Text Processing using Machine Learning

Language Modelling

Liling Tan

04 Dec 2019

OVER
5,500 GRADUATE ALUMNI

OFFERING OVER
120 ENTERPRISE IT, INNOVATION & LEADERSHIP PROGRAMMES

TRAINING OVER
120,000 DIGITAL LEADERS & PROFESSIONALS

Overview

- **N-gram Language Modelling**
  - N-grams, Skip-grams, So many -grams
  - Perplexity and Entropy
  - Limitations

- **Neural Language Modelling**
  - Cost functions
  - Tricks to prevent overfitting

N-gram Language Modelling

Language Model

- Language Modelling is task of ***predicting which word comes next***

- ***Predict next word*** x_i, ***given all context words up till the current word***, $x_1, \dots, x_{i-1}$

- Language Model aka. ***assigning probability of text as an accumulated probability of all individual words given their contexts***

Language Model

Calculate Probability of a Sentence

$$P(X) = \prod_{i=1}^{I} P(\boldsymbol{x_i} | \boldsymbol{x_1}, \dots, \boldsymbol{x_{i-1}})$$

Next word **Context**

P("he likes to drink coffee")
= P(he) + P(likes | he) + P(to | he likes)
+ P(drink | he likes to) + P(coffee | he likes to drink)

Language Model

How to compute:

$$P(X) = \prod_{i=1}^{I} P(\boldsymbol{x_i} | \boldsymbol{x_1}, \dots, \boldsymbol{x_{i-1}})$$

Count("he likes to drink coffee")
= Count(he) * Count(likes | he) * Count(to | he likes)
* Count(drink | he likes to)
* Count(coffee | he likes to drink)

Language Model

How to compute:

$$P(X) = \prod_{i=1}^{I} P(x_i | x_1, \dots, x_{i-1})$$

Count("he likes to drink coffee")
= Count(he) * Count(likes | he) * Count(to | he likes)
  * Count(drink | he likes to)
  * Count(coffee | he likes to drink)

Too many possibilities!!!

Language Model

How to compute:

$$P(X) = \prod_{i=1}^{I} P(\boldsymbol{x_i} | \boldsymbol{x_1}, \ldots, \boldsymbol{x_{i-1}})$$

P(drink | he likes to) ≈ Count(drink | likes to)

P(coffee | he likes to drink) ≈ Count(coffee | to drink)

Ngram Language Model

Calculate Probability of a Sentence

$$P(X) = \prod_{i=1}^{I} P(\boldsymbol{x_i} | \boldsymbol{x_{i-n}}, \dots, \boldsymbol{x_{i-1}})$$

Next word

Context (specified window)

P("he likes to drink coffee")

= P(he) + P(likes | he) + P(to | he likes)

+ P(drink | likes to) + P(coffee | to drink)

Ngram Language Model

- Count and divide

$$P_{ML}(x_i \mid x_{i-n+1}, \ldots, x_{i-1}) := \frac{c(x_{i-n+1}, \ldots, x_i)}{c(x_{i-n+1}, \ldots, x_{i-1})}$$

- Add smoothing to deal with zero counts

$$\begin{aligned} P(x_i \mid x_{i-n+1}, \ldots, x_{i-1}) =& \lambda P_{ML}(x_i \mid x_{i-n+1}, \ldots, x_{i-1}) \\ &+ (1-\lambda) P(x_i \mid x_{1-n+2}, \ldots, x_{i-1}) \end{aligned}$$

Evaluating Language Model

- **Log-likelihood:**

$$LL(\mathcal{E}_{test}) = \sum_{E \in \mathcal{E}_{test}} \log P(E)$$

- **Per-word Log Likelihood:**

$$WLL(\mathcal{E}_{test}) = \frac{1}{\sum_{E \in \mathcal{E}_{test}} |E|} \sum_{E \in \mathcal{E}_{test}} \log P(E)$$

- **Per-word (Cross) Entropy:**

$$H(\mathcal{E}_{test}) = \frac{1}{\sum_{E \in \mathcal{E}_{test}} |E|} \sum_{E \in \mathcal{E}_{test}} -\log_2 P(E)$$

- **Perplexity:**

$$ppl(\mathcal{E}_{test}) = 2^{H(\mathcal{E}_{test})} = e^{-WLL(\mathcal{E}_{test})}$$

Perplexity

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

Normalized by number of words

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}}$$

Inverse probability of test set

Perplexity

- Maximizing probability = Minimizing perplexity

- **What is perplexity in Deep Learning models?**

NLLoss = $\ln \mathcal{L}(\theta) = -\sum_{i=1}^{n} y_i \ln(\hat{y})$

CE = $\mathcal{L}(\theta) = -\frac{1}{n}\sum_{i=1}^{n} y_i \ln(\hat{y})$

Lower is better!!!

Generating text with Ngram LM

While not </s>:

  calculate probability of possible next words

  choose a word from the top N most probable

<context> = “<s> he likes to”

P(drink | <context>) > P(coffee | <context>)

Limitations of Ngram Language Models

- Storage and retrieving ngrams probabilities
- Sparsity and smoothing hacks
- Context windows limits long-distance dependencies
- Frequencies says little about semantics

The Shannon Game

- Life is like a box of ______

- Life is too short to miss out on the beautiful things like a double _____

- Science is organized knowledge, wisdom is organized ____

The Shannon Game

- Life is like a box of **chocolate**
- Life is too short to miss out on the beautiful things like a double **cheeseburger**
- Science is organized knowledge, wisdom is organized **life**

Language Model Evaluation

- A language model that can ***predict the right words***, i.e. ***assign a higher probability to words that occurs*** is a better model

- Traditionally, LM is evaluated on **perplexity**

- Perplexity is the ***inverse probability of the test set***, normalized by the number of words

Environment Setup

Open Anaconda Navigator.

Go to the PyTorch installation page, copy the command as per configuration: https://pytorch.org/get-started/locally/

Fire up the terminal in Anaconda Navigator.

Start a Jupyter Notebook.

Download http://bit.ly/ANLP-Session5NgramLM

Import the .ipynb to the Jupyter Notebook

RNN vs N-gram Language Model

N-gram model

RNN with increased complexity

Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

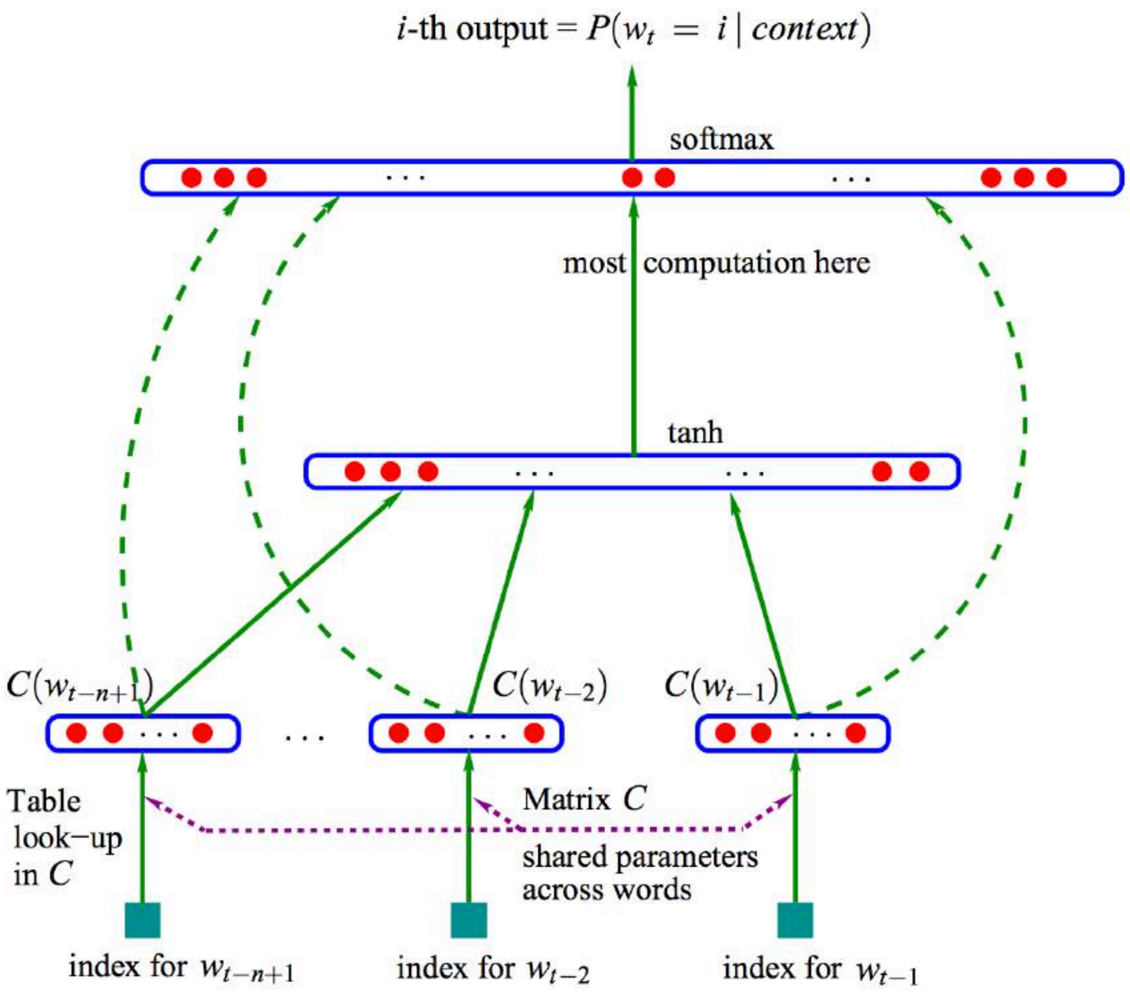
From https://research.fb.com/building-an-efficient-neural-language-model-over-a-billion-words/

