

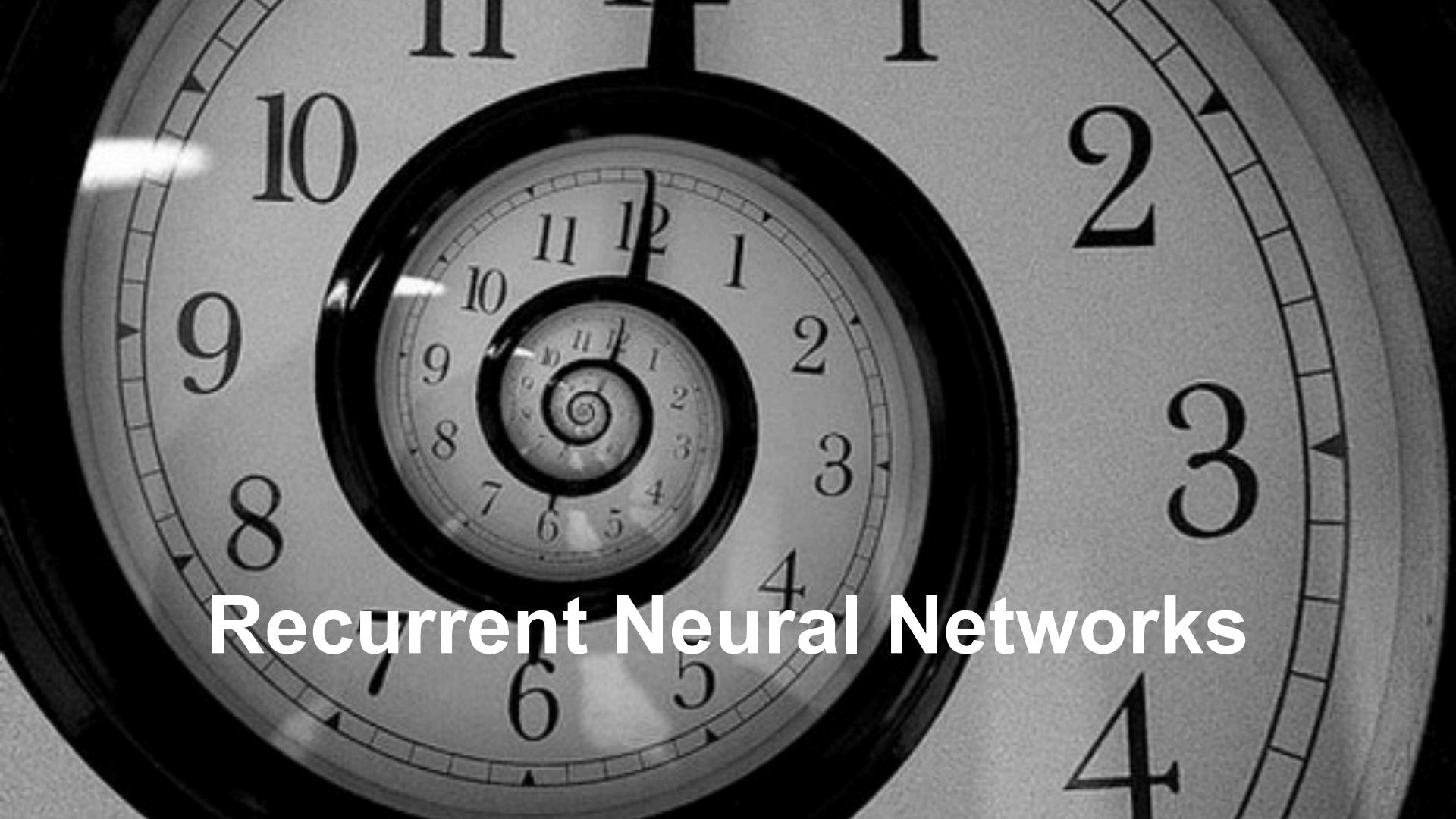
# Introduction to Deep Learning

## 6. Recurrent Networks

Haibin Lin and Leonard Lausen

[gluon-nlp.mxnet.io](http://gluon-nlp.mxnet.io)

8:30-9:00	Continental Breakfast
9:00-9:45	Introduction and Setup
9:45-10:30	Neural Networks 101
10:30-10:45	Break
10:45-11:15	Machine Learning Basics
11:15-11:45	Context-free Representations for Language
11:45-12:15	Convolutional Neural Networks
12:15-13:15	Lunch Break
13:15-14:00	Recurrent Neural Networks
14:00-14:45	Attention Mechanism and Transformer
14:45-15:00	Coffee Break
15:00-16:15	Contextual Representations for Language
16:15-17:00	Language Generation

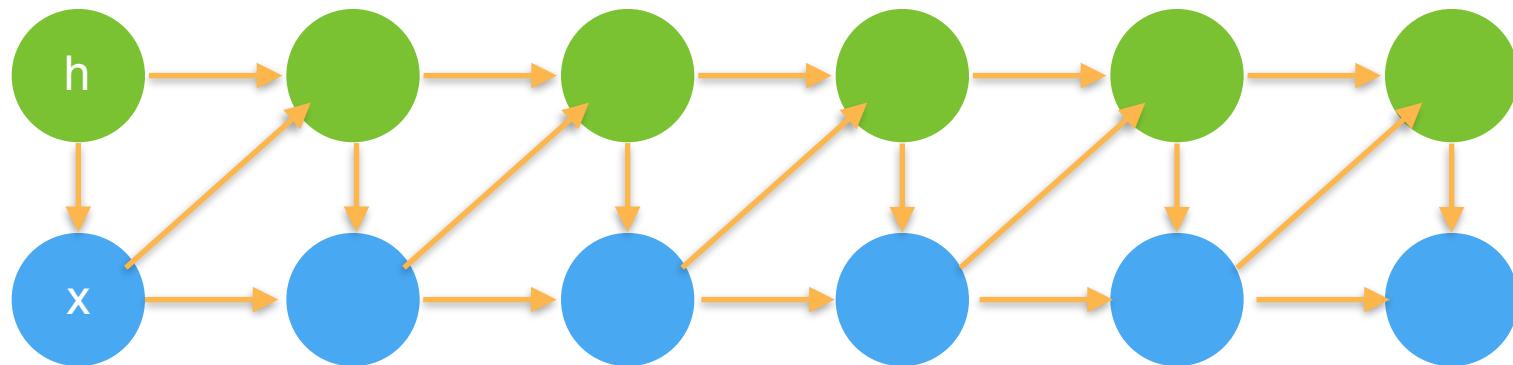


# Recurrent Neural Networks

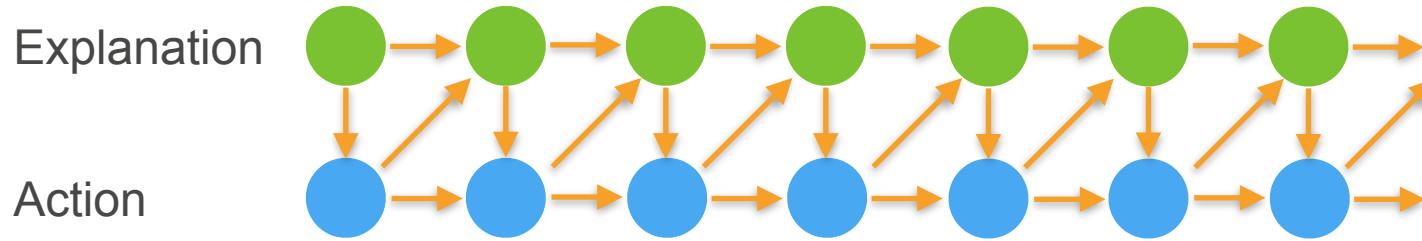
# Latent Variable Autoregressive Models

Latent state summarizes all the relevant information about the past. So we get  $h_t = f(x_1, \dots, x_{t-1}) = f(h_{t-1}, x_{t-1})$

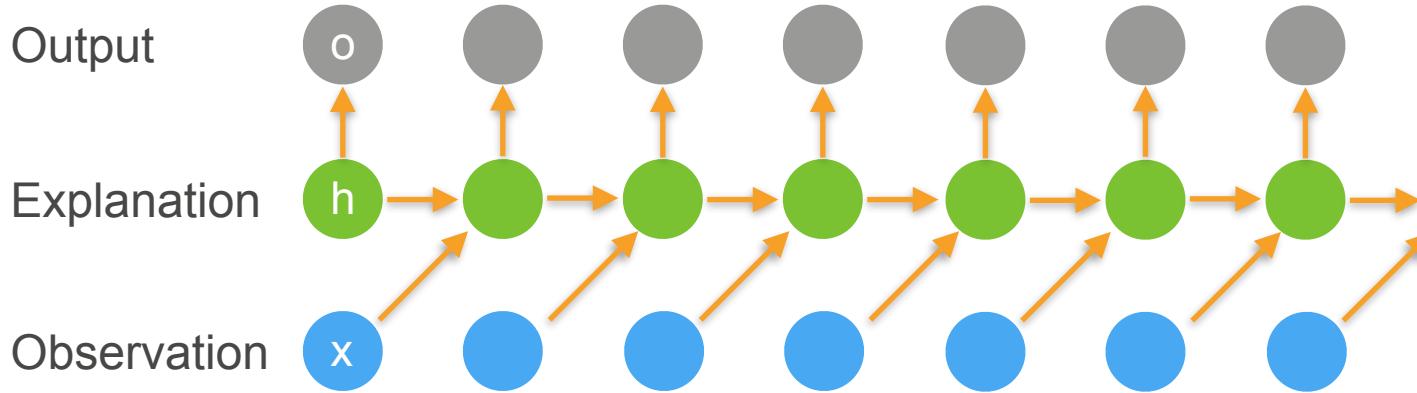
$p(h_t | h_{t-1}, x_{t-1})$  and  $p(x_t | h_t, x_{t-1})$



