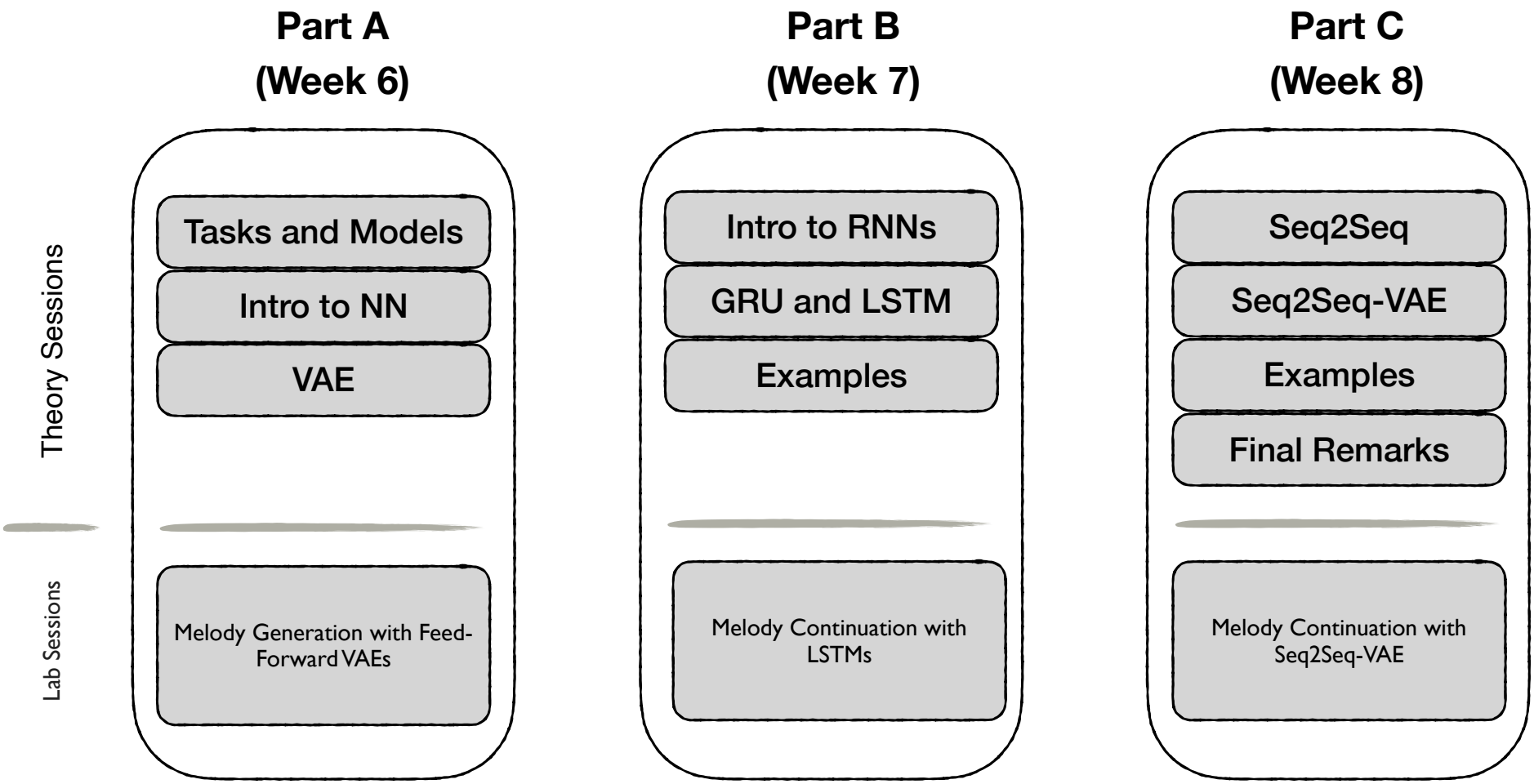
Recurrent Neural Nets for Music Generation

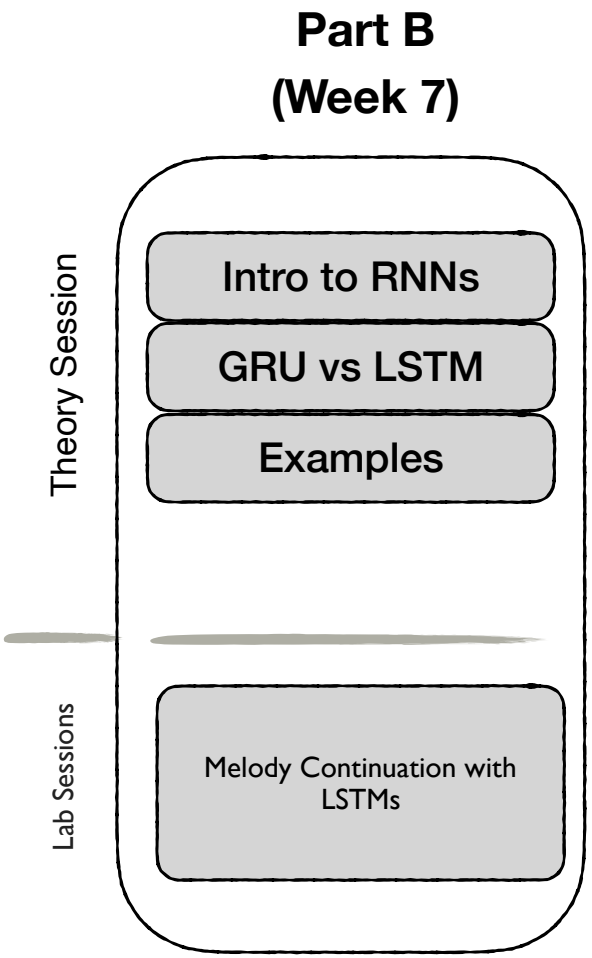
Computational Music Creativity, SMC (2020/21)

Prepared by Behzad Haki, February 2021

Tentative Plan (Weeks 6 to 8)

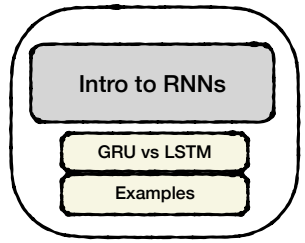


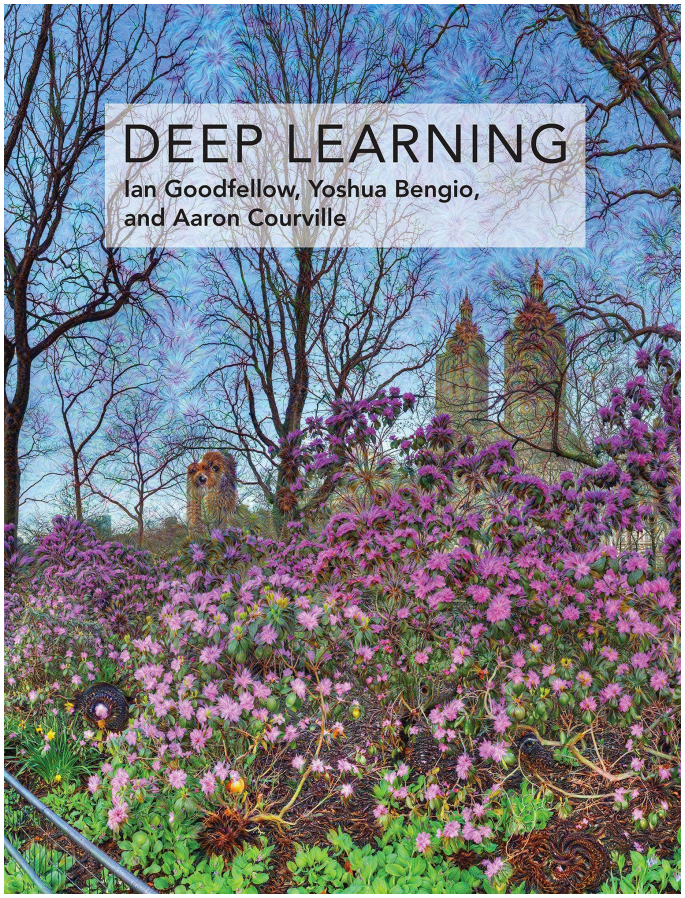
Plan for Today



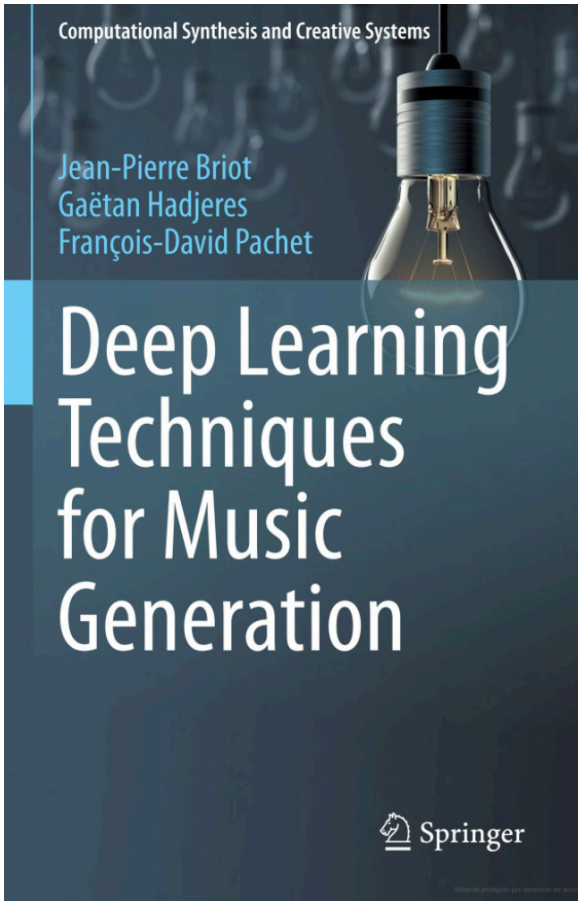
1. We'll start with a discussion on RNNs
2. We'll talk about some recurrent building blocks
3. We'll talk about some generative models based on RNNs

References

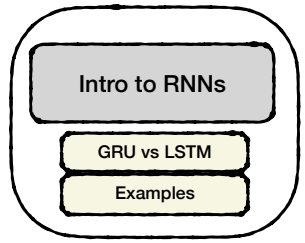




[1] Goodfellow, Ian, et al. *Deep learning*. Vol. 1. No. 2. Cambridge: MIT press, 2016. Chapter 10.

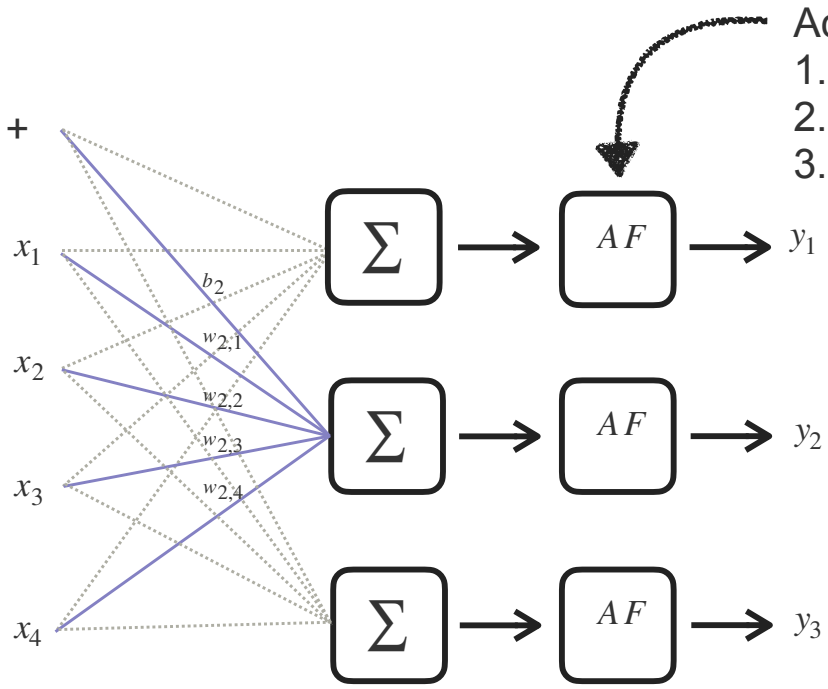


[2] Briot, Jean-Pierre, Gaëtan Hadjeres, and François Pachet. *Deep learning techniques for music generation*. Springer, 2020.



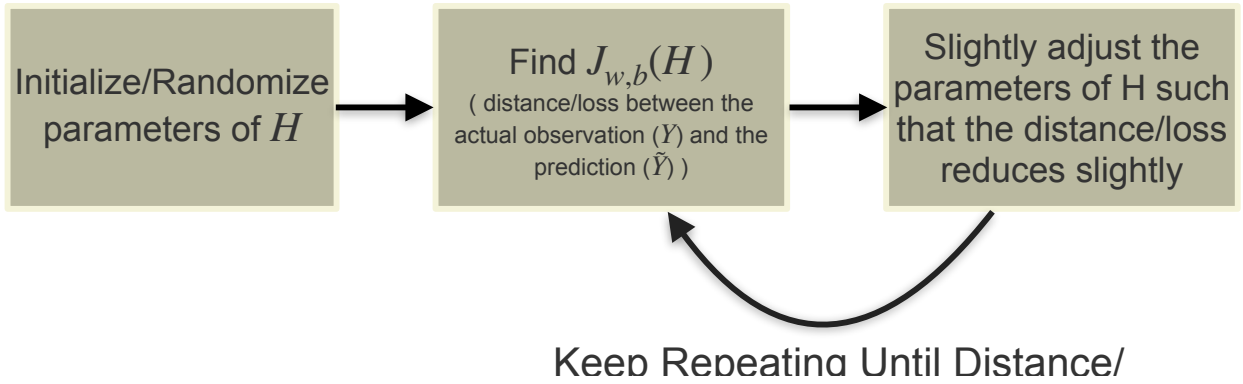
Multi-Layer Perceptron (MLP)

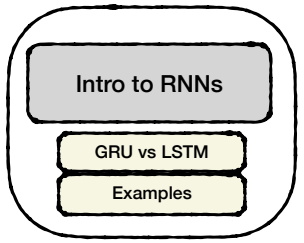
Review of Last Lecture



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

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

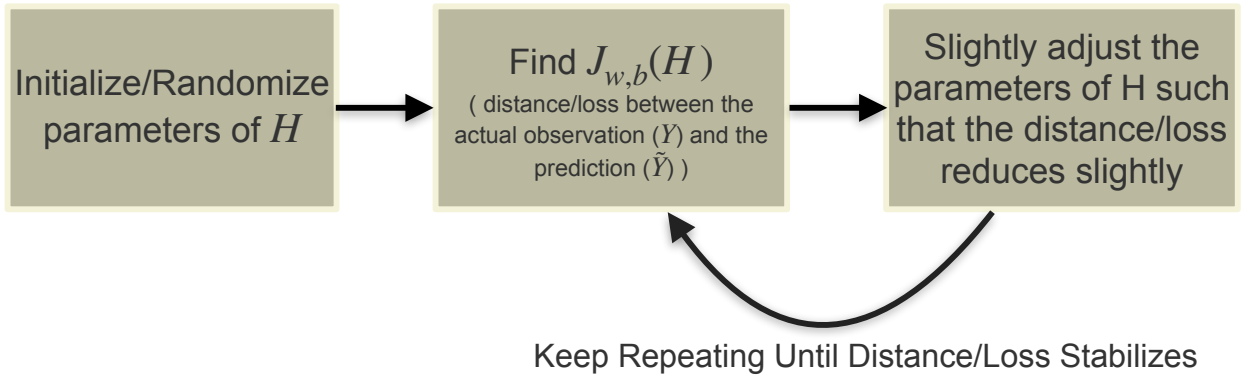




Revisiting MLPs for Sequential Data

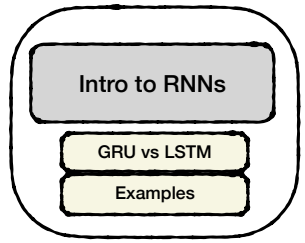
Issue 2: Vanishing Gradient

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!