Neural Language Modelling

Deep Language Models

- Calculate weights/features of context
- Based on the weights, compute probabilities
- Optimize weights using gradient descent to minimize errors on probabilities computation

A Neural Probabilistic Language Model



(Bengio et al. 2004)

Figure 1: Neural architecture: $f(i, w_{t-1}, \cdots, w_{t-n+1}) = g(i, C(w_{t-1}), \cdots, C(w_{t-n+1}))$ where g is the neural network and $C(i)$ is the i-th word feature vector.

Neural Language Model

likes $\begin{bmatrix} 2.5 \\ 1.9 \\ 1.3 \\ -0.2 \\ 2.8 \end{bmatrix}$ to $\begin{bmatrix} 1.9 \\ -1.2 \\ 1.3 \\ -0.3 \\ 0.8 \end{bmatrix}$ $\longrightarrow$ $\boxed{\tanh(W_1 x + b_1)} = \begin{bmatrix} 1.5 \\ -0.7 \\ 1.5 \\ -1.6 \\ 1.4 \end{bmatrix}$

scores | probs

$\boxed{W_2} \begin{bmatrix} 1.5 \\ -0.7 \\ 1.5 \\ -1.6 \\ 1.4 \end{bmatrix} = \begin{bmatrix} 0.3 \\ -0.1 \\ 1.9 \\ -0.5 \\ 2.7 \end{bmatrix} \rightarrow \boxed{\text{Softmax}} \rightarrow \begin{bmatrix} 0.06 \\ 0.04 \\ 0.27 \\ 0.02 \\ 0.61 \end{bmatrix}$ drink

Neural Language Model

likes $\begin{bmatrix} 2.5 \\ 1.9 \\ 1.3 \\ -0.2 \\ 2.8 \end{bmatrix}$ to $\begin{bmatrix} 1.9 \\ -1.2 \\ 1.3 \\ -0.3 \\ 0.8 \end{bmatrix}$ $\rightarrow$ $\boxed{\tanh(W_1 x + b_1)}$ $= \begin{bmatrix} 1.5 \\ -0.7 \\ 1.5 \\ -1.6 \\ 1.4 \end{bmatrix}$

Lookup function

scores

probs

$$\boxed{W_2} \begin{bmatrix} 1.5 \\ -0.7 \\ 1.5 \\ -1.6 \\ 1.4 \end{bmatrix} = \begin{bmatrix} 0.3 \\ -0.1 \\ 1.9 \\ -0.5 \\ 2.7 \end{bmatrix} \rightarrow \boxed{\text{Softmax}} \rightarrow \begin{bmatrix} 0.06 \\ 0.04 \\ 0.27 \\ 0.02 \\ 0.61 \end{bmatrix}$$

drink

Neural Language Model

likes $\begin{bmatrix} 2.5 \\ 1.9 \\ 1.3 \\ -0.2 \\ 2.8 \end{bmatrix}$ to $\begin{bmatrix} 1.9 \\ -1.2 \\ 1.3 \\ -0.3 \\ 0.8 \end{bmatrix}$ $\longrightarrow$ $\boxed{\tanh(W_1 x + b_1)} = \begin{bmatrix} 1.5 \\ -0.7 \\ 1.5 \\ -1.6 \\ 1.4 \end{bmatrix}$

Transform

$$\boxed{W_2} \begin{bmatrix} 1.5 \\ -0.7 \\ 1.5 \\ -1.6 \\ 1.4 \end{bmatrix} = \underbrace{\begin{bmatrix} 0.3 \\ -0.1 \\ 1.9 \\ -0.5 \\ 2.7 \end{bmatrix}}_{\text{scores}} \rightarrow \boxed{\text{Softmax}} \rightarrow \underbrace{\begin{bmatrix} 0.06 \\ 0.04 \\ 0.27 \\ 0.02 \\ 0.61 \end{bmatrix}}_{\text{probs}}$$

drink

Neural Language Model

likes $\begin{bmatrix} 2.5 \\ 1.9 \\ 1.3 \\ -0.2 \\ 2.8 \end{bmatrix}$ to $\begin{bmatrix} 1.9 \\ -1.2 \\ 1.3 \\ -0.3 \\ 0.8 \end{bmatrix}$ $\longrightarrow$ $\boxed{\tanh(W_1 x + b_1)} = \begin{bmatrix} 1.5 \\ -0.7 \\ 1.5 \\ -1.6 \\ 1.4 \end{bmatrix}$

Predict

$$\boxed{W_2 \begin{bmatrix} 1.5 \\ -0.7 \\ 1.5 \\ -1.6 \\ 1.4 \end{bmatrix}} = \underbrace{\begin{bmatrix} 0.3 \\ -0.1 \\ 1.9 \\ -0.5 \\ 2.7 \end{bmatrix}}_{\text{scores}} \rightarrow \boxed{\text{Softmax}} \rightarrow \underbrace{\begin{bmatrix} 0.06 \\ 0.04 \\ 0.27 \\ 0.02 \\ 0.61 \end{bmatrix}}_{\text{probs}}$$

drink

Neural Language Model

likes $\begin{bmatrix} 2.5 \\ 1.9 \\ 1.3 \\ -0.2 \\ 2.8 \end{bmatrix}$ to $\begin{bmatrix} 1.9 \\ -1.2 \\ 1.3 \\ -0.3 \\ 0.8 \end{bmatrix}$ $\longrightarrow$ $\boxed{\tanh(W_1 x + b_1)}$ $= \begin{bmatrix} 1.5 \\ -0.7 \\ 1.5 \\ -1.6 \\ 1.4 \end{bmatrix}$ Similar context gets similar hidden states

Similar words gets similar vector

Similar outputs gets similar rows in the softmax matrix

$$\boxed{W_2} \begin{bmatrix} 1.5 \\ -0.7 \\ 1.5 \\ -1.6 \\ 1.4 \end{bmatrix} = \underbrace{\begin{bmatrix} 0.3 \\ -0.1 \\ 1.9 \\ -0.5 \\ 2.7 \end{bmatrix}}_{\text{scores}} \rightarrow \boxed{\text{Softmax}} \rightarrow \underbrace{\begin{bmatrix} 0.06 \\ 0.04 \\ 0.27 \\ 0.02 \\ 0.61 \end{bmatrix}}_{\text{probs}}$$

drink

Neural Language Model

likes $\begin{bmatrix} 0.5 \\ 2.9 \\ 1.3 \\ -2.2 \\ 2.8 \end{bmatrix}$ to $\begin{bmatrix} 0.9 \\ -0.2 \\ 1.7 \\ -5.3 \\ 2.3 \end{bmatrix}$ $\longrightarrow$ $\boxed{\tanh(W_1 x + b_1)}$ $= \begin{bmatrix} 1.5 \\ -0.7 \\ 1.5 \\ -1.6 \\ 1.4 \end{bmatrix}$

Simiilar context gets similar hidden states

Similar words gets similar vector

Look up from W_2 matrix (Press et al. 2016)

Similar outputs gets similar rows in the softmax matrix

$$\boxed{W_2} \begin{bmatrix} 1.5 \\ -0.7 \\ 1.5 \\ -1.6 \\ 1.4 \end{bmatrix} = \underbrace{\begin{bmatrix} 0.3 \\ -0.1 \\ 1.9 \\ -0.5 \\ 2.7 \end{bmatrix}}_{\text{scores}} \rightarrow \boxed{\text{Softmax}} \rightarrow \underbrace{\begin{bmatrix} 0.06 \\ 0.04 \\ 0.27 \\ 0.02 \\ 0.61 \end{bmatrix}}_{\text{probs}}$$

drink

Loss Function for Language Models

Loss function on step t is usual cross-entropy between our predicted probability distribution $\hat{\boldsymbol{y}}^{(t)}$, and the true next word $\boldsymbol{y}^{(t)} = \boldsymbol{x}^{(t+1)}$:

$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{j=1}^{|V|} y_j^{(t)} \log \hat{y}_j^{(t)}$$

Average this to get overall loss for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta)$$

Deep Language Models

- Calculate weights/features of context
- Based on the weights, compute probabilities
- Optimize weights using gradient descent to minimize errors on probabilities computation
- **When do we stop?** If we have softmax as the last layer activation, cost function is most probably?

