Recurrent Neural Networks

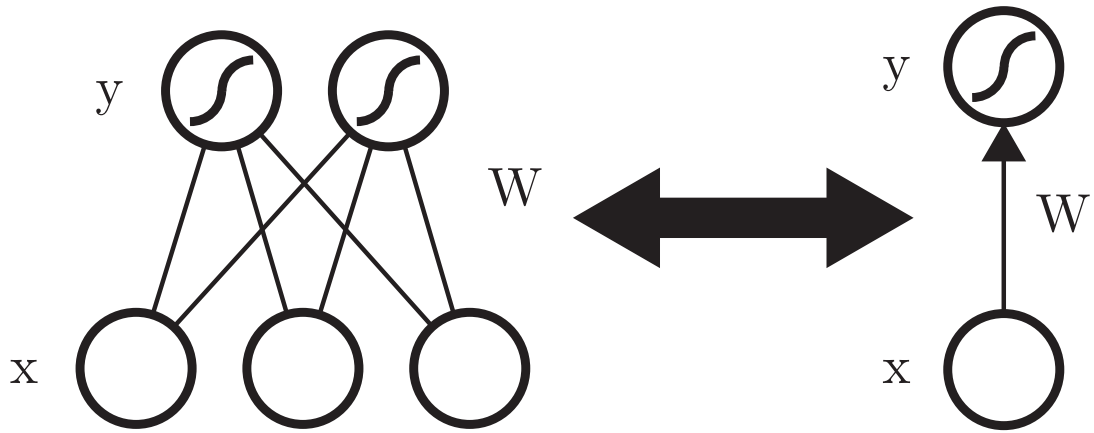
Félix Gontier, Mathieu Lagrange

felix.gontier@ls2n.fr

6 janvier 2020

Outline

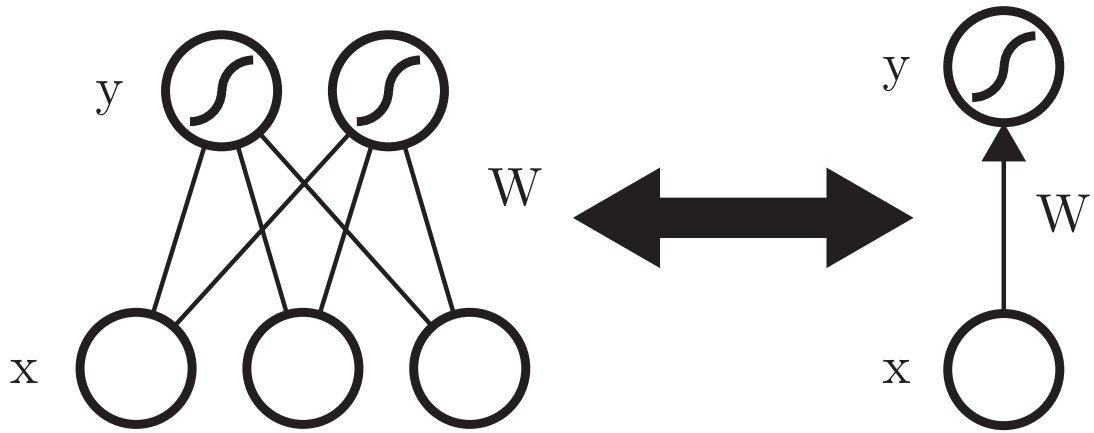
Neural Networks



- x: Input data
- y: Output data
- W: Weight matrix
- f: Activation function
- Forward propagation: $y = f(Wx + b)$

Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 1958

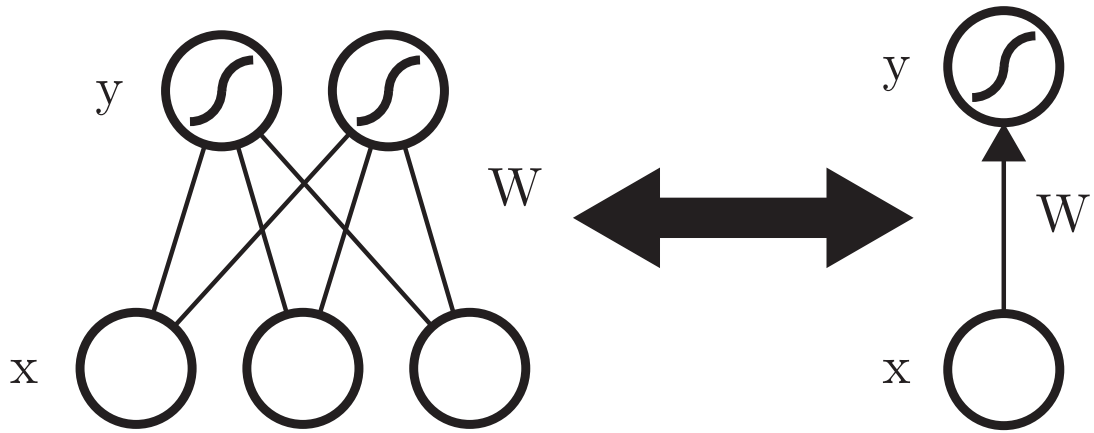
Neural Networks



- x: Input data
- y: Output data
- W: Weight matrix
- f: Activation function
- Forward propagation: $y = f(Wx + b)$

Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 1958

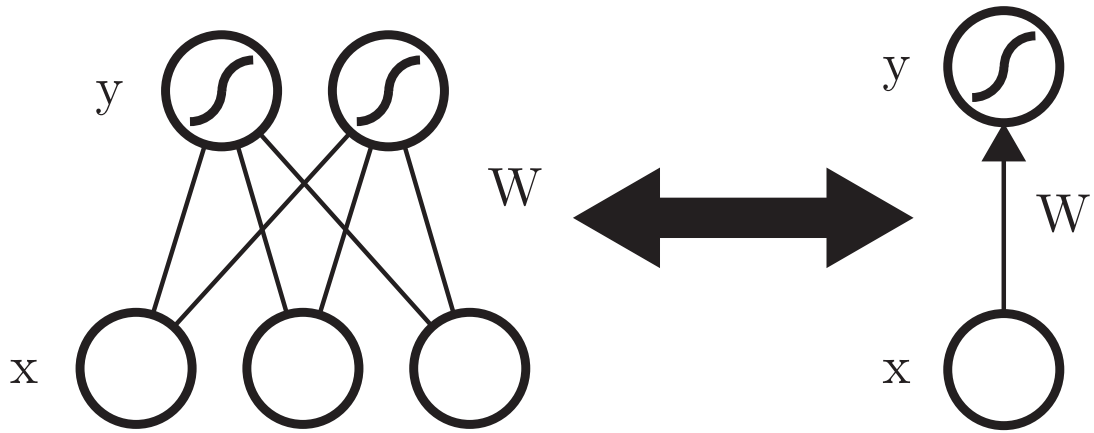
Neural Networks



- x: Input data
- y: Output data
- W: Weight matrix
- f: Activation function
- Forward propagation: $y = f(Wx + b)$

Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 1958

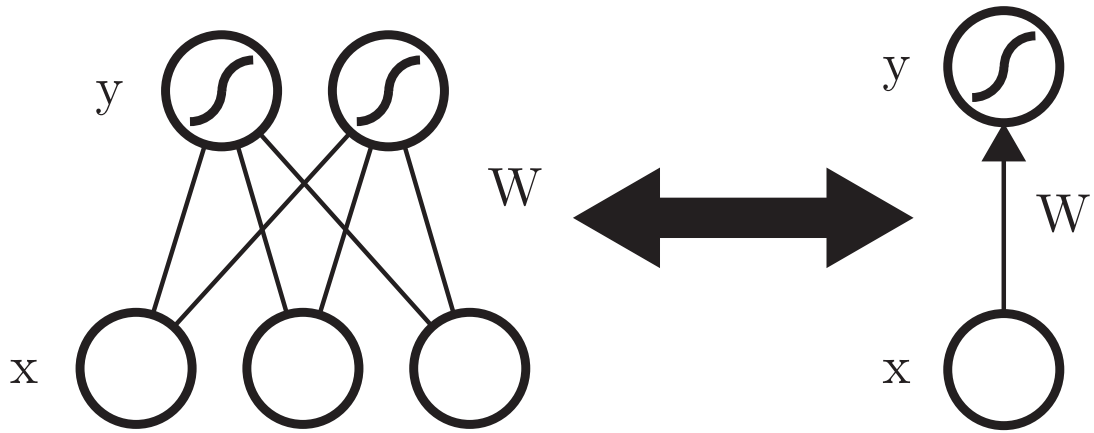
Neural Networks



- x: Input data
- y: Output data
- W: Weight matrix
- f: Activation function
- Forward propagation: $y = f(Wx + b)$

Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 1958

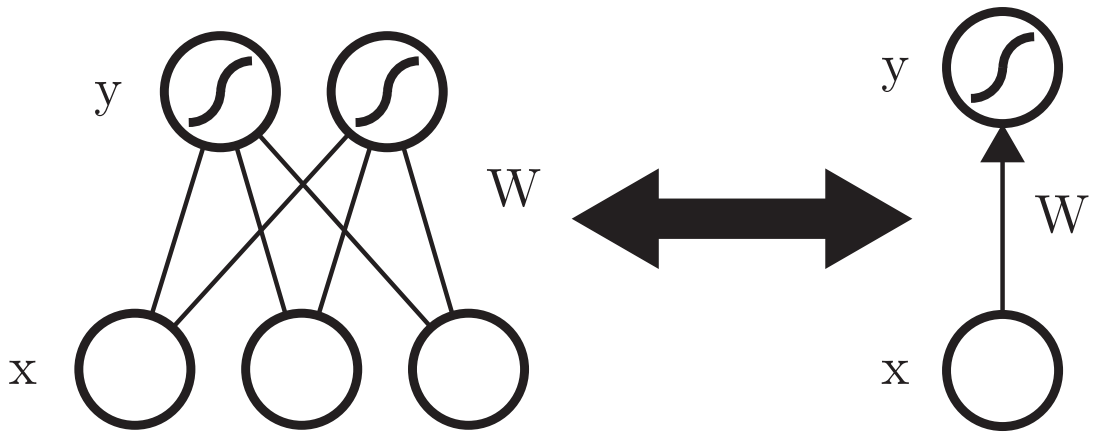
Neural Networks



- x: Input data
- y: Output data
- W: Weight matrix
- f: Activation function
- Forward propagation: $y = f(Wx + b)$

Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 1958

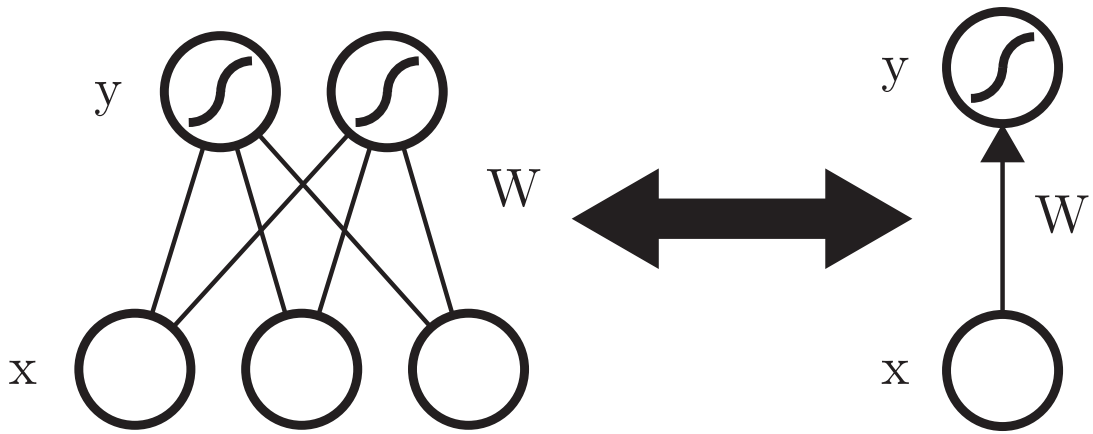
Neural Networks Backpropagation



- Loss function $L(y, \hat{y})$ (y: target output, $\hat{y}$: predicted output)
- Parameter update with Stochastic Gradient Descent (SGD): $W = W - \lambda \frac{\partial L}{\partial W}$
- Chain rule of backpropagation $\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W}$

Robbins et al. A Stochastic Approximation Method. Ann. Math. Statist., 1951

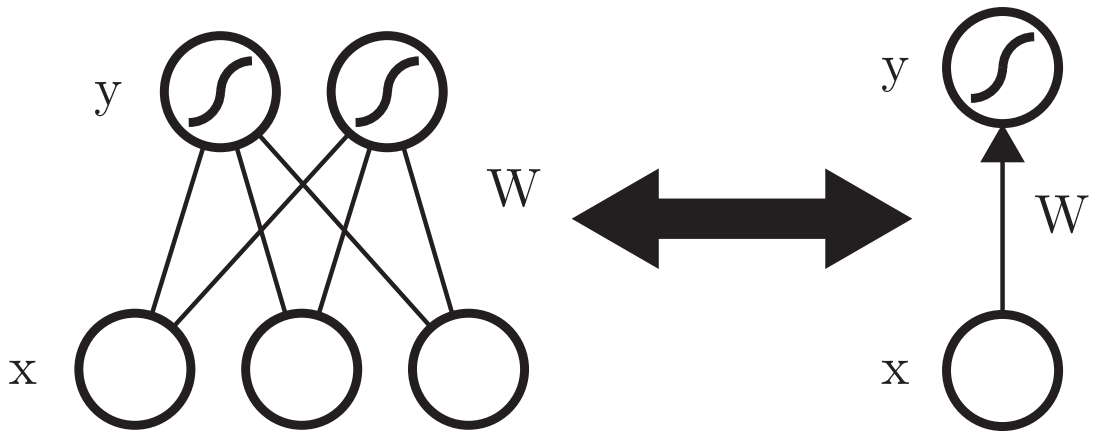
Neural Networks Backpropagation



- Loss function $L(y, \hat{y})$ (y: target output, $\hat{y}$: predicted output)
- Parameter update with Stochastic Gradient Descent (SGD): $W = W - \lambda \frac{\partial L}{\partial W}$
- Chain rule of backpropagation $\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W}$

Robbins et al. A Stochastic Approximation Method. Ann. Math. Statist., 1951

Neural Networks Backpropagation



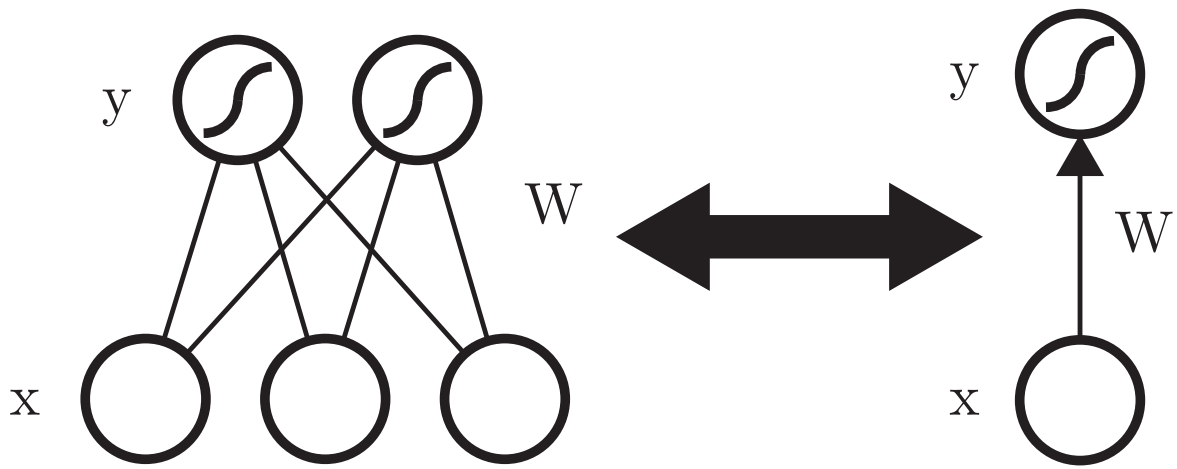
- Loss function $L(y, \hat{y})$ (y: target output, $\hat{y}$: predicted output)
- Parameter update with Stochastic Gradient Descent (SGD): $W = W - \lambda \frac{\partial L}{\partial W}$
- Chain rule of backpropagation $\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W}$

Robbins et al. A Stochastic Approximation Method. Ann. Math. Statist., 1951

Neural Networks Backpropagation: Example

Architecture hyperparameters

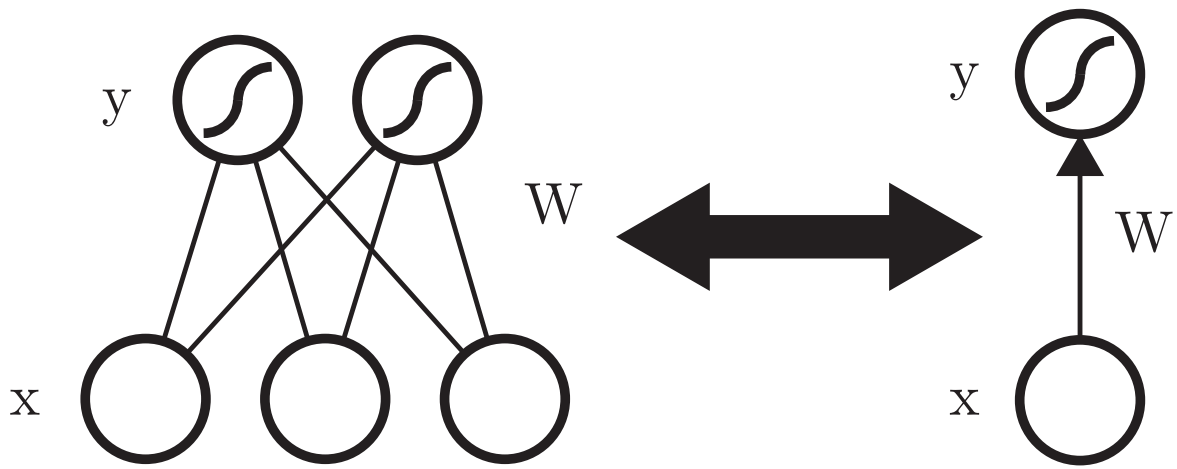
- Activation function $f = tanh(x)$
- Learning rate $\lambda = 0.1$



Neural Networks Backpropagation: Example

Architecture hyperparameters

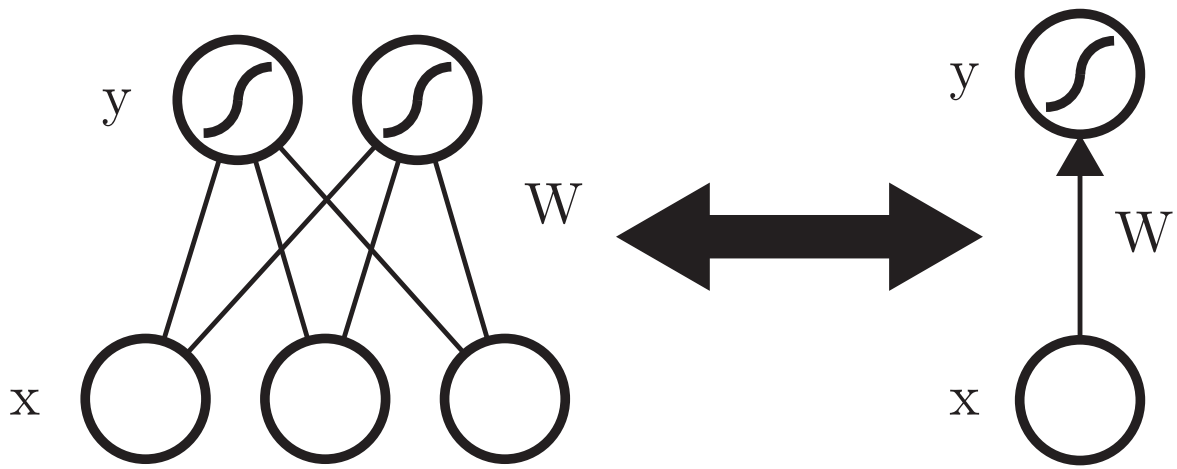
- Activation function $f = tanh(x)$
- Learning rate $\lambda = 0.1$



Neural Networks Backpropagation: Example

Training data

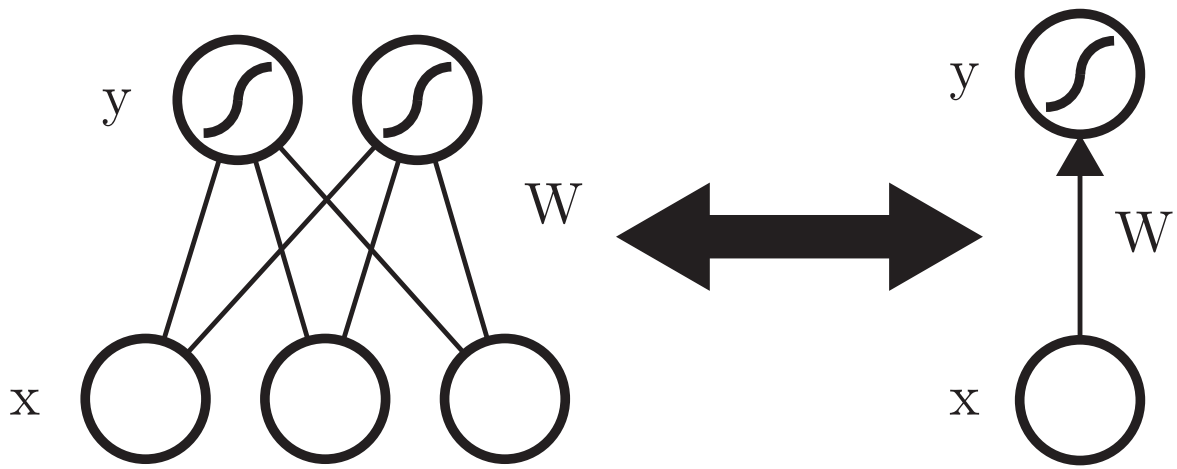
- Input $x = \begin{bmatrix} 0.1 & 0.3 & 0.2 \end{bmatrix}^T$
- Output $y = \begin{bmatrix} -0.5 & 0.5 \end{bmatrix}^T$
- Randomly initialized weight matrix $W = \begin{bmatrix} 0 & -0.1 & 0.4 \\ 0.3 & -0.4 & -0.2 \end{bmatrix}$



Neural Networks Backpropagation: Example

Training data

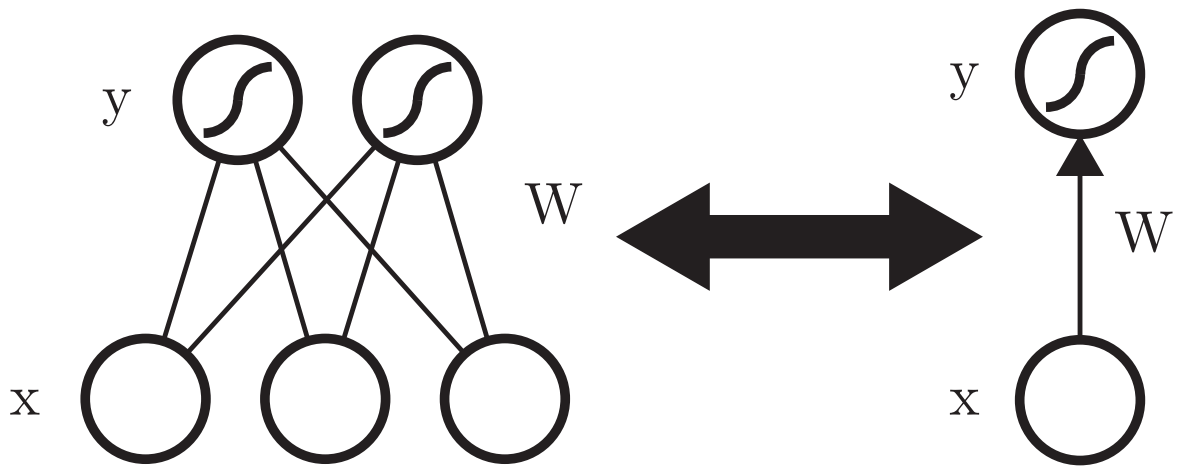
- Input $x = \begin{bmatrix} 0.1 & 0.3 & 0.2 \end{bmatrix}^T$
- Output $y = \begin{bmatrix} -0.5 & 0.5 \end{bmatrix}^T$
- Randomly initialized weight matrix $W = \begin{bmatrix} 0 & -0.1 & 0.4 \\ 0.3 & -0.4 & -0.2 \end{bmatrix}$



Neural Networks Backpropagation: Example

Training data

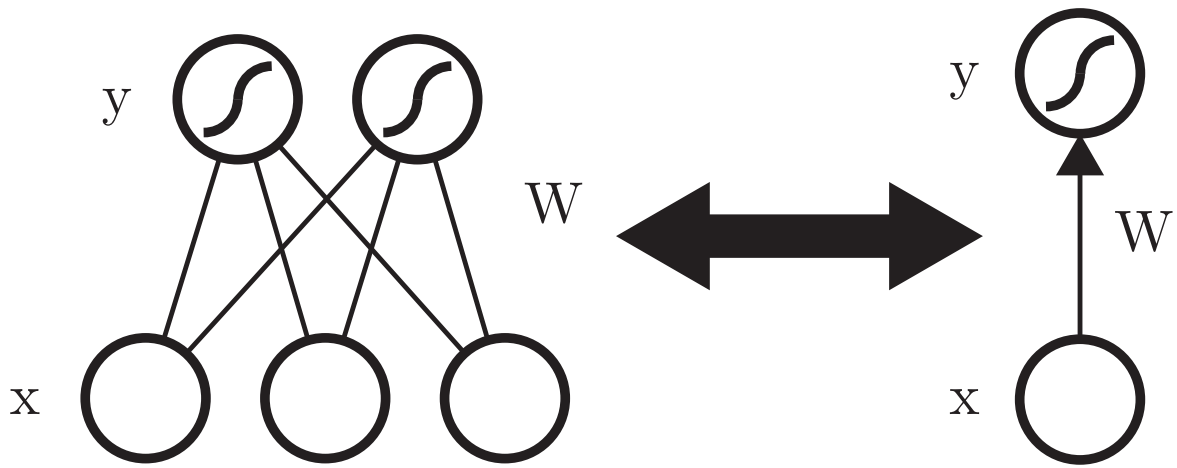
- Input $x = \begin{bmatrix} 0.1 & 0.3 & 0.2 \end{bmatrix}^T$
- Output $y = \begin{bmatrix} -0.5 & 0.5 \end{bmatrix}^T$
- Randomly initialized weight matrix $W = \begin{bmatrix} 0 & -0.1 & 0.4 \\ 0.3 & -0.4 & -0.2 \end{bmatrix}$



