Recurrent Neural Network

Xiaogang Wang

xgwang@ee.cuhk.edu.hk

March 1, 2015

Outline

Sequential data

- Sequence of words in an English sentence
- Acoustic features at successive time frames in speech recognition
- Successive frames in video classification
- Rainfall measurements on successive days in Hong Kong
- Daily values of current exchange rate
- Nucleotide base pairs in a strand of DNA
- **Instead of making independent predictions on samples, assume the dependency among samples and make a sequence of decisions for sequential samples**

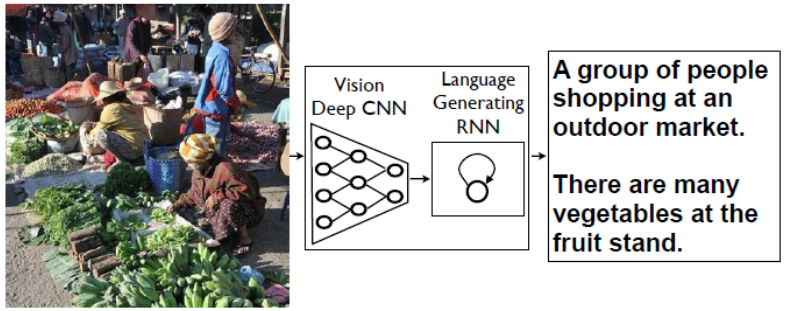
Modeling sequential data

- Sample data sequences from a certain distribution

$$P(\mathbf{x}_1, \ldots, \mathbf{x}_T)$$

- Generate natural sentences to describe an image

$$P(\mathbf{y}_1, \ldots, \mathbf{y}_T | I)$$



- Activity recognition from a video sequence

$$P(\mathbf{y}|\mathbf{x}_1, \ldots, \mathbf{x}_T)$$

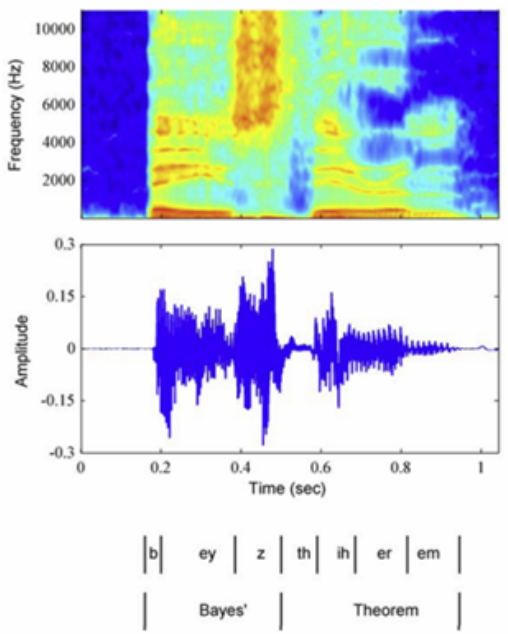
Modeling sequential data

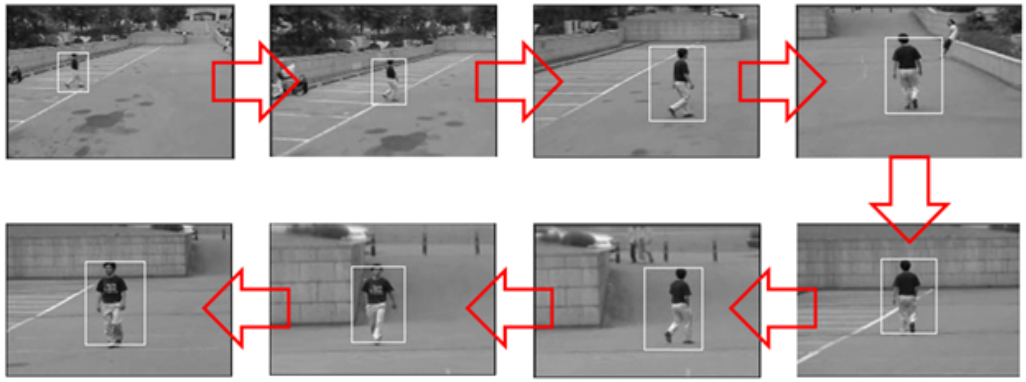
- Speech recognition

$$P(\mathbf{y}_1, \ldots, \mathbf{y}_T | \mathbf{x}_1, \ldots, \mathbf{x}_T)$$

- Object tracking

$$P(\mathbf{y}_1, \ldots, \mathbf{y}_T | \mathbf{x}_1, \ldots, \mathbf{x}_T)$$





Modeling sequential data

- Generate natural sentences to describe a video

$$P(\mathbf{y}_1, \ldots, \mathbf{y}_{T'} | \mathbf{x}_1, \ldots, \mathbf{x}_T)$$

- Language translation

$$P(\mathbf{y}_1, \ldots, \mathbf{y}_{T'} | \mathbf{x}_1, \ldots, \mathbf{x}_T)$$

Ulrich UNK , membre du conseil d' administration du constructeur automobile Audi , affirme qu' il s' agit d' une pratique courante depuis des années pour que les téléphones portables puissent être collectés avant les réunions du conseil d' administration afin qu' ils ne soient pas utilisés comme appareils d' écoute à distance .

Ulrich Hackenberg , membre du conseil d' administration du constructeur automobile Audi , déclare que la collecte des téléphones portables avant les réunions du conseil , afin qu' ils ne puissent pas être utilisés comme appareils d' écoute à distance , est une pratique courante depuis des années .

Modeling sequential data

- Use the chain rule to express the joint distribution for a sequence of observations

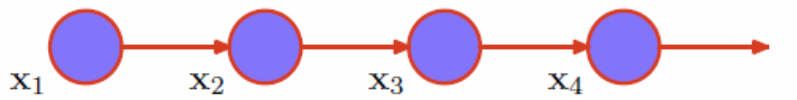
$$p(\mathbf{x}_1, \ldots, \mathbf{x}_T) = \prod_{t=1}^{T} p(\mathbf{x}_t | \mathbf{x}_1, \ldots, \mathbf{x}_{t-1})$$

- Impractical to consider general dependence of future dependence on all previous observations $p(\mathbf{x}_t | \mathbf{x}_{t-1}, \ldots, \mathbf{x}_0)$
  - Complexity would grow without limit as the number of observations increases
- It is expected that recent observations are more informative than more historical observations in predicting future values

Markov models

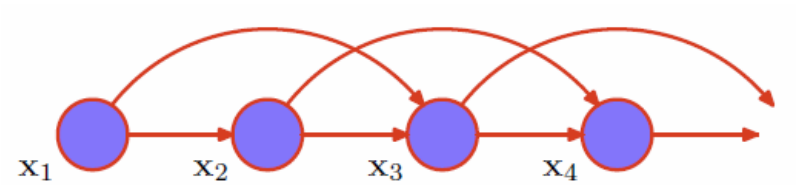
- Markov models assume dependence on most recent observations
- First-order Markov model

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_T) = \prod_{t=1}^{T} p(\mathbf{x}_t | \mathbf{x}_{t-1})$$



- Second-order Markov model

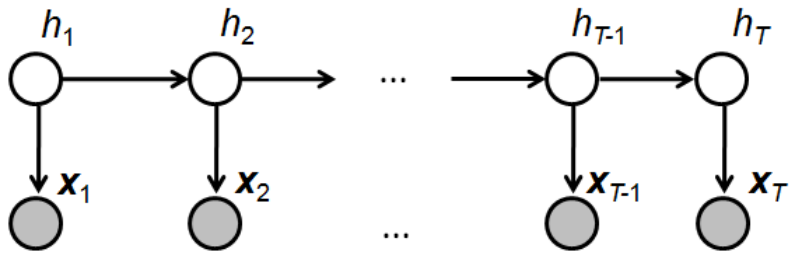
$$p(\mathbf{x}_1, \ldots, \mathbf{x}_T) = \prod_{t=1}^{T} p(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{x}_{t-2})$$



Hidden Markov Model (HMM)

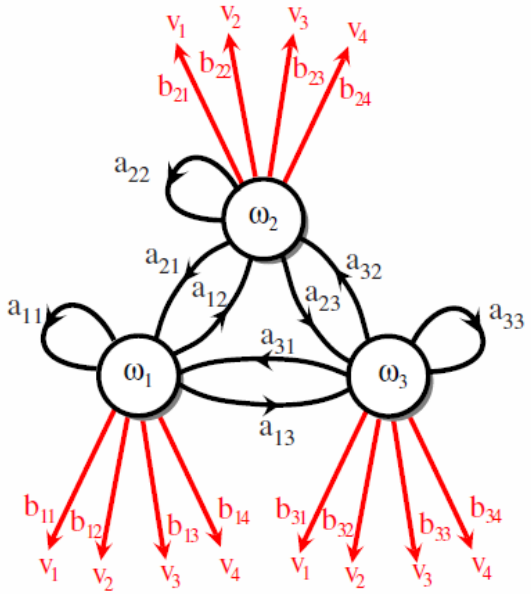
- A classical way to model sequential data
- Sequence pairs $h_1, h_2, \ldots, h_T$ (hidden variables) and $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T$ (observations) are generated by the following process
  - Pick h_1 at random from the distribution $P(h_1)$. Pick $\mathbf{x}_1$ from the distribution $p(\mathbf{x}_1|h_1)$
  - For $t = 2$ to T
    - Choose h_t at random from the distribution $p(h_t|h_{t-1})$
    - Choose $\mathbf{x}_t$ at random from the distribution $p(\mathbf{x}_t|h_t)$
- The joint distribution is