# Recurrent Neural Networks (with hidden state)



# Recurrent Neural Networks (with hidden state)



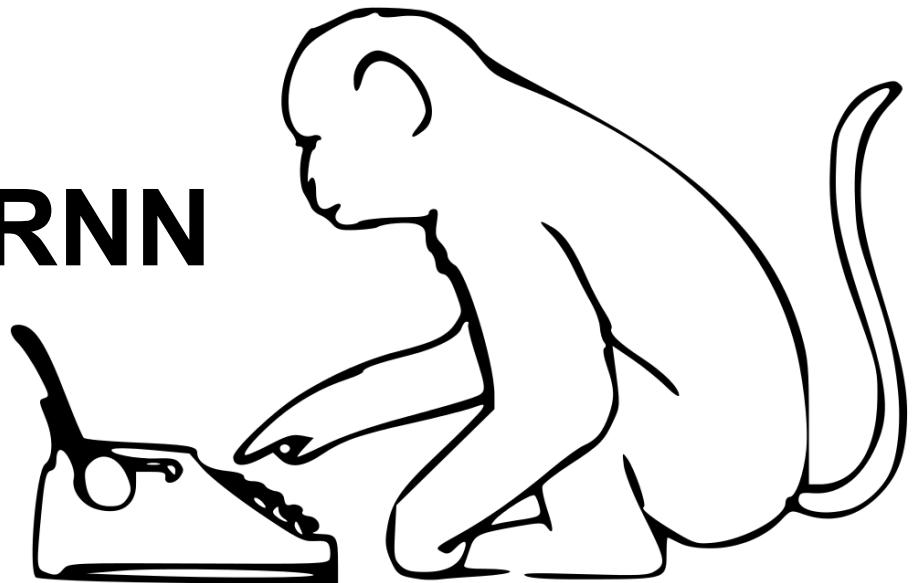
- Hidden State update

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

- Observation update

$$\mathbf{o}_t = \phi(\mathbf{W}_{ho}\mathbf{h}_t + \mathbf{b}_o)$$

# Implementing an RNN Language Model



# Input Encoding

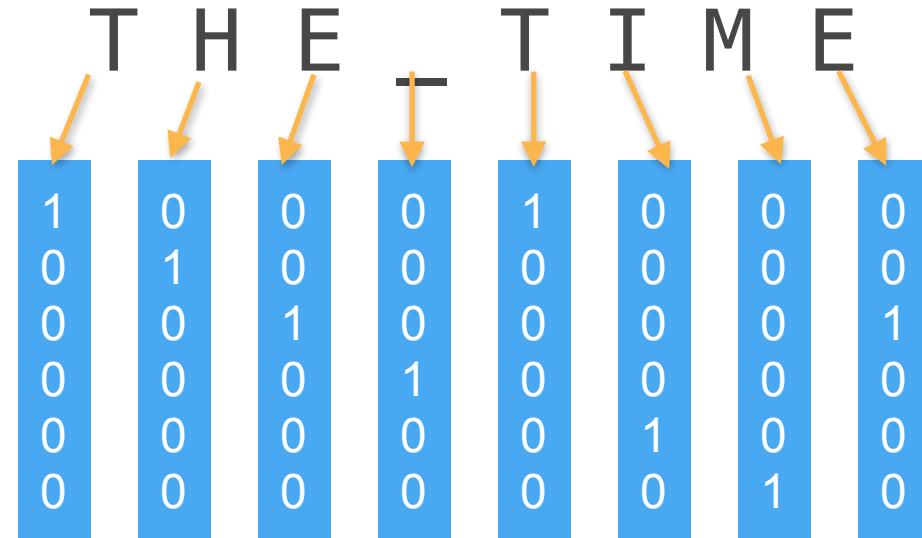
- Need to map input tokens to vectors
    - Pick granularity (words, characters, subwords)
    - Map to indicator vectors

```
nd.one_hot(nd.array([0, 2]), vocab_size)
```

- Multiply by embedding matrix  $\mathbf{W}$

# Input Encoding

Canonical Vectors  $v$



Embedding Matrix  $W$



Embedded Vectors  $v'$



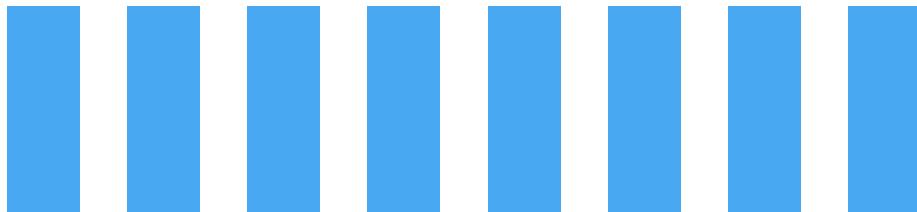
# RNN with hidden state mechanics

- Input  
vector sequence  $\mathbf{x}_1, \dots, \mathbf{x}_T$
- Hidden States  
vector sequence  $\mathbf{h}_1, \dots, \mathbf{h}_T$  where  $\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$
- Output  
vector sequence  $\mathbf{o}_1, \dots, \mathbf{o}_T$  where  $\mathbf{o}_t = g(\mathbf{h}_t)$

Read sequence to generate hidden states, then start generating outputs. Often outputs (symbols) are used as input for next hidden state (and thus output).

# Output Decoding

Output Vectors  $\mathbf{o}$



Decoding Matrix  $\mathbf{W}'$



$$p(y | \mathbf{o}) \propto \exp \left( \mathbf{v}_y^\top \mathbf{o} \right) = \exp(\mathbf{o}[y])$$

One-hot decoding



# Gradients

- Long chain of dependencies for backprop
  - Need to keep a lot of intermediate values in memory
  - Butterfly effect style dependencies
  - Gradients can vanish or diverge (more on this later)
- Clipping to prevent divergence

$$\mathbf{g} \leftarrow \min \left( 1, \frac{\theta}{\|\mathbf{g}\|} \right) \mathbf{g}$$

rescales to gradient of size at most  $\theta$

# Perplexity

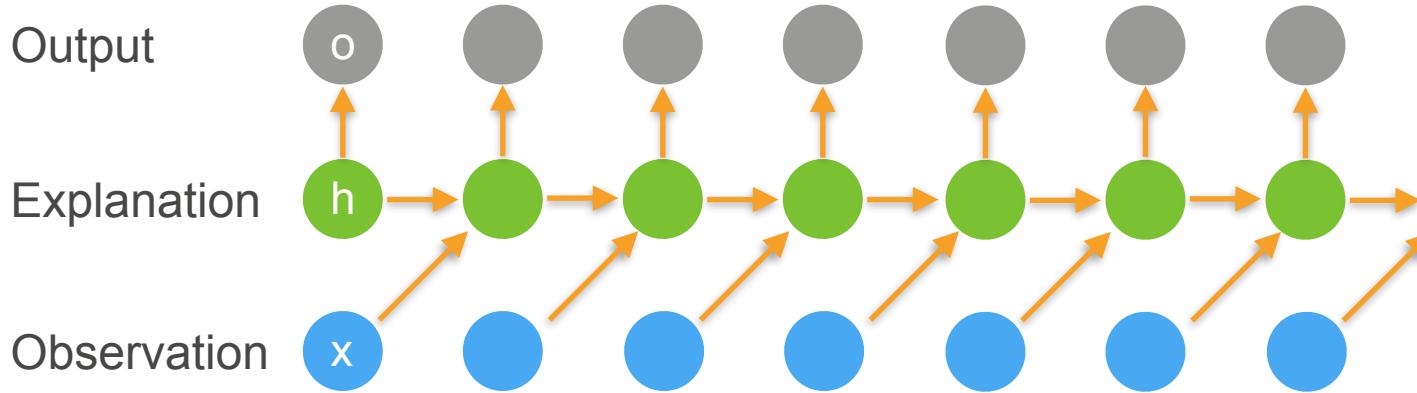
- Typically measure accuracy with log-likelihood
  - This makes outputs of different length incomparable (e.g. bad model on short output has higher likelihood than excellent model on very long output)
  - Normalize log-likelihood to sequence length
    - $\sum_{t=1}^T \log p(y_t | \text{model})$  vs.  $\pi := -\frac{1}{T} \sum_{t=1}^T \log p(y_t | \text{model})$

- Perplexity is exponentiated version  $\exp(\pi)$   
(effectively number of possible choices on average)



# Truncated Backprop Through Time

# Recurrent Neural Networks (with hidden state)



- Hidden State update

$$h_t = f(h_{t-1}, x_{t-1}, w)$$

- Observation update

$$o_t = g(h_t, w)$$

# Objective function

- RNN generates output which needs to be compared to target labels

$$L(x, y, w) = \sum_{t=1}^T l(y_t, o_t)$$

- Gradient

$$\partial_w L = \sum_{t=1}^T \partial_w l(y_t, o_t)$$

$$= \sum_{t=1}^T \partial_{o_t} l(y_t, o_t) \left[ \partial_w g(h_t, w) + \partial_{h_t} g(h_t, w) \partial_w h_t \right]$$

# Latent State Gradient $\partial_w h_t$

- Objective Function

$$\partial_w L = \sum_{t=1}^T \partial_w l(y_t, o_t) = \sum_{t=1}^T \partial_{o_t} l(y_t, o_t) \left[ \partial_w g(h_t, w) + \partial_{h_t} g(h_t, w) \partial_w h_t \right]$$

