boldsymbol{W}}\ f$, then $\nabla_{\boldsymbol{U}}\ f$
sample $(\boldsymbol{x}_4, \hat{\boldsymbol{y}}_4)$: compute $\nabla_{\boldsymbol{V}}\ f$, then $\nabla_{\boldsymbol{W}}\ f$, then $\nabla_{\boldsymbol{U}}\ f$
sample $(\boldsymbol{x}_3, \hat{\boldsymbol{y}}_3)$: compute $\nabla_{\boldsymbol{V}}\ f$, then $\nabla_{\boldsymbol{W}}\ f$, then $\nabla_{\boldsymbol{U}}\ f$

Some variants on Plain Recurrent NN:

- many inputs ($\boldsymbol{x}_t$), single output ($\boldsymbol{y}_t$). Used in sentiment analysis
- **bidirectional Recurrent NNs**: plain RNNs capture time-dependence in chronological order: $\boldsymbol{h}_1, ..., \boldsymbol{h}_t$. Bidirectional RNNs also capable of capturing reverse chronological order: $\boldsymbol{g}_t, ..., \boldsymbol{g}_1$.
  widely applied in handwriting and speech recognition, bioinformatics
- **Encoder-decoder Recurrent NN:** designed to handle input and output sequences that have different lengths.
  widely used in language translation (e.g., one sentence may have different number of words in different languages)
- **Deep Recurrent NN:** more hidden layers between input and output
  depth increase approximation capacity (but make optimization harder)
- **Deep Recursive NN:** computational graph is a tree
  used for Natural Language processing and computer vision

Outline

1. Recurrent Neural Networks

2. Long Short Term Memory Networks

Practical Problems for Deep RNNs:

Challenges for learning long term dependencies:
many layers (matrix multiplications) needed to capture long-term dependence
i.e., **deep RNNs** needed

challenge for deep RNNs: output becomes too small after multiplication with many matrices

illustrate with a simple RNN (w/out nonlinearity), $\boldsymbol{W} \in \mathbb{R}^{n \times n}$

$$\boldsymbol{h}_t = \boldsymbol{W}\boldsymbol{h}_{t-1} \Rightarrow \boldsymbol{h}_T = \underbrace{\boldsymbol{W}\boldsymbol{W}...\boldsymbol{W}}_{\text{T times}}\boldsymbol{h}_1 = \boldsymbol{W}^T\boldsymbol{h}_1 \Rightarrow \|\boldsymbol{h}_T\|_2 \leq (\lambda_{\max}[\boldsymbol{W}])^T \ \|\boldsymbol{h}_1\|_2$$

if $\lambda_{\max}[\boldsymbol{W}] \ < 1 \Rightarrow \|\boldsymbol{h}_T\|_2 \to 0,\ T \to \infty$
output, $\boldsymbol{h}_T$, becomes too small, so does it gradient **vanishing gradient**

depends on spectral radius of $\boldsymbol{W}$

problem peculiar to RNNs (due to **loops** b/w input and output)

some fixes to vanishing gradient: skip connections, leaky hidden units, echo state networks; see [Goodfellow, 2016], Chap 10
Exploding gradient: $\lambda_{\min}[\boldsymbol{W}] \ > 1 \Rightarrow \|\boldsymbol{h}_T\|_2 \to \infty,\ T \to \infty$
output, $\boldsymbol{h}_T$, blows up: so does the gradient of $\boldsymbol{h}_T$ (rarely occurs)

Long Short Term Memory (LSTM):

Long Short Term Memory (LSTM) Networks:
among most efficient implementations of RNNs
alleviate vanishing grad: add more paths in NN where gradient well defined

Idea: additional loops in NN where gradient does not vanish

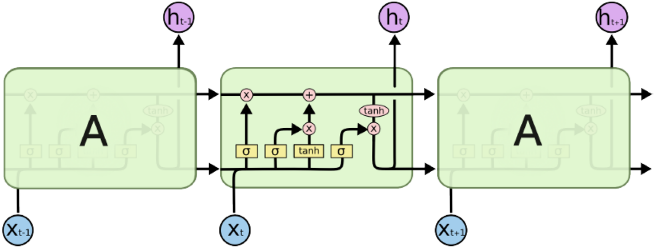
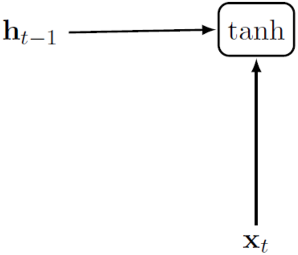