Cost Function for Neural LM

- Negative Log Loss Function (`torch.nn.NLLoss`)
- Maximum Likelihood Estimation (MLE) ==
  Minimize the NLLoss of all sentences
- i.e. Find the parameters that make the sentences in the training data most likely

Tricks to Prevent Overfitting

- **Shuffle the input sentences**
  - Imagine seeing "*Justin Bieber is the best singer, baby, baby, baby, oh*" 100x at the start of the corpus…

- **Early stopping** based on some validation set

- **Dropout** during training

Tricks to Prevent Overfitting

- **Shuffle the input sentences**
  - Imagine seeing “*Justin Bieber is the best singer, baby, baby, baby, oh*” 100x at the start of the corpus…
- **Early stopping** based on some validation set
- **Dropout** during training
- **Mini-batching** makes training much faster

Mini-batching

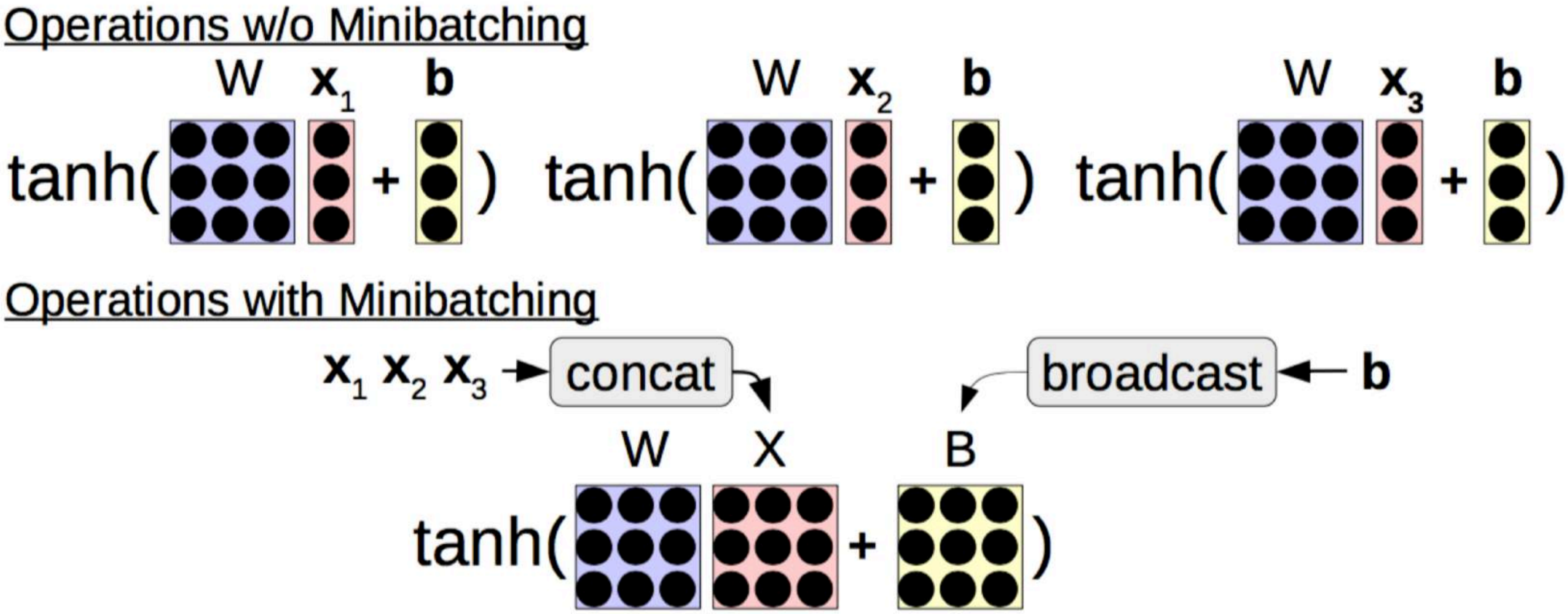


Image from CMU NN4NLP Course (Neubig, 2019)

Summary

LM + RNN Knowledge Checklist

- **Language Modelling**
  - LM is predicting the next word given history
  - Negative Log Loss as cost function for simple LM
  - Shuffle data, stop early on validation set, always dropout

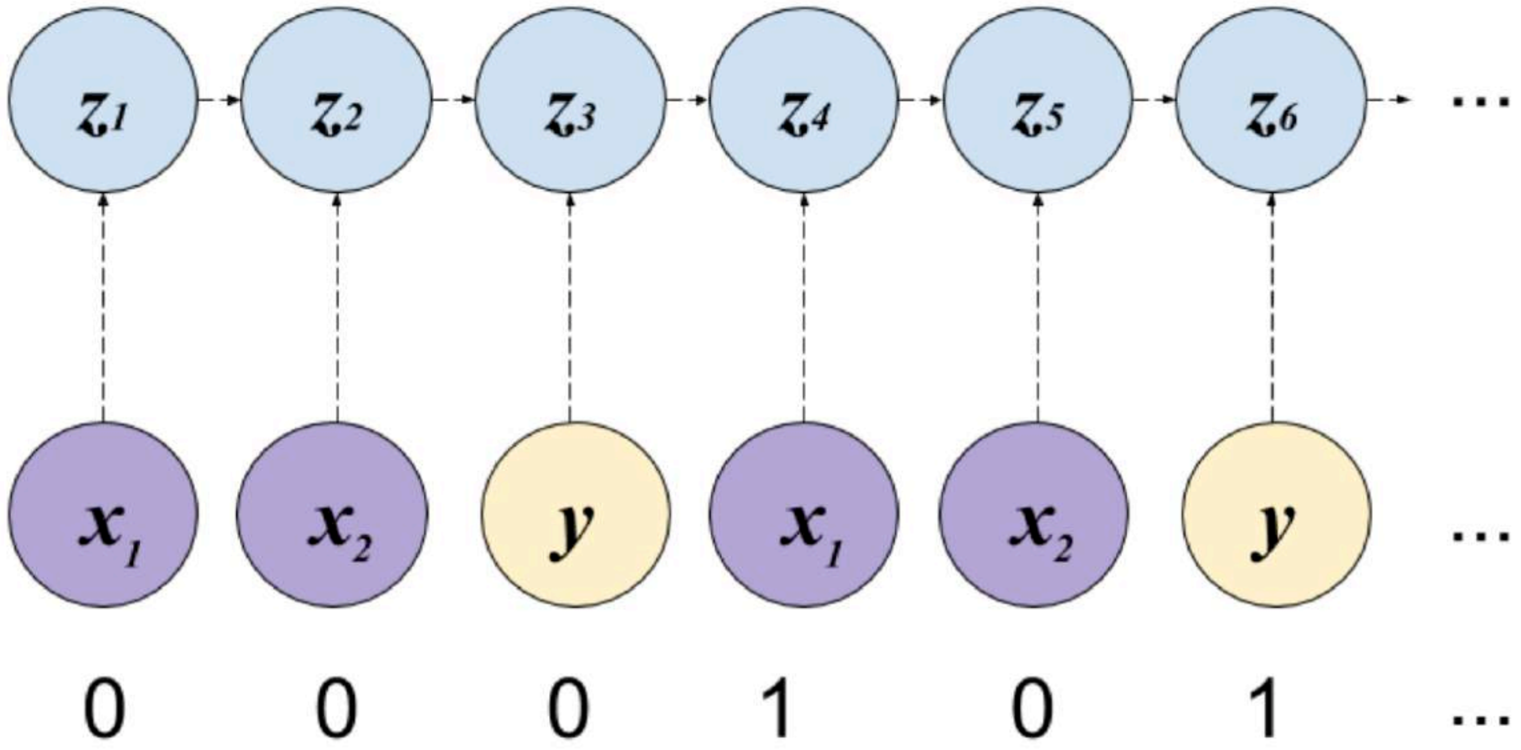
Recurrent Neural Nets

(*Almost*) Everything is a Sequence in Language

- Natural language is full of sequential data
- Word == sequence of characters
- Sentence == sequence of words
- Dialog/Discourse == sequence of sentences

