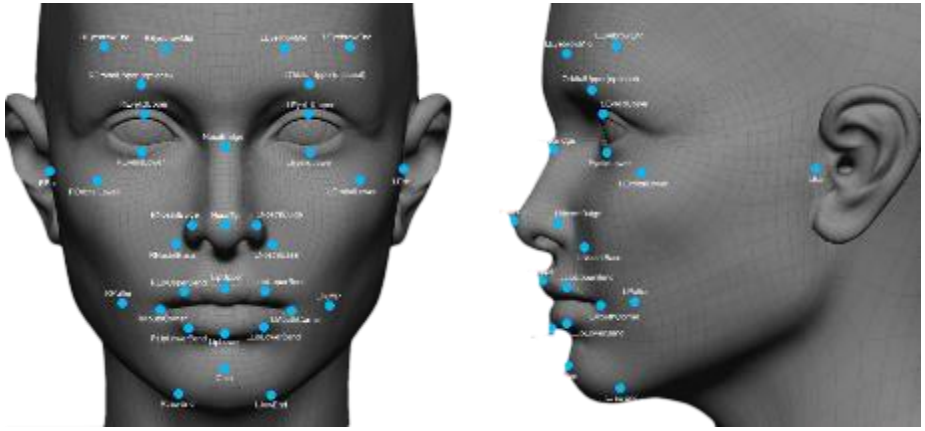


Recurrent neural networks. Part 1

Lecture # 10 of MIPT course: Introduction to machine learning

Speaker: Alexey O. Seleznev

PhD in Computational Chemistry

5VISION TEAM

Moscow

2017

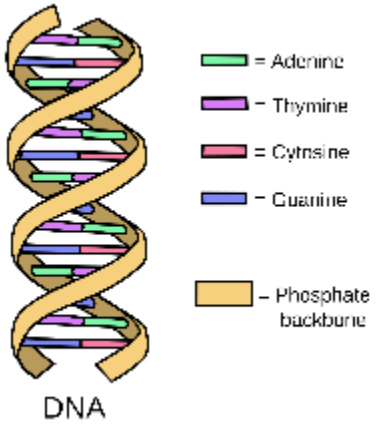
OUTLINE

- Traditional approaches to sequence modeling
  - Sequence modeling
  - Traditional approaches and their drawbacks
- Introduction to recurrent neural networks (RNN)
- RNN training
  - Backpropagation through time
  - Exploding & vanishing gradient problem
- Gated RNN
  - Long short-term memory (LSTM)
  - Gated recurrent unit (GRU)
- References

Traditional Approaches to Sequence Modeling

Sequence Modeling

- **Definition**: finding and exploiting statistically relevant patterns between data examples where the values are delivered in a sequence
- Typical examples:
  - Bioinformatics (DNA, RNA codes)
  - Natural language processing
  - Stock market
  - Speech recognition
  - Weather forecasts



Traditional Approaches and Their Drawbacks

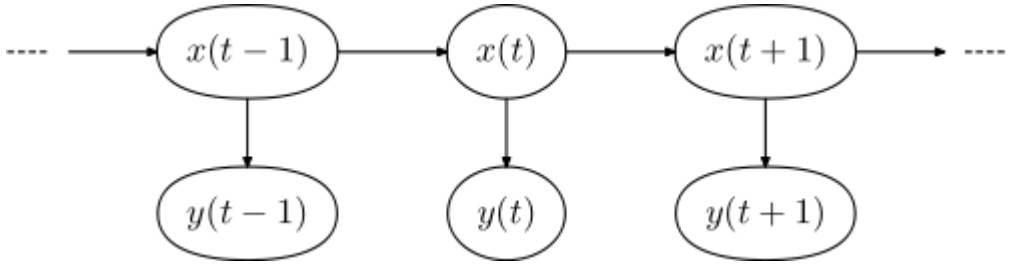
- Approach 1: autoregressive models and their generalizations, e.g. AR(2), MA(1), ARIMA(2, 1, 2)
  - Idea: present current value as a noised sum of n previous values

$$\vec{X}_t = \phi_0 + \phi_1 \vec{X}_{t-1} + \phi_2 \vec{X}_{t-2} + \epsilon_t \quad \text{for AR(2)}$$

  - The training consists in finding parameters ϕ by e.g. solving Yule–Walker equations
  - Drawbacks: linear models, the static nature of the underlying processes, does not analyze causation (just tries to fit the points)