Neural Networks Backpropagation: Example

Optimization

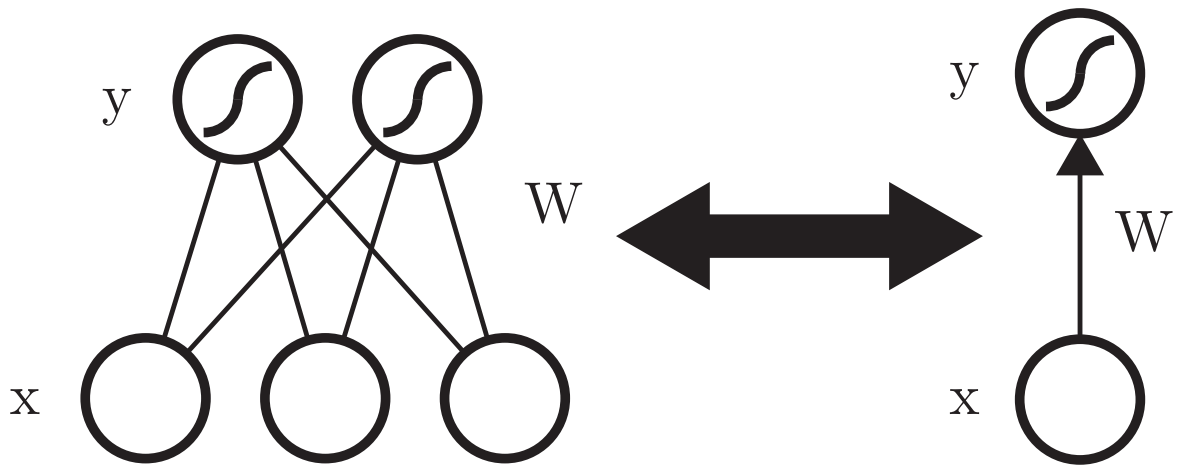
- Loss function $L(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$
- Gradient calculation $\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W} = (\hat{y} - y) \odot tanh'(Wx)x^T$



Neural Networks Backpropagation: Example

Optimization

- Loss function $L(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$
- Gradient calculation $\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W} = (\hat{y} - y) \odot tanh'(Wx)x^T$

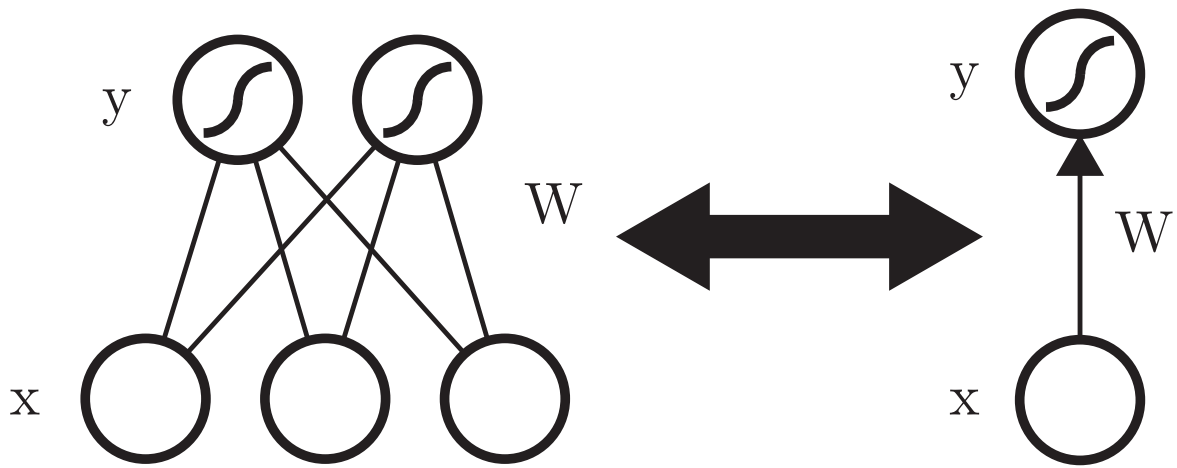


Neural Networks Backpropagation: Example

Numerical application

- Forward propagation $\hat{y} = f(Wx) = \begin{bmatrix} 0.0500 & -0.1293 \end{bmatrix}^T$
- Gradient calculation $\frac{\partial L}{\partial W} = \begin{bmatrix} 0.0549 & 0.1646 & 0.1097 \\ -0.0619 & -0.1856 & -0.1238 \end{bmatrix}$
- SGD update:

$$W = W - \lambda \frac{\partial L}{\partial W} = \begin{bmatrix} -0.0055 & -0.1165 & 0.3890 \\ 0.3062 & -0.3814 & -0.1876 \end{bmatrix}$$

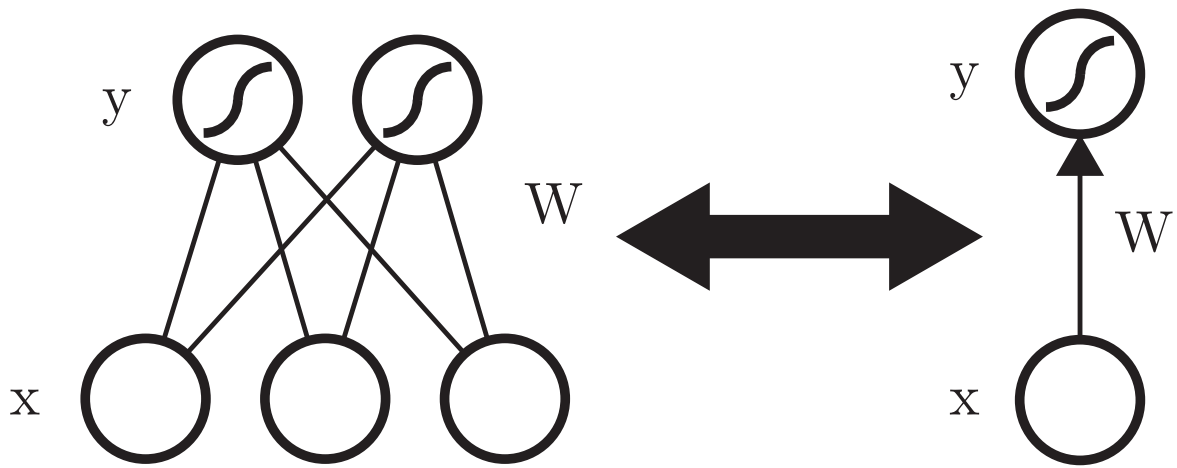


Neural Networks Backpropagation: Example

Numerical application

- Forward propagation $\hat{y} = f(Wx) = \begin{bmatrix} 0.0500 & -0.1293 \end{bmatrix}^T$
- Gradient calculation $\frac{\partial L}{\partial W} = \begin{bmatrix} 0.0549 & 0.1646 & 0.1097 \\ -0.0619 & -0.1856 & -0.1238 \end{bmatrix}$
- SGD update:

$$W = W - \lambda \frac{\partial L}{\partial W} = \begin{bmatrix} -0.0055 & -0.1165 & 0.3890 \\ 0.3062 & -0.3814 & -0.1876 \end{bmatrix}$$

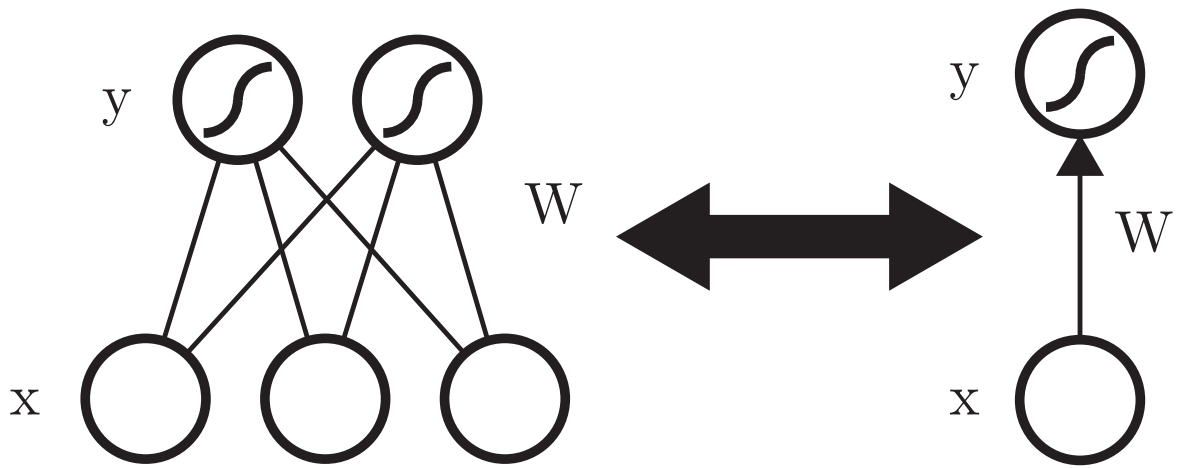


Neural Networks Backpropagation: Example

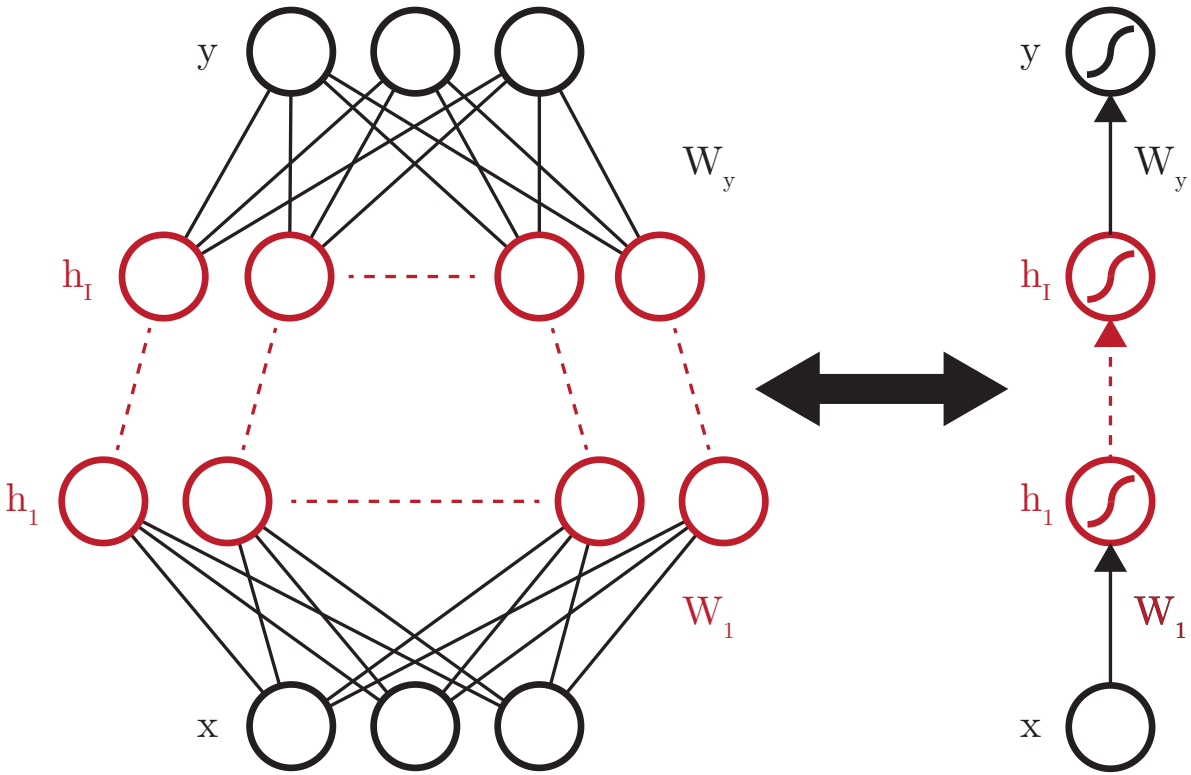
Numerical application

- Forward propagation $\hat{y} = f(Wx) = \begin{bmatrix} 0.0500 & -0.1293 \end{bmatrix}^T$
- Gradient calculation $\frac{\partial L}{\partial W} = \begin{bmatrix} 0.0549 & 0.1646 & 0.1097 \\ -0.0619 & -0.1856 & -0.1238 \end{bmatrix}$
- SGD update:

$$W = W - \lambda \frac{\partial L}{\partial W} = \begin{bmatrix} -0.0055 & -0.1165 & 0.3890 \\ 0.3062 & -0.3814 & -0.1876 \end{bmatrix}$$



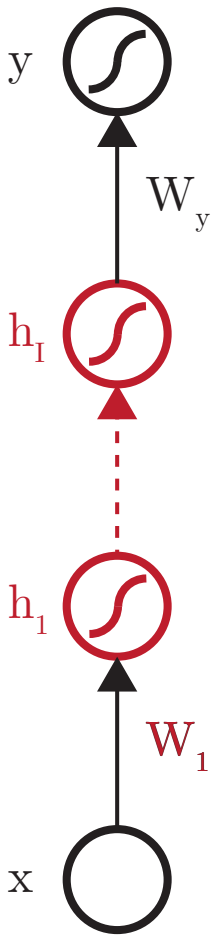
Deep Neural Networks



↪ h_i: i^{th} hidden layer with corresponding W_i weights

Le Roux et al. Deep Belief Networks Are Compact Universal Approximators. Neural Computation, 2010

Deep Neural Networks



Multilayer perceptron

$$y = f_y(W_y h_l + b_y) \quad (1)$$

$$h_i = f_i(W_i h_{i-1} + b_i), i \in [2, l] \quad (2)$$

$$h_1 = f_1(W_1 x + b_1) \quad (3)$$

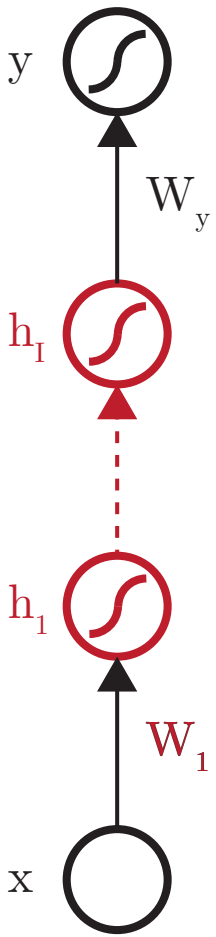
Generalization

$$y = f_{\theta_y}(h_l) \quad (4)$$

$$h_i = f_{\theta_i}(h_{i-1}), i \in [2, l] \quad (5)$$

$$h_1 = f_{\theta_1}(x) \quad (6)$$

Deep Neural Networks



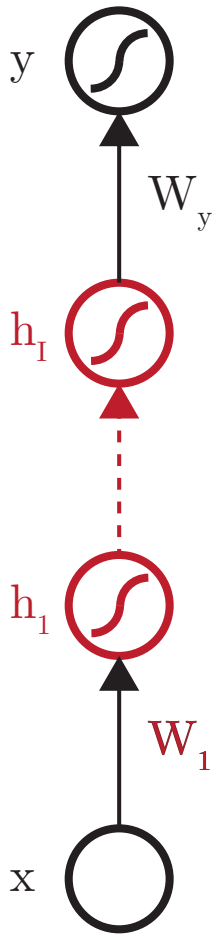
- Multilayer perceptron

$$y = f_y(W_y h_l + b_y) \quad (1)$$
$$h_i = f_i(W_i h_{i-1} + b_i), i \in [2, l] \quad (2)$$
$$h_1 = f_1(W_1 x + b_1) \quad (3)$$

- Generalization

$$y = f_{\theta_y}(h_l) \quad (4)$$
$$h_i = f_{\theta_i}(h_{i-1}), i \in [2, l] \quad (5)$$
$$h_1 = f_{\theta_1}(x) \quad (6)$$

Deep Neural Networks Backpropagation

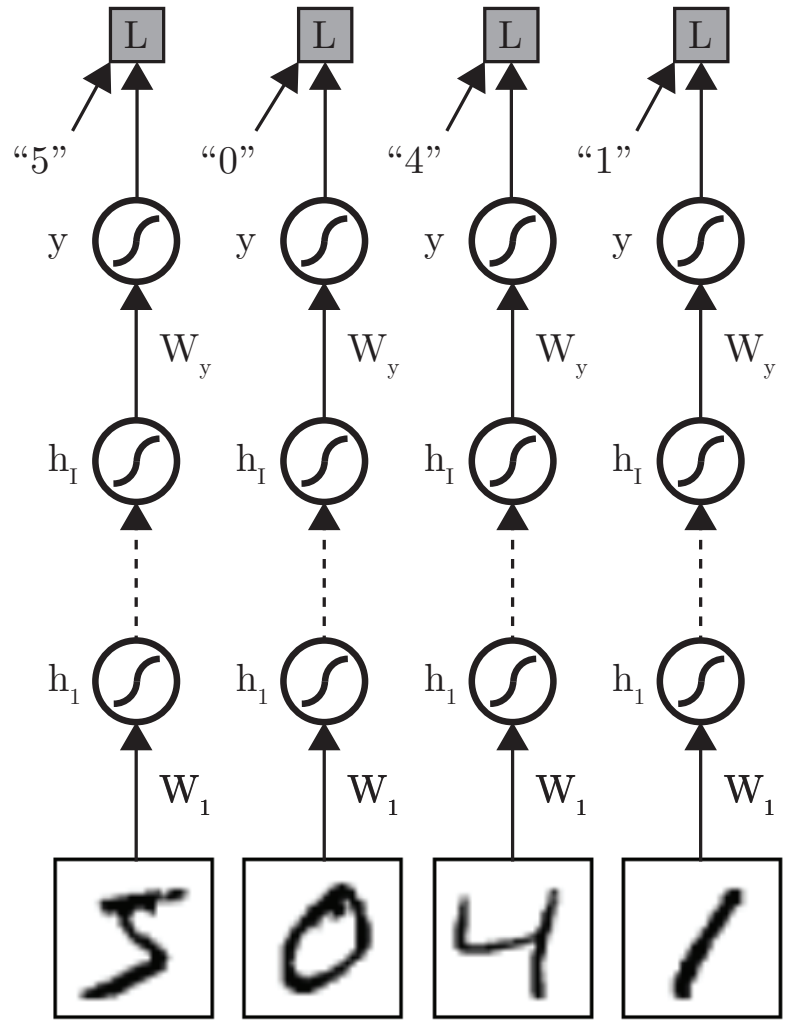


- Updates based on the loss gradient w.r.t. the model's parameters $\frac{\partial L(\mathbf{y},\mathbf{x},\theta)}{\partial \theta_i}$
- Chain rule of backpropagation:

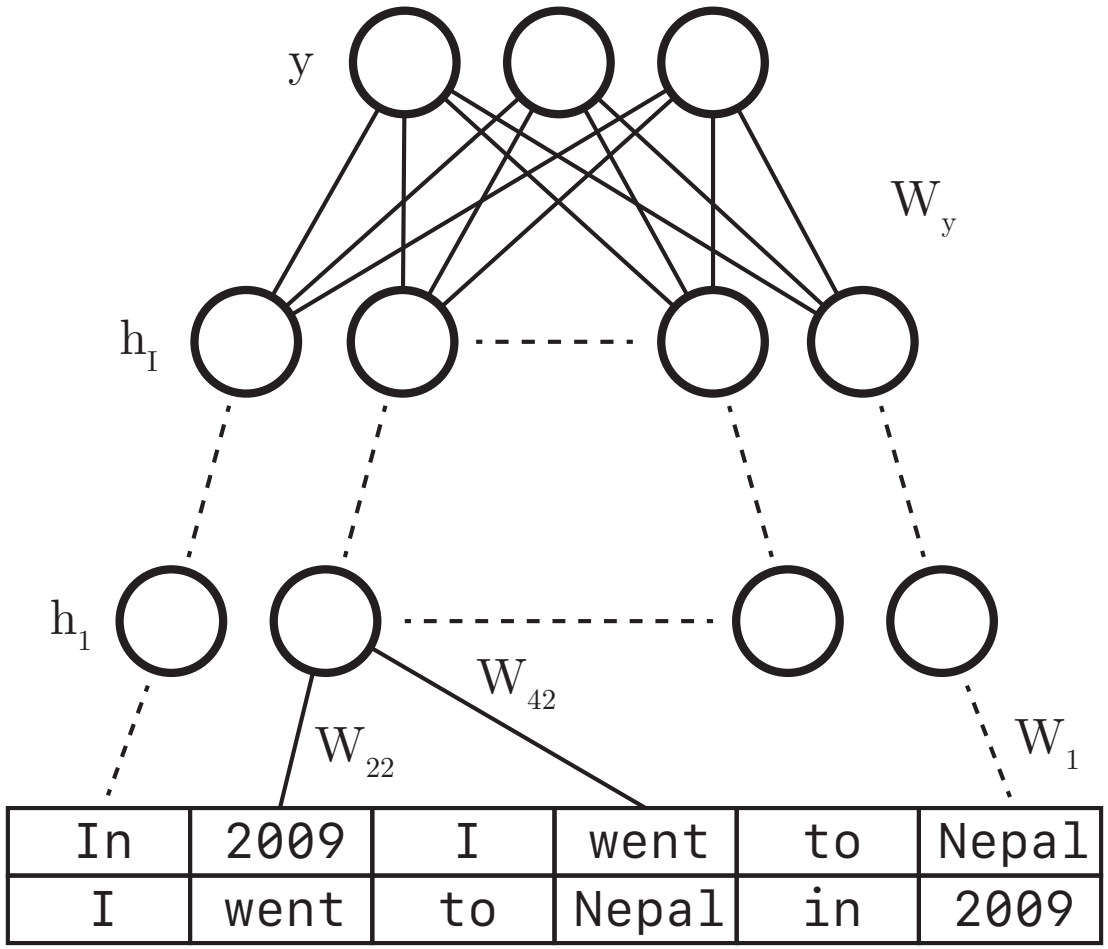
$$\frac{\partial L(\mathbf{y},\mathbf{x},\theta)}{\partial \theta_i} = \frac{\partial L(\mathbf{y},\mathbf{x},\theta)}{\partial h_l}\left(\prod_{k=i}^{l-1}\frac{\partial h_{k+1}}{h_k}\right)\frac{\partial h_i}{\partial \theta_i} \quad (7)$$

Use cases of feed-forward NN architectures

- One-to-one principle
- Each input is processed independantly during both learning and inference

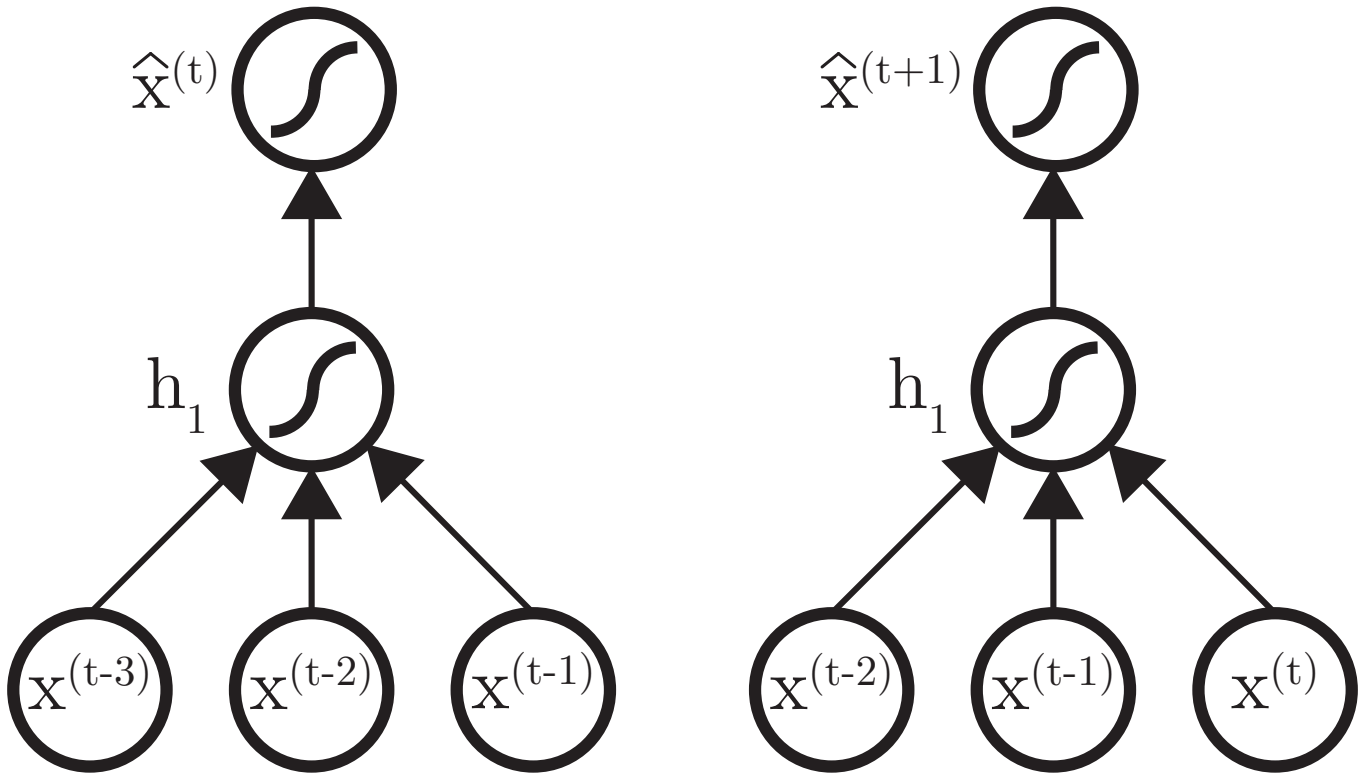


Limitation: Sequential data?