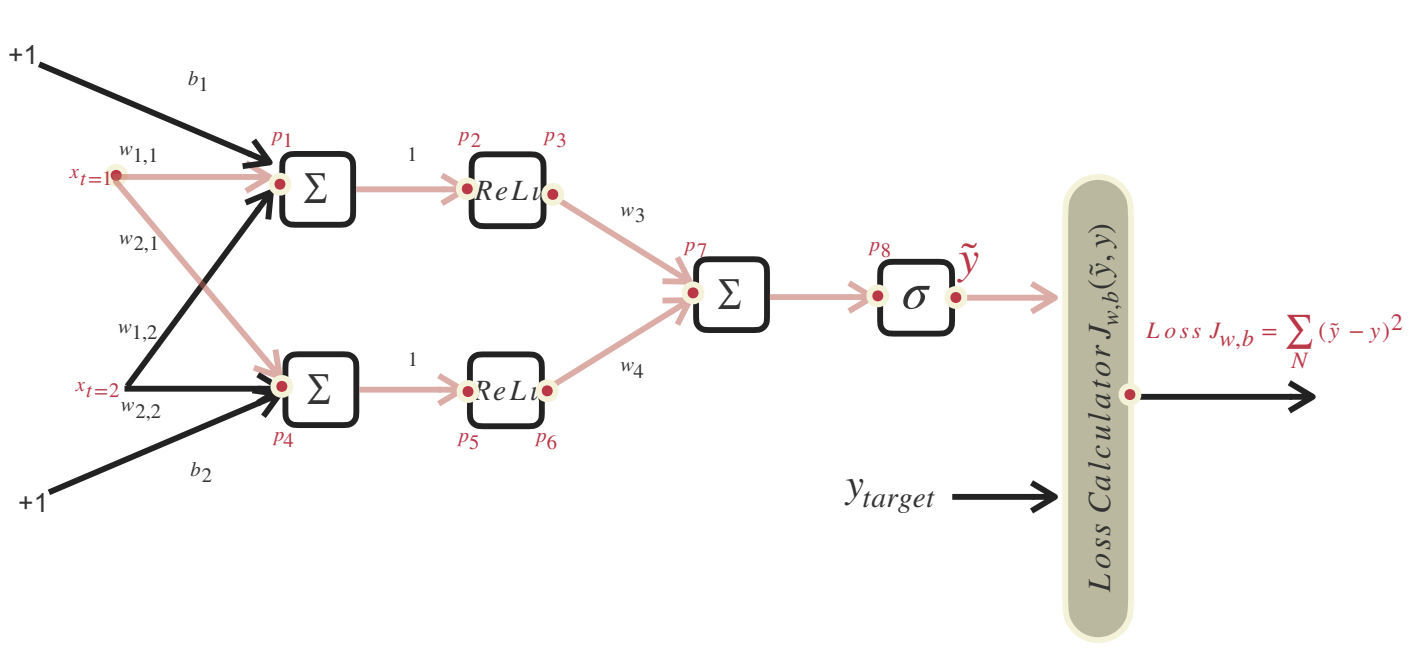
To find the gradients, we use back-propagation!

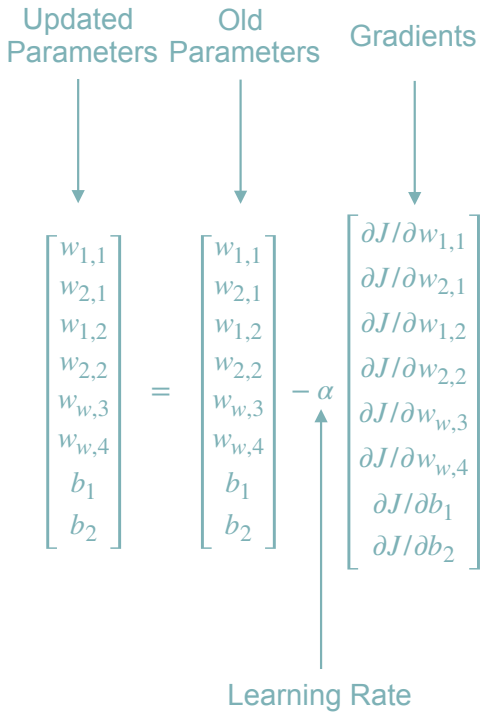


Revisiting MLPs for Sequential Data

Issue 1: Vanishing Gradient

Below is a simple example for back-propagation of gradients

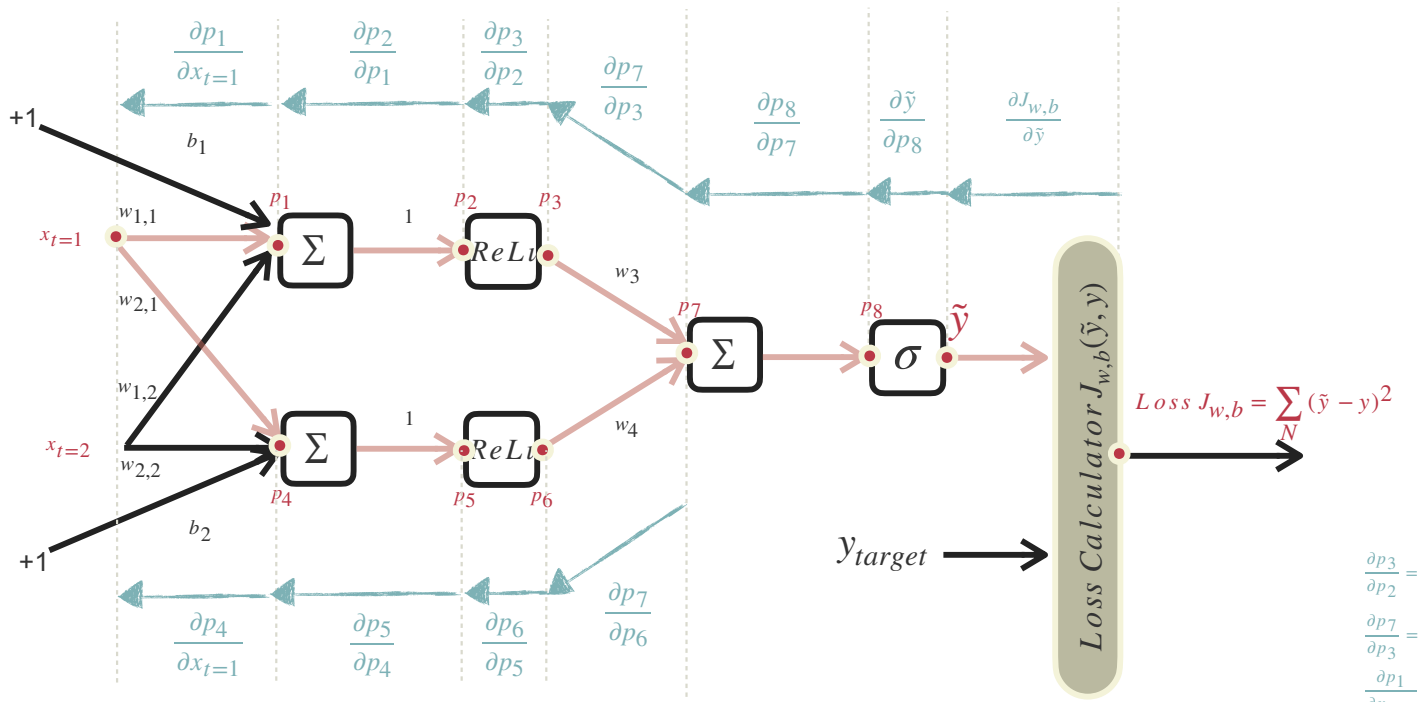




Revisiting MLPs for Sequential Data

Issue 1: Vanishing Gradient

Below is a simple example for back-propagation of gradients



$$\begin{bmatrix} w_{1,1} \\ w_{2,1} \\ w_{1,2} \\ w_{2,2} \\ w_{w,3} \\ w_{w,4} \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} w_{1,1} \\ w_{2,1} \\ w_{1,2} \\ w_{2,2} \\ w_{w,3} \\ w_{w,4} \\ b_1 \\ b_2 \end{bmatrix} - \alpha \begin{bmatrix} \partial J/\partial w_{1,1} \\ \partial J/\partial w_{2,1} \\ \partial J/\partial w_{1,2} \\ \partial J/\partial w_{2,2} \\ \partial J/\partial w_{w,3} \\ \partial J/\partial w_{w,4} \\ \partial J/\partial b_1 \\ \partial J/\partial b_2 \end{bmatrix}$$

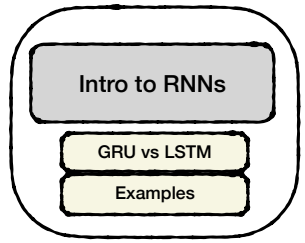
$$\frac{\partial p_3}{\partial p_2} = \frac{\partial ReLu(p_2)}{\partial p_2} \in \{0,1\} \qquad \frac{\partial J_{w,b}}{\partial \tilde{y}} = \frac{2}{N} * \sum (\tilde{y} - y)$$

$$\frac{\partial p_7}{\partial p_3} = w_3, \frac{\partial p_7}{\partial p_6} = w_4 \qquad \frac{\partial \tilde{y}}{\partial p_8} = \frac{\partial \sigma(p_8)}{\partial p_8} = \sigma(x)(1 - \sigma(x))$$

$$\frac{\partial p_1}{\partial x_{t=1}} = w_{1,1}, \frac{\partial p_4}{\partial x_{t=1}} = w_{2,1} \qquad \frac{\partial p_8}{\partial p_7} = \frac{\partial p_2}{\partial p_1} = \frac{\partial p_5}{\partial p_4} = 1$$

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

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

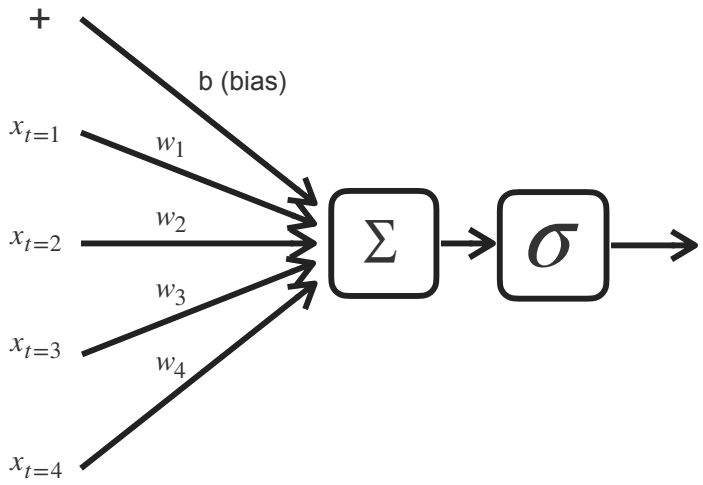


Revisiting MLPs for Sequential Data

Issue 2: Fixed number of time-steps

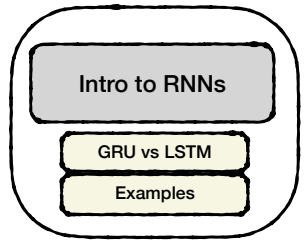
Let's assume we have the following model which tries to predict whether a hi-hat should be played given a sequence of hi-hats previously played.

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



$$\tilde{x}_{t=5} = P(x_{t=5} \mid x_{t=1}, x_{t=2}, x_{t=3}, x_{t=4})$$

$$= \sigma(Wx + b) = \sigma([w_1 \quad w_2 \quad w_3 \quad w_4] \begin{bmatrix} x_{t=1} \\ x_{t=2} \\ x_{t=3} \\ x_{t=4} \end{bmatrix} + b)$$

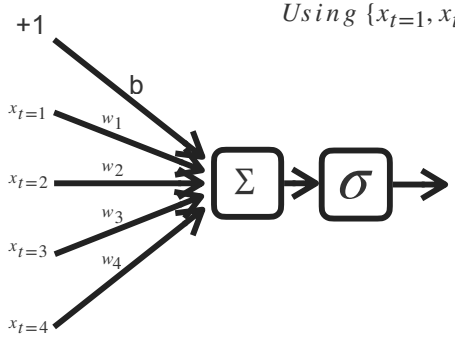


Revisiting MLPs for Sequential Data

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.

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

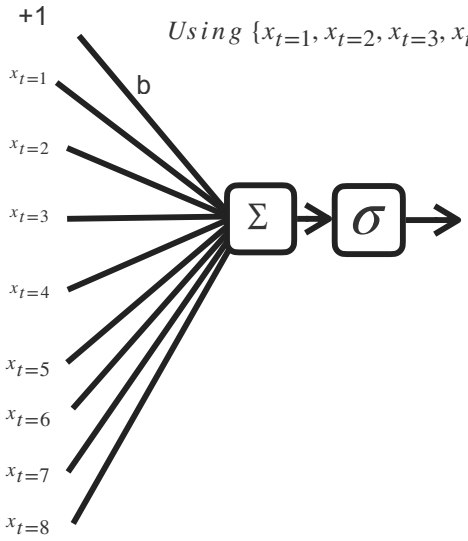


$$\tilde{x}_{t=5} = P(x_{t=5} | x_{t=1}, x_{t=2}, x_{t=3}, x_{t=4})$$

$$= \sigma(Wx + b) = \sigma([w_1 \quad w_2 \quad w_3 \quad w_4] \begin{bmatrix} x_{t=1} \\ x_{t=2} \\ x_{t=3} \\ x_{t=4} \end{bmatrix} + b)$$

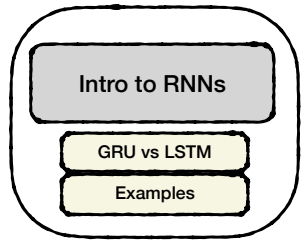
If we want to base our prediction on a longer sequencer, we need to re-structure the model (and hence re-train) as follows:

$$Using\ \{x_{t=1}, x_{t=2}, x_{t=3}, x_{t=4}, x_{t=5}, x_{t=6}, x_{t=7}, x_{t=8}\} \Rightarrow Predict\ x_{t=9}$$



$$\tilde{x}_{t=9} = P(x_{t=9} | x_{t=1}, x_{t=2}, x_{t=3}, x_{t=4}, x_{t=5}, x_{t=6}, x_{t=7}, x_{t=8})$$

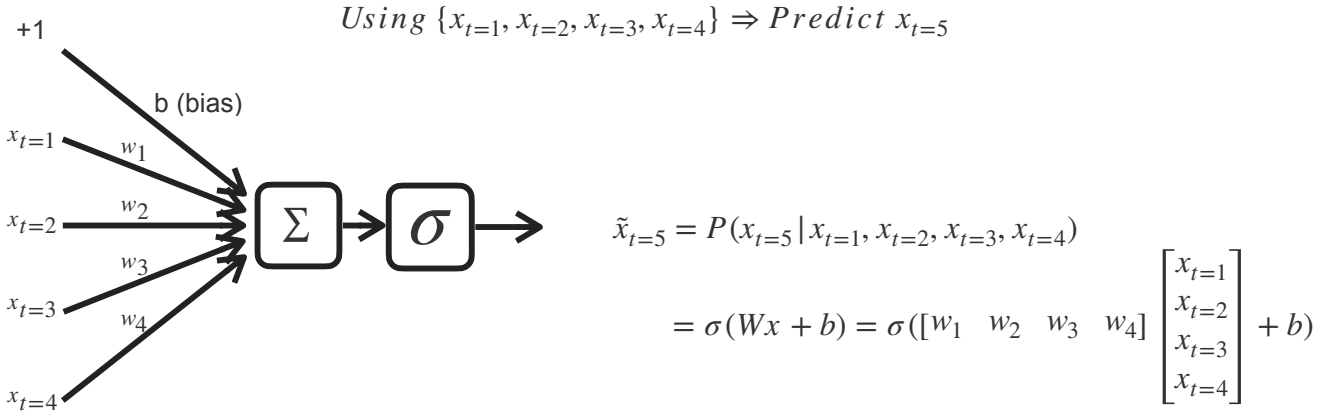
$$= \sigma(Wx + b) = \sigma([w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5 \quad w_6 \quad w_7 \quad w_8] \begin{bmatrix} x_{t=1} \\ x_{t=2} \\ x_{t=3} \\ x_{t=4} \\ x_{t=5} \\ x_{t=6} \\ x_{t=7} \\ x_{t=8} \end{bmatrix} + b)$$



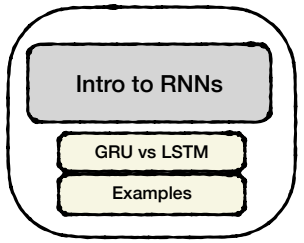
Revisiting MLPs for Sequential Data

Issue 2: Varying input length

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.



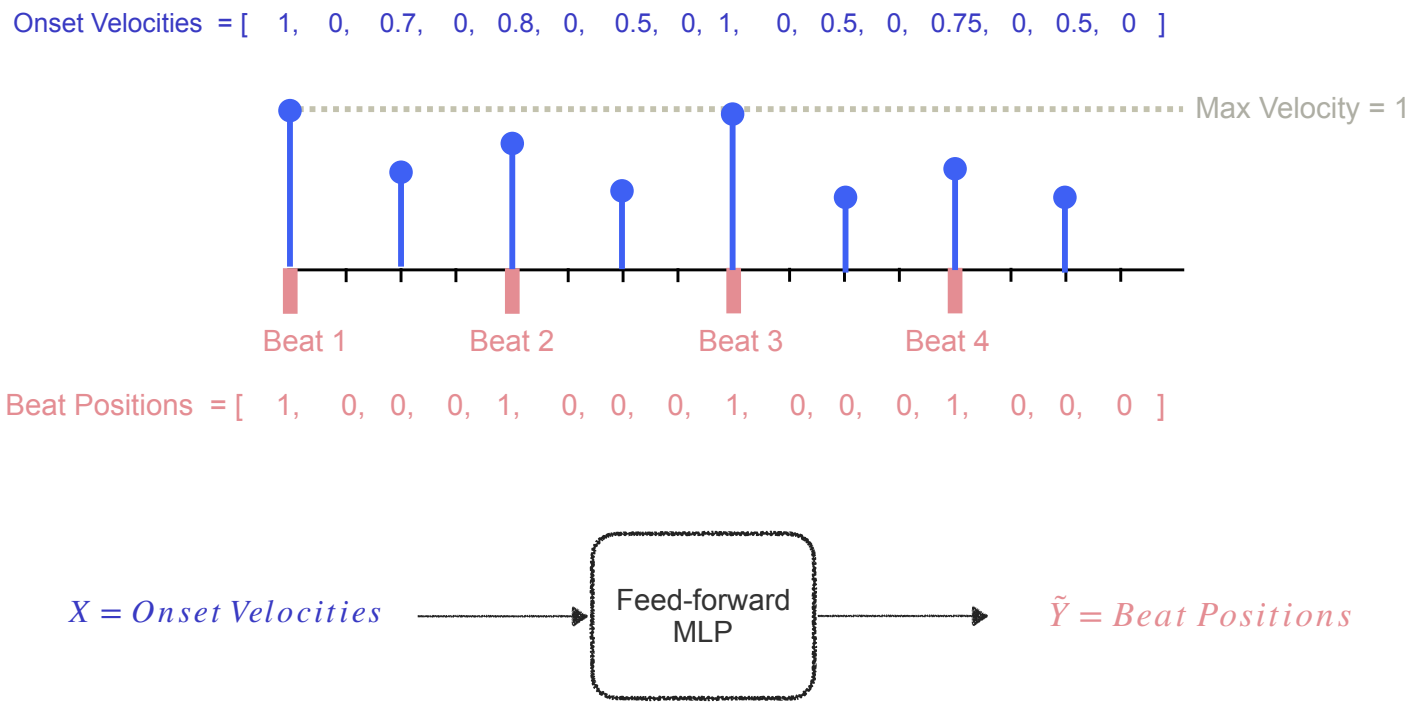
Moreover, how would we modify the above architecture to accommodate inputs of varying length?