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_T, h_1, \ldots, h_T, \boldsymbol{\theta}) = P(h_1) \prod_{t=2}^{T} P(h_t|h_{t-1}) \prod_{t=1}^{T} p(\mathbf{x}_t|h_t)$$



Parameters of HMM

- Initial state parameters $P(h_1 = \omega_i) = \pi_i$, $\sum_i \pi_i = 1$
- Transition probabilities $P(h_t = \omega_j | h_{t-1} = \omega_i) = a_{ij}$, $\sum_j a_{ij} = 1$. $\mathbf{A} = [a_{ij}]$ is transition matrix
- Emission probabilities $p(\mathbf{x}_t | h_t)$
  - If $\mathbf{x}_t$ is a single discrete variable, $\mathbf{x}_t = x_t \in \{v_1, \ldots, v_m\}$, $P(x_t = v_k | z_t = \omega_j) = b_{jk}$, $\sum_k b_{jk} = 1$



Parameters of HMM

- **Evaluation problem**: determine the probability that a particular data sequence $\mathbf{x}_1, \ldots, \mathbf{x}_T$ was generated by that model,

$$P(\mathbf{x}_1, \ldots, \mathbf{x}_T)$$

- **Decoding problem**: given a set of observations $\mathbf{x}_1, \ldots, \mathbf{x}_T$, determine the most likely sequence of hidden states $h_1, \ldots, h_T$ that lead to the observations

$$(h_1^*, \ldots, h_T^*) = \text{argmax}_{(h_1, \ldots, h_T)} P(h_1, \ldots, h_T | \mathbf{x}_1, \ldots, \mathbf{x}_T)$$

- **Learning problem**: given a set of training observations of visible variables, without knowing the hidden variables, determine the parameters of HMM.

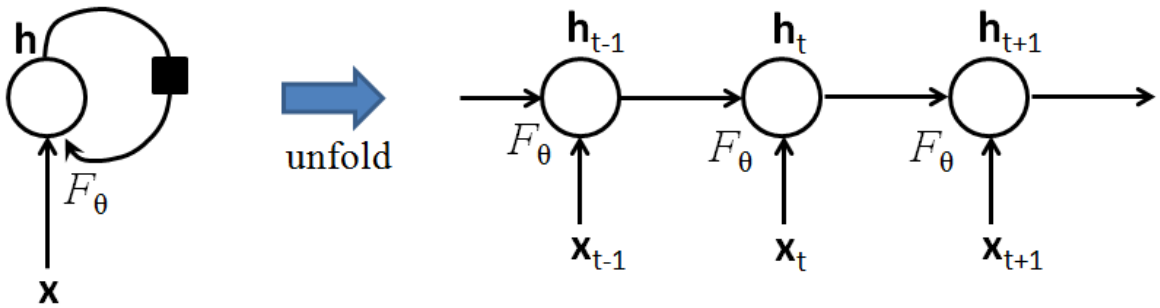
$$\theta^* = P(\mathbf{x}_1, \ldots, \mathbf{x}_T; \theta)$$

Recurrent neural networks (RNN)

- While HMM is a generative model RNN is a discriminative model
- Model a dynamic system driven by an external signal $\mathbf{x}_t$

$$\mathbf{h}_t = F_\theta(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

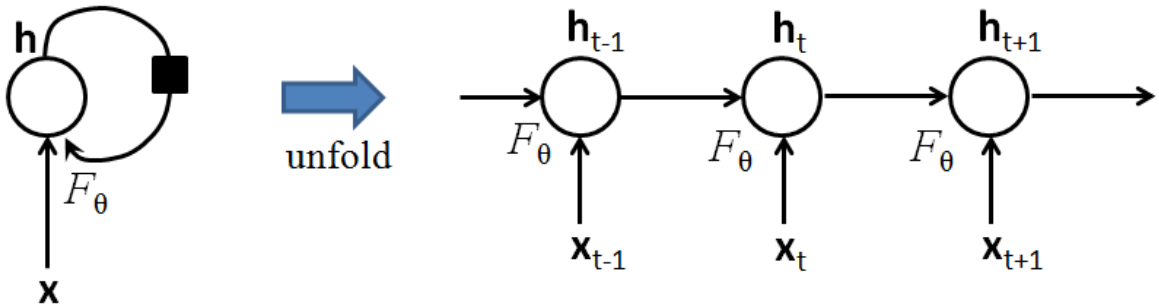
- $\mathbf{h}_t$ contains information about the whole past sequence. The equation above implicitly defines a function which maps the whole past sequence $(\mathbf{x}_t, \ldots, \mathbf{x}_1)$ to the current sate $\mathbf{h}_t = G_t(\mathbf{x}_t, \ldots, \mathbf{x}_1)$



Left: physical implementation of RNN, seen as a circuit. The black square indicates a delay of 1 time step. Right: the same seen as an unfolded flow graph, where each node is now associated with one particular time instance.

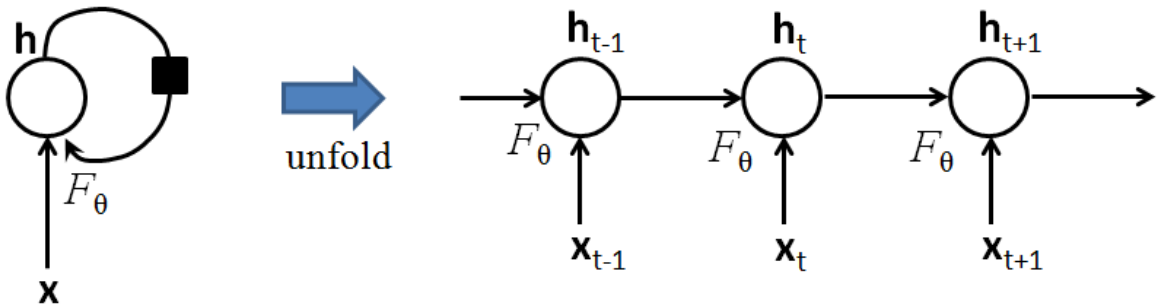
Recurrent neural networks (RNN)

- The summary is lossy, since it maps an arbitrary length sequence $(\mathbf{x}_t, \ldots, \mathbf{x}_1)$ to a fixed length vector $\mathbf{h}_t$. Depending on the training criterion, $\mathbf{h}_t$ keeps some important aspects of the past sequence.
- Sharing parameters: the same weights are used for different instances of the artificial neurons at different time steps
- Share a similar idea with CNN: replacing a fully connected network with local connections with parameter sharing
- It allows to apply the network to input sequences of different lengths and predict sequences of different lengths



Recurrent neural networks (RNN)

- **Sharing parameters for any sequence length allows more better generalization properties.** If we have to define a different function G_t for each possible sequence length, each with its own parameters, we would not get any generalization to sequences of a size not seen in the training set. One would need to see a lot more training examples, because a separate model would have to be trained for each sequence length.



A vanilla RNN to predict sequences from input

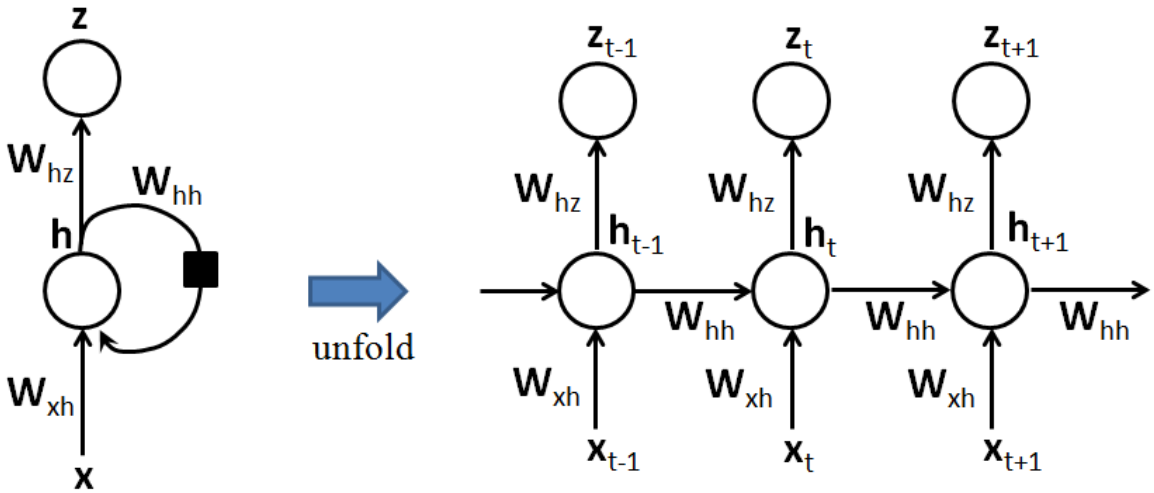
$$P(y_1, \ldots, y_T | \mathbf{x}_1, \ldots, \mathbf{x}_T)$$