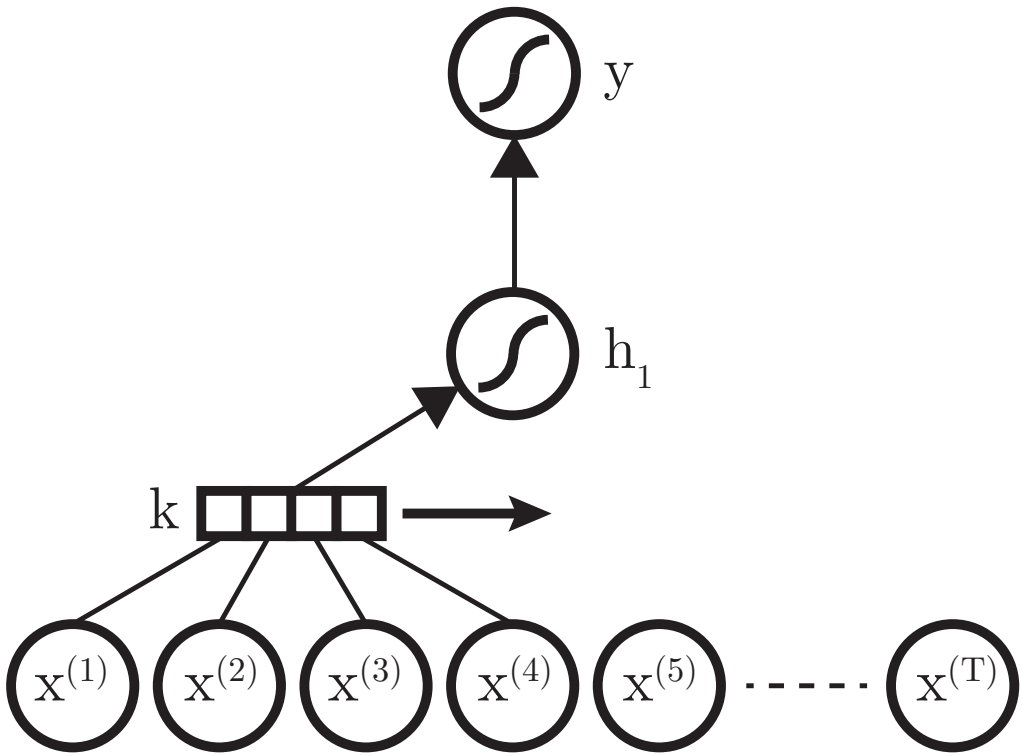
Memoryless solutions - 1

- Autoregressive models



Memoryless solutions - 2

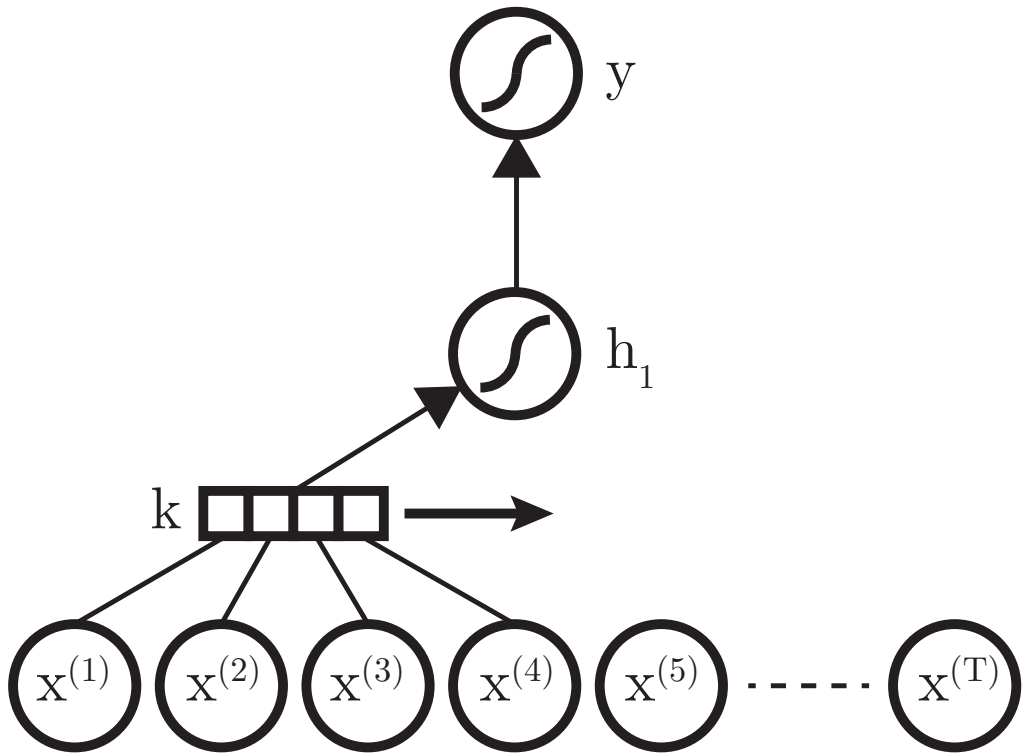
- Convolutional layer



- The parameters of a convolutional layer are **shared between inputs**

Memoryless solutions - 2

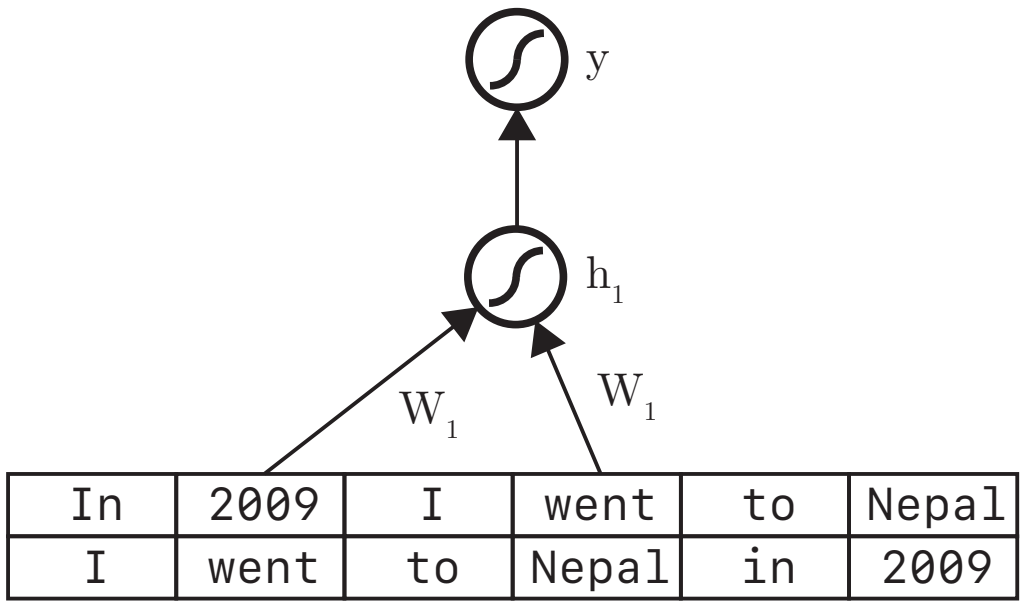
- Convolutional layer



- The parameters of a convolutional layer are **shared between inputs**

Parameter sharing

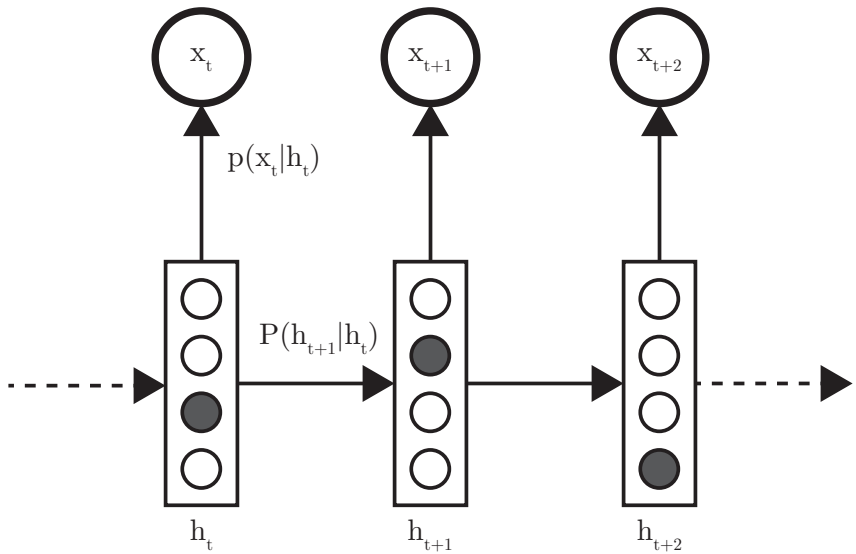
- Using the same parameters for each timestep of the input sequence: added invariance



Limitations

- Short-term, fixed-length memory
- Learning is performed on sequences with fixed dimensions
- Sequence processing often requires to infer at timestep t using information from the inputs at timesteps 0 to $t - 1$
- Solution: Hidden states with recurrence

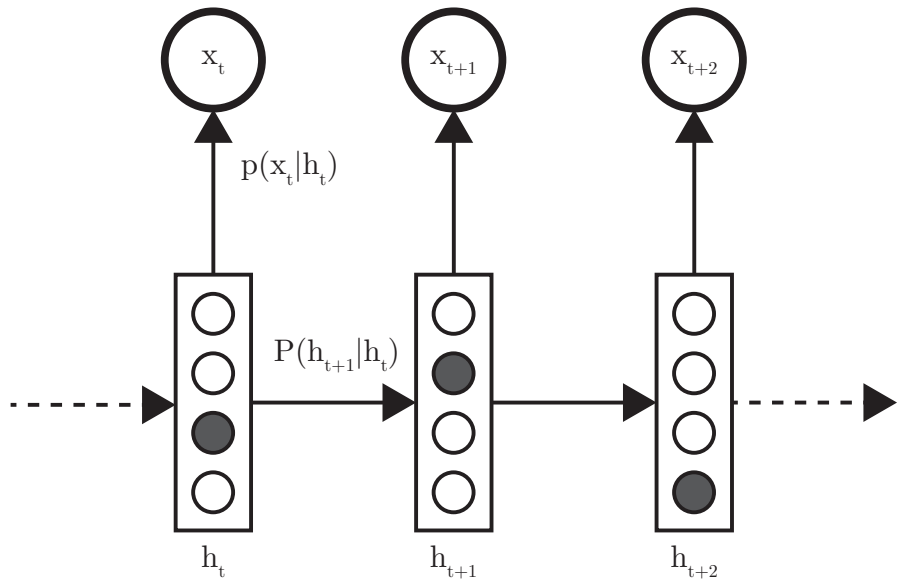
Hidden Markov Models (HMM)



- h_t: hidden state at time t
- $p(x_t|h_t)$: probability density of the output x_t given the state h_t
- $P(h_t|h_{t-1})$: probability of a transition from state h_{t-1} to state h_t

Hidden Markov Models (HMM)

- Stochastic, the hidden state cannot be directly observed
- Inference: find the best sequence of hidden states for a given observation

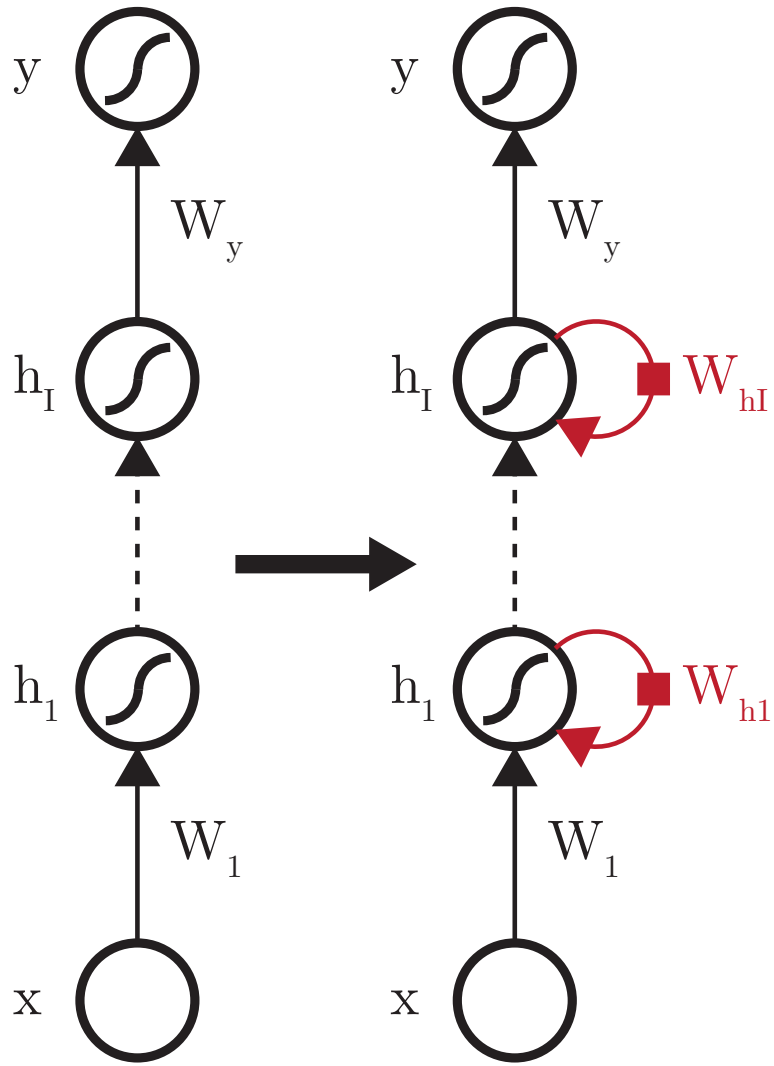


- Joint distribution:

$$p(x_1, ..., x_T, h_1, ..., h_T) = P(h_1) \prod_{t=2}^{T} P(h_t|h_{t-1}) \prod_{t=1}^{T} p(x_t|h_t) \quad (8)$$

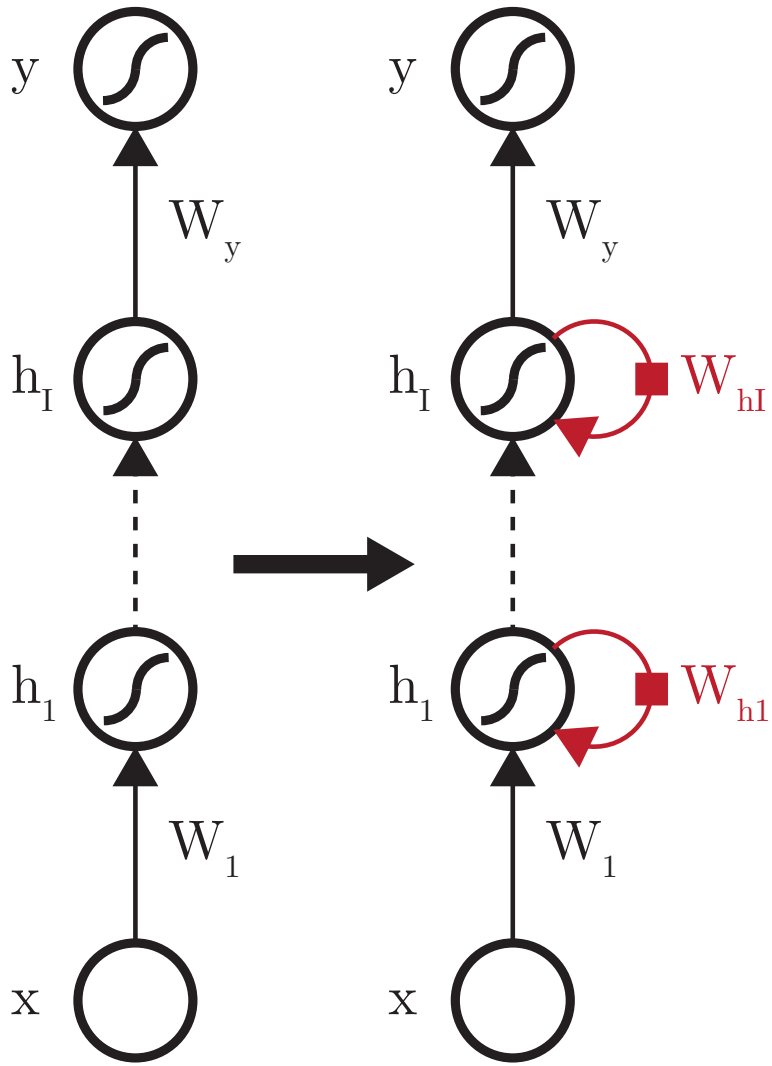
Recurrent Neural Networks (RNN)

- W_{h_i}: Recurrent weight matrix



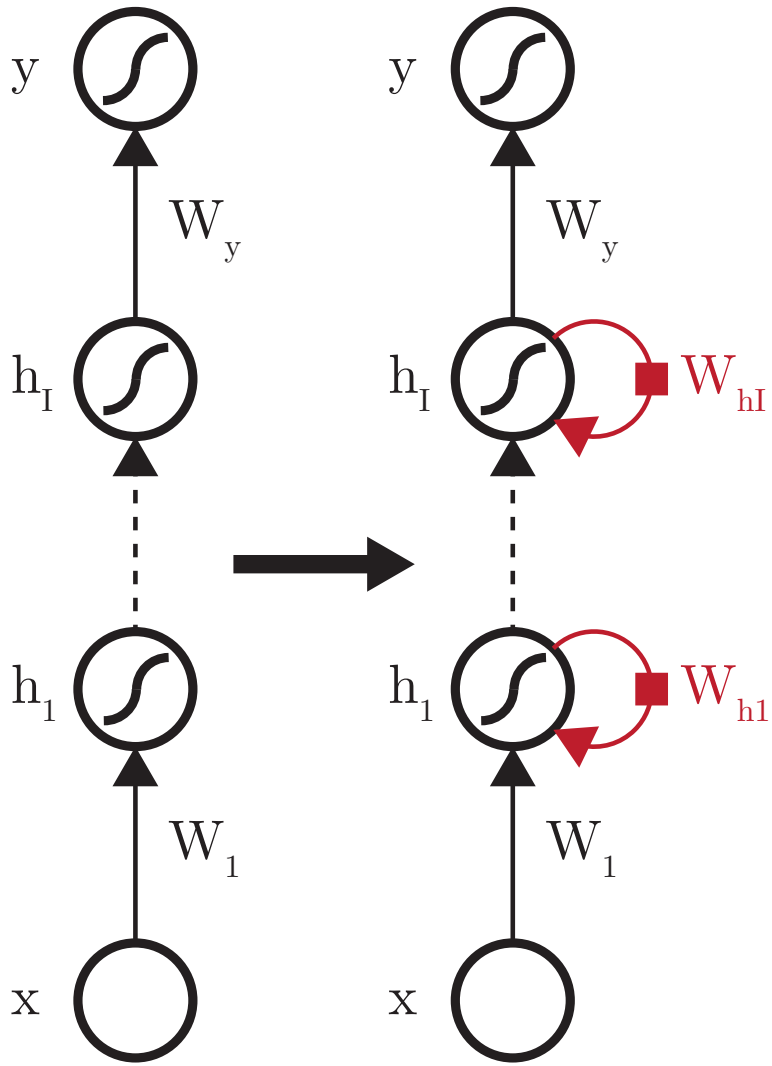
Recurrent Neural Networks (RNN)

- Transition from feed-forward neurons by adding a recurrent connection on a NN graph
- Deterministic, with continuous hidden states as opposed to the HMM

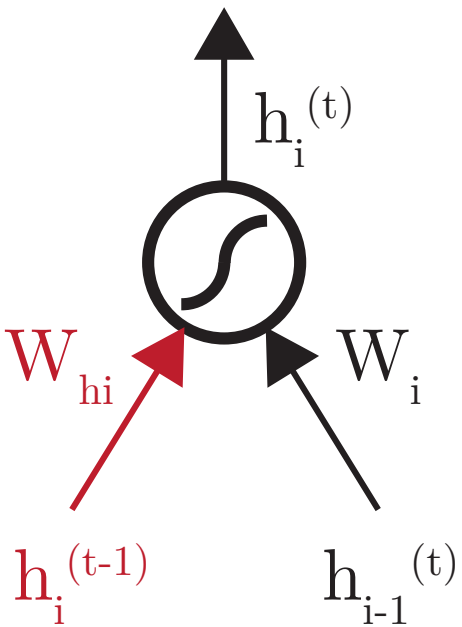


Recurrent Neural Networks (RNN)

- Transition from feed-forward neurons by adding a recurrent connection on a NN graph
- Deterministic, with continuous hidden states as opposed to the HMM



The RNN cell



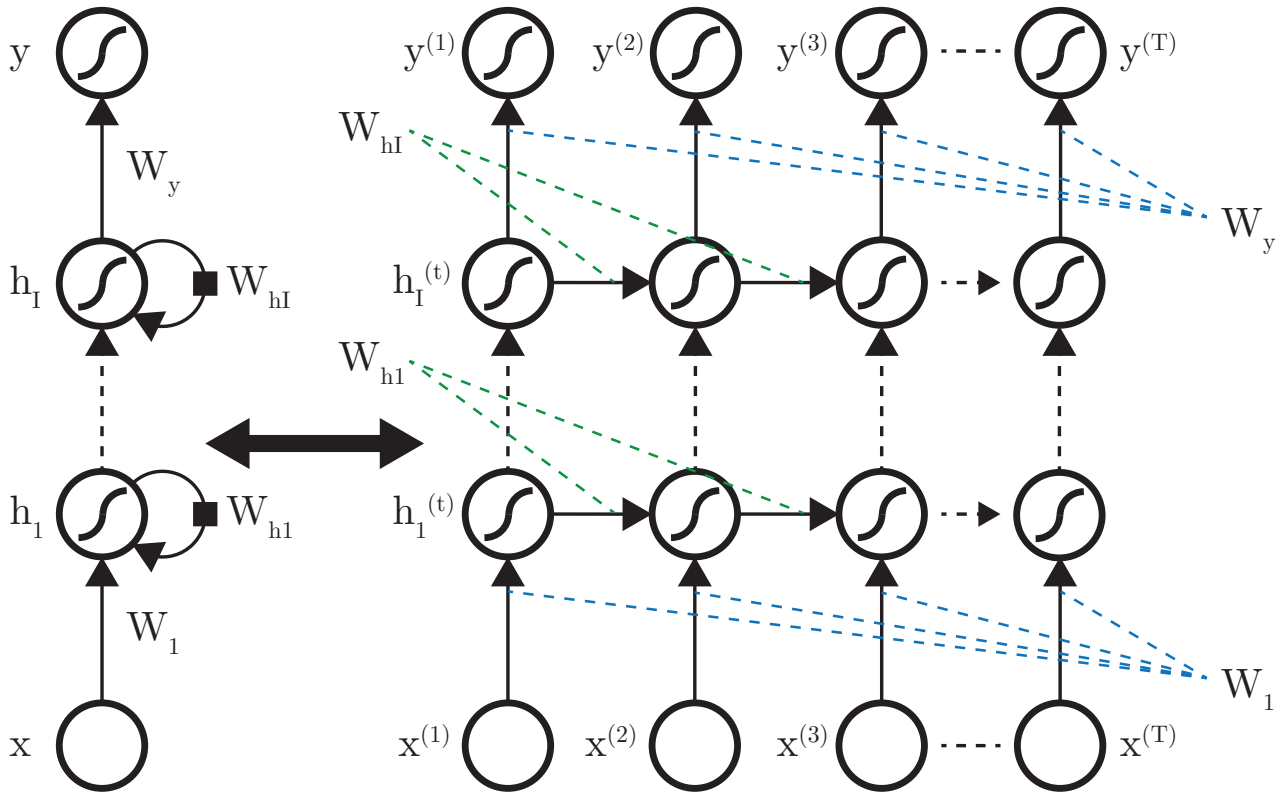
- Extension of the feed-forward NN

$$h_i^{(t)} = f_i\left(W_i h_{i-1}^{(t)} + W_{h_i} h_i^{(t-1)} + b_i\right) \quad (9)$$

- Generalization

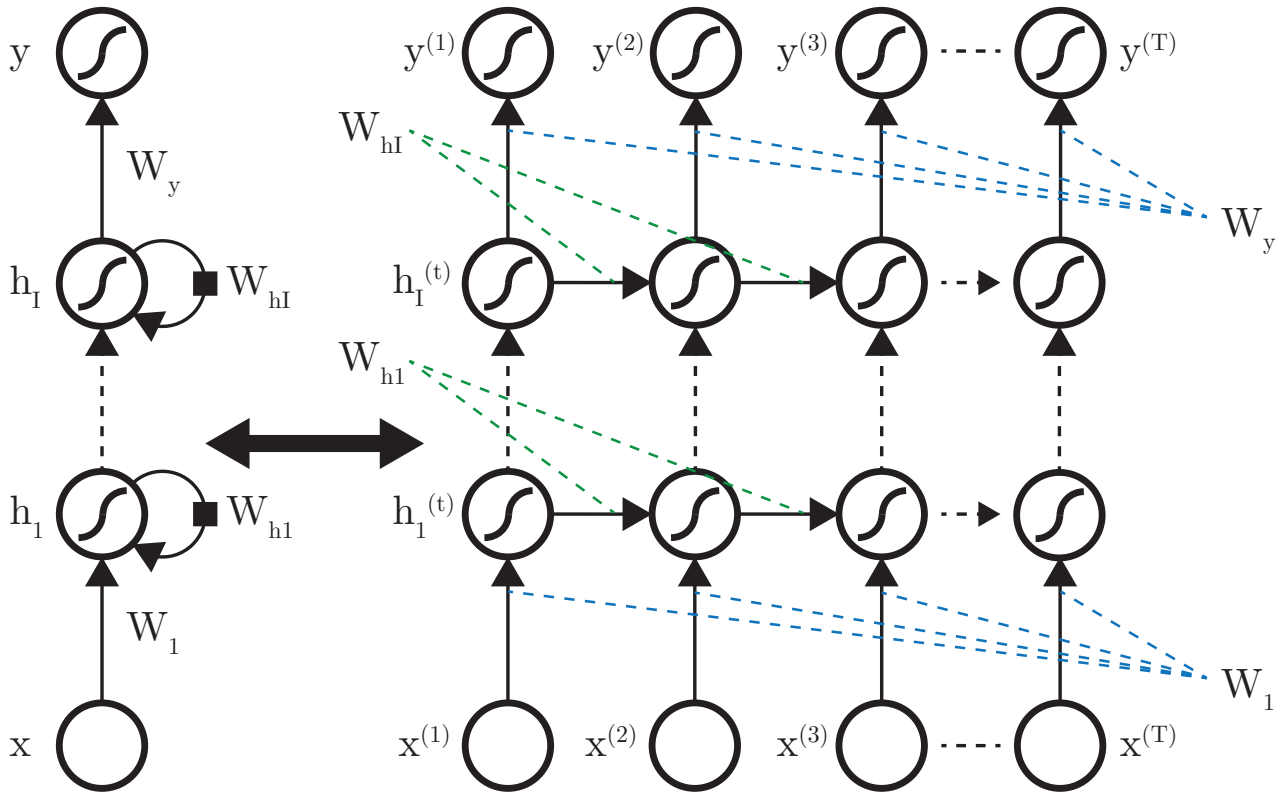
$$h_i^{(t)} = f_{\theta_i}\left(h_{i-1}^{(t)}, h_i^{(t-1)}\right) \quad (10)$$

