

# **Recurrent Neural Nets for Music Generation**

**Computational Music Creativity, SMC (2020/21)**

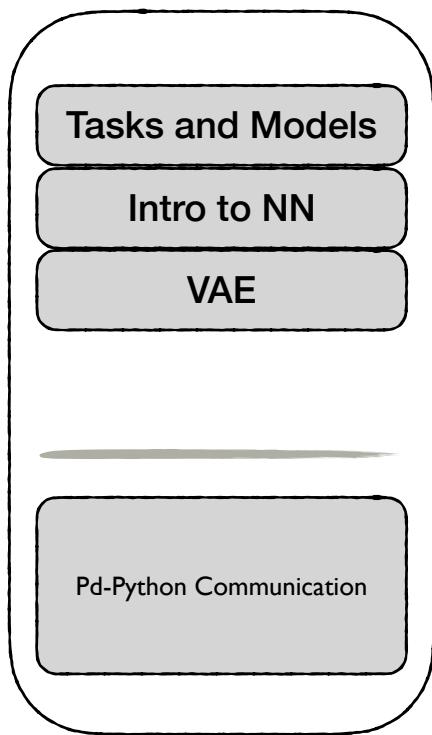
**Prepared by Behzad Haki, February 2021**

# Tentative Plan

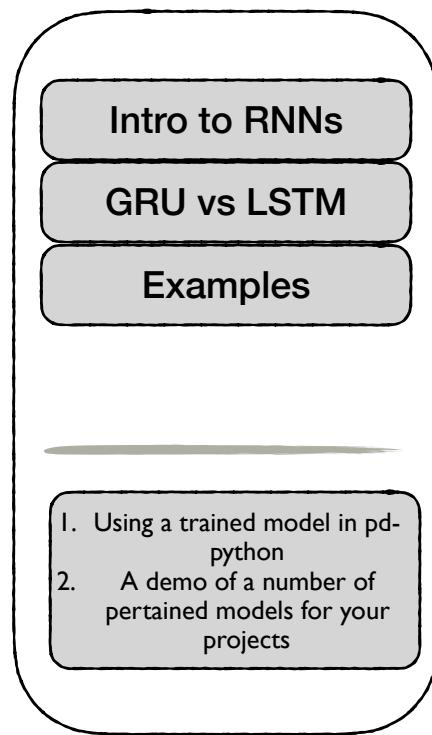
Theory Sessions

Lab Sessions

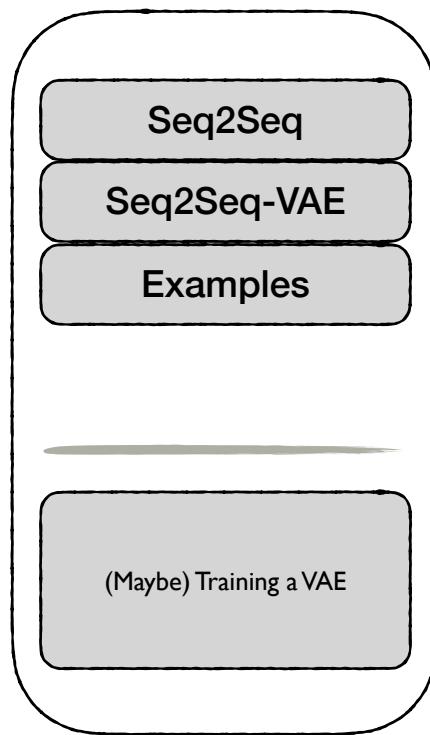
## Part 1



## Part 2(A)



## Part 2(B)



# Plan for Today

## Part B (Week 7)

Theory Session

Intro to RNNs

GRU vs LSTM

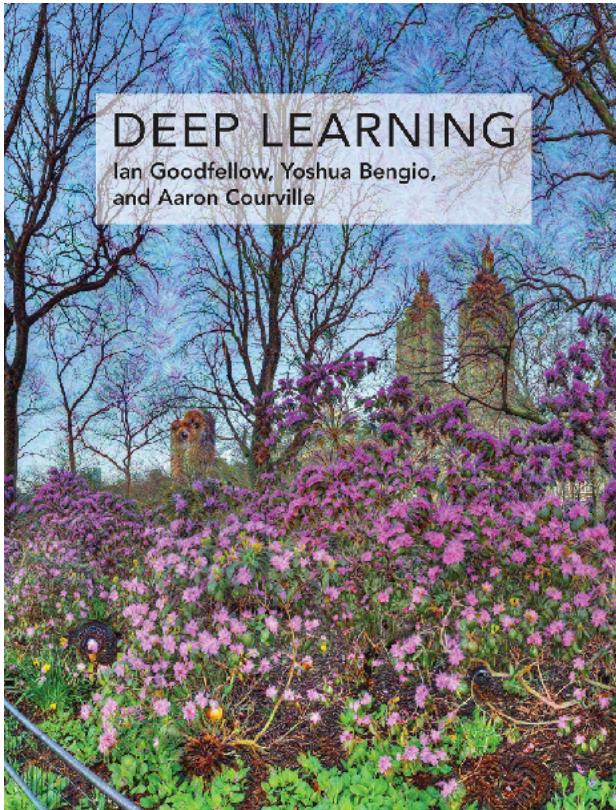
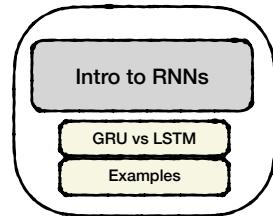
Examples

Lab Sessions

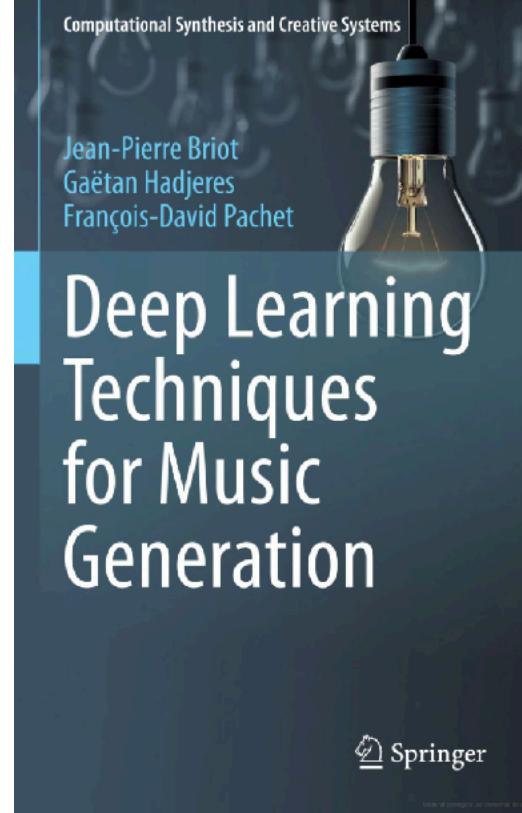
Melody Continuation with  
LSTMs

- 1. We'll start with a discussion on RNNs
- 2. We'll talk about some recurrent building blocks
- 3. We'll talk about some generative models based on RNNs

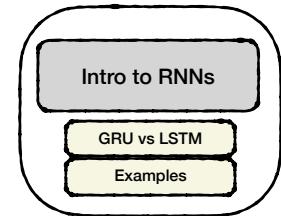
# References



[1] Goodfellow, Ian, et al. *Deep learning*. Vol. 1. No. 2. Cambridge: MIT press, 2016. Chapter 10.

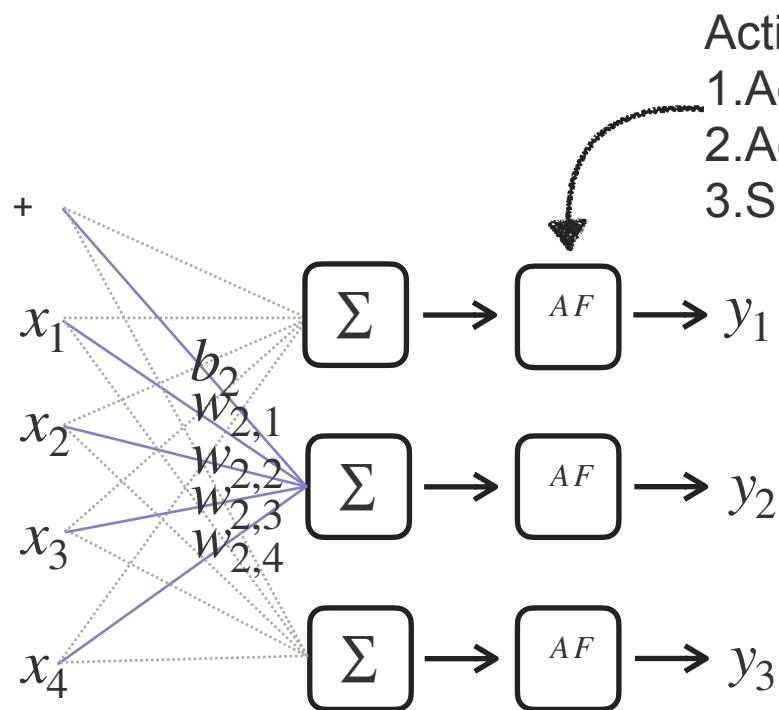


[2] Briot, Jean-Pierre, Gaëtan Hadjeres, and François Pachet. *Deep learning techniques for music generation*. Springer, 2020.



# Multi-Layer Perceptron (MLP)

Review of Last Lecture



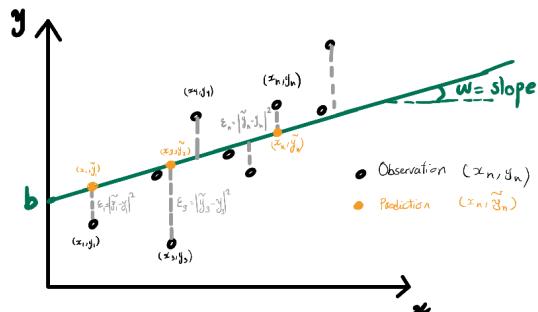
Activation Function, used to:

1. Add Nonlinearity,
2. Activate/Deactivate neuron,
3. Specify the type of output (probability vs. Scalar)

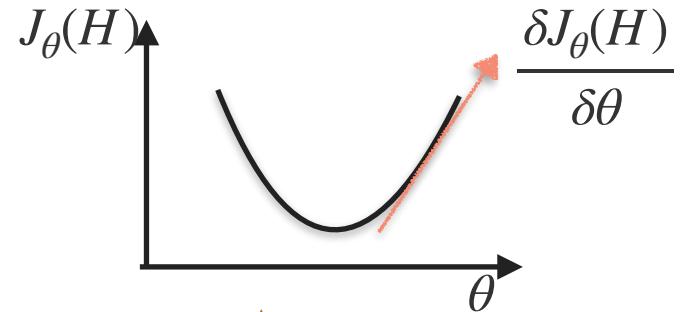
$$AF\left(\begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}\right) = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = AF(Wx + b) =$$

# Multi-Layer Perceptron (MLP)

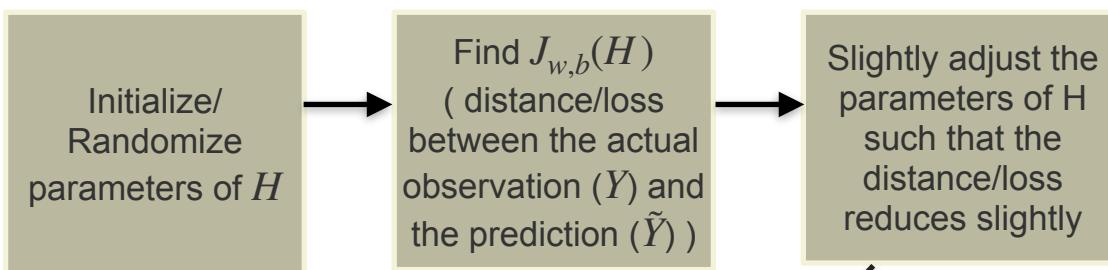
## Review of Last Lecture



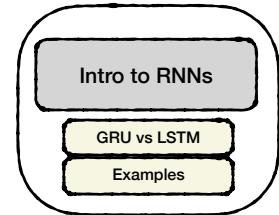
$$J_{w,b}(H) = 1/m \sum_{i=1}^m (y_i - \tilde{y})^2 = 1/m \sum_{i=1}^m (y_i - H(x_i))^2$$



To get to minimum, move  $\theta$  in the opposite direction of the gradient



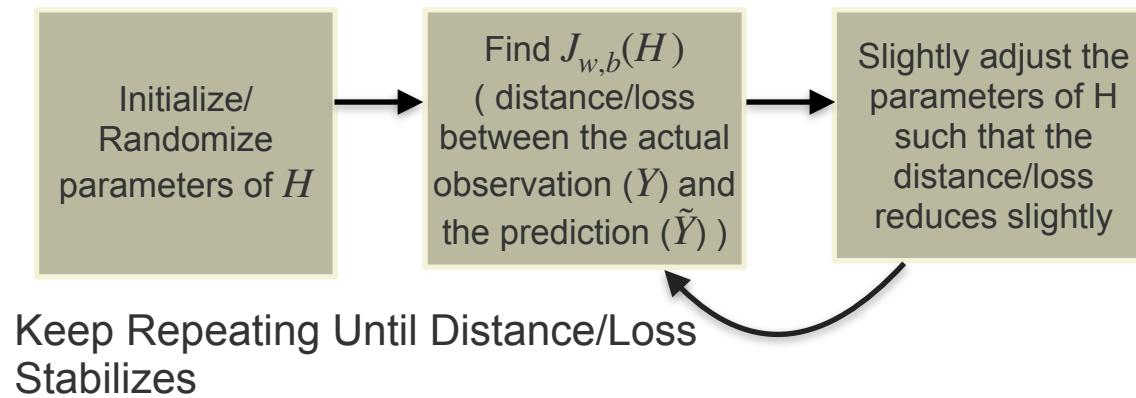
Keep Repeating Until Distance/Loss Stabilizes



# Revisiting MLPs for Sequential Data

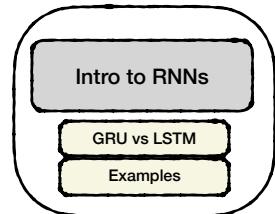
## Issue 1: Vanishing Gradient

Remember the discussion about model training/optimization from last time?!



We use the gradient of our loss function w.r.t parameters in our model to recursively update the parameters until we see no more significant improvement in the model!

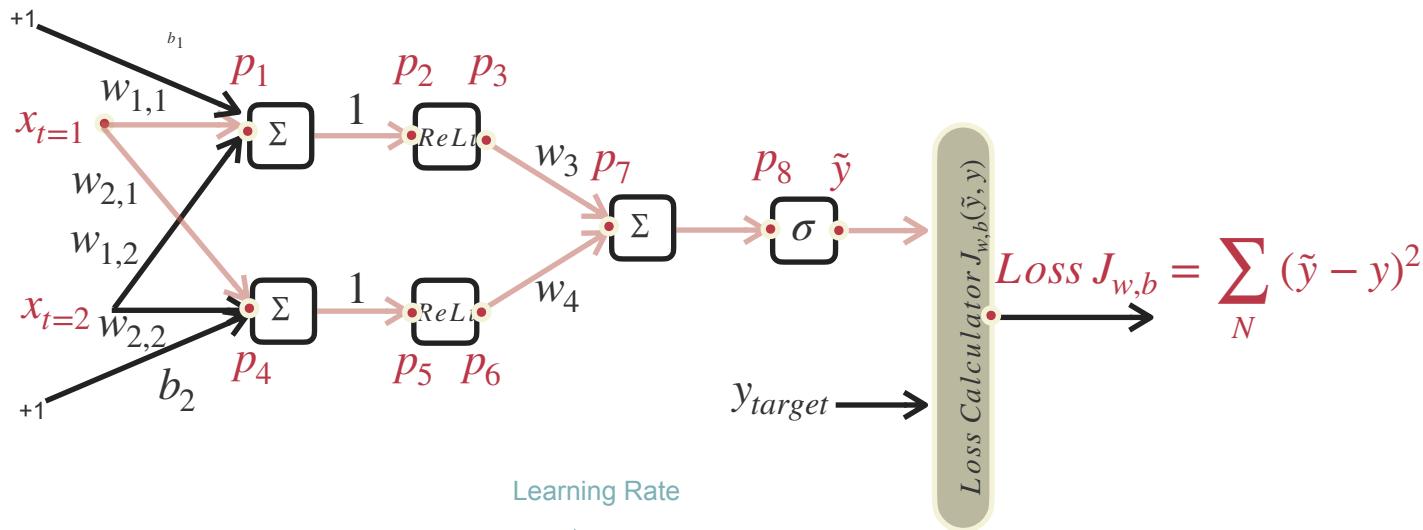
To find the gradients, we use back-propagation!