Traditional Approaches and Their Drawbacks

❖ Approach 2: Hidden Markov models

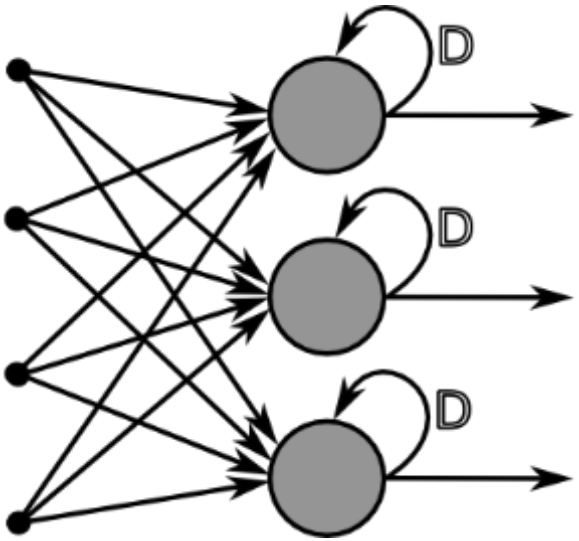


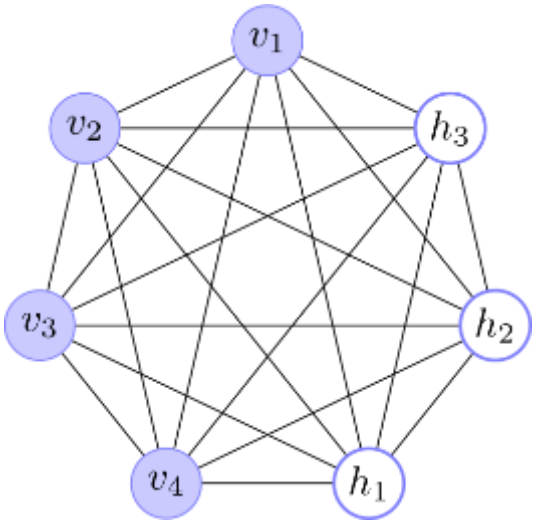
- Idea: guess underlying state of the system. HMMs have a discrete one-of-N hidden state. Transitions between states are stochastic and controlled by a transition matrix. The outputs produced by a state are stochastic
- To predict the next output we need to infer the probability distribution over hidden states
- Drawbacks: limited memory (N hidden states), poor channel to exchange information between two time steps

Introduction to Recurrent Neural Networks

Introduction to Recurrent Neural Networks

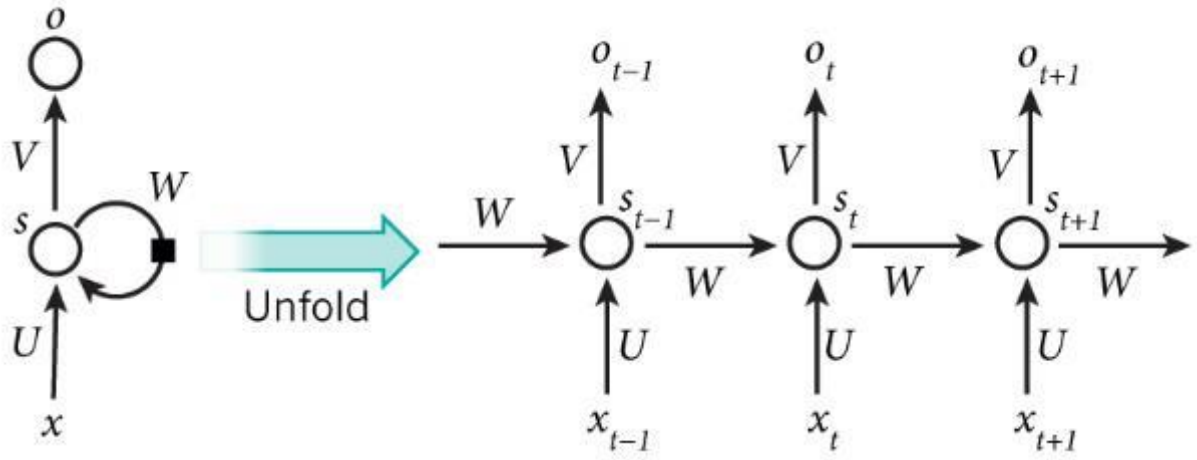
- **Definition**: RNN is a class of artificial neural network where connections among some units form a directed cycle(s)
- Cycles represent the influence of the present value of a variable on its own value at a future time step
- RNNs can use their internal memory (hidden states in cycles) to emulate dynamic processes