- Gradient Recursion

$$\partial_w h_t = \partial_w f(x_t, h_{t-1}, w) + \partial_h f(x_t, h_{t-1}, w) \partial_w h_{t-1}$$

$$= \sum_{i=t}^1 \left[ \prod_{j=t}^i \partial_h f(x_j, h_{j-1}, w) \right] \partial_w f(x_i, h_{i-1}, w)$$

# Latent State Gradient $\partial_w h_t$

- Gradient Recursion

$$\partial_w h_t = \sum_{i=t}^1 \left[ \prod_{j=t}^i \partial_h f(x_j, h_{j-1}, w) \right] \partial_w f(x_i, h_{i-1}, w)$$

Too Many Terms

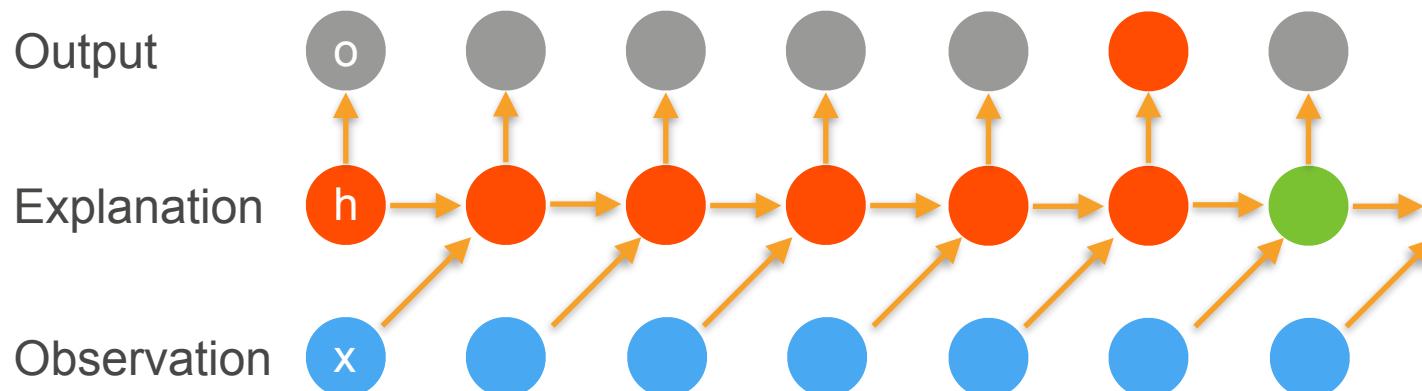
Unstable  
(divergence)

expensive

# Latent State Gradient $\partial_w h_t$

- Gradient Recursion

$$\partial_w h_t = \sum_{i=t}^1 \left[ \prod_{j=t}^i \partial_h f(x_j, h_{j-1}, w) \right] \partial_w f(x_i, h_{i-1}, w)$$



# Latent State Gradient $\partial_w h_t$

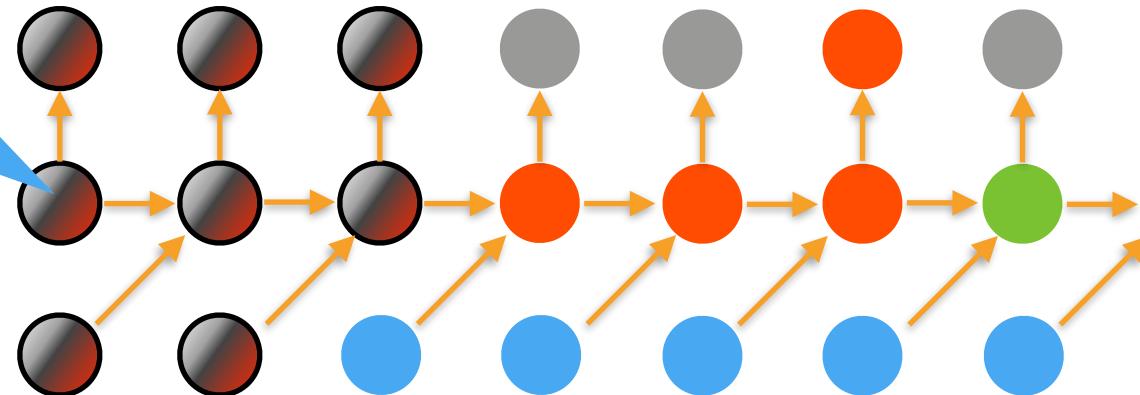
- Gradient Recursion

$$\partial_w h_t = \sum_{i=t}^1 \left[ \prod_{j=t}^i \partial_h f(x_j, h_{j-1}, w) \right] \partial_w f(x_i, h_{i-1}, w)$$

Drop  
gradients

Explanation

Observation

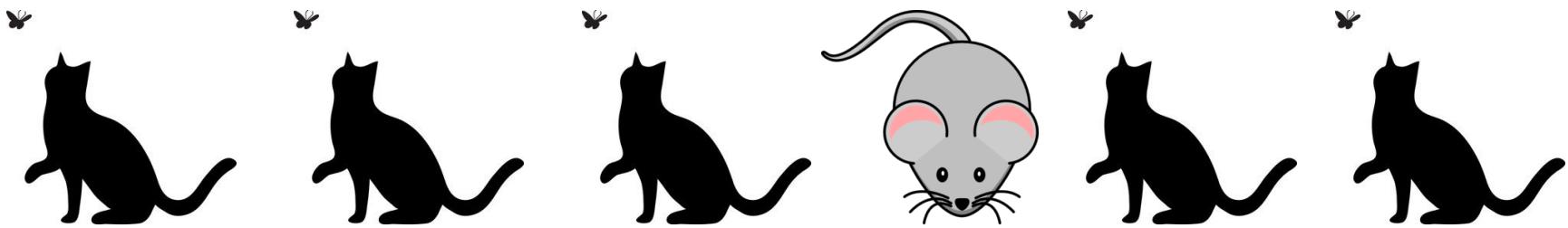


# Gated Recurrent Unit (GRU)



# Paying attention to a sequence

- Not all observations are equally relevant

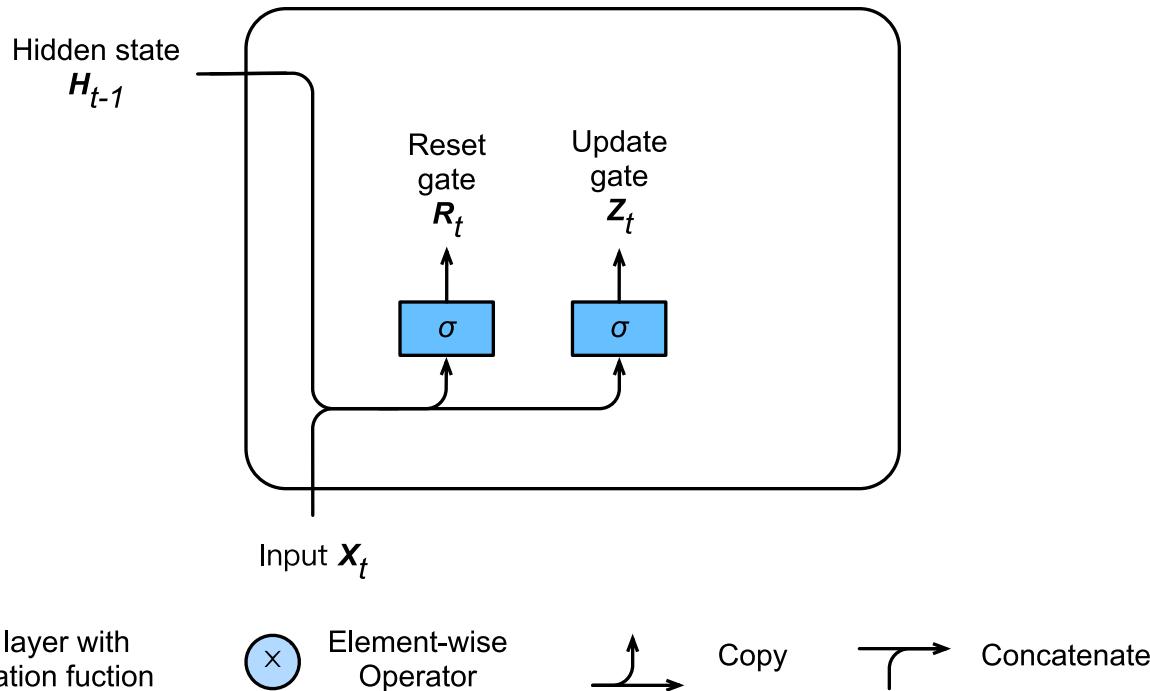


- Only remember the relevant ones
  - Need mechanism to **pay attention (update gate)**
  - Need mechanism to **forget (reset gate)**