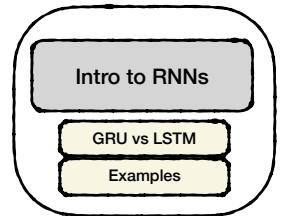
# Revisiting MLPs for Sequential Data

## Issue 1: Vanishing Gradient

Below is a simple example for back-propagation of gradients



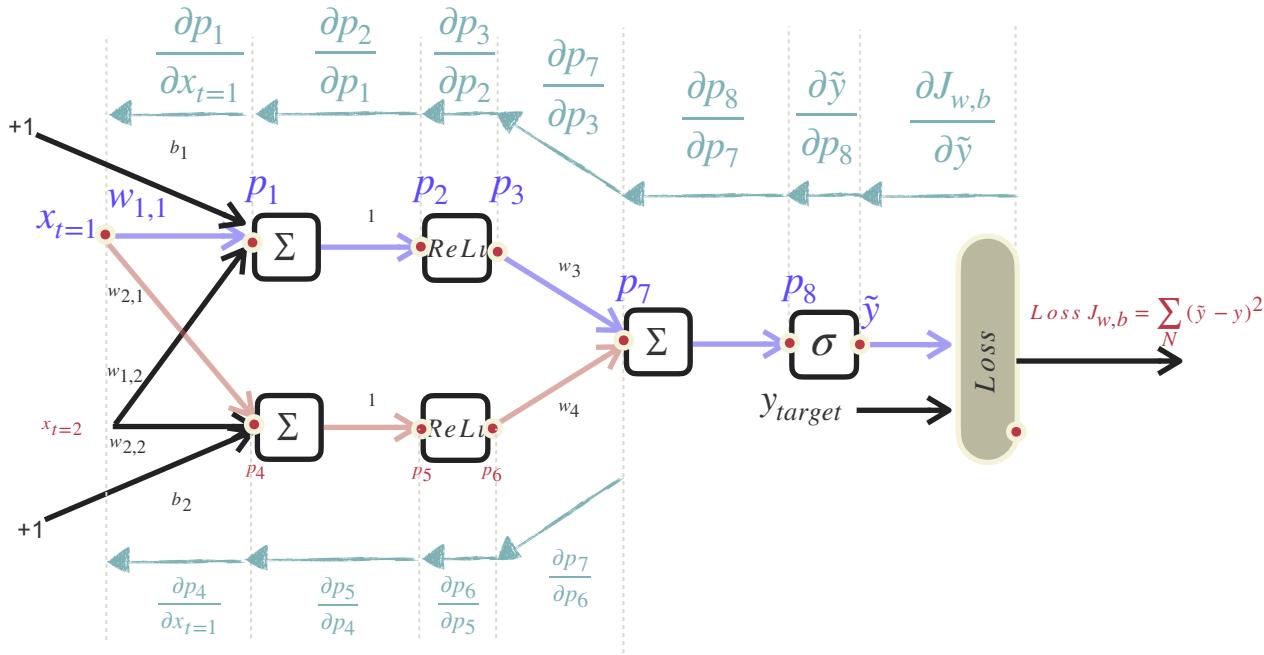
$$\begin{aligned}
 \text{Updated Parameters} &= \text{Old Parameters} - \alpha \text{Gradients} \\
 \begin{bmatrix} w_{1,1} \\ w_{2,1} \\ w_{1,2} \\ w_{2,2} \\ w_{w,3} \\ w_{w,4} \\ b_1 \\ b_2 \end{bmatrix} &= \begin{bmatrix} w_{1,1} \\ w_{2,1} \\ w_{1,2} \\ w_{2,2} \\ w_3 \\ w_4 \\ b_1 \\ b_2 \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial J}{\partial w_{1,1}} \\ \frac{\partial J}{\partial w_{2,1}} \\ \frac{\partial J}{\partial w_{1,2}} \\ \frac{\partial J}{\partial w_{2,2}} \\ \frac{\partial J}{\partial w_3} \\ \frac{\partial J}{\partial w_4} \\ \frac{\partial J}{\partial b_1} \\ \frac{\partial J}{\partial b_2} \end{bmatrix}
 \end{aligned}$$



# Revisiting MLPs for Sequential Data

## Issue 1: Vanishing Gradient

Below is a simple example for back-propagation of gradients



$$\frac{\partial p_3}{\partial p_2} = \frac{\partial \text{ReLU}(p_2)}{\partial p_2} \in \{0,1\}$$

$$\frac{\partial p_7}{\partial p_3} = w_3, \frac{\partial p_7}{\partial p_6} = w_4$$

$$\frac{\partial p_1}{\partial x_{t=1}} = w_{1,1}, \frac{\partial p_4}{\partial x_{t=1}} = w_{2,1}$$

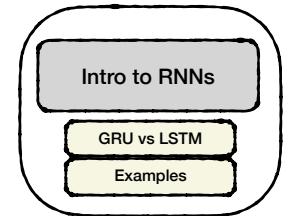
$$\frac{\partial J_{w,b}}{\partial \tilde{y}} = \frac{2}{N} * \sum_N (\tilde{y} - y)$$

$$\frac{\partial \tilde{y}}{\partial p_8} = \frac{\partial \sigma(p_8)}{\partial p_8} = \sigma(x)(1 - \sigma(x))$$

$$\frac{\partial p_8}{\partial p_7} = \frac{\partial p_2}{\partial p_1} = \frac{\partial p_5}{\partial p_4} = 1$$

$$\frac{\partial p_1}{\partial w_{1,1}} = \frac{\partial (x_{t=0} w_{1,1})}{\partial w_{1,1}}$$

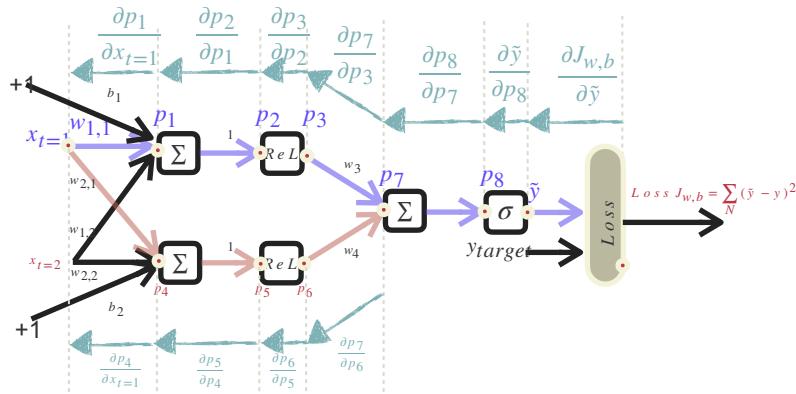
$$\frac{\partial J_{w,b}}{\partial w_{1,1}} = \frac{\partial J_{w,b}}{\partial \tilde{y}} \frac{\partial \tilde{y}}{\partial p_8} \frac{\partial p_8}{\partial p_7} \frac{\partial p_7}{\partial p_3} \frac{\partial p_3}{\partial p_2} \frac{\partial p_2}{\partial p_1} \frac{\partial p_1}{\partial w_{1,1}} = \frac{\partial J_{w,b}}{\partial \tilde{y}} \frac{\partial \tilde{y}}{\partial p_8} \frac{\partial p_8}{\partial p_7} \frac{\partial p_7}{\partial p_3} \frac{\partial p_3}{\partial p_2} \frac{\partial p_2}{\partial p_1} x_{t=0}$$



# Revisiting MLPs for Sequential Data

## Issue 1: Vanishing Gradient

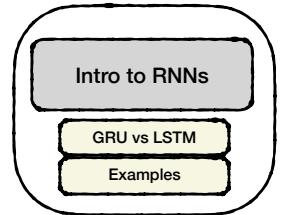
Below is a simple example for back-propagation of gradients



Sometimes these partials become so small that  $\frac{\partial J_{w,b}}{\partial x_{w_{1,1}}}$  becomes zero

Hence, no further training can be done —> Vanishing Gradient

$$\frac{\partial J_{w,b}}{\partial w_{1,1}} = \frac{\partial J_{w,b}}{\partial \tilde{y}} \frac{\partial \tilde{y}}{\partial p_8} \frac{\partial p_8}{\partial p_7} \frac{\partial p_7}{\partial p_3} \frac{\partial p_3}{\partial p_2} \frac{\partial p_2}{\partial p_1} \frac{\partial p_1}{\partial w_{1,1}} = \frac{\partial J_{w,b}}{\partial \tilde{y}} \frac{\partial \tilde{y}}{\partial p_8} \frac{\partial p_8}{\partial p_7} \frac{\partial p_7}{\partial p_3} \frac{\partial p_3}{\partial p_2} \frac{\partial p_2}{\partial p_1} x_{t=0}$$

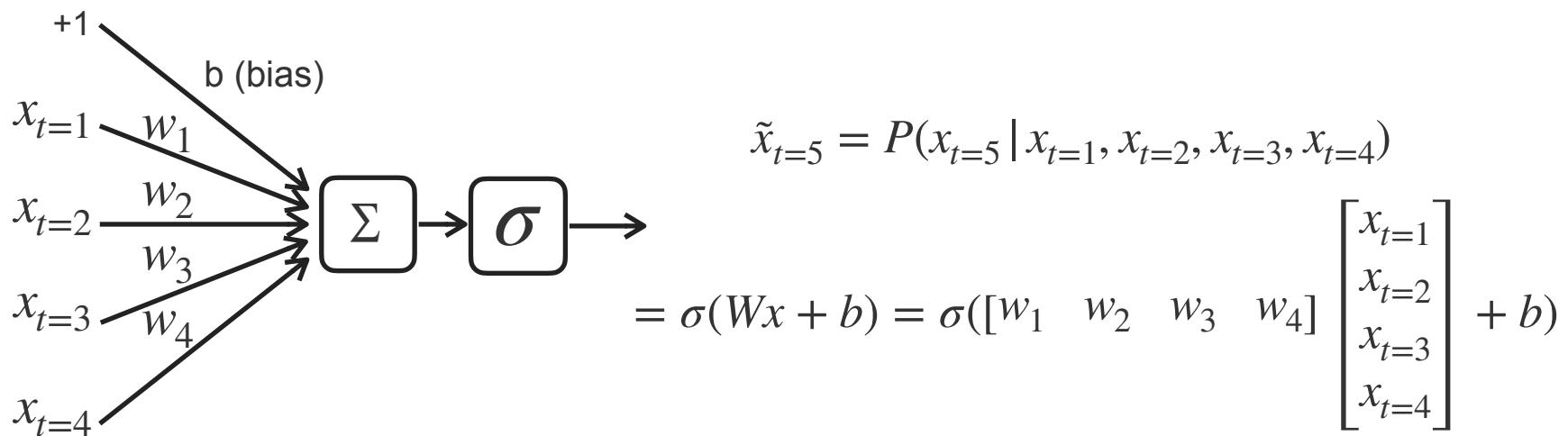


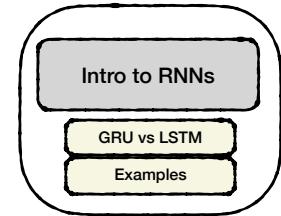
# Revisiting MLPs for Sequential Data

## Issue 2: Fixed number of time-steps

Let's assume we have the following model which tries to predict whether a hi-hat should be played given a sequence of hi-hats previously played.

*Using  $\{x_{t=1}, x_{t=2}, x_{t=3}, x_{t=4}\} \Rightarrow \text{Predict } x_{t=5}$*

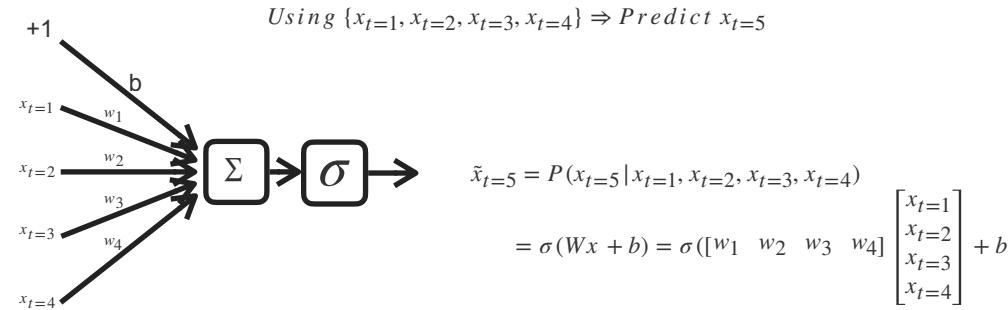




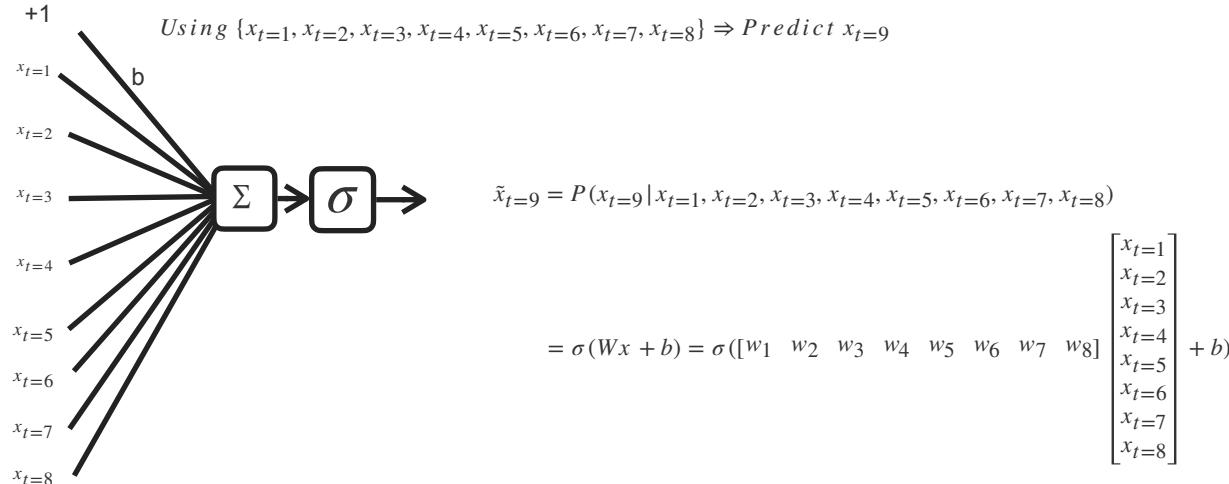
# Revisiting MLPs for Sequential Data

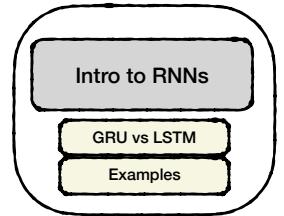
## Issue 2: Fixed number of time-steps

Let's assume we have the following model which tries to predict whether a hi-hat should be played given a sequence of hi-hats previously played.