Introduction to Recurrent Neural Networks

- In practice, an unfolded version of RNN is used



- From the unfolded version, one can see 2 ideas that RNNs capitalize on:
  - Parameter sharing over time steps (generalization: application of the same rules to objects of the similar nature + independence of sequence length)
  - Ability to learn which aspects of sequences to keep and with what precision (hidden states)

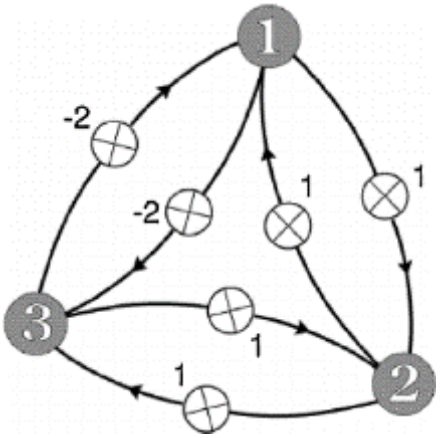
Introduction to Recurrent Neural Networks

- RNNs resemble convolutional networks (e.g. idea of parameter sharing). What is the difference?

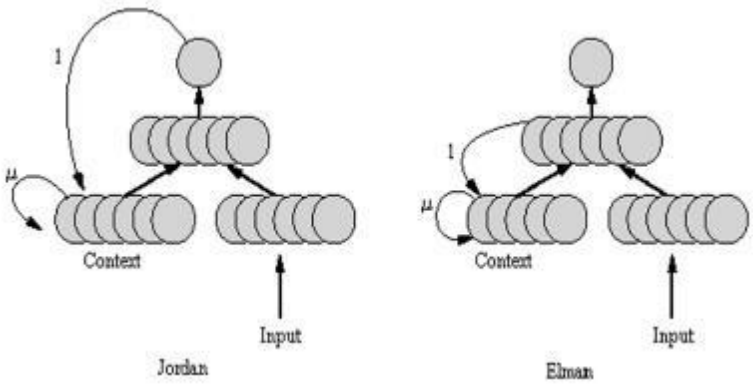
Type of Network	Parameter sharing	Ability to consider the context	Ability to consider dynamic context	Ability to consider long-term dependencies
CNN	+	+	-	-
RNN	+	+	+	+

Introduction to Recurrent Neural Networks

- Simple RNNs (historically important, but currently outdated):
  - Hopfield networks (associative memory)
    - Input and output are interrelated
    - Units take only 2 values {0, 1} or {-1, 1}
    - Use a specific learning algorithm
    - Can restore corrupted patterns