Recurrent Neural Net (RNN)

(Input + **Prev_Hidden**) -> Hidden -> Output



(Elman, 1990)

Recurrent Neural Net (RNN)

(Input + Empty_Hidden) -> Hidden -> Output

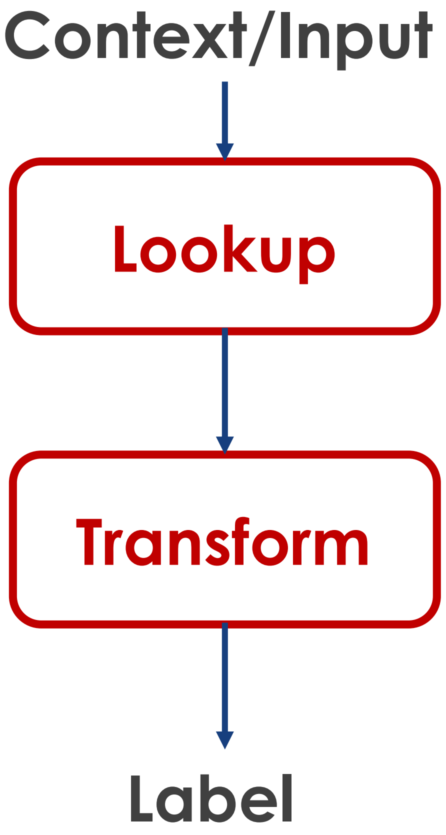
(Input + Prev_Hidden) -> Hidden -> Output

(Input + Prev_Hidden) -> Hidden -> Output

(Input + Prev_Hidden) -> Hidden -> Output

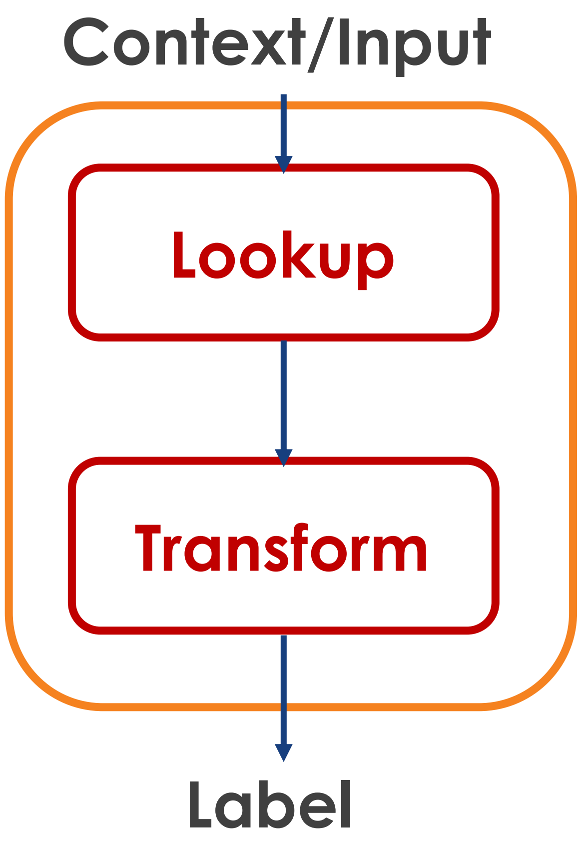
(Source: A really good blogpost on RNN by Trask: https://iamtrask.github.io/2015/11/15/anyone-can-code-lstm/)

Feed-Forward Neural Nets




```
x = torch.tensors([0,14,29,…])

embedding = nn.Embedding()
embed = embedding(x)

lin = nn.Linear()
prediction = lin(embed)

x = torch.max(prediction)
```

Feed-Forward Neural Nets




```
x = torch.tensors([0,14,29,…])
```

```
model = nn.Sequential(...)
prediction = model(x)
```

```
x = torch.max(prediction)
```

PyTorch RNNCell Module

RNNCell

CLASS `torch.nn.RNNCell(input_size, hidden_size, bias=True, nonlinearity='tanh')` [SOURCE]

An Elman RNN cell with tanh or ReLU non-linearity.

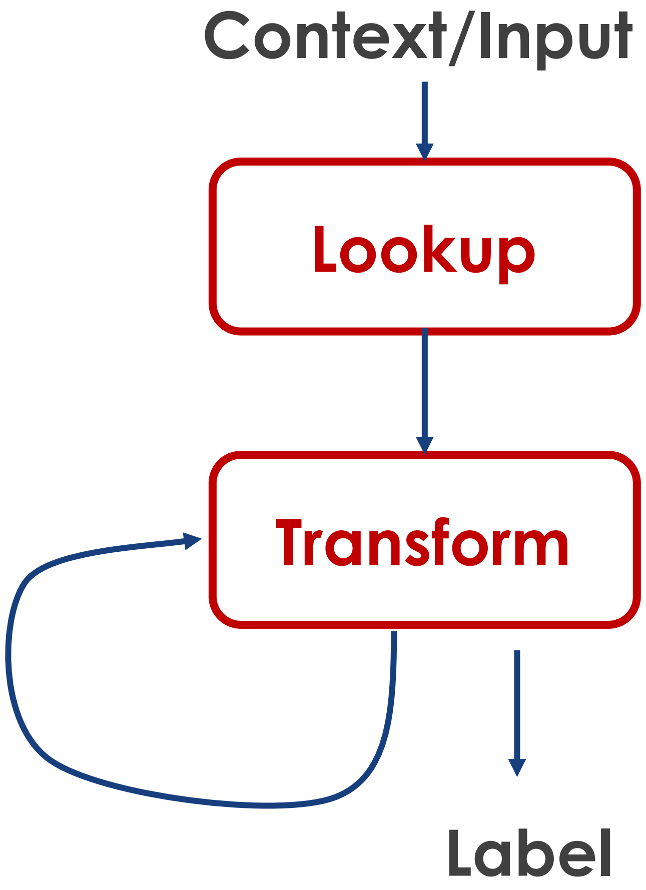
$$h' = \tanh(w_{ih} x + b_{ih} + w_{hh} h + b_{hh})$$

If `nonlinearity` is *'relu'*, then ReLU is used in place of tanh.

Parameters:
- **input_size** – The number of expected features in the input *x*
- **hidden_size** – The number of features in the hidden state *h*
- **bias** – If `False`, then the layer does not use bias weights *b_ih* and *b_hh*. Default: `True`
- **nonlinearity** – The non-linearity to use. Can be either 'tanh' or 'relu'. Default: 'tanh'

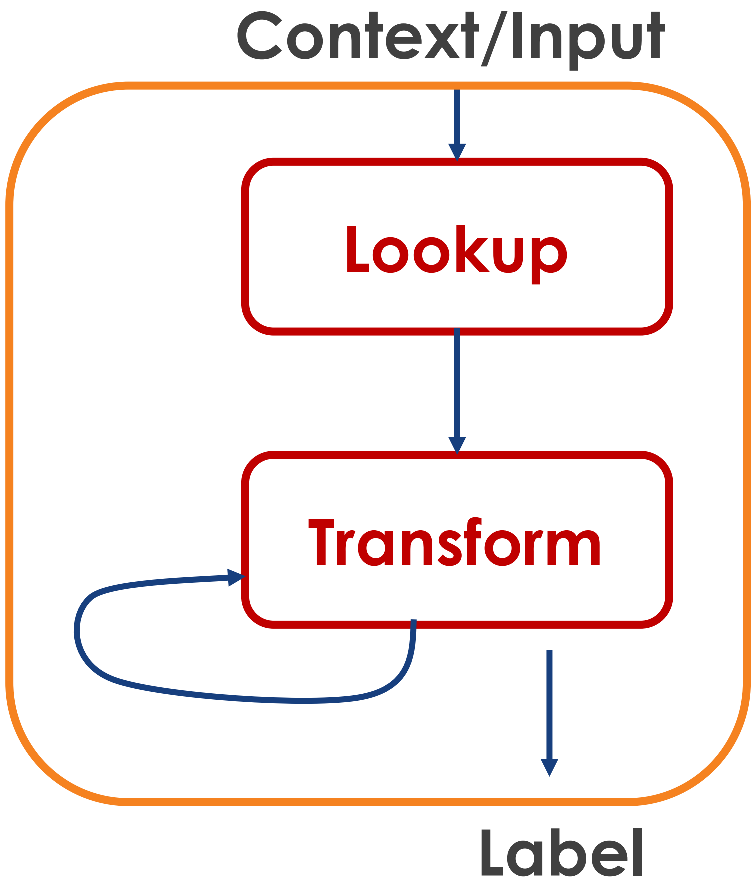
From PyTorch documentation https://pytorch.org/docs/stable/nn.html#rnncell