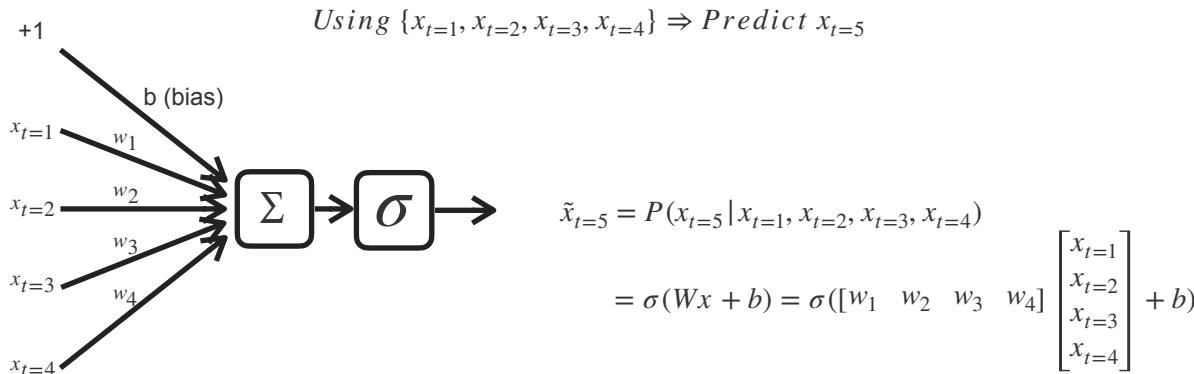
If we want to base our prediction on a longer sequencer, we need to re-structure the model (and hence re-train) as follows:



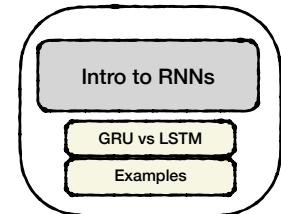


# Revisiting MLPs for Sequential Data

## Issue 2: Varying input length



Moreover, how would we modify the above architecture to accommodate inputs of varying length?

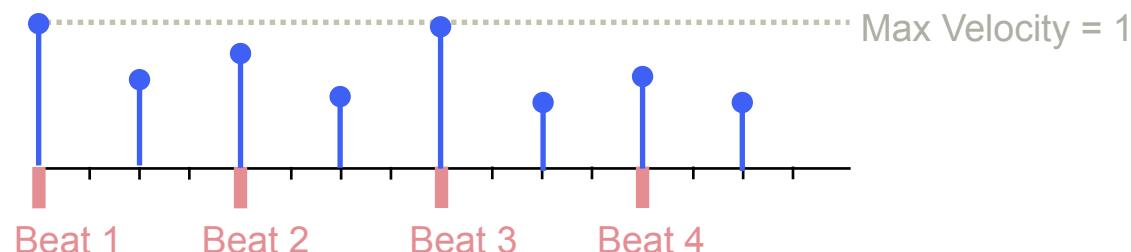


# Revisiting MLPs for Sequential Data

**Issue 3: Generalization of rules/patterns/relationships unfolding over time**

Let's say that we want to design a model such that given a sequence of time quantized onsets with varying velocities, the model tries to predict where the beats are located

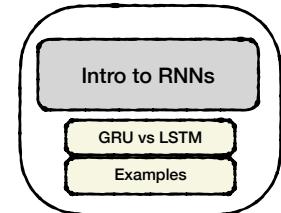
Onset Velocities = [ 1, 0, 0.7, 0, 0.8, 0, 0.5, 0, 1, 0, 0.5, 0, 0.75, 0, 0.5, 0 ]



Beat Positions = [ 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0 ]



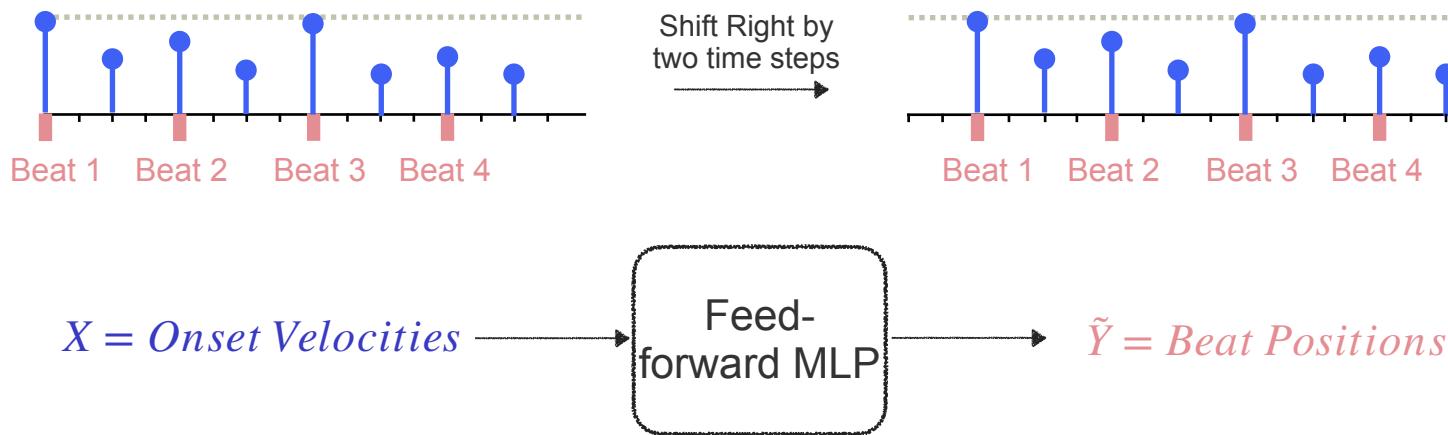
A fully connected feed-forward network would have separate parameters for all input features across time, from which it tries to come up with “rules” that best describe all the possible (or rather seen) examples in the input.



# Revisiting MLPs for Sequential Data

**Issue 3: Generalization of rules/patterns/relationships unfolding over time**

Now what if our onset observations are slightly shifted?



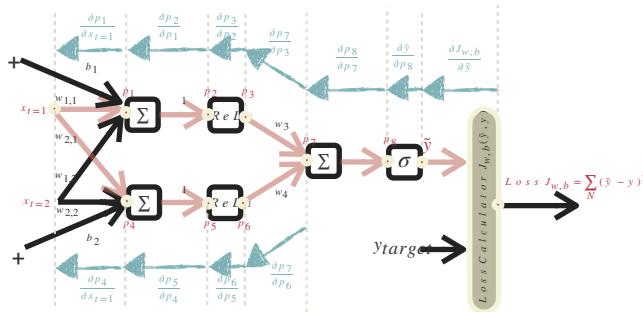
The FF-MLP network is not capable of extrapolating its knowledge of the original pattern to correctly predict the shifted version.

In order to be able to correctly predict the shifted version, the model needs to be trained on enough examples of shifted inputs!

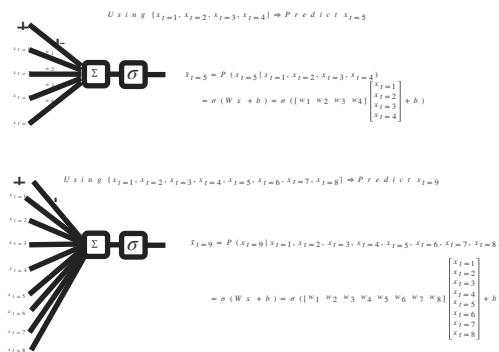
The issue here is that we need to come up with generalized rules that describe the relationships between events with respect to time difference! In other words, we need to learn about events as they unfold over time rather than treating events as points in a fixed grid!

# Revisiting MLPs for Sequential Data

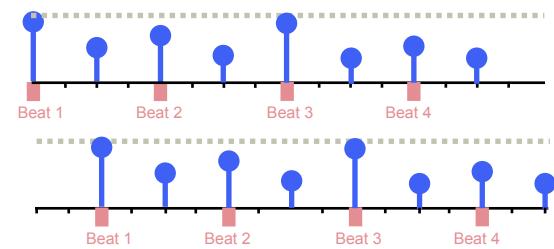
## Issues: Recap



Issue 1: Vanishing Gradients

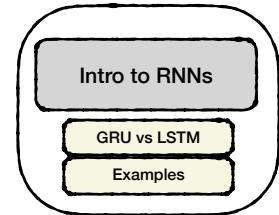


Issue 2: Varying Input Lengths



Issue 3: Generalization of rules/patterns/relationships unfolding over time

Recurrent neural nets can be used to address some of the issues with feed-forward MLPs for sequential data

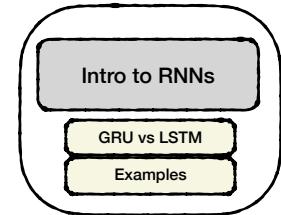


# Recurrent Neural Nets (RNNs)

## Overview

RNNs are a family of neural nets specialized for processing sequential data. These models can deal with longer sequences as well as sequences of varying length [1].

“To go from multilayer networks to recurrent networks, we need to take advantage of one of the early ideas found in machine learning and statistical models of the 1980s: sharing parameters across different parts of a model. **Parameter sharing** makes it possible to **extend and apply** the model to examples **of different forms** (different lengths, here) and **generalize across them**. **If we had separate parameters for each value of the time index, we could not generalize to sequence lengths not seen during training, nor share statistical strength across different sequence lengths and across different positions in time.** **Such sharing is particularly important when a specific piece of information can occur at multiple positions within the sequence.**” (p.386 in [1])



# Recurrent Neural Nets (RNNs)

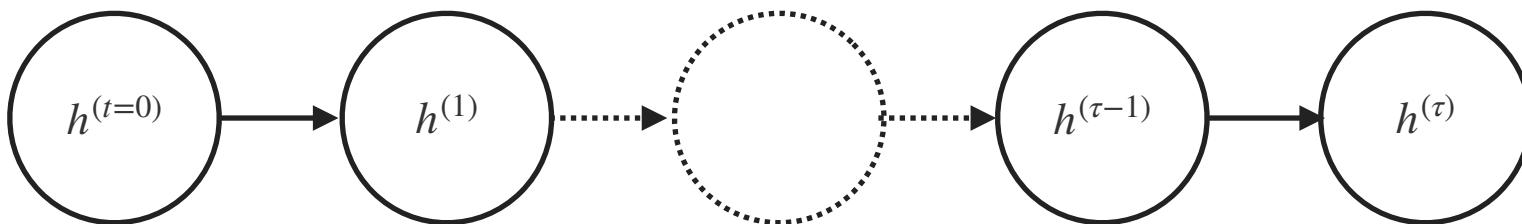
What does recurrence mean?

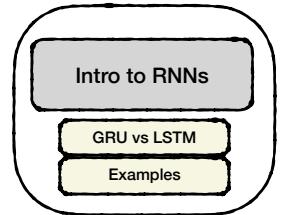
Recurrence means that the state of a model depends on its previous states:

$$h^{(t)} = f(h^{(t-1)}, \theta_{w,b})$$

Where  $h^{(t=\tau)}$  denotes the state of the model at time  $\tau$ , and  $\theta_{w,b}$  denotes the weights and biases in the model which are not time dependant (i.e. shared between different times).

In other words, we use a new parameter ( $h$ ) to keep track of the history of a model's previous states before making a prediction at the present time step  $\tau$ .





# Recurrent Neural Nets (RNNs)

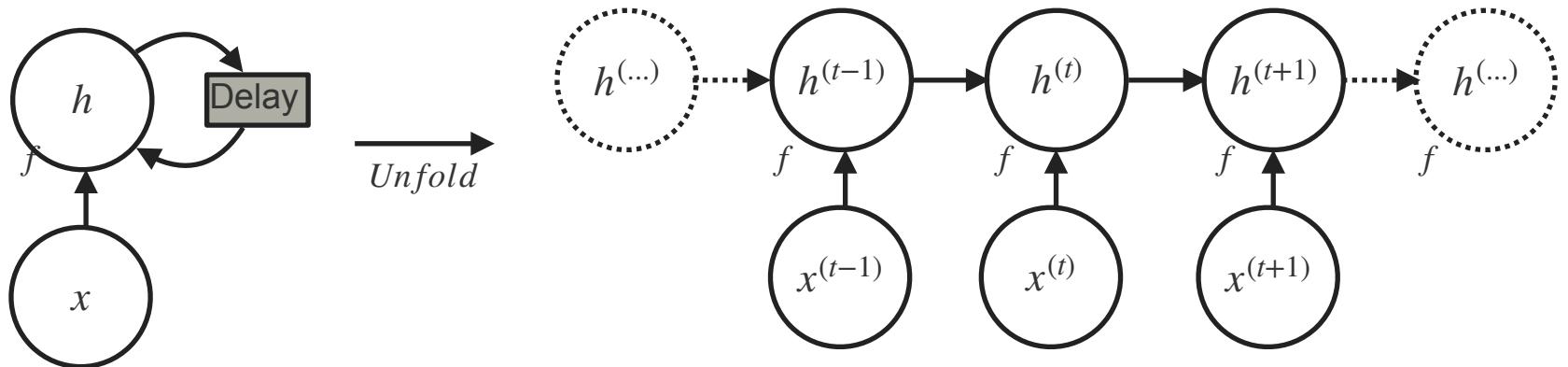
How can we make the recurrence to be also dependant on past inputs?

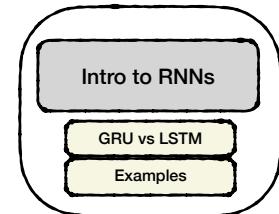
In reality, the state vector should also keep track of the whole past sequence:

$$h^{(t)} = g^{(t)}(x^{(t)}, x^{(t-1)}, x^{(t-2)}, \dots, x^{(2)}, x^{(1)}) = f(h^{(t-1)}, x^{(t)}, \theta_{w,b}) \quad (\text{page 367 of [1]})$$

Constant over time

Changes over time





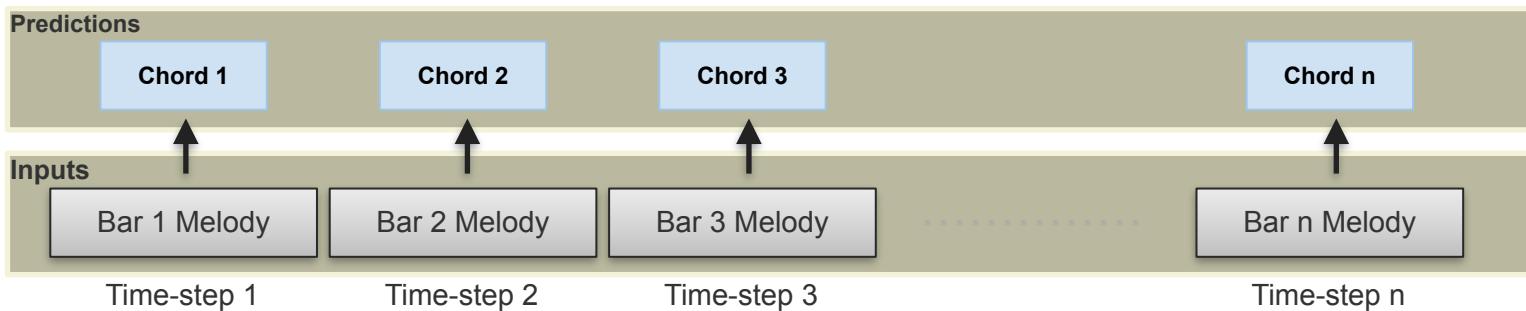
# Recurrent Neural Nets (RNNs)

## How do we make predictions over time?

There are two types of predictions that we typically do using RNNs

### 1. Predicting an output at every time step

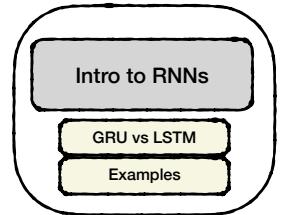
Example: Generate a chord at the beginning of each bar based on a sequence of melodies provided