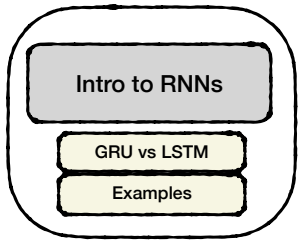
Revisiting MLPs for Sequential Data

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



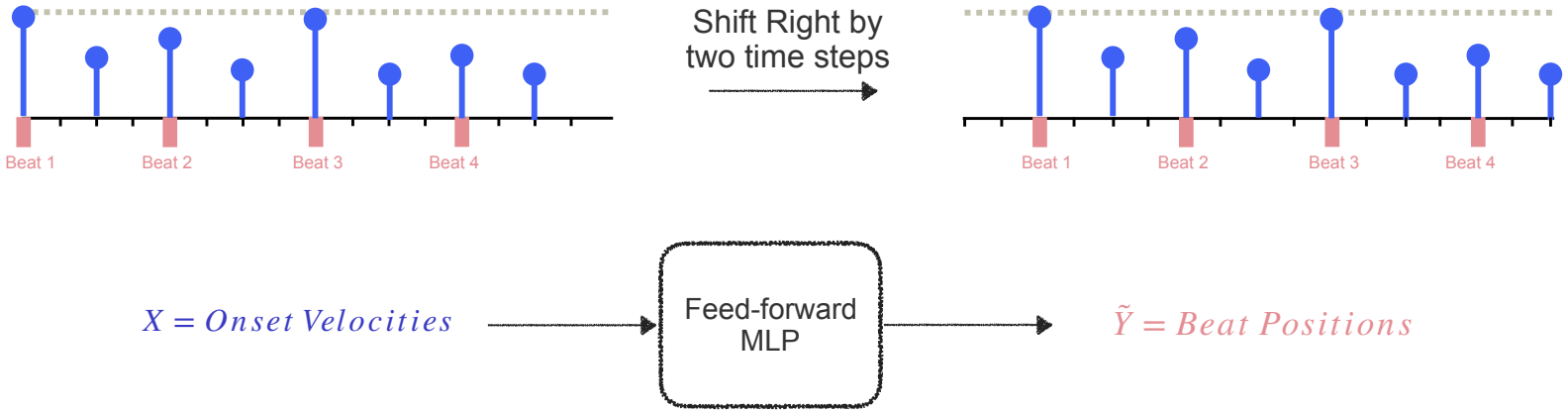
A fully connected feed-forward network would have separate parameters for all input features across time, from which It tries to come up with "rules" that best describe all the possible (or rather seen) examples in the input.



Revisiting MLPs for Sequential Data

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

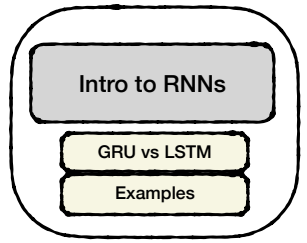
Now what if our onset observations are slightly shifted?



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

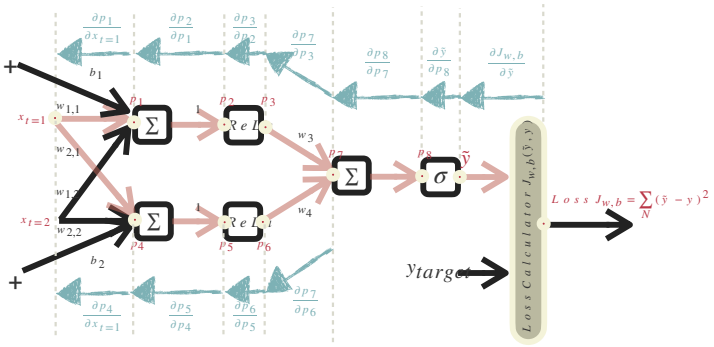
In order to be able to correctly predict the shifted version, the model needs to be trained on enough examples of shifted inputs!

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!

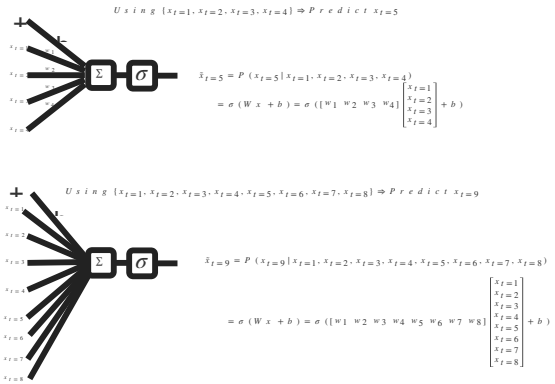


Revisiting MLPs for Sequential Data

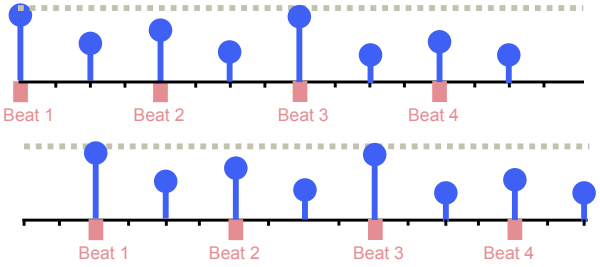
Issues: Recap



Issue 1: Vanishing Gradients



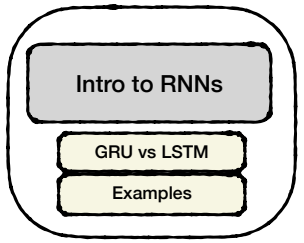
Issue 2: Varying Input Lengths



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

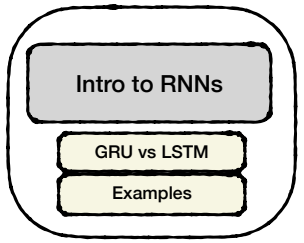
Recurrent Neural Nets (RNNs)



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)

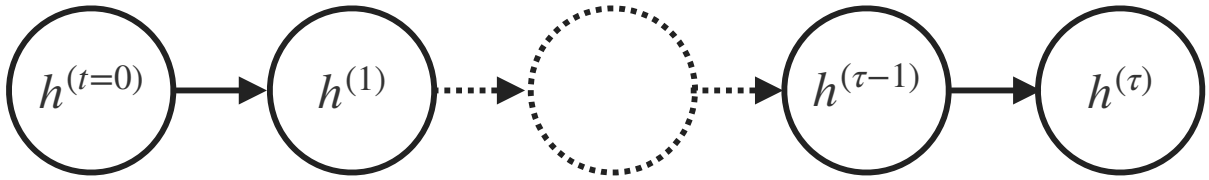
What does recurrence mean?

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

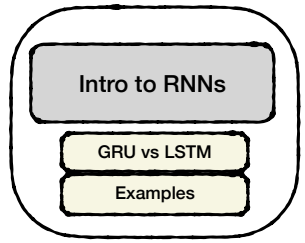
$$h^{(t)} = f(h^{(t-1)}, \theta_{w,b})$$

Where $h^{(t=\tau)}$ denotes the state of the model at time τ, 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 τ.



Recurrent Neural Nets (RNNs)



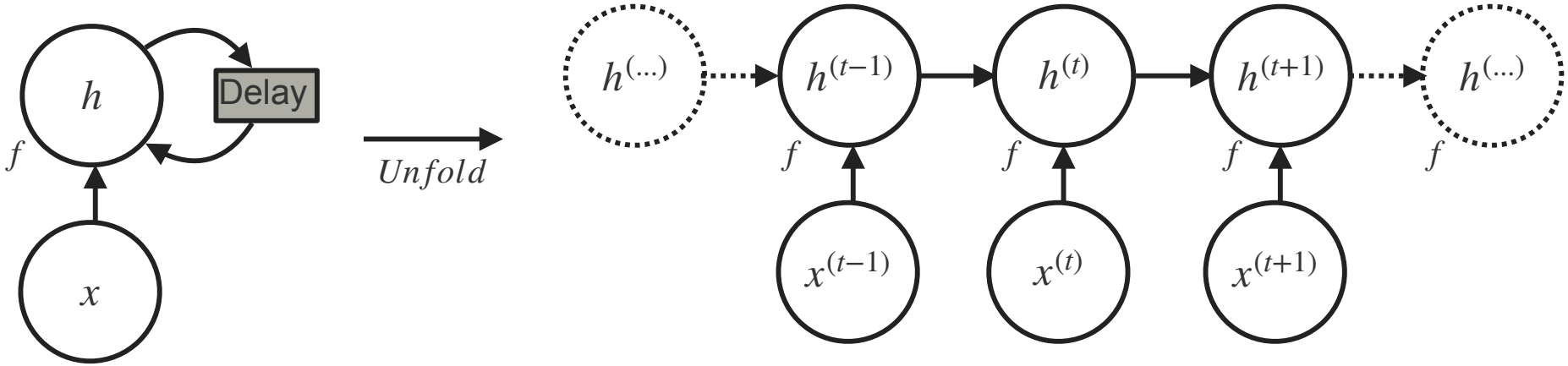
How can we make the recurrence to be also dependant on past inputs?

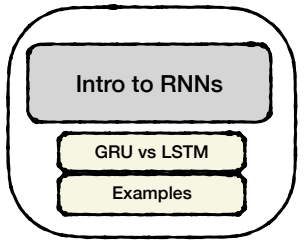
In reality, the state vector should also keep track of the whole past sequence:

Constant over time

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

Changes over time





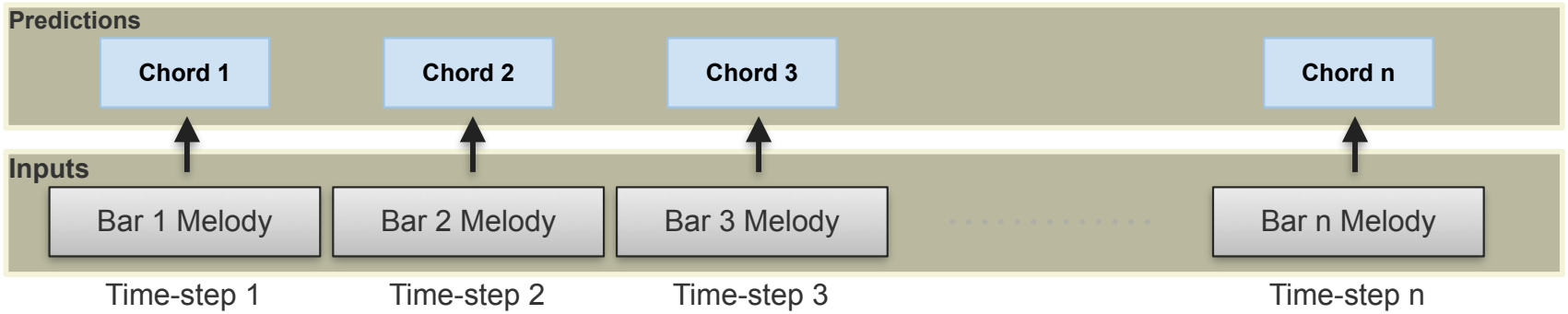
Recurrent Neural Nets (RNNs)

How do we make predictions over time?

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

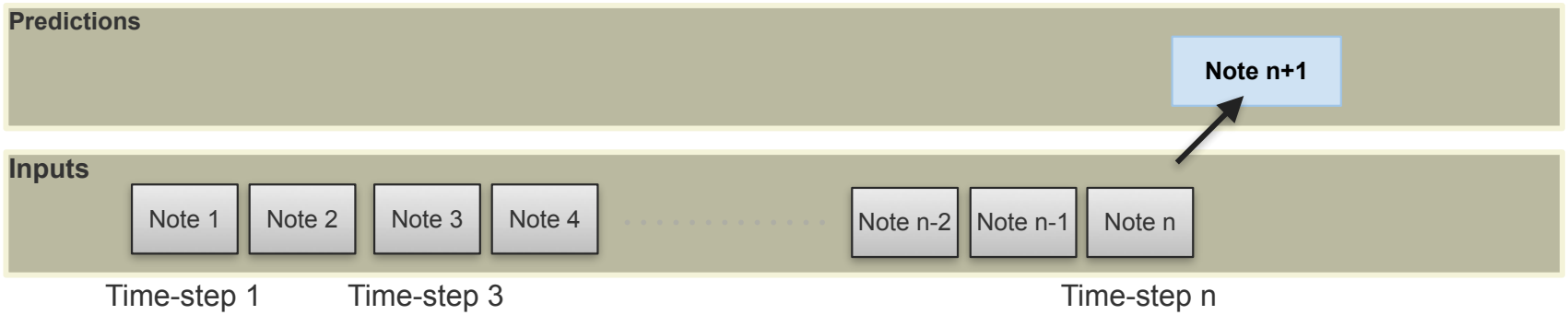
1. Predicting an output at every time step

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



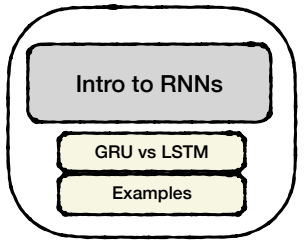
2. Predicting an output at the very end of a sequence

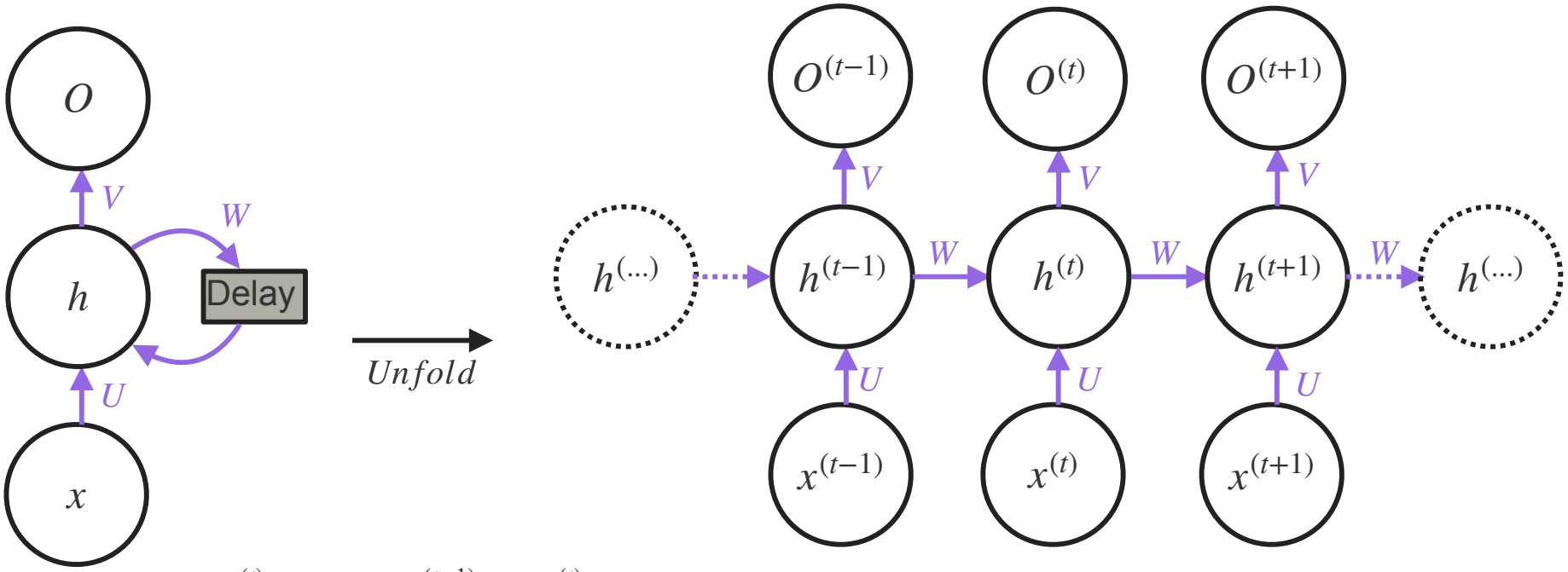
Example: Find the most probable note following a given melody



