PROBABILISTIC AND DIFFERENTIABLE PROGRAMMING

V4: Deep Learning II
(RNNs and Reservoir)

Özgür L. Özçep
Universität zu Lübeck
Institut für Informationssysteme
Today’s Agenda

1. Follow me:
   Recurrent networks

2. Some things to remember, some things to forget:
   Long short term memory

3. Forget to learn the hiddens:
   Reservoir Computing
Example Named Entity recognition

\[
\begin{array}{cccccccc}
  x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\
\end{array}
\]

- **x**: Jon and Ethan gave deep learning lectures

\[
\begin{array}{cccccccc}
  y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 \\
  1 & 0 & 1 & 0 & 0 & 0 & 0 \\
\end{array}
\]

- In this case input and output vector of length 7
- But naturally longer sequences are possible
Image classification
In: image
Out: Classifier

Image captioning
In: image
Out: sentence

Sentimental analysis
In: sentence
Out: sentiment

Machine translation
In: sentence
Out: sentence

Synced video
In: video
Out: real-time labels
Why Not a Standard Feed Forward Network?

- For a task such as “Named Entity Recognition” a MLP would have several disadvantages
  - The inputs and outputs may have varying lengths
  - The features wouldn’t be shared across different temporal positions in the network
    - Note that 1-D convolutions can be (and are) used to address this, in addition to RNNs
- To interpret a sentence or to predict tomorrow's weather it is necessary to remember what happened in the past
- To facilitate this we would like to add a feedback loop delayed in time
RNN Architecture

\[ o(t) = \hat{y}(t) \]

Output

\[ h(t) \]

Hidden Units

\[ h(t-1) \]

Delay

\[ x(t) \]

Inputs

\[ weights \ V \]

\[ weights \ U \]

\[ weights \ W \]

- RNNs are NNs for processing sequential data
- Contain directed cycles in their computational graph
  - Another form of "more structure" in DL
  - Another form of parameter sharing in DL

\[ aw = u \cup v \cup w \]
RNN Architecture

Left: feed forward neural network
Middle: a simple recurrent neural network
Right: Fully connected recurrent neural network
An RNN is just a recursive function invocation

- Output update
  \[ \hat{y}(t) = f_o(x(t), h(t - 1)|AW) \]
- State update
  \[ h(t) = f_h(x(t), h(t - 1)|AW) \]
- If \( \hat{y}(t) \) depends on the input \( x(t - 2) \), then prediction will be
  \[ f_o(x(t), f_h(x(t - 1), f_h(x(t - 2), f_h(x(t - 3)|AW)|AW)|AW) |AW) \]
- Gradients of this with respect to the weights can be found with the chain rule
Variants of RNNs

- Depending on the instantiation of $f_h()$
  - Elman (Vanilla/Simple Networks)
  - Jordan (not discussed here)
  - LSTM (discussed here)
  - GRU (Gated recurrent unit; not discussed here) bka

- Elman
  - $h_t = f_h(Ux_t + b_U + Wh_{t-1} + b_W) = f_h(a_h(t))$
  - $\hat{y}_t = o(t) = f_o(Vh_t + b_o) = f_o(a_o(t))$
  - $f_h$ is usually $tanh$
  - $f_o$ identity or logit
RNNs combine two properties which make them very powerful.

1. Distributed hidden state that allows them to store a lot of information about the past efficiently. This is because several different units can be active at once, allowing them to remember several things at once.

2. Non-linear dynamics that allows them to update their hidden state in complicated ways.

- In particular: RNNs are universal approximators
Going Deep with RNNs

- You can go deep w.r.t. time unfolding (some do not consider this as going deep)
- As RNNs calculate functions, you can compose them (stack the RNNs)
  \[ \hat{y}(t) = f^2_0(f^1_0(x(t), h^1(t - 1)|AW_1), h^2(t - 1)|AW_2) \]
  - The output of the inner RNN at time \( t \) is fed into the input of the outer RNN which produces the prediction \( \hat{y} \)
- You could of course also add feedforward parts into the input block or the output block or the hidden block
Example: Character-level language modelling

- An RNN that learns to ‘generate’ English text by learning to predict the next character in a sequence
- This is “Character-level Language Modelling”

![Diagram showing an RNN model for character-level language modelling](http://karpathy.github.io/2015/05/21/rnn-effectiveness/)
Training and sampling the Language Model

• The training data is just text data (e.g. sequences of characters)
• The task is unsupervised (or rather self-supervised): given the previous characters predict the next one
• All you need to do is train on a reasonable sized corpus of text
• Overfitting could be a problem: dropout is very useful here

• Once the model is trained can generate text
  – See examples at http://karpathy.github.io/2015/05/21/rnn-effectiveness/
Recurrent Neural Networks unfolded

- Can unravel/unfold network into feed forward
  - can apply gradient descent/timed backpropagation (BPTT: Backpropagation through time)
  - Minimize error $\sum_t \left| y(t) - \hat{y}(t) \right|^2$ over all time steps
Back Propagation Through Time (BPTT)

- BPTT learning algorithm is an extension of standard backpropagation that performs gradients descent on an unfolded network.