# Gating

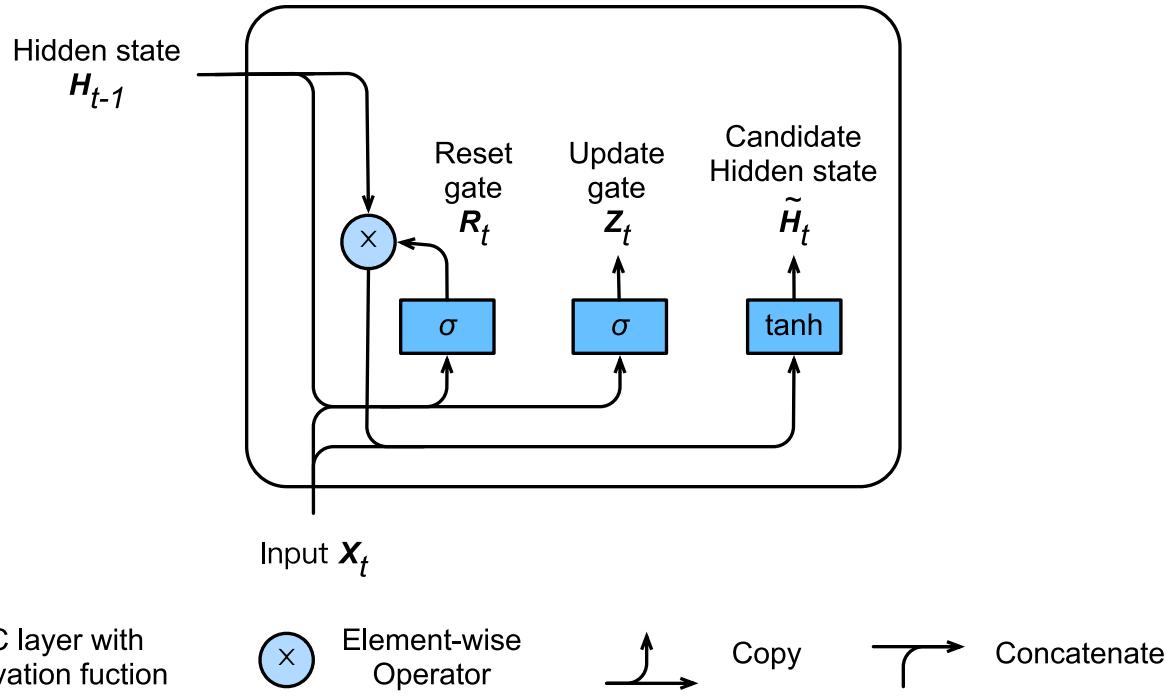
$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r),$$

$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$$



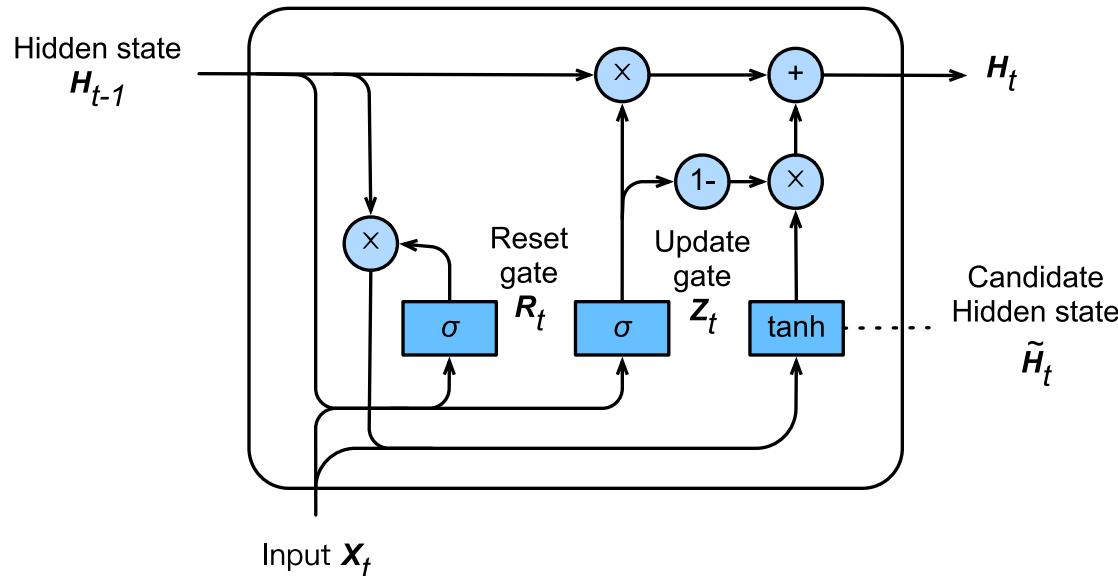
# Candidate Hidden State

$$\tilde{H}_t = \tanh(X_t W_{xh} + (R_t \odot H_{t-1}) W_{hh} + b_h)$$



# Hidden State

$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t$$



FC layer with  
activation function



Element-wise  
Operator



Copy



Concatenate

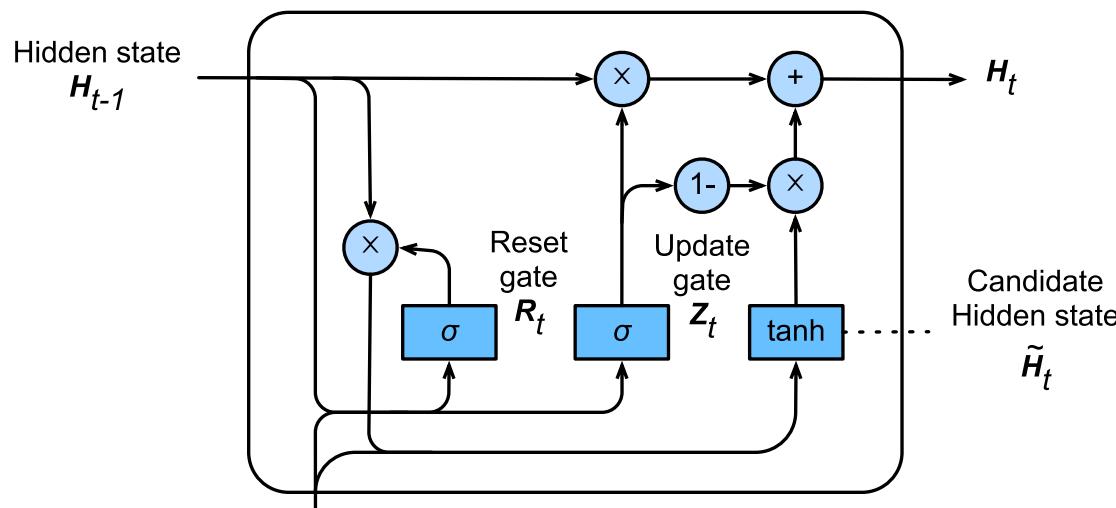
# Summary

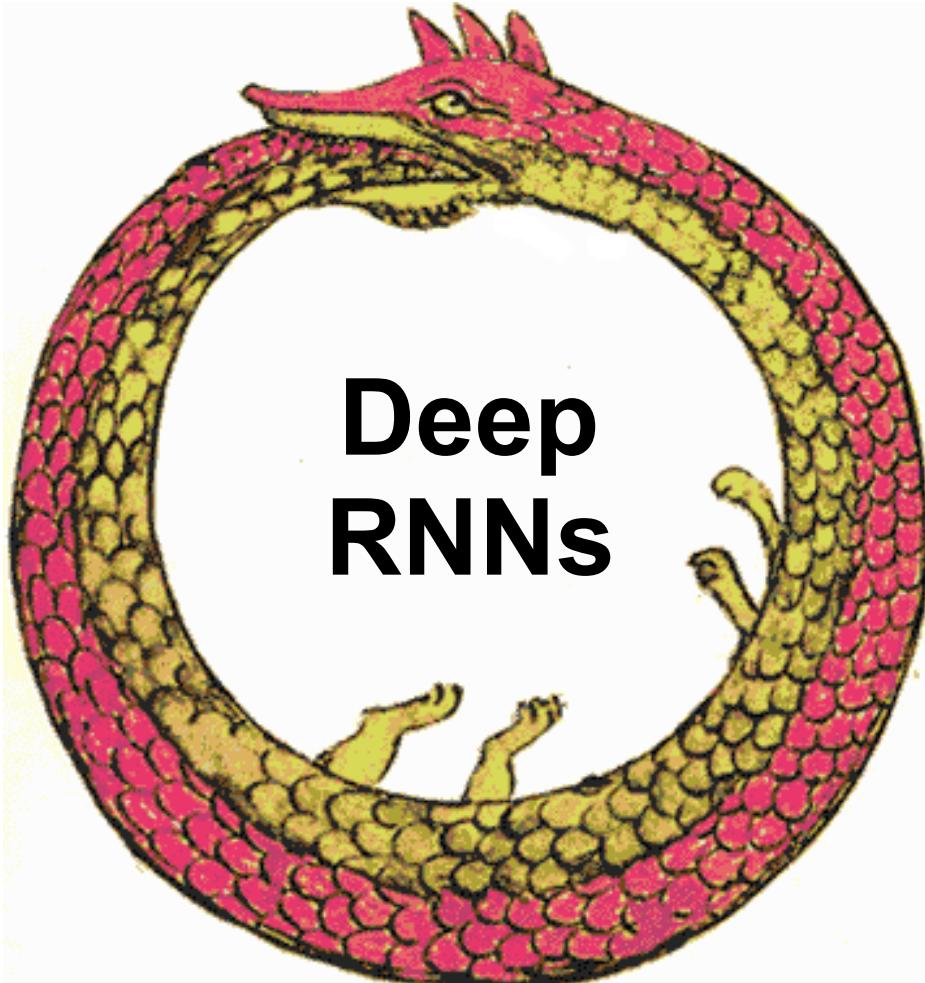
$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r),$$

$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$$

$$\tilde{H}_t = \tanh(X_t W_{xh} + (R_t \odot H_{t-1}) W_{hh} + b_h)$$

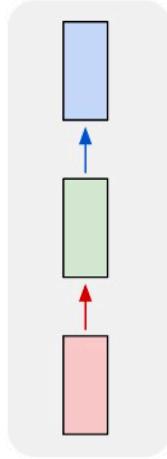
$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t$$