Recurrent Neural Nets (RNNs)

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

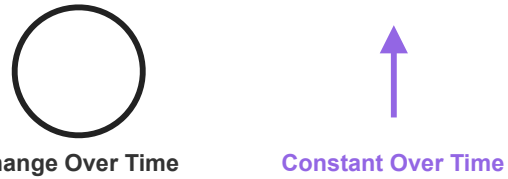




$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}x^{(t)}$$

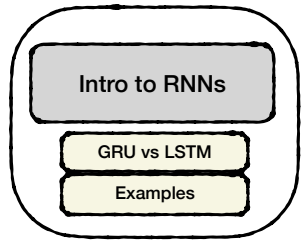
$$\boldsymbol{h}^{(t)} = tanh(\boldsymbol{a}^{(t)})$$

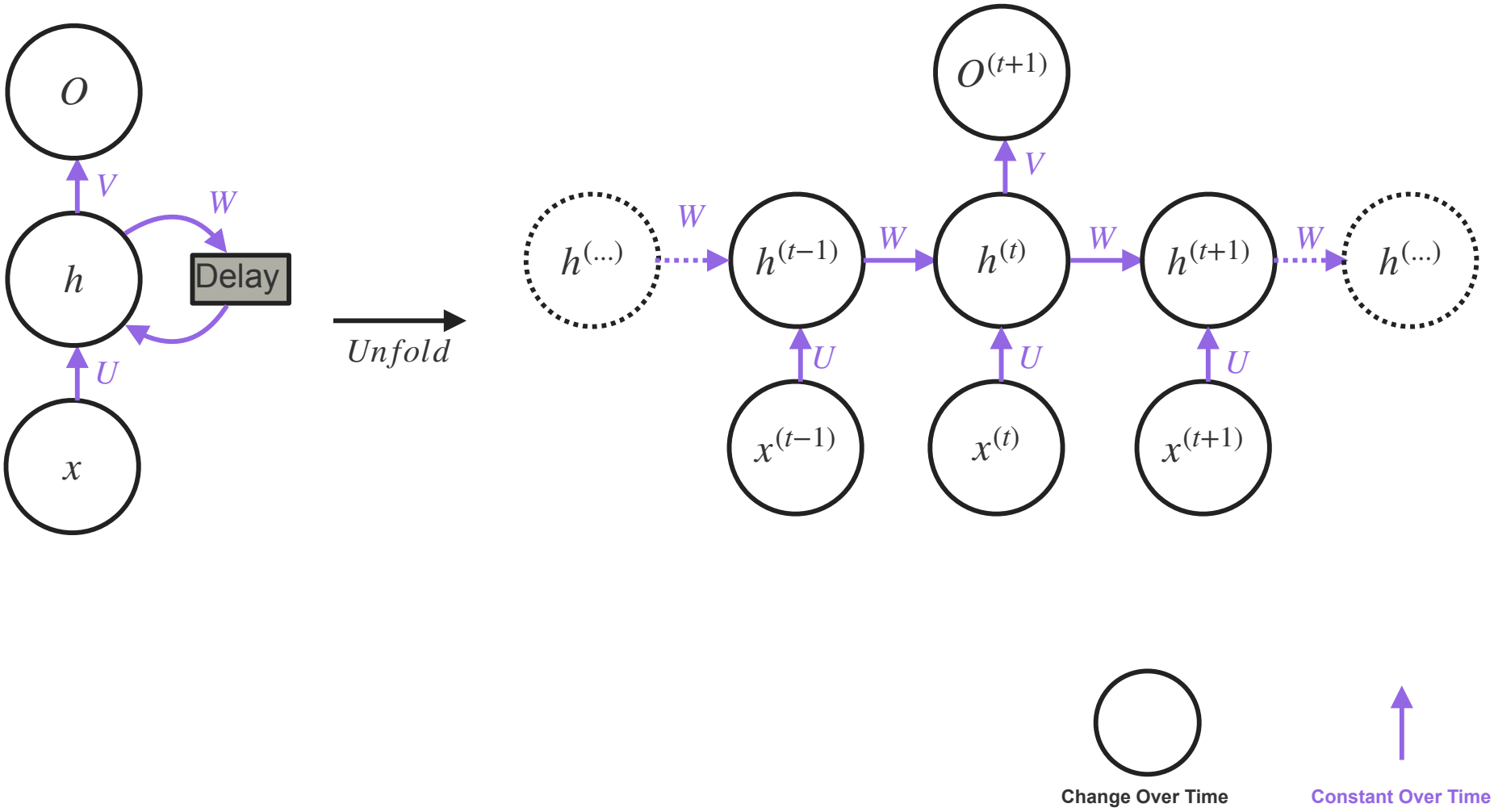
$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}$$



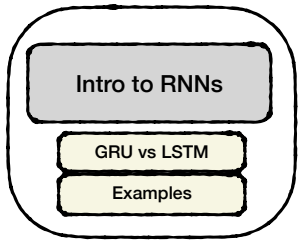
Recurrent Neural Nets (RNNs)

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





Recurrent Neural Nets (RNNs)



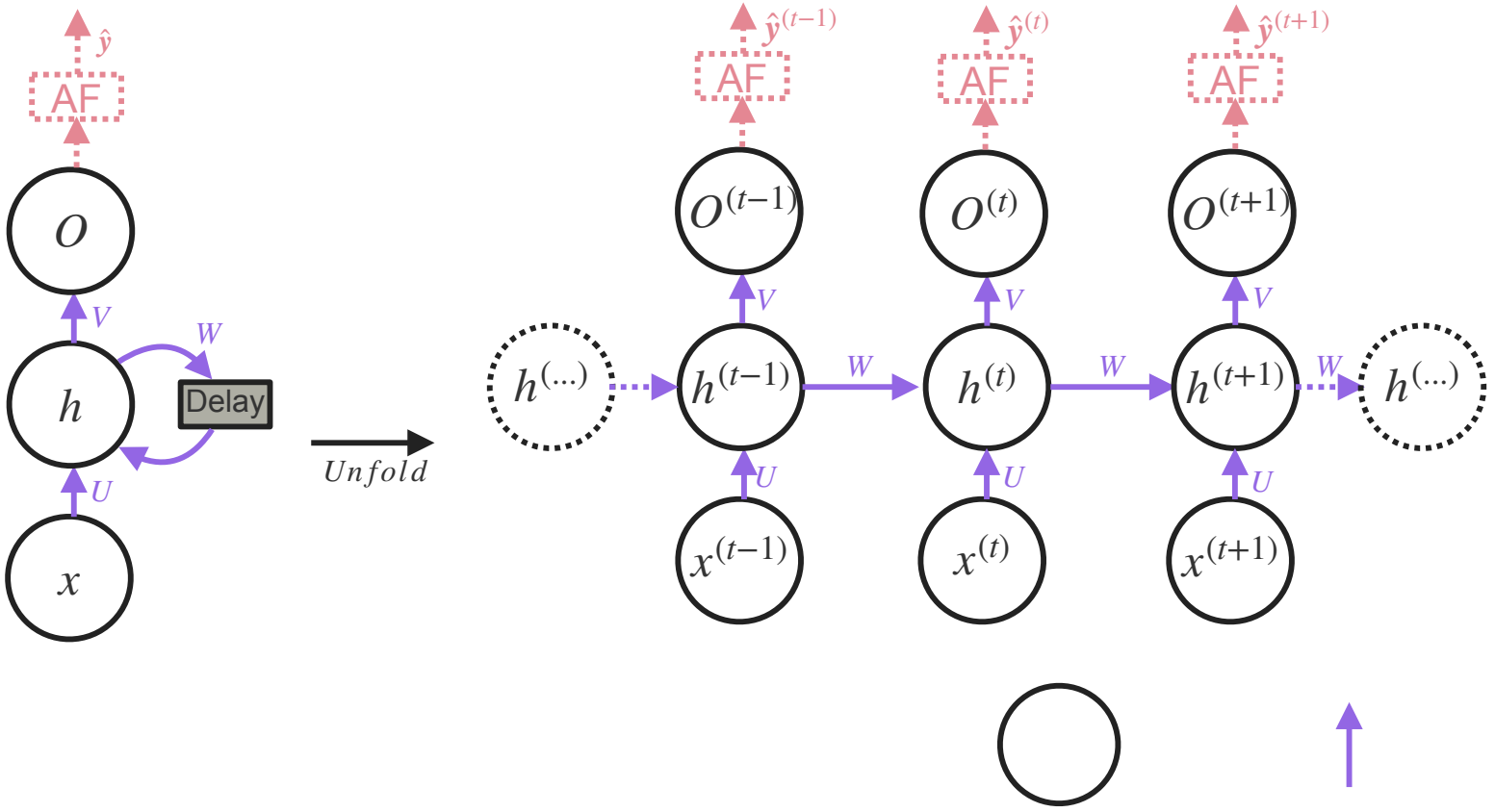
Step 1: Predict output $\hat{y}$ at required time-steps using the output layer O

(4) $\hat{y}^{(t)} = AF(o^{(t)})$

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

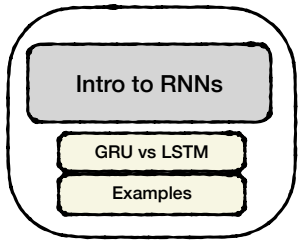
(2) $h = tanh(a^{(t)})$

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



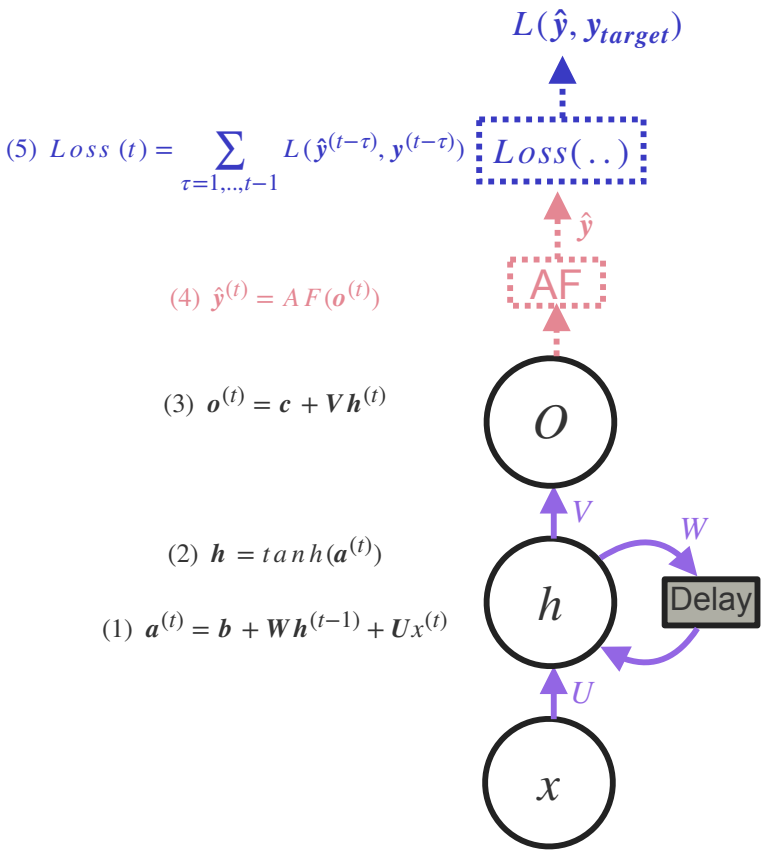
https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577

Recurrent Neural Nets (RNNs)

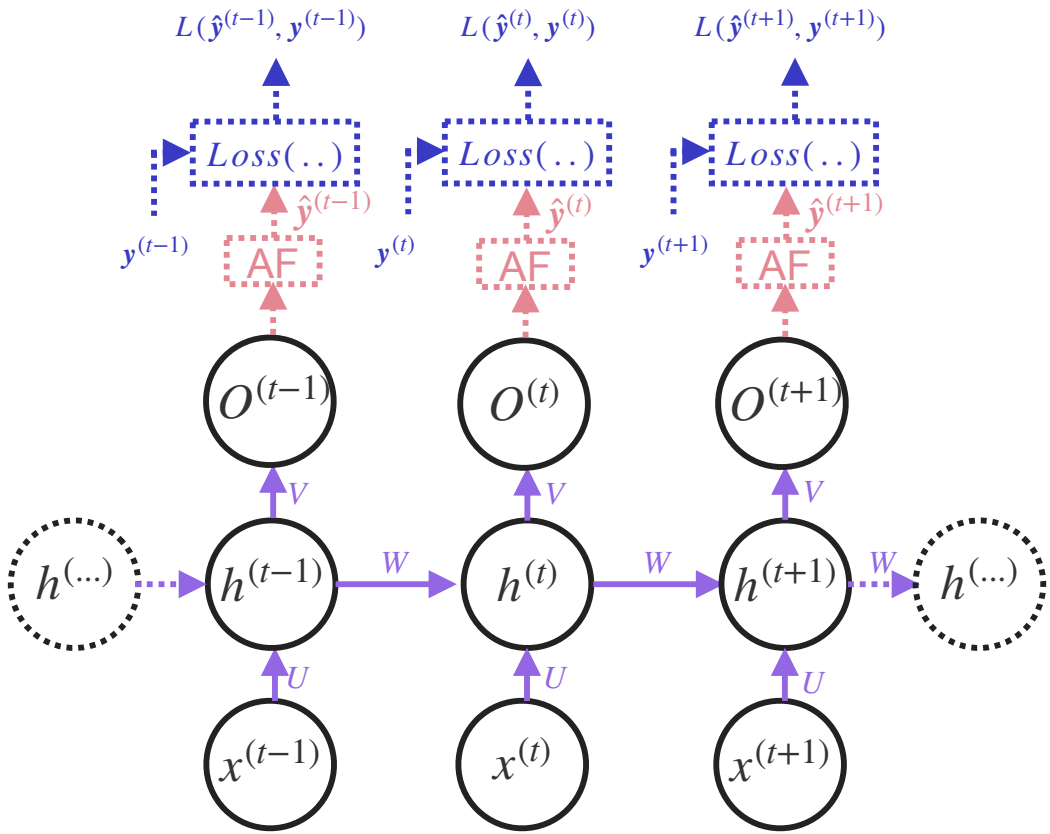


Step 2: Loss Calculation

$$Total\ Loss\ (t) = \sum_{\tau=1,..,t-1} L(\hat{y}^{(t-\tau)}, y^{(t-\tau)})$$



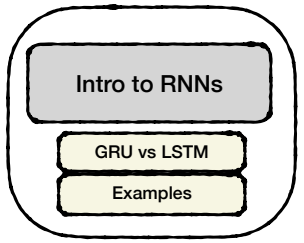
Unfold



Change Over Time

Constant Over Time

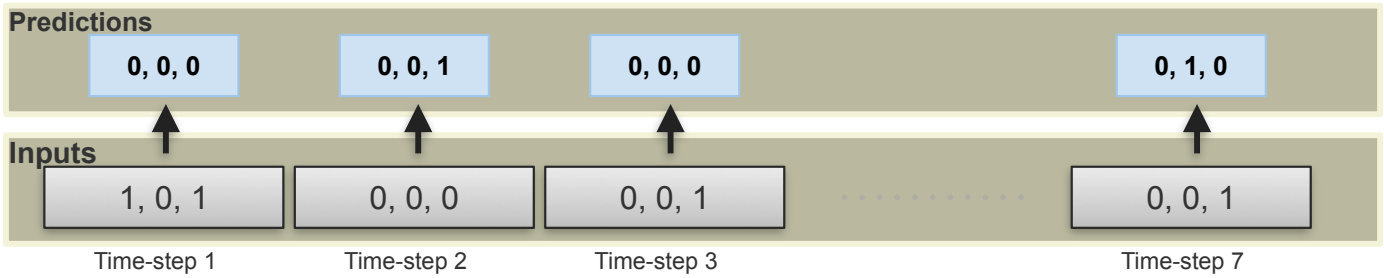
https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577

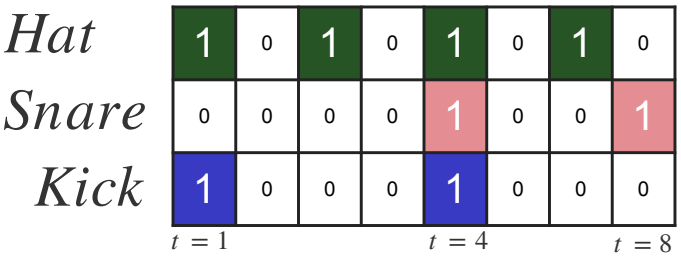


Recurrent Neural Nets (RNNs)

Example: Drum Pattern Continuation

Objective: Predict Next Time-Step Given Past Sequence





First Iteration: Given x_1 predict x_2 —>

Second Iteration: Given x_1 and x_2 predict x_3 —>

Third Iteration: Given x_1 , x_2, x_3 predict x_4 —>

. . .