### 2. Predicting an output at the very end of a sequence

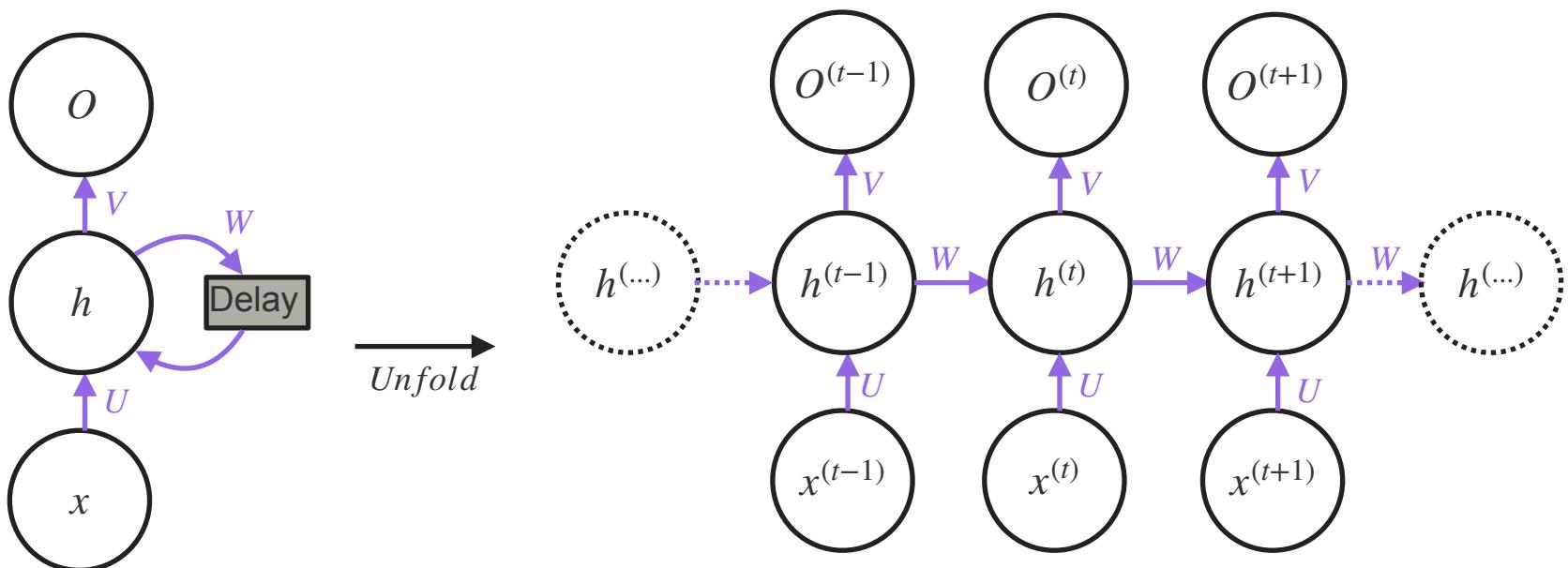
Example: Find the most probable note following a given melody





# Recurrent Neural Nets (RNNs)

Predicting an output at every time step (p. 369 in [1])

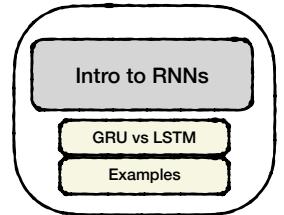


$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$

$$h^{(t)} = \tanh(a^{(t)})$$

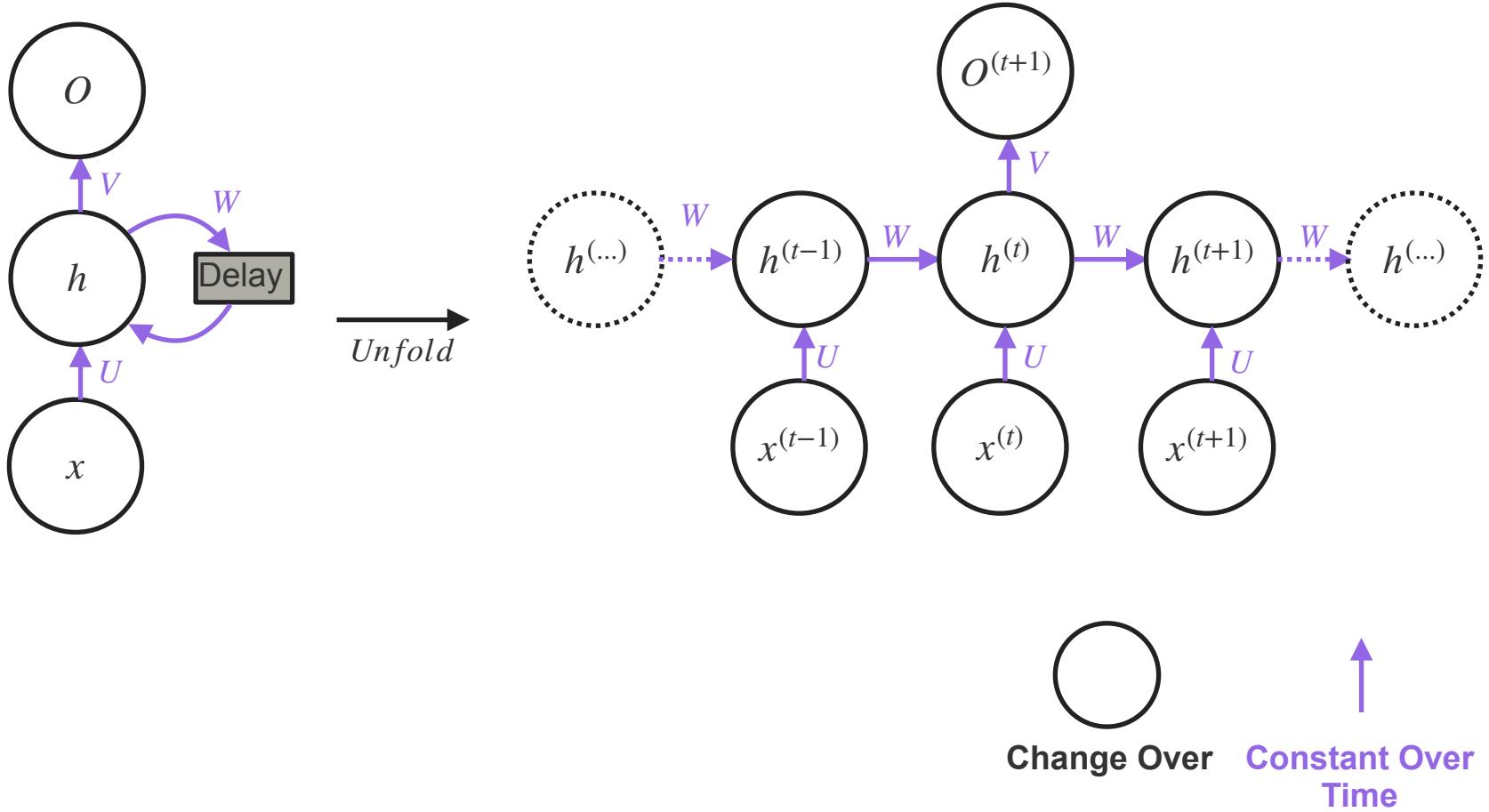
$$o^{(t)} = c + Vh^{(t)}$$

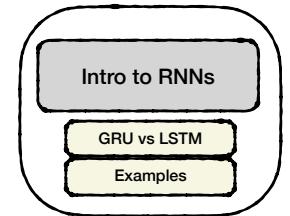
 Change Over Time  
 Constant Over Time



# Recurrent Neural Nets (RNNs)

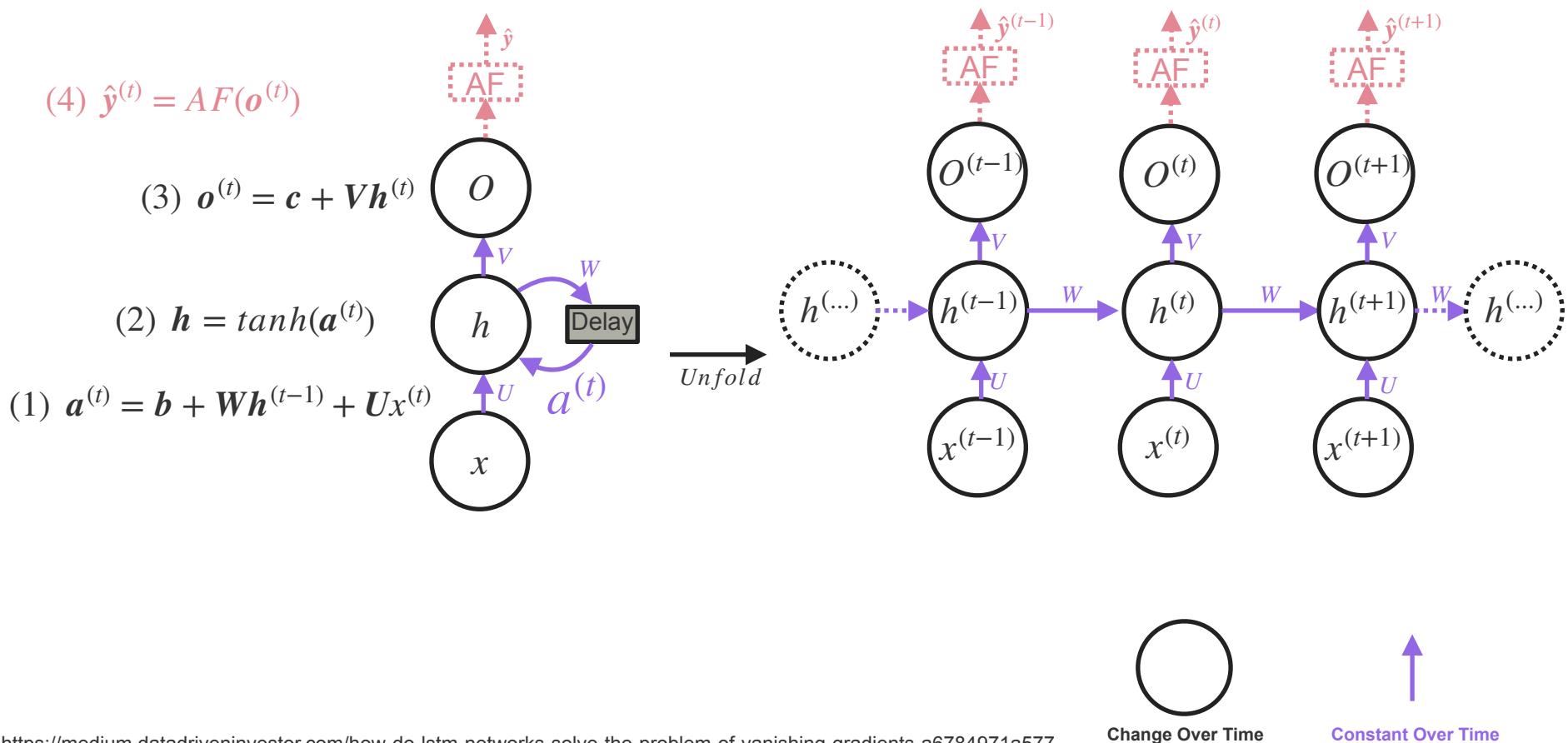
Predicting an output at the very end of a sequence (p. 371 in [1])

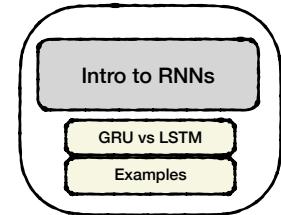




# Recurrent Neural Nets (RNNs)

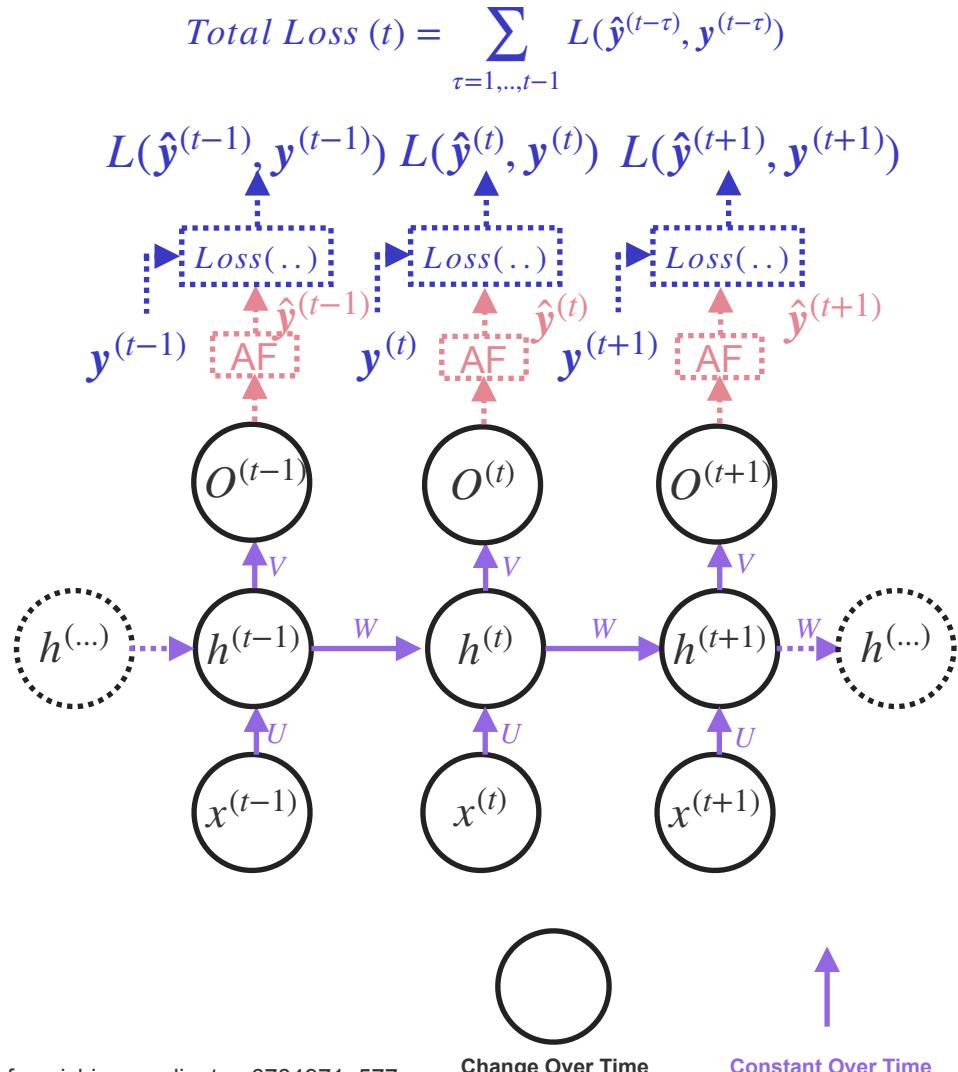
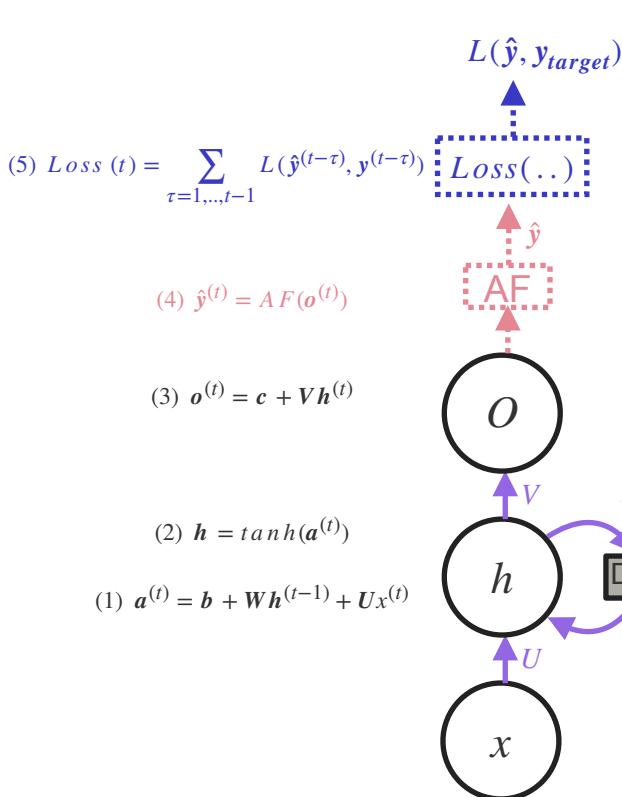
**Step 1: Predict output  $\hat{y}$  at required time-steps using the output layer  $O$**