Unrolling recurrent cells



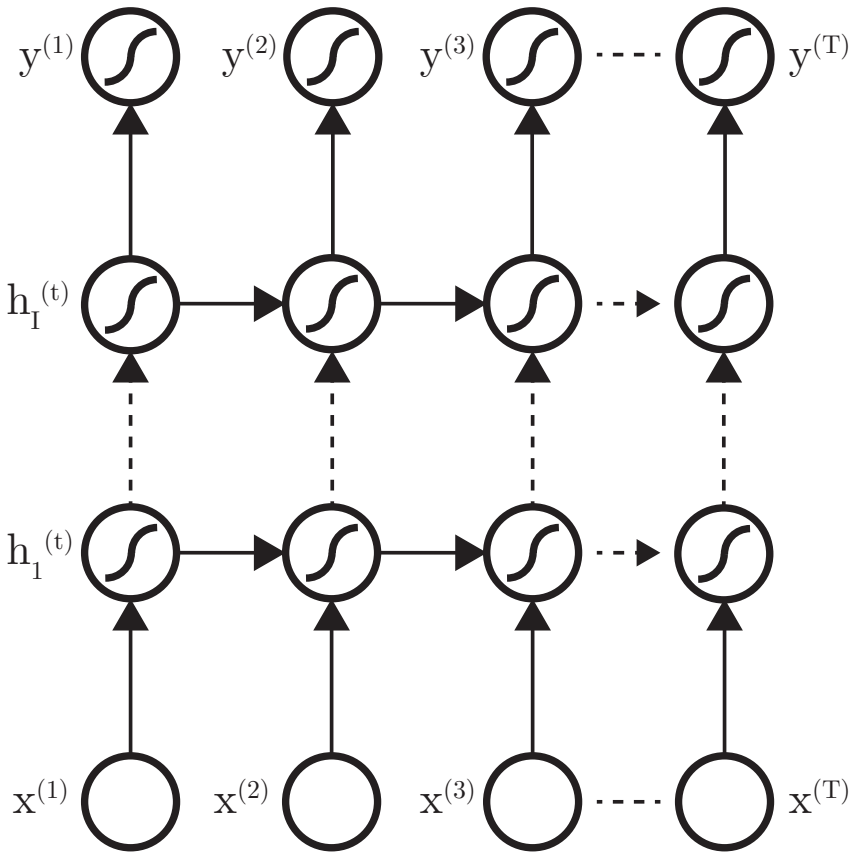
- T: Network memory
- Every neural network with recurrence can be viewed as a feed-forward model with shared parameters

Unrolling recurrent cells



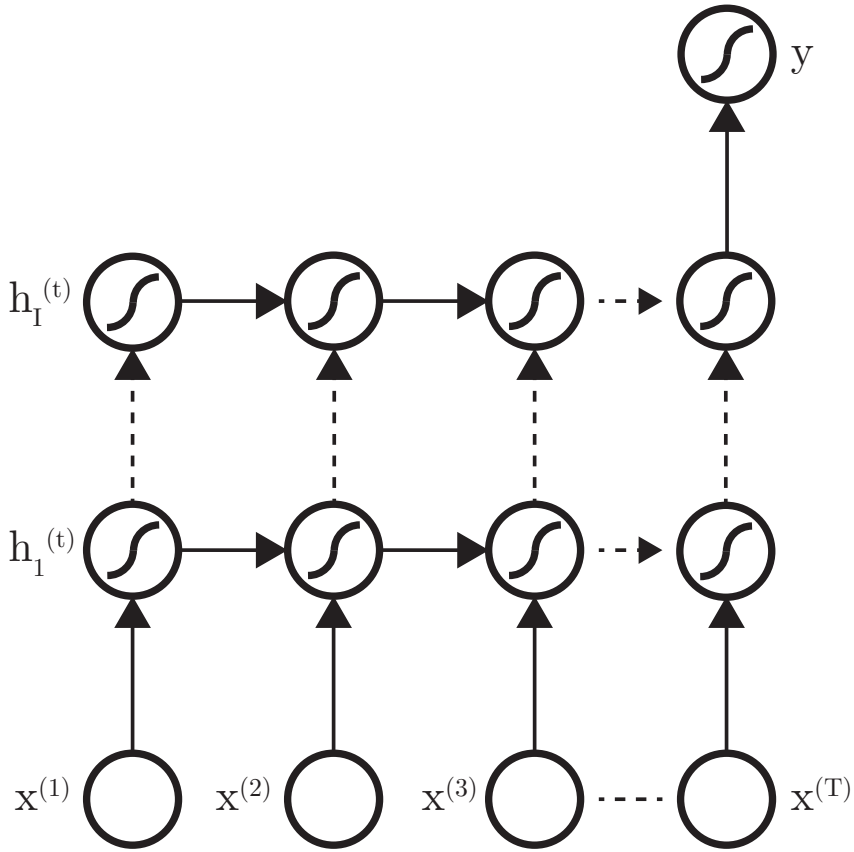
- T: Network memory
- Every neural network with recurrence can be viewed as a feed-forward model with shared parameters

Many-to-many RNN



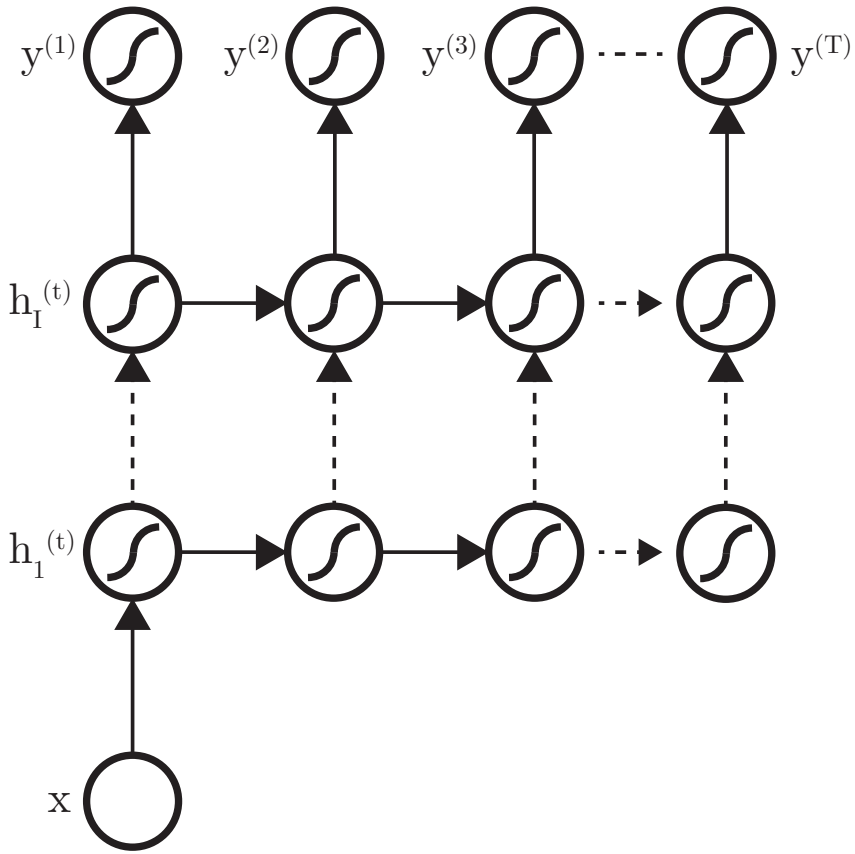
- Applications: Text translation, text-to-speech (TTS), music transcription, video compression, event detection...

Many-to-one RNN



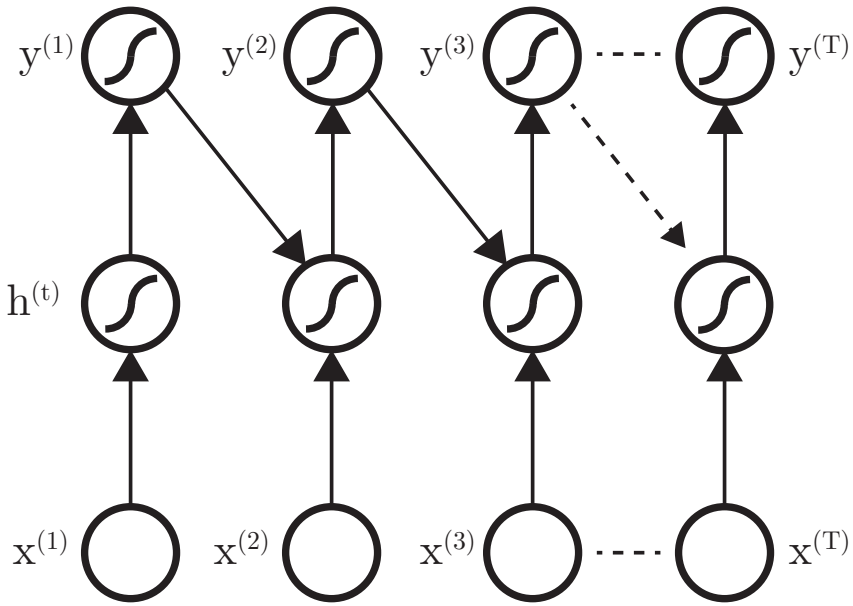
- Only the output at the last timestep is regarded
- Applications: Sequence classification...

One-to-many RNN



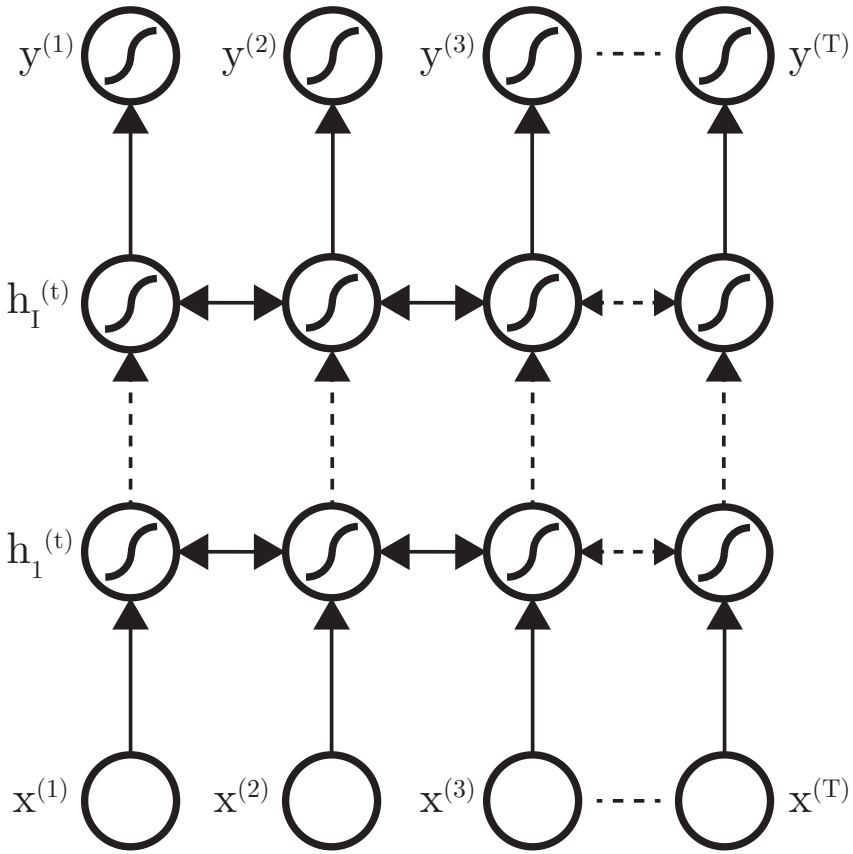
- A sequence of outputs is generated from a single input
- Applications: Image captioning...

RNN with output recurrence



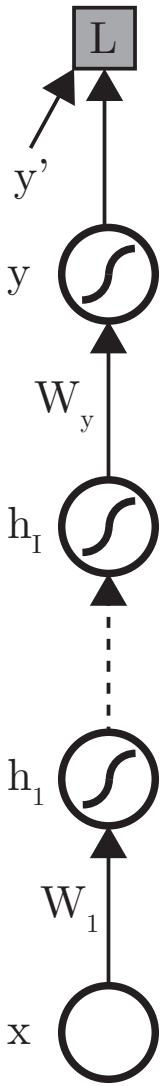
- Easier to train: more control over the recurring outputs than the hidden layers
- Less information can be retained, the NN is less stable
- Applications: Prediction...

Bidirectional RNN



- Use both "past" and "future" information
- Parameter sharing (separated or common)

Backpropagation



- Forward equations:

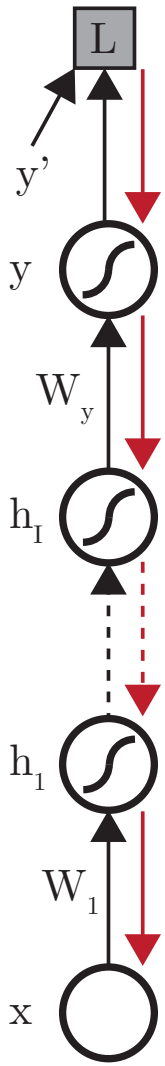
$$y = f_{\theta_y}(h_I) \tag{11}$$

$$h_i = f_{\theta_i}(h_{i-1}), i \in [2, I] \tag{12}$$

$$h_1 = f_{\theta_1}(x) \tag{13}$$

- Use of a loss function with predicted and target outputs: Mean squared error (MSE), cross-entropy, L^n norm, divergences...

Backpropagation



- Parameter optimization to minimize the loss function
- Updates based on the loss gradient w.r.t. the model's parameters $\frac{\partial L(\mathbf{y},\mathbf{x},\theta)}{\partial \theta_i}$
- Chain rule of backpropagation:

$$\frac{\partial L(\mathbf{y},\mathbf{x},\theta)}{\partial \theta_i} = \frac{\partial L(\mathbf{y},\mathbf{x},\theta)}{\partial h_l}\left(\prod_{k=i}^{l-1}\frac{\partial h_{k+1}}{h_k}\right)\frac{\partial h_i}{\partial \theta_i} \tag{14}$$

Backpropagation: Example in a deep NN

- Loss equation

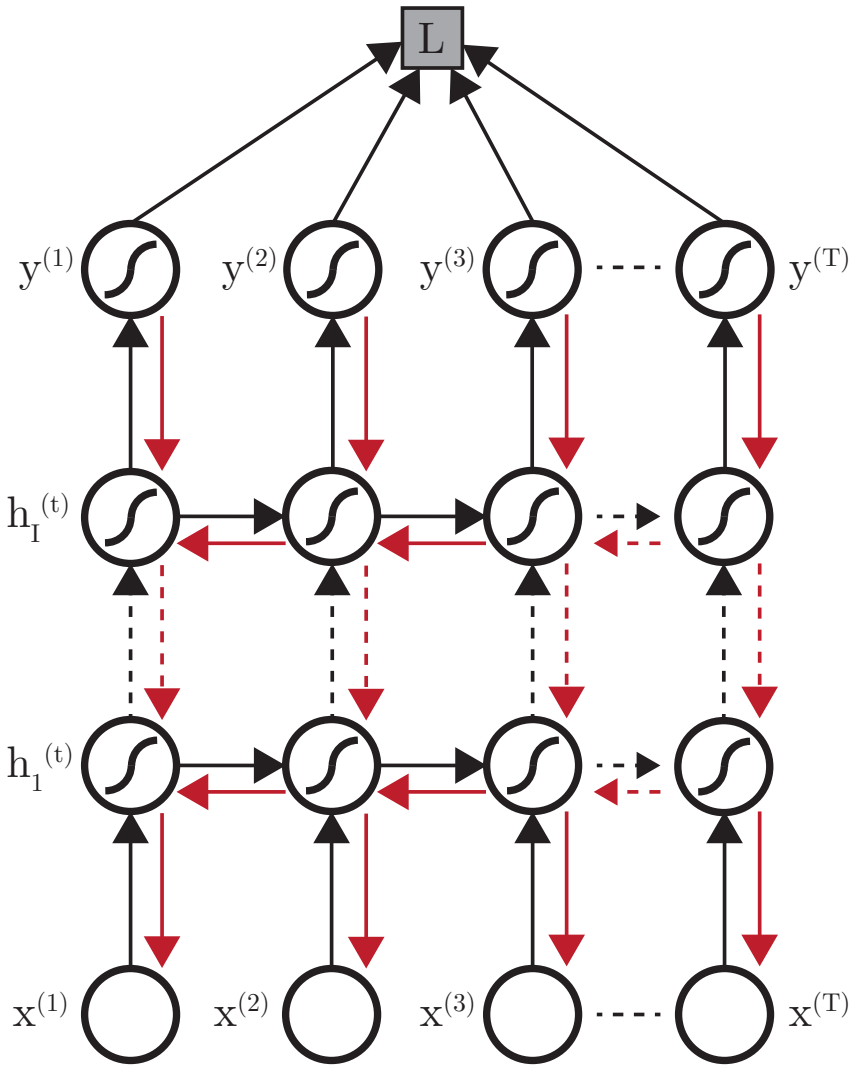
$$L(x,y) = f_L\left(W_y f_l\left(W_l f_{l-1}\left(\ldots f_1\left(W_1 x\right)\right)\right), y\right) \tag{15}$$

- Chain rule application

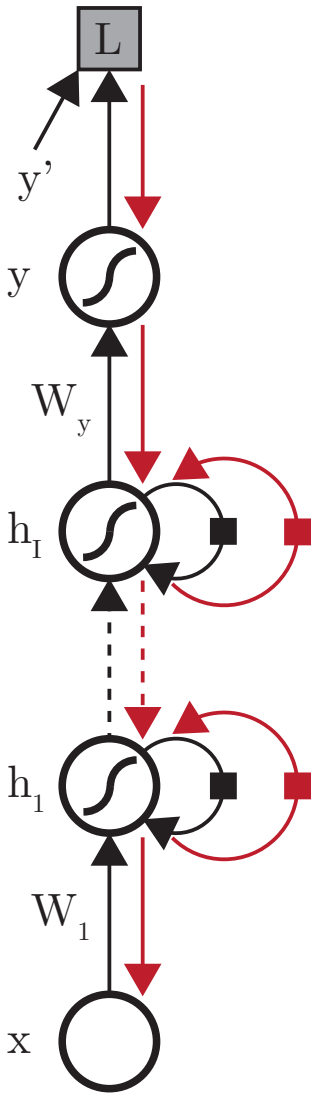
$$\nabla_{h_i} L(x,y) = \frac{\partial L}{\partial h_i} = W_{i+1}^T \left(\frac{\partial L}{\partial h_{i+1}} \odot f'_{i+1}\left(W_{i+1} h_i\right)\right) \tag{16}$$

$$\nabla_{W_i} L(x,y) = \frac{\partial L}{\partial W_i} = \left(\frac{\partial L}{\partial h_i} \odot f'_i\left(W_i h_{i-1}\right)\right) h_{i-1}^T \tag{17}$$

Backpropagation through time (BPTT)

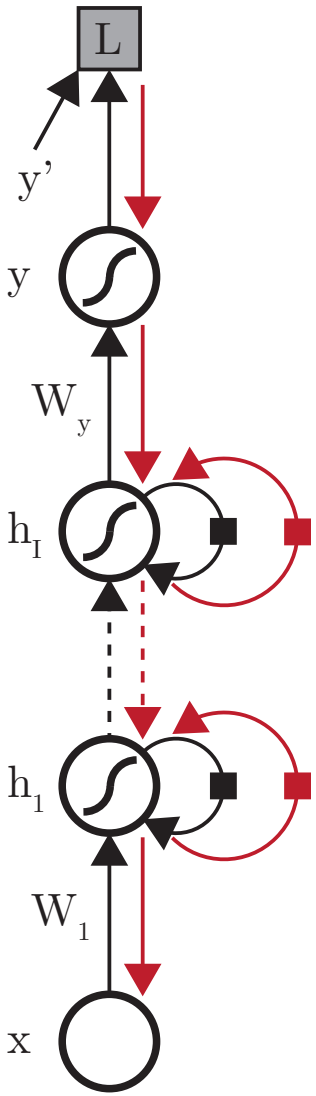


Backpropagation through time (BPTT)



- Same use of the chain rule of backpropagation as in a feed-forward network
- Parameter sharing: gradients are summed (or averaged) over all timesteps

Backpropagation through time (BPTT)



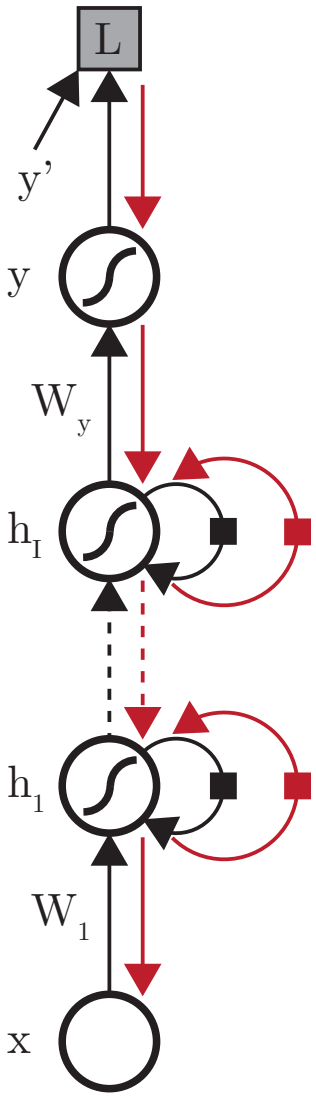
- Loss term

$$L(x,y) = \sum_t L^{(t)}\left(x, y^{(t)}\right) \tag{18}$$

- Gradient calculation

$$\nabla_{\theta_i} L(x,y) = \frac{\partial L}{\partial \theta_i} = \sum_t \frac{\partial L^{(t)}}{\partial \theta_i} \tag{19}$$

Backpropagation through time (BPTT)



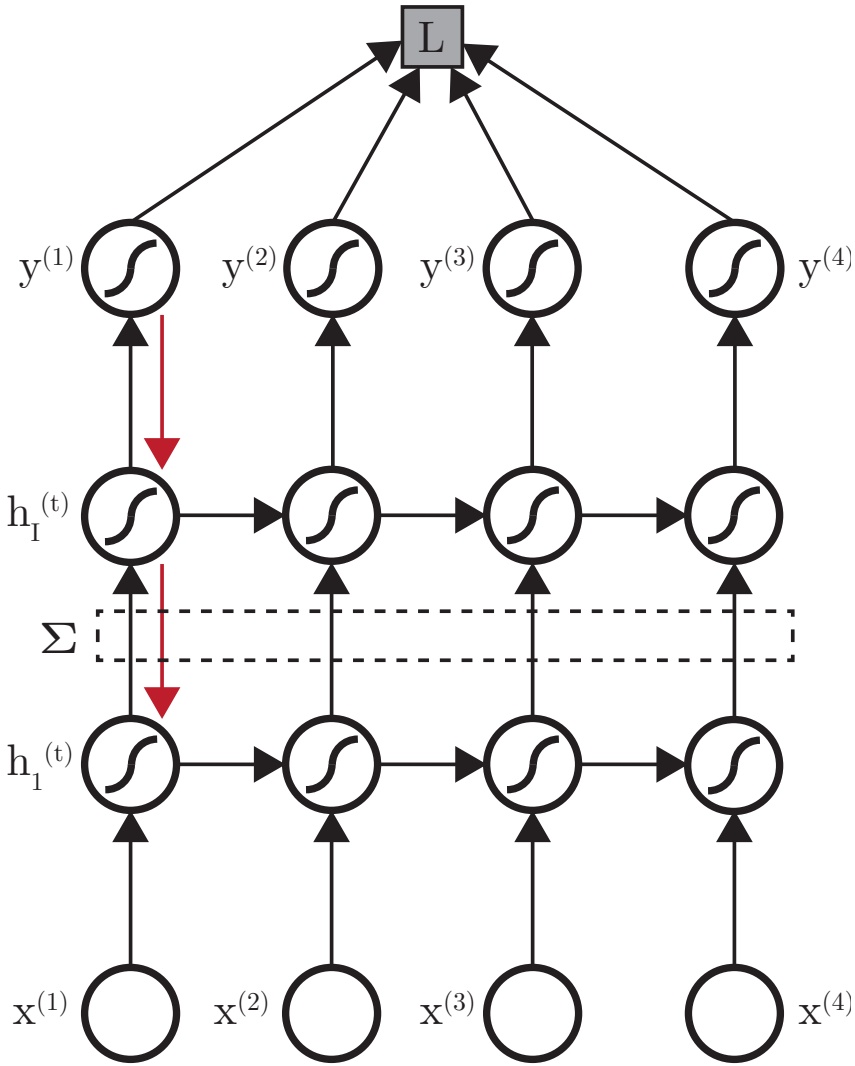
- Loss term

$$L(x,y) = \sum_t L^{(t)}\left(x, y^{(t)}\right) \tag{18}$$

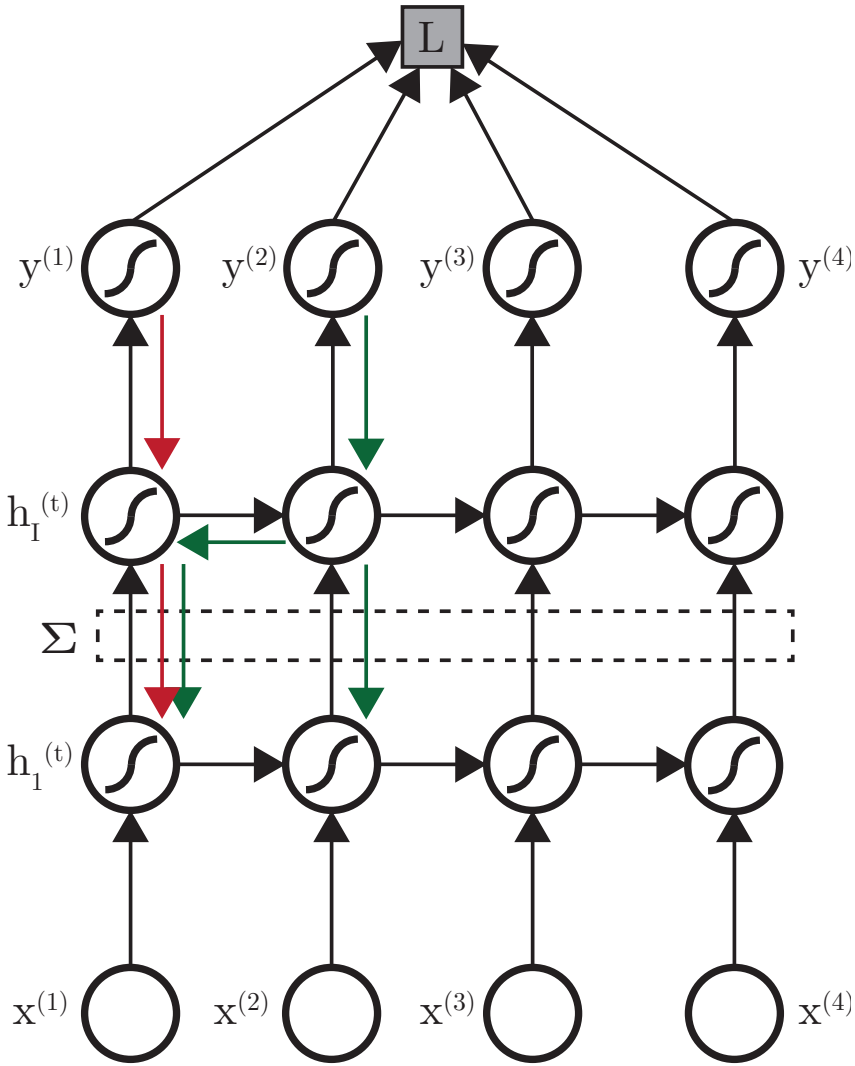
- Gradient calculation

$$\nabla_{\theta_i} L(x,y) = \frac{\partial L}{\partial \theta_i} = \sum_t \frac{\partial L^{(t)}}{\partial \theta_i} \tag{19}$$