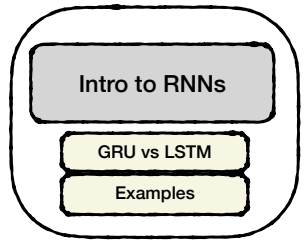

```
sequence = torch.tensor([[[1, 0, 1], [0, 0, 0], [0, 0, 1], [0, 0, 0], [1, 1, 1], [0, 0, 0], [0, 0, 1], [0, 1, 0]]],
                        dtype =torch.float32)   # Shape (batch_size, time steps, feature per step) -> (1, 8, 3)


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)

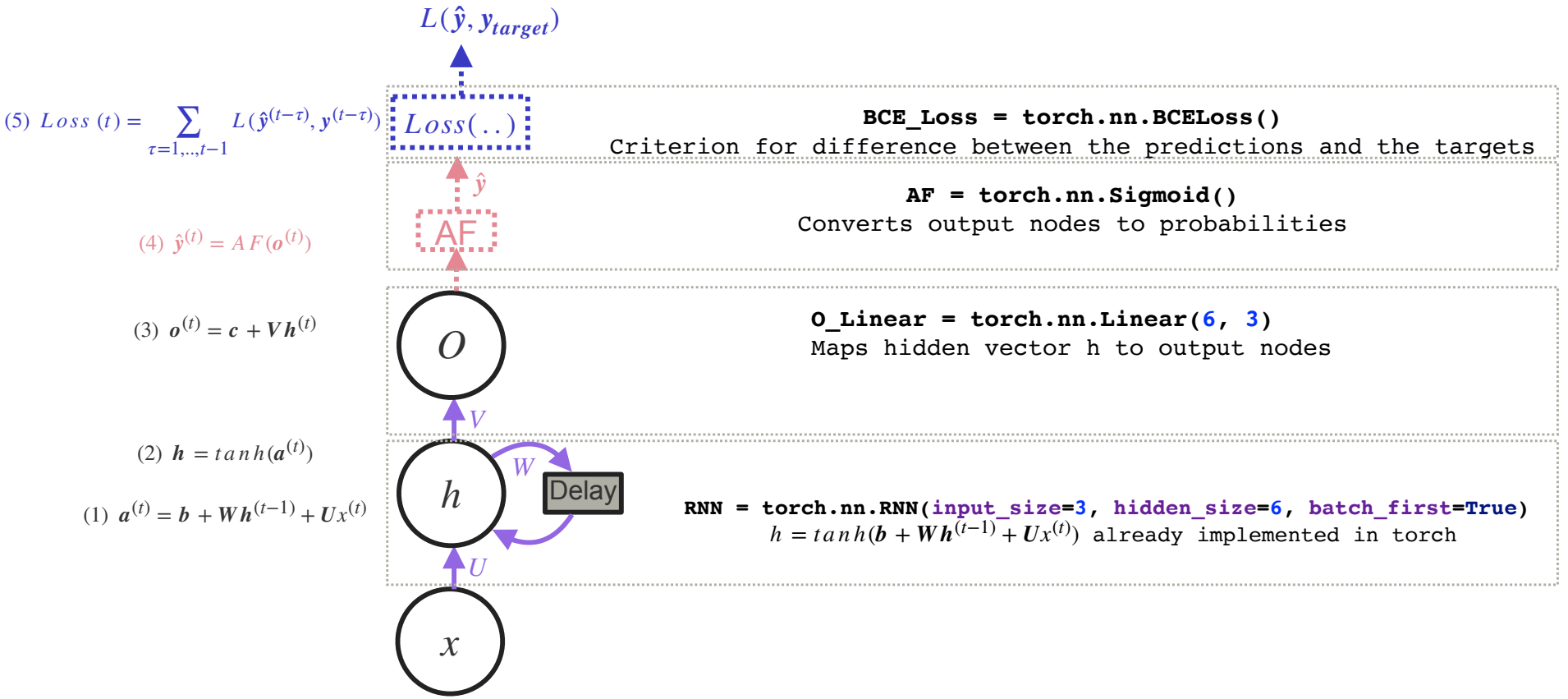
Example: Drum Pattern Continuation

Objective: Predict Next Time-Step Given Past Sequence

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
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



Setup Network

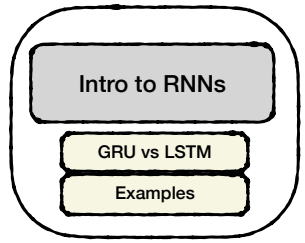


Recurrent Neural Nets (RNNs)

Example: Drum Pattern Continuation

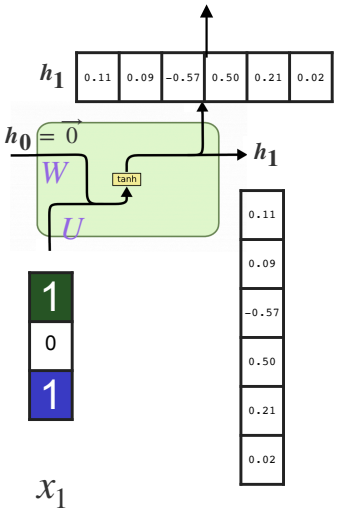
Objective: Predict Next Time-Step Given Past Sequence

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
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



Time-Step 1

First Iteration: Given x_1 predict x_2 —> $P(y_1 = x_2 \,|\, x_1)$



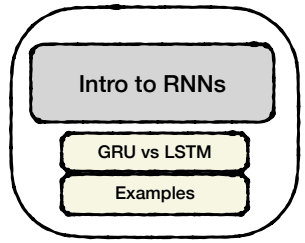

```
_, h1 = RNN(x1, h0)

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

Recurrent Neural Nets (RNNs)

Example: Drum Pattern Continuation

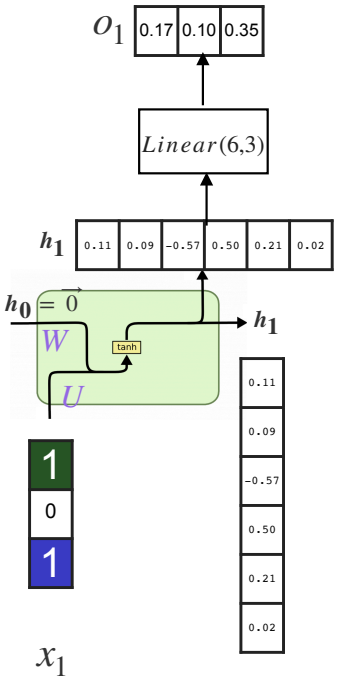
Objective: Predict Next Time-Step Given Past Sequence

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
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



Time-Step 1

First Iteration: Given x_1 predict x_2 —> $P(y_1 = x_2 \mid x_1)$




```
o1 = O_Linear(h1)

o1
     tensor([[[0.1742, 0.1092, 0.3523]]])
```

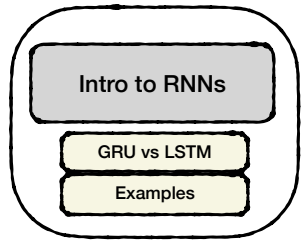
```
_, h1 = RNN(x1, h0)

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

Recurrent Neural Nets (RNNs)

Example: Drum Pattern Continuation

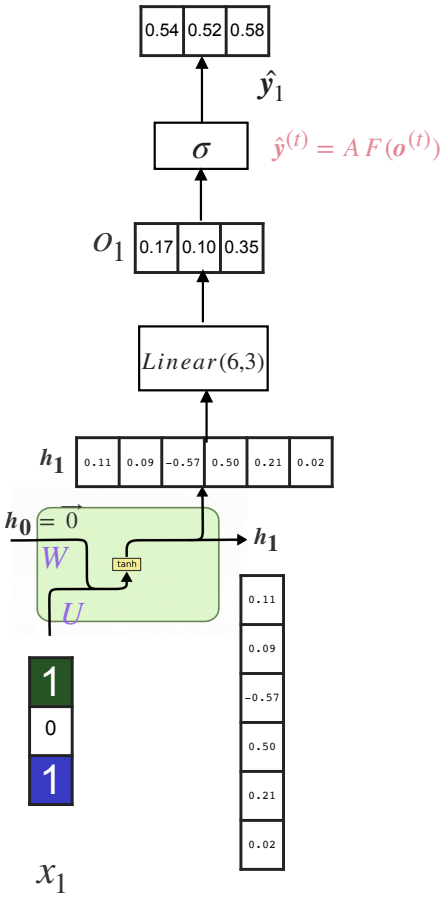
Objective: Predict Next Time-Step Given Past Sequence

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
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



Time-Step 1

First Iteration: Given x_1 predict x_2 —> $P(y_1 = x_2 \,|\, x_1)$

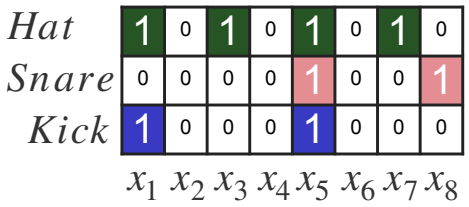


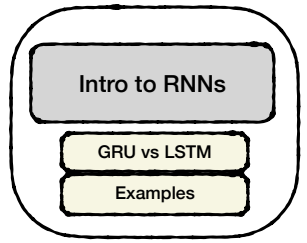

```
y_predict1 = AF(o1)

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]]])
```

Recurrent Neural Nets (RNNs)





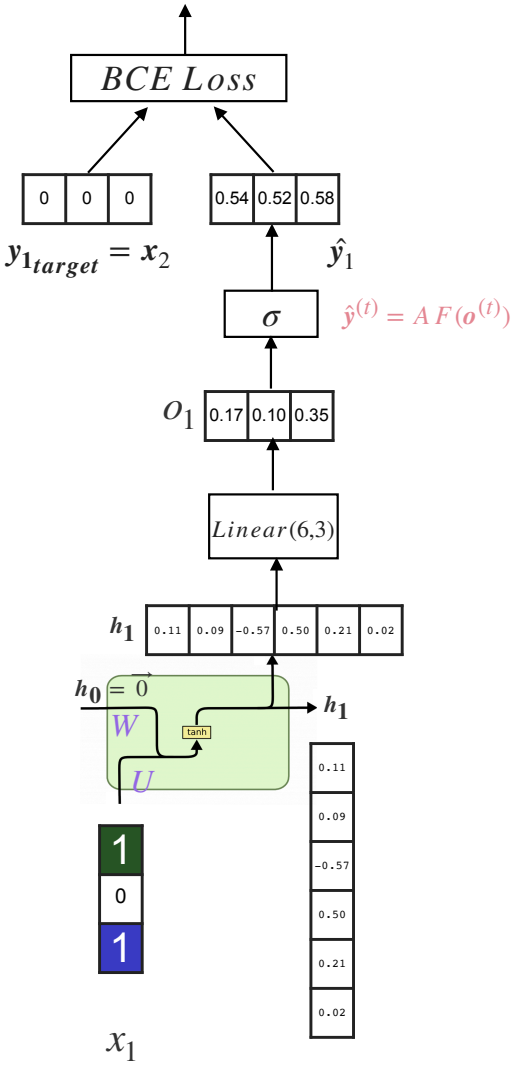
Example: Drum Pattern Continuation

Objective: Predict Next Time-Step Given Past Sequence

Time-Step 1

First Iteration: Given x_1 predict x_2 —> $P(y_1 = x_2 | x_1)$

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




```
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]]])
```

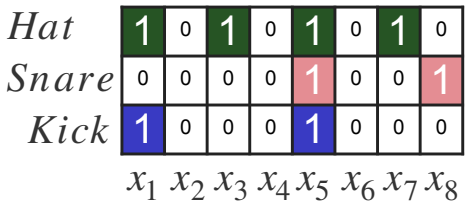
```
o1 = O_Linear(h1)

o1
     tensor([[[0.1742, 0.1092, 0.3523]]])
```

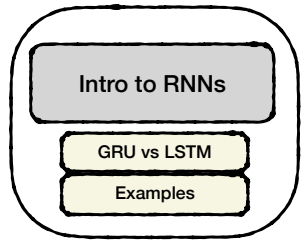
```
_, h1 = RNN(x1, h0)

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

Recurrent Neural Nets (RNNs)

Example: Drum Pattern Continuation

Objective: Predict Next Time-Step Given Past Sequence

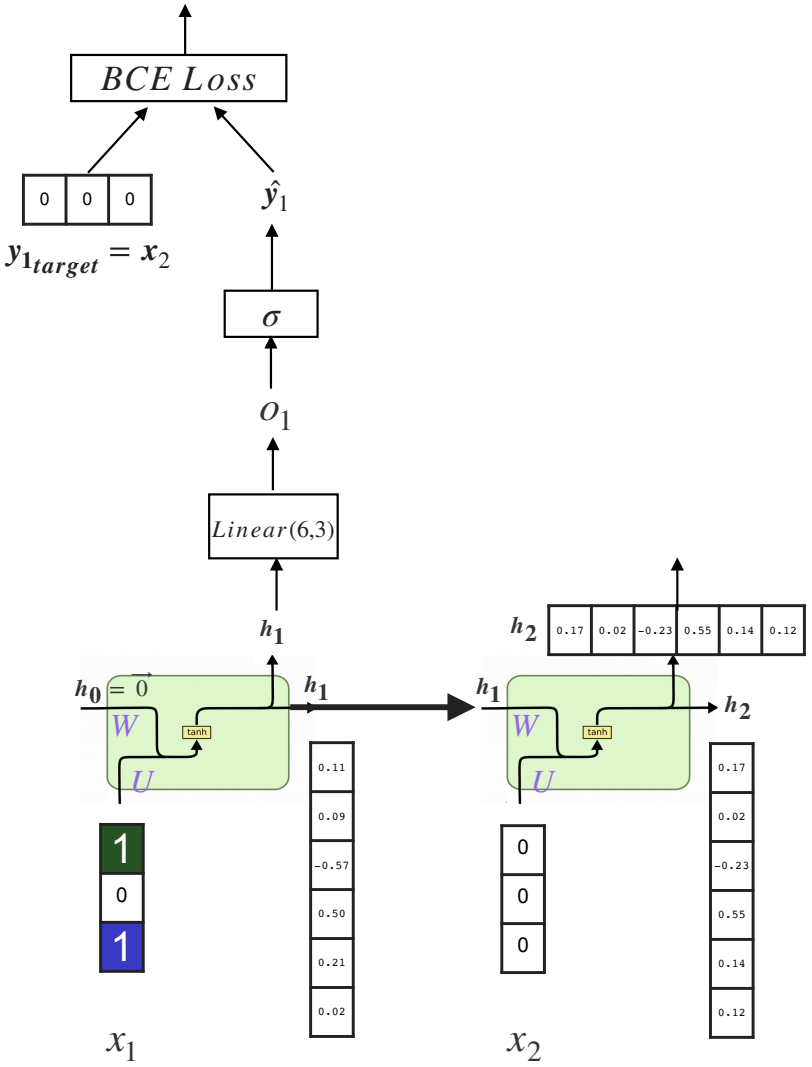




Time-Step 2

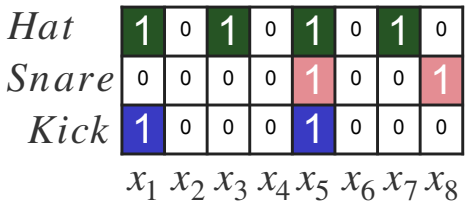
Second Iteration: Given x_1 and x_2 predict x_3 —> $P(y_2 = x_3 | x_1, x_2)$

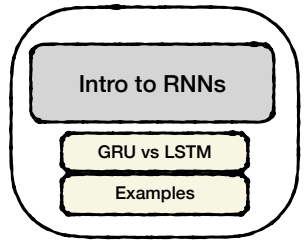
$L^{(1)} = 0.8060 = P(y = x_2 | x_1)$




```
_, h2 = RNN(x2, h1)

h2:
    tensor([[[0.1749,  0.0217, -0.2394,  0.5537,  0.1435,  0.1275]]])
```

Recurrent Neural Nets (RNNs)