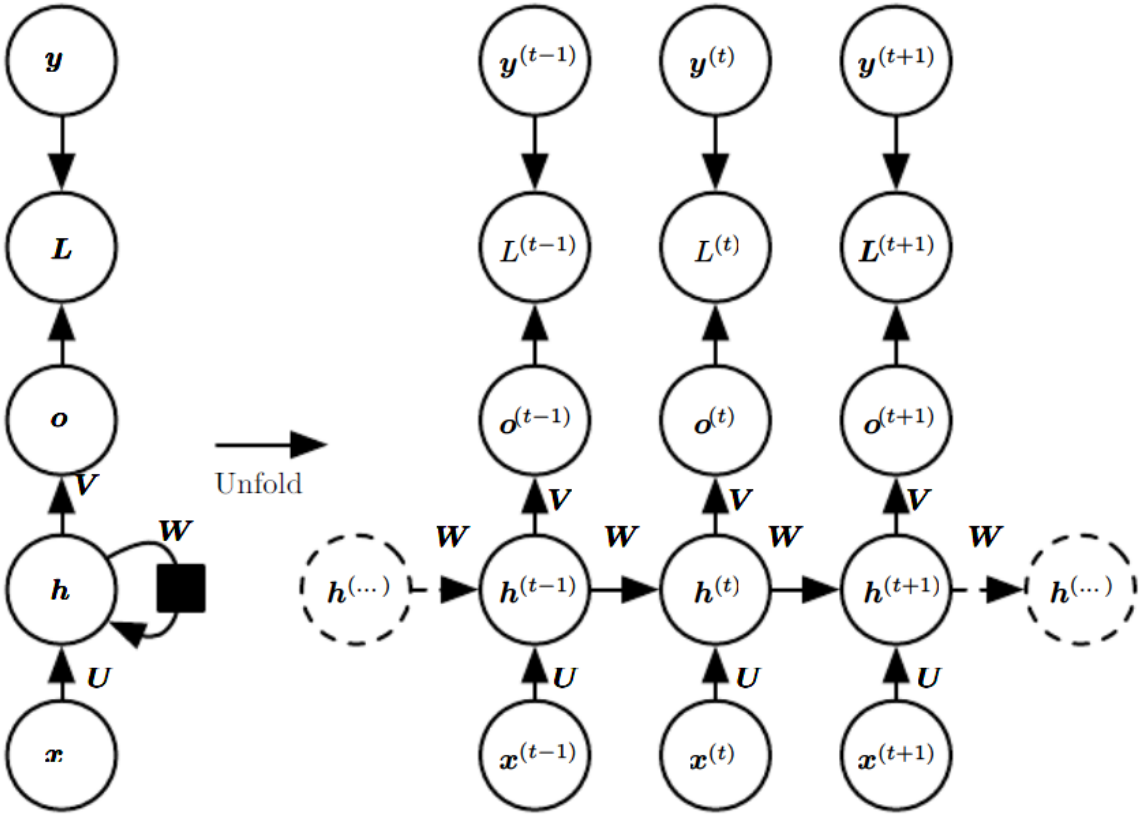


  - Jordan and Elman networks
    - Fixed feedback connections
    - The idea of the context
    - Trained via traditional BP



Introduction to Recurrent Neural Networks

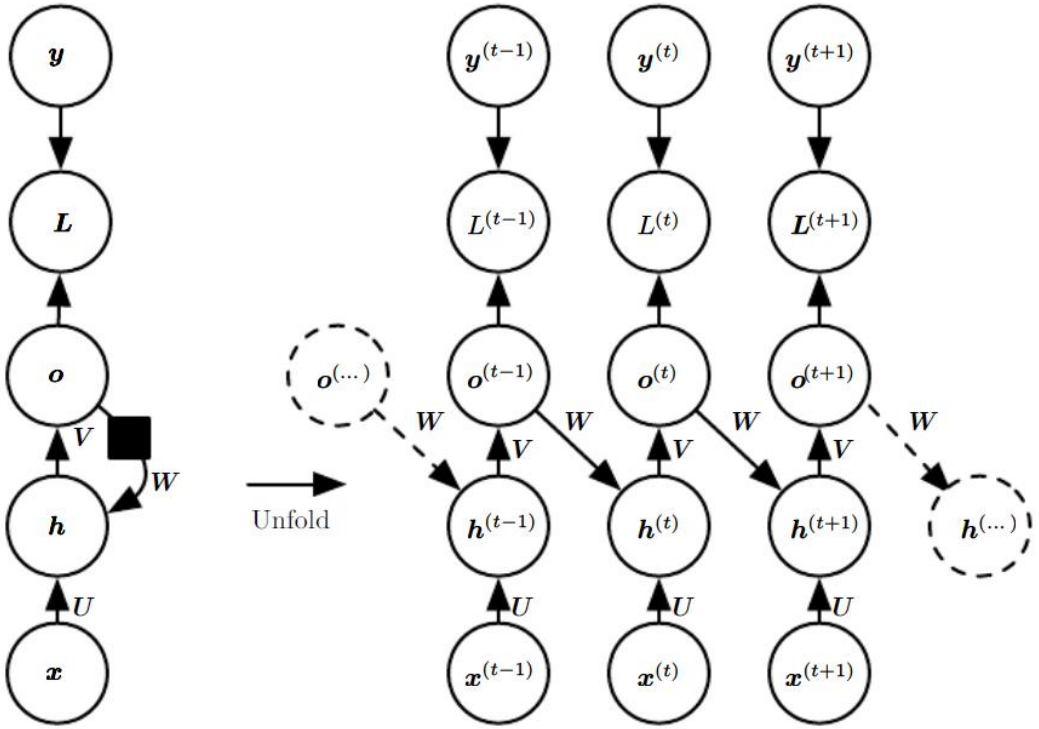
- 3 important patterns in modern RNNs
  - Networks producing an output at each time step and having recurrent connections among hidden units (most typical form!)