Semi Hand-rolled Recurrent Neural Nets (RNN)



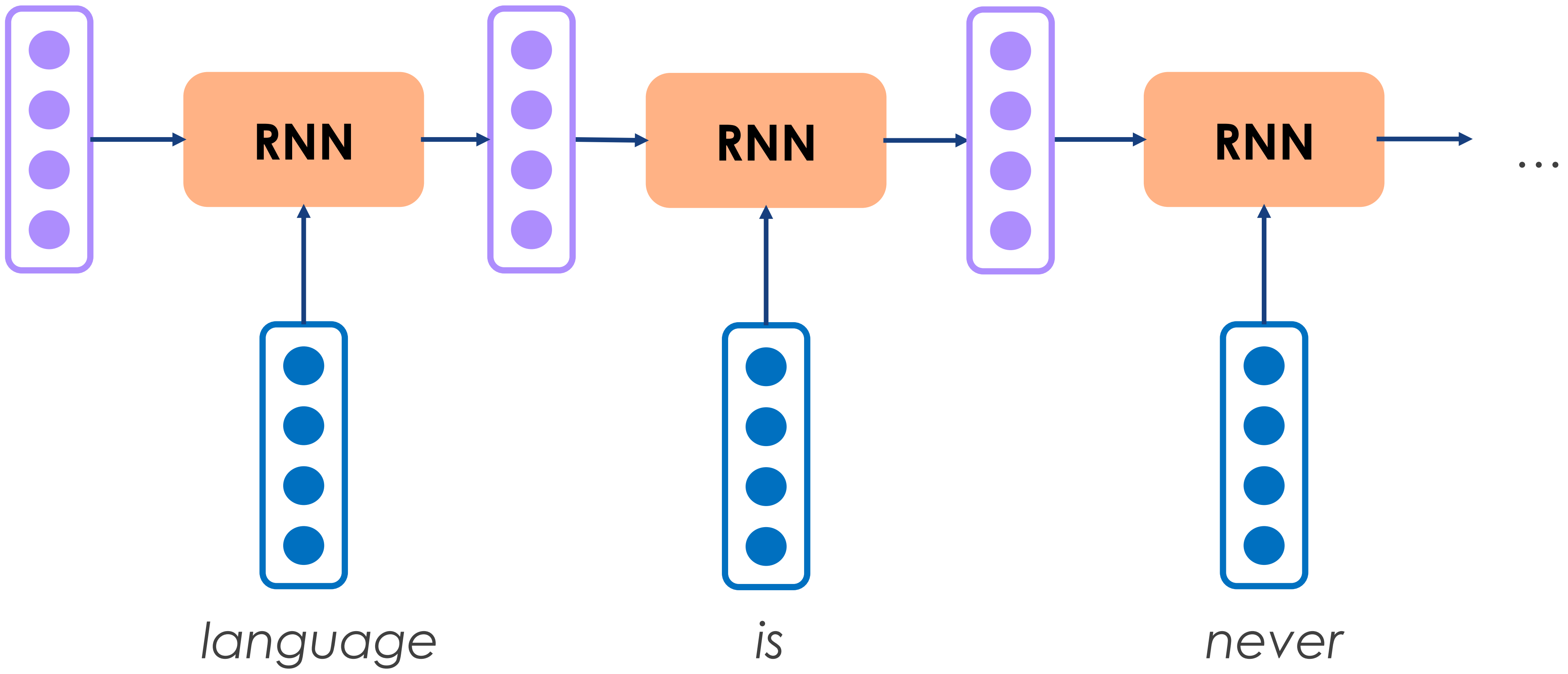

```
x = torch.tensors([0,14,29,…])
embedding = nn.Embedding()
model = nn.Sequential(...)
rnn = nn.RNNCell()
hiddens = []
for x_i in x:
   embed = embedding(x_i)
   hiddens.append(rnn(embed))
prediction = model(hiddens.flatten())
x = torch.max(prediction)
```

Recurrent Neural Nets (RNN) with PyTorch

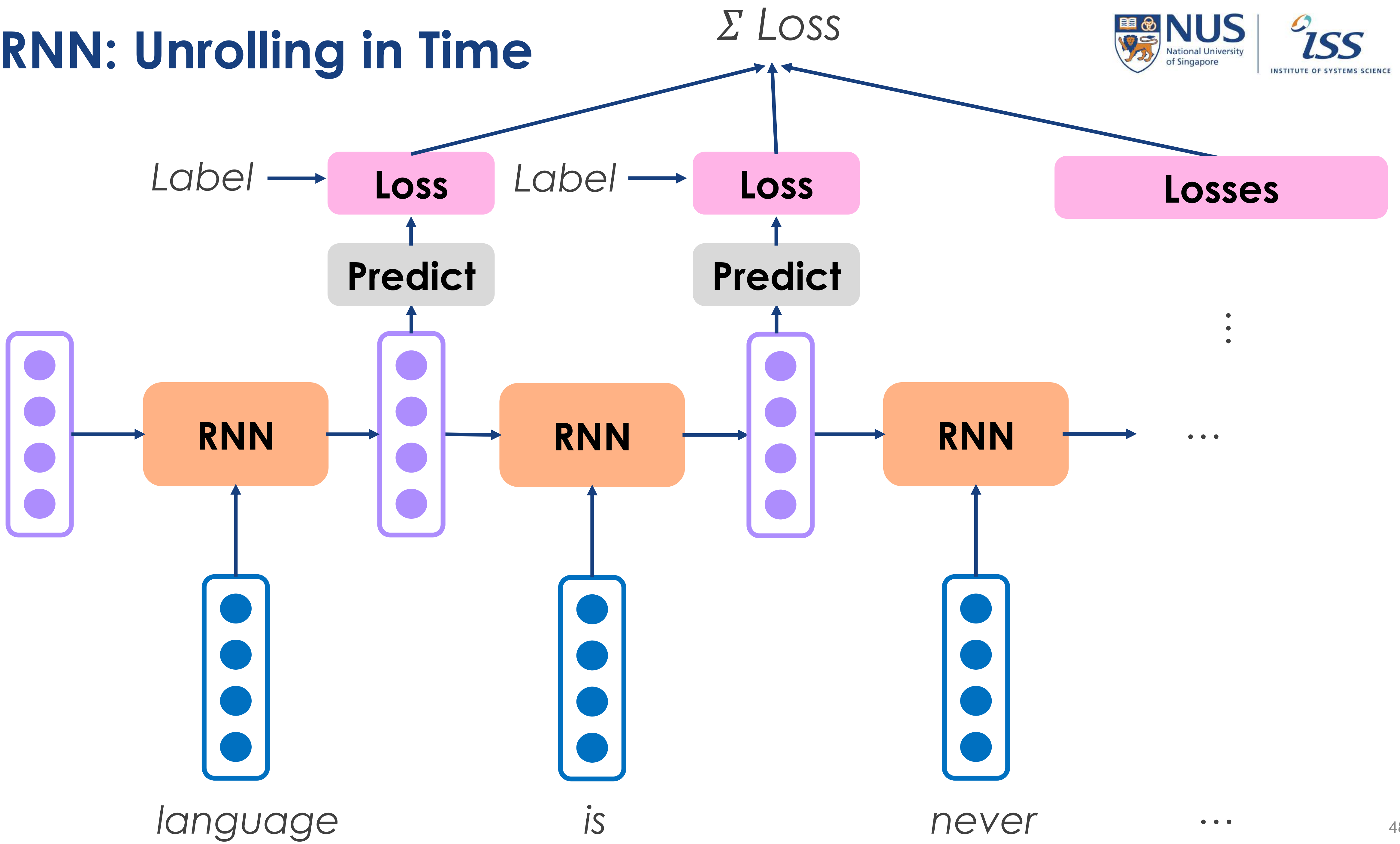



```
x = torch.tensors([0,14,29,…])

model = nn.Sequential(
            nn.Embedding, nn.RNN,
            nn.Linear, nn.ReLU)
prediction = model(x)

x = torch.max(prediction)
```

