Sequence Modeling: Recurrent Neural Networks

Yagmur Gizem Cinar, James Crowley, (Eric Gaussier)

Intelligent Systems: Reasoning and Recognition

20 March 2018

Reference Book

Deep Learning
Ian Goodfellow and Yoshua Bengio and Aaron Courville
MIT Press
2016

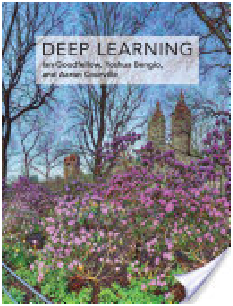


Table of Contents

Examples of Sequence Tasks

- Speech Recognition

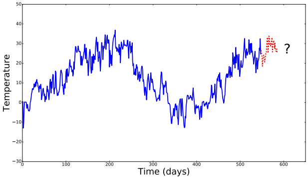
  Speech $\rightarrow$ text
- Music generation

  $\emptyset \rightarrow$ notes
- Sentiment Analysis

  "It's a good purchase, I would recommend to a friend." $\rightarrow * * * * *$
- Machine translation

  "the cat is on the rug" $\rightarrow$ "le chat est sur le tapis"
- Named entity recognition

  "Hawking was born in Oxford" $\rightarrow$ "Hawking was born in Oxford"
- Time series forecasting



Recurrent Neural Networks (RNNs)

- Specialized for processing a sequence $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\tau)}$
- RNNs can scale to longer sequence than networks without sequence-based specialization
- Parameter sharing across different parts of a model enables to extend to different forms and generalize
- For RNNs sharing parameters across time and generalize to different length of sequences
- Example:

  "I went to Nepal in 2009"
  "In 2009, I went to Nepal"
- A feedforward network for a fixed sized sentence can learn rules separately at each position
- A RNN share same weights across several time steps

Recurrent Neural Networks (RNNs)

- Parameter sharing with the convolution across 1-D temporal sequence
  - Basis for time-delay networks
  - Parameter sharing across time but shallow
  - Output is a function of neighbouring members
  - Using same convolution kernel at each time step
- For RNNs output is a function of previous members of output
- Output members are produced using the same update rule
- Recurrent parameter sharing leads to a deep computational graph
- A sequence $\mathbf{x}^{(t)}$
  - Time index $t \in [1, \ldots, \tau]$
  - In practice, minibatches of sequences with different length τ
  - Time might refer to a position in the sequence

Unfolding Computational Graphs

A computational graph formalizes the structure of a set of computations including mapping inputs and parameters to outputs and loss

unfolding a recurrent/recursive computation to computational graph with a repetitive structure

Classical form of dynamical system:

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

where $\mathbf{s}^{(t)}$ is the state of the system

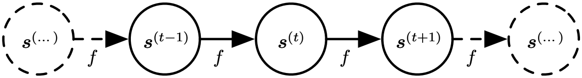


Figure 1: Unfolded computational graph of classical dynamical system[1].

[1] I Goodfellow, Y Bengio, and A Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

Unfolding Computational Graphs

Classical form of dynamical system:

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

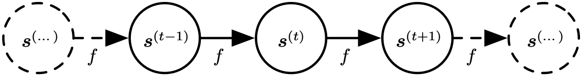


Figure 2: Unfolded computational graph of classical dynamical system[2].

For finite τ time steps, we can unfold by the same definition $\tau - 1$ times
For $\tau = 3$ time steps:

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

[2] I Goodfellow, Y Bengio, and A Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

Unfolding Computational Graphs

Another dynamical system driven by an external signal $\mathbf{x}^{(t)}$

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

Any function with recurrence can be considered a recurrent network
Rewriting above equation using variable $\mathbf{h}$, hidden units

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

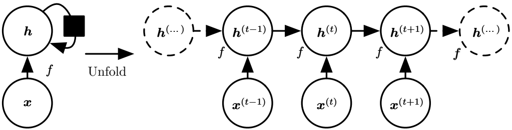
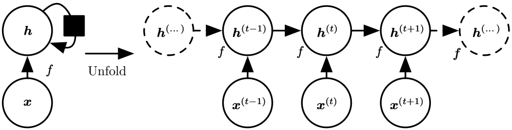


Figure 3: An RNN with no output[3].

where has information about the whole sequence

[3] I Goodfellow, Y Bengio, and A Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

