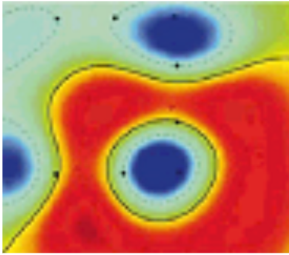


Lecture 9 | Recurrent Neural Networks

Outline

Stateless vs. Stateful Models

- Characterization
- Examples

Recurrent Neural Networks (RNNs)

- General formulation of a RNN
- Examples of practical RNNs
  (e.g. standard, bidirectional, encoder-decoder)
- Choosing the initial state

The Difficulty of Training RNNs

- The vanishing/exploding gradient problem

LSTM Architecture for RNNs

Applications of RNNs

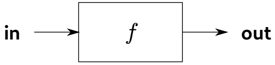
Part 1

Stateless vs. Stateful Models

Stateless vs. Stateful Models

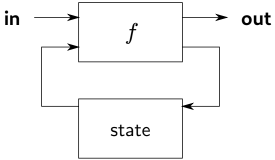
Stateless Predictor

- The prediction is simply a function of the data given as input.
- The data given as input could be e.g. a simple vector of measurement, or a sequence of such vectors (a time series).



Stateful Predictor

- The prediction is a function that produces a prediction from the input and the current state of the system. The function also outputs the future state of the system.



Stateless vs. Stateful Models

Example: Stateless model for moving average

- Equation:

$$y_t = \alpha \cdot x_t + \beta x_{t-1} + \gamma x_{t-2}$$

- This model can be interpreted as a sliding window through the input sequence, and has a finite horizon.
- Assuming an input time series $(x_1, x_2, \dots, x_T)$, values $y_3, y_4, \dots$ can be predicted directly.
- This equation can be modeled using a Convolutional Neural Network (CNN).

Example: Stateful model for moving average

- Equation:

$$\begin{bmatrix} y_t \\ h_t \end{bmatrix} = \begin{bmatrix} h_t \\ \gamma h_{t-1} + (1-\gamma) x_t \end{bmatrix}$$

- This model has an infinite horizon.
- Assuming an input time series $(x_1, x_2, \dots, x_T)$ one needs to specify an initial state h_0 to compute any of the predicted values y_t.

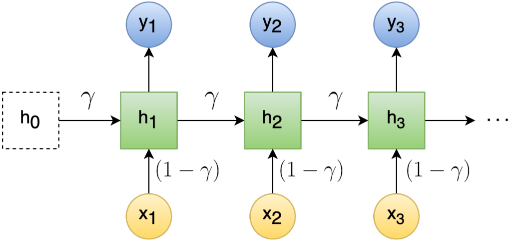
Stateful Models

Stateful Models

- Can be useful when we are dealing with sequential data like natural language, audio, stock prices, and etc.

Questions

- How can we build a network that can solve the second equation of the previous slide?



Part 2

Recurrent Neural Networks

Towards a General Formulation: RNNs

- The model studied above can be generalized by the equation:

$$\begin{bmatrix} \boldsymbol{y}_t \\ \boldsymbol{h}_t \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{h}_{t-1} \end{bmatrix}.$$

  The matrices A, B, C, D can be learned from the data, e.g. to minimize the divergence between the output time series $\boldsymbol{y}$ and some ground-truth time series $\boldsymbol{t}$.

- The model above can be further generalized to:

$$\begin{bmatrix} \boldsymbol{y}_t \\ \boldsymbol{h}_t \end{bmatrix} = f_\theta \left(\begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{h}_{t-1} \end{bmatrix} \right)$$

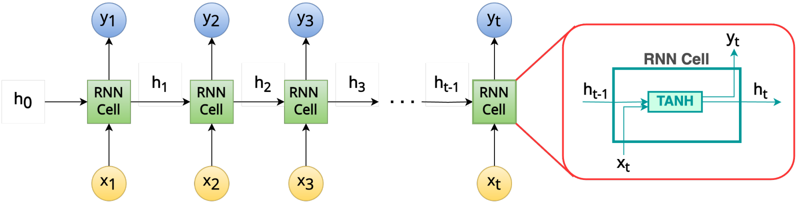
  where f_θ can be any function, e.g. a neural network, with a set of parameters θ to be learned.

RNN Visualization

The Vanilla RNN defined by the equations:

$$h_t = tanh(W_{hh}^T h_{t-1} + W_{xh}^T x_t)$$
$$y_t = W_{hy}^T h_t$$

can be visualized as:



Observation:

- A RNN can be seen as a big neural network composed of a large number of sub neural networks with shared parameters. The whole architecture can be trained via backprop.
- The function f_θ is composed of multiple times. If f_θ is a neural network of depth L, the RNN becomes a network of depth $L \cdot T$.