Goodfellow et al (book)

Introduction to Recurrent Neural Networks

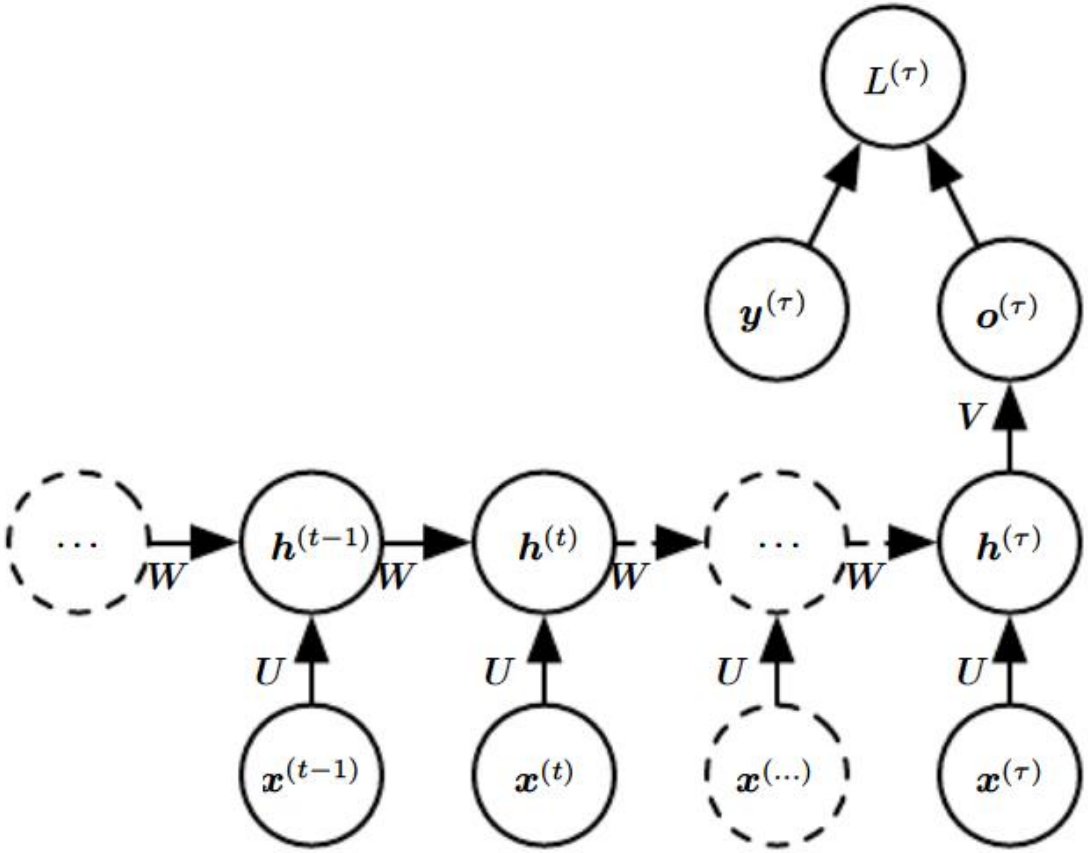
- Networks producing an output at each time step and having recurrent connections only from the output at one time step to the hidden units at the next time step (+parallelization, online learning; -reduced capabilities)



Goodfellow et al (book)

Introduction to Recurrent Neural Networks

- Networks with recurrent connections between hidden units, that read an entire sequence and then produce a single output



Goodfellow et al (book)

Break 5 min

RNN Training

Backpropagation Through Time

❖ **Let us first review the standard backpropagation procedure**:

- Forward pass. Compute activations of internal and output units

$$x_i^{l+1}(n) = f(z_i^l) = f\left(\sum_{j=1,\ldots,K(l)} w_{ij}^l x_j(n)\right)$$

- Backward pass. Iterative procedure to calculate $\frac{\partial Loss}{\partial w_{ij}}$

$$\delta_i^{out}(n) = \frac{\partial Loss}{\partial f}\frac{\partial f(u)}{\partial u}\Big|_{u=z_i^l(n)}$$