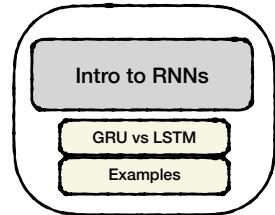




# Recurrent Neural Nets (RNNs)

## Step 2: Loss Calculation

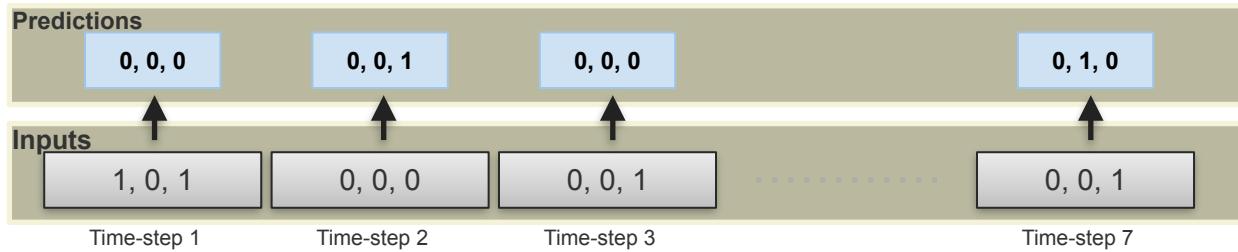




# Recurrent Neural Nets (RNNs)

**Example: Drum Pattern Continuation**

**Objective:** Predict Next Time-Step Given Past Sequence



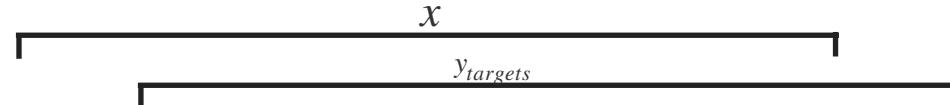
<i>Hat</i>	1	0	1	0	1	0	1	0
<i>Snare</i>	0	0	0	0	1	0	0	1
<i>Kick</i>	1	0	0	0	1	0	0	0
	<i>t</i> = 1			<i>t</i> = 4			<i>t</i> = 8	

First Iteration: Given  $x_1$  predict  $x_2 \rightarrow P(y_1 = x_2 | x_1)$

Second Iteration: Given  $x_1$  and  $x_2$  predict  $x_3 \rightarrow P(y_2 = x_3 | x_1, x_2)$

Third Iteration: Given  $x_1, x_2, x_3$  predict  $x_4 \rightarrow P(y_3 = x_4 | x_1, x_2, x_3)$

...

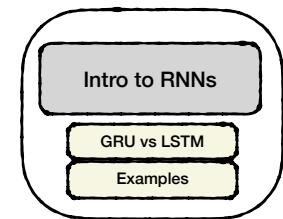


```
sequence = torch.tensor([[[1, 0, 1], [0, 0, 0], [0, 0, 1], [0, 0, 0], [1, 1, 1], [0, 0, 0], [0, 0, 1], [0, 1, 0]]],  
dtype=torch.float32) # Shape (batch_size, time steps, feature per step) -> (1, 8, 3)
```

```
x = sequence[:, -1, :] # the first dimension is batch index (in our case batch size is 1)  
y_targets = sequence[:, 1:, :] # the first dimension is batch index (in our case batch size is 1)
```

# Recurrent Neural Nets (RNNs)

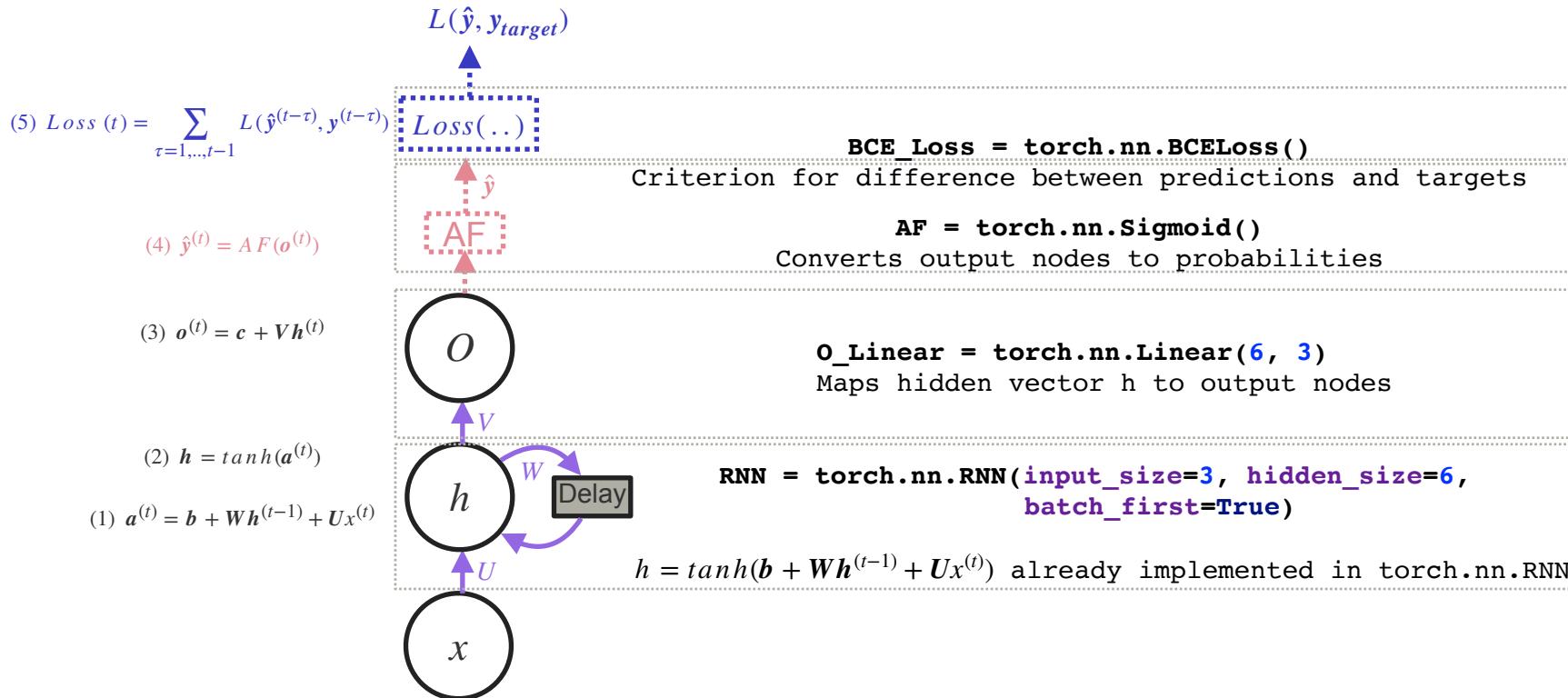
Hat	1	0	1	0	1	0	1	0
Snare	0	0	0	0	1	0	0	1
Kick	1	0	0	0	1	0	0	0
	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	$x_8$



**Example: Drum Pattern Continuation**

**Objective:** Predict Next Time-Step Given Past Sequence

## Setup Network

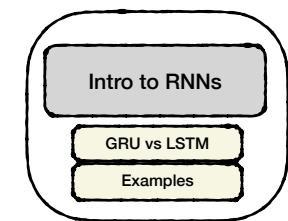


# Recurrent Neural Nets (RNNs)

**Example: Drum Pattern Continuation**

**Objective:** Predict Next Time-Step Given Past Sequence

Hat	1	0	1	0	1	0	1	0
Snare	0	0	0	0	1	0	0	1
Kick	1	0	0	0	1	0	0	0
	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	$x_8$



Time-Step 1



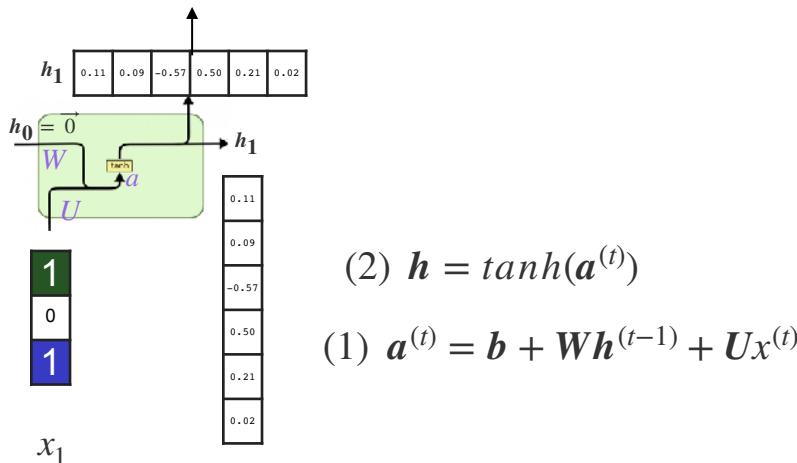
First Iteration: Given  $x_1$  predict  $x_2 \rightarrow P(y_1 = x_2 | x_1)$

```

h1:
  tensor([[[ 0.1136,  0.0974, -0.5761,  0.5002,  0.2093,  0.0224]]])

_, h1 = RNN(x1, h0)

```

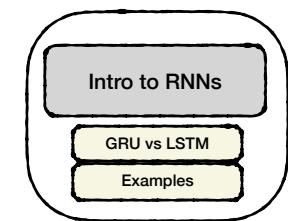


# Recurrent Neural Nets (RNNs)

**Example: Drum Pattern Continuation**

**Objective:** Predict Next Time-Step Given Past Sequence

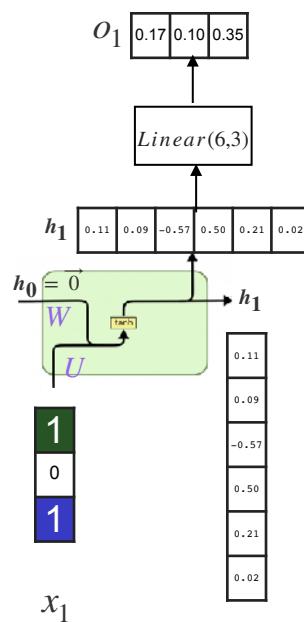
Hat	1	0	1	0	1	0	1	0
Snare	0	0	0	0	1	0	0	1
Kick	1	0	0	0	1	0	0	0
	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	$x_8$



Time-Step 1



First Iteration: Given  $x_1$  predict  $x_2 \rightarrow P(y_1 = x_2 | x_1)$



**o1:**

```
tensor([[[0.1742, 0.1092, 0.3523]]])
```

**o1 = O\_Linear(h1)**

```
_, h1 = RNN(x1, h0)
h1:
tensor([[[ 0.1136, 0.0974, -0.5761, 0.5002, 0.2093, 0.0224]]])
```

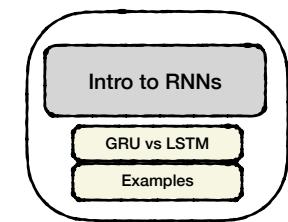
# Recurrent Neural Nets (RNNs)

**Example: Drum Pattern Continuation**

**Objective:** Predict Next Time-Step Given Past Sequence

Hat	1	0	1	0	1	0	1	0
Snare	0	0	0	0	1	0	0	1
Kick	1	0	0	0	1	0	0	0

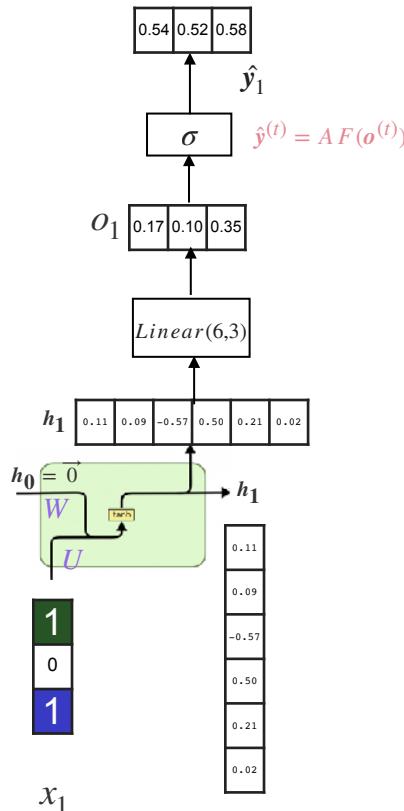
$x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8$



Time-Step 1



First Iteration: Given  $x_1$  predict  $x_2 \rightarrow P(y_1 = x_2 | x_1)$



**y\_predict1:**

```
tensor([[0.5434, 0.5273, 0.5872]]))
```

**y\_predict1 = AF(o1)**

```
o1 = O_Linear(h1)
o1
tensor([[0.1742, 0.1092, 0.3523]])
```

$\text{_, } h1 = \text{RNN}(x1, h0)$

```
h1:
tensor([[0.1136, 0.0974, -0.5761, 0.5002, 0.2093, 0.0224]]))
```

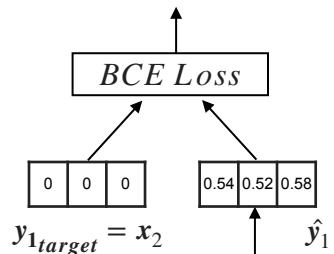
# Recurrent Neural Nets (RNNs)

**Example: Drum Pattern Continuation**

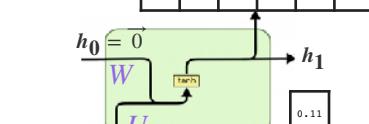
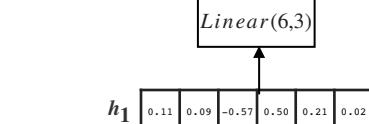
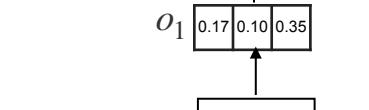
**Objective:** Predict Next Time-Step Given Past Sequence

Hat	1	0	1	0	1	0	1	1	0
Snare	0	0	0	0	1	0	0	1	
Kick	1	0	0	0	1	0	0	0	

$$L^{(1)} = 0.8060 = P(y = x_2 | x_1)$$



$$y_{1\text{target}} = x_2$$



```
loss = BCE_Loss(y_predict1, y1_target)
losses.append(loss)
```

```
loss
tensor(0.8060)

losses
[0.8059980273246765]
```

```
y_predict1 = AF(o1)
y_predict1
tensor([[0.5434, 0.5273, 0.5872]])
```

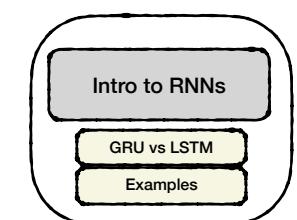
```
o1 = O_Linear(h1)
o1
tensor([[0.1742, 0.1092, 0.3523]])
```

```
_, h1 = RNN(x1, h0)
h1:
tensor([[[ 0.1136, 0.0974, -0.5761, 0.5002, 0.2093, 0.0224]]])
```