Unfolding Computational Graphs



where

- has information about the whole sequence
- Circuit diagram (left)
- e.g. biological neural network
- Black square indicates a delay of a single time step, from state t to $t + 1$
- Unfolded graph (right) maps a circuit to a computational graph with repeated parts
- Unfolded graph size depends on the sequence length

Unfolding Computational Graphs

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

For a task requiring predicting the future from the past,

- $\mathbf{h}^{(t)}$ becomes a kind of lossy summary
- $\mathbf{h}^{(t)}$ is a fixed-length vector mapping from arbitrary length sequence $(\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}, \ldots, \mathbf{x}^{(1)})$
- Depending on the training criterion, selectively keep some aspects
- Ex: statistical language modeling predict next word given previous words
- Most challenging recovering input sequence from $\mathbf{h}^{(t)}$, e.g. autoencoders

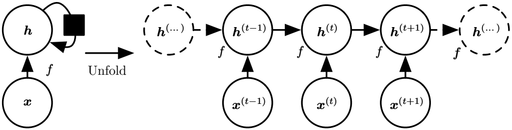
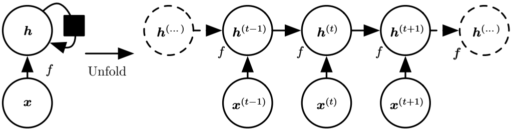


Figure 4: An RNN with no output[4].

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

Unfolding Computational Graphs

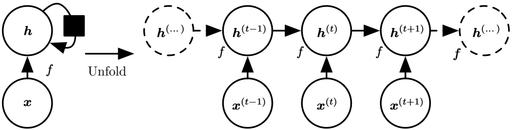


- Unfolded recurrence after t steps with a function $g^{(t)}$

$$
\begin{aligned}
\mathbf{h}^{(t)} &= g^{(t)}(\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}, \dots, \mathbf{x}^{(2)}, \mathbf{x}^{(1)}) \\
&= f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \theta)
\end{aligned}
$$

- $g^{(t)}$ takes whole past sequence $(\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}, \dots, \mathbf{x}^{(2)}, \mathbf{x}^{(1)})$ and produce current state
- Unfolded recurrent structure factorize $g^{(t)}$ into repeated f

Unfolding Computational Graphs



- Unfolding process:

  + Regardless of sequence length model has fixed input size (since specified in terms of state transitions)
  + Use of same transition function f with the same parameters at every time step

- These two factors enables to learn a single shared model

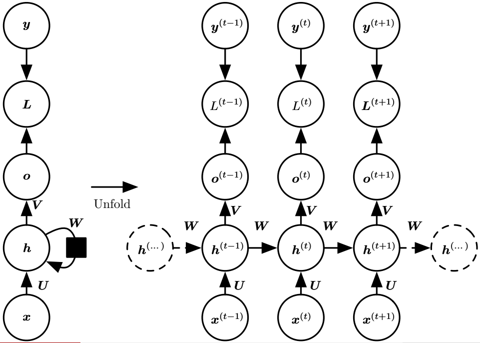
  Generalization to sequence lengths not observed in training
  Able to train with fewer examples required without than parameter sharing

- Recurrent graph is succinct
- Unfolded graph is explicit and illustrates information flow in forward and backward in time

Recurrent Neural Networks

Important design patterns:

1. An output at each time step and recurrent connections between hidden units
2. An output at each time step and recurrent connections only from the output at one time step to hidden units at the next time step
3. Recurrent connections between hidden units and a single output after reading the entire sequence



Recurrent Neural Networks

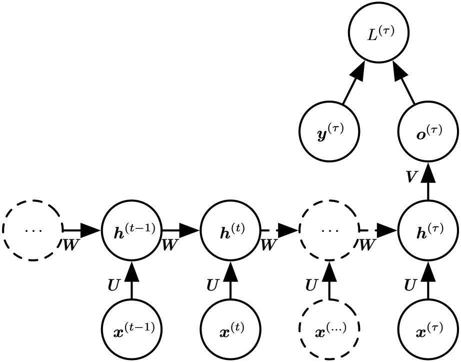


Figure 6: Time-unfolded RNN with a single output at the end[6].