- Forward propagation equations, assuming that hyperbolic tangent non-linearities are used in the hidden units and softmax is used in output for classification problems

$$\mathbf{h}_t = \tanh(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h)$$

$$\mathbf{z}_t = \text{softmax}(\mathbf{W}_{hz}\mathbf{h}_t + \mathbf{b}_z)$$

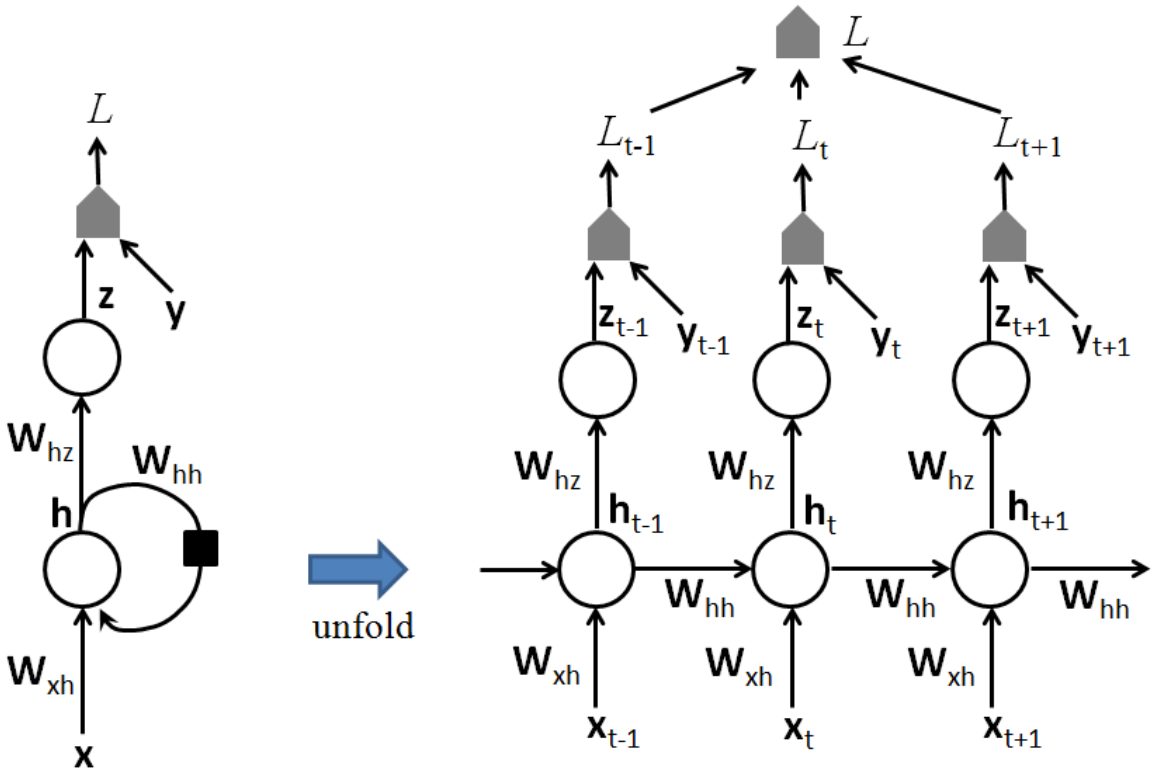
$$p(y_t = c) = z_{t,c}$$



Cost function

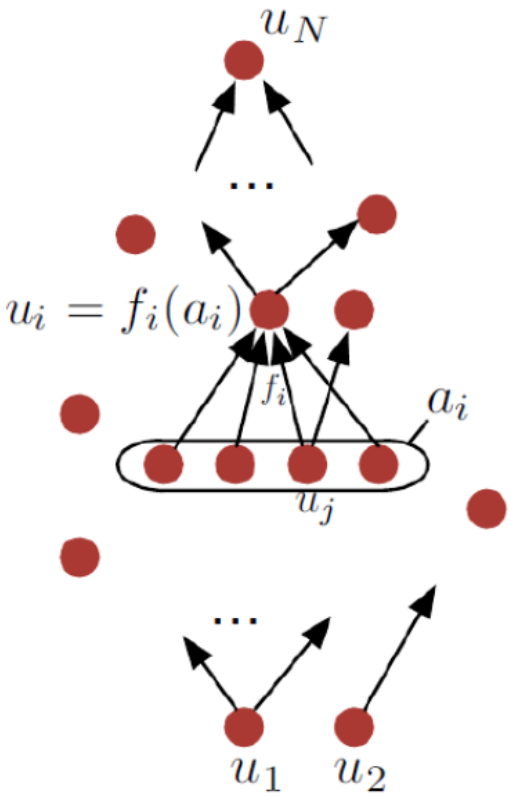
- The total loss for a given input/target sequence pair (**x**, **y**), measured in cross entropy

$$L(\mathbf{x}, \mathbf{y}) = \sum_t L_t = \sum_t -\log z_{t,y_t}$$



Backpropagation on RNN

- Review BP on flow graph



$\frac{\partial u_N}{\partial u_N} \leftarrow 1$
for $j = N-1$ down to 1 **do**
  $\frac{\partial u_N}{\partial u_j} \leftarrow \sum_{i:j\in \text{parents}(i)} \frac{\partial u_N}{\partial u_i}\frac{\partial u_i}{\partial u_j}$
end for
return $\left(\frac{\partial u_N}{\partial u_i}\right)_{i=1}^{M}$

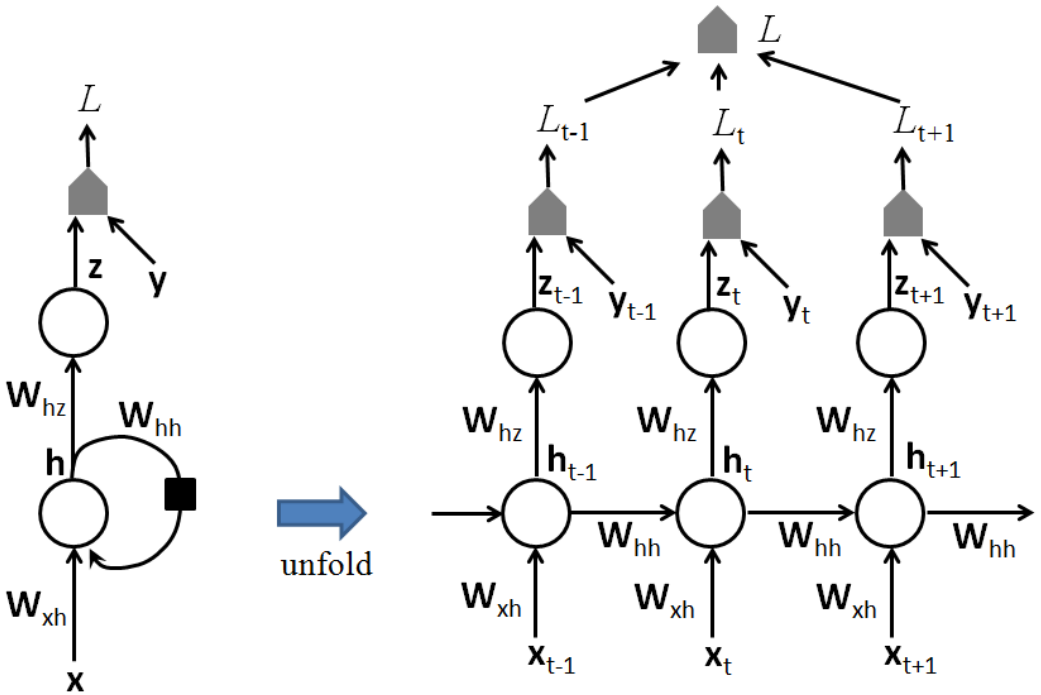
(Bengio et al. Deep Learning 2014)

$$\frac{\partial u_N}{\partial w_{ji}} = \frac{\partial u_N}{\partial u_i}\frac{\partial u_i}{\partial net_i}\frac{\partial net_i}{\partial w_{ji}}$$

Gradients on $\mathbf{W}_{hz}$ and $\mathbf{b}_z$

$$\frac{\partial L}{\partial L_t} = 1, \qquad \frac{\partial L}{\partial \mathbf{z}_t} = \frac{\partial L}{\partial L_t}\frac{\partial L_t}{\partial \mathbf{z}_t} = \frac{\partial L_t}{\partial \mathbf{z}_t}$$