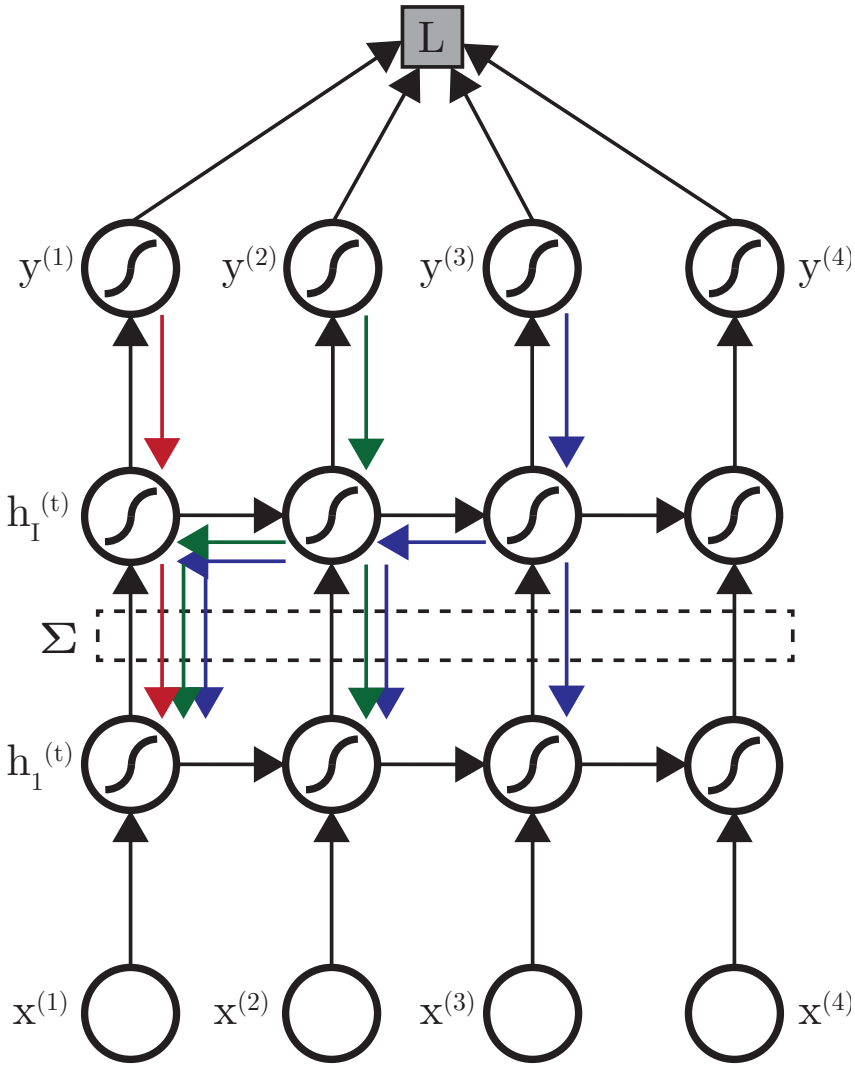
Backpropagation through time (BPTT): example



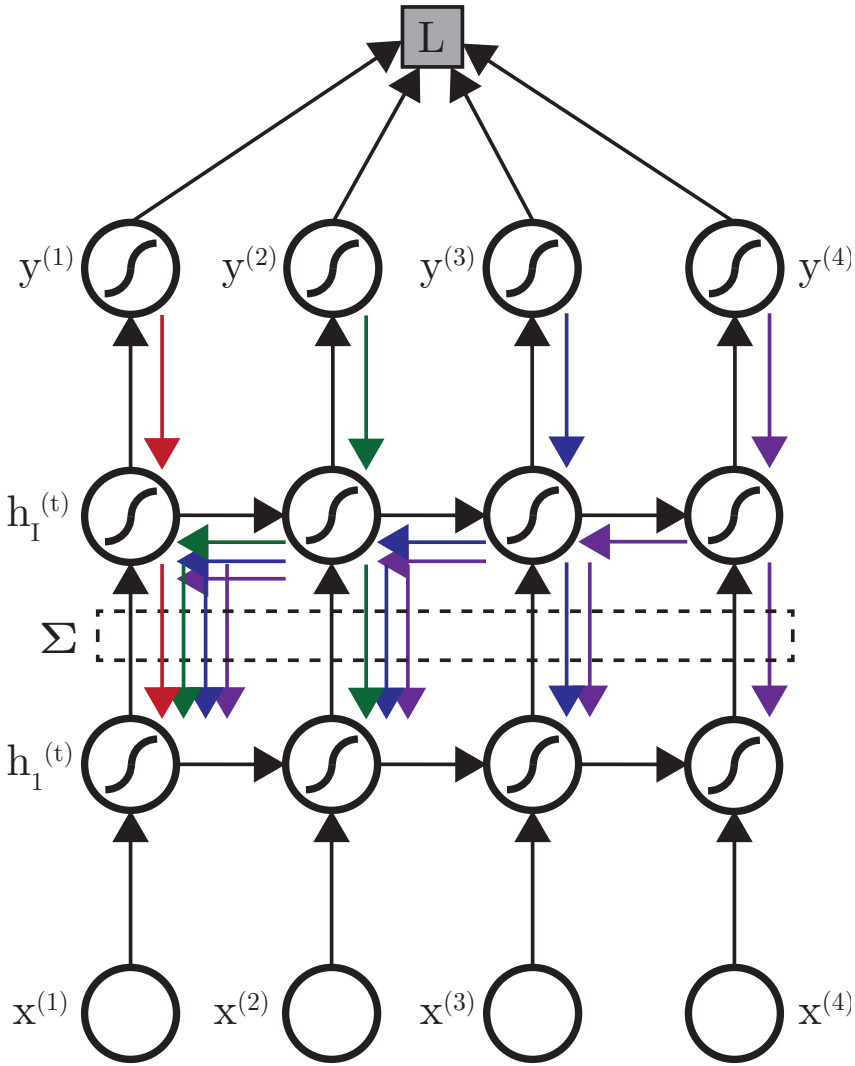
Backpropagation through time (BPTT): example



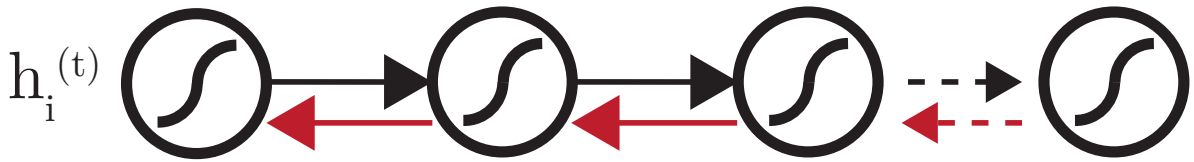
Backpropagation through time (BPTT): example



Backpropagation through time (BPTT): example



Limitations of the RNN



- If the RNN cell forward propagation is:

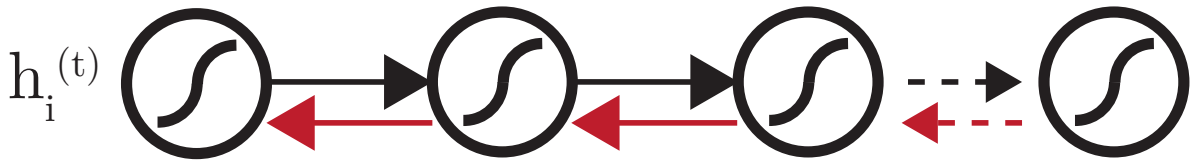
$$h_i^{(t)} = f_i\left(W_i h_{i-1}^{(t)} + W_{h_i} h_i^{(t-1)} + b_i\right) \tag{20}$$

- Then :

$$\frac{\partial h_i^{(t)}}{\partial h_i^{(t-1)}} = W_{h_i}^T f_i'\left(W_i h_{i-1}^{(t)} + W_{h_i} h_i^{(t-1)} + b_i\right) \tag{21}$$

  is used multiplicatively to backpropagate one timestep.
- Backpropagation through time multiplies gradients by the recurrent weight matrix W_{h_i} at each timestep

Limitations of the RNN



- If the RNN cell forward propagation is:

$$h_i^{(t)} = f_i\left(W_i h_{i-1}^{(t)} + W_{h_i} h_i^{(t-1)} + b_i\right) \tag{20}$$

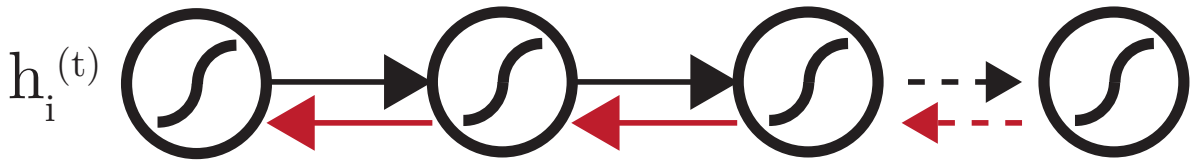
- Then :

$$\frac{\partial h_i^{(t)}}{\partial h_i^{(t-1)}} = W_{h_i}^T f_i'\left(W_i h_{i-1}^{(t)} + W_{h_i} h_i^{(t-1)} + b_i\right) \tag{21}$$

  is used multiplicatively to backpropagate one timestep.

- Backpropagation through time multiplies gradients by the recurrent weight matrix W_{h_i} at each timestep

Limitations of the RNN



- If the RNN cell forward propagation is:

$$h_i^{(t)} = f_i\left(W_i h_{i-1}^{(t)} + W_{h_i} h_i^{(t-1)} + b_i\right) \tag{20}$$

- Then :

$$\frac{\partial h_i^{(t)}}{\partial h_i^{(t-1)}} = W_{h_i}^T f_i'\left(W_i h_{i-1}^{(t)} + W_{h_i} h_i^{(t-1)} + b_i\right) \tag{21}$$

  is used multiplicatively to backpropagate one timestep.
- Backpropagation through time multiplies gradients by the recurrent weight matrix W_{h_i} at each timestep

Limitations of the RNN

- Largest singular value λ of the recurrent weight matrix
  - $\max_i |\lambda_i| > 1$: Exploding gradient (Necessary condition)
  - $\max_i |\lambda_i| < 1$: Vanishing gradient (Sufficient condition)
- The vanishing or exploding of the gradient is exponential in $O(n^T)$ where T is the total duration of the network
- In RNNs, this means unstabilities or difficulties learning long-term dependancies in the data

---

Hochreiter et al. Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies 2001

Limitations of the RNN

- Largest singular value λ of the recurrent weight matrix
  - $\max_i |\lambda_i| > 1$: Exploding gradient (Necessary condition)
  - $\max_i |\lambda_i| < 1$: Vanishing gradient (Sufficient condition)
- The vanishing or exploding of the gradient is exponential in $O(n^T)$ where T is the total duration of the network
- In RNNs, this means unstabilities or difficulties learning long-term dependancies in the data

---

Hochreiter et al. Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies 2001

Limitations of the RNN

- Largest singular value λ of the recurrent weight matrix
  - $\max_i |\lambda_i| > 1$: Exploding gradient (Necessary condition)
  - $\max_i |\lambda_i| < 1$: Vanishing gradient (Sufficient condition)
- The vanishing or exploding of the gradient is exponential in $O(n^T)$ where T is the total duration of the network
- In RNNs, this means unstabilities or difficulties learning long-term dependancies in the data

---

Hochreiter et al. Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies 2001

Solution 1: Echo state networks

- Idea: fix the difficult-to-learn parameters (recurrent, input)
- Only the output weights are learned with smart initialization of other parameters
- Generally a sparse initialization with a spectral radius ($|\lambda|$ maximum) close to 1 is efficient
- The training is equivalent to that of a perceptron
- Much less representational power

Jaeger. The "echo state" approach to analysing and training recurrent neural networks. GMD Report, 2010

Solution 1: Echo state networks

- Idea: fix the difficult-to-learn parameters (recurrent, input)
- Only the output weights are learned with smart initialization of other parameters
- Generally a sparse initialization with a spectral radius ($|\lambda|$ maximum) close to 1 is efficient
- The training is equivalent to that of a perceptron
- Much less representational power

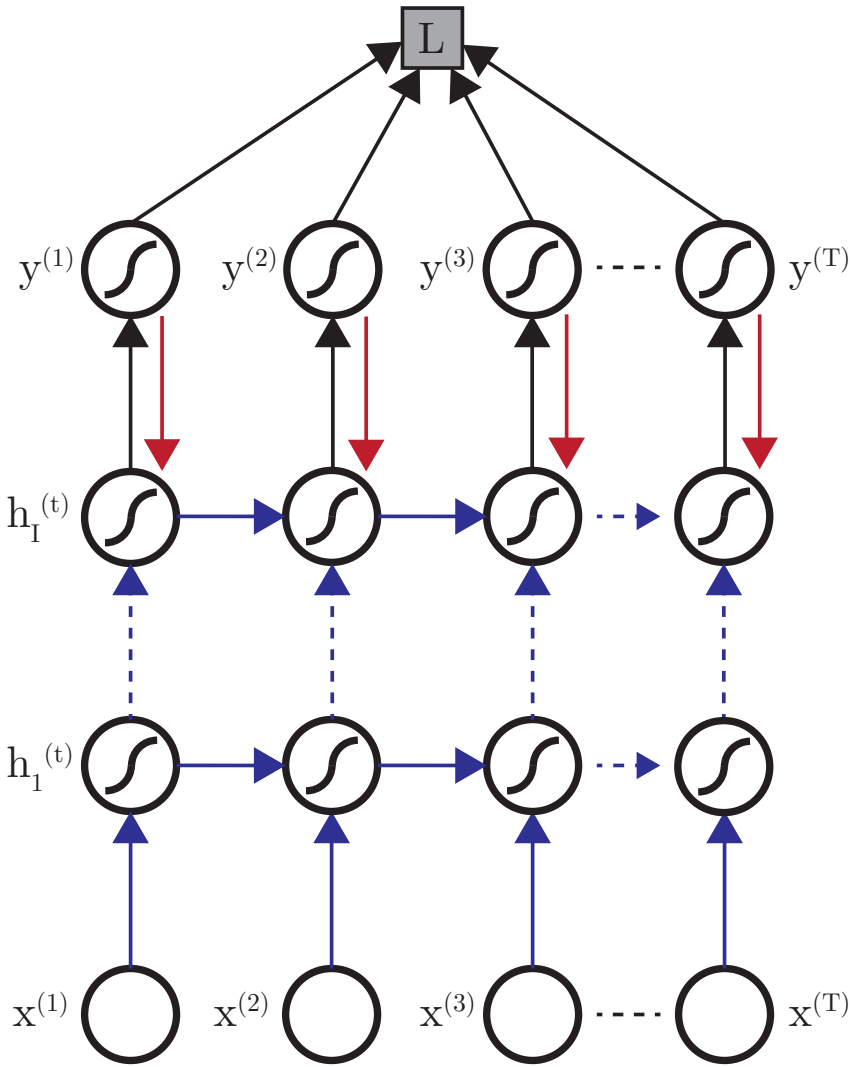
Jaeger. The "echo state" approach to analysing and training recurrent neural networks. GMD Report, 2010

Solution 1: Echo state networks

- Idea: fix the difficult-to-learn parameters (recurrent, input)
- Only the output weights are learned with smart initialization of other parameters
- Generally a sparse initialization with a spectral radius ($|\lambda|$ maximum) close to 1 is efficient
- The training is equivalent to that of a perceptron
- Much less representational power

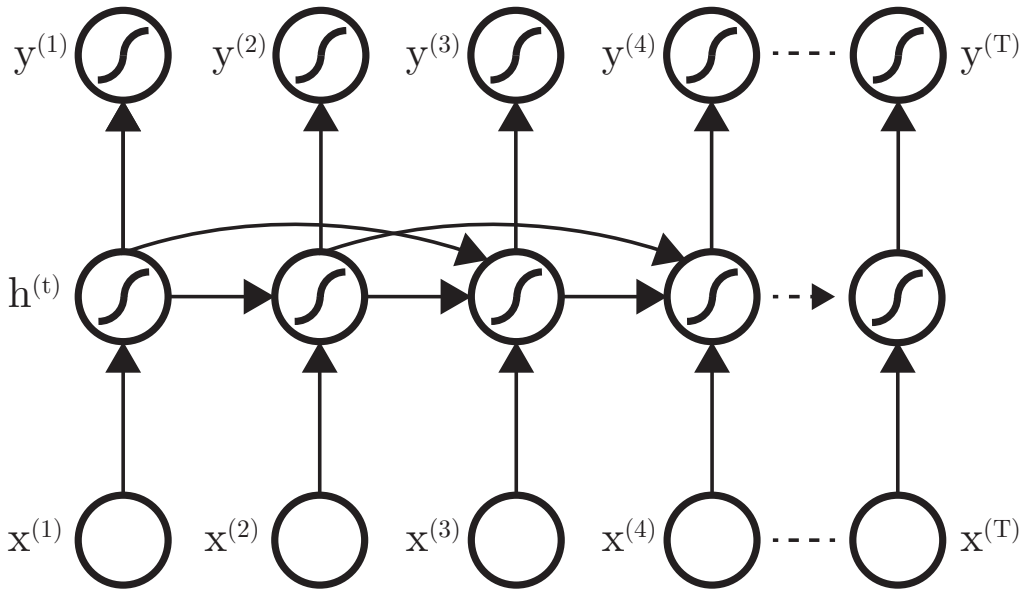
Jaeger. The "echo state" approach to analysing and training recurrent neural networks. GMD Report, 2010

Solution 1: Echo state networks



Solution 2: Leaky units

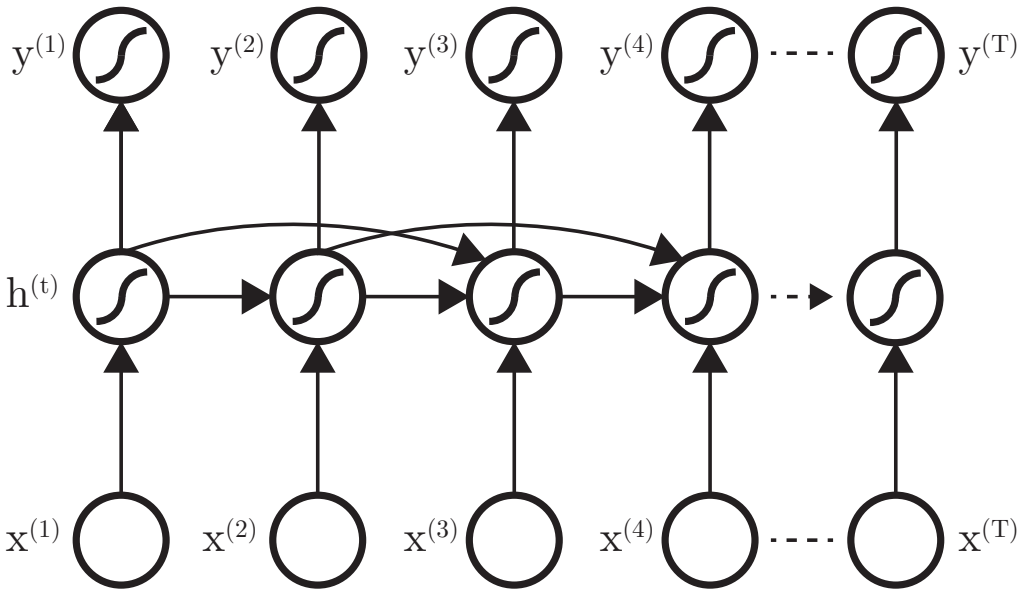
- Introduce a delay, skip connections on the unrolled graph at different time scales



- Gradients vanishing and exploding are in $O(n^{T/\tau})$ (τ being the skip length)
- More parameters to learn

Solution 2: Leaky units

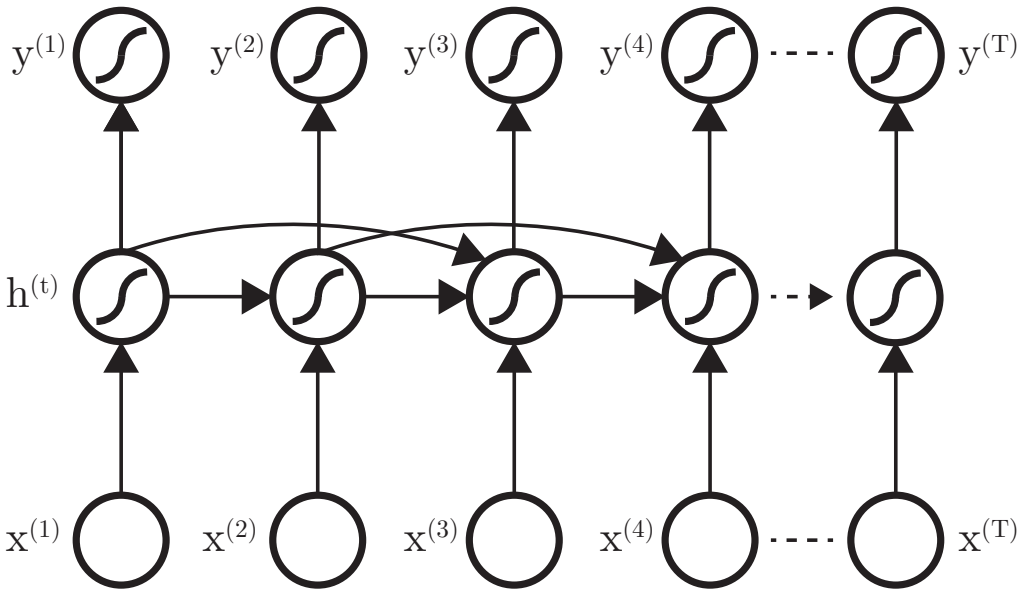
- Introduce a delay, skip connections on the unrolled graph at different time scales



- Gradients vanishing and exploding are in $O(n^{T/\tau})$ (τ being the skip length)
- More parameters to learn

Solution 2: Leaky units

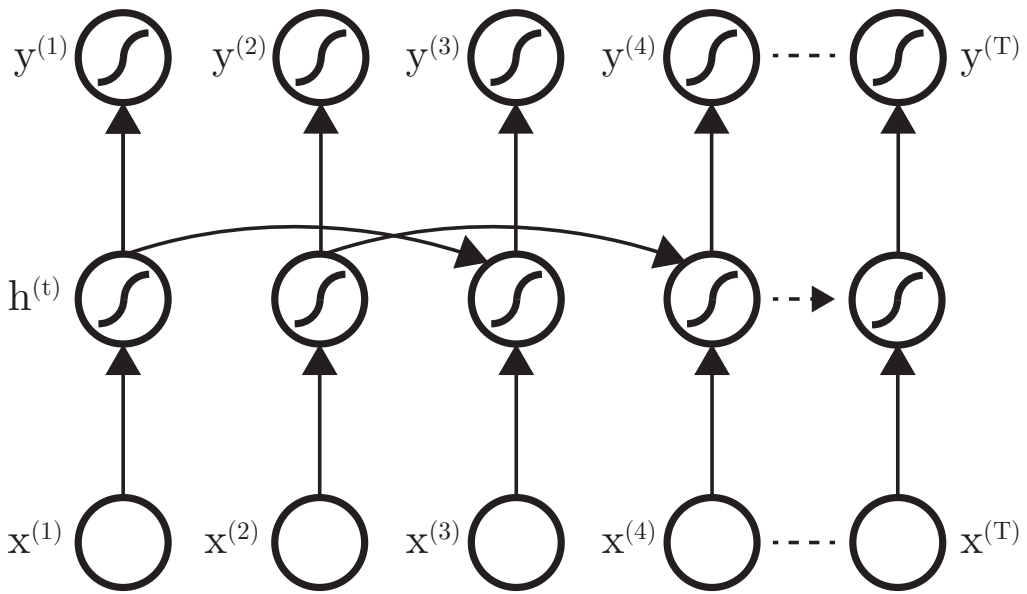
- Introduce a delay, skip connections on the unrolled graph at different time scales



- Gradients vanishing and exploding are in $O(n^{T/\tau})$ (τ being the skip length)
- More parameters to learn

Solution 2: Leaky units

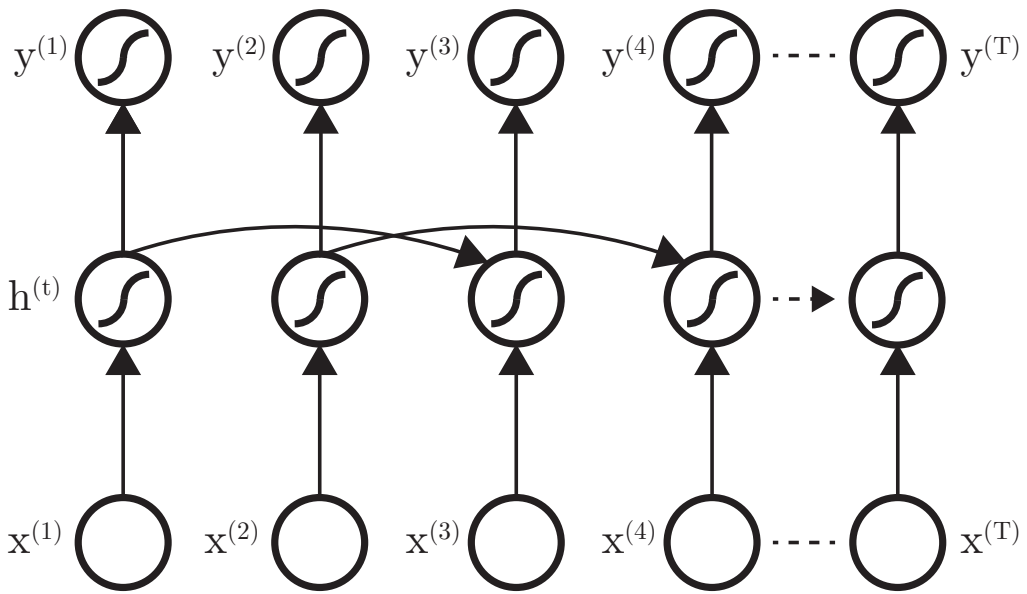
- Variant with direct recurrent connections removed (RNN multi time scales)



- Less parameters to learn: reduced complexity
- Less representational power of short-term data dependancies

Solution 2: Leaky units

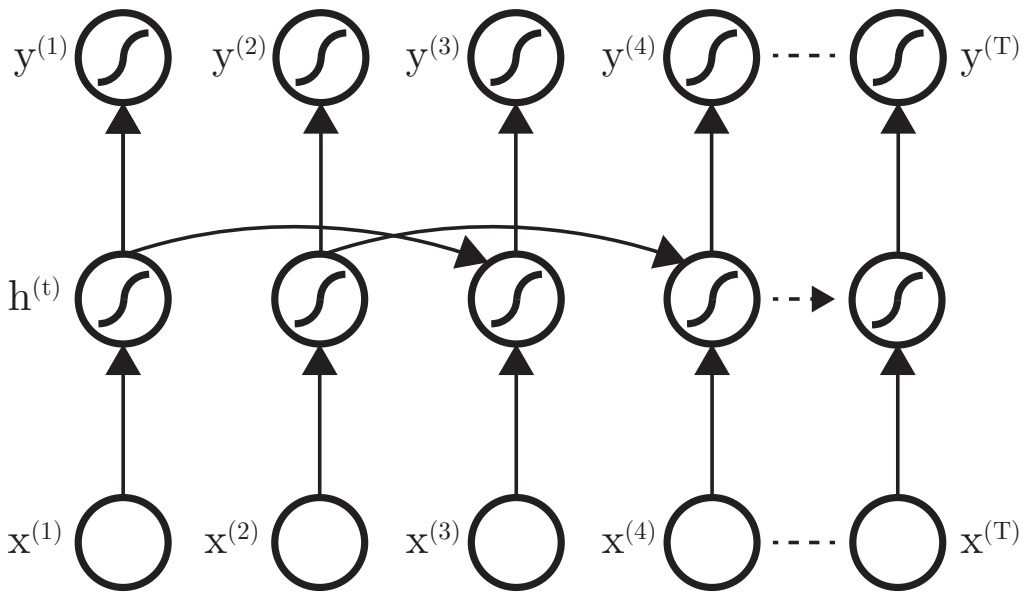
- Variant with direct recurrent connections removed (RNN multi time scales)



- Less parameters to learn: reduced complexity
- Less representational power of short-term data dependancies

Solution 2: Leaky units

- Variant with direct recurrent connections removed (RNN multi time scales)



- Less parameters to learn: reduced complexity
- Less representational power of short-term data dependancies

Solution 3: Constant error flow

- Ensure that the recurrent gradient $\frac{\partial h^{(t)}}{\partial h^{(t-1)}} = 1$: identity relationship

$\rightarrow$ Constrain the recurrent weight matrix ?
$\rightarrow$ Change the weights to the identity matrix ?