Time-Step 1



First Iteration: Given  $x_1$  predict  $x_2 \rightarrow P(y_1 = x_2 | x_1)$

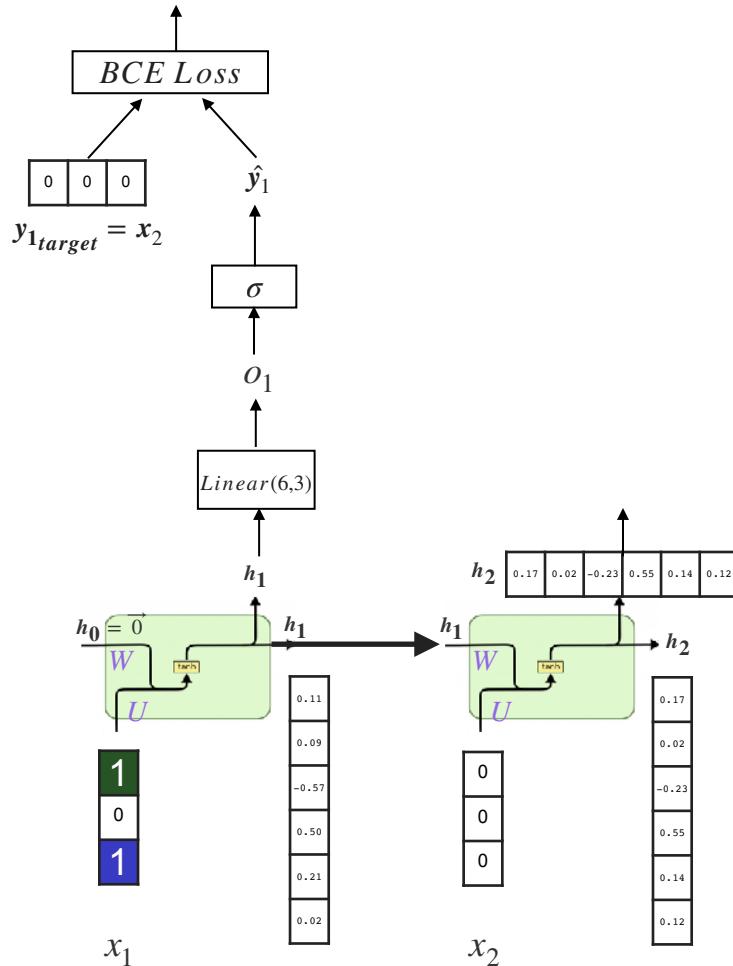


# Recurrent Neural Nets (RNNs)

**Example: Drum Pattern Continuation**

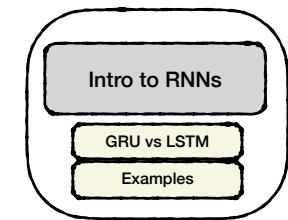
**Objective:** Predict Next Time-Step Given Past Sequence

$$L^{(1)} = 0.8060 = P(y = x_2 | x_1)$$



Hat	1	0	1	0	1	0	1	0
Snare	0	0	0	0	1	0	0	1
Kick	1	0	0	0	1	0	0	0

$x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8$



**Time-Step 2**

Second Iteration: Given  $x_1$  and  $x_2$  predict  $x_3 \rightarrow P(y_2 = x_3 | x_1, x_2)$

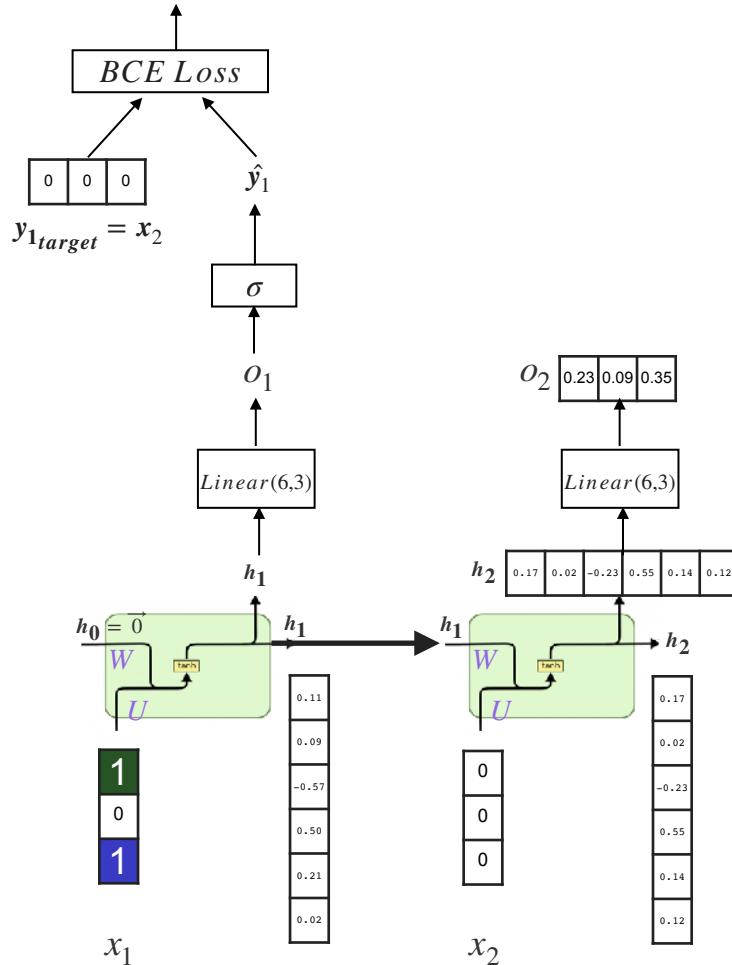
```
_ , h2 = RNN(x2, h1)
h2:
tensor([[ 0.1749,   0.0217,  -0.2394,   0.5537,   0.1435,   0.1275]])
```

# Recurrent Neural Nets (RNNs)

**Example: Drum Pattern Continuation**

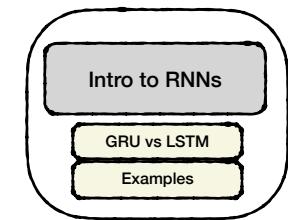
**Objective:** Predict Next Time-Step Given Past Sequence

$$L^{(1)} = 0.8060 = P(y = x_2 | x_1)$$



Hat	1	0	1	0	1	0	1	0
Snare	0	0	0	0	1	0	0	1
Kick	1	0	0	0	1	0	0	0

$x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8$



Time-Step 2  
↓

Second Iteration: Given  $x_1$  and  $x_2$  predict  $x_3 \rightarrow P(y_2 = x_3 | x_1, x_2)$

```

02 = O_Linear(h2)
02
tensor([[0.2361, 0.0946, 0.3567]])

_, h2 = RNN(x2, h1)
h2:
tensor([[0.1749, 0.0217, -0.2394, 0.5537, 0.1435, 0.1275]])

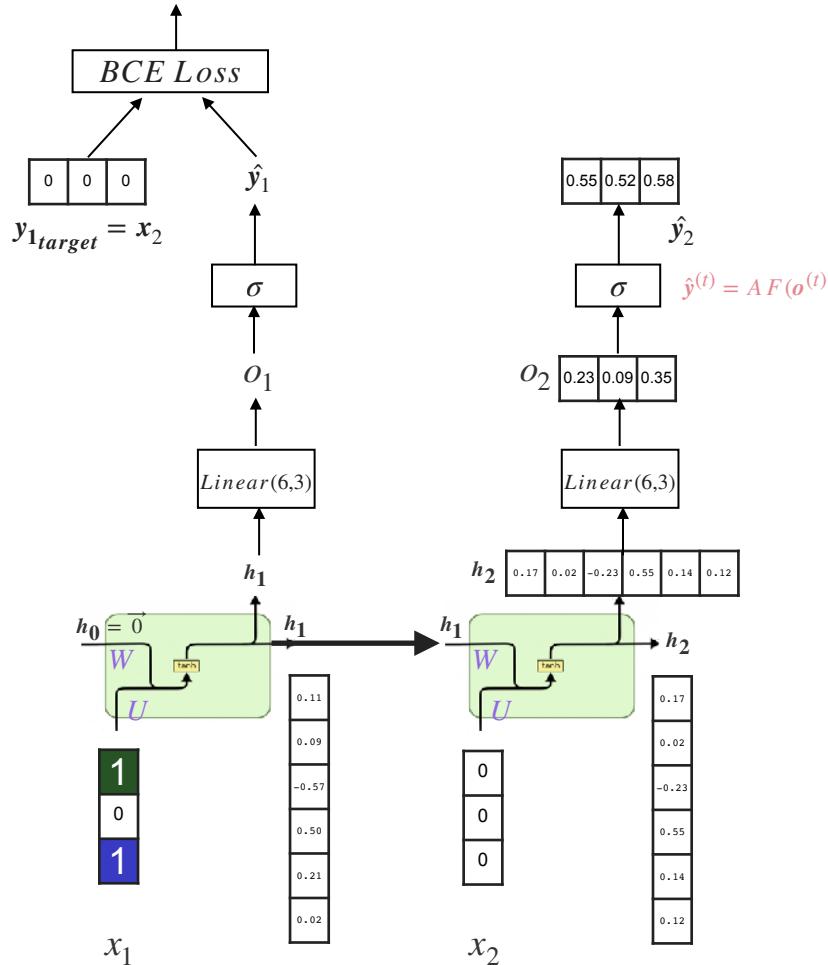
```

# Recurrent Neural Nets (RNNs)

**Example: Drum Pattern Continuation**

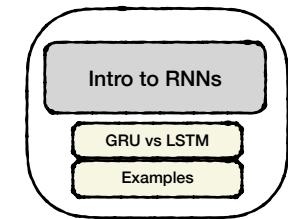
**Objective:** Predict Next Time-Step Given Past Sequence

$$L^{(1)} = 0.8060 = P(y = x_2 | x_1)$$



Hat	1	0	1	0	1	0	1	0
Snare	0	0	0	0	1	0	0	1
Kick	1	0	0	0	1	0	0	0

$x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8$



Time-Step 2  
↓

Second Iteration: Given  $x_1$  and  $x_2$  predict  $x_3 \rightarrow P(y_2 = x_3 | x_1, x_2)$

```

y_predict2 = AF(o2)
y_predict2
tensor([[0.5587, 0.5236, 0.5882]])

o2 = o_Linear(h2)
o2
tensor([[0.2361, 0.0946, 0.3567]])

_, h2 = RNN(x2, h1)
h2:
tensor([[0.1749, 0.0217, -0.2394, 0.5537, 0.1435, 0.1275]])

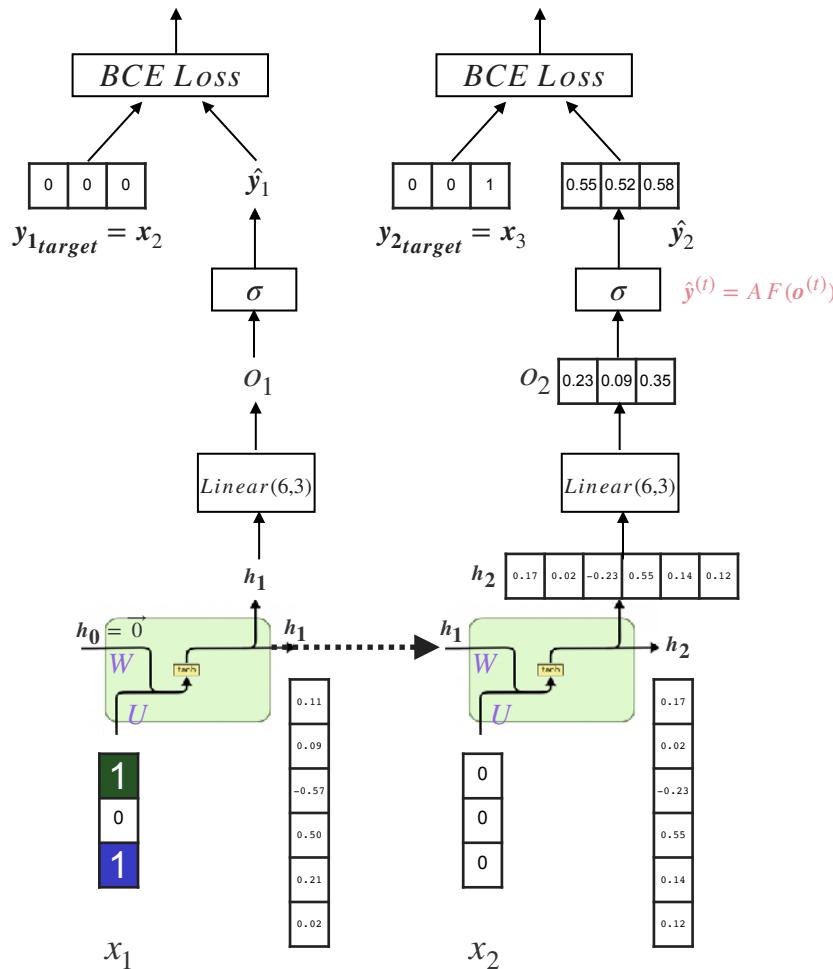
```

# Recurrent Neural Nets (RNNs)

**Example: Drum Pattern Continuation**

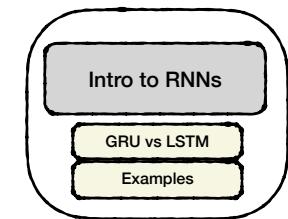
**Objective:** Predict Next Time-Step Given Past Sequence

$$L^{(1)} = 0.8060 = P(y = x_2 | x_1) \quad L^{(2)} = 0.6968 = P(y_2 = x_3 | x_1, x_2)$$



Hat	1	0	1	0	1	0	1	0
Snare	0	0	0	0	1	0	0	1
Kick	1	0	0	0	1	0	0	0

$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8$



Time-Step 2  
↓

Second Iteration: Given  $x_1$  and  $x_2$  predict  $x_3 \rightarrow P(y_2 = x_3 | x_1, x_2)$

```

loss = BCE_Loss(y_predict2, y2_target)
losses.append(loss)

loss
tensor(0.6968)

losses
[0.8059980273246765, 0.6967602372169495]

y_predict2 = AF(o2)
y_predict2
tensor([[0.5587, 0.5236, 0.5882]])

o2 = O_Linear(h2)
o2
tensor([[0.2361, 0.0946, 0.3567]])

_, h2 = RNN(x2, h1)
h2:
tensor([[0.1749, 0.0217, -0.2394, 0.5537, 0.1435, 0.1275]])

```

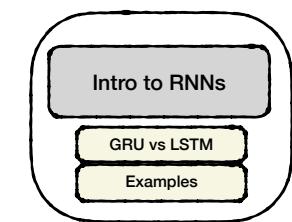
# Recurrent Neural Nets (RNNs)