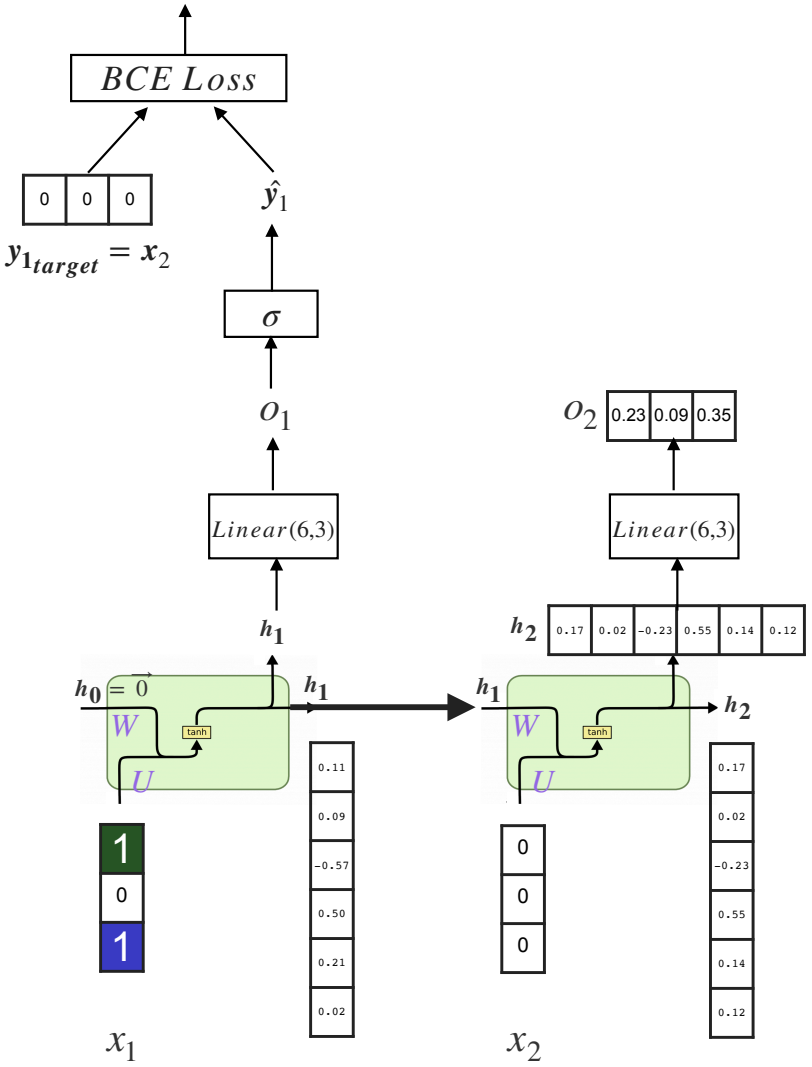
Example: Drum Pattern Continuation

Objective: Predict Next Time-Step Given Past Sequence

Time-Step 2

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

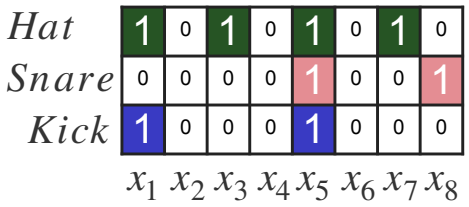
$L^{(1)} = 0.8060 = P(y = x_2 | x_1)$

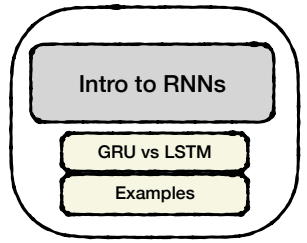



```
O2 = O_Linear(h2)

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)





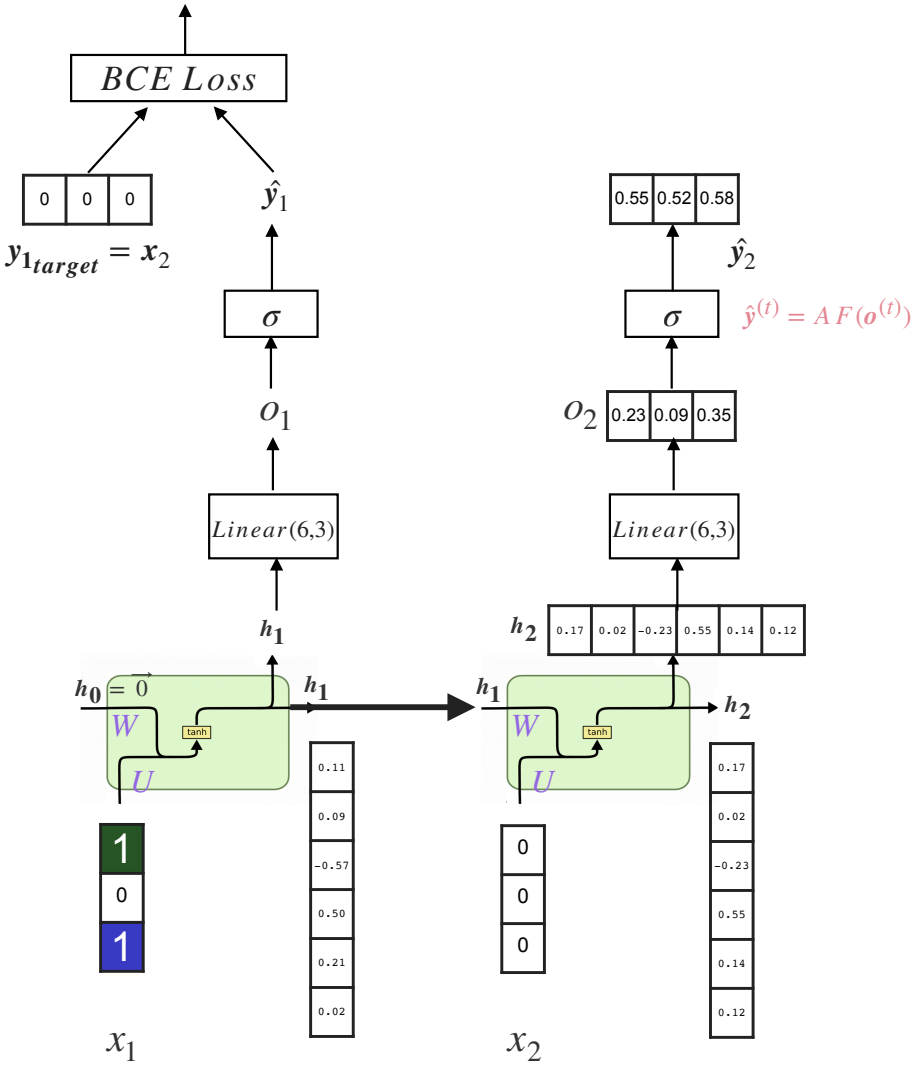
Example: Drum Pattern Continuation

Objective: Predict Next Time-Step Given Past Sequence

Time-Step 2

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

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

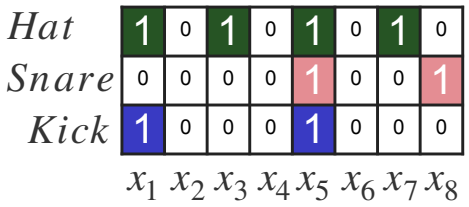


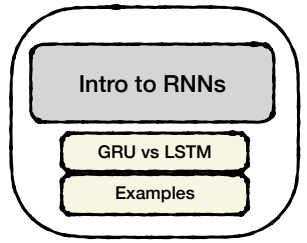

```
y_predict2 = AF(o2)

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)





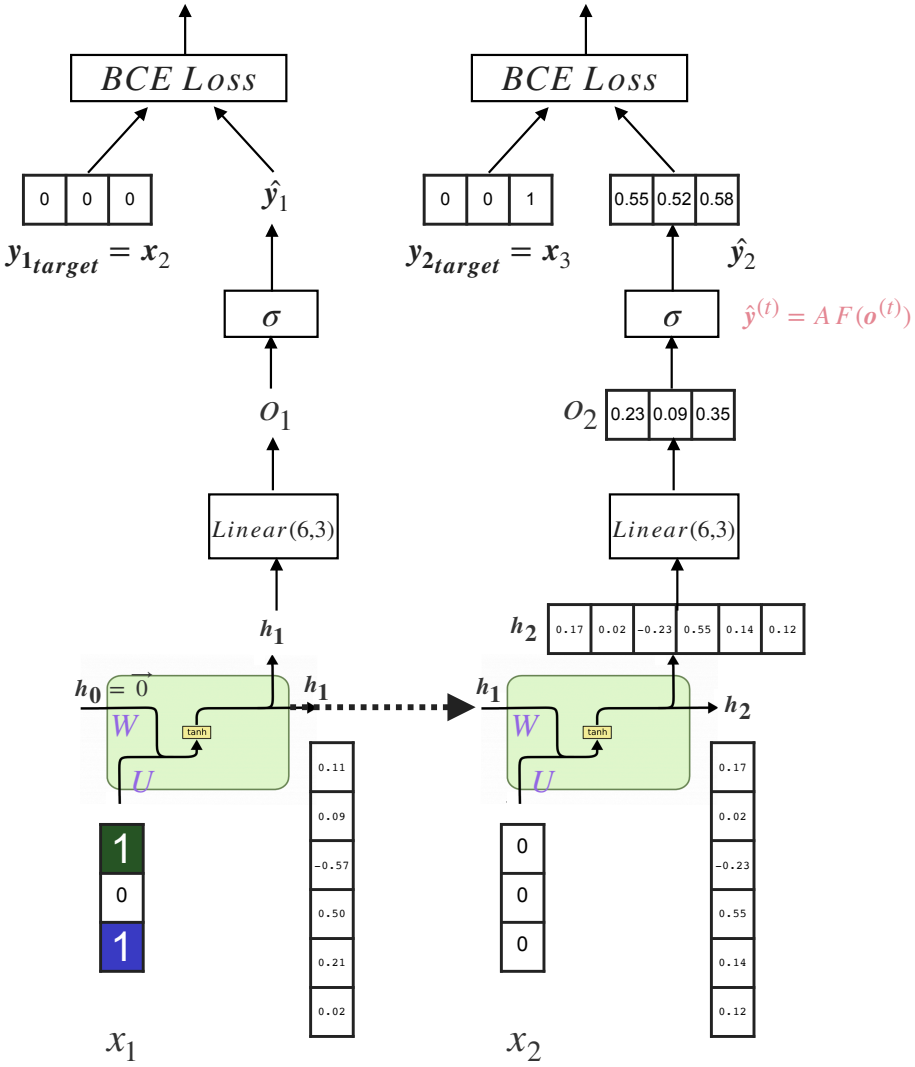
Example: Drum Pattern Continuation

Objective: Predict Next Time-Step Given Past Sequence

Time-Step 2

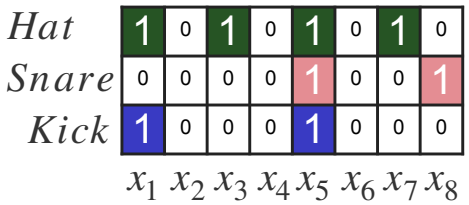
Second Iteration: Given x_1 and x_2 predict x_3 —> $P(y_2 = x_3 | x_1, x_2)$