Sequence-to-Sequence (Seq2Seq)

- Converting an input sequence of tokens into an output sequence of tokens.

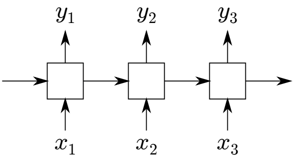
Machine Translation

Example 1:
Input sequence (source language): "Ich spreche Deutsch."
Output sequence (target language): "I speak German."

Example 2:
Input sequence (source language): "Ich gehe morgen ins Kino."
Output sequence (target language): "I am going to the cinema tomorrow."

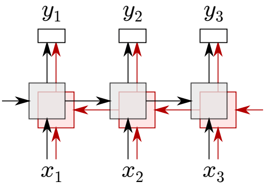
RNN Architectures for Sequence-to-Sequence

Standard (unidirectional) RNNs:



- Generate the output at the same time as the input is received → enable a strong coupling between the two sequences.
- Cannot use information about later time steps when generating the output sequence (problem for e.g. translation).

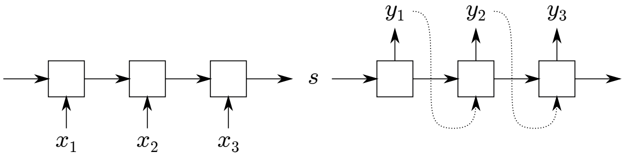
Bidirectional RNNs



- Add a RNN in reverse direction in order to incorporate information from future values in the sequence.

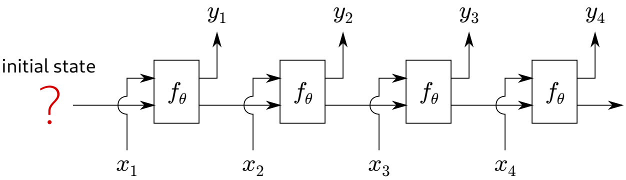
RNN Architectures

Encoder-Decoder RNNs:



- Instead of generating the output sequence at the same time as we process the input sequence, first create a global representation of the input sequence s, and then, generate the output sequence from s.
- This ability to read throught the whole sequence before generating is useful for tasks such as machine translation.

The Problem of Initial States



Problem:

- Unlike the input data, the RNN's initial state (at time $t = 0$) is not given and must be initialized to some value.

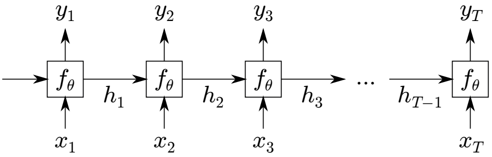
Possible approaches:

- Set it to some arbitrary value (e.g. $\boldsymbol{h}_0 = \mathbf{0}$).
- Set it at random (the RNN will then learn to desensitize itself to the initial state).
- Use one of the two approaches above **and** simulate the RNN for a few time steps in order to generate an initial state that is more plausible.

Part 3

Difficulty of RNN Training

RNN Optimization: Pathological Gradients



The objective to optimize for a RNN is typically expressed as:

$$\mathcal{E} = \ell(y_1, t_1) + \cdots + \ell(y_T, t_T)$$

The gradient of the objective w.r.t. the parameter vector θ can be expressed via the chain rule:

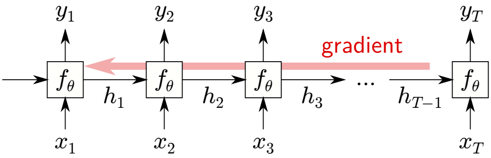
$$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{t=1}^{T} \frac{\partial \mathcal{E}}{\partial y_t} \cdot \left(\frac{\partial^+ y_t}{\partial \theta} + \frac{\partial y_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial \theta} \right)$$

$$\frac{\partial h_{t-1}}{\partial \theta} = \sum_{s=2}^{t-1} \underbrace{\left(\prod_{i=s}^{t-1} \frac{\partial h_i}{\partial h_{i-1}} \right)}_{P_{s,t}} \frac{\partial^+ h_{s-1}}{\partial \theta}$$

Y. Bengio, Simard, and Frasconi 1994

Pascanu, Mikolov, and Yoshua Bengio 2013

RNN Optimization: Pathological Gradients



Observation:

- In the previous slide, we could express the error gradient $\partial\mathcal{E}/\partial\theta$ as a sum over indices $t = 1 \dots T$, and $s = 2 \dots t-1$, where each summand contains a product structure of the type.

$$P_{s,t} = \Big(\prod_{i=s}^{t-1} \frac{\partial h_i}{\partial h_{i-1}}\Big)$$

- On one extreme, the summand corresponding to indices $s = 2$ and $t = T$ features a very large product structure of $T-2$ terms.
- On the other extreme, for summands where $s = t-1$ the product structure totally vanishes (and just becomes an identity matrix I).

Y. Bengio, Simard, and Frasconi 1994
Pascanu, Mikolov, and Yoshua Bengio 2013

RNN Optimization: Pathological Gradients

Analysis for the Linear Model:

- Recall that the linear model is given by the equations:

$$\begin{bmatrix} y_i \\ h_i \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \cdot \begin{bmatrix} x_i \\ h_{i-1} \end{bmatrix}$$

- For such a model, the matrix $P_{s,t}$ can be computed in closed form:

$$P_{s,t} = \left(\prod_{i=s}^{t-1} \frac{\partial h_i}{\partial h_{i-1}} \right) = D^{t-s}$$

hence, $P_{2,T} = D^{T-2}$.

Eigenvalue Decomposition

If D is diagonalizable, the matrix can be rewritten as $D = Q\Lambda Q^{-1}$ with Λ containing the eigenvalues of D, then

$$D^2 = Q\Lambda \underbrace{Q^{-1}Q}_{I} \Lambda Q^{-1} = Q\Lambda^2 Q^{-1}$$

and after a few steps, $D^{T-2} = Q\Lambda^{T-2}Q^{-1}$.

Y. Bengio, Simard, and Frasconi 1994

Pascanu, Mikolov, and Yoshua Bengio 2013

RNN Optimization: Pathological Gradients

Two cases for the Linear RNNs:

$\max_k \lambda_k > 1$ The norm of the matrix D^{T-2} will keep increasing as T becomes large $\rightarrow$ gradients tend to explode.

$\max_k \lambda_k < 1$ The norm of the matrix D^{T-2} will keep decreasing as T becomes large $\rightarrow$ gradients tend to vanish.

Possible Solutions:

- Utilizing gradient clipping helps mitigating the issue of exploding gradients.
- Choosing a particular class of functions for the RNN that is shown to be more robust to the vanishing/exploding gradient problem.

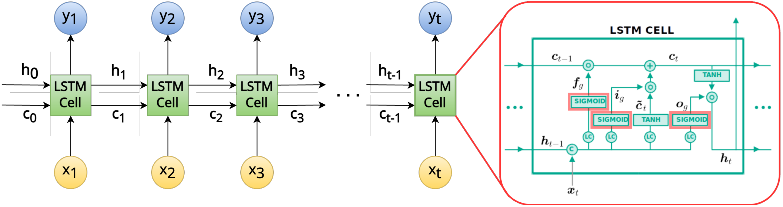
Y. Bengio, Simard, and Frasconi 1994
Pascanu, Mikolov, and Yoshua Bengio 2013

Part 4

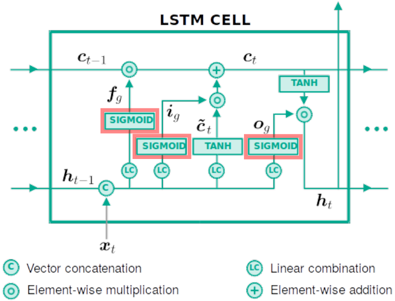
Long Short-Term Memory

Long Short-Term Memory

- The LSTM is an enhanced RNN architecture where the building blocks (cells) are equipped with special functions to stabilize learning, particularly by alleviating the issue of vanishing/exploding gradients.
- The LSTM cell, in comparison to a standard RNN cell, has an additional (more stable) state $\boldsymbol{c}_t$, that is only accessed through gate functions.



Long Short-Term Memory



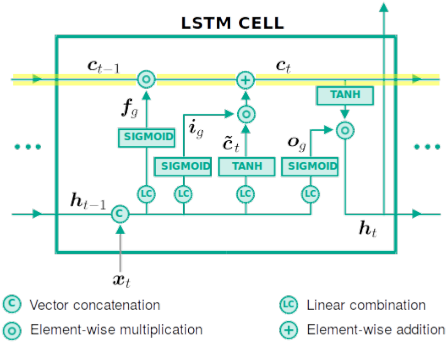
$$f_g = \sigma(W_f^T x_t + U_f^T h_{t-1} + b_f)$$
$$i_g = \sigma(W_i^T x_t + U_i^T h_{t-1} + b_i)$$
$$o_g = \sigma(W_o^T x_t + U_o^T h_{t-1} + b_o)$$
$$c_t' = tanh(W_c^T x_t + U_c^T h_{t-1} + b_c)$$
$$c_t = f_g \odot c_{t-1} + i_g \odot c_t'$$
$$h_t = o_g \odot tanh(c_t)$$

Observation:

- The state $\boldsymbol{c}$ is only accessed through three gates (a gate is a multiplication by a sigmoid). The '*forget gate*' $\boldsymbol{f}_g$ performs an 'erase' operation. The '*input gate*' $\boldsymbol{i}_g$ performs a 'write' operation. The '*output gate*' $\boldsymbol{o}_g$ performs a 'read' operation.

Hochreiter and Schmidhuber 1997

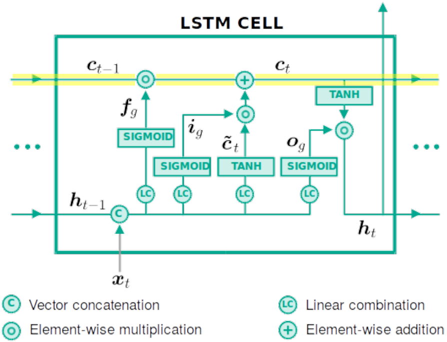
Long Short-Term Memory



Observation (2):

- The state $\boldsymbol{c}$ stays stable over time (it is only erased or updated when the input gate is open), and there are no weight matrices or nonlinearities transforming $\boldsymbol{c}$ over different time steps, i.e. by default it stays constant.

Long Short-Term Memory



Observation (3):

- The gradient flows well and predictably along the path $c_{t-1}, c_t, \ldots$. In particular, the addition operation does not change the gradient. The gradient can then only be dampened by the forget gate, and *never* amplified.

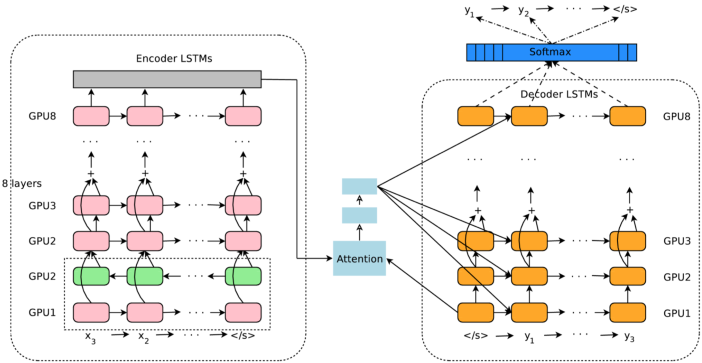
Part 5

RNN Applications

RNNs for Machine Translation

Google Neural Machine Translation:

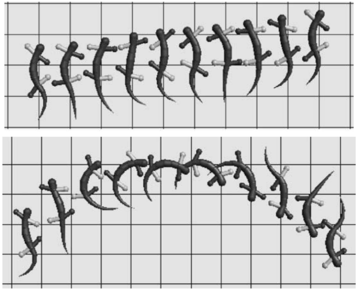
- Encoder-Decoder architecture with input word vectors in the source language, and output in the target language.
- Stack of LSTMs with residual connections through the stack for better gradient flow. First layer is bidirectional.
- Many more details (attention mechanisms, few-shot learning procedure, etc.)



RNNs for Modeling Motion

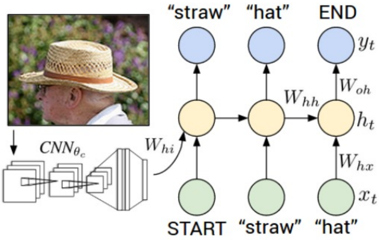
Idea:

- Learn a recurrent neural network model of motion (e.g. of a salamander) from observed behavior.
- The motion can then be steered by forcing certain neurons or input of the RNN to take specific values.
- The model can be analyzed for insights into the mechanisms of locomotion.



RNNs for Image Captioning

- A pre-trained CNN can be used to extract high-level features from the input image and produce an image representation.
- The image representation is passed to the RNN as its initial state.



Summary

Summary

- Recurrent neural networks (RNNs) are a special type of neural networks where the internal representation depends both on the input and on the neural network's state.
- RNNs are therefore time-dependent. This makes them natural architectures for modeling processes over time such as the evolution of dynamical systems or more generally sequential data.
- RNNs can be unfolded in time, resulting in deep neural networks with a number of layers proportional to the number of time steps, and shared parameters between the multiple layers.
- In practice, RNNs are hard to train due to the vanishing/exploding gradient problem. A powerful extension of RNNs that exhibits higher stability is the LSTM.

References

References

- Bengio, Y., P. Simard, and P. Frasconi (1994). "Learning long-term dependencies with gradient descent is difficult". In: *IEEE Transactions on Neural Networks.*
- Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation.*
- Ijspeert, A.J., Alessandro Crespi, and jean-marie Cabelguen (2005). "Simulation and Robotics Studies of Salamander Locomotion: Applying Neurobiological Principles to the Control of Locomotion in Robots". In: *Neuroinformatics.*
- Karpathy, Andrej and Li Fei-Fei (2015). "Deep visual-semantic alignments for generating image descriptions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition.*
- Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2013). "On the difficulty of training recurrent neural networks". In: *International Conference on Machine Learning.*
- Wu, Yonghui et al. (2016). "Google's neural machine translation system: Bridging the gap between human and machine translation". In: *arXiv preprint arXiv:1609.08144.*