**Example: Drum Pattern Continuation**

**Objective:** Predict Next Time-Step Given Past Sequence

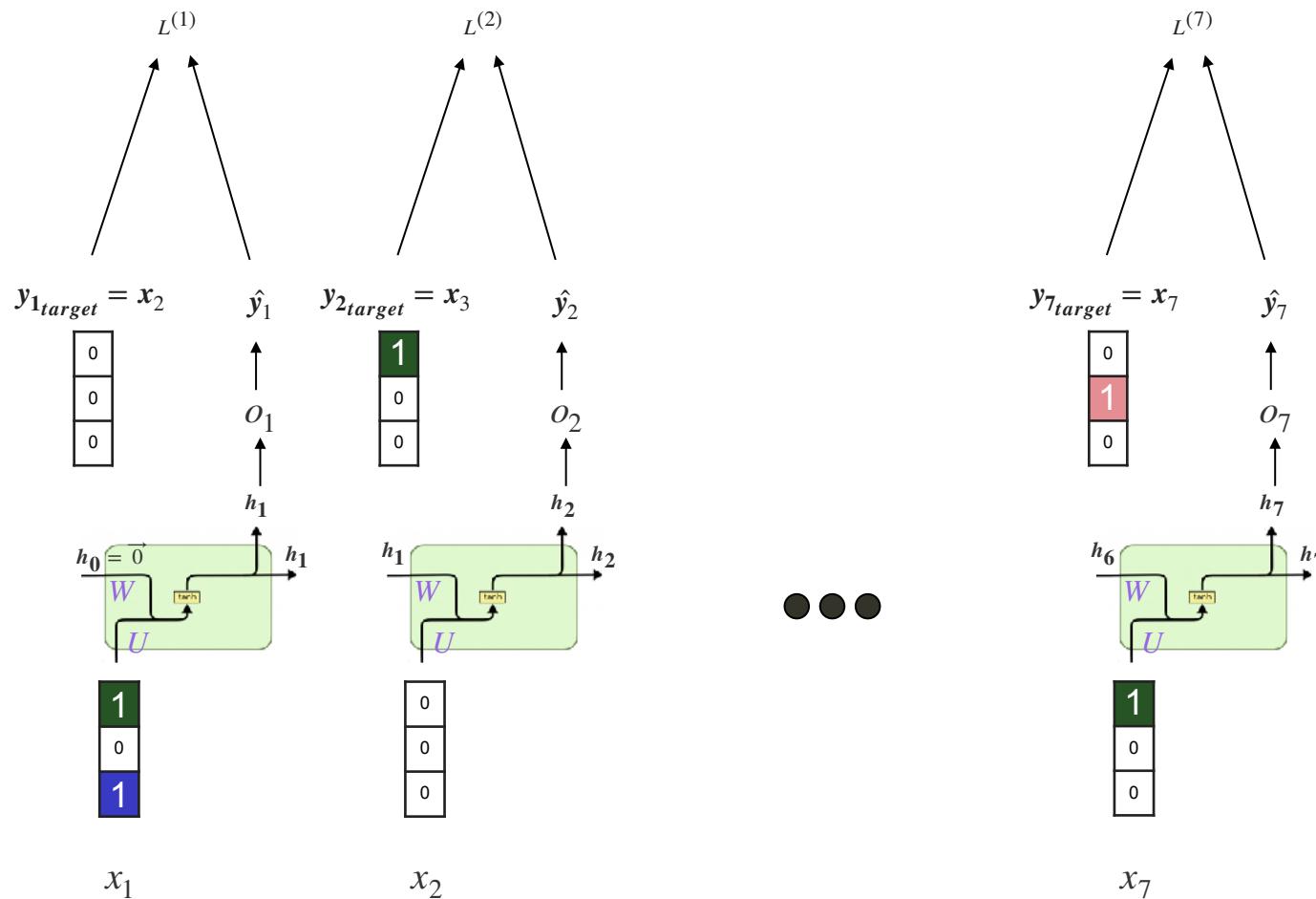
Hat	1	0	1	0	1	0	1	1	0
Snare	0	0	0	0	0	1	0	0	1
Kick	1	0	0	0	0	1	0	0	0

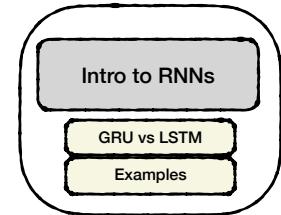
$x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8$



After 7 Iterations

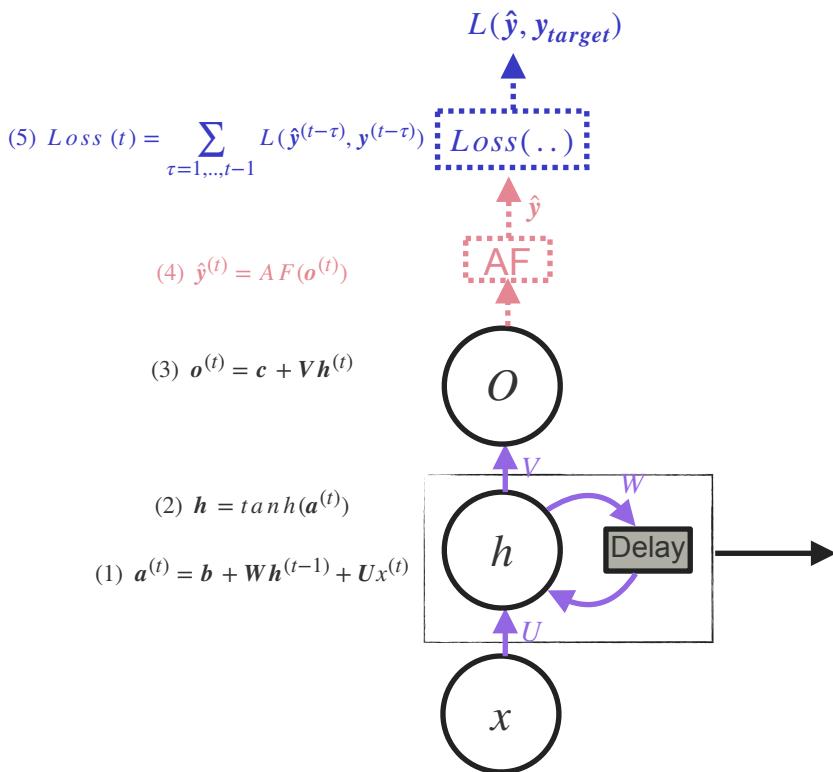
$$\text{Total Loss} = L^{(1)} + L^{(2)} + \dots + L^{(7)}$$





# Recurrent Neural Nets (RNNs)

A Major Issue of Vanilla RNNs: Vanishing or Exploding Gradients

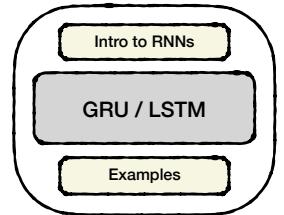


Repeatedly back-propagating over time through this part results in the gradients either **vanishing** to zero or **exploding** to infinite

This makes the architecture quite problematic for dealing with longer sequences.

We don't cover the proof here, but you can find it on pages 374-375 of [1], and in [3]

We can modify this section to improve on these shortcomings a bit. LSTMs and GRUs are some of the most common replacements for this section



# GRU (Gated Recurrent Unit)

## Overview

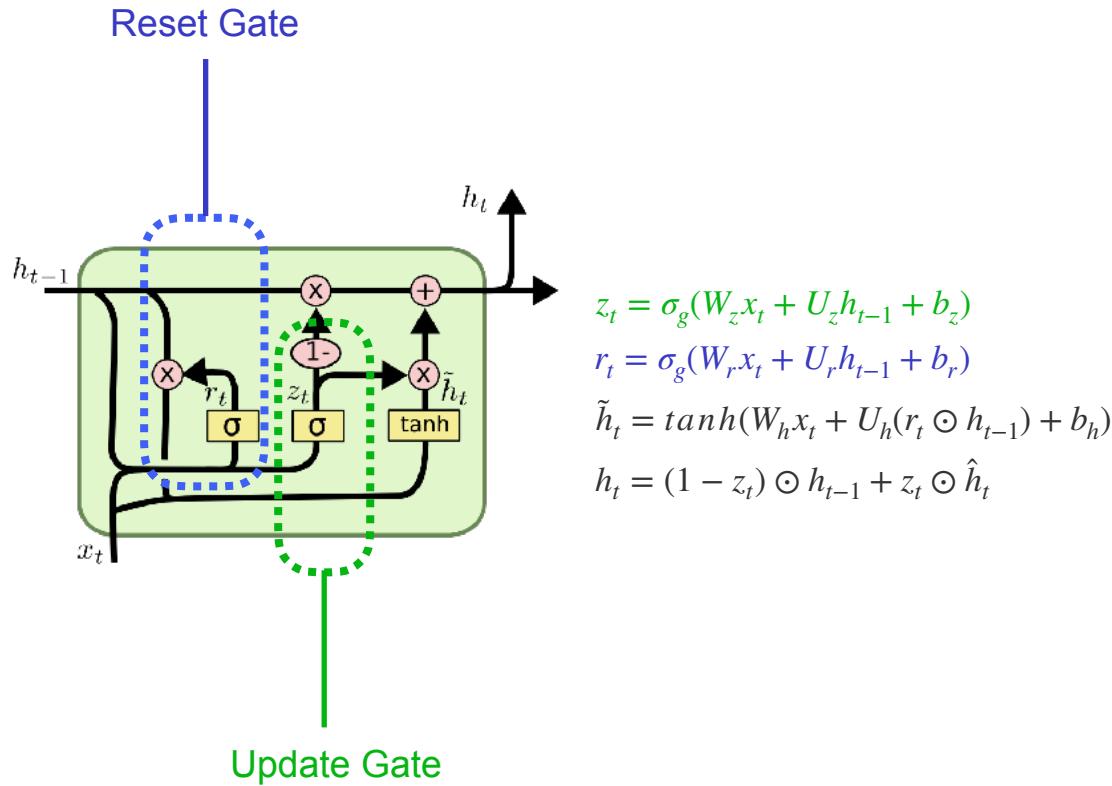
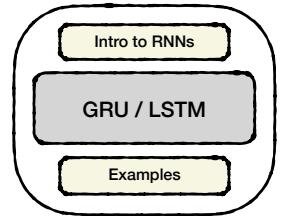


Figure from: [4] <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

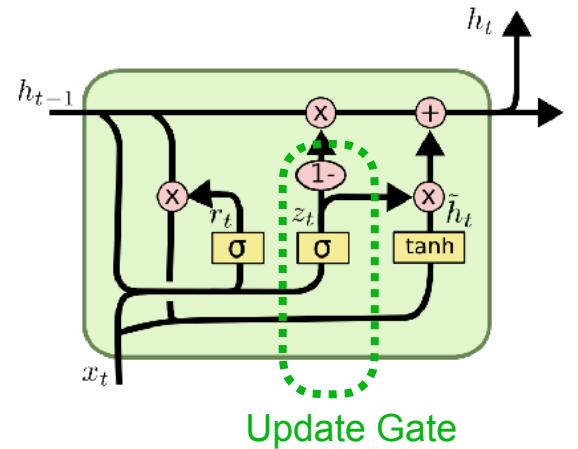


# GRU (Gated Recurrent Unit)

How does the update gate work?

Update Gate is in charge of deciding how much of the information accumulated from past along with how much of information extracted from the current input should carry on to the next time-step!

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t$$

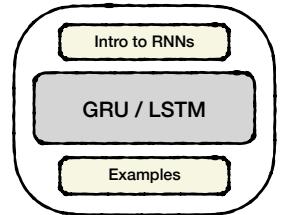


$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$

$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$

$$\tilde{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t$$



# GRU (Gated Recurrent Unit)

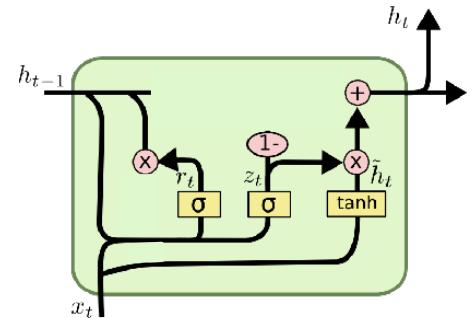
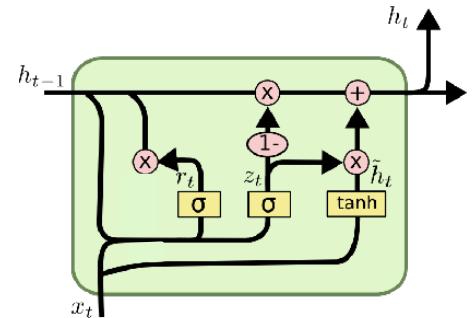
How does the reset gate work?

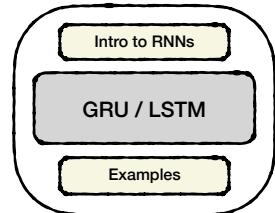
If the Reset Gate is fully active ( $r_t = 0$ ), the new candidate for hidden state ( $\tilde{h}_t$ ) will only depend on the existing input; just like the first iteration in the vanilla RNN case

$$\tilde{h}_t = \tanh(W_h x_t + U_h (\vec{0} \odot h_{t-1}) + b_h) = \tanh(W_h x_t + b_h)$$

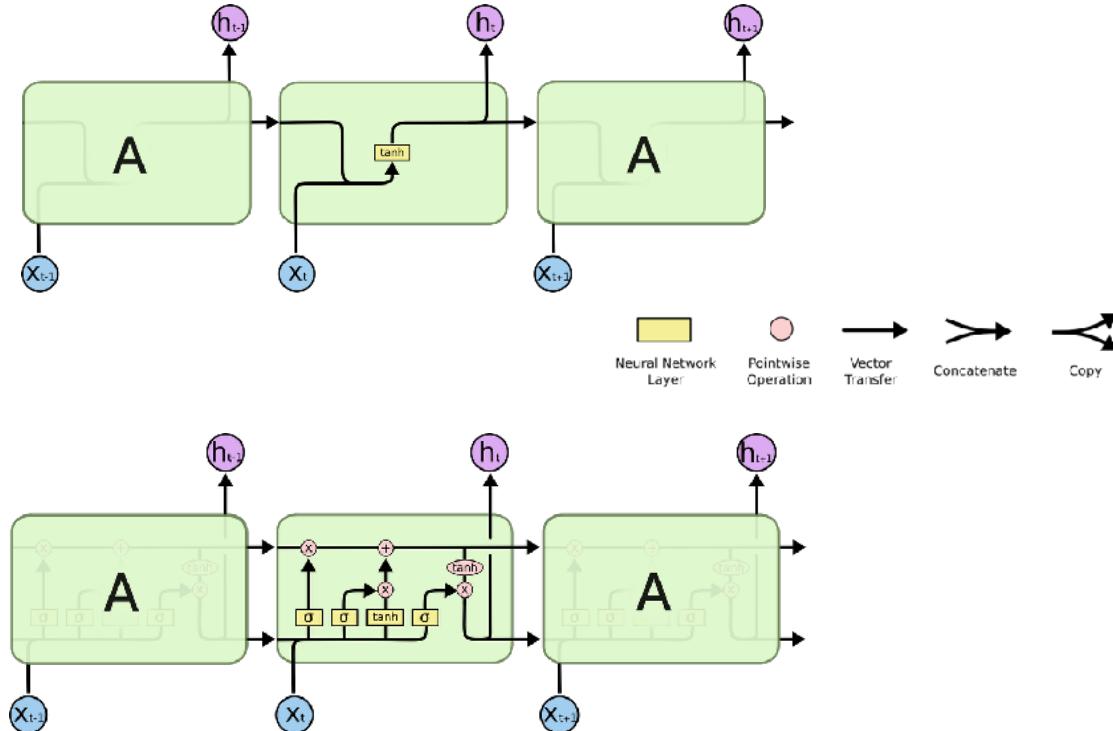
Now, if the update gate decides that the hidden state ( $h_t$ ) should be replaced with the candidate ( $\tilde{h}_t$ ) (i.e.  $z_t = 1$ ), the new hidden state will carry no information from the past sequence!

$$h_t = (1 - z_t) \odot h_{t-1} + (1 - 1) \odot \tilde{h}_t = \hat{h}_t$$



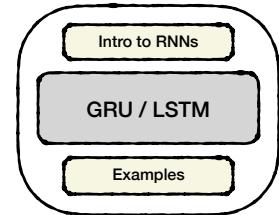


# LSTMs (Long Short-Term Memory)



Figures from <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>, for a detailed explanation refer to this source

LSTMs have similar mechanisms as GRUs (with some additional operations) so as to mediate the propagation of information from past sequence, current time-step to future time-steps.