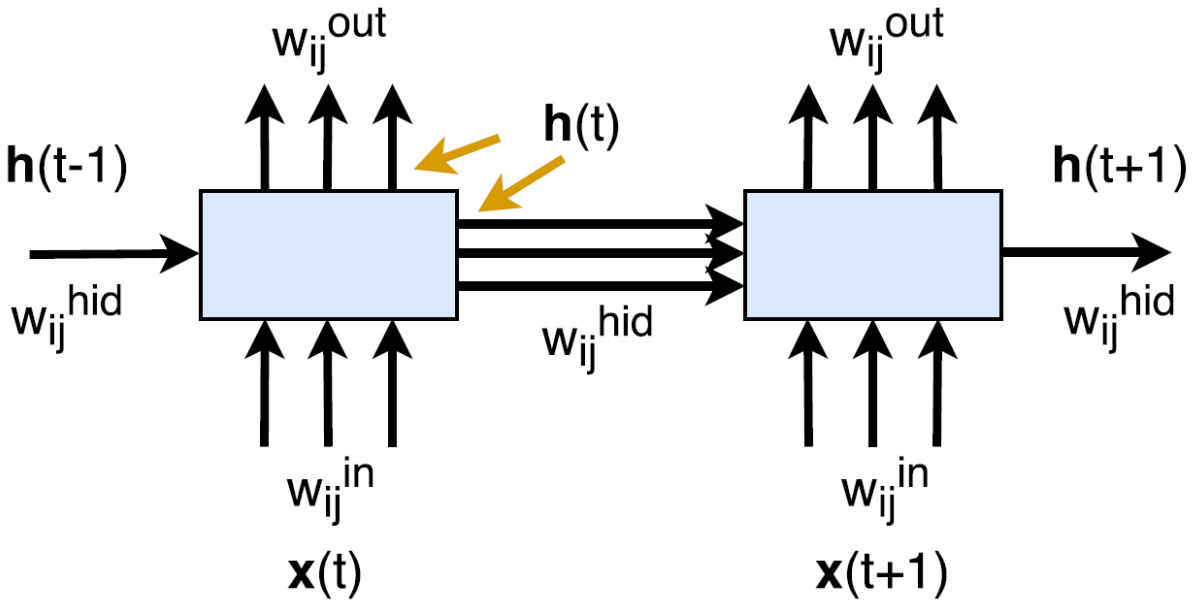
$$\delta_i^l(n) = \sum_{j=1}^{K(l+1)} \delta_j^{l+1}(n) w_{ji}^{l+1} \frac{\partial f(u)}{\partial u}\Big|_{u=z_i^l(n)}$$

- Weights update

$$w_{ij}^{l-1} = w_{ij}^{l-1} + \frac{\mu}{N}\sum_{n=1}^{N} \delta_i^l(n) x_j^{l-1}(n)$$

Backpropagation Through Time

- Backpropagation through time is similar to traditional backpropagation for an unfolded RNN (now n -> t):



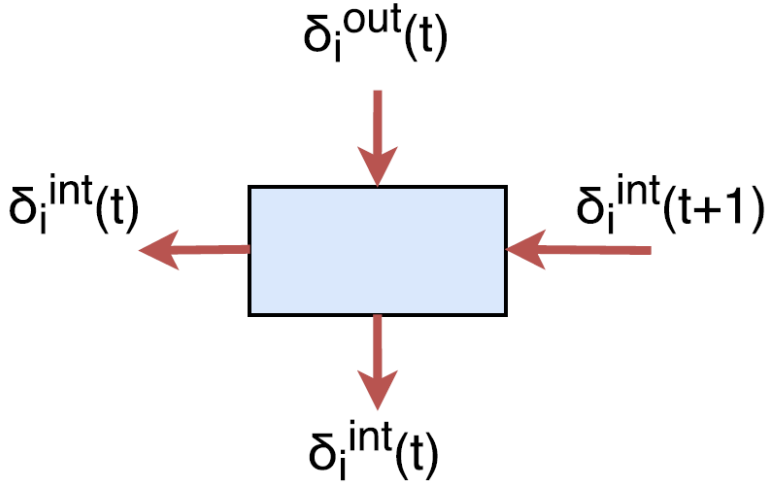
- Forward pass. Compute activations of internal and output units

$$x_i^{l+1}(t) = f^{out}\left(\sum_{j=1}^{K(l+1)} w_{ij}^{out} h_j(t)\right)$$

$$h_i(t) = f^{int}\left(\sum_{j=1}^{K(l)} w_{ij}^{in} x_j^l(t) + \sum_{j=1}^{K(l+1)} w_{ij}^{hid} h_j(t-1)\right)$$