- The gradient descent weight updates have contributions from each time step.

- The errors have to be back-propagated through time as well as through the network.
RNN Backward Pass

- Loss function depends on the activation of the hidden layer through its influence on the output layer and through its influence on the hidden layer at the next step.
  - $h_t = f_h(x, h_{t-1}, W)$
  - $o_t = f_o(h_t, V)$
- The interesting part is the calculation of the gradient w.r.t. the hidden parameters $W$
  - $E = \sum_{t=1}^{T} E_t$ (error in RNN)
  - $\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial o_t} \frac{\partial o_t}{\partial h_t} \frac{\partial h_t}{\partial W}$ (by chain rule)
  - $\frac{\partial h_t}{\partial W} = \frac{\partial f_h(x_t, h_{t-1}, W)}{\partial W} + \frac{\partial f_h(x_t, h_{t-1}, W)}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W}$
  - $\frac{\partial h_t}{\partial W} = \frac{\partial f_h(x_t, h_{t-1}, W)}{\partial W} + \sum_{i=1}^{t-1}(\prod_{j=i+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}) \frac{\partial f_h(x_i, h_{i-1}, W)}{\partial W}$ (by solving the recursion)
Here they come again: Vanishing and exploding Gradients

Solution: Long short term memory networks (LSTMs)
LONG SHORT TERM MEMORY
LSTM - introduction

- LSTM was invented to solve the vanishing gradients problem.
- LSTM maintain a more constant error flow in backpropogation.
  - Long term memory by specific hidden state \( c(t) = c(t-1) \)
  - Sometimes one has to forget and sometimes have to change the memory
  - To do this use gates saturating at 0 (read/write denied) and 1 (read/write allowed) => Sigmoid
- LSTM can handle global dependencies (1000 time steps)
Figure 2.1: Left: RNN with one fully recurrent hidden layer. Right: LSTM network with memory blocks in the hidden layer (only one is shown).
LSTM Architecture

![LSTM Architecture Diagram]
LSTM Architecture - overview
LSTM Architecture – long term memory cell

- Each memory cell contains a node with a self-connected recurrent edge of fixed weight one
- Ensures that the gradient can pass across many time steps without vanishing
- CEC (constant error carousel)

=> Long term memory
- In contrast: Previous outputs from hidden: short term memory

\[ c(t) = z(t) \odot i(t) + c(t - 1) \]
LSTM Architecture – input

\[ a_z(t) = W_z x(t) + R_z y(t - 1) \]
\[ z(t) = g(a_z(t)) \]

(control forwarding of input and previous step information)
LSTM Architecture – input gate

\[ a_{in}(t) = W_{in}x(t) + R_{in}y(t - 1) \]

\[ i(t) = \sigma(a_{in}(t)) \]

(control write access to memory cells)
LSTM Architecture – Output gate

\[ a_{out}(t) = W_{out}x(t) + R_{out}y(t - 1) \]
\[ o(t) = \sigma(a_{out}(t)) \]

(control read access to memory cell)
LSTM Architecture – Output gate

\[ y(t) = h(c(t)) \odot o(t) \]

(control outputting of memory cell content via o(t))
LSTM Forward Pass

- The cell state $c$ is updated based on its current state and 3 inputs: $a_z, a_{in}, a_{out}$

\[
\begin{align*}
    a_z(t) &= W_z x(t) + R_z (y(t-1)), z(t) = g(a_z(t)) \\
    a_{in}(t) &= W_{in} x(t) + R_{in} (y(t-1)), i(t) = \sigma(a_{in}(t)) \\
    c(t) &= z(t) \odot i(t) + c(t-1) \\
    a_{out}(t) &= W_{out} x(t) + R_{out} (y(t-1)), o(t) = \sigma(a_{out}(t)) \\
    y(t) &= h(c(t)) \odot o(t)
\end{align*}
\]
LSTM Backward Pass

- Errors arriving at cell outputs are propagated to the CEC
- Errors can stay for a long time inside the CEC
- This ensures non-decaying error
- Can bridge time lags between input events and target signals

- (details left out here)
An addition: Handling unbounded memory

\[ c(t) = z(t) \odot i(t) + c(t - 1) \to \text{grows linearly} \]