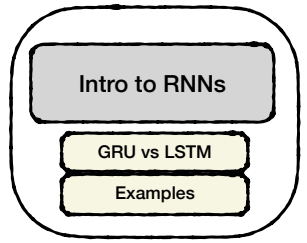
$L^{(1)} = 0.8060 = P(y = x_2 | x_1)$ $L^{(2)} = 0.6968 = P(y_2 = x_3 | x_1, x_2)$




```
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)



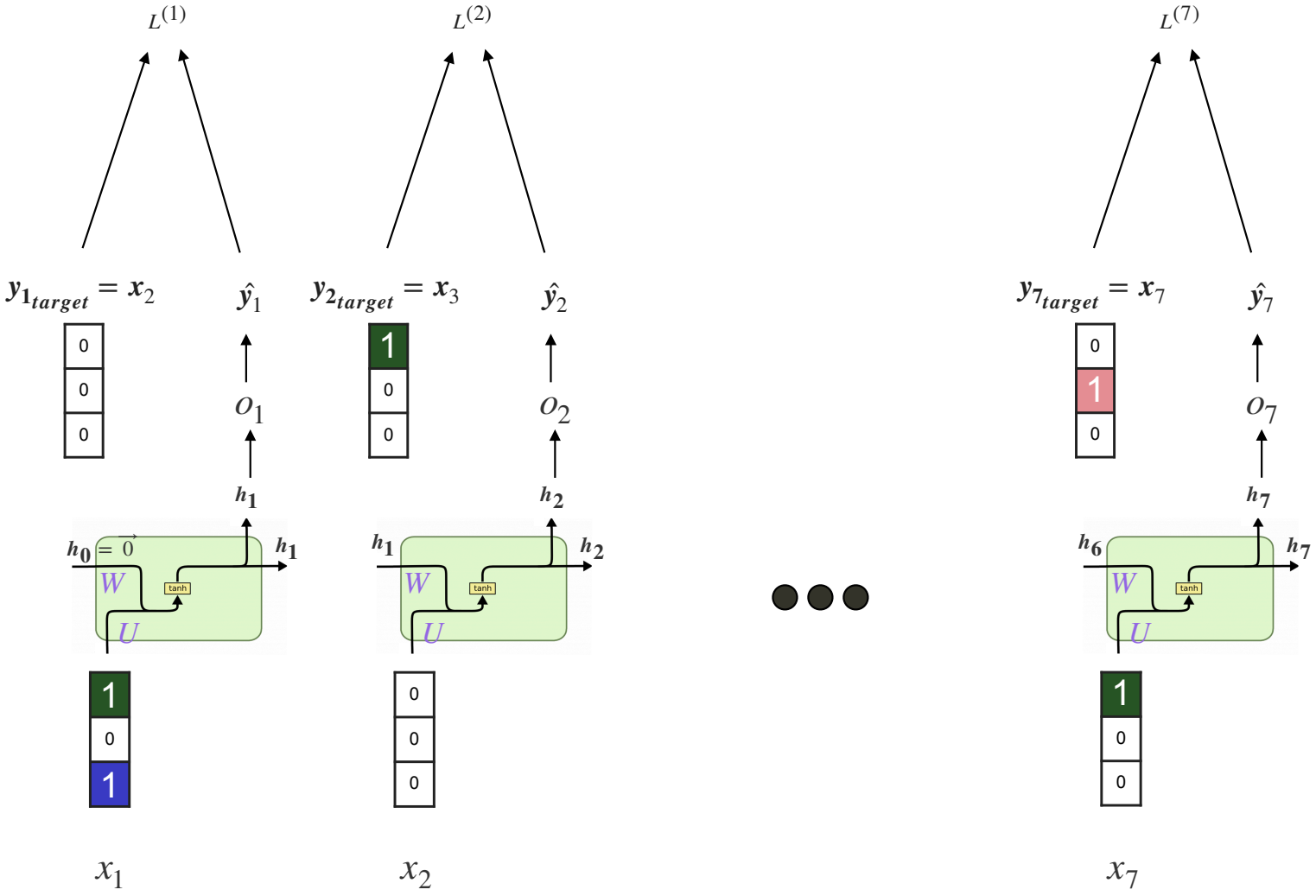


Example: Drum Pattern Continuation

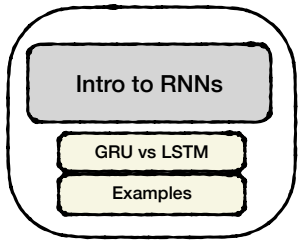
Objective: Predict Next Time-Step Given Past Sequence

After 7 Iterations

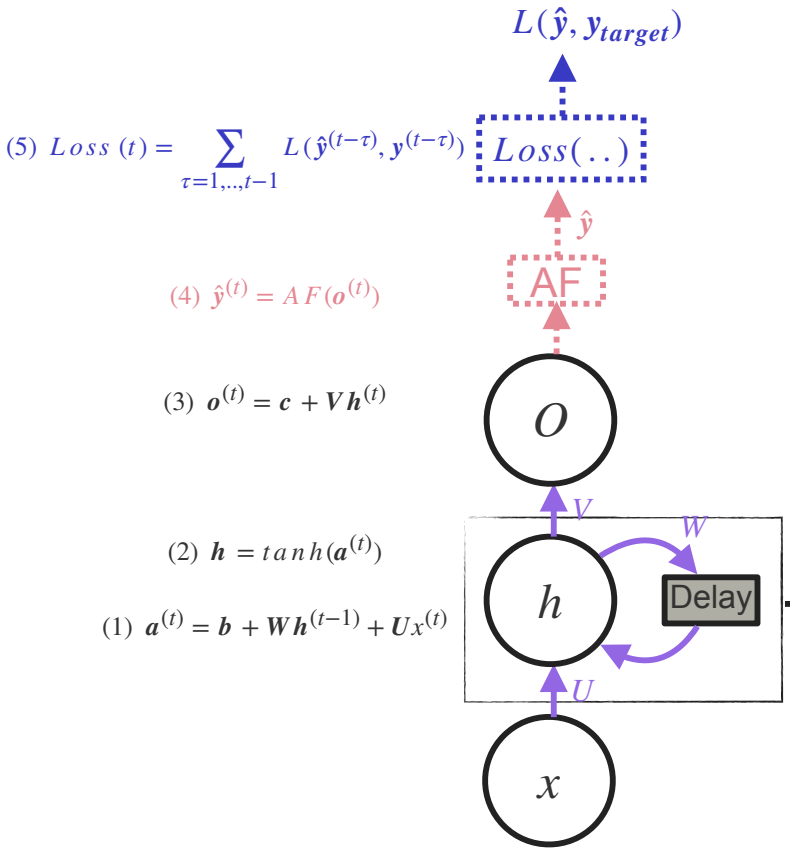
$$Total\ Loss = L^{(1)} + L^{(2)} + \ldots + L^{(7)}$$



Recurrent Neural Nets (RNNs)



A Major Issue of Vanilla RNNs: Vanishing or Exploding Gradients



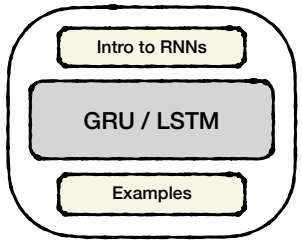
Repeatedly back-propagating over time through this part results in the gradients either **vanishing** to zero or **exploding** to infinite

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]

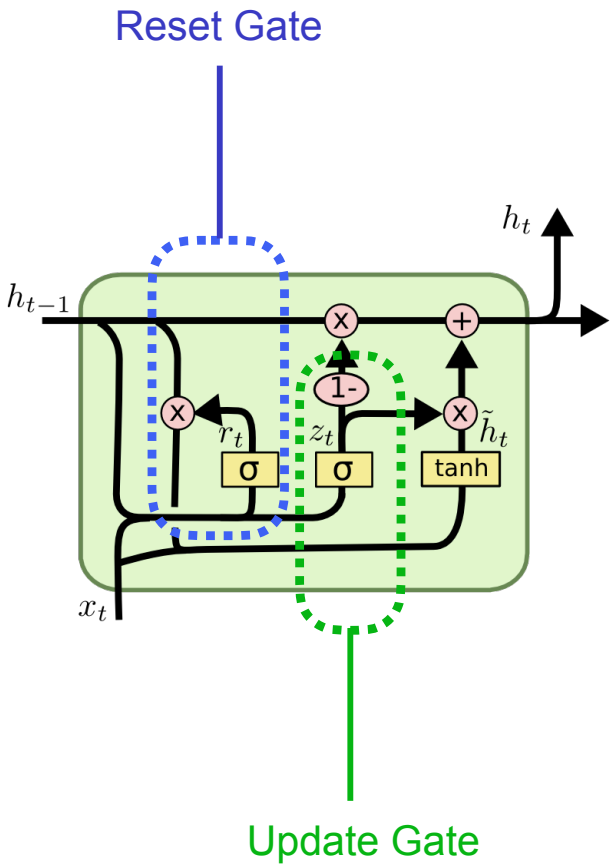
We can modify this section to improve on these shortcomings a bit. LSTMs and GRUs are some of the most common replacements for this section

[3] https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577



GRU (Gated Recurrent Unit)

Overview



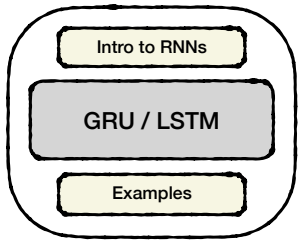
$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$

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

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

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

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

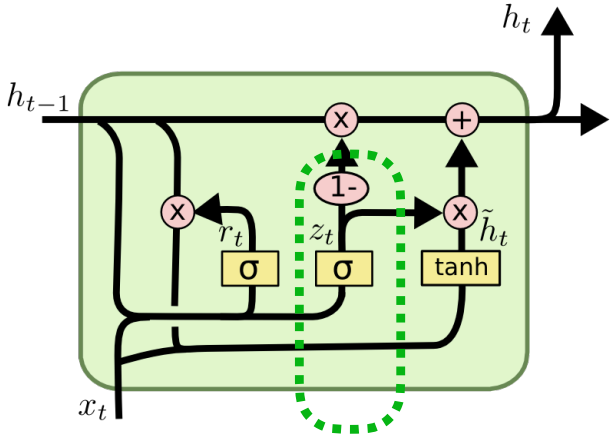


GRU (Gated Recurrent Unit)

How does the update gate work?

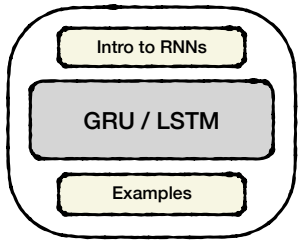
Update Gate is in charge of deciding how much of the information accumulated from past along with how much of information extracted from the current input should carry on to the next time-step!

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



Update Gate

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$
$$\tilde{h}_t = tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t$$

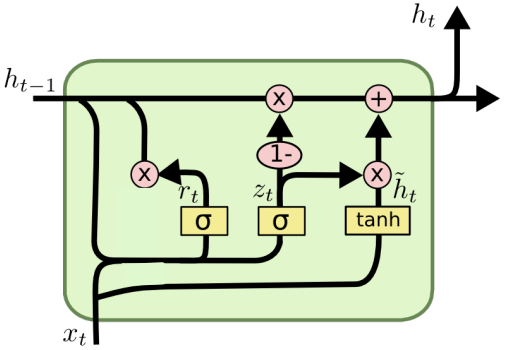


GRU (Gated Recurrent Unit)

How does the reset gate work?

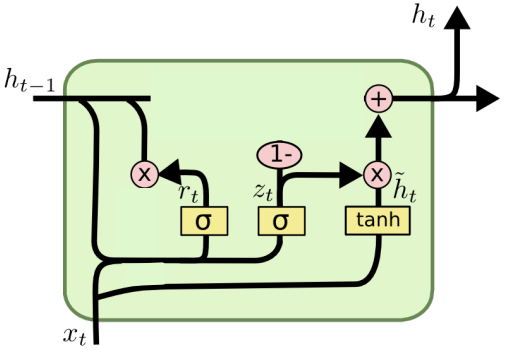
If the Reset Gate is fully active ($r_t = 0$), the new candidate for hidden state ($\tilde{h}_t$) will only depend on the existing input; just like the first iteration in the vanilla RNN case

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

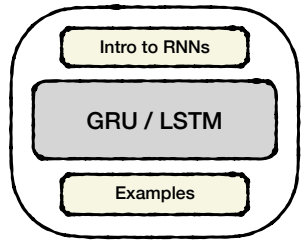


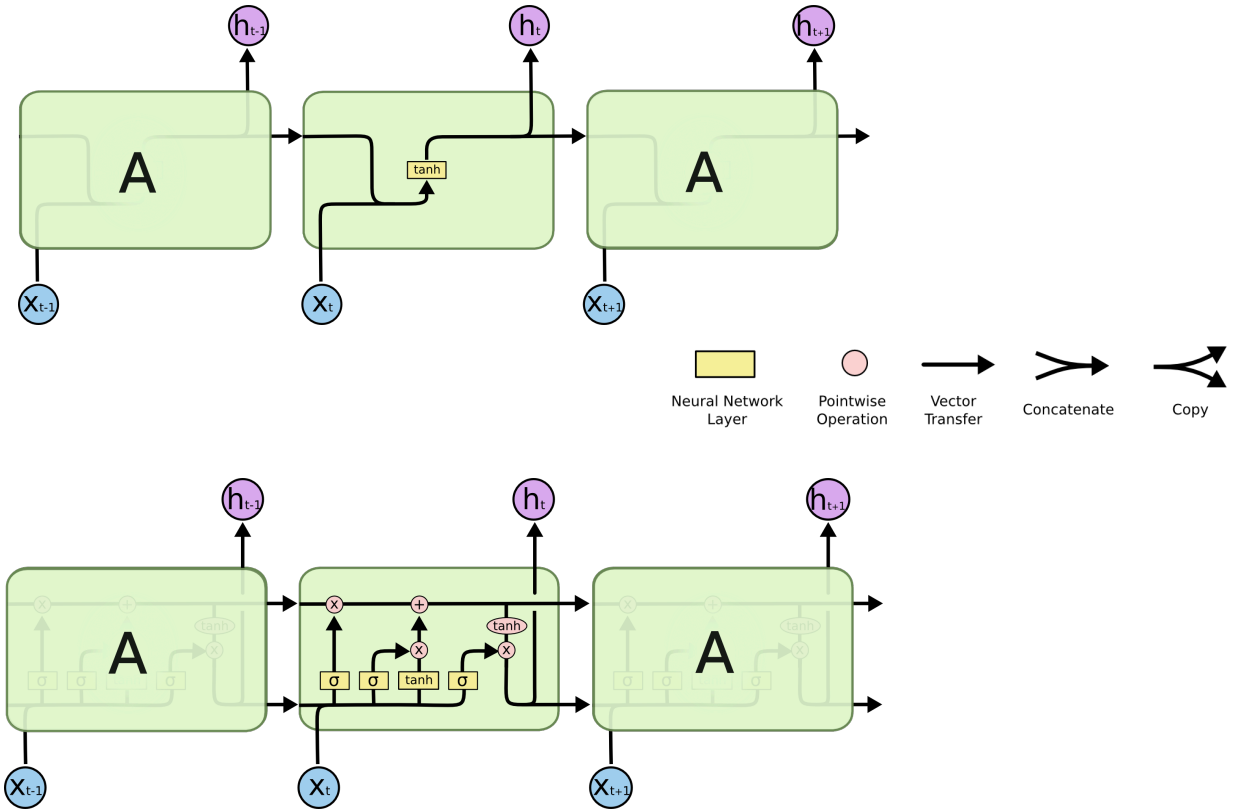
Now, if the update gate decides that the hidden state (h_t) should be replaced with the candidate ($\tilde{h}_t$) (i.e. $z_t = 1$), the new hidden state will carry no information from the past sequence!

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



LSTMs (Long Short-Term Memory)