[6] I Goodfellow, Y Bengio, and A Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

Recurrent Neural Networks

An RNN outputs at each time step with recurrent connections between hidden units

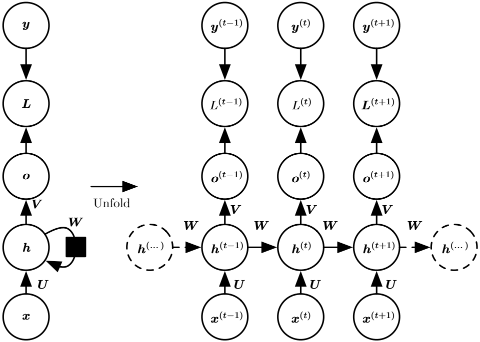


Figure 7: Recurrent hidden units[7].

[7] I Goodfellow, Y Bengio, and A Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

Recurrent Neural Networks

An RNN outputs at each time step with recurrent connections between hidden units

$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)}$$
$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$$
$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)}$$
$$\hat{\mathbf{y}}^{(t)} = \text{softmax}(\mathbf{o}^{(t)})$$

b, **c** are biases, weight matrices **U** (input-to-hidden), **W** (hidden-to-hidden), **V** (hidden-to-output)

Total loss between sequence of **x** and corresponding sequence of **y**:

$$L\left(\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\tau)}\}, \{\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(\tau)}\}\right)$$
$$= \sum_t L^{(t)}$$
$$= -\sum_t \log p_{\text{model}}\left(y^{(t)} | \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)}\}\right)$$

Recurrent Neural Networks

An RNN outputs at each time step with recurrent connections between hidden units

$$L\left(\{\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(\tau)}\},\{\mathbf{y}^{(1)},\ldots,\mathbf{y}^{(\tau)}\}\right)=\sum_t L^{(t)}$$
$$=-\sum_t \log p_{\text{model}}\left(y^{(t)}|\{\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(t)}\}\right)$$

$L^{(t)}$ negative log-likelihood of $y^{(t)}$ given $\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(t)}$
Computing gradient of this loss function expensive

- Forward propagation pass, backward propagation pass
- Runtime $\mathcal{O}(\tau)$ and cannot be reduced with parallelization
- Memory $\mathcal{O}(\tau)$
- Back-propagation applied to unrolled graph with $\mathcal{O}(\tau)$ **back-propagation through time** (BPTT)

Recurrent Neural Networks

An RNN outputs at each time step with recurrent connections from output to next step hidden units

- strictly less powerful without hidden-to-hidden recurrence
- output has to capture all past history

\+ training can be parallelized with gradient computed in isolation

For training **teacher forcing** can be used

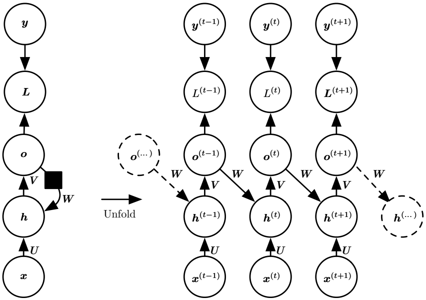


Figure 8: An RNN with recurrence connection from output to hidden[8].

Teacher Forcing

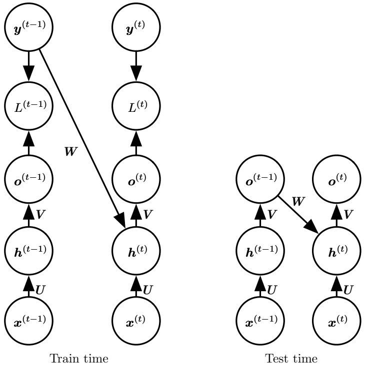


Figure 9: Illustration of teacher forcing[9].

[9] I Goodfellow, Y Bengio, and A Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

Bidirectional RNNs

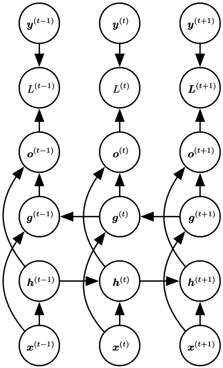


Figure 10: Bidirectional RNN[10].