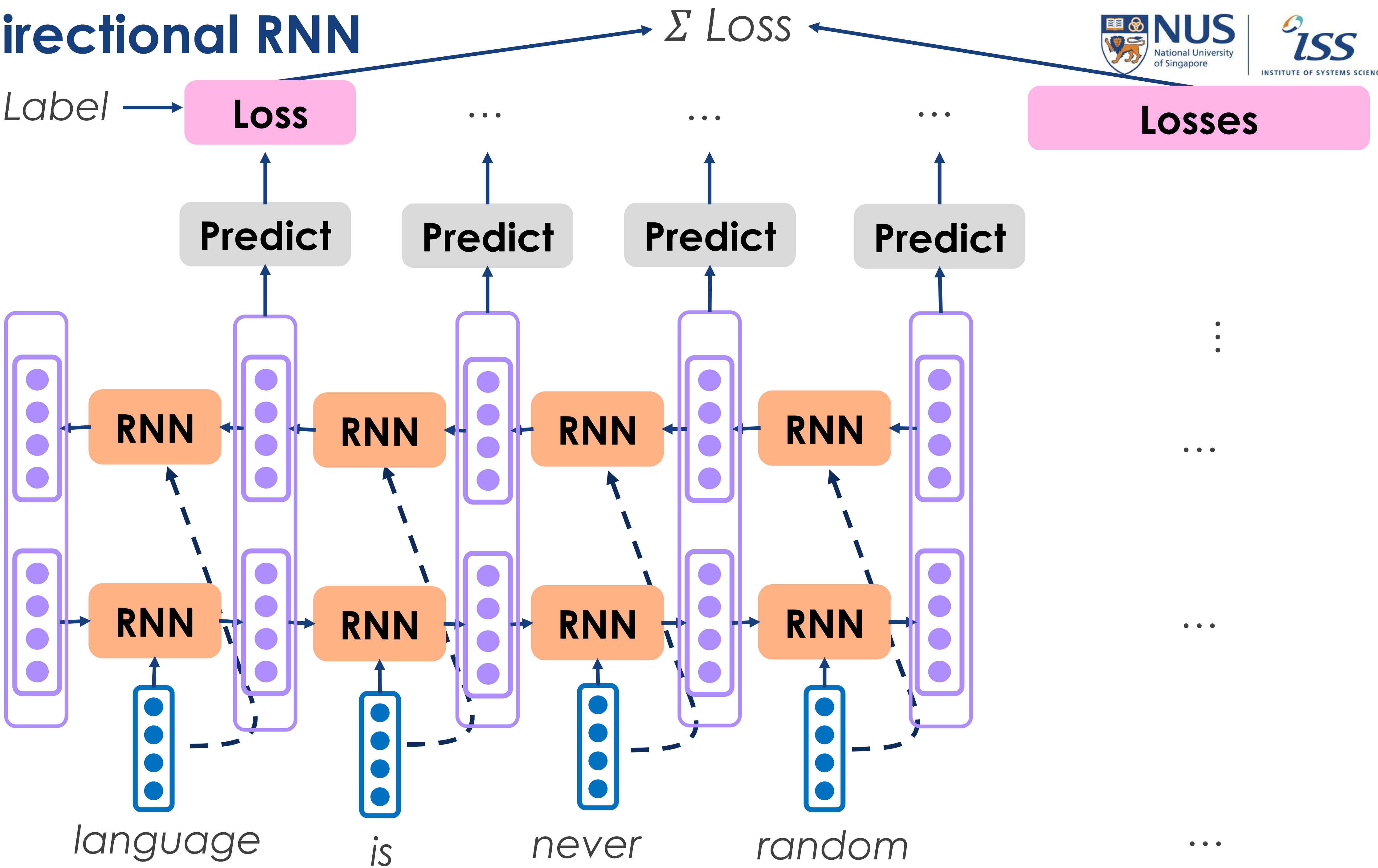
RNN: Unrolling in Time



RNN: Unrolling in Time



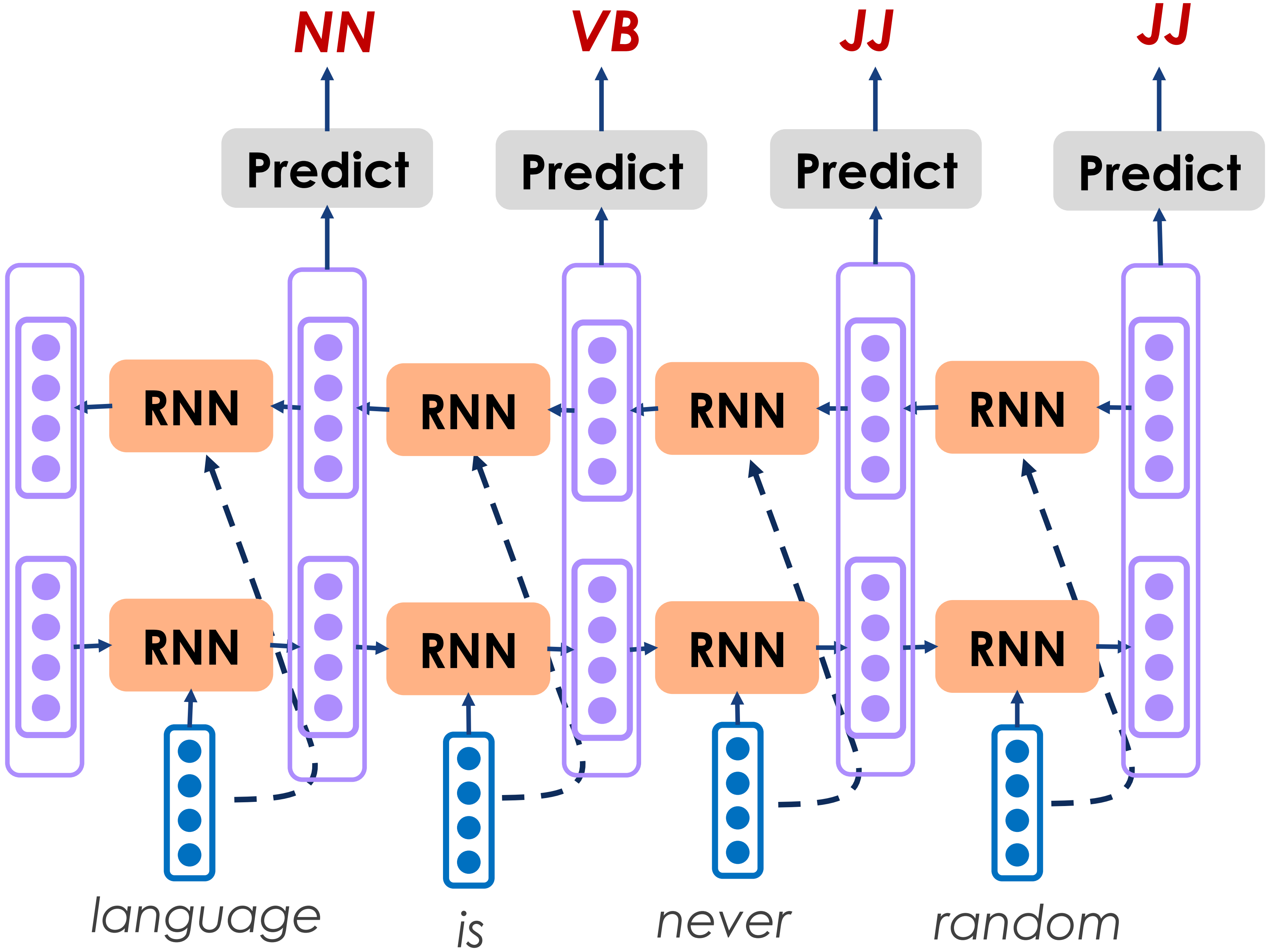
Bi-Directional RNN



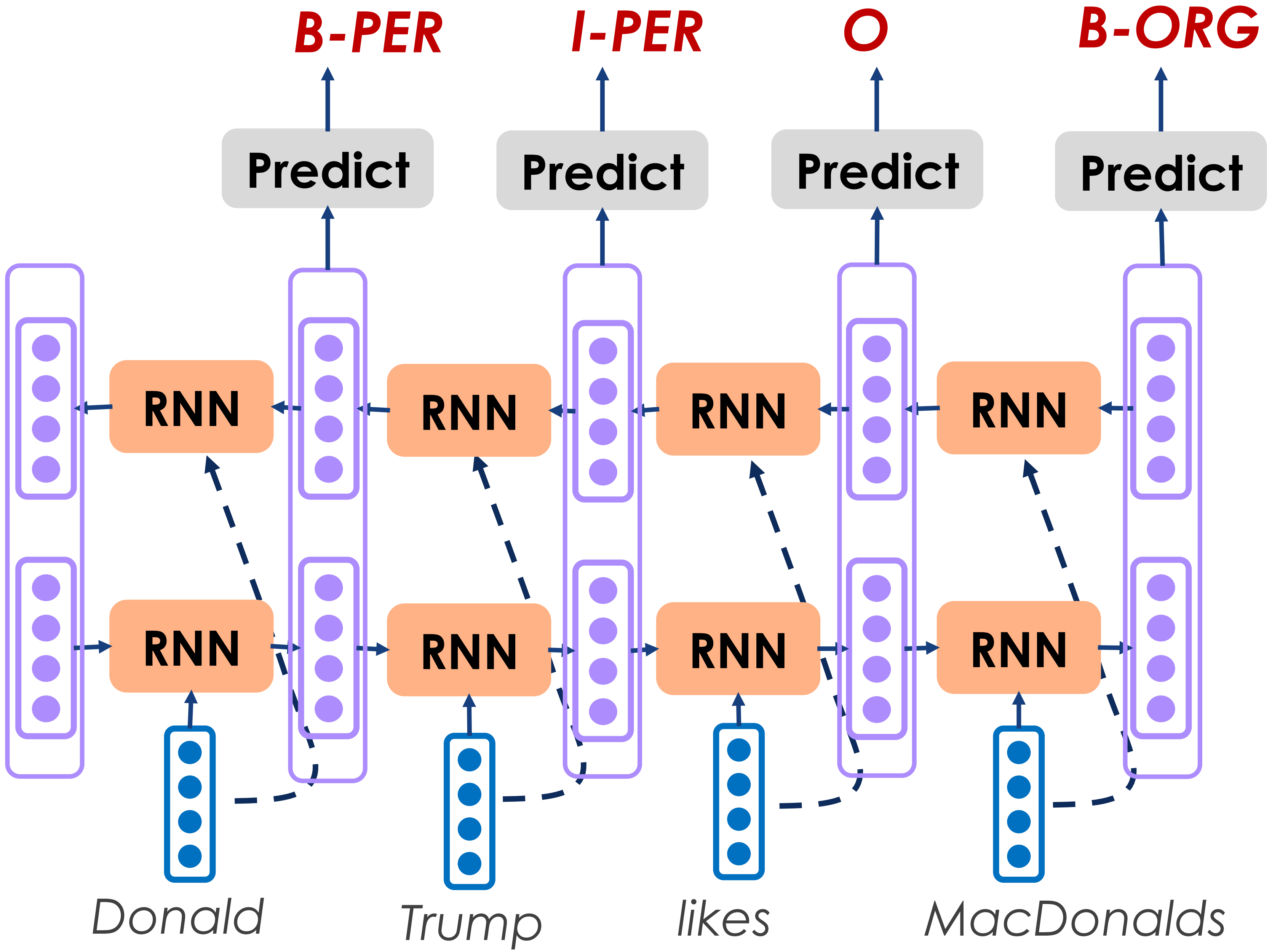
Sequence Predictions

- **Token Tagging**
  - RNN generates a prediction per token, so we can do most NLP annotations where each token comes with their respective tag, e.g. POS, NER, etc.