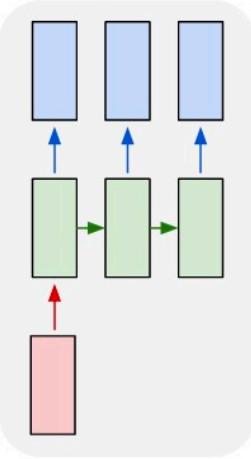


# Using RNNs

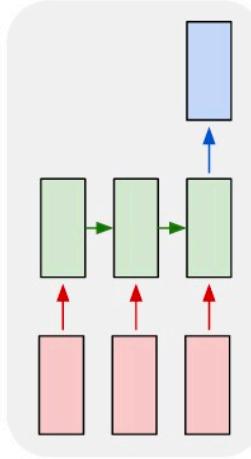
one to one



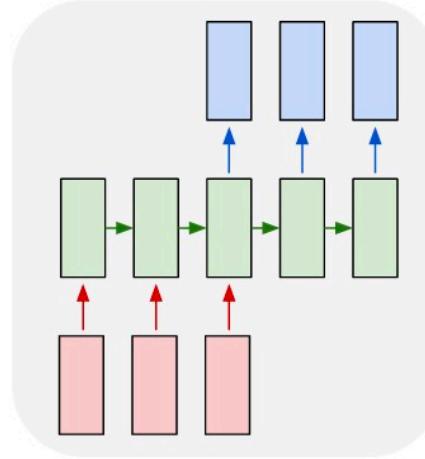
one to many



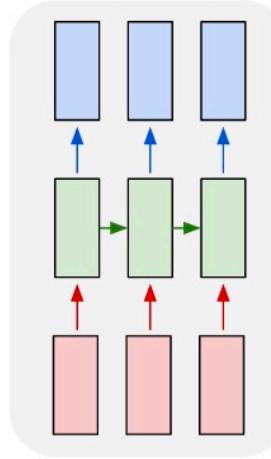
many to one



many to many



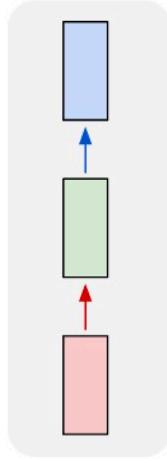
many to many



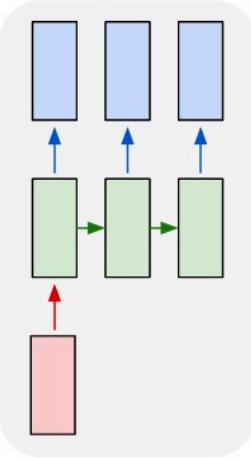
- Encode sequence
- Decode sequence
- Do both

# Using RNNs

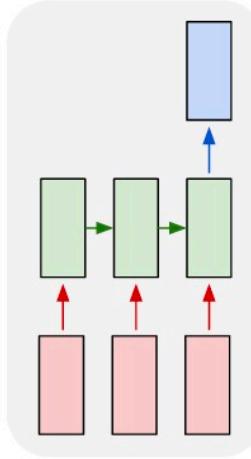
one to one



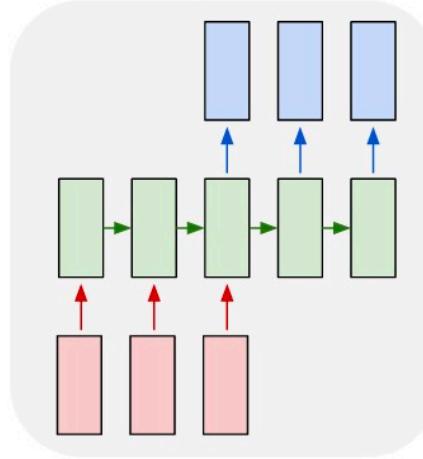
one to many



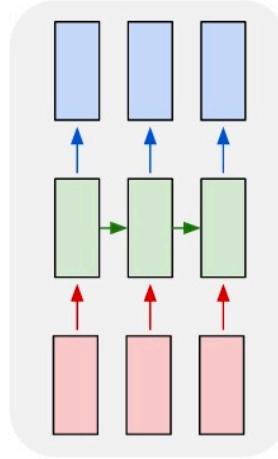
many to one



many to many



many to many



Poetry  
Generation

Sentiment  
Analysis

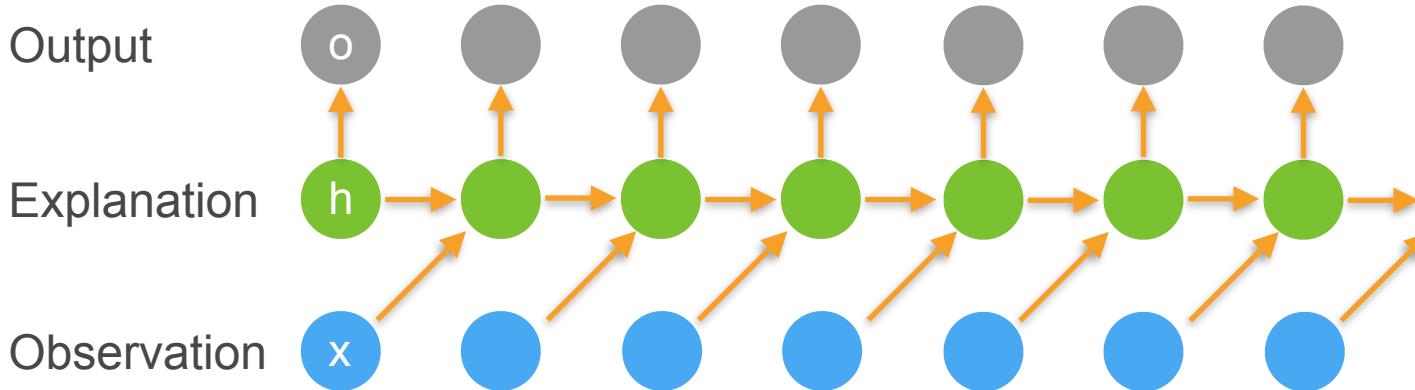
Document  
Classification

Question  
Answering

Machine  
Translation

Named  
Entity  
Tagging

# Recall - Recurrent Neural Networks



- Hidden State update

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

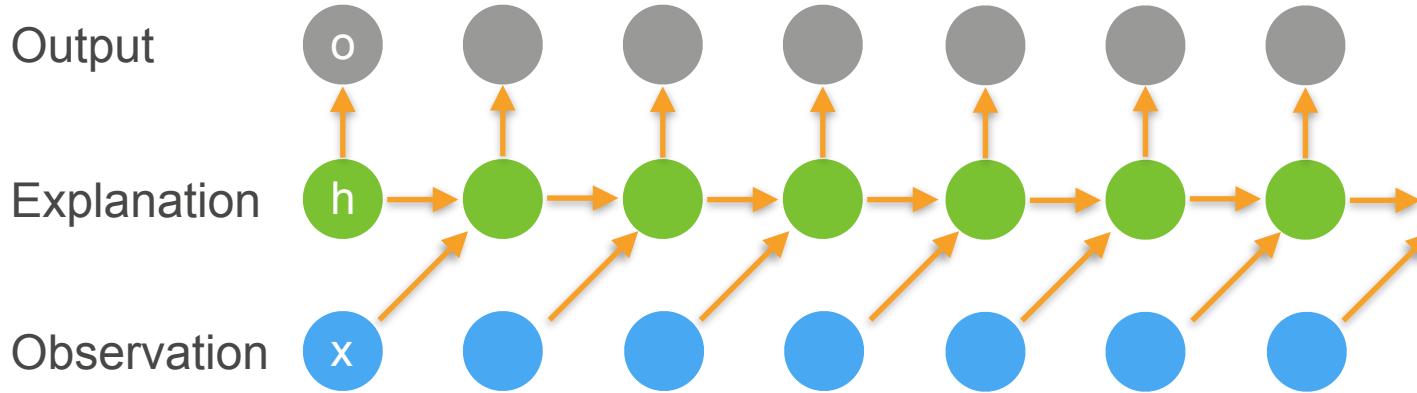
- Observation update

$$\mathbf{o}_t = \phi(\mathbf{W}_{ho}\mathbf{h}_t + \mathbf{b}_o)$$

How to make  
more nonlinear?



# Plan A - Nonlinearity in the units



- Hidden State update

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

- Observation update

$$\mathbf{o}_t = \phi(\mathbf{W}_{ho}\mathbf{h}_t + \mathbf{b}_o)$$

Replace with  
MLP?

# Plan A - Nonlinearity in the units

- Keeps the structure of the latent space
- More complex gradients (very costly)
- E.g. Zoph et al, 2018 learned cells with ~40 units  
**(slow and expensive - nobody uses them in practice)**

- Hidden State update

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

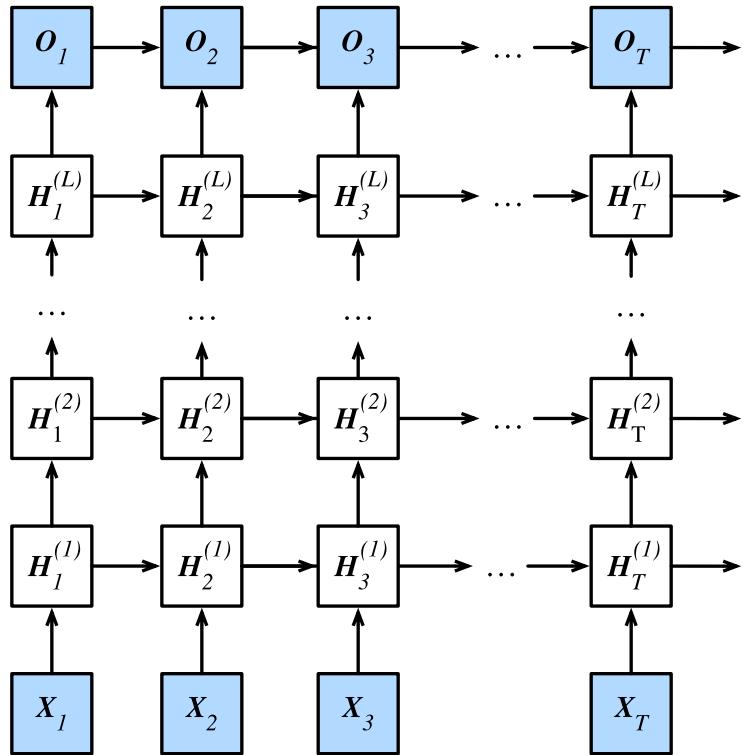
- Observation update

$$\mathbf{o}_t = \phi(\mathbf{W}_{ho}\mathbf{h}_t + \mathbf{b}_o)$$

Replace with  
MLP?