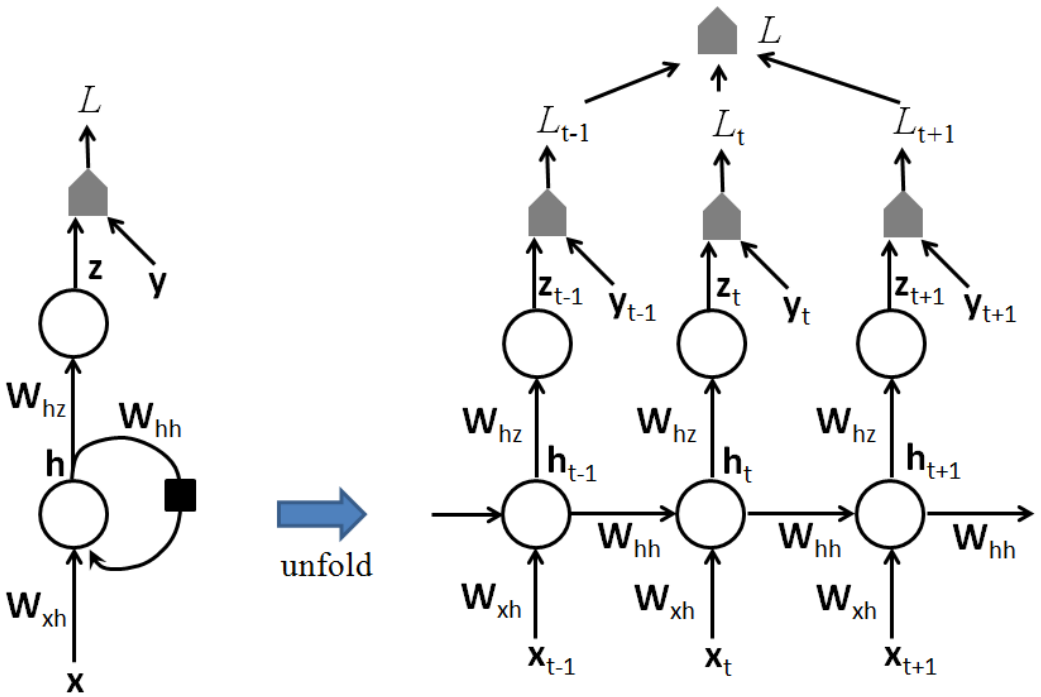
$$\frac{\partial L}{\partial \mathbf{W}_{hz}} = \sum_t \frac{\partial L_t}{\partial \mathbf{z}_t}\frac{\partial \mathbf{z}_t}{\partial \mathbf{W}_{hz}}, \qquad \frac{\partial L}{\partial \mathbf{b}_z} = \sum_t \frac{\partial L_t}{\partial \mathbf{z}_t}\frac{\partial \mathbf{z}_t}{\partial \mathbf{b}_z}$$



Gradients on $\mathbf{W}_{hh}$ and $\mathbf{W}_{xh}$

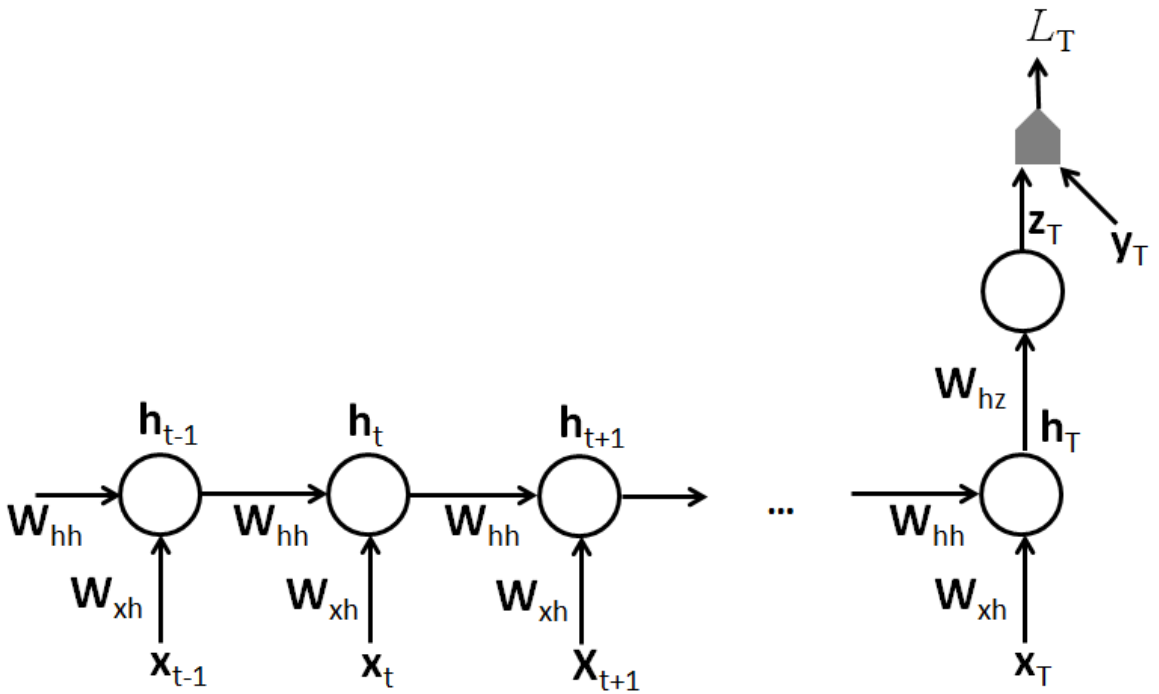
$$\frac{\partial L}{\partial \mathbf{W}_{hh}} = \sum_t \frac{\partial L}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_{hh}}$$

$$\frac{\partial L}{\partial \mathbf{h}_t} = \frac{\partial L}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t} + \frac{\partial L}{\partial \mathbf{z}_t} \frac{\partial \mathbf{z}_t}{\partial \mathbf{h}_t}$$



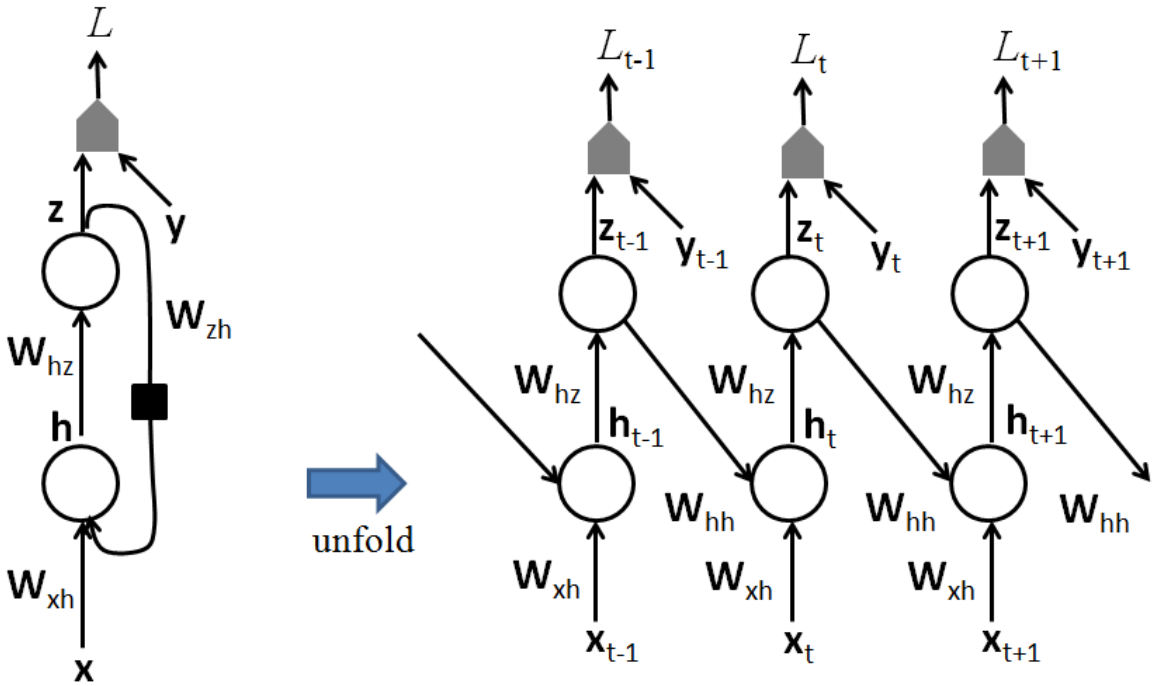
Predict a single output at the end of the sequence

- Such a network can be used to summarize a sequence and produce a fixed-size representation used as input for further processing. There might be a target right at the end or the gradient on the output $\mathbf{z}_t$ can be obtained by backpropagation from further downsteam modules



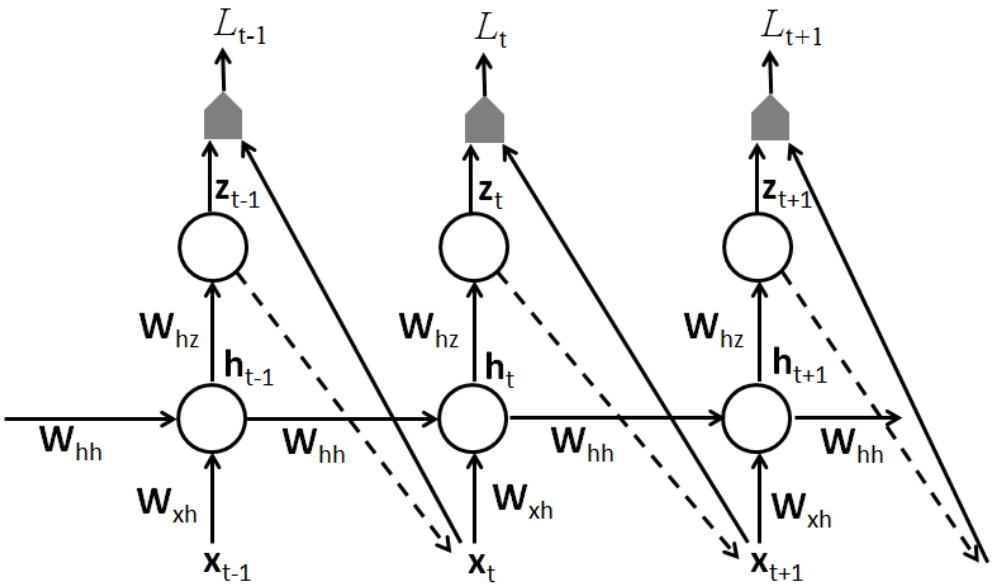
Network with output recurrence

- Memory is from the prediction of the previous target, which limits its expressive power but makes t easier to train