Figure: Chris Olah

basic unit of LSTM is a memory cell (shaded green). Let us zoom-in to understand it

LSTM

Build LSTM gradually step by step.



snapshot of input-output at time t
source: Trivedi, Kodor, 2017

$$\tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{U}\boldsymbol{x}_t)$$

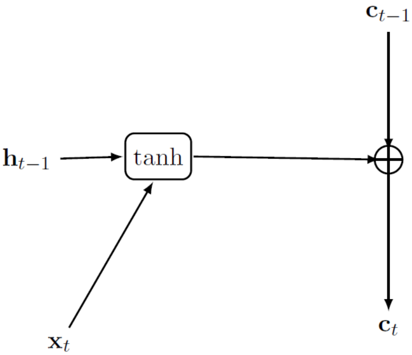
$\tanh()$: non-linear activation (element-by-element func)

$\tilde{\boldsymbol{c}}_t$: output of non-linear activation (tanh)

$\boldsymbol{U}, \boldsymbol{V}$: matrices of appropriate size

so far it looks like first layer of a plain RNN

LSTM



$$\tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{U}\boldsymbol{x}_t)$$
$$\boldsymbol{c}_t = \boldsymbol{c}_{t-1} + \tilde{\boldsymbol{c}}_t$$

snapshot of input-output at time t
source: Trivedi, Kodor, 2017

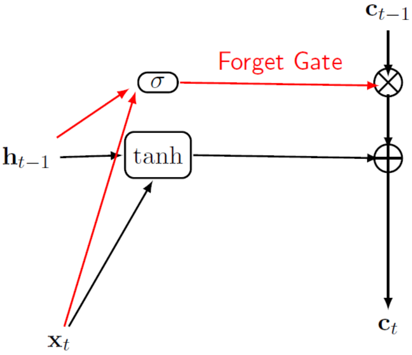
$\tilde{\boldsymbol{c}}_t$: output of non-linear activation (tanh)

$\boldsymbol{c}_t$: running sum of output of non-linear activation, over t

sum increases with t and may blow-up. how to prevent it?

scale down previous output $\boldsymbol{c}_{t-1}$: **forget gate**

LSTM



snapshot of input-output at time t
source: Trivedi, Kodor, 2017

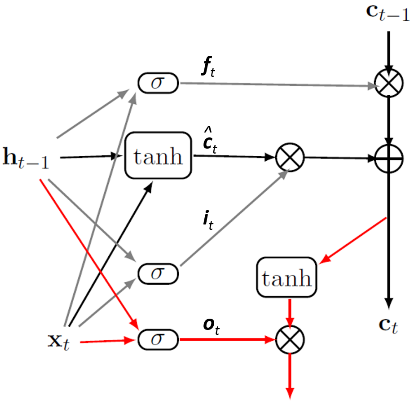
$$\tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{U}\boldsymbol{x}_t)$$
$$\boldsymbol{c}_t = \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \tilde{\boldsymbol{c}}_t$$
$$\boldsymbol{f}_t = \boldsymbol{\sigma}_f(\boldsymbol{W}_f\boldsymbol{h}_{t-1} + \boldsymbol{U}_f\boldsymbol{x}_t)$$

$\boldsymbol{W}_f, \boldsymbol{U}_f$: weight matrices

LSTM with forget gate:

multiply (element by element) previous output $\boldsymbol{c}_{t-1}$, by forget factor $\boldsymbol{f}_t$:
less importance on previous samples
$\boldsymbol{W}_f, \boldsymbol{U}_f$: two additional weights to be learned

LSTM



snapshot of input-output at time t
source: Trivedi, Kodor, 2017

fully-fledged LSTM

$$\tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{U}\boldsymbol{x}_t)$$
$$\boldsymbol{c}_t = \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \tilde{\boldsymbol{c}}_t \circ \boldsymbol{i}_t$$
$$\boldsymbol{f}_t = \boldsymbol{\sigma}_f(\boldsymbol{W}_f\boldsymbol{h}_{t-1} + \boldsymbol{U}_f\boldsymbol{x}_t)$$

Define input/output gates:

$$\boldsymbol{i}_t = \boldsymbol{\sigma}_i(\boldsymbol{W}_i\boldsymbol{h}_{t-1} + \boldsymbol{U}_i\boldsymbol{x}_t)$$
$$\boldsymbol{o}_t = \boldsymbol{\sigma}_o(\boldsymbol{W}_o\boldsymbol{h}_{t-1} + \boldsymbol{U}_o\boldsymbol{x}_t)$$
$$\boldsymbol{h}_t = \boldsymbol{o}_t \circ \tanh(\boldsymbol{c}_t)$$