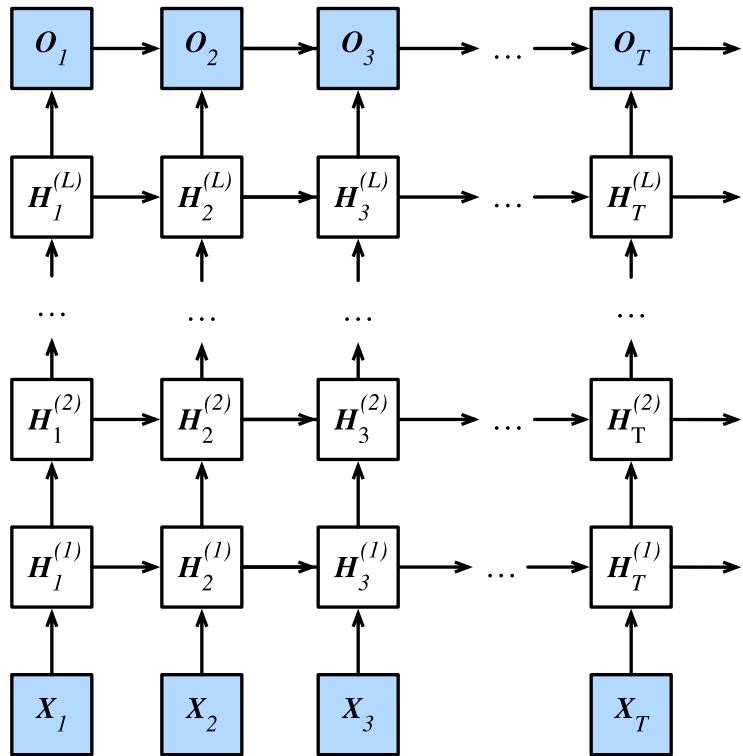


# Plan B - We go deeper



- Shallow RNN
  - Input
  - Hidden layer
  - Output
- Deep RNN
  - Input
  - **Hidden layer**
  - **Hidden layer**
  - ...
  - Output

# Plan B - We go deeper



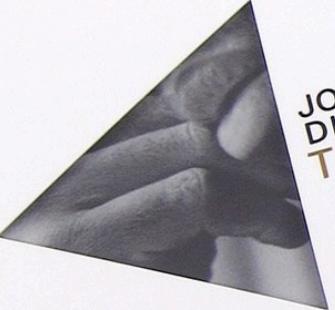
$$\mathbf{H}_t = f(\mathbf{H}_{t-1}, \mathbf{X}_t)$$

$$\mathbf{O}_t = g(\mathbf{H}_t)$$

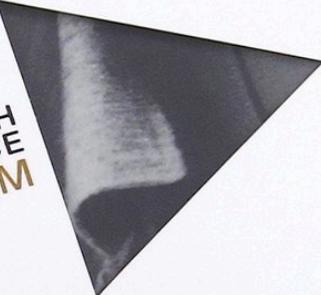
$$\mathbf{H}_t^1 = f_1(\mathbf{H}_{t-1}^1, \mathbf{X}_t)$$

$$\mathbf{H}_t^j = f_j(\mathbf{H}_{t-1}^j, \mathbf{H}_t^{j-1})$$

$$\mathbf{O}_t = g(\mathbf{H}_t^L)$$



JOHN COLTRANE BOTH  
DIRECTIONS AT ONCE  
**THE LOST ALBUM**



# Bidirectional RNNS



# The Future Matters

I am \_\_\_\_\_

I am \_\_\_\_\_ very hungry,

I am \_\_\_\_\_ very hungry, I could eat half a pig.

# The Future Matters

I am **happy**.

I am **not** very hungry,

I am **very** very hungry, I could eat half a pig.

# The Future Matters

I am **happy**.

I am **not** very hungry,

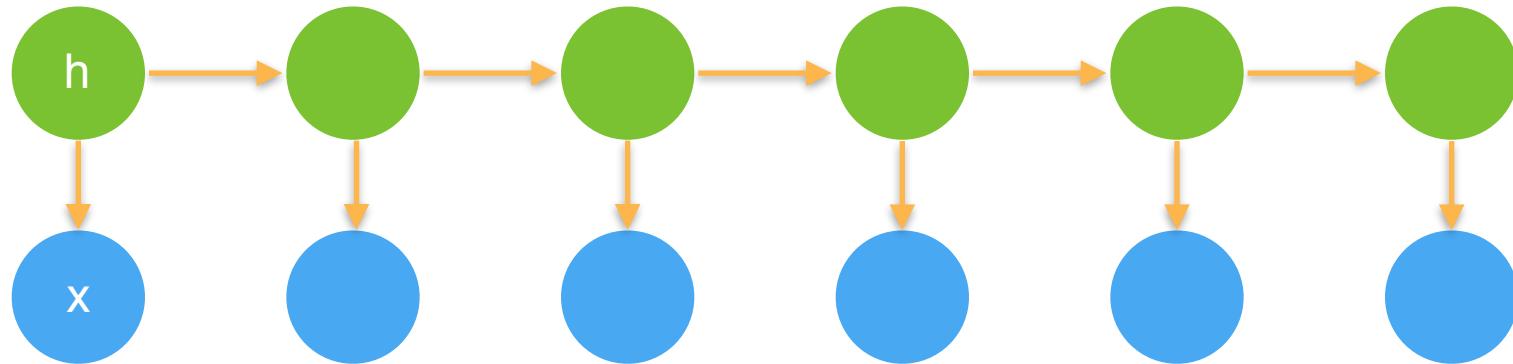
I am **very** very hungry, I could eat half a pig.

- Very different words to fill in, depending on past and **future** context of a word.
- RNNs so far only look at the past
- In interpolation (fill in) we can use the future, too.

# Flashback - Graphical Models

- Hidden Markov Model

$$p(h_t | h_{t-1}, x_{t-1}) \text{ and } p(x_t | h_t, x_{t-1})$$



- Can model sequence jointly and solve by dynamic programming

# Dynamic programming

- Joint probability

$$p(x, h) = p(h_1)p(x_1 | h_1) \prod_{i=2}^T p(h_i | h_{i-1})p(x_i | h_i)$$

# Dynamic programming

$$\begin{aligned} p(x) &= \sum_h p(h_1)p(x_1 | h_1) \prod_{i=2}^T p(h_t | h_{t-1})p(x_t | h_t) \\ &= \sum_{h_2, \dots, h_T} \underbrace{\left[ \sum_{h_1} p(h_1)p(x_1 | h_1)p(h_2 | h_1) \right]}_{=: \pi_2(h_2)} p(x_2 | h_2) \prod_{i=2}^T p(h_t | h_{t-1})p(x_t | h_t) \\ &= \sum_{h_3, \dots, h_T} \underbrace{\left[ \sum_{h_2} \pi_2(h_2)p(x_2 | h_2)p(h_3 | h_2) \right]}_{=: \pi_3(h_3)} p(x_3 | h_3) \prod_{i=3}^T p(h_t | h_{t-1})p(x_t | h_t) \end{aligned}$$

# Dynamic programming

- Joint probability

$$p(x, h) = p(h_1)p(x_1 | h_1) \prod_{i=2}^T p(h_i | h_{i-1})p(x_i | h_i)$$

- Forward pass

$$\pi_{t+1}(h_{t+1}) = \sum_{h_t} \pi_t(h_t)p(x_t | h_t)p(h_{t+1} | h_t)$$

# Dynamic programming

$$p(x) = \sum_h \prod_{i=1}^{T-1} p(h_t | h_{t-1}) p(x_t | h_t) \cdot p(h_T | h_{T-1}) p(x_T | h_T)$$

$$= \sum_{h_1, \dots, h_{T-1}} \prod_{i=1}^{T-1} p(h_t | h_{t-1}) p(x_t | h_t) \cdot \underbrace{\left[ \sum_{h_T} p(h_T | h_{T-1}) p(x_T | h_T) \right]}_{=: \rho_{T-1}(h_{T-1})}$$

$$= \sum_{h_1, \dots, h_{T-2}} \prod_{i=1}^{T-2} p(h_t | h_{t-1}) p(x_t | h_t) \cdot \underbrace{\left[ \sum_{h_{T-1}} p(h_{T-1} | h_{T-2}) p(x_{T-1} | h_{T-1}) \right]}_{=: \rho_{T-2}(h_{T-2})}$$