Backpropagation Through Time

- Backward pass



$$\delta_i^{out}(t) = \frac{\partial Loss(t)}{\partial f}\frac{\partial f(u)}{\partial u}|_{u=z_i^{out}(t)} \quad \text{or}$$

$$\delta_i^{out}(t) = \delta_i^{l+1}(t) = \sum_{j=1}^{K(l+2)} \delta_j^{l+2}(t) w_{ji}^{l+2} \frac{\partial f^{out}(u)}{\partial u}|_{u=z_i^{out}(t)}$$

$$\delta_i^{int}(t) = \sum_{j=1}^{K(l+1)} \delta_j^{int}(t+1) w_{ji}^{hid} \frac{\partial f^{int}(u)}{\partial u}|_{u=z_i^{int}(t+1)} + \sum_{j=1}^{K(l+1)} \delta_j^{out}(t) w_{ji}^{out}$$

Backpropagation Through Time

- Weights update

$$w_{ij}^{out} = w_{ij}^{out} + \mu \sum_{t=1}^{T} \delta_i^{out}(t) h_j(t)$$

$$w_{ij}^{in} = w_{ij}^{in} + \mu \sum_{t=1}^{T} \delta_i^{int}(t) x_j(t) \frac{\partial f^{int}(u)}{\partial u}|_{u=z_i^{int}(t)}$$

$$w_{ij}^{hid} = w_{ij}^{hid} + \mu \sum_{t=1}^{T} \delta_i^{int}(t) h_j(t-1) \frac{\partial f^{int}(u)}{\partial u}|_{u=z_i^{int}(t)}$$

❖ In contrast to BP, BPTT has:

- Sum over t in loss function (vs mean over n in BP)
- Slow convergence
- Acute exploding & vanishing gradient problem

Exploding & Vanishing Gradient Problem

- Problem: As t proceeds in RNN (forward or backward), absolute values of $\boldsymbol{\delta}_{\mathbf{int}}$ components either vanish or explode. Information deadlock
- Origin of the problem:

$$\delta_i^{int}(t) = \sum_{j=1}^{K(l+1)} \delta_j^{int}(t+1) w_{ji}^{hid} \frac{\partial f^{int}(u)}{\partial u}|_{u=z_i^{int}(t+1)} + \sum_{j=1}^{K(l+1)} \delta_j^{out}(t) w_{ji}^{out}$$

  - Repeatedly multiplying by a matrix $\mathbf{W}$ is equivalent to $\mathbf{W}^t$. In turn,

$$\boldsymbol{W}^t = \left(\boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}\right)^t = \boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})^t\boldsymbol{V}^{-1}$$

    It means that if eigenvalues are <<1, $\boldsymbol{\delta}_{\mathbf{int}}$ will vanish and vice versa
  - In other words, parameter sharing over different time steps is responsible for the problem. That is why, the problem is not acute for feedforward networks (no parameter sharing, some eigenvalues are >1, other are <1)
- The challenge of preserving long-term dependencies

Exploding & Vanishing Gradient Problem

- How to solve or, at least, to alleviate the problem?
  - 'Manual' solution (gradient clipping; regularizers)
  - Echo state networks
    - Idea: fix input-hidden and hidden-hidden weights and train only hidden-output weights.
    - How to initialize fixed weights to prevent forward explosion? Eigenvalues of the state-to-state transition function Jacobian should be close to 1.
  - Skip connections through time
  - Leaky units
    - Idea: have units with linear self-connections and a weight near 1 on these connections. Example (moving average μ for v)