[10] I Goodfellow, Y Bengio, and A Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

Bidirectional RNNs

- A causal structure, state at time t captures only past
- Many applications output $\mathbf{y}^{(t)}$ after processing the whole input sequence
  e.g. in speech recognition, handwriting recognition
- Bidirectional RNN combines RNN moves forward through time $\mathbf{h}$ and backward through time $\mathbf{g}$

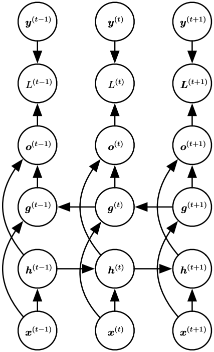


Figure 11: Bidirectional RNN[a].

[a] I Goodfellow, Y Bengio, and A Courville. *Deep L*
http://www.deeplearningbook.org. MIT Press, 20

Drawing samples from an RNN model

Sampling from conditional distribution at each time step
How to determine the length of the sequence:

- If output is a symbol from a vocabulary, having a special symbol indicate the end of a sequence
- An extra Bernoulli output to decide to continue or halt
  - E.g. RNN produces a sequence of real numbers, new output unit usually sigmoid with cross entropy loss.
  - Sigmoid maximize log-probability of the sequence ends or not
- An extra output to determine the sequence length τ
  - An extra output predicts the integer τ
  - E.g. sample an integer τ and then sample τ steps of data
  - Here, RNN needs an extra input consist of value of τ or number of remaining steps $\tau - t$
  - Extra input to avoid abrupt ending sequence

$$P(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\tau)}) = P(\tau)P(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\tau)} \mid \tau)$$

Modeling Sequences Conditioned on Context with RNNs

RNNs with a single vector of $\mathbf{x}$ as input instead of sequence of vectors $\mathbf{x}^{(t)}$
Providing an extra input to an RNN:

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

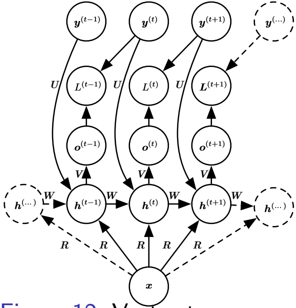


Figure 12: Vector to sequence[11].

[11] I Goodfellow, Y Bengio, and A Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

Modeling Sequences Conditioned on Context with RNNs

An input sequence of $\mathbf{x}^{(t)}$ instead of a single input
Conditional distribution of $P(\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(\tau)} \mid \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\tau)})$ makes a conditional independence assumption

$$\prod_t P(\mathbf{y}^t | \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)})$$

By adding connections from output at time t to hidden unit at time $t+1$:
Hence, the output values are not forced to be conditionally independent for this model:

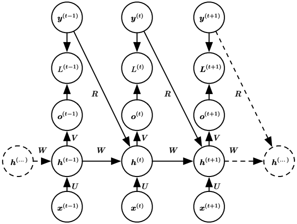


Figure 13: Hidden and output recurrence[12].

Encoder-Decoder Sequence-to-Sequence Architectures

An RNN map an input sequence to an output sequence which is not necessarily the same length

e.g. speech recognition, machine translation, question answering

The input to RNN is called the context C, summarize the input sequence $\mathbf{X} = (\mathbf{x}^{(1)}, \ldots, x^{(n_x)})$

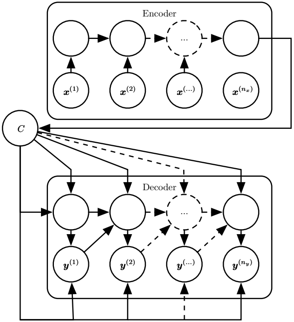


Figure 14: Sequence-to-sequence Architecture[13].

Encoder-Decoder Sequence-to-Sequence Architectures

An **encoder** or **reader** or **input** process input to the sequence
A **decoder** or **writer** or **output** conditioned on a fixed length vector to generate $Y = (\mathbf{y}^{(1)}, \ldots, y^{(n_y)})$ Two RNNs trained to jointly maximize average of $\log P(\mathbf{y}^{(1)}, \ldots, y^{(n_y)} | \mathbf{x}^{(1)}, \ldots, x^{(n_x)})$
The last state of encoder $\mathbf{h}_{n_x}$ typically used as C

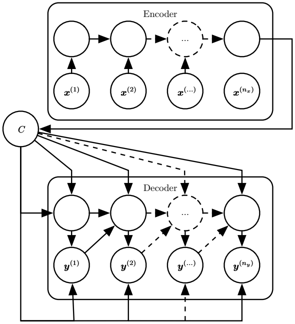


Figure 15: Sequence-to-sequence Architecture[14].

