CS5242 : Neural Networks and Deep Learning

Lecture 8a: Recurrent Neural Networks II

Semester 1 2022/23

Ai Xin

aixin@comp.nus.edu.sg

Department of Computer Science
National University of Singapore (NUS)

Slides Credit: Xavier Bresson

Outline

- Word embedding
- Vanilla RNN implementation
- Batch of documents
- Training loop of VRNNs
- Application to NLP
- LSTM
- Deep RNNs and RNN variants
- Conclusion

Outline

- **Word embedding**
- Vanilla RNN implementation
- Batch of documents
- Training loop of VRNNs
- Application to NLP
- LSTM
- Deep RNNs and RNN variants
- Conclusion

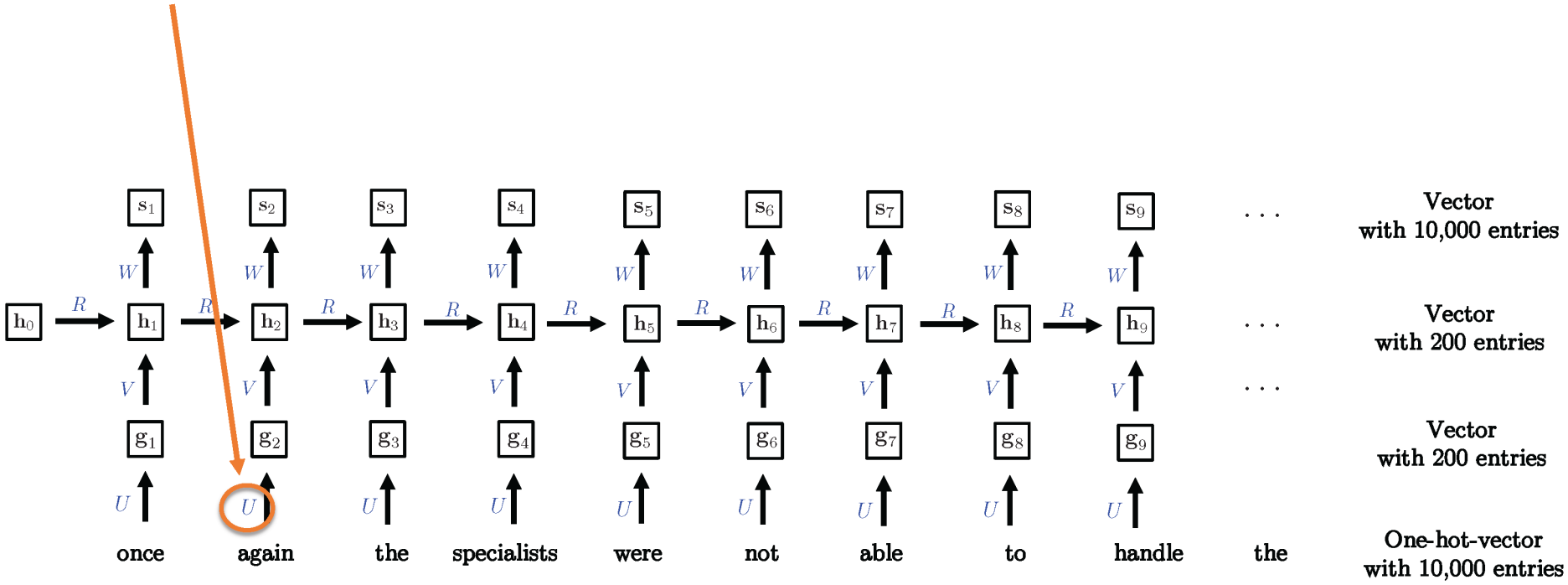
Word embedding

- Categorical variables (dictionary of 10,000 words) are usually
  - represented by one-hot vectors and
  - embedded in a linear space.

- The embedding space:
  - finds the best possible (linear) representation of categorical variables for the task at hand.
  - is learned by backpropagation.