$$\mu^{(t)} \leftarrow \alpha \mu^{(t-1)} + (1 - \alpha) v^{(t)}$$

  - Gated RNNs

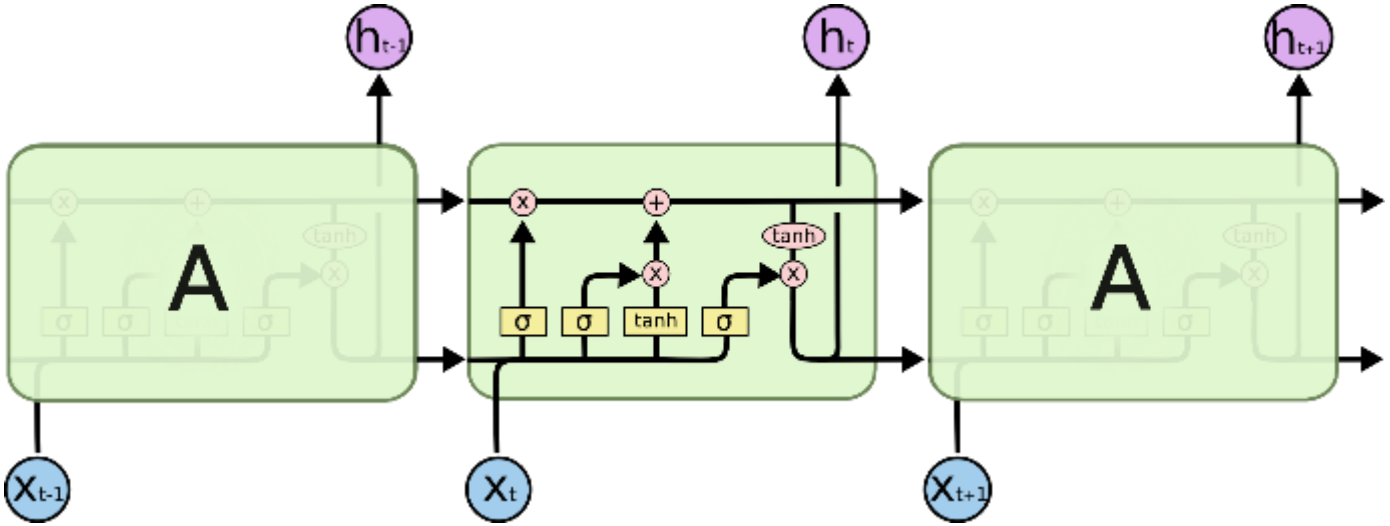
Gated RNNs

Gated RNNs

- Like leaky units, gated RNNs are based on the idea of creating paths through time that have derivatives that neither explode nor vanish
- Leaky units obtain this effect by manually choosing constants. It allows preservation of information for many time steps
- Once the information has been used, it might be useful to forget it.
- In gated RNNs, the neural network decides when to store new information and when to forget it by utilizing memory gates.

LSTM

❖ How does LSTM work?



$$\text{LSTM}: x_t^l, h_{t-1}^l, c_{t-1}^l \rightarrow h_t^l, c_t^l$$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} T_{2n,4n} \begin{pmatrix} x_t^l \\ h_{t-1}^l \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

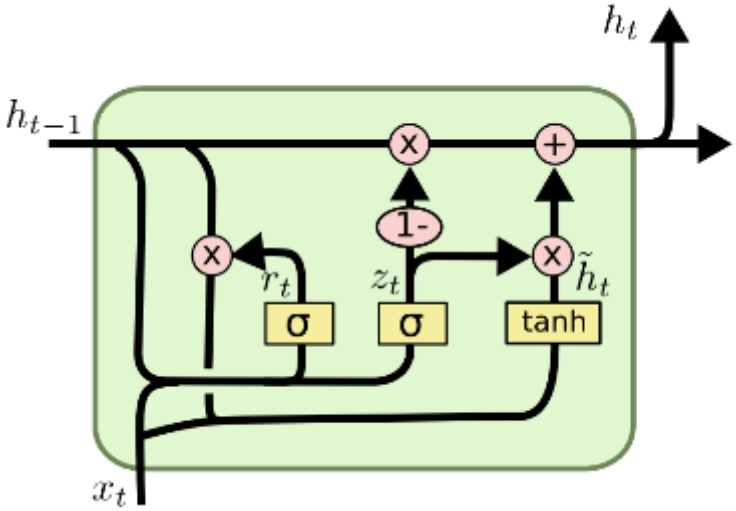
$$h_t^l = o \odot \tanh(c_t^l)$$

Hochreiter and Schmidhuber (1997)

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

GRU

❖ How does GRU work?



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Cho et al. (2014)

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

References

- I. Goodfellow, Y. Bengio, A. Courville. *Deep Learning*. The MIT Press 2016
- S. Hochreiter, J. Schmidhuber. *Long Short-Term Memory*. Journal Neural Computation. Vol. 9 (1997)
- K. Cho, B. Merrienboer, D. Bahdanau, Y. Bengio. *On the properties of neural machine translation: Encoder-decoder approaches*. ArXiv 1409.1259

General Information

- Next Lecture:
  - Title: ‘Recurrent neural networks: Part 2’
  - Main topics:
    - Sequence-to-sequence models
    - Bidirectional and deep RNNs
    - Other algorithms for training RNNs
    - Attention layers (?)
    - Discussion on assignment #3
  - Schedule: 17 May, Wednesday 18:30
- Assignments:
  - Submissions for assignments #2 and #3 should be made through Kaggle