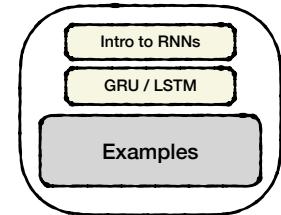
# GRU vs. LSTM

**GRUs and LSTMs both:**

1. Allow for processing/prediction of longer sequences
2. Mediate the propagation of information from past sequence, current time-step to future time-steps.
3. Significantly make improvements in solving the vanishing/exploding gradient issues

**Which one should we use in a recurrent network?**

1. GRUs train faster and need less training data
2. LSTMs can deal with longer sequences
3. Given the amount of data/computational resources you have available, ideally you should test both to see which one works better for the target task



# Examples

## Performance RNN

Ian Simon and Sageev Oore. "Performance RNN: Generating Music with Expressive Timing and Dynamics." Magenta Blog, 2017.

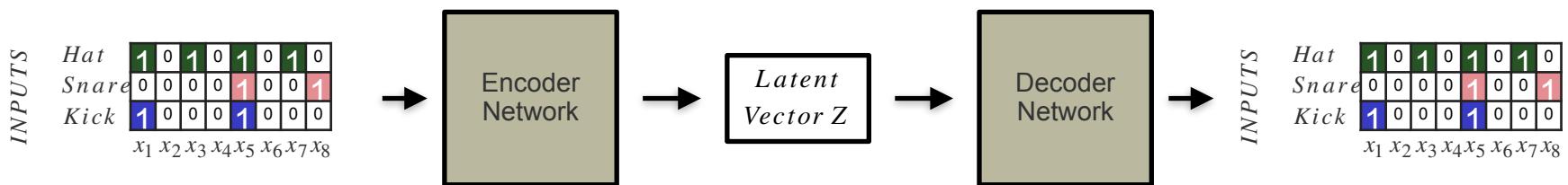
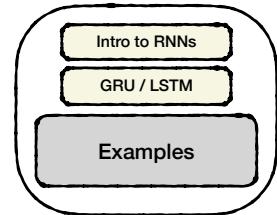
<https://magenta.tensorflow.org/performance-rnn>

## LSTMetallica

Choi, Keunwoo, George Fazekas, and Mark Sandler. "Text-based LSTM networks for automatic music composition." *arXiv preprint arXiv:1604.05358* (2016).

<https://keunwoochoi.wordpress.com/2016/02/23/lstmmetallica/>

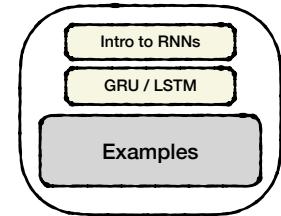
# Auto-encoders for sequential data



We want to encode an input **sequence** into a vector  $z$  and, from which we decode/reconstruct the original input **sequence**



We can use **Sequence-to-Sequence (Seq2Seq)** architectures for this purpose



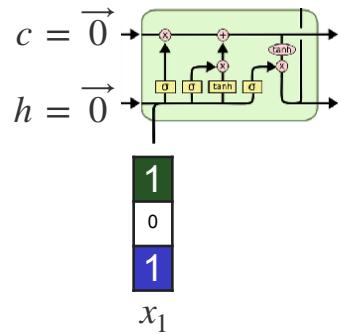
## Sequence to Sequence Architecture (Auto-encoder for sequential data)

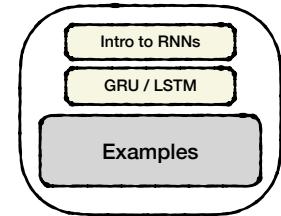
### Step 1: Encoding the input

INPUTS

<i>Hat</i>	1	0	1	0	1	0	1	0
<i>Snare</i>	0	0	0	0	1	0	0	1
<i>Kick</i>	1	0	0	0	1	0	0	0

$x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8$





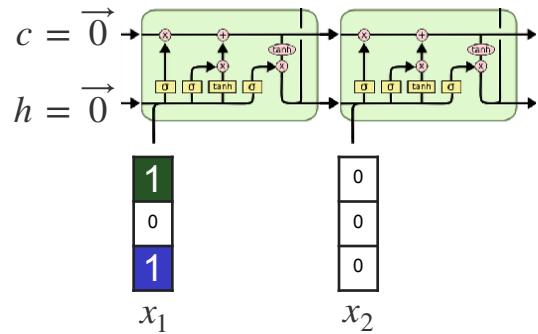
## Sequence to Sequence Architecture (Auto-encoder for sequential data)

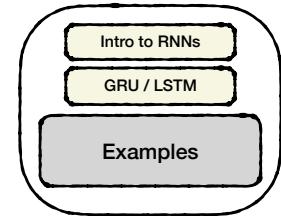
### Step 1: Encoding the input

INPUTS

<i>Hat</i>	1	0	1	0	1	0	1	0
<i>Snare</i>	0	0	0	0	1	0	0	1
<i>Kick</i>	1	0	0	0	1	0	0	0

$x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8$





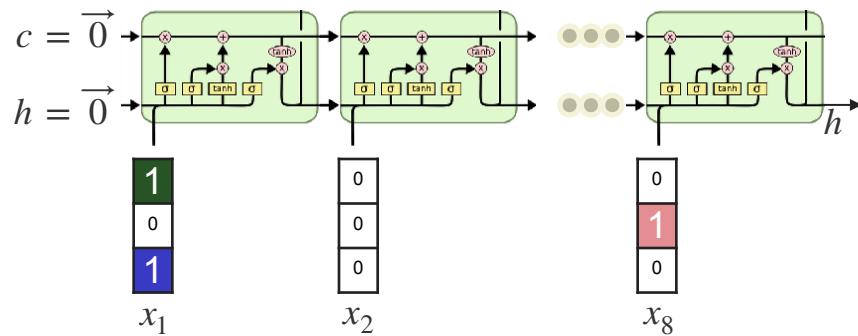
## Sequence to Sequence Architecture (Auto-encoder for sequential data)

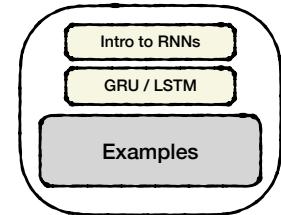
### Step 1: Encoding the input

*INPUTS*

<i>Hat</i>	1	0	1	0	1	0	1	0
<i>Snare</i>	0	0	0	0	1	0	0	1
<i>Kick</i>	1	0	0	0	1	0	0	0

$x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8$





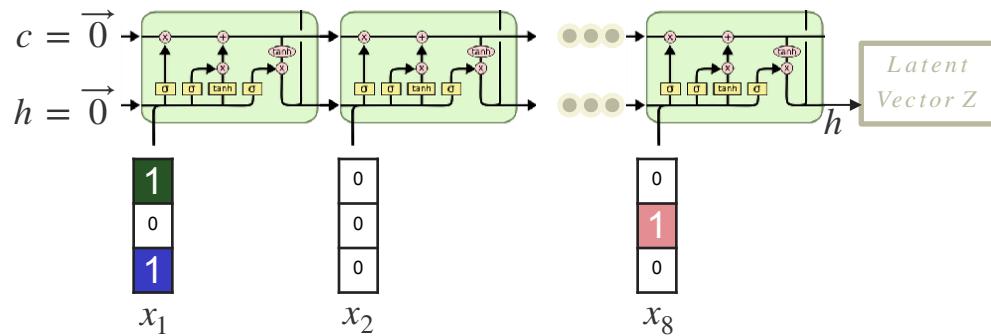
## Sequence to Sequence Architecture (Auto-encoder for sequential data)

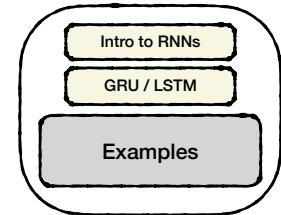
Step 2: Map last hidden vector to a latent vector Z

INPUTS

<i>Hat</i>	1	0	1	0	1	0	1	0
<i>Snare</i>	0	0	0	0	1	0	0	1
<i>Kick</i>	1	0	0	0	1	0	0	0

$x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8$





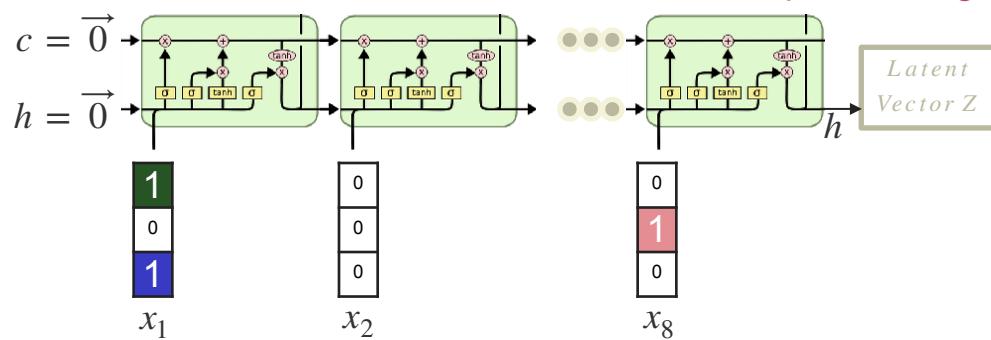
## Sequence to Sequence Architecture (Auto-encoder for sequential data)

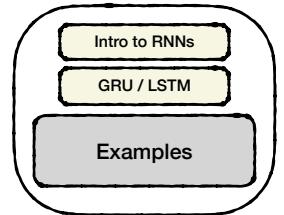
**Step 3: Decode the Output (reconstruct the input using Z)**

INPUTS																									
Hat	<table border="1"><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr></table>	1	0	1	0	1	0	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	0	0	0
1	0	1	0	1	0	1	0																		
0	0	0	0	1	0	0	1																		
1	0	0	0	1	0	0	0																		
Snare	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	1	1	0	0	0	1	0	0	0
0	0	0	0	1	0	0	1																		
0	0	0	0	1	0	0	1																		
1	0	0	0	1	0	0	0																		
Kick	<table border="1"><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr></table>	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0
1	0	0	0	1	0	0	0																		
0	0	0	0	1	0	0	0																		
0	0	0	0	0	1	0	0																		
	$x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8$																								

OUTPUTS																												
Hat	<table border="1"><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	1	0	1	0	1	0	1	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	1	0	0	0	0
1	0	1	0	1	0	1	0	0																				
0	0	0	0	1	0	0	1	0																				
1	0	0	0	1	0	0	0	0																				
Snare	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	1	0	1	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	1	0																				
0	0	0	0	1	0	0	1	0																				
1	0	0	0	1	0	0	0	0																				
Kick	<table border="1"><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0																				
0	0	0	0	0	0	0	0	1																				
0	0	0	0	0	0	0	0	0																				
<end >	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1																				
0	0	0	0	0	0	0	0	0																				
0	0	0	0	0	0	0	0	0																				
	$x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8 \ x_9$																											

Added to denote  
The end of sequence;  
only needed when  
dealing with various  
sequence lengths





## Sequence to Sequence Architecture (Auto-encoder for sequential data)

### Step 3: Decode the Output (reconstruct the input using Z)

**INPUTS**

<i>Hat</i>	1	0	1	0	1	0	1	0
<i>Snare</i>	0	0	0	0	1	0	0	1
<i>Kick</i>	1	0	0	0	1	0	0	0

$x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8$

**OUTPUTS**

<i>Hat</i>	1	0	1	0	1	0	1	0
<i>Snare</i>	0	0	0	0	1	0	0	1
<i>Kick</i>	1	0	0	0	1	0	0	0
<end>	0	0	0	0	0	0	0	1

$x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8 \ x_9$

$$L_1$$

BCE Loss

1
0
1
0

$x_1$

0.2
0.4
0.1
0.3

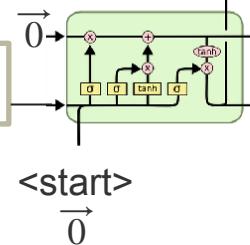
$\hat{x}_1$

$P(\hat{x}_1 = x_1 | latent\ z, <start>)$

$AF$

$O$

$h_t$

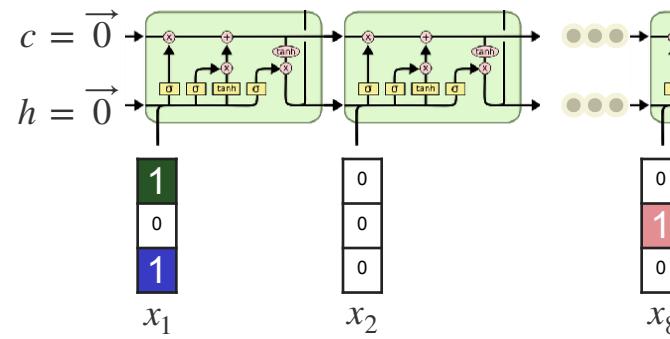


$h$

$h_t$

$<start>$

0



## Sequence to Sequence Architecture (Auto-encoder for sequential data)

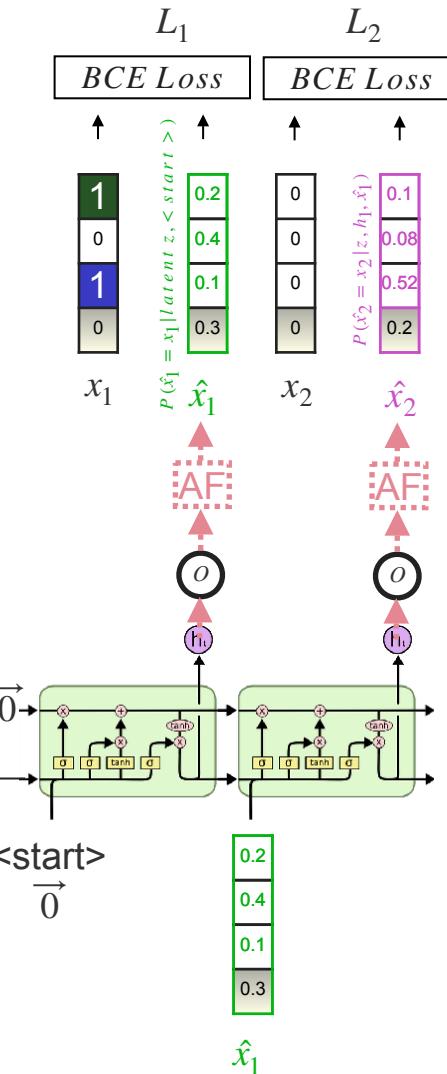
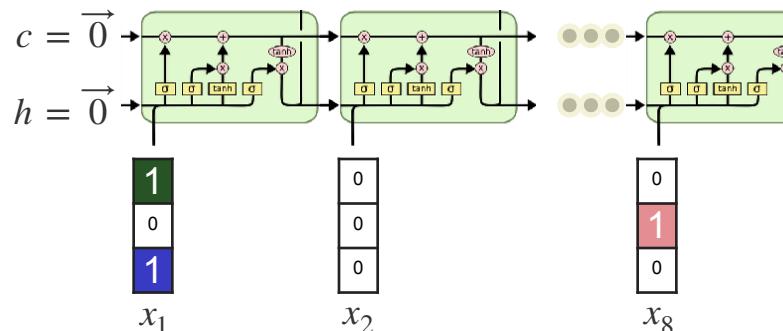
### Step 3: Decode the Output (reconstruct the input using Z)

INPUTS	
Hat	
Snare	
Kick	

$x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8$

OUTPUTS	
Hat	
Snare	
Kick	
<end>	

$x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8 \ x_9$



## Sequence to Sequence Architecture (Auto-encoder for sequential data)

### Step 3: Decode the Output (reconstruct the input using Z)

INPUTS	
Hat	
Snare	
Kick	

$x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8$

OUTPUTS	
Hat	
Snare	
Kick	
<end>	

$x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8 \ x_9$