[14] [illegible]

Deep Recurrent Networks

RNNs decomposed into three main blocks of parameters and associated transformations:

- from input to hidden state
- from previous hidden state to next hidden state
- from hidden state to output

Increasing the depth of the RNNs improves

(a) hierarchical hidden recurrent states

(b) deeper computation introduced to input-to-hidden, hidden-to-hidden, and hidden-to-output.

(c) Skip connections can handle path-lengthening effect

Deep Recurrent network larger capacity of representation but might increase the difficulty of optimization

Deep Recurrent Networks

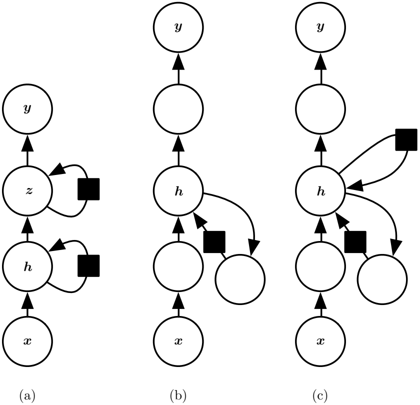


Figure 16: An RNN can be deep many ways. (a) The recurrent state organized into groups hierarchically. (b) Introduced in input-to-hidden, hidden-to-hidden, and hidden-to-output. (c) Skip connections can handle path-lengthening effect[16].

[15] I Goodfellow, Y Bengio, and A Courville. *Deep Learning*. ... MIT Press, 2016.

The Challenge of Long-Term Dependencies

RNNS might construct a very deep computational graphs by repeatedly applying the same operation at each time step of a long sequence

Gradients propagated many stages tends to vanish or explode

Even if the network is stable (can store memories and gradients not exploding), exponentially smaller weights are given to long-term interactions than short ones

Recurrence relation (for a simple network without nonlinear activation and input $\mathbf{x}$)

$$\mathbf{h}^{(t)} = \mathbf{W}\mathbf{h}^{(t-1)}$$

After t time steps (repeatedly multiplying with $\mathbf{W}$)

$$\mathbf{h}^{(t)} = (\mathbf{W}^{t})^{\mathsf{T}}\mathbf{h}^{(0)}$$

Gated RNNs

The most effective sequence models used in practical applications are **gated RNNs**

- **Long short-term memory (LSTM)**
- **gated recurrent unit (GRU)**

Like leaky units goal is to create paths through time that have derivatives do not vanish nor explode

- Leaky units has connection weights that manually chosen or learned
- Gated RNNs generalizes this to connections weights that may change at each time step
- Leaky units allows to accumulate information
- But once this information is used, it might be useful to forget
- A mechanism to forget the old state by setting it to 0
- Gated RNNs learn to decide when to forget a state

Long Short-Term Memory (LSTM)

- Core contribution of initial LSTM model[17] is self-loops to introduce paths that gradient can flow
- Gers et al. make the weight on this self-loop weight conditioned on the context[18]
- With gated weight of self-loop (controlled by another unit), time scale integration dynamically controlled
- The LSTM is very sucessful in many domains: handwriting detection and generation, time series forecasting, machine translation, speech recognition..
- LSTM recurrent networks have "LSTM cells" that have internal recurrence, a self loop, in addition to outer recurrence of RNN

[17] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780.

[18] Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins. "Learning to Forget: Continual Prediction with LSTM". In: *Neural Comput.* 12.10 (Oct. 2000), pp. 2451–2471.

Long Short-Term Memory (LSTM)

- Each LSTM cell has the same inputs and outputs with gating units controlling information flow
- State unit $s_i^{(t)}$ has a linear self-loop
- Self-loop weight of $s_i^{(t)}$ controlled by a **forget gate** unit $f_i^{(t)}$ for time step t and cell i
- $f_i^{(t)}$ sets weight of $s_i^{(t)}$ a value of (0,1) via sigmoid unit:

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right)$$

- $\mathbf{x}^{(t)}$ is the current input, $\mathbf{h}^{(t)}$ is the current hidden layer vector
- $\mathbf{b}^f$ bias, $\mathbf{U}^f$ input weights and $\mathbf{W}^f$ recurrent weights for the forget gates

Long Short-Term Memory (LSTM)

- The LSTM cell internal state update:

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma \left(b_i + \sum_j U_{i,j} \, x_j^{(t)} + \sum_j W_{i,j} \, h_j^{(t-1)} \right)$$

- **b** bias, **U** input weights and **W** recurrent weights into the LSTM cell
- The **external input gate** $g_i^{(t)}$ is calculated as forget gate but with own parameters:

$$g_i^{(t)} = \sigma \left(b_i^g + \sum_j U_{i,j}^g \, x_j^{(t)} + \sum_j W_{i,j}^g \, h_j^{(t-1)} \right)$$

- The output $h_i^{(t)}$ of the LSTM cell can also be shut off via **output gate** $q_i^{(t)}$ which also uses a sigmoid unit for gating:

$$h_i^{(t)} = \tanh \left(s_i^{(t)} \right) q_i^{(t)}$$

$$q_i^{(t)} = \sigma \left(b_i^o + \sum_j U_{i,j}^o \, x_j^{(t)} + \sum_j W_{i,j}^o \, h_j^{(t-1)} \right)$$

Long Short-Term Memory (LSTM)

Variant of LSTM uses the cell state $s_i^{(t)}$ as an extra input to the three gates, this would require three additional parameters

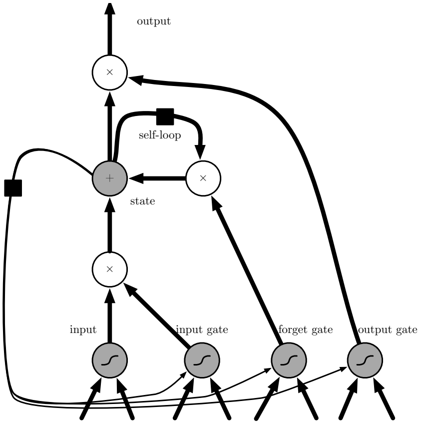


Figure 17: Block diagram of the LSTM[19].

Gated Recurrent Unit (GRU)

Comparison to LSTM there is a single gating unit simultaneously controls the forgetting factor and the decision to update the state unit

- The update equations

$$h_i^{(t)} = u_i^{(t-1)} h_i^{(t-1)} + (1 - u_i^{(t-1)})\, \sigma\left(b_i + \sum_j U_{i,j}\, x_j^{(t-1)} + \sum_j W_{i,j}\, r_j^{(t-1)} h_j^{(t-1)}\right)$$

- **u** stands for update gate:

$$u_i^{(t)} = \sigma\left(b_i^u + \sum_j U_{i,j}^u\, x_j^{(t)} + \sum_j W_{i,j}^u\, h_j^{(t)}\right)$$

- **r** stands for reset gate:

$$r_i^{(t)} = \sigma\left(b_i^r + \sum_j U_{i,j}^r\, x_j^{(t)} + \sum_j W_{i,j}^r\, h_j^{(t)}\right)$$

Gated Recurrent Unit (GRU)

Reset and update gates can individually ignore parts of the state vector

- Update gates act like conditional leaky integrator that
- Update gates linearly integrate any dimension
  - choosing to copy it or completely
  - ignore it by replacing it with new target state value
  - target state is state the leaky integrator wants to converge towards
- Reset gates control which parts of the state get used to compute the next target state
- Reset gates introduce an additional nonlinear effect in the relationship between past and future state

Optimization for Long-Term Dependencies

When the parameter gradient is very large, gradient descent parameter update could throw parameters far away

A simple solution is **clipping gradients**

- Clip gradient from a minibatch element-wise before parameter update
- Clip the norm $||\mathbf{g}||$ of gradient $\mathbf{g}$ before parameter update

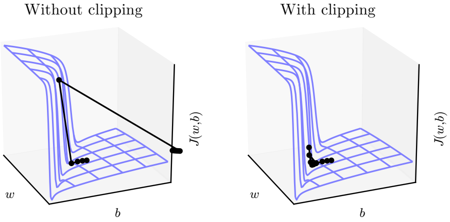


Figure 18: Gradient Clipping[20].

[20] I Goodfellow, Y Bengio, and A Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

Recurrent Neural Networks

Questions?

References

Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins. "Learning to Forget: Continual Prediction with LSTM". In: *Neural Comput.* 12.10 (Oct. 2000), pp. 2451–2471.

I Goodfellow, Y Bengio, and A Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780.