# Dynamic programming

- Joint probability

$$p(x, h) = p(h_1)p(x_1 | h_1) \prod_{i=2}^T p(h_i | h_{i-1})p(x_i | h_i)$$

- Forward pass

$$\pi_{t+1}(h_{t+1}) = \sum_{h_t} \pi_t(h_t)p(x_t | h_t)p(h_{t+1} | h_t)$$

- Backward pass

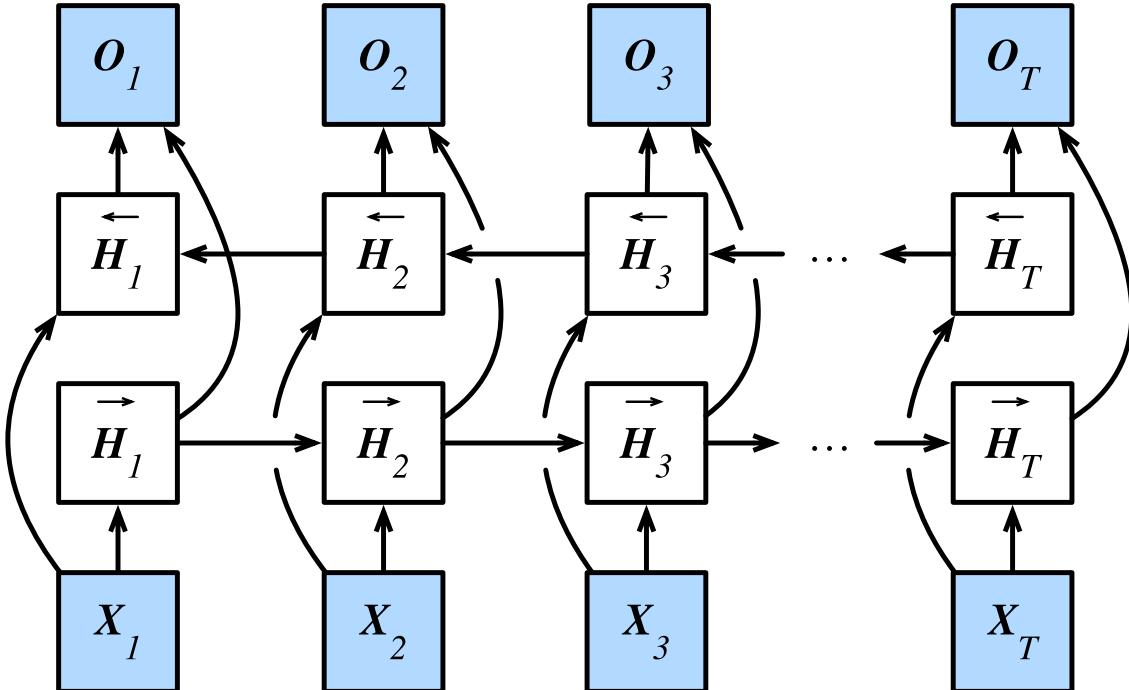
$$p(x_j | x_{-j}) \propto \sum_{h_j} \pi_j(h_j)\rho_j(h_j)p(x_j | h_j)$$

I WANT

IT ALL

Can we do this with RNNs, too?

# Bidirectional RNN



- One RNN forward
- Another one backward
- Combine both hidden states for output generation

```
epoch 600, perplexity 1.016867, time 0.15 sec
- travellerer cumplhp peougunininininin suppepepepepepepepe
- time travellererer fuf this shanatatatatatatatatatatatatatata
epoch 800, perplexity 1.007069, time 0.15 sec
- traveller hime of copspepepep smefsffff'::::::::::::::::::
- time travellerer prefififididididididididididididididididi
epoch 1000, perplexity 1.001932, time 0.15 sec
- travellererererererererererererererererererererererererer
- time travellererererererererererererererererererererererer
```

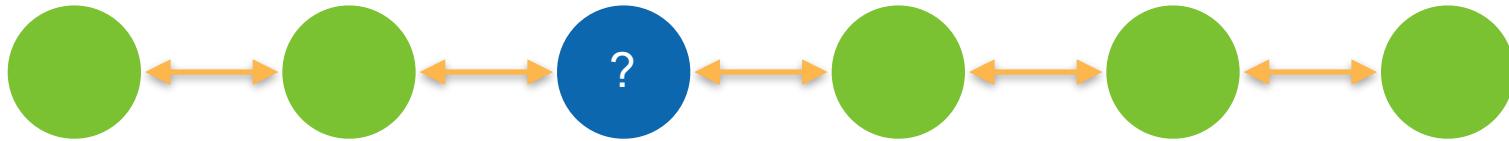
**This does not work for  
sequence generation**

```
epoch 600, perplexity 1.016867, time 0.15 sec
- travellerer cumplph peougunininininin suppepepepepepepepe
- time travellererer fuf this shanatatatatatatatatatatatatatata
epoch 800, perplexity 1.007069, time 0.15 sec
- traveller hime of copspepepep smefsffff'::::::::::::::::::
- time travellerer prefififididididididididididididididididi
epoch 1000, perplexity 1.001932, time 0.15 sec
- travellererererererererererererererererererererererererer
- time travellererererererererererererererererererererererer
```