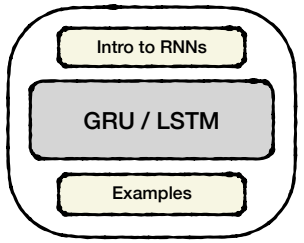




Figures from https://colah.github.io/posts/2015-08-Understanding-LSTMs/, for a detailed explanation refer to this source

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



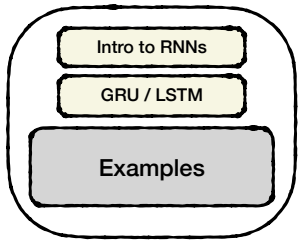
GRUs and LSTMs both:

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.

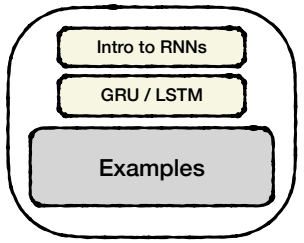
https://magenta.tensorflow.org/performance-rnn

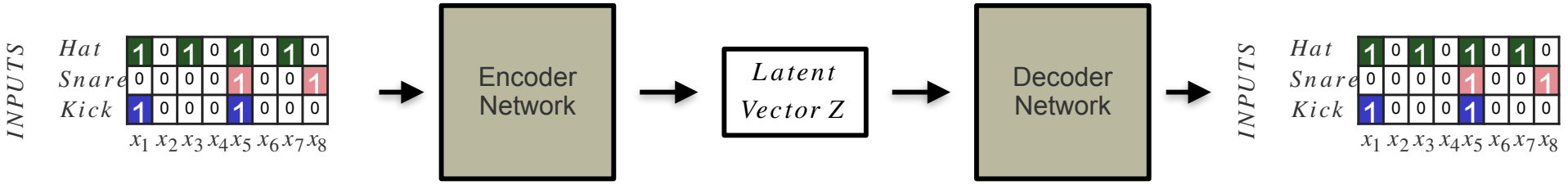
LSTMetallica

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/lstmetallica/

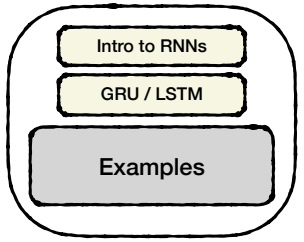
Auto-encoders for sequential data





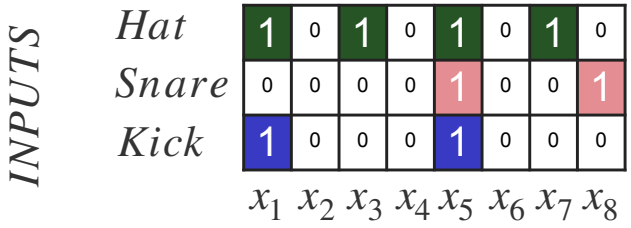
We want to encode an input **sequence** into a vector z and,
from which we decode/reconstruct the original input **sequence**

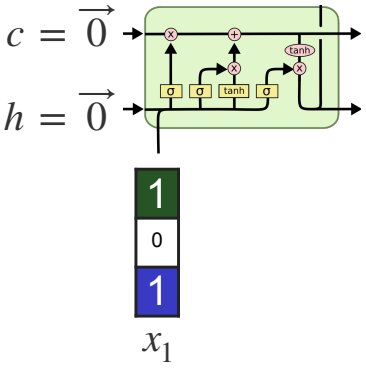
We can use **Sequence-to-Sequence (Seq2Seq)** architectures for this purpose

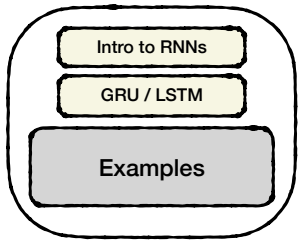


Sequence to Sequence Architecture (Auto-encoder for sequential data)

Step 1: Encoding the input

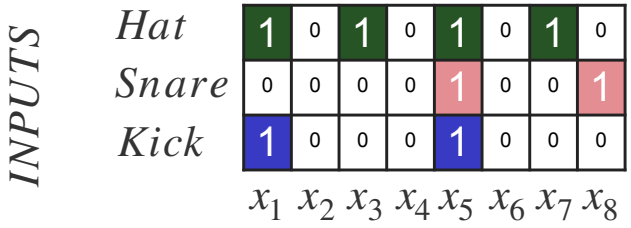


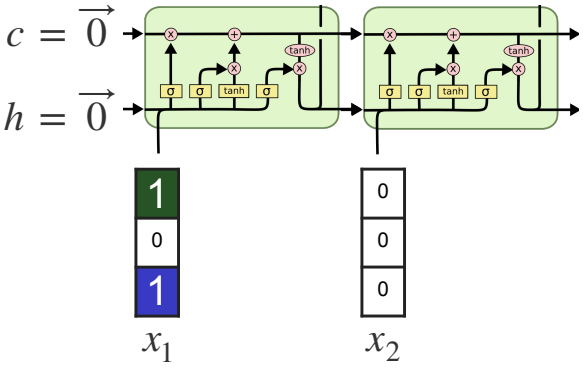


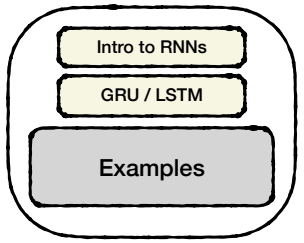


Sequence to Sequence Architecture (Auto-encoder for sequential data)

Step 1: Encoding the input

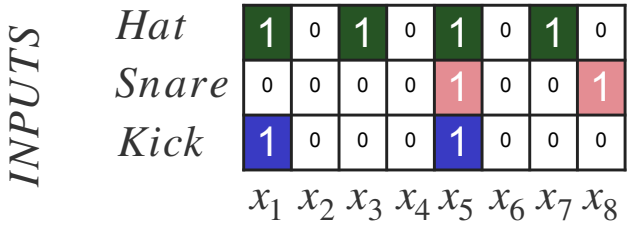


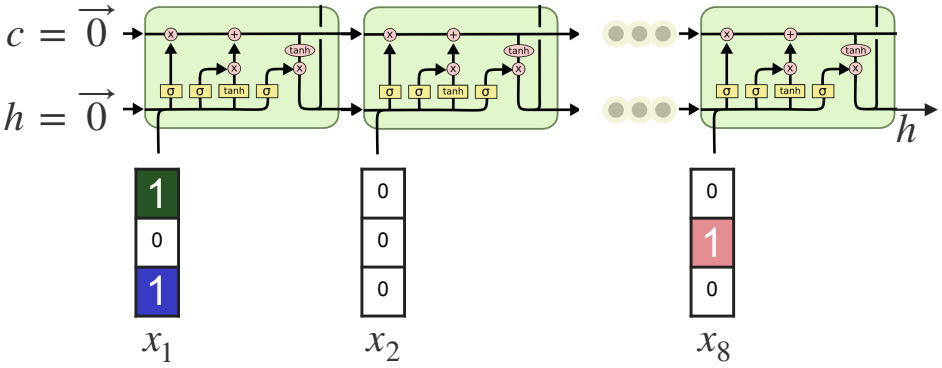


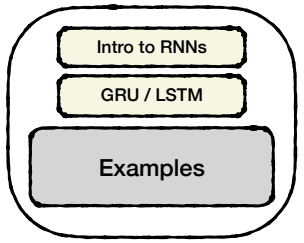


Sequence to Sequence Architecture (Auto-encoder for sequential data)

Step 1: Encoding the input

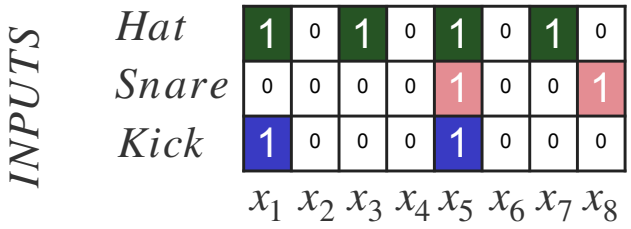


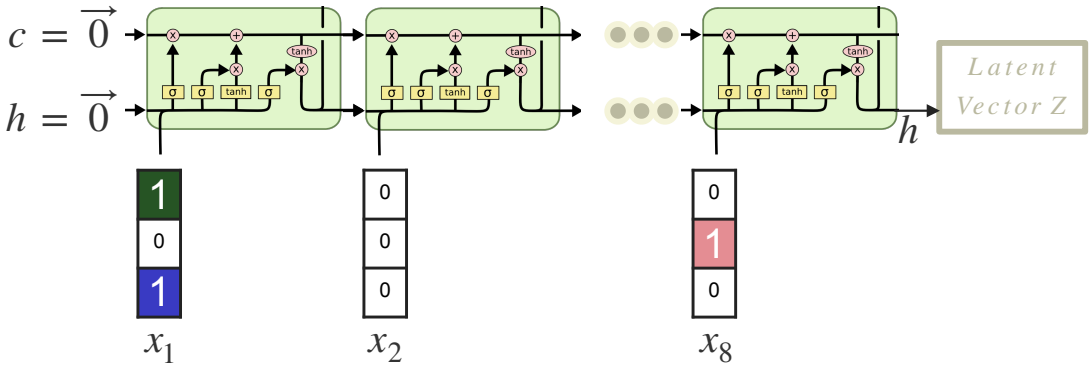




Sequence to Sequence Architecture (Auto-encoder for sequential data)

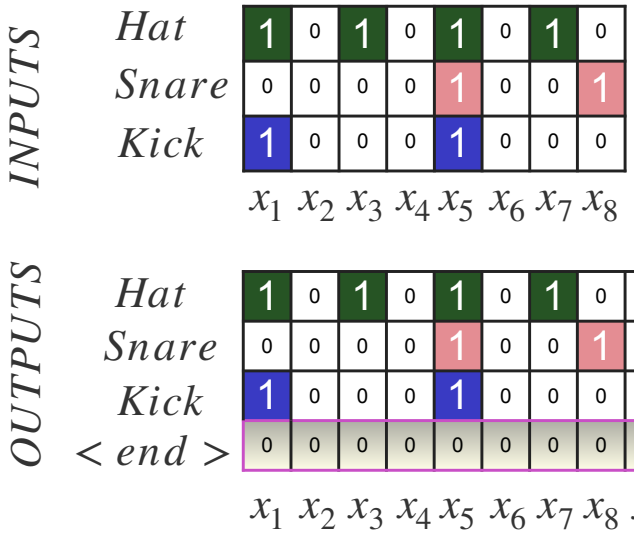
Step 2: Map last hidden vector to a latent vector Z



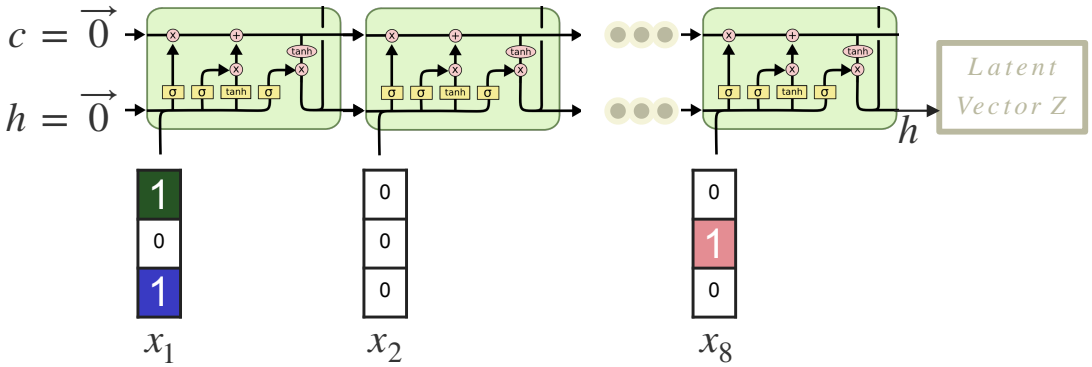


Sequence to Sequence Architecture (Auto-encoder for sequential data)

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

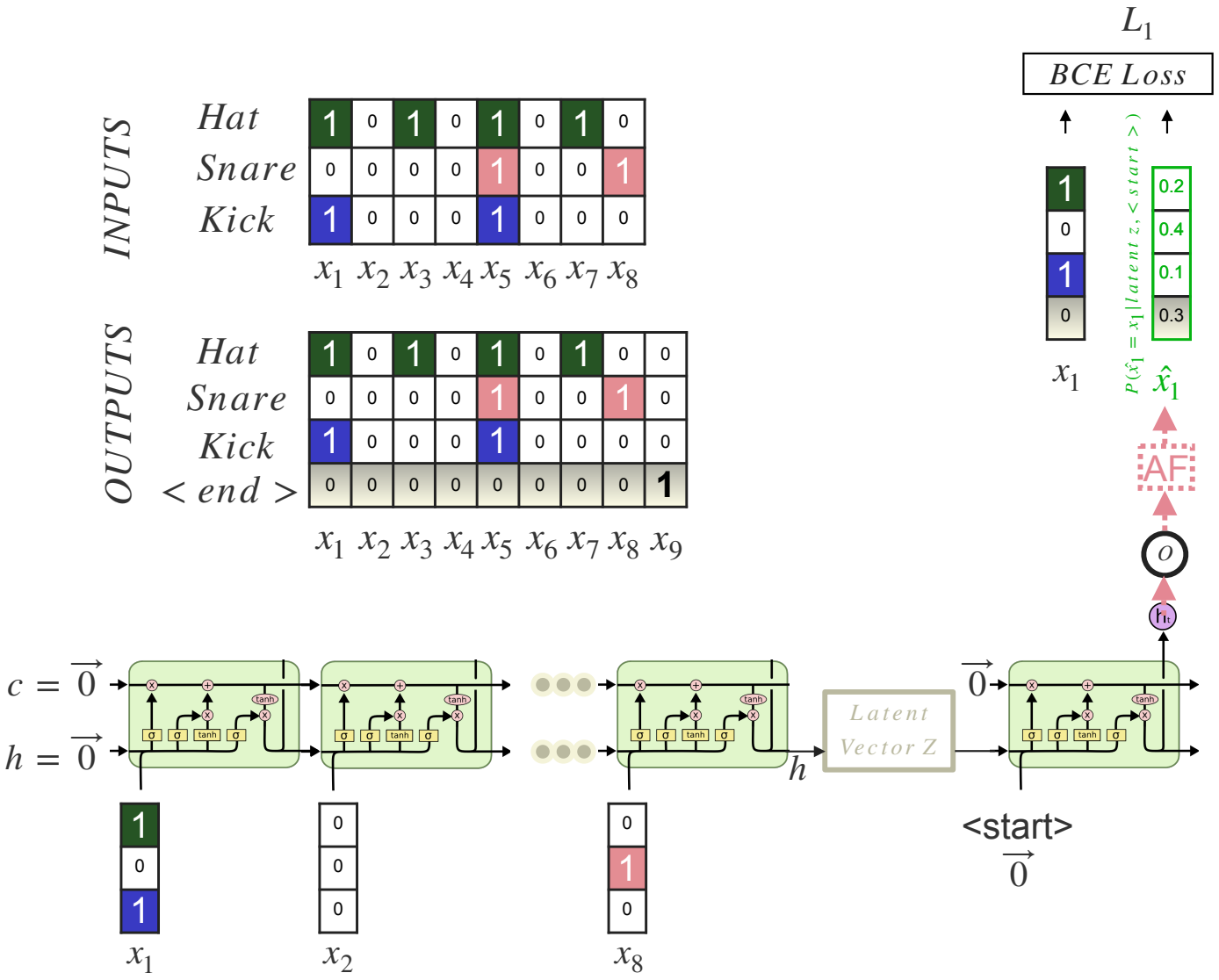


Added to denote The end of sequence; only needed when dealing with various sequence lengths



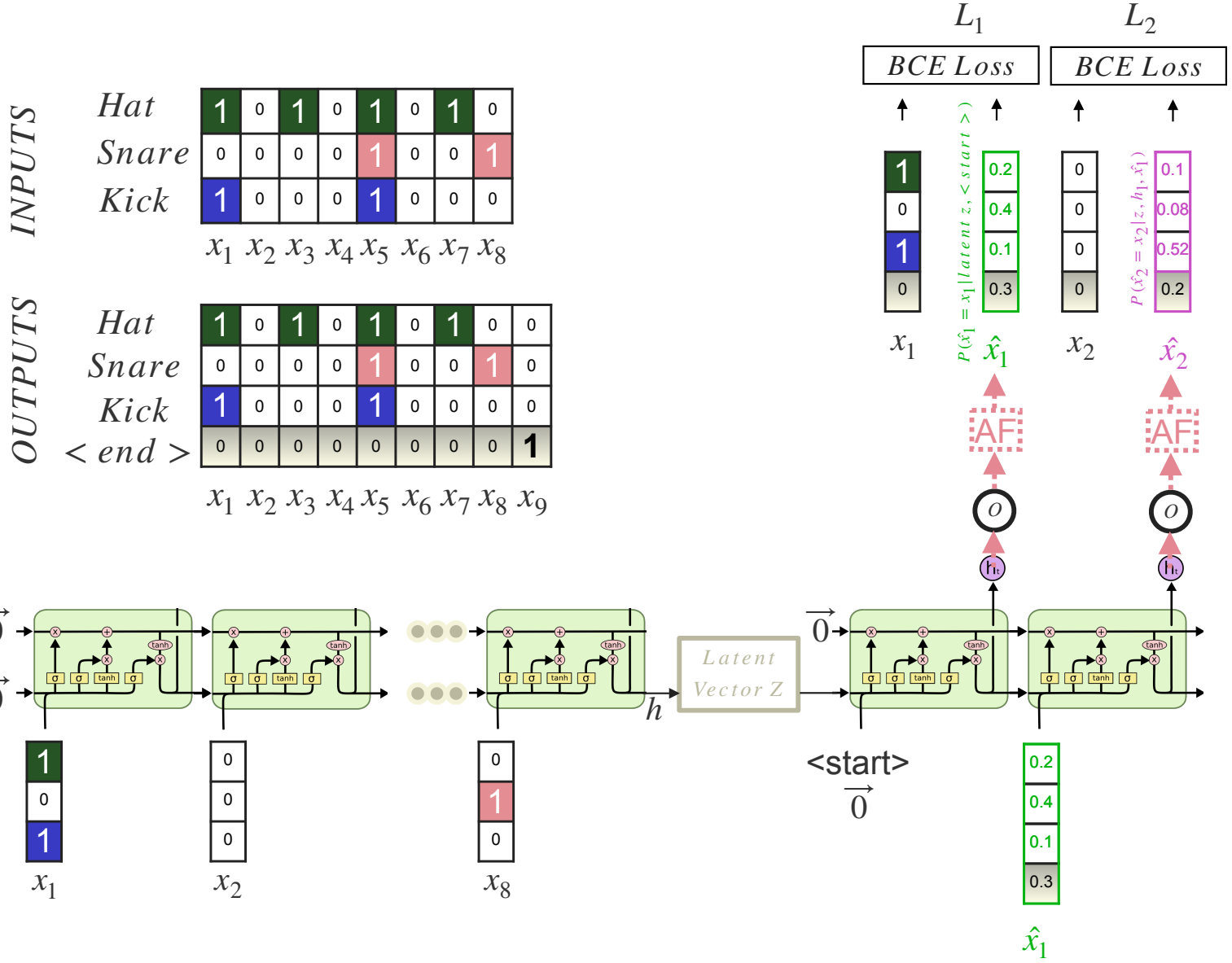
Sequence to Sequence Architecture (Auto-encoder for sequential data)

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



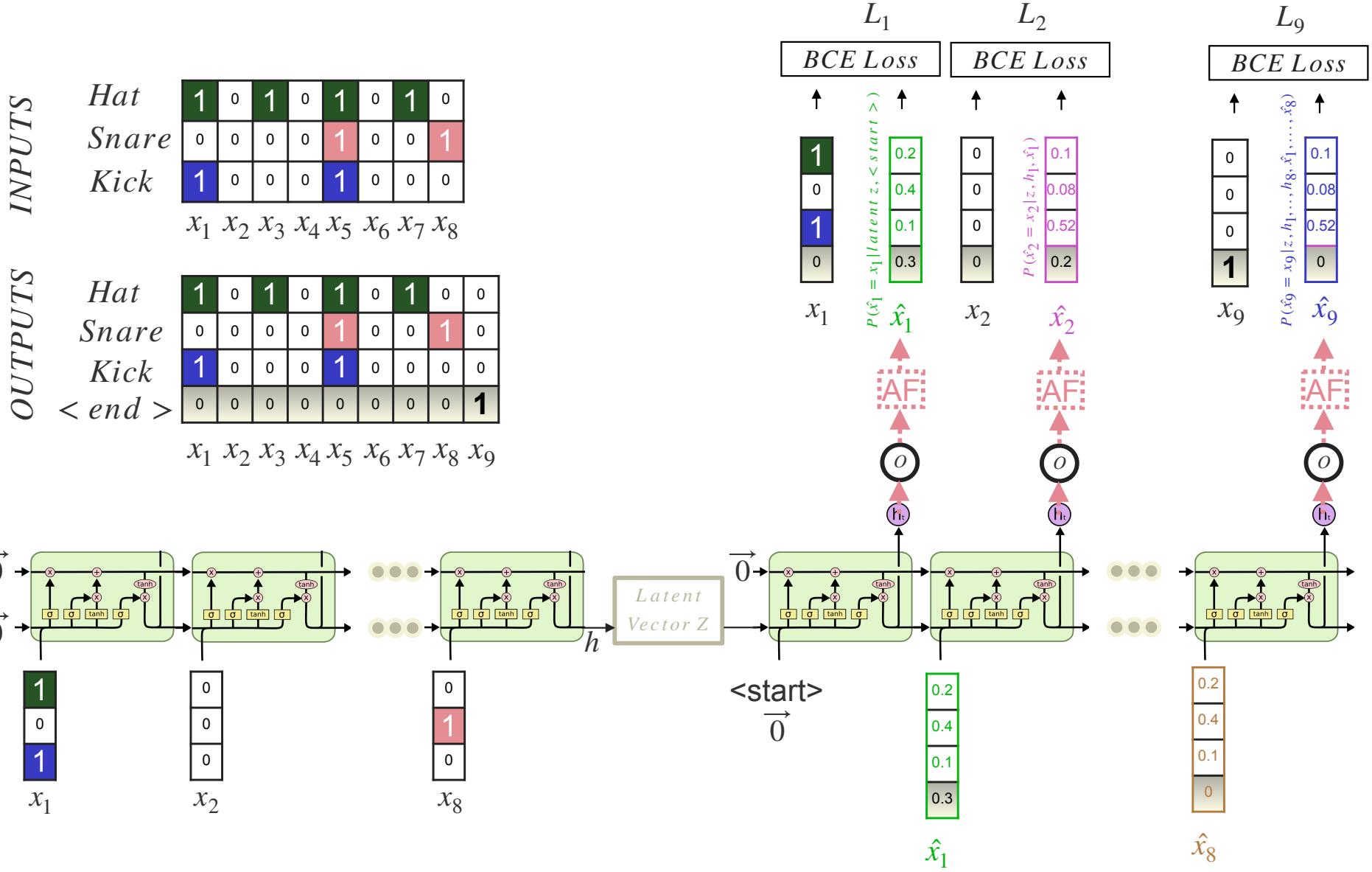
Sequence to Sequence Architecture (Auto-encoder for sequential data)

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



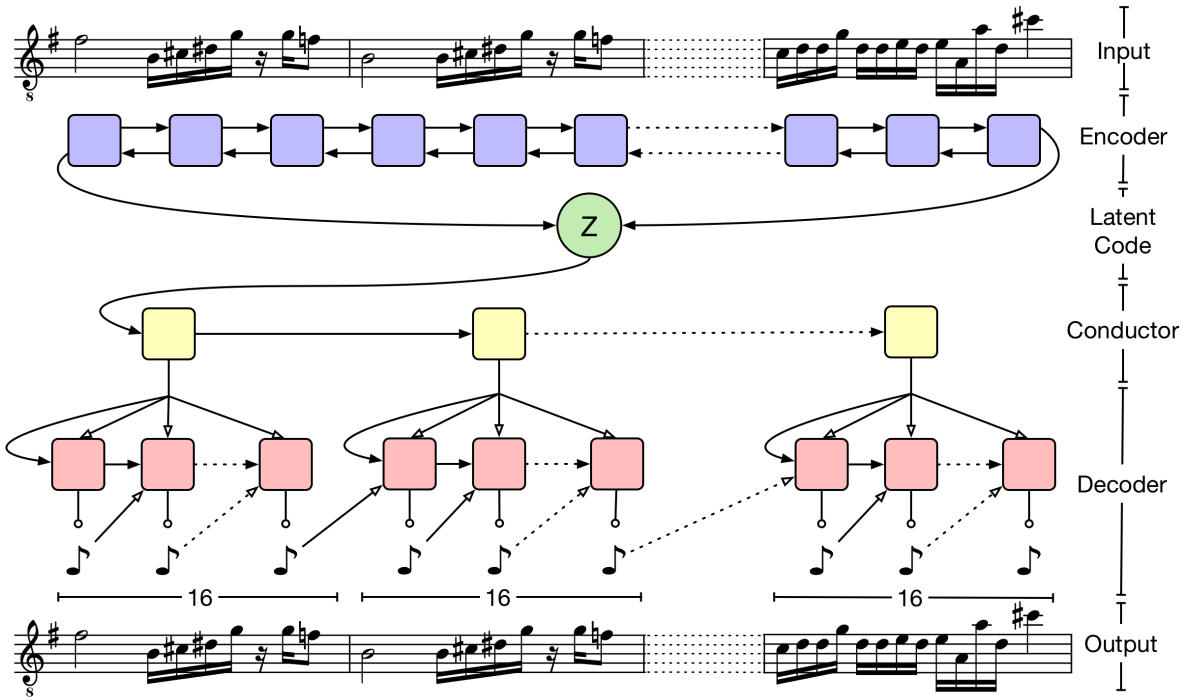
Sequence to Sequence Architecture (Auto-encoder for sequential data)

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



Next Sessions

We will talk about sequence to sequence architectures (using LSTMs) as well as sequence to sequence VAEs.



Source: https://magenta.tensorflow.org/music-vae