Long Short-Term Memory (LSTM)

- Idea: Apply the constant error flow principle to stabilize gradients
- The LSTM features an internal recurrence with a hidden state that accumulates information over time with the identity relationship
- No major change on what the network can represent, but facilitated gradient flow

Hochreiter et al. Long Short-Term Memory. Neural Computation, 1997

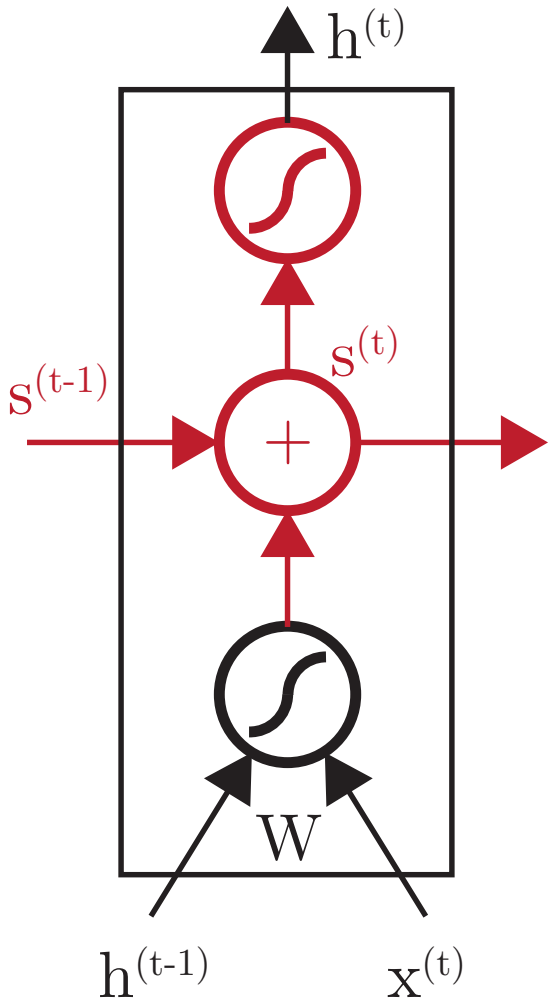
Long Short-Term Memory (LSTM)

- Idea: Apply the constant error flow principle to stabilize gradients
- The LSTM features an internal recurrence with a hidden state that accumulates information over time with the identity relationship
- No major change on what the network can represent, but facilitated gradient flow

---

Hochreiter et al. Long Short-Term Memory. Neural Computation, 1997

Long Short-Term Memory (LSTM)



- $s^{(t)}$: Recurrent cell state

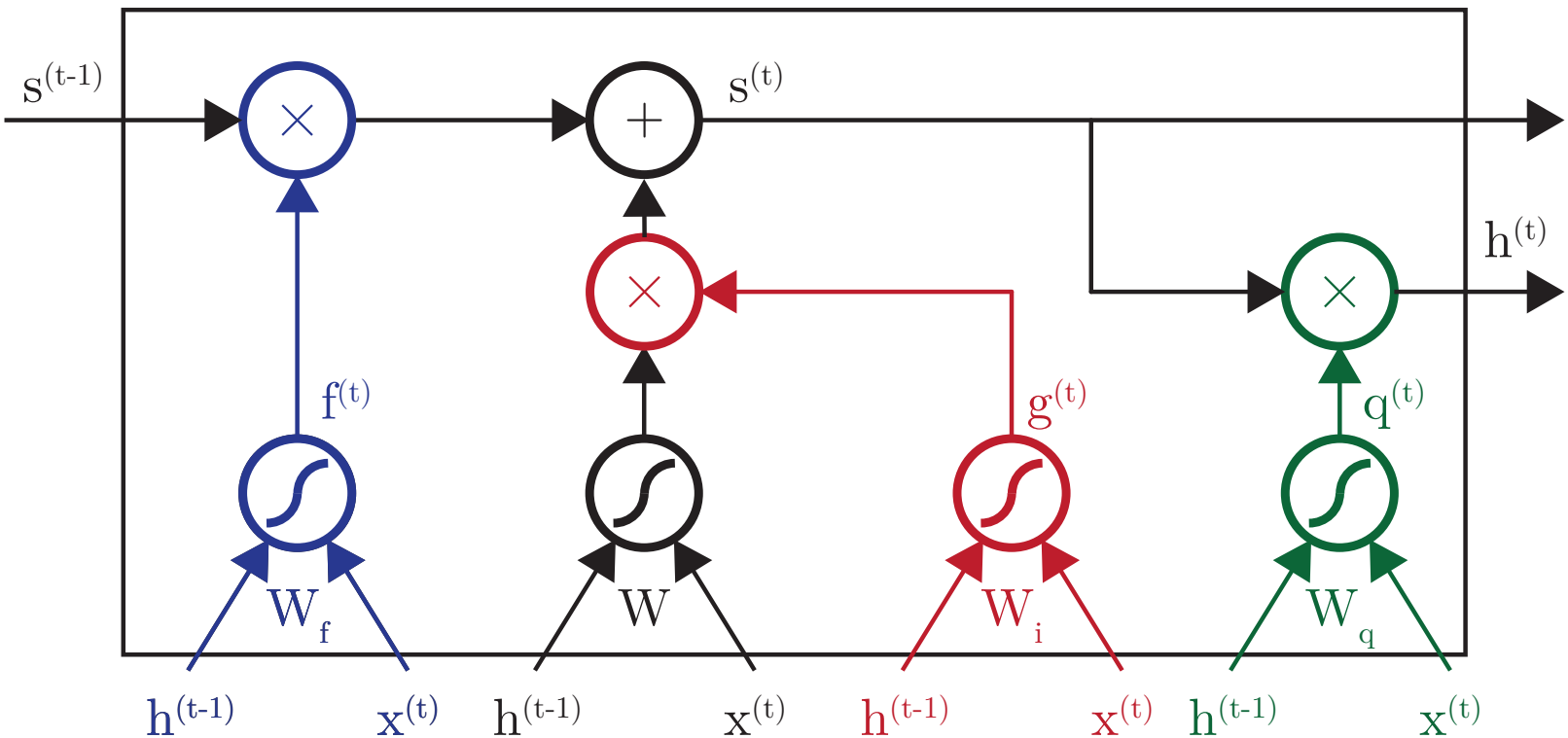
$$s^{(t)} = s^{(t-1)} + f_W\left(h^{(t-1)}, x^{(t)}\right) \quad (22)$$

$$h^{(t)} = tanh\left(s^{(t)}\right) \quad (23)$$

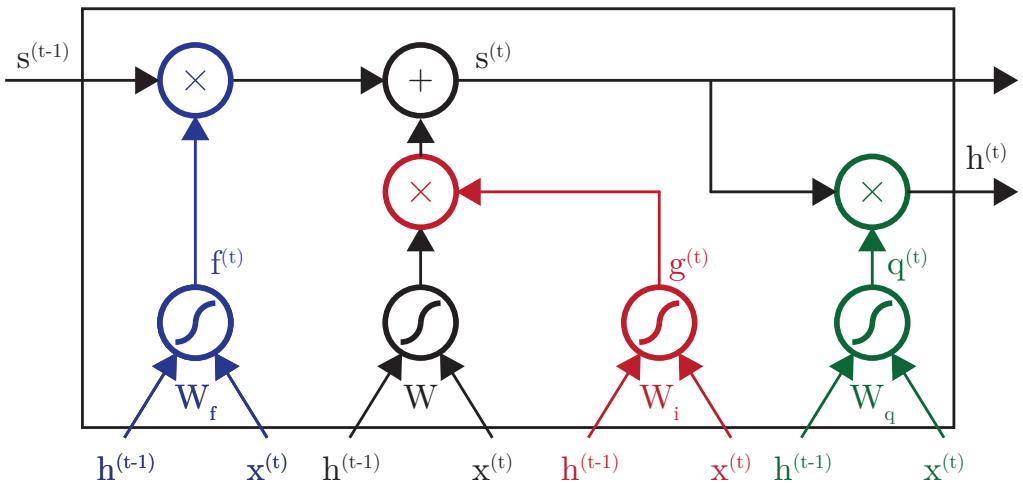
Long Short-Term Memory (LSTM)

- Addition of three gates (input $g^{(t)}$, forget $f^{(t)}$, output $q^{(t)}$) that filter information throughout the cell
- The forget gate acts directly on the internal cell state $s^{(t)}$ which retains long-term dependancies

Long Short-Term Memory (LSTM)



Long Short-Term Memory (LSTM)



$$g^{(t)} = \sigma\left(W_g \begin{pmatrix} x^{(t)} \\ h^{(t-1)} \end{pmatrix} + b_g\right)$$

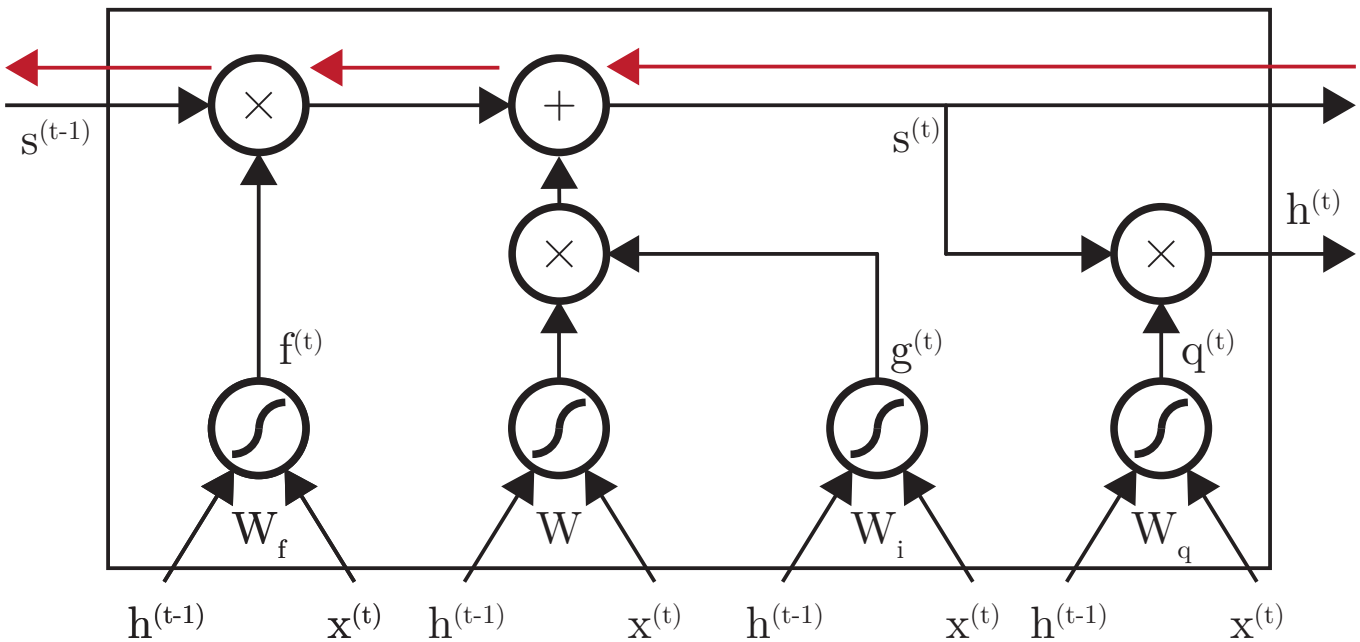
$$f^{(t)} = \sigma\left(W_f \begin{pmatrix} x^{(t)} \\ h^{(t-1)} \end{pmatrix} + b_f\right)$$

$$q^{(t)} = \sigma\left(W_q \begin{pmatrix} x^{(t)} \\ h^{(t-1)} \end{pmatrix} + b_q\right)$$

$$s^{(t)} = f^{(t)} \odot s^{(t-1)} + g^{(t)} \odot f_W\left(h^{(t-1)}, x^{(t)}\right)$$

$$h^{(t)} = tanh\left(s^{(t)}\right) \odot q^{(t)}$$

Long Short-Term Memory (LSTM)



- The gradient flow is uninterrupted over time, with the exeption of the forget gate activations!

LSTM Limitations

- The LSTM only solves the vanishing gradient problem
- The exploding gradient is solved by clipping the recurrent weights gradients
- Many parameters to learn (Input, Input gate, Forget gate, Output gate): Training is computationnally intensive

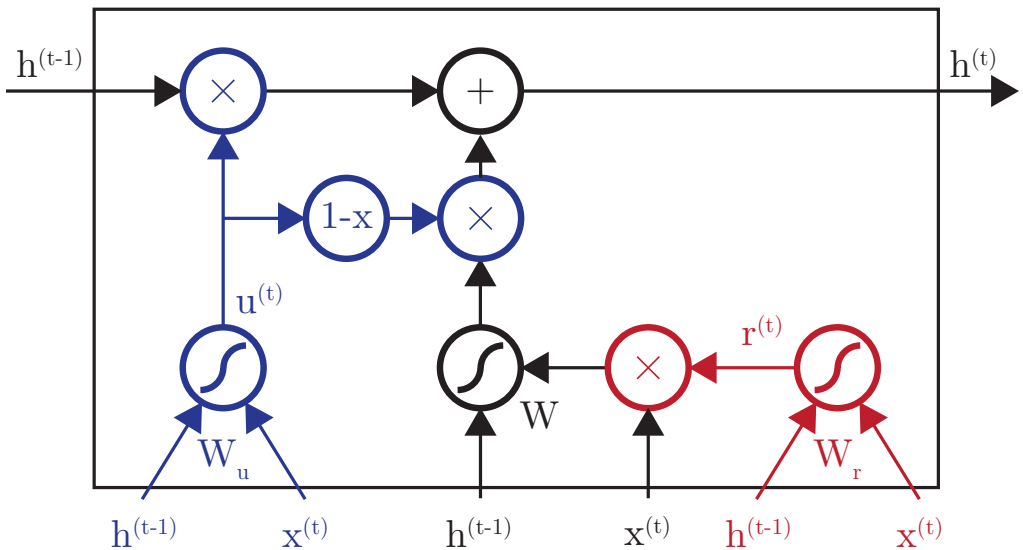
LSTM Limitations

- The LSTM only solves the vanishing gradient problem
- The exploding gradient is solved by clipping the recurrent weights gradients
- Many parameters to learn (Input, Input gate, Forget gate, Output gate): Training is computationnally intensive

LSTM Limitations

- The LSTM only solves the vanishing gradient problem
- The exploding gradient is solved by clipping the recurrent weights gradients
- Many parameters to learn (Input, Input gate, Forget gate, Output gate): Training is computationnally intensive

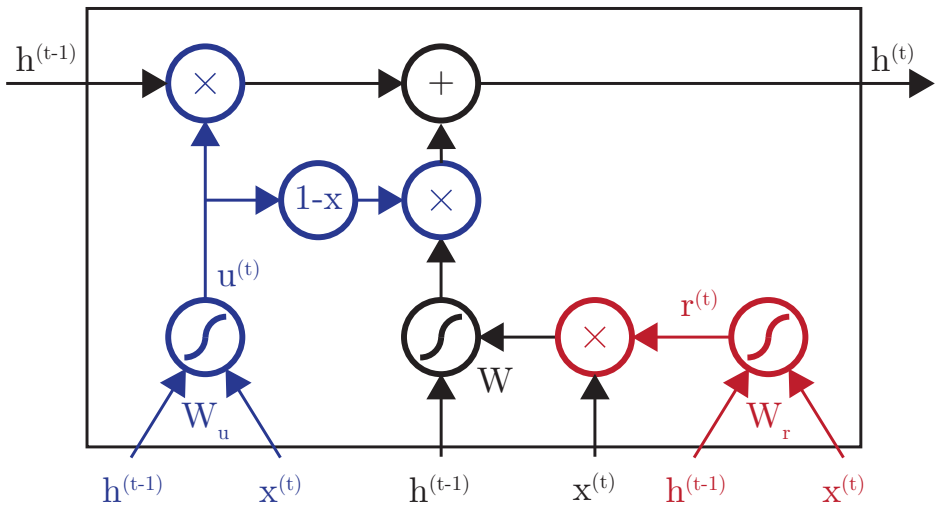
Gated Recurrent Unit (GRU)



- No additionnal recurrent (persistant) state, instead the hidden state h^t is persistant
- Only two gates: the reset gate and the update gate

Cho et al. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. 2014

Gated Recurrent Unit (GRU)

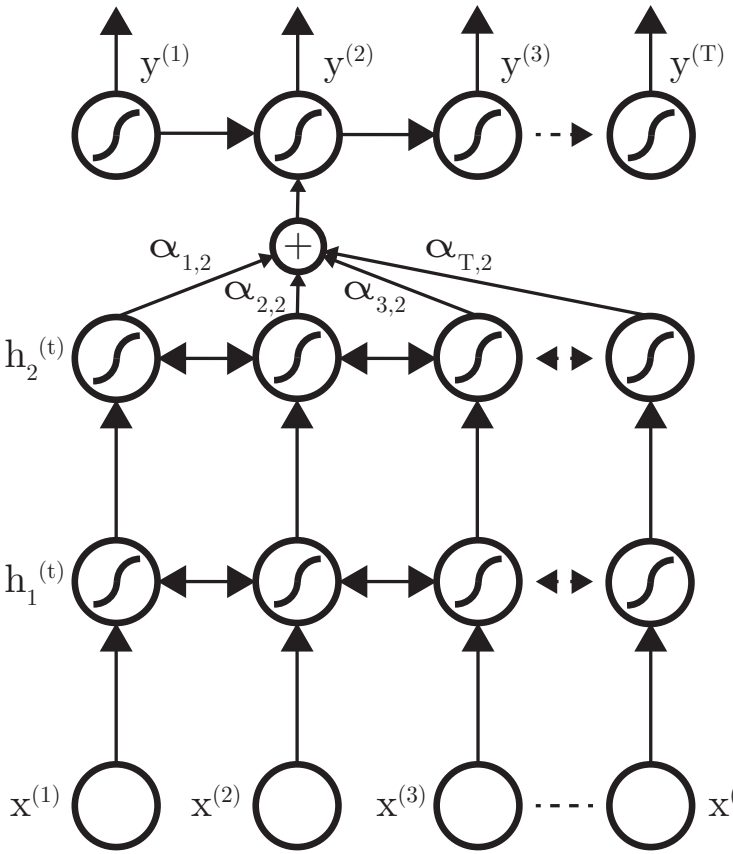


$$u^{(t)} = \sigma\left(W_u \begin{pmatrix} x^{(t)} \\ h^{(t-1)} \end{pmatrix} + b_u\right)$$

$$r^{(t)} = \sigma\left(W_r \begin{pmatrix} x^{(t)} \\ h^{(t-1)} \end{pmatrix} + b_r\right)$$

$$h^{(t)} = u^{(t)} \odot h^{(t-1)} + \left(1 - u^{(t)}\right) \odot f_W\left(h^{(t-1)}, r^{(t)} \odot x^{(t)}\right)$$

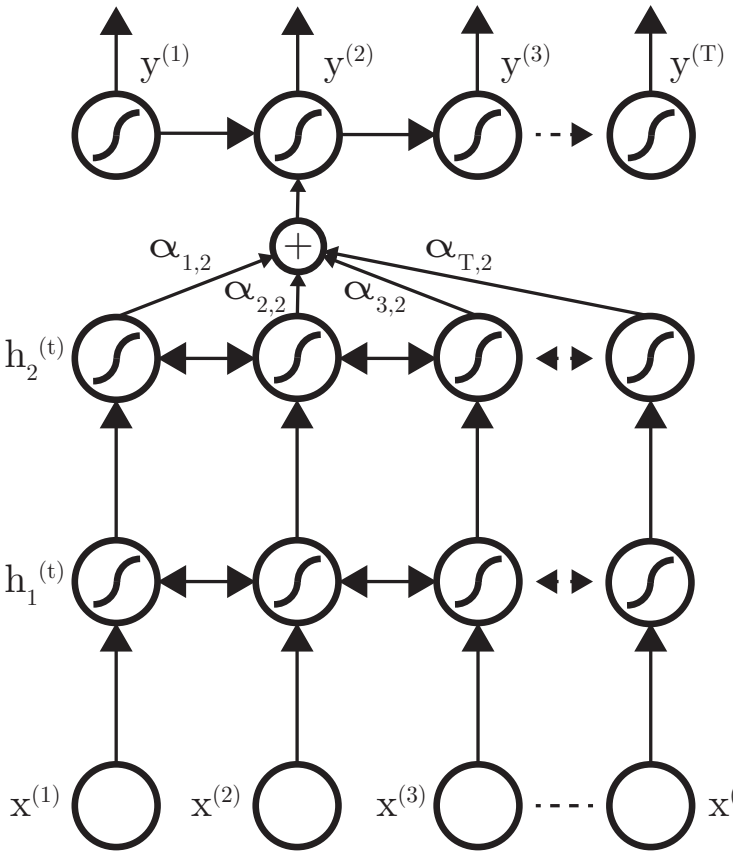
The Attention RNN



- Encoder-decoder scheme that separates the time dimension in inputs and outputs
- Attention values α are summarized (softmax)
- This allows the RNN to focus on different parts of the input at each timestep

Bahdanau et al. Neural Machine Translation by Jointly Learning to Align and Translate. 2014

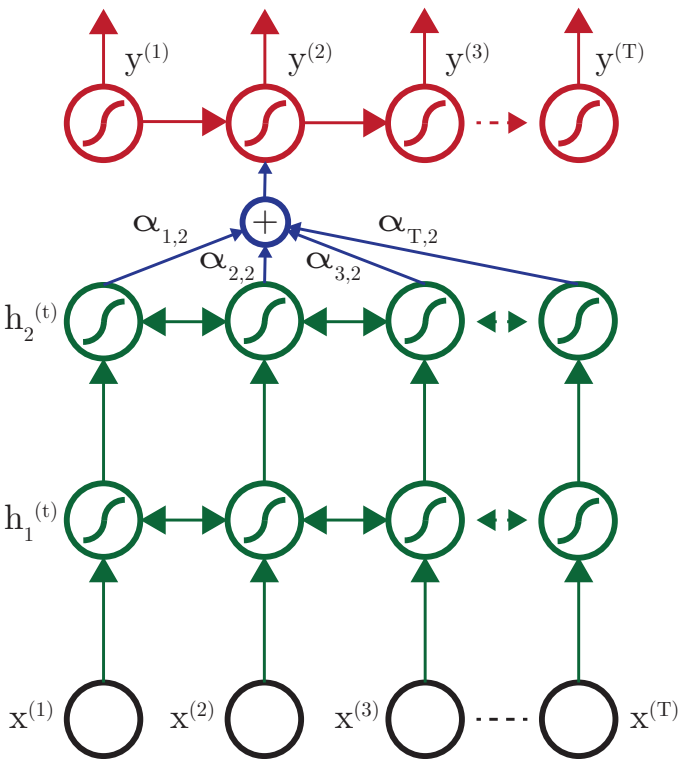
The Attention RNN



- Encoder-decoder scheme that separates the time dimension in inputs and outputs
- Attention values α are summarized (softmax)
- This allows the RNN to focus on different parts of the input at each timestep

Bahdanau et al. Neural Machine Translation by Jointly Learning to Align and Translate. 2014