Sequence Tagging



Sequence Tagging

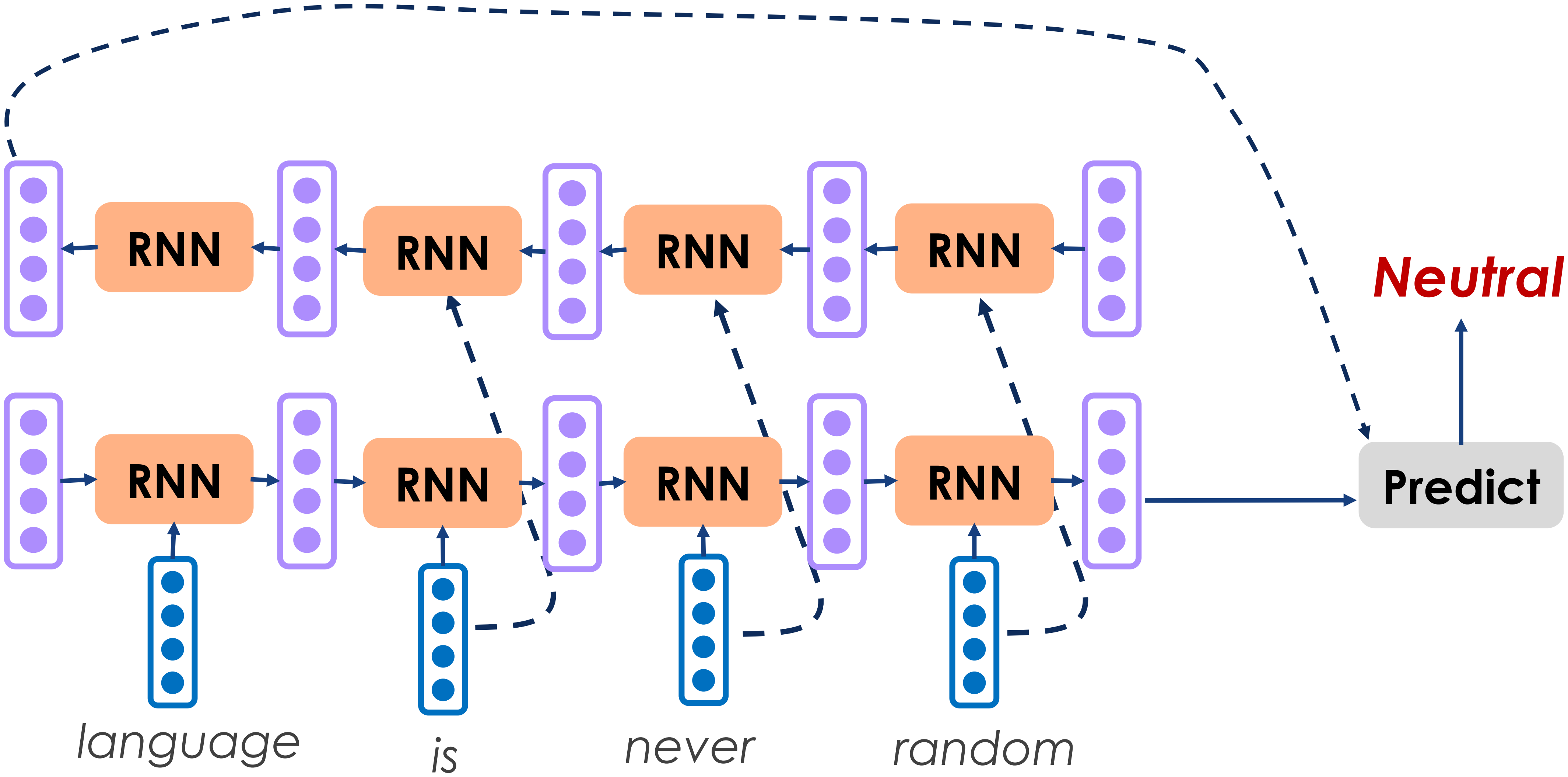


Sequence Predictions

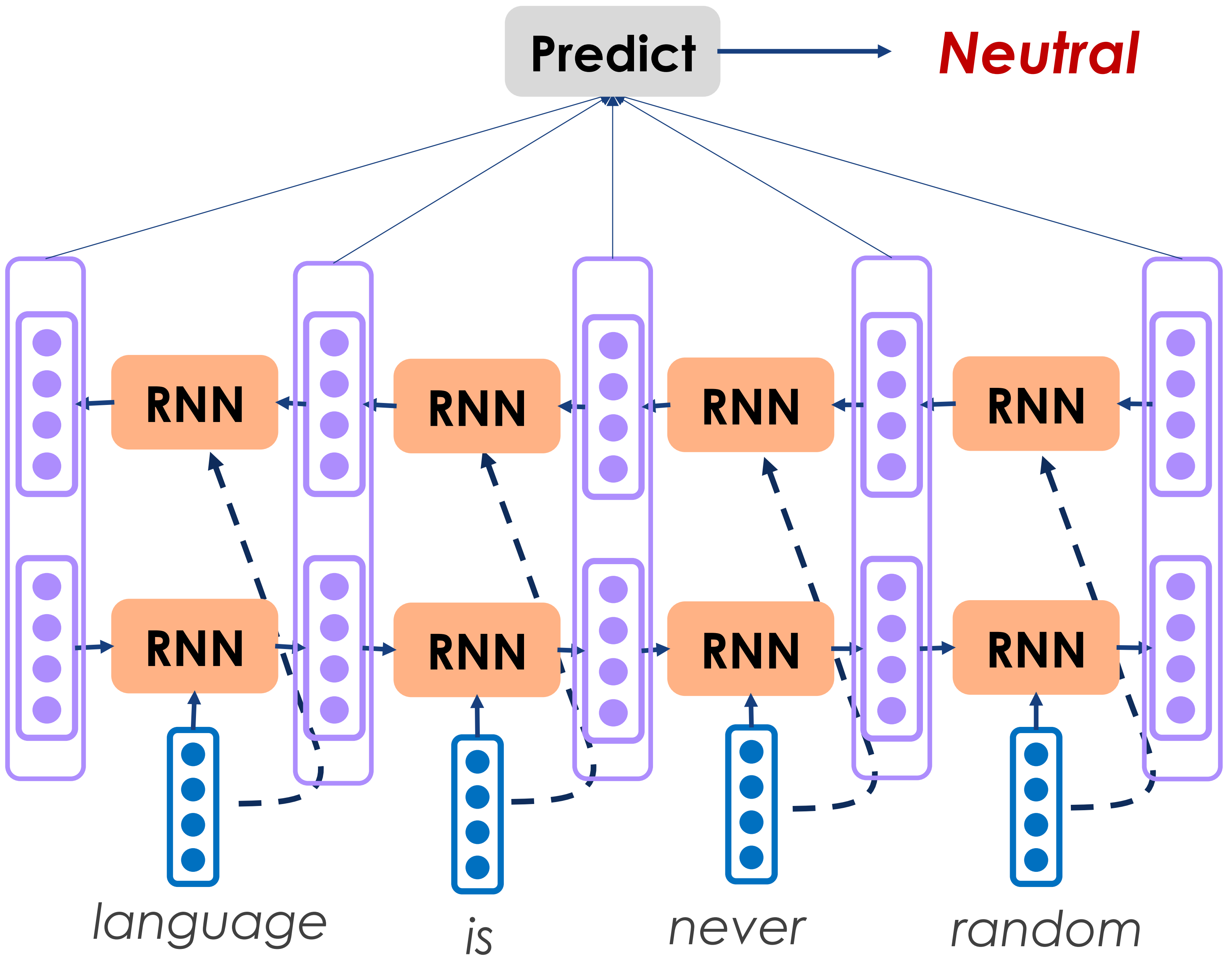
- **Token Tagging**
  - RNN generates a prediction per token, so we can do most NLP annotations where each token comes with their respective tag, e.g. POS, NER, etc.

- **Sentence/Text Classification**
  - Put the end or aggregate of the hidden layer(s) under Softmax to produce probabilistic classifier

Text Classification



Text Classification



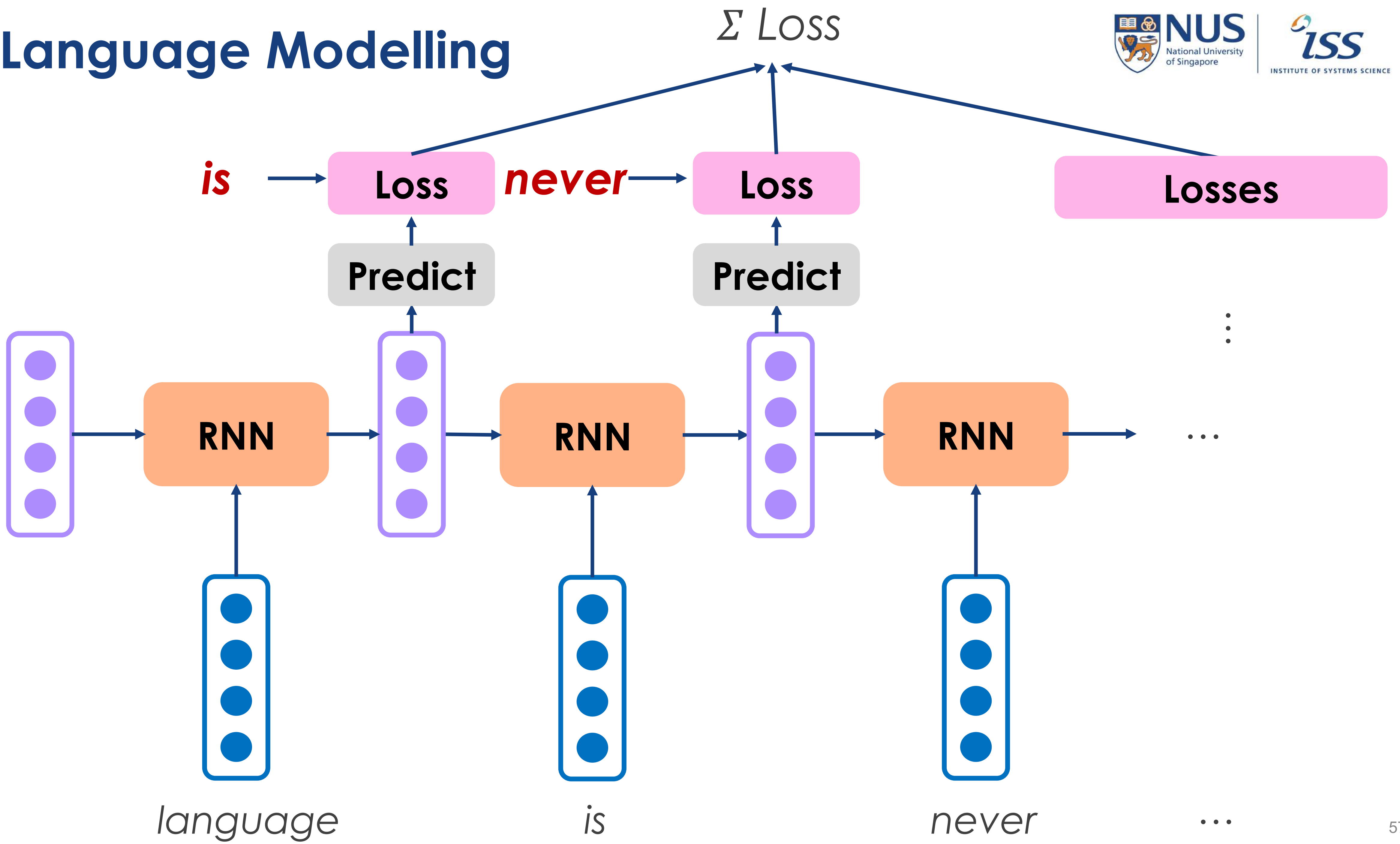
Sequence Predictions

- **Token Tagging**
  - RNN generates a prediction per token, so we can do most NLP annotations where each token comes with their respective tag, e.g. POS, NER, etc.

- **Sentence/Text Classification**
  - Put the end or aggregate of the hidden layer(s) under Softmax to produce probabilistic classifier

- **Sequence Generation**
  - Using the previous hidden layer as input to predict the next most probable word

Language Modelling



Summary

LM + RNN Knowledge Checklist

- **Language Modelling**
  - LM is predicting the next word given history
  - Negative Log Loss as cost function for simple LM
  - Shuffle data, stop early on validation set, always dropout
- **Recurrent Neural Net**
  - Treat each word as a "layer" that needs its own weights
  - Pass hidden layer from previous to next word
  - Predict at each time step or at the end

References (for generation)

Reference Code for Generation

- Character RNN Generation (PyTorch Docs)
  - https://pytorch.org/tutorials/intermediate/char_rnn_generation_tutorial.html
- Character RNN Generation (NLP for PyTorch)
  - https://github.com/joosthub/PyTorchNLPBook/tree/master/chapters/chapter_7/7_3_surname_generation
- Spro's Char RNN
  - https://github.com/spro/practical-pytorch/tree/master/char-rnn-generation

Fin