For a continuous input stream \( \rightarrow \)
\( c(t) \) may grow in an unbounded fashion \( \rightarrow \)
can cause a saturation in \( h(t) \)

\[ \delta_c(t) = \delta_y(t) h'(c(t)) \odot o(t) \]

Small gradients
LSTM possible remedy by forget gate

\[
f(t) = \sigma \left( W_f x(t) + R_f (y(t - 1)) \right) \\
\]
\[
c(t) = z(t) \odot i(t) + f(t) \odot c(t - 1)
\]
Success Story of LSTMs

• LSTMs have been used to win many competitions in speech and handwriting recognition.
• Major technology companies (Google, Apple, and Microsoft) are using LSTMs
  – Google used LSTM for speech recognition on the smartphone, for Google Translate.
  – Apple uses LSTM for the ”Quicktype” function on the iPhone and for Siri.
  – Amazon uses LSTM for Amazon Alexa.
  – In 2017, Facebook performed some 4.5 billion automatic translations every day using long short-term memory networks1.
Reservoir Computing (RC): Idea

- Idea: Separate state space calculation from output calculation (as they serve different purposes)
  - State space represents input (history) in high-dimensional (kernel trick)
  - Output spaces: Merger of states for desired output
- Uses recurrent structures without the training
  - Fixed (random) topology
  - Linear “readout” function is trained
Reservoir Computing: History

- Boyd/Chua (95): Mathematical Foundation (without input feedback)
- Jaeger (2001): Echo State Networks (engineering)
- Maass (2002): Liquid State Machines (neuroscience)
- ...
- And now: various physical reservoir computing approaches (morphological computing, cellular automata, etc.) (see Tanaka et al. 19)
Reservoir Computing

• (De-facto & required) Properties of reservoir:
  – Exact topology, connectivity, weights: not important
  – Has to have fading memory/echo state property: when not chaotic (as $k \to \infty$) effect of $h(t)$ and $x(t)$ on $h(t + k)$ vanishes
  – Can be ensured by choosing spectral radius of weight matrix (largest absolute eigenvalue) smaller than 1
  – Reservoir size can be large: no over-fitting
• Training with linear regression (or similar):
  – No local minima, no problems with recurrent structure, one shot learning
  – Can do regression, classification, prediction
Reservoir Computing

- RC does on-line computing: prediction at every time-step
- Theoretically:
  - Any time-invariant filter (\( F(x(t)) = F(x(t), t) \)) with fading memory can be learned
  - But: unable to implement generic FSMs
- Hence add output feedback, Maass (2006)
  - Also non-fading memory filters: generic FSMs
  - Ability to simulate any n-th order dynamical system
  - Turing universal
Usual setup and training

- Create random weight matrices
- Rescale reservoir weights so that max absolute eigenvalue close to one (edge of stability)
- Excite reservoir with input and record all states
- Train readouts by minimizing $(AV - B)^2$
Link to FSMs

FSM
RC
RC with output feedback
RC: Applications

- Chaotic time series prediction
- Speech recognition on small vocabulary: outperform HMM-based recognizer (Sphinx)
- Digits recognition
- Robot control
- System identification
- Noise removal/modelling
- …
Larger example: speech

Speech

Pre-processing

Reservoir

Readout

Post-processing

Reservoir state

Downsampling

Mean

WTA

'6'
RC: novel computing paradigm

- RC presents a novel way of looking at computation
- “Random” dynamic systems can be used by only training a linear readout layer
- RC already used to show general computing capabilities of:
  - Microcolumn structure in the cortex
  - Gene regulatory network
  - The visual cortex of a real cat
- Implementations:
  - “Bucket of water”, aVLSI, digital hardware
  - Photonics
Different Flavors of RC

- Water is mechanically perturbed (with motors)
- Complex response of the surface
- Readout is digitized picture frame + processing (vision)

Fernando and Sojakka, 2003
Uh, a lecture with a hopefully useful

APPENDIX
Color Convention in this course

- Formulae, when occurring inline
- Newly introduced terminology and definitions
- Important results (observations, theorems) as well as emphasizing some aspects
- Examples are given with standard orange with possibly light orange frame
- Comments and notes
- Algorithms
Today’s lecture is based on the following

- Jonathon Hare: Lectures 12, 13 of course „COMP6248 Differentiable Programming (and some Deep Learning)“
  http://comp6248.ecs.soton.ac.uk/
- Michael Green & Shaked Perek: Recurrent networks And Long Short Term Memory link
- Karpathy: The unreasonable effectiveness of recurrent Neural Networks
  http://karpathy.github.io/2015/05/21/rnn-effectiveness/
- Benjamin Schrauven et al: An overview of reservoir computing, ESANN 2007 (paper and slides, link)
- Helmut Hauser, 2013: Introduction to Reservoir Computing
- Deep Dive into deep learning, chapter 8
References