$\boldsymbol{W}_i, \boldsymbol{U}_i, \boldsymbol{W}_o, \boldsymbol{U}_o$: weight matrices

Unpacking the fully fledged LSTM:

Information added/deleted from cell via input, forget or output gates
each gate modeled by two matrices $\boldsymbol{W}_x, \boldsymbol{U}_x$ $(x = \{o, i, f\})$, that are learned from training

Cell state: $\boldsymbol{c}_t = \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \tilde{\boldsymbol{c}}_t \circ \boldsymbol{i}_t$
think of $\boldsymbol{c}_t$ as state of the cell
updated as func of info to be deleted ($\boldsymbol{f}_t$), and to be added ($\boldsymbol{i}_t$)

Forget gate: $\boldsymbol{f}_t = \boldsymbol{\sigma}_f(\boldsymbol{W}_f \boldsymbol{h}_{t-1} + \boldsymbol{U}_f \boldsymbol{x}_t)$
$\boldsymbol{f}_t$ multiplies previous output, $\boldsymbol{c}_{t-1}$, to reduce importance of previous samples
by learning $\boldsymbol{W}_f, \boldsymbol{U}_f$ LSTM throw away some unnecessary information

Input gate: $\boldsymbol{i}_t = \boldsymbol{\sigma}_i(\boldsymbol{W}_i \boldsymbol{h}_{t-1} + \boldsymbol{U}_i \boldsymbol{x}_t)$
what information from input to store in cell ?
by multiplying output of non-linearity, $\tilde{\boldsymbol{c}}_t$
intuition: learning $\boldsymbol{W}_i, \boldsymbol{U}_i$ LSTM decides what information to add to cell (store)

Output gate: $\boldsymbol{o}_t = \boldsymbol{\sigma}_o(\boldsymbol{W}_o \boldsymbol{h}_{t-1} + \boldsymbol{U}_o \boldsymbol{x}_t)$
pass a “filtered version” of output (same intuition as input)
intuition: learning $\boldsymbol{W}_o, \boldsymbol{U}_o$ LSTM decides what information to pass to output

Update cell state: $\boldsymbol{c}_t = \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \tilde{\boldsymbol{c}}_t \circ \boldsymbol{i}_t$
update state taking into account forget gate $\boldsymbol{f}_t$, and input $\boldsymbol{i}_t$

Related issues LSTM:

Training LSTM: similar to Recurrent NNs
in addition to learning $\boldsymbol{W}, \boldsymbol{U}$ for Recurrent NNs
for LSTM the weight matrices, $\boldsymbol{W}_i, \boldsymbol{U}_i, \boldsymbol{W}_o, \boldsymbol{U}_o, \boldsymbol{W}_f, \boldsymbol{U}_f$, also learned.
BPTT may still be used (derivations more complex)

Gated Recurrent Unit: is a related implementation of LSTM
see [Goodfellow, 2016], Chapter 10

Conclusions

Several DL architectures

- Recurrent NNs: motivated by learning sequential data
  math model of plain recurrent NN, training with BPTT, some variants
  challenge of long term dependencies

- LSTM: practical solution to bypass challenge of long term dependencies
  solution to vanishing gradient
  discussed math model for LSTM, role of gates: input, output, forget

Some references

J. Wu, "Introduction to convolutional neural networks ", May 1, 2017, available on arxiv

A. Goodfellow, Y. Bendgio, A Courville, "Deep Learning", MIT Press, Chap. 8, 9

Y. LeCun, L Bottou, G. Orr, K-B Muller, "Efficient Backprop", Neural Networks: Tricks of the trade

S. Trivedi, R. Kondor, CMSC 35246 Deep Learning, Spring 2017

C. M. Bishop, "Pattern Recognition and Machine Learning", Springer-Verlag, 2006

S. Haykin, "Neural Networks and Learning Machines", 3rd, 2009, Chap 4

Check Tutorial by Ruslan Salakhutdinov:
`http://www.cs.cmu.edu/~rsalakhu/`

Deep Learning

Lecture 8: Deep Learning Architectures (Part I)

University of Agder,
Kristiansand, Norway

Prepared by: Hadi Ghauch⋆, Hossein S. Ghadikolaei†

⋆ Telecom ParisTech

† Royal Institute of Technology, KTH

https://sites.google.com/view/fundl/home

April 2019