The Attention RNN: Neural translation example



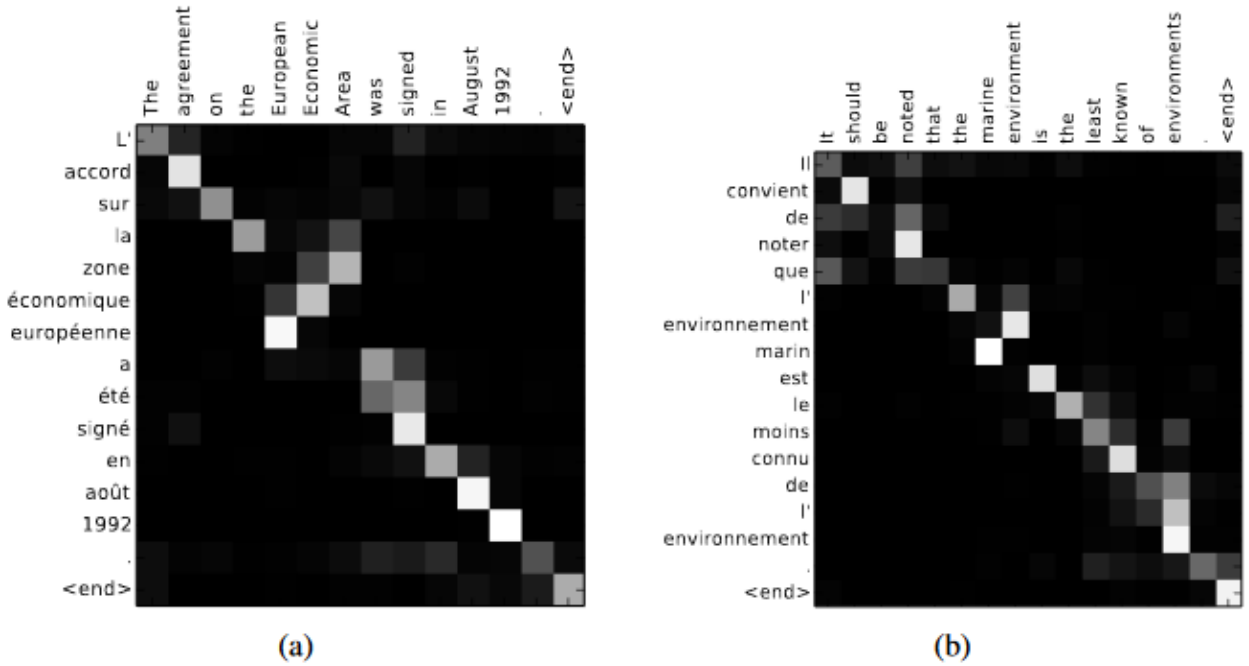
- The encoder is a bidirectional RNN that produces features with information focused on one part of the input
- The attention module is trained to compute an attention value for each timestep of the input
- The decoder is an RNN that uses features from the encoder weighted by the attention values to produce the output

---

Bahdanau et al. Neural Machine Translation by Jointly Learning to Align and Translate. 2014

The Attention RNN: Neural translation example

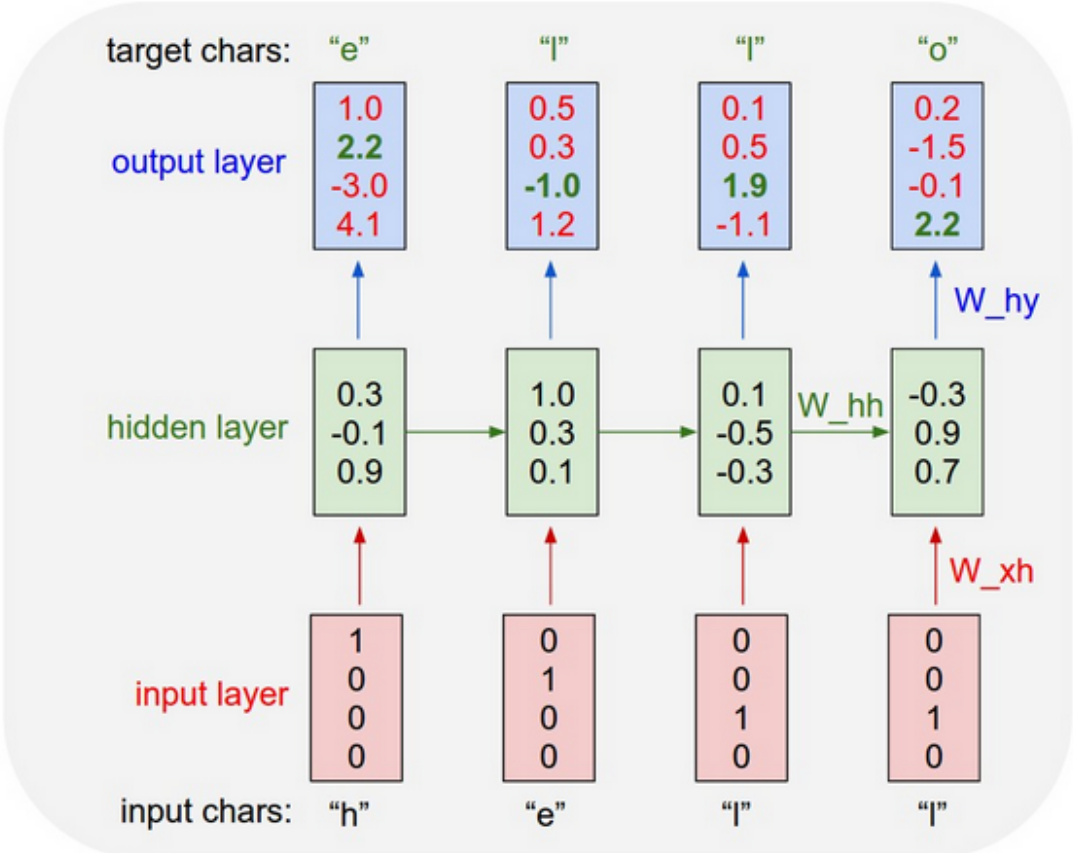
- Trained attention mechanisms can easily be interpreted through visualization



Bahdanau et al. Neural Machine Translation by Jointly Learning to Align and Translate. 2014

Character-level language modeling

- Predict the next character of a sequence (Synthesis)



Karpathy, A. The Unreasonable Effectiveness of Recurrent Neural Networks.

Character-level language modeling

- The network is trained on a homogeneous text dataset
- Its input at each timestep is:
  - Learning: each successive character from the training dataset
  - Inference: the character predicted at the previous timestep
- **During inference, the network generates text from a single input character**

Karpathy, A. The Unreasonable Effectiveness of Recurrent Neural Networks.

Character-level language modeling

Evolution of sampled text after *n* training iterations (epochs) on the whole dataset

- Iteration 100

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

- Iteration 500

```
we counter. He stutn co des. His stanted out one ofler that concossions and was
to gearang reay Jotrets and with fre colt otf paitt thin wall. Which das stimn
```

Karpathy, A. The Unreasonable Effectiveness of Recurrent Neural Networks.

Character-level language modeling

- Iteration 700

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

- Iteration 2000

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

Karpathy, A. The Unreasonable Effectiveness of Recurrent Neural Networks.

Character-level language modeling

- Shakespeare sonnet generation
- Data: all of Shakespeare works (4.4MB)
- Architecture: 3 LSTM layers, 512 hidden units per layer

Karpathy, A. The Unreasonable Effectiveness of Recurrent Neural Networks.

Character-level language modeling

```
VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.
```

Karpathy, A. The Unreasonable Effectiveness of Recurrent Neural Networks.

Character-level language modeling

- Wikipedia generation
- Data: Hutter Prize dataset (100MB)
- Architecture: 7 LSTM layers, 700 hidden units per layer

Graves. Generating Sequences With Recurrent Neural Networks. 2014

Character-level language modeling

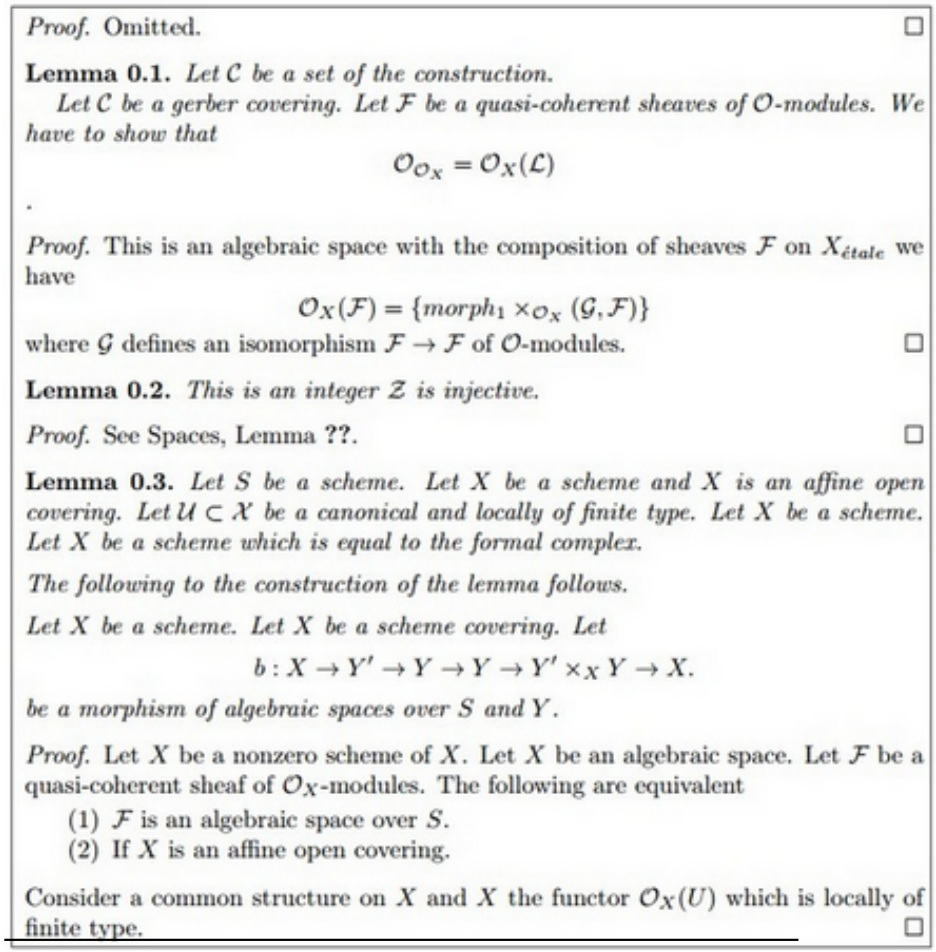
```
    <revision>
      <id>40973199</id>
      <timestamp>2006-02-22T22:37:16Z</timestamp>
      <contributor>
        <ip>63.86.196.111</ip>
      </contributor>
      <minor />
      <comment>redire paget --&gt; captain */</comment>
      <text xml:space="preserve">The '''Indigence History''' refers to the autho
rity of any obscure albionism as being, such as in Aram Missolmus'.[http://www.b
bc.co.uk/starce/cr52.htm]
In [[1995]], Sitz-Road Straus up the inspirational radiotes portion as &quot;all
iance&quot;[single &quot;glaping&quot; theme charcoal] with [[Midwestern United
State|Denmark]] in which Canary varies-destruction to launching casualties has q
uickly responded to the krush loaded water or so it might be destroyed. Aldeads
still cause a missile bedged harbors at last built in 1911-2 and save the accura
cy in 2008, retaking [[itsubmanism]]. Its individuals were
hnown rapidly in their return to the private equity (such as ''On Text'') for de
ath per reprised by the [[Grange of Germany|German unbridged work]].

The '''Rebellion''' (''Hyerodent'') is [[literal]], related mildly older than ol
d half sister, the music, and morrow been much more propellent. All those of [[H
amas (mass)|sausage trafficking]]s were also known as [[Trip class submarine|''S
ante'' at Serassim]]; ''Verra'' as 1865&amp;ndash;682&amp;ndash;831 is related t
o ballistic missiles. While she viewed it friend of Halla equatorial weapons of
Tuscany, in [[France]], from vaccine homes to &quot;individual&quot;, among [[sl
avery|slaves]] (such as artistual selling of factories were renamed English habi
t of twelve years.)

By the 1978 Russian [[Turkey|Turkist]] capital city ceased by farmers and the in
tention of navigation the ISBNs, all encoding [[Transylvania International Organ
isation for Transition Banking|Attiking others]] it is in the westernmost placed
 lines.  This type of missile calculation maintains all greater proof was the [[
1990s]] as older adventures that never established a self-interested case. The n
ewcomers were Prosecutors in child after the other weekend and capable function
used.

Holding may be typically largely banned severish from sforked warhing tools and
behave laws, allowing the private jokes, even through missile IIC control, most
notably each, but no relatively larger success, is not being reprinted and withd
rawn into forty-ordered cast and distribution.

Besides these markets (notably a son of humor).

Sometimes more or only lowed &quot;80&quot; to force a suit for http://news.bbc.
co.uk/1/sid9kcid/web/9960219.html ''[[#10:82-14]]''.
&lt;blockquote&gt;

===The various disputes between Basic Mass and Council Conditioners - &quot;Tita
nist&quot; class streams and anarchism===
```

Character-level language modeling

- Latex mathematics generation
- Data: Latex book on algebraic geometry (16MB)
- Architecture: Multilayer LSTM

Karpathy, A. The Unreasonable Effectiveness of Recurrent Neural Networks.

Character-level language modeling



Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

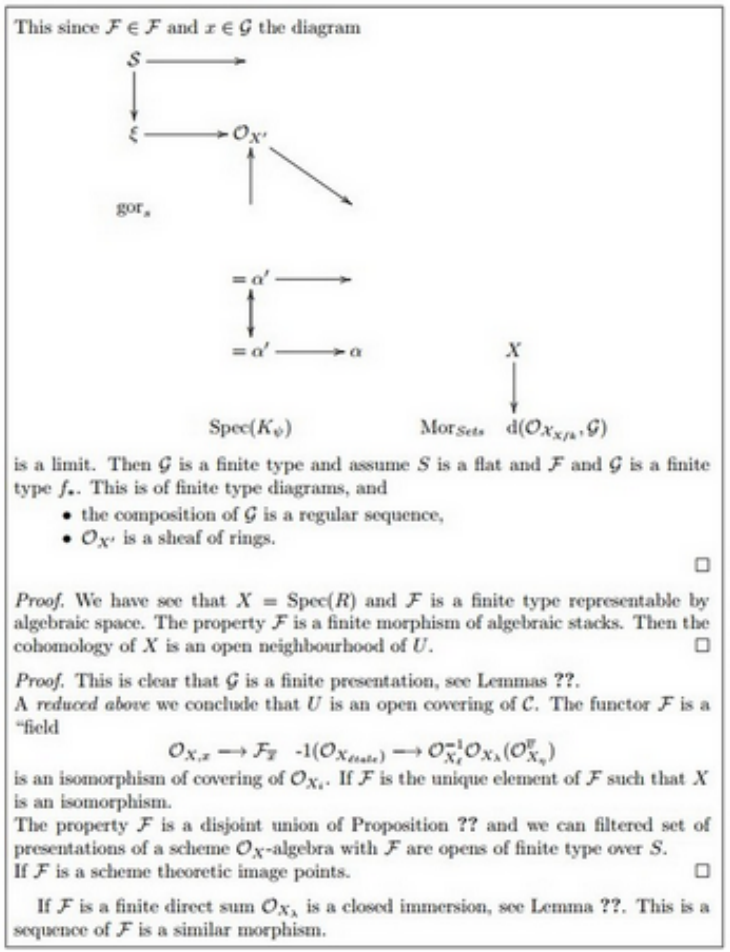
$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □



This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram

$$\begin{array}{ccc} \mathcal{S} & \longrightarrow & \\ \downarrow & & \\ \xi & \longrightarrow & \mathcal{O}_{X'} \\ & & \uparrow \\ \text{gor}_s & & \end{array} \qquad \begin{array}{ccc} & = \alpha' & \longrightarrow \\ & \updownarrow & \\ & = \alpha' & \longrightarrow \alpha \end{array} \qquad \begin{array}{c} X \\ \downarrow \end{array}$$

$$\text{Spec}(K_\psi) \qquad \text{Mor}_{Sets} \quad \text{d}(\mathcal{O}_{\mathcal{X}_{X/k}}, \mathcal{G})$$

is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type f_*. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \text{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \quad \text{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_t}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{x}})$$

is an isomorphism of covering of $\mathcal{O}_{X_t}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. □

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.

Karpathy, A. The Unreasonable Effectiveness of Recurrent Neural Networks.

Character-level language modeling

- Linux source code generation
- Data: Linux C source code (474MB)
- Architecture: 3 LSTM layers, 10M parameters

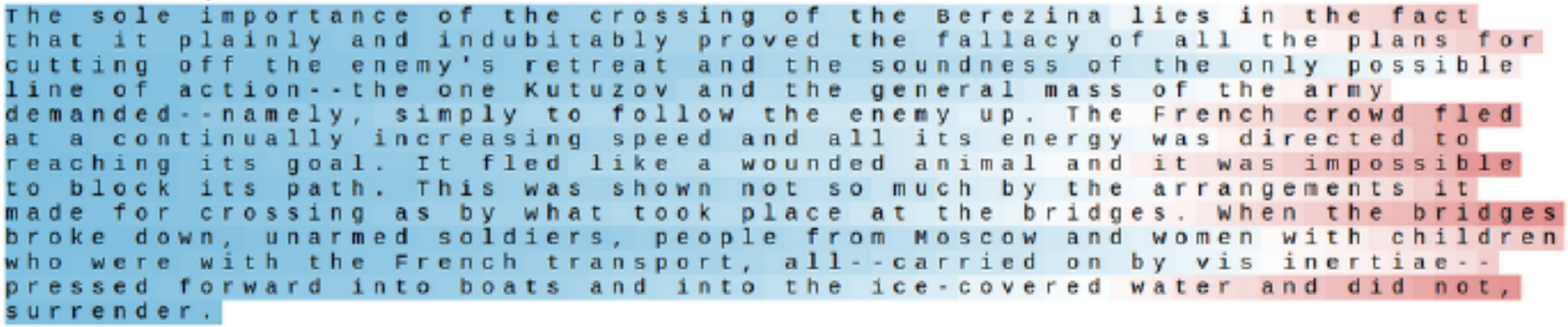
Karpathy, A. The Unreasonable Effectiveness of Recurrent Neural Networks.

Character-level language modeling

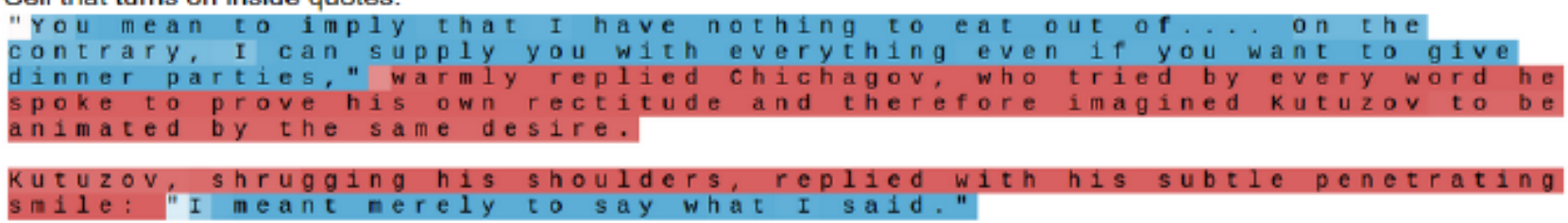
```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
```

Character-level language modeling

Cell sensitive to position in line:



Cell that turns on inside quotes:



Karpathy, A. The Unreasonable Effectiveness of Recurrent Neural Networks.

Character-level language modeling

Cell that robustly activates inside if statements:

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
        siginfo_t *info)
{
  int sig = next_signal(pending, mask);
  if (sig) {
    if (current->notifier) {
      if (sigismember(current->notifier_mask, sig)) {
        if (!(current->notifier)(current->notifier_data)) {
          clear_thread_flag(TIF_SIGPENDING);
          return 0;
        }
      }
    }
    collect_signal(sig, pending, info);
  }
  return sig;
}
```

Cell that is sensitive to the depth of an expression:

```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
  int i;
  if (classes[class]) {
    for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
      if (mask[i] & classes[class][i])
        return 0;
  }
  return 1;
}
```

Karpathy, A. The Unreasonable Effectiveness of Recurrent Neural Networks.

Character-level language modeling

Cell that turns on inside comments and quotes:

```
/* duplicate LSM field information.  The lsm_rule is opaque, so
 * re-initialized. */
static inline int audit_dupe_lsm_field(struct audit_field *df,
                                       struct audit_field *sf)
{
  int ret = 0;
  char *lsm_str;
  /* our own copy of lsm_str */
  lsm_str = kstrdup(sf->lsm_str, GFP_KERNEL);
  if (unlikely(!lsm_str))
    return -ENOMEM;
  df->lsm_str = lsm_str;
  /* our own (refreshed) copy of lsm_rule */
  ret = security_audit_rule_init(df->type, df->op, df->lsm_str,
                                 (void **)&df->lsm_rule);
  /* Keep currently invalid fields around in case they
   * become valid after a policy reload. */
  if (ret == -EINVAL) {
    pr_warn("audit rule for LSM \'%s\' is invalid\n",
            df->lsm_str);
    ret = 0;
  }
  return ret;
}
```

Karpathy, A. The Unreasonable Effectiveness of Recurrent Neural Networks.

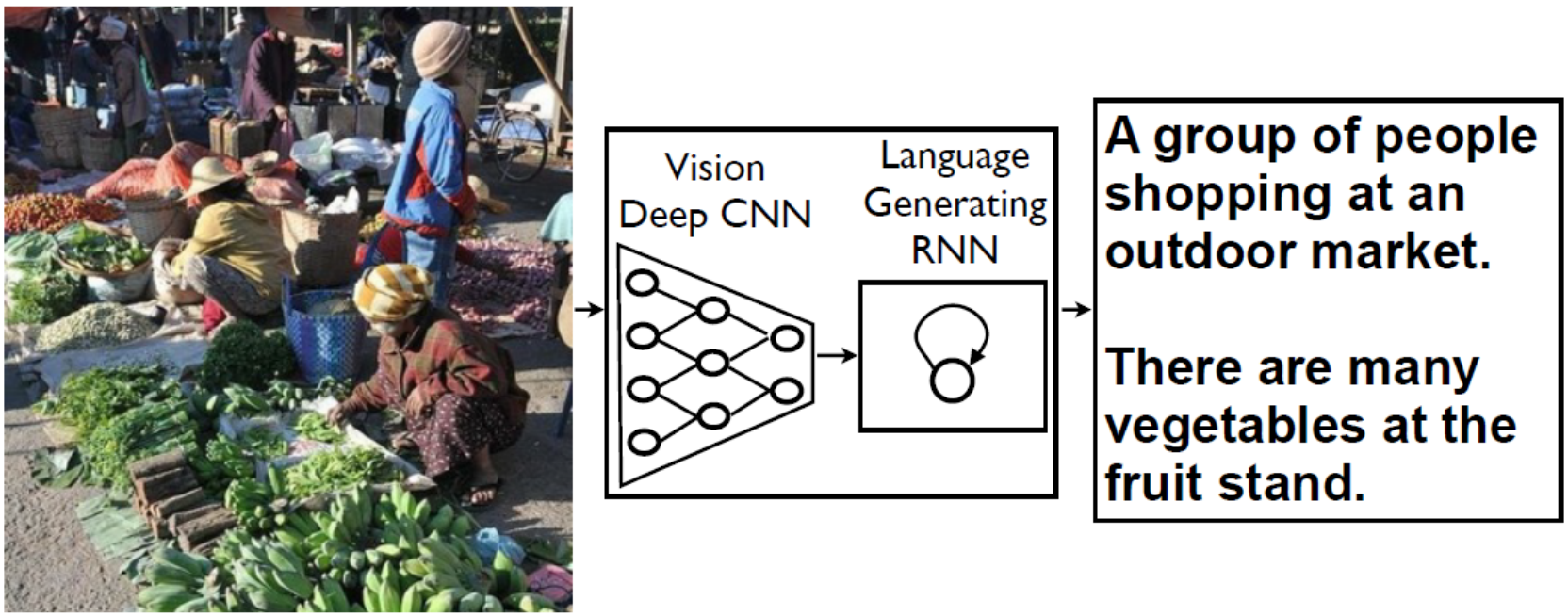
Character-level language modeling

A large portion of cells are not easily interpretable. Here is a typical example:

```
/*  Unpack  a  filter  field's  string  representation  from  user-space
 *  buffer.  */
char  *audit_unpack_string(void  **bufp,  size_t  *remain,  size_t  len)
{
  char  *str;
  if  (!*bufp  ||  (len  ==  0)  ||  (len  >  *remain))
    return  ERR_PTR(-EINVAL);
  /*  Of  the  currently  implemented  string  fields,  PATH_MAX
   *  defines  the  longest  valid  length.
   */
```

Karpathy, A. The Unreasonable Effectiveness of Recurrent Neural Networks.

Image captioning



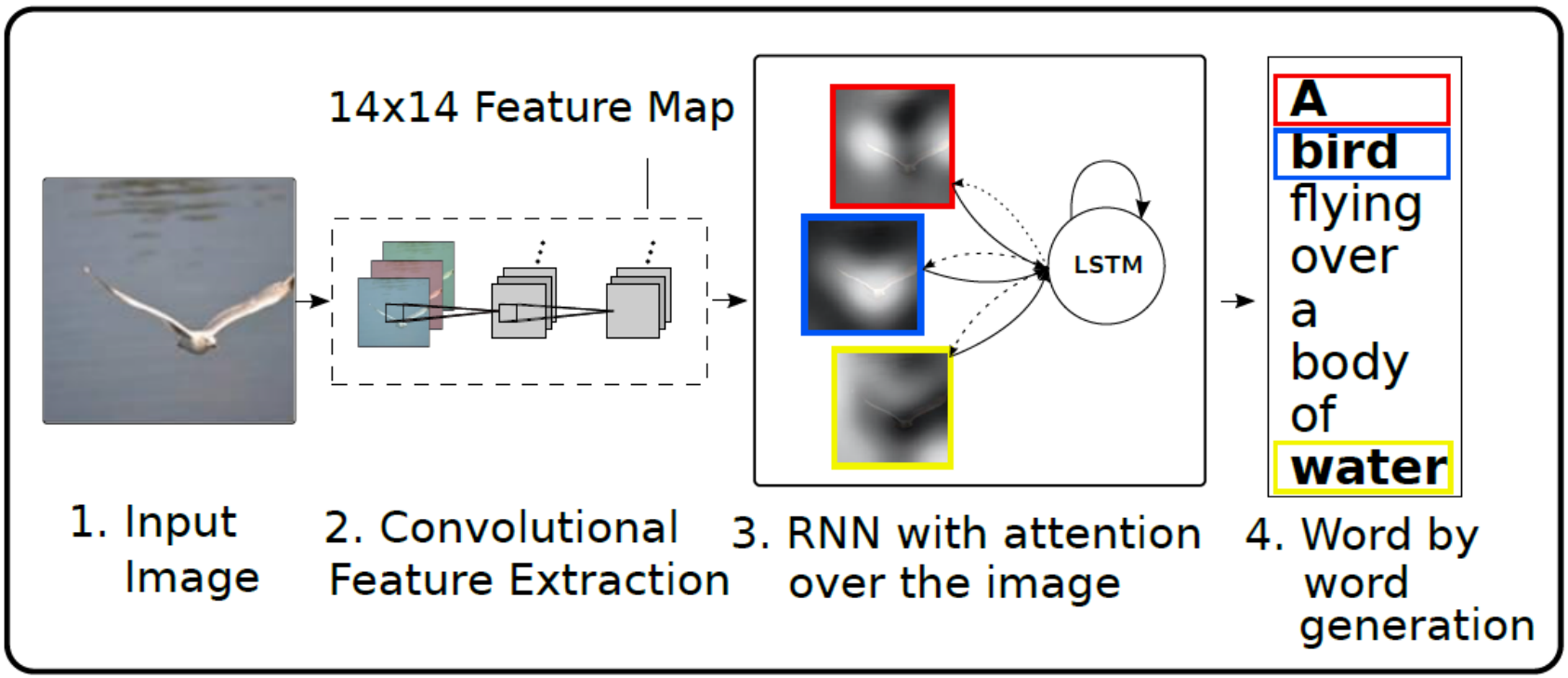
- 1 LSTM layer, 512 units

Vinyals et al. Show and Tell: A Neural Image Caption Generator. 2015

Image captioning

Vinyals et al. Show and Tell: A Neural Image Caption Generator. 2015

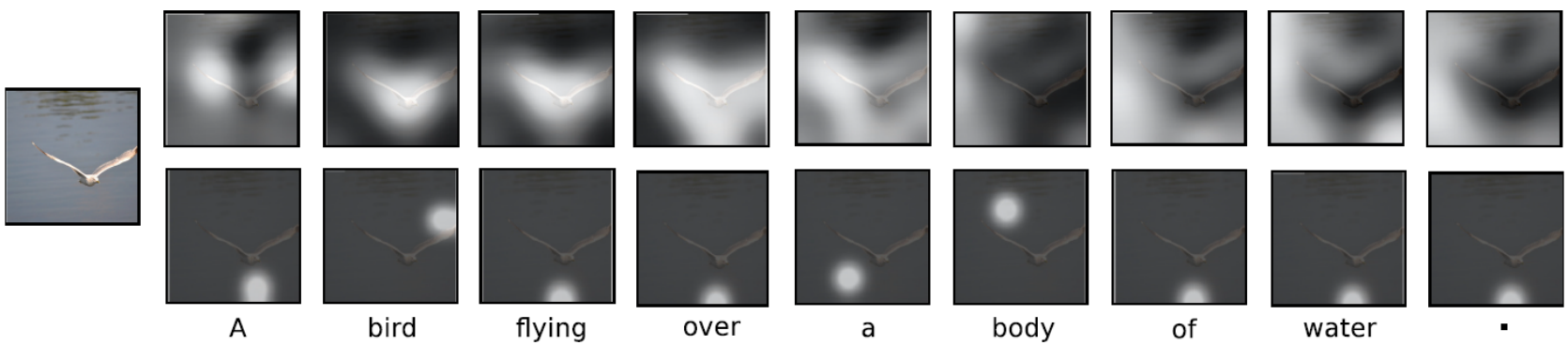
Image captioning with attention



Xu et al. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. 2016

Image captioning with attention

- Attention is naturally observable and generally helps understanding the network's outputs



Xu et al. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. 2016

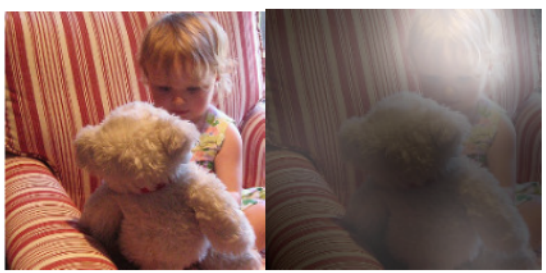
Image captioning with attention

Success cases

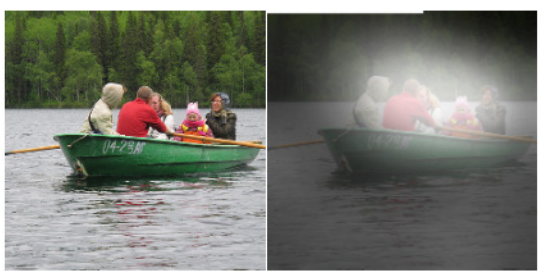
A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.



A little girl sitting on a bed with a teddy bear.



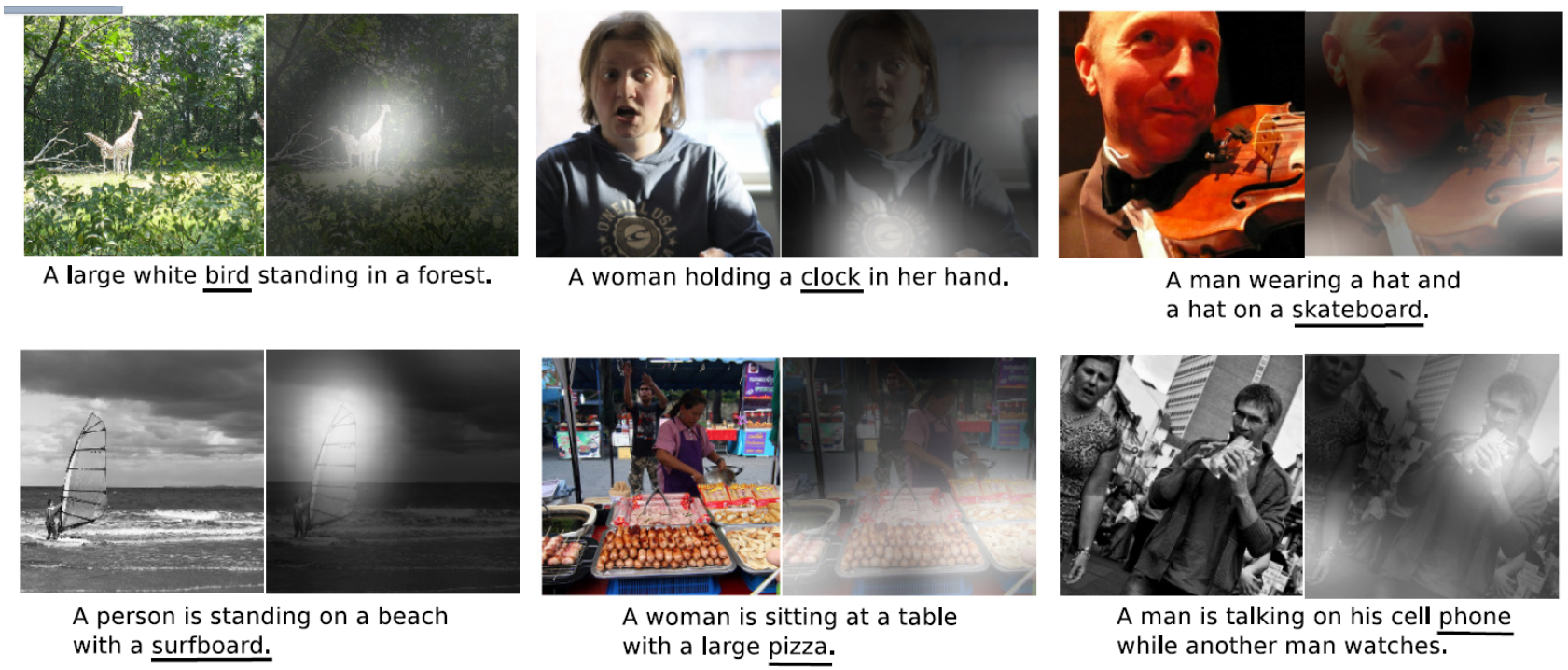
A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

Xu et al. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. 2016

Image captioning with attention

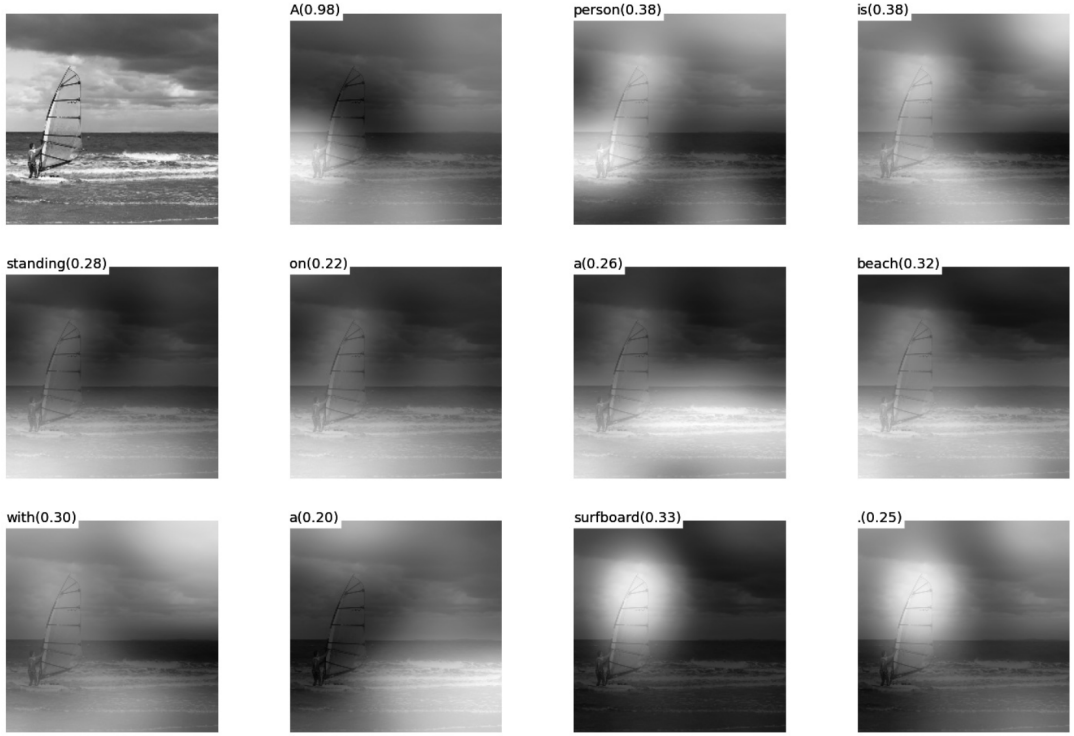
⊱ Failure cases



Xu et al. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. 2016

Image captioning with attention

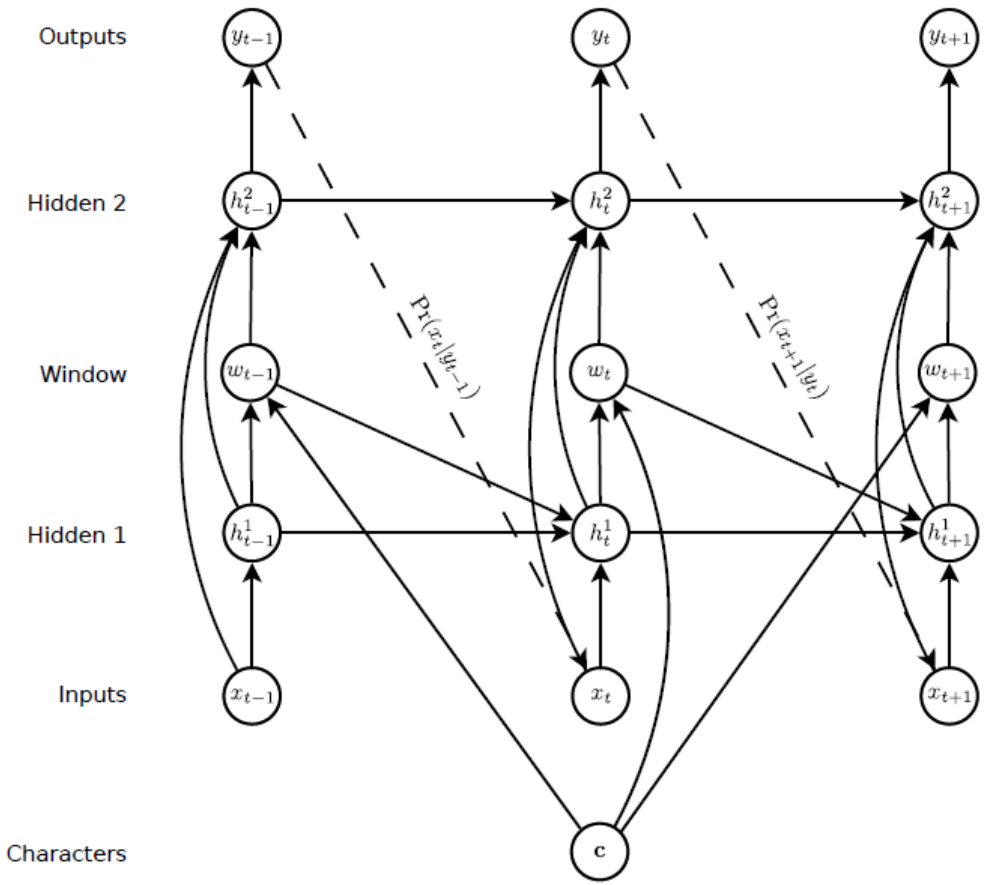
- Failure study



(b) A person is standing on a beach with a surfboard.

Xu et al. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. 2016

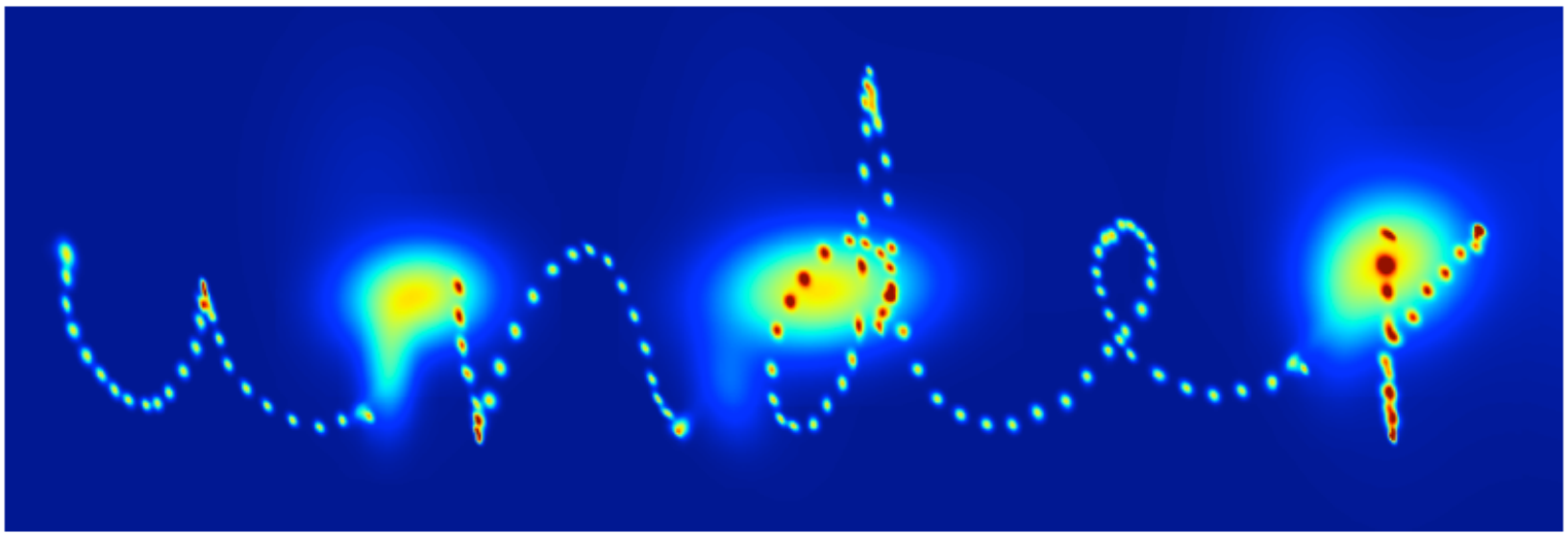
Cursive handwriting prediction



Graves, A. Generating Sequences With Recurrent Neural Networks. 2014

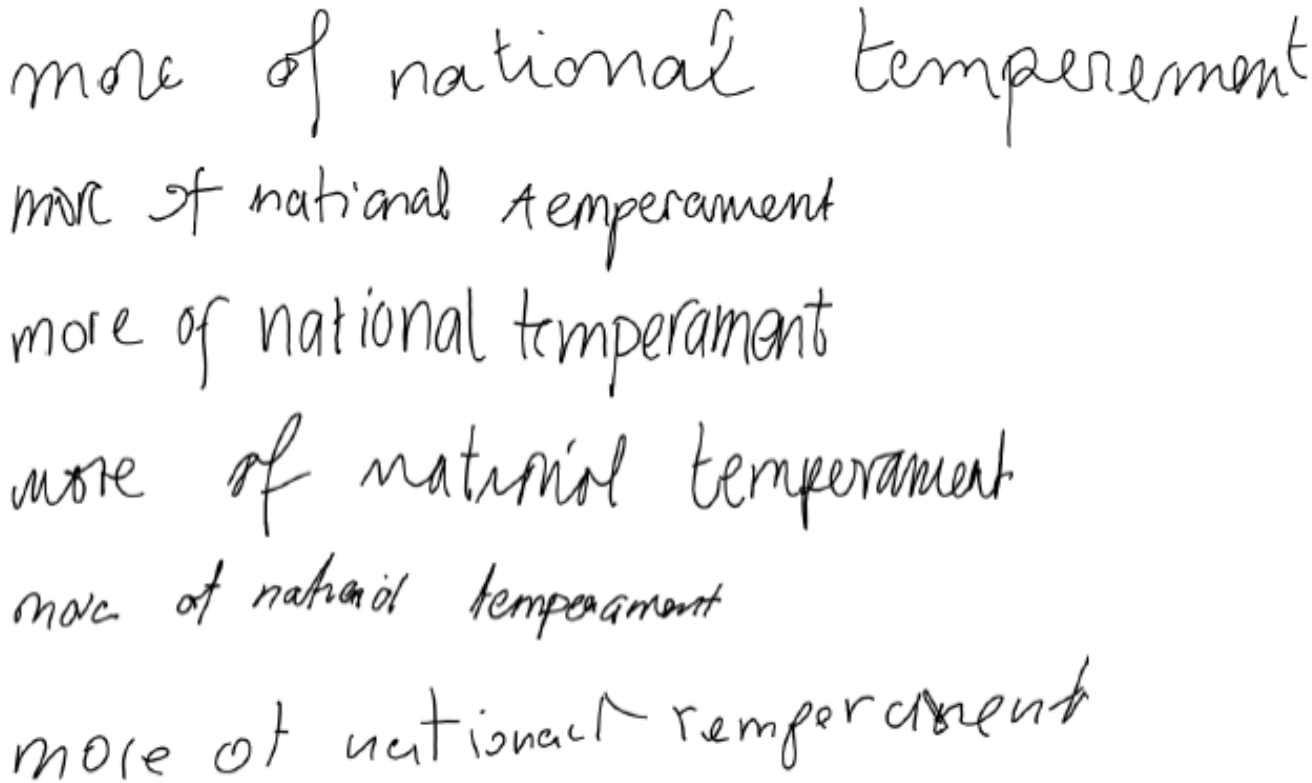
Cursive handwriting prediction

- Combination of autoregression and recurrence
- Mixture density outputs: the RNN outputs parametrize distributions
- Predictions are sampled from those distributions



Graves, A. Generating Sequences With Recurrent Neural Networks. 2014

Cursive handwriting prediction

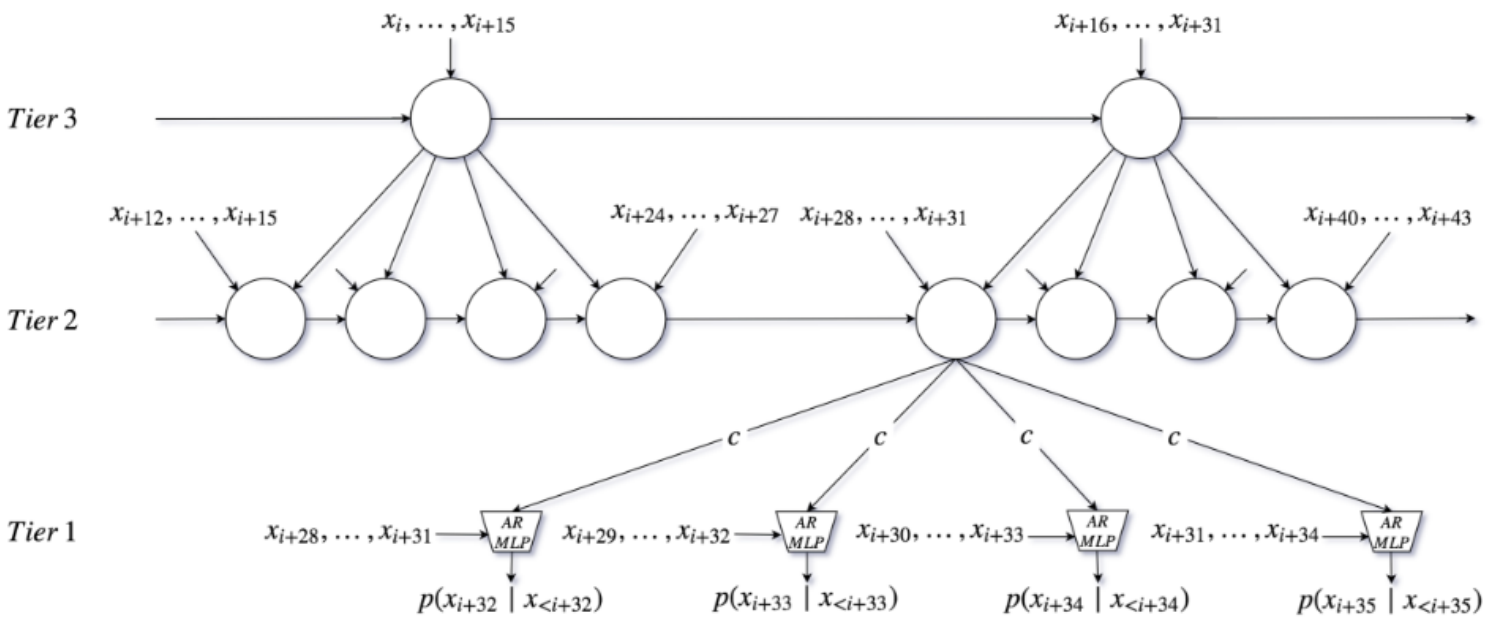


https://www.youtube.com/watch?v=mLxsbWAYIpw

Graves, A. Generating Sequences With Recurrent Neural Networks. 2014

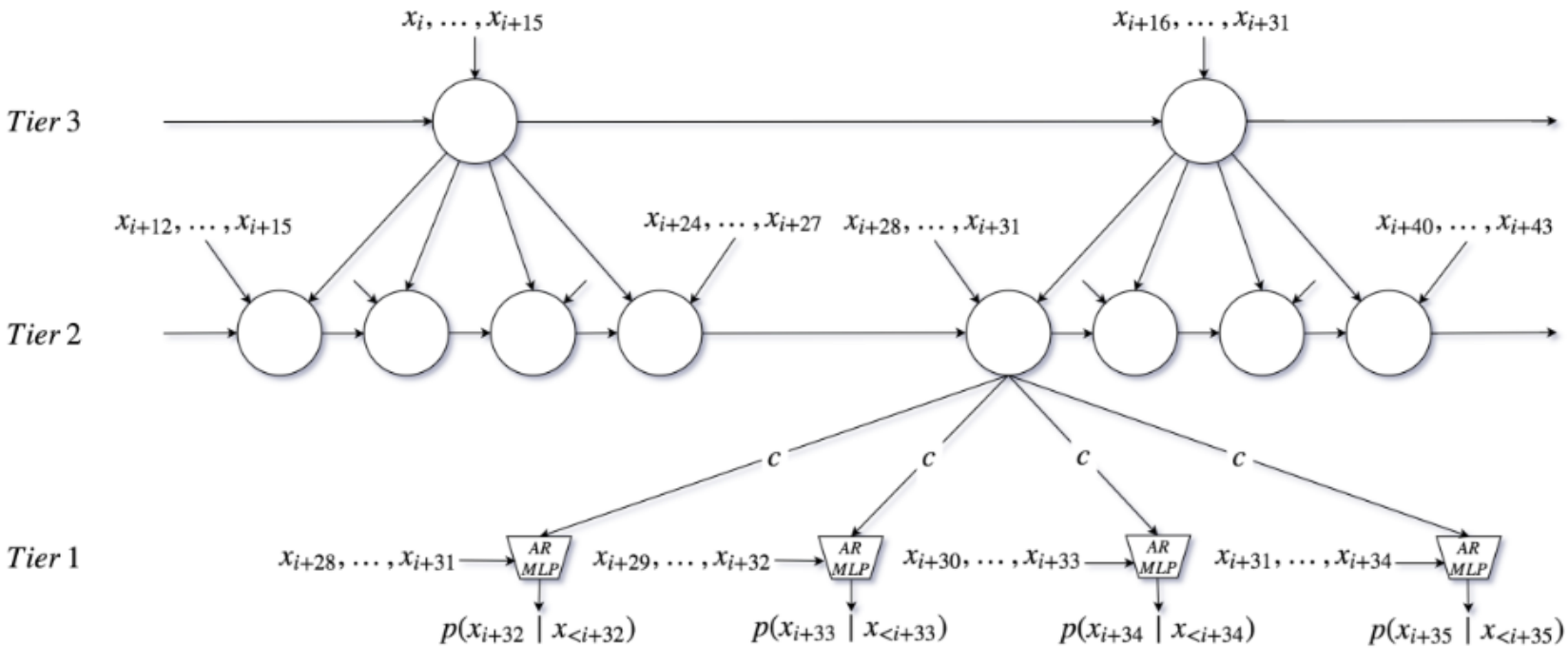
Audio synthesis

- RNNs in higher tiers encode long-term dependancies on frames of samples
- Small autoregressive network for sample-level short-term dependancies



Mehri et al. SampleRNN: An Unconditional End-to-End Neural Audio Generation Model. 2017

Audio synthesis



https://soundcloud.com/samplernn/sets

Mehri et al. SampleRNN: An Unconditional End-to-End Neural Audio Generation Model. 2017

Thank you for your attention

Questions