# Why?

# Reasons

- Training time



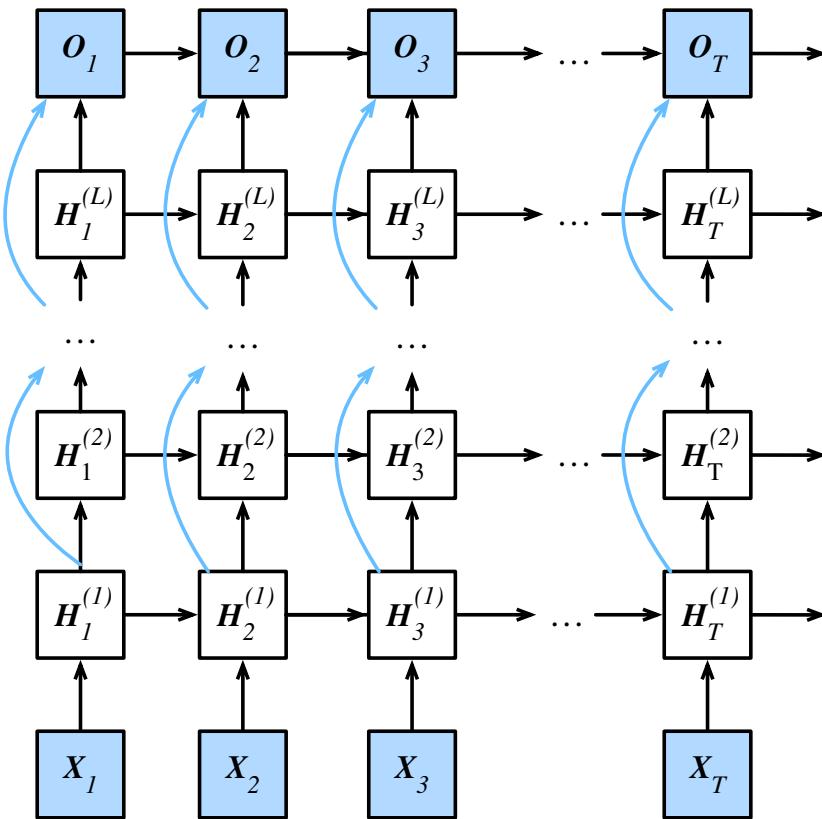
- Test time



Next  
lecture

Can still use it to **encode** the sequence

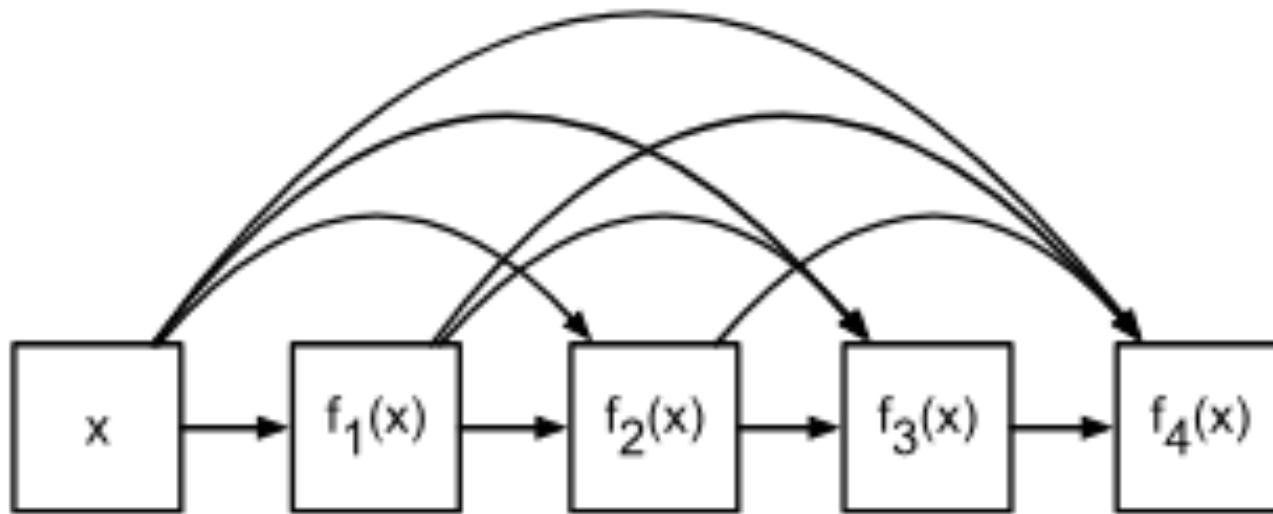
# Residual RNNs



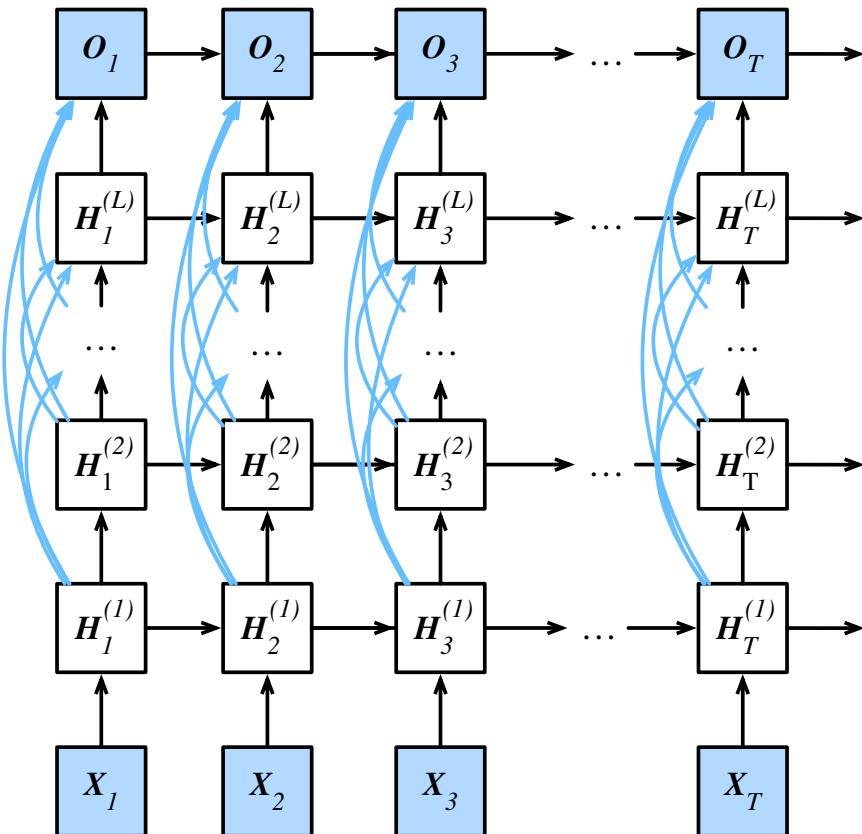
$$\bar{H}_t^{(2i)} = H_t^{(2i)} + H_t^{(2i)-1}$$

- Input of every second layer is also added to its output (residual connection)
- Variants
  - Simple addition
  - Nonlinearity before addition
  - Could also concatenate

# What about DenseNet?



# RNN with DenseNet Connections



$$\bar{\mathbf{H}}_t^{(t)} = [\mathbf{H}_t^{(t)}, \bar{\mathbf{H}}_t^{t-1}]$$

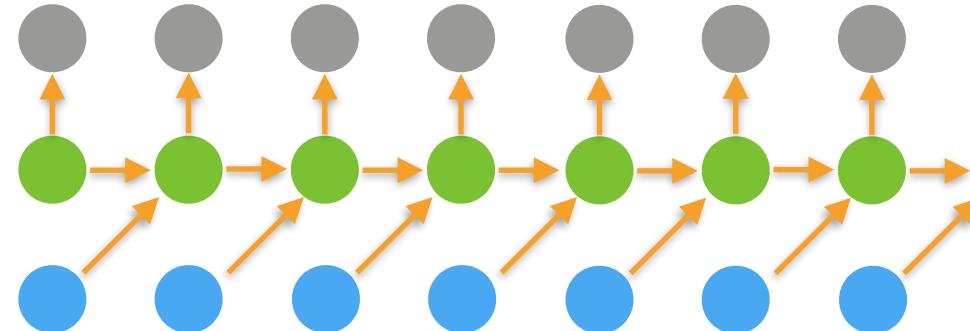
- Concatenate outputs of previous layers as input top the next layer
- Occasionally add transition layers to reduce dimensionality

# Regularization In RNNs



# Overfitting

- RNNs overfit just like any other model
- Sequential dependence is more difficult to control
  - Capacity in depth can be controlled, e.g. by dropout
  - For sequential part need to decide how to deal with variable inputs, e.g. input might be skipped)
  - If we use dropout we might miss relevant aspects in the coordinates.



# Flashback - Applying Dropout

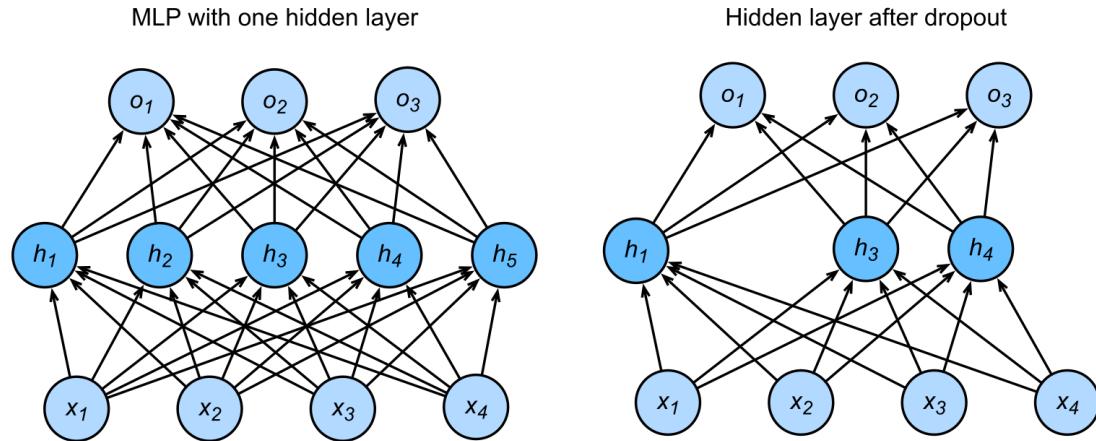
- Often apply dropout on the output of hidden fully-connected layers

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$

$$\mathbf{o} = \mathbf{W}_2 \mathbf{h}' + \mathbf{b}_2$$

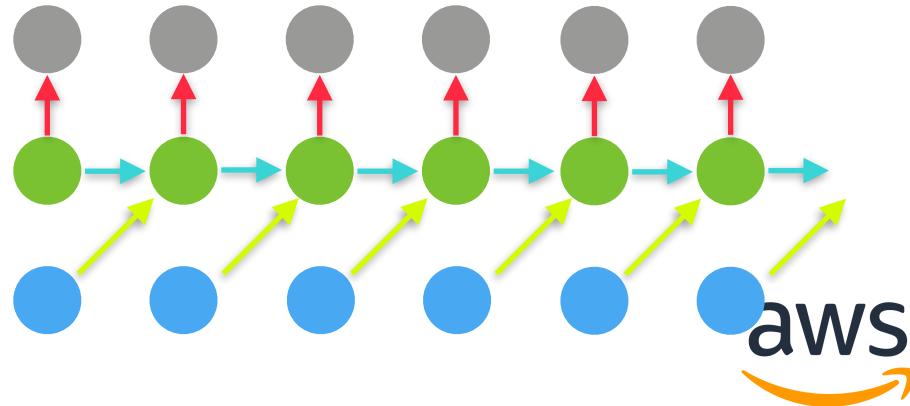
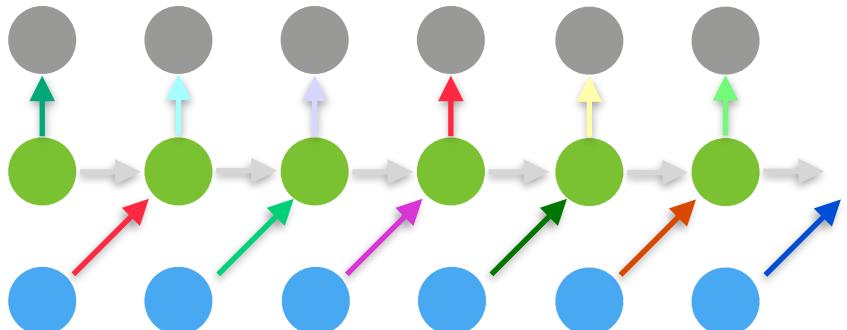
$$\mathbf{y} = \text{softmax}(\mathbf{o})$$



- At **inference** time dropout is inactive, i.e.  $\mathbf{h}' = \text{dropout}(\mathbf{h})$

# Variational Dropout (Gal & Ghahramani, 2015)

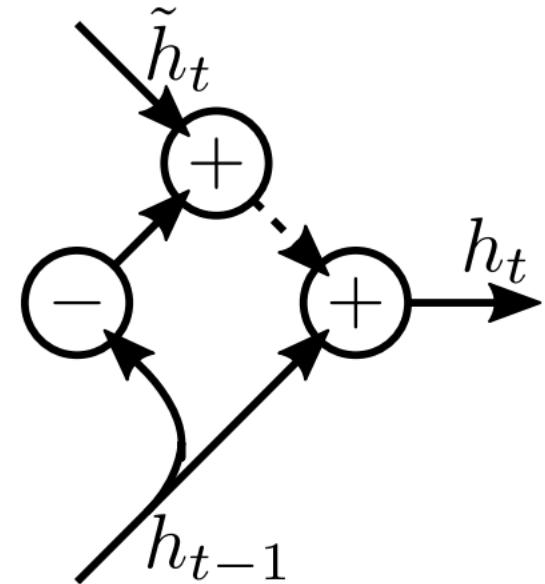
- Regular Dropout
  - Use only per time slice
  - Different mask per slice
- Same mask across all time windows
  - Use also for temporal aspect



# Zoneout (Krueger et al., 2016)

- Robustness against skipping observations in sequence
- Robustness of state representation relative to hidden state updates
- Skip hidden state update and keep the same as previously during training

$$h_t = h_{t-1}$$



# Many more tricks

- Parameter averaging (Merity et al., 2017)  
Train RNN and average weights over run
- Stochastic Weight Averaging (Wilson et al., 2018)  
Same approach but keep on changing learning rate
- Fraternal Dropout (Zolna et al., 2017)  
Dropout while minimizing variation between outputs to increase robustness to parametrization

...