Generative RNN modeling $P(\mathbf{x}_1, \ldots, \mathbf{x}_T)$

- It can generate sequences from this distribution
- At the training stage, each $\mathbf{x}_t$ of the observed sequence serves both as input (for the current time step) and as target (for the previous time step)
- The output $\mathbf{z}_t$ encodes the parameters of a conditional distribution $P(\mathbf{x}_{t+1}|\mathbf{x}_1, \ldots, \mathbf{x}_t) = P(\mathbf{x}_{t+1}|\mathbf{z}_t)$ for $\mathbf{x}_{t+1}$ given the past sequence $\mathbf{x}_1, \ldots, \mathbf{x}_t$



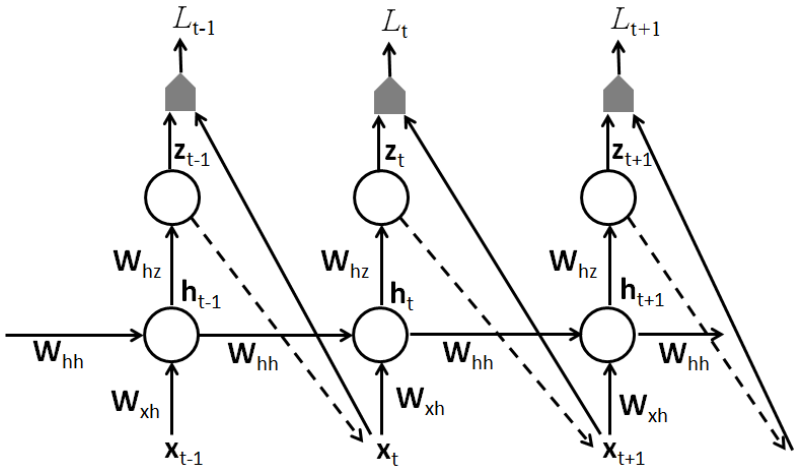
Generative RNN modeling $P(\mathbf{x}_1, \ldots, \mathbf{x}_T)$

- Cost function: negative log-likelihood of $\mathbf{x}$, $L = \sum_t L_t$

$$P(\mathbf{x}) = P(\mathbf{x}_1, \ldots, \mathbf{x}_T) = \prod_{t=1}^{T} P(\mathbf{x}_t | \mathbf{x}_{t-1}, \ldots, \mathbf{x}_1)$$

$$L_t = -\log P(\mathbf{x}_t | \mathbf{x}_{t-1}, \ldots, \mathbf{x}_1)$$

- In generative mode, $\mathbf{x}_{t+1}$ is sampled from the conditional distribution $P(\mathbf{x}_{t+1} | \mathbf{x}_1, \ldots, \mathbf{x}_t) = P(\mathbf{x}_{t+1} | \mathbf{z}_t)$ (dashed arrows) and then that generated sample $\mathbf{x}_{t+1}$ is fed back as input for computing the next state $\mathbf{h}_{t+1}$



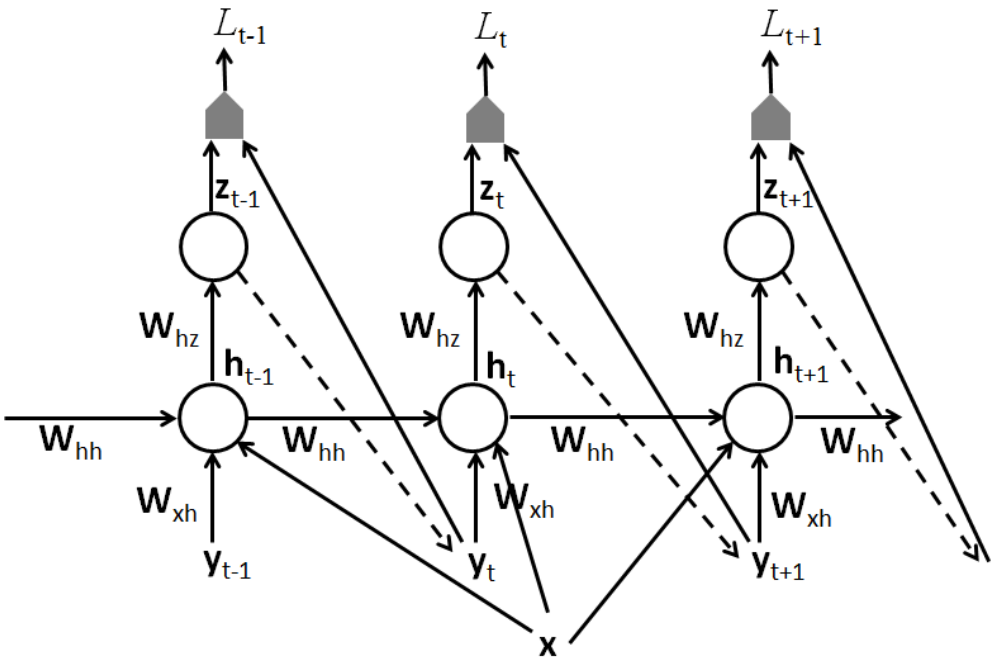
Generative RNN modeling $P(\mathbf{x}_1, \ldots, \mathbf{x}_T)$

- If RNN is used to generate sequences, one must also incorporate in the output information allowing to stochastically decide when to stop generating new output elements
- In the case when the output is a symbol taken from a vocabulary, one can add a special symbol corresponding to the end of a sequence
- One could also directly directly model the length T of the sequence through some parametric distribution. $P(\mathbf{x}_1, \ldots, \mathbf{x}_T)$ is decomposed into

$$P(\mathbf{x}_1, \ldots, \mathbf{x}_T) = P(\mathbf{x}_1, \ldots, \mathbf{x}_T | T)P(T)$$

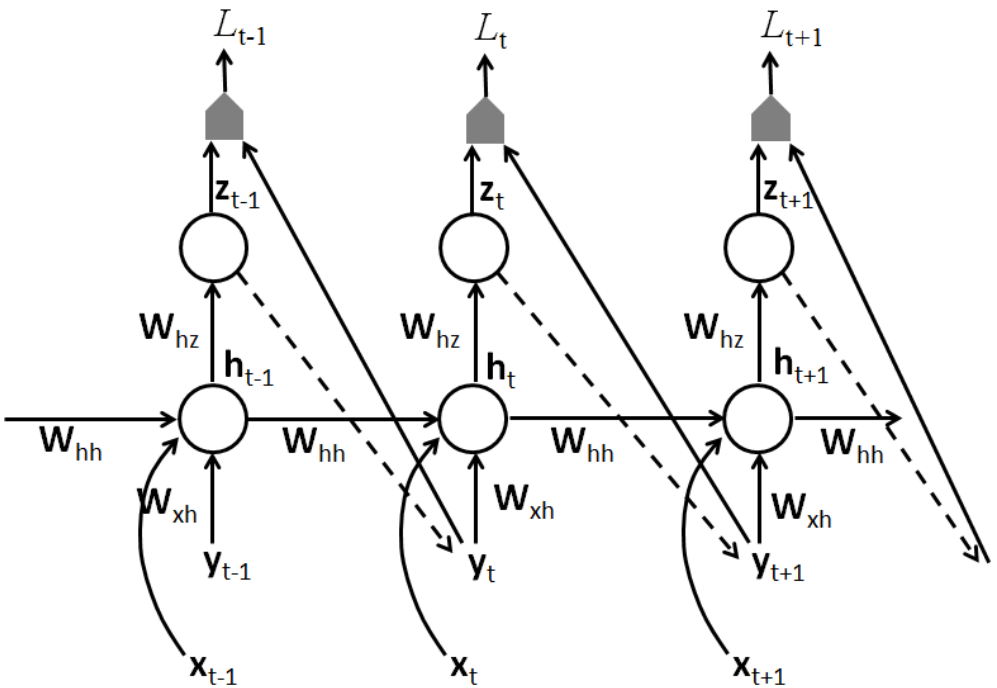
RCNNs to represent conditional distributions $P(\mathbf{y}|\mathbf{x})$

- If $\mathbf{x}$ is a fixed-sized vector, we can simply make it an extra input of the RNN that generates the $\mathbf{y}$ sequence. Some common ways of providing the extra input
  - as an extra input at each time step, or
  - as the initial state $\mathbf{h}_0$, or
  - both
- Example: generate caption for an image



RCNNs to represent conditional distributions $P(\mathbf{y}|\mathbf{x})$

- The input $\mathbf{x}$ is a sequence of the same length as the output sequence $\mathbf{y}$
- Removing the dash lines, it assumes $\mathbf{y}_t$'s are independent of each other when the past input sequence is given, i.e.
  $P(\mathbf{y}_t|\mathbf{y}_{t-1}, \ldots, \mathbf{y}_1, \mathbf{x}_t, \ldots, \mathbf{x}_1) = P(\mathbf{y}_t|\mathbf{x}_t, \ldots, \mathbf{x}_1)$
- Without the conditional independence assumption, add the dash lines and the prediction of $\mathbf{y}_{t+1}$ is based on both the past $\mathbf{x}$'s and past $\mathbf{y}$'s

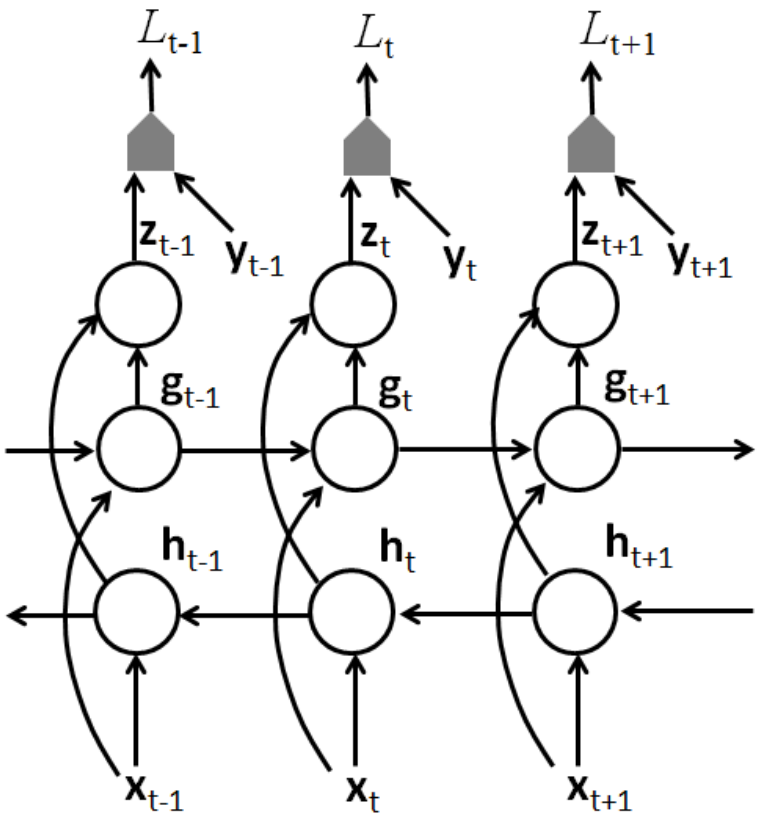


Bidirectional RNNs

- In some applications, we want to output at time t a prediction regarding an output which may depend on the whole input sequence
  - In speech recognition, the correct interpretation of the current sound as a phoneme may depend on the next few phonemes because co-articulation and may depend on the next few words because of the linguistic dependencies between words
- Bidirectional recurrent neural network was proposed to address such need
- It combines a forward-going RNN and a backward-going RNN
- The idea can be extended to 2D input with four RNN going in four directions

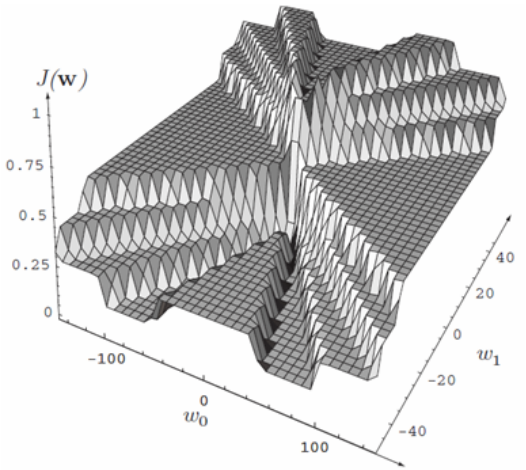
Bidirectional RNNs

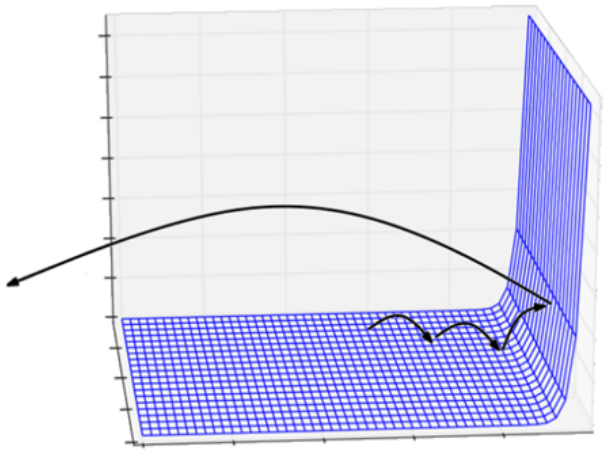
- $\mathbf{g}_t$ summaries the information from the past sequence, and $\mathbf{h}_t$ summaries the information from the future sequence



Plateaus and cliffs

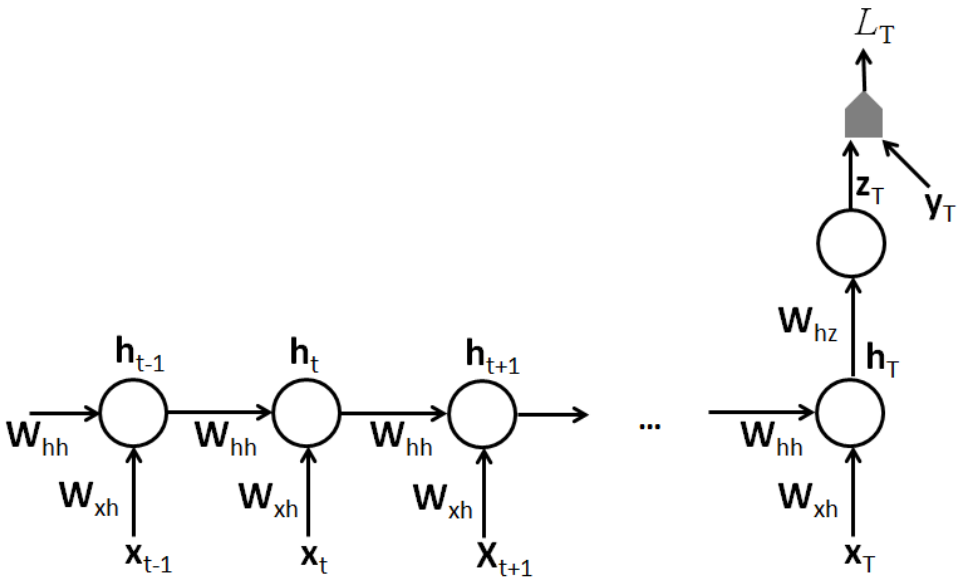
- The error surfaces of training deep neural networks include local minima, plateaus (regions where error varies only slightly as a function of weights), and cliffs (regions where the gradients rise sharply)
- Plateaus and cliffs are more important barriers to training neural networks than local minima
  - It is very difficult (or slow) to effectively update the parameters in plateaus
  - When the parameters approach a cliff region, the gradient update step can move the learner towards a very bad configuration, ruining much progress made during recent training iterations.





Vanishing and exploding gradients

- Training a very deep net makes the problem even more serious, since after BP through many layers, the gradients become either very small or very large
- RNN can be treated as a deep net when modeling long term dependency



Vanishing and exploding gradients

- In very deep nets and recurrent nets, the final output is composed of a large number of non-linear transformations
- Even though each of these non-linear stages may be relatively smooth, their composition is going to be much "more non-linear", in the sense that the derivatives through the whole composition will tend to be either very small or very large, with more ups and downs

When composing many non-linearities (like the activation non-linearity in a deep or recurrent neural network), the result is highly non-linear, typically with most of the values associated with a tiny derivative, some values with a large derivative, and many ups and downs (not shown here)

Vanishing and exploding gradients

This arises because the Jacobian (matrix of derivatives) of a composition is the product of the Jacobian of each stage, i.e. if

$$f = f_T \circ f_{T-1} \circ \dots f_2 \circ f_1$$

The Jacobian matrix of derivatives of $f(x)$ with respect to its input vector $\mathbf{x}$ is

$$f' = f'_T f'_{T-1} \dots f'_2 f'_1$$

where

$$f' = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}$$

and

$$f'_t = \frac{\partial f_t(\boldsymbol{\alpha}_t)}{\partial \boldsymbol{\alpha}_t}$$

where $\boldsymbol{\alpha}_t = f_{t-1}(f_{t-1}(\dots f_2(f_1(\mathbf{x}))))$, i.e. composition has been replaced by matrix multiplication

Vanishing and exploding gradients

- In the scalar case, we can imagine that multiplying many numbers together tends to be either very large or very small
- In the special case where all the numbers in the product have the same value α, this is obvious, since α^T goes to 0 if $\alpha < 1$ and to ∞ if $\alpha > 1$ as T increases
- The more general case of non-identical numbers be understood by taking the logarithm of these numbers, considering them to be random, and computing the variance of the sum of these logarithms. Although some cancellation can happen, the variance grows with T. If those numbers are independent, it grows linearly with T, which means that the product grows roughly as e^T.
- This analysis can be generalized to the case of multiplying square matrices

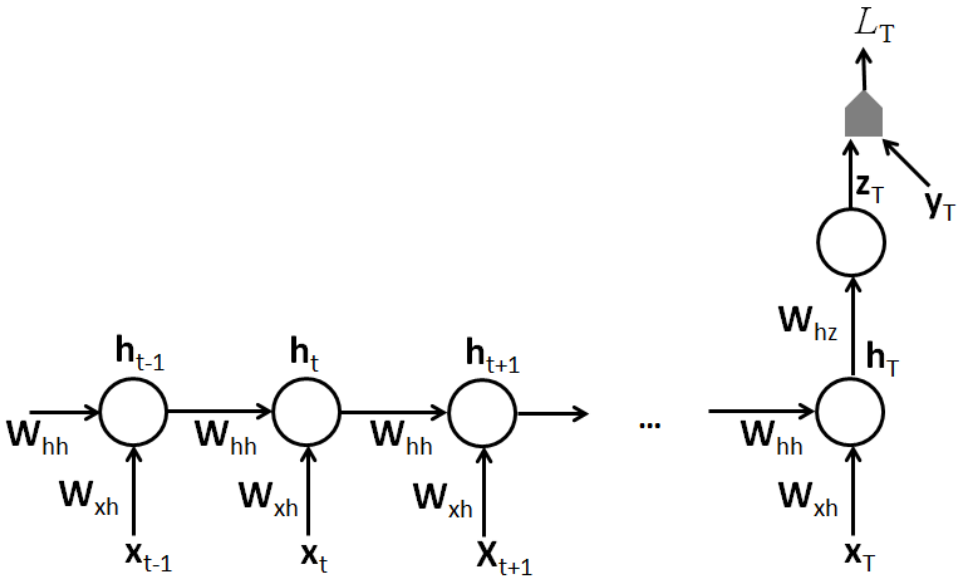
Difficulty of Learning Long-Term Dependencies

- Consider the gradient of a loss L_T at time T with respect to the parameter θ of the recurrent function F_θ

$$\mathbf{h}_t = F_\theta(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

$$\frac{\partial L_T}{\partial \theta} = \sum_{t \leq T} \frac{\partial L_T}{\partial \mathbf{h}_T} \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t} \frac{\partial F_\theta(\mathbf{h}_{t-1}, \mathbf{x}_t)}{\partial \theta}$$

$\frac{\partial L_T}{\partial \mathbf{h}_T} \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t} \frac{\partial F_\theta(\mathbf{h}_{t-1}, \mathbf{x}_t)}{\partial \theta}$ encodes long-term dependency when $T - t$ is large



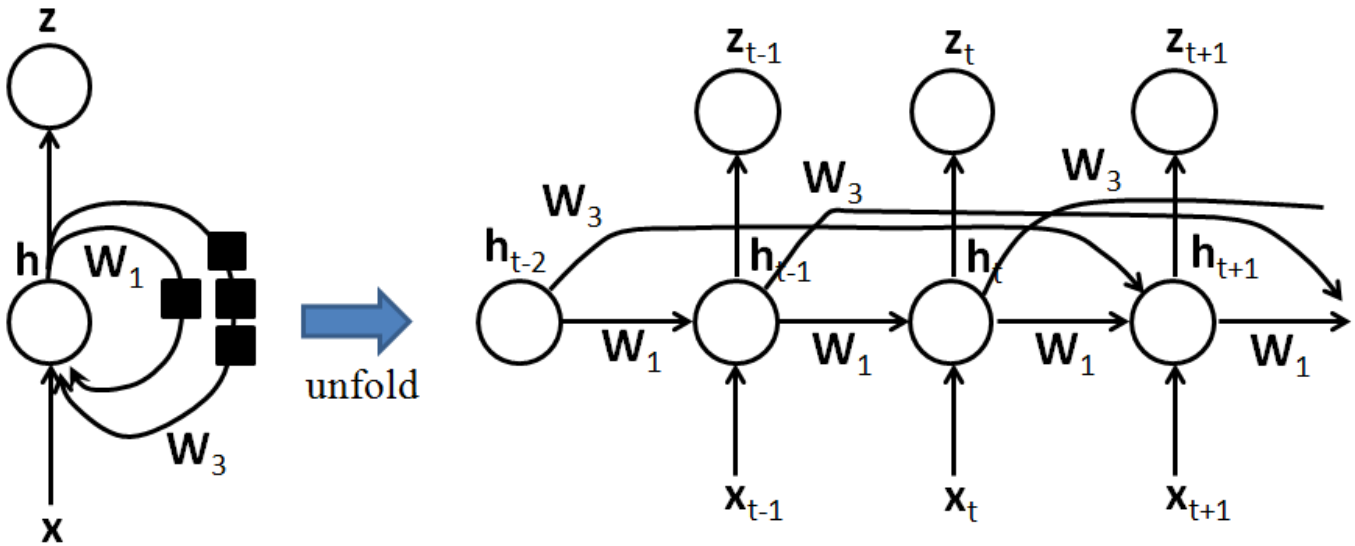
Difficulty of Learning Long-Term Dependencies

$$\frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t} = \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_{T-1}} \frac{\partial \mathbf{h}_{T-1}}{\partial \mathbf{h}_{T-2}} \cdots \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}$$

- Each layer-wise Jacobian $\frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}$ is the product of two matrices: (a) the recurrent matrix $\mathbf{W}$ and (b) the diagonal matrix whose entries are the derivatives of the non-linearities associated with the hidden units, which variy depending on the time step. This makes it likely that successive Jacobians have simliar eigenvectors, making the product of these Jacobians explode or vanish even faster
- $\frac{\partial L_T}{\partial \theta}$ is a weighted sum of terms over spans $T - t$, with weights that are exponentially smaller (or larger) for long-term dependencies relating the state at t to the state at T
- The signal about long term dependecies will tend to be hidden by the smallest fluctuations arising from short-term dependenties

Combine short and long paths in unfolded flow graph

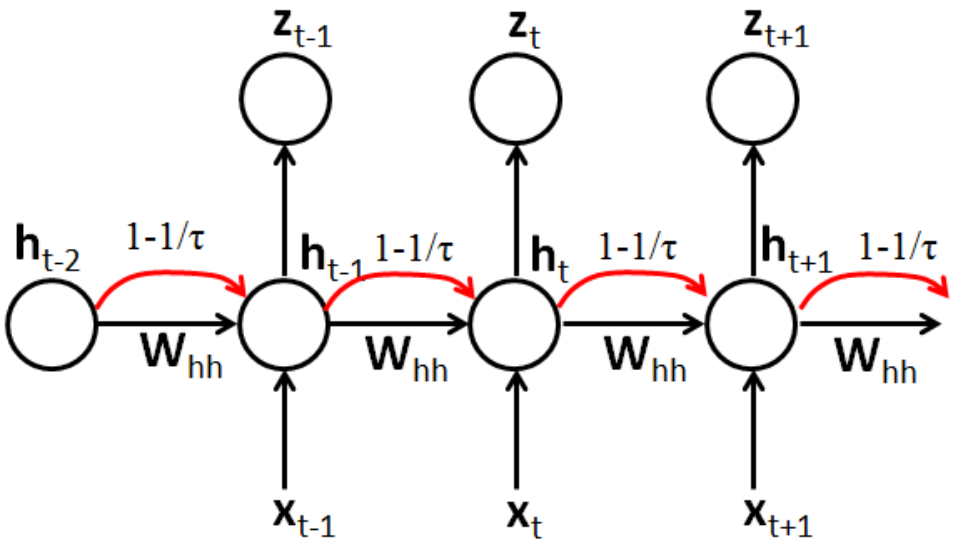
- Longer-delay connections allow to connect the past states to future states through short paths
- Gradients will vanish exponentially with respect to the number of time steps
- If we have recurrent connections with a time-delay of D, the instead of the vanishing or explosion going as $O(\lambda^T)$ over T steps (where λ is largest eigenvalue of the Jacobians $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$), the unfolded recurrent network now has paths through which gradients grow as $O(\lambda^{T/D})$ because the number of effective steps is T/D



Leaky units with self-connections

$$\mathbf{h}_{t+1} = (1 - \frac{1}{\tau_i})\mathbf{h}_t + \frac{1}{\tau_i}\tanh(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_t + \mathbf{b}_h)$$

- The new value of the state $\mathbf{h}_{t+1}$ is a combination of linear and non-linear parts of $\mathbf{h}_t$
- The errors are easier to be back propagated through the paths of red lines, which are linear



Leaky units with self-connections

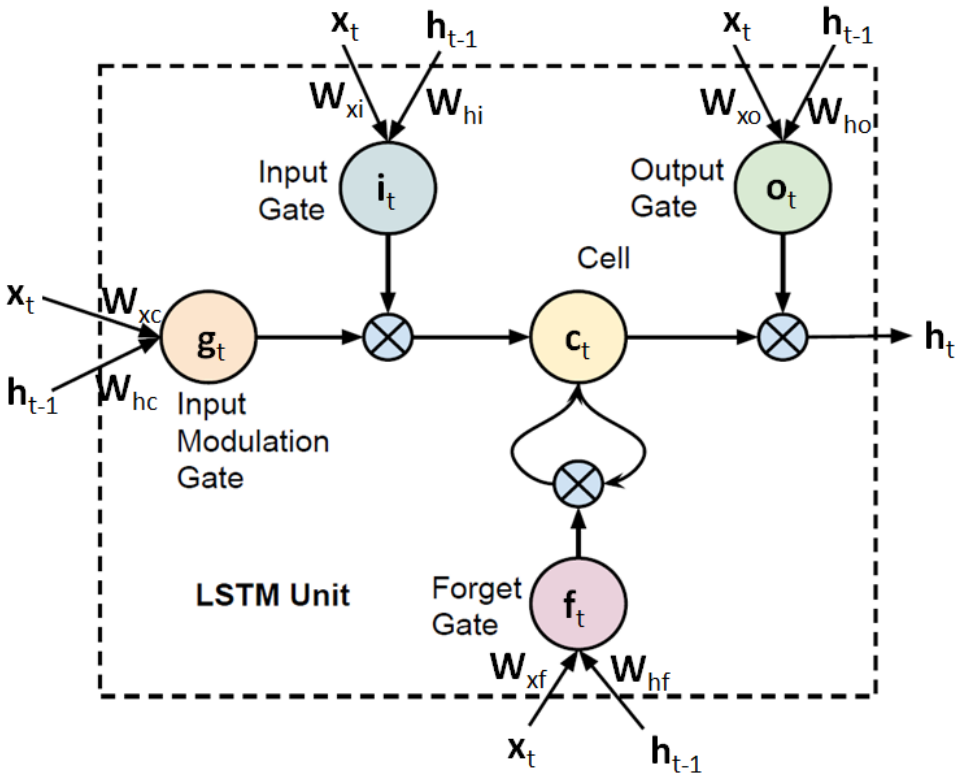
- When $\tau = 1$, there is no linear self-recurrence, only the nonlinear update which we can find in ordinary recurrent networks
- When $\tau > 1$, this linear recurrence allows gradients to propagate more easily. When τ is large, the sate changes very slowly, integrating the past values associated with the input sequence
- τ controls the rate of forgetting old states. It can be viewed as a smooth variant of the idea of the previous model
- By associating different time scales τ with different units, one obtains different paths corresponding to different forgetting rates
- Those time constants can be fixed manually or can be learned as free parameters

Long Short-Term Memory (LSTM) net

- In the leaky units with self-connections, the forgetting rate is constant during the whole sequence.
- The role of leaky units is to accumulate information over a long duration. However, once that information gets used, it might be useful for the neural network to forget the old state.
  - For example, if a video sequence is composed as subsequences corresponding to different actions, we want a leaky unit to accumulate evidence inside each subsequnece, and we need a mechanism to forget the old state by setting it to zero and starting to count from fresh when starting to process the next subsequence
- The forgetting rates are expected to be different at different time steps, depending on their previous hidden sates and current input (conditioning the forgetting on the context)
- Parameters controlling the forgetting rates are learned from train data

Long Short-Term Memory (LSTM) net

$$\mathbf{f}_t = \sigma(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f),\ \mathbf{i}_t = \sigma(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{b}_i),$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o)$$
$$\mathbf{g}_t = \tanh(\mathbf{W}_{xc}\mathbf{x}_t + \mathbf{W}_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c),\ \mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t),\ \mathbf{z}_t = \text{softmax}(\mathbf{W}_{hz}\mathbf{h}_t + \mathbf{b}_z)$$

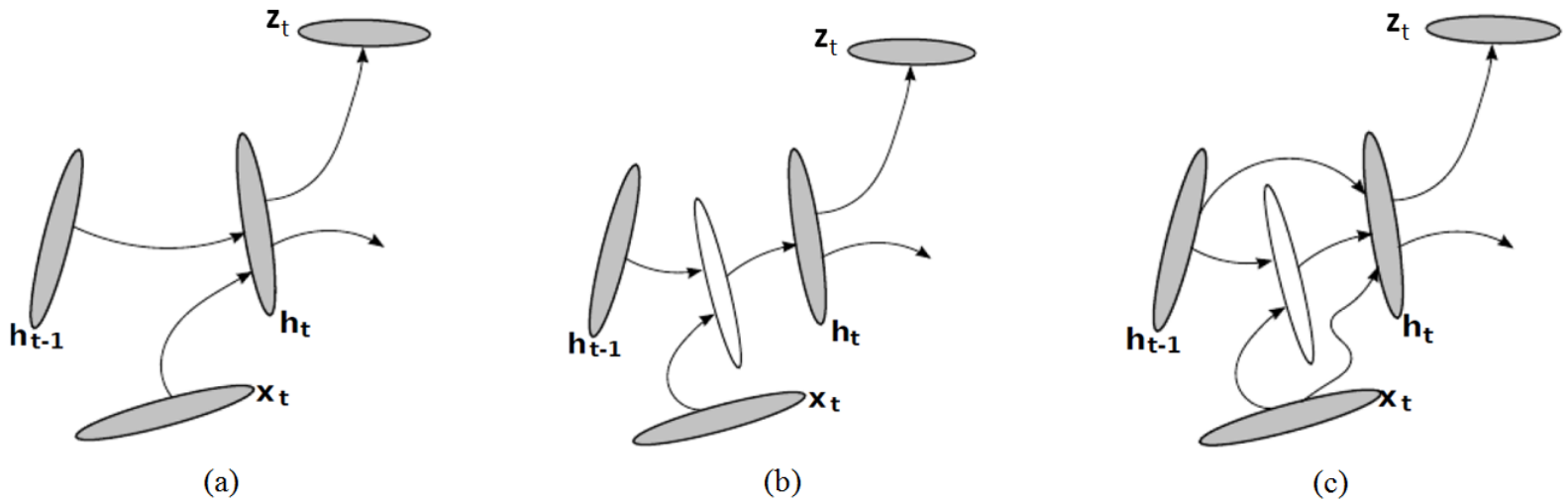


Long Short-Term Memory (LSTM) net

- The core of LSTM is a memory cell $\mathbf{c}_t$ which encodes, at every time step, the knowledge of the inputs that have been observed up to that step.
- The memory cell $\mathbf{c}_t$ has the same inputs ($\mathbf{h}_{t-1}$ and $\mathbf{x}_t$) and outputs ($\mathbf{h}_t$) as a normal recurrent network, but has more parameters and a system of gating units that controls the flow of information
- $\mathbf{c}_t$ has a linear self-connection similar to the leaky units, but the self-connection weight is controlled by a forget gate unit $\mathbf{f}_t$, that sets this weight to a value between 0 and 1 via a sigmoid unit
  $\mathbf{f}_t = \sigma(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f)$
- The input gate unit $\mathbf{i}_t$ is computed similarly to the forget gate, but with its own parameters
- The output $\mathbf{h}_t$ of the LSTM cell can also be shut off , via the output gate $\mathbf{o}_t$ ($\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$), which is also a sigmoid unit for gating
  $\mathbf{o}_t = \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o)$

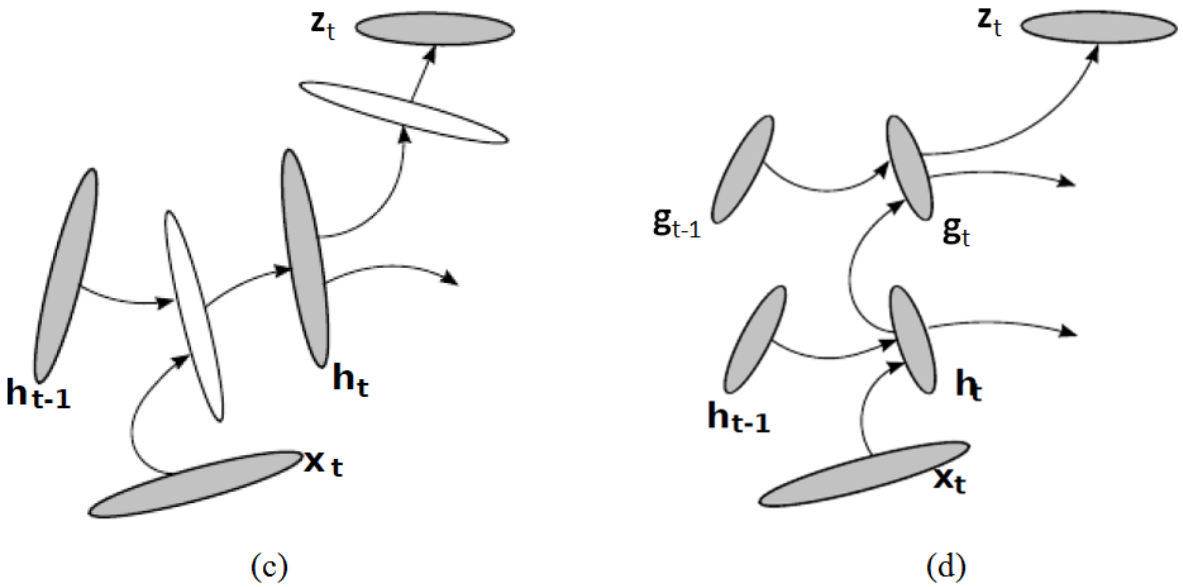
Long Short-Term Memory (LSTM) net

- (a): A vanilla RNN with input sequence and an output sequence
- (b): Add a deep hidden-to-hidden transformation
- (c): Skip connections and allow gradients to flow more easily backwards in spite of the extra non-linearity due to the intermediate hidden layer



Long Short-Term Memory (LSTM) net

- (c): Depth can also be added in the hidden-to-output transform
- (d): A hierarchy of RNNs, which can be stacked on top of each other



Applications

To be continued...

Reading Materials

- R. O. Duda, P. E. Hart, and D. G. Stork, “Pattern Classification,” Chapter 6, 2000.
- Y. Bengio, I. J. Goodfellow and A. Courville, “Sequence Modeling: Recurrent and Recursive Nets” in “Deep Learning”, Book in preparation for MIT Press, 2014.