Recurrent Neural Nets

Michael Scherer

Deep Learning Reading Group

June 1, 2017

Contents

Contents

Feedforward Neural Nets

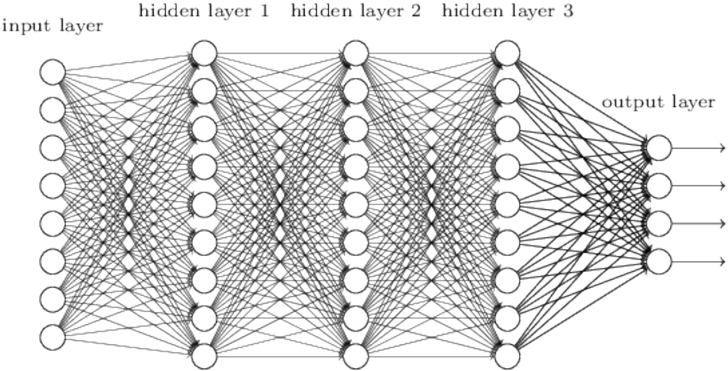


Figure: A fully connected Feedforward Neural Network. Taken from [1].

[1] Michael A. Nielsen. *Neural Networks and Deep Learning*. http://neuralnetworksanddeeplearning.com/index.html. Determination Press, 2015.

Deep Convolutional Neural Nets

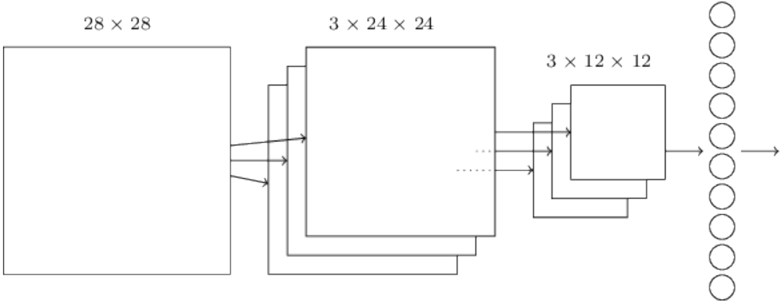


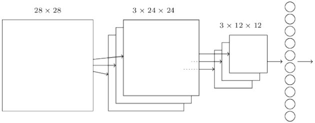
Figure: General Architecture of a Convolutional Neural Network. Taken from [1].

[1] Nielsen, *Neural Networks and Deep Learning*.

Sequential Data

Deep Convolutional Nets

Grid of Values



Recurrent Neural Nets (RNNs)

Sequence of Values

$$\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}$$

consisting of vectors	$\boldsymbol{x}^{(t)}$
for time step index	t
ranging from	1 to τ

Central Concept: Parameter Sharing

"I went to Nepal in 2009."

"In 2009, I went to Nepal."

Central Concept: Parameter Sharing

Feedforward Network

- separate parameter for each input feature
- learn the rules of the language for each position separately

Recurrent Neural Network

- same weights for different positions
- parameter sharing

Contents

Unfolding Computational Graphs

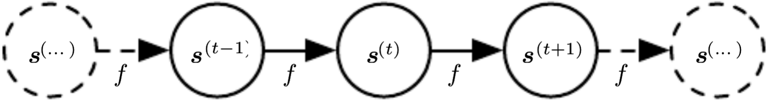


Figure: An Directed Acyclic Graph taken from [2]

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

Unfolding Computational Graphs

$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}; \boldsymbol{\theta})$$

Unfolding for $\tau = 3$ time steps

$$\begin{aligned}\boldsymbol{s}^{(3)} &= f(\boldsymbol{s}^{(2)}; \boldsymbol{\theta}) \\ &= f(f(\boldsymbol{s}^{(1)}; \boldsymbol{\theta}); \boldsymbol{\theta})\end{aligned}$$

Circuit Diagram vs. Unfolded Computational Graph

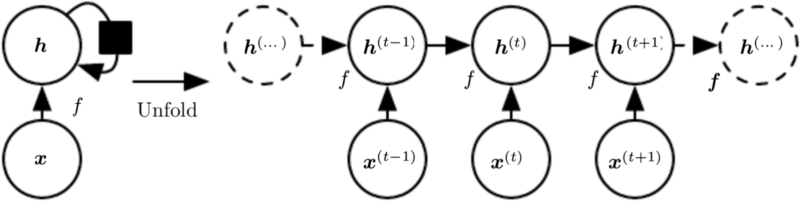


Figure: Circuit diagram (left) vs. unfolded computational graph (right). The black square indicates a time delay of 1 time step. Taken from [3].

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})$$

[3] Goodfellow, Bengio, and Courville, *Deep Learning*.

Advantages of Unfolding

- Independent of the input sequence length, because the next step is always a function of the prior step.
- At each time step, the same transition function f can be used.

Contents

Design Patterns

Three important design patterns of RNNs exist

- **Type I** Produce an output at each time step & recurrent connections between hidden units
- **Type II** Produce an output at each time step & recurrent connections only from the outputs at one time step to the hidden units of the next
- **Type III** Produce a single output & recurrent connections between hidden units

Design Patterns

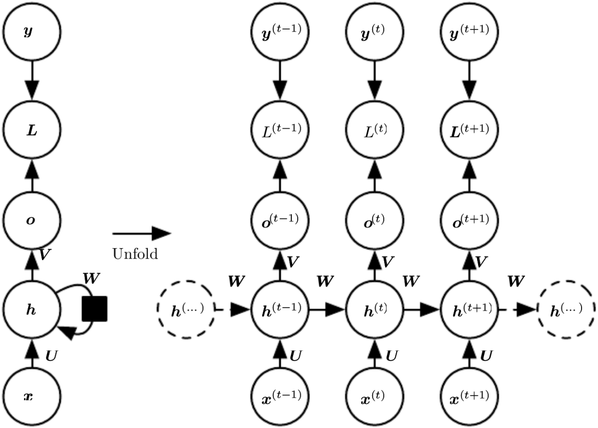


Figure: A TypeI RNN. Taken from [3].

[3] Goodfellow, Bengio, and Courville, *Deep Learning*.

Design Patterns

Three important design patterns of RNNs exist

- **Type I** Produce an output at each time step & recurrent connections between hidden units
- **Type II** Produce an output at each time step & recurrent connections only from the outputs at one time step to the hidden units of the next
- **Type III** Produce a single output & recurrent connections between hidden units

Design Patterns

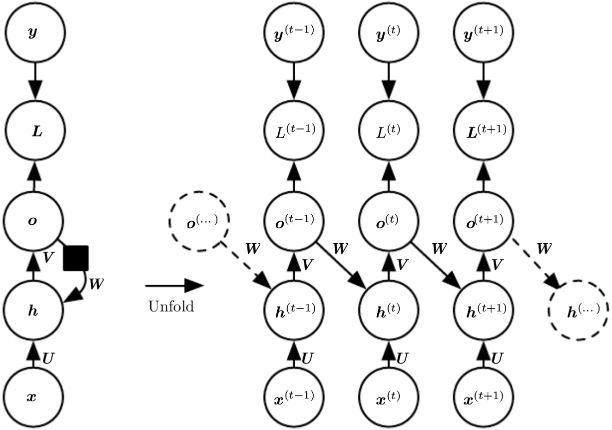


Figure: A TypeII RNN. Taken from [3].

---

[3] Goodfellow, Bengio, and Courville, *Deep Learning*.

Design Patterns

Three important design patterns of RNNs exist

Type I Produce an output at each time step & recurrent connections between hidden units

Type II Produce an output at each time step & recurrent connections only from the outputs at one time step to the hidden units of the next

Type III Produce a single output & recurrent connections between hidden units

Design Patterns

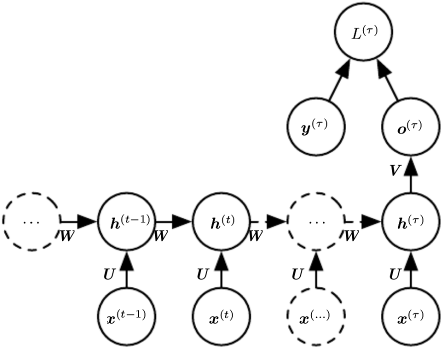


Figure: A TypeIII RNN. Taken from [3].

[3] Goodfellow, Bengio, and Courville, *Deep Learning*.

Design Patterns

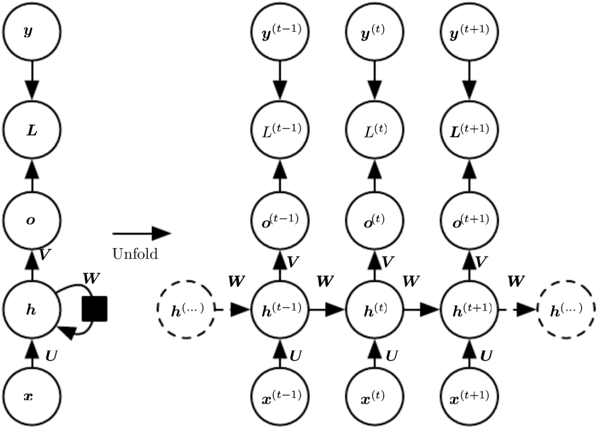


Figure: A TypeI RNN. Taken from [3].

[3]Goodfellow, Bengio, and Courville, *Deep Learning*.

Update Functions

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}$$

$\boldsymbol{a}$: update function

$\boldsymbol{b}$: bias vector

$\boldsymbol{W}$: weight matrix for hidden-to-hidden connections

$\boldsymbol{U}$: weight matrix for input-to-hidden connections

Update Functions

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}$$
$$\boldsymbol{h}^{(t)} = tanh(\boldsymbol{a}^{(t)})$$

$\boldsymbol{a}$: update function

$\boldsymbol{b}$: bias vector

$\boldsymbol{W}$: weight matrix for hidden-to-hidden connections

$\boldsymbol{U}$: weight matrix for input-to-hidden connections

Update Functions

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}$$
$$\boldsymbol{h}^{(t)} = tanh(\boldsymbol{a}^{(t)})$$
$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}$$

$\boldsymbol{c}$: bias vector

$\boldsymbol{V}$: weight matrix for hidden-to-output connections

Update Functions

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= softmax(\boldsymbol{o}^{(t)})
\end{aligned}
$$

Loss Function

$$
\begin{aligned}
&L(\{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(\tau)}\}, \{\boldsymbol{y}^{(1)}, ..., \boldsymbol{y}^{(\tau)}\}) \\
&= \sum_t L^{(t)} \\
&= -\sum_t \log p_{model}(y^{(t)} | \{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(t)}\})
\end{aligned}
$$

Teacher Forcing

Three important design patterns of RNNs exist

Type II Produce an output at each time step & recurrent connections only from the outputs at one time step to the hidden units of the next

Teacher Forcing

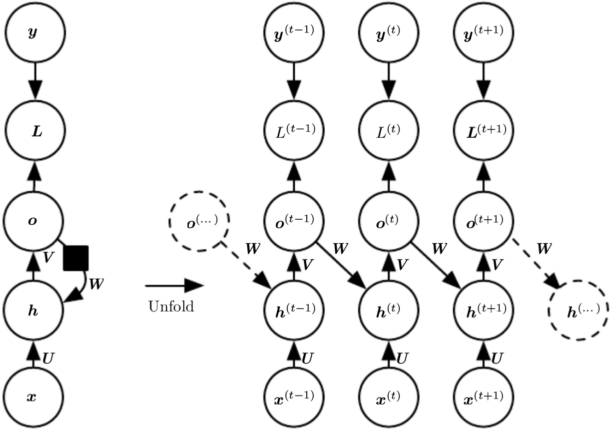


Figure: A TypeII RNN lacking hidden-to-hidden connections. Taken from [3].

[3] Goodfellow, Bengio, and Courville, *Deep Learning*.

Teacher Forcing

	No hidden-to-hidden connections
+	predictions at time steps are decoupled
+	parallelization possible
-	output units need to capture information about the past
-	strictly less powerful

Models with recurrent connections from their outputs leading back into the model can be trained by *teacher forcing*.

Teacher Forcing

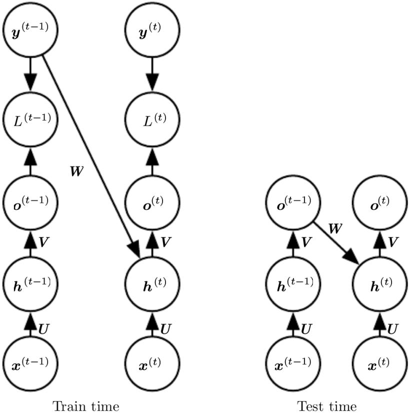


Figure: Illustration of teacher forcing. At training time, the *correct output* is used as input, whereas, at testing time, the model's output is used. Taken from [4].

[4] Goodfellow, Bengio, and Courville, *Deep Learning*.

Back-Propagation Through Time

- Application of the generalized back-propagation algorithm to the unrolled computational graph
- Application to the unrolled graph is called *back-propagation through time (BPTT)* algorithm

Gradient in TypeI RNNs

$$\nabla_{h^{(t)}} L = \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}} \right)^T (\nabla_{h^{(t+1)}} L) + \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \right)^T (\nabla_{o^{(t)}} L)$$

$$= \underbrace{\boldsymbol{W}^T}_{\substack{\text{weight matrix for} \\ \text{hidden-to-hidden} \\ \text{connections}}} (\nabla_{h^{(t+1)}} L) \underbrace{diag\left(1 - \left(\boldsymbol{h}^{(t+1)}\right)^2\right)}_{\text{derivative of } tanh}$$

$$+ \underbrace{\boldsymbol{V}^T}_{\substack{\text{weight matrix for} \\ \text{hidden-to-output} \\ \text{connections}}} (\nabla_{o^{(t)}} L)$$

Directed Graphical Models

- When using a log-likelihood training objective, an RNN is trained to estimate the conditional distribution of the next sequence element $y^{(t)}$ given the past inputs
- Maximize the log-likelihood

$$\log p(\boldsymbol{y}^{(t)}|\{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(t)}\})$$

- With connections from outputs at one time step to the next time step

$$\log p(\boldsymbol{y}^{(t)}|\{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(t)}\}, \{\boldsymbol{y}^{(1)}, ..., \boldsymbol{y}^{(t-1)}\})$$

Directed Graphical Models

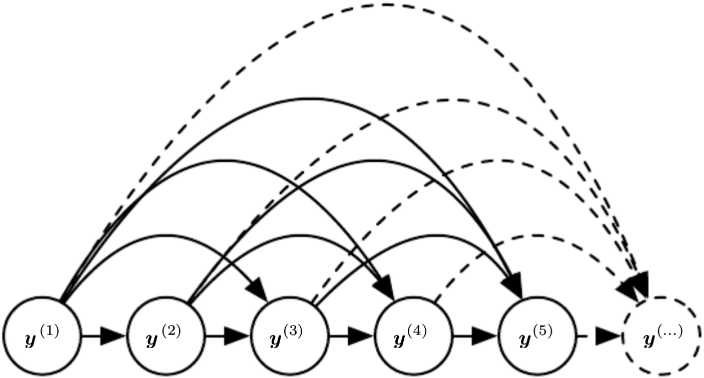


Figure: Fully connected graphical model representing influences from every past y. Taken from [5].

[5] Goodfellow, Bengio, and Courville, *Deep Learning*.

Directed Graphical Models

- Markov assumption to achieve computational efficiency, only edges from $\{y^{(t-k)}, ..., y^{(t-1)}\}$ to $y^{(t)}$
- RNNs provide efficient parametrization of the joint distribution
- Variable $y^{(i)}$ in the past may influence a variable $y^{(t)}$ via its effects on $\boldsymbol{h}$
- **But:** conditional probability distribution over the variables at time $t-1$ given the variables at time t is *stationary*

Directed Graphical Models

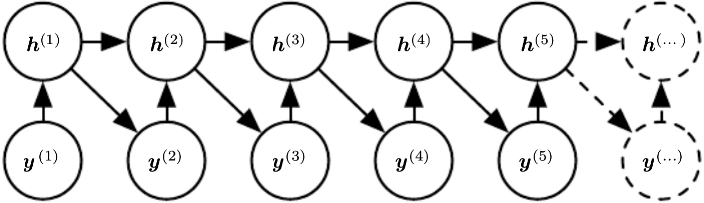


Figure: Introducing the state variable helps to see how an efficient parametrization can be obtained. Taken from [6].

[6] Goodfellow, Bengio, and Courville, *Deep Learning*.

Modeling Sequences Conditioned on Context

- Switching from modeling joint distribution over the y variables to conditional distribution over y given $\boldsymbol{x}$
- $P(\boldsymbol{y}; \boldsymbol{\theta})$ can be reinterpreted as $P(\boldsymbol{y}|\boldsymbol{\omega})$ with $\boldsymbol{\omega} = \boldsymbol{\theta}$
- This can be extended to represent a distribution $P(\boldsymbol{y}|\boldsymbol{x})$ with, again, $P(\boldsymbol{y}|\boldsymbol{\omega})$ and making $\boldsymbol{\omega}$ a function of $\boldsymbol{x}$

Modeling Sequences Conditioned on Context

Three possibilities to provide an extra input $\boldsymbol{x}$

- as an extra input at each time step
- as the initial state $\boldsymbol{h}^{(0)}$
- both

Modeling Sequences Conditioned on Context

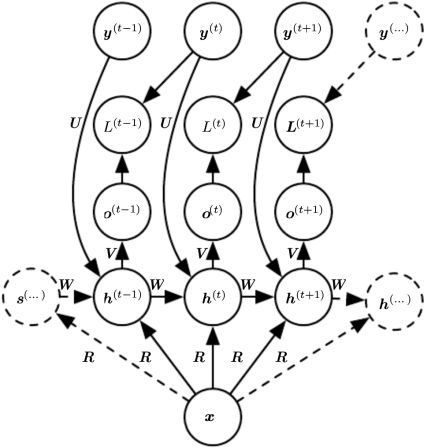


Figure: An RNN mapping a fixed-length vector into the distribution. Taken from [6].

[6] Goodfellow, Bengio, and Courville, *Deep Learning*.

Modeling Sequences Conditioned on Context

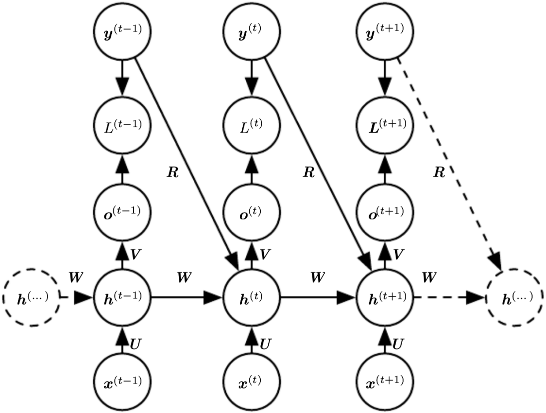


Figure: An RNN mapping a variable-length vector into the distribution. Taken from [7].

[7] Goodfellow, Bengio, and Courville, *Deep Learning*.

Contents

Bidirectional RNNs

- Instead of only considering 'past' states, predictions dependent on the **whole input sequence** could be of interest
- Example: DNA sequences
- Bidirectional RNNs combine an RNN that moves forward in time and one that moves backward
  - $\boldsymbol{h}^{(t)}$ moves forward through time
  - $\boldsymbol{g}^{(t)}$ moves backward through time

Bidirectional RNNs

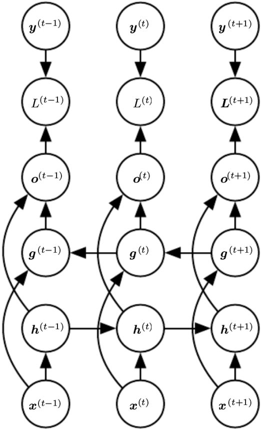


Figure: A typical representation of a bidirectional RNN. Taken from [7].

[7] Goodfellow, Bengio, and Courville, *Deep Learning*.

Contents

Encoder-Decoder Architecture

- Map input- to an output-sequences which are not necessarily of the same length
- In contrast to previous model, the lengths n_x and n_y are not required to fulfill $n_x = n_y = \tau$
- Encoder produces an fixed-length vector C which is input to the Decoder
- Major limitation occurs when context C it too short to summarize a long input sequence

Encoder-Decoder Architecture

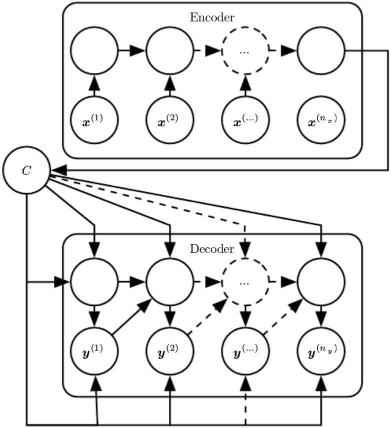


Figure: Visualization of the Encoder-Decoder Architecture. Taken from [7].

[7] Goodfellow, Bengio, and Courville, *Deep Learning*.

Conclusions

- Recurrent Neural Nets are specialized for sequential (e.g. time series) data
- Unfolding enables parameter sharing
- Teacher Forcing enables parallelization of the computation
- Bidirectional RNNs are able to incorporate information from the whole input sequence
- Encoder-Decoder Architecture enables generating outputs of different lengths as the inputs

References

Nielsen, Michael A. *Neural Networks and Deep Learning*. http://neuralnetworksanddeeplearning.com/index.html. Determination Press, 2015.

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

URL: https://en.wikipedia.org/wiki/Softmax_function.

Thank you for your attention!

Appendix

Softmax Function

The softmax function is used to highlight the largest value of a vector and suppress value which are significantly below the maximum [7].

$$\sigma(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ for } j = 1, ..., K$$

[7] URL: https://en.wikipedia.org/wiki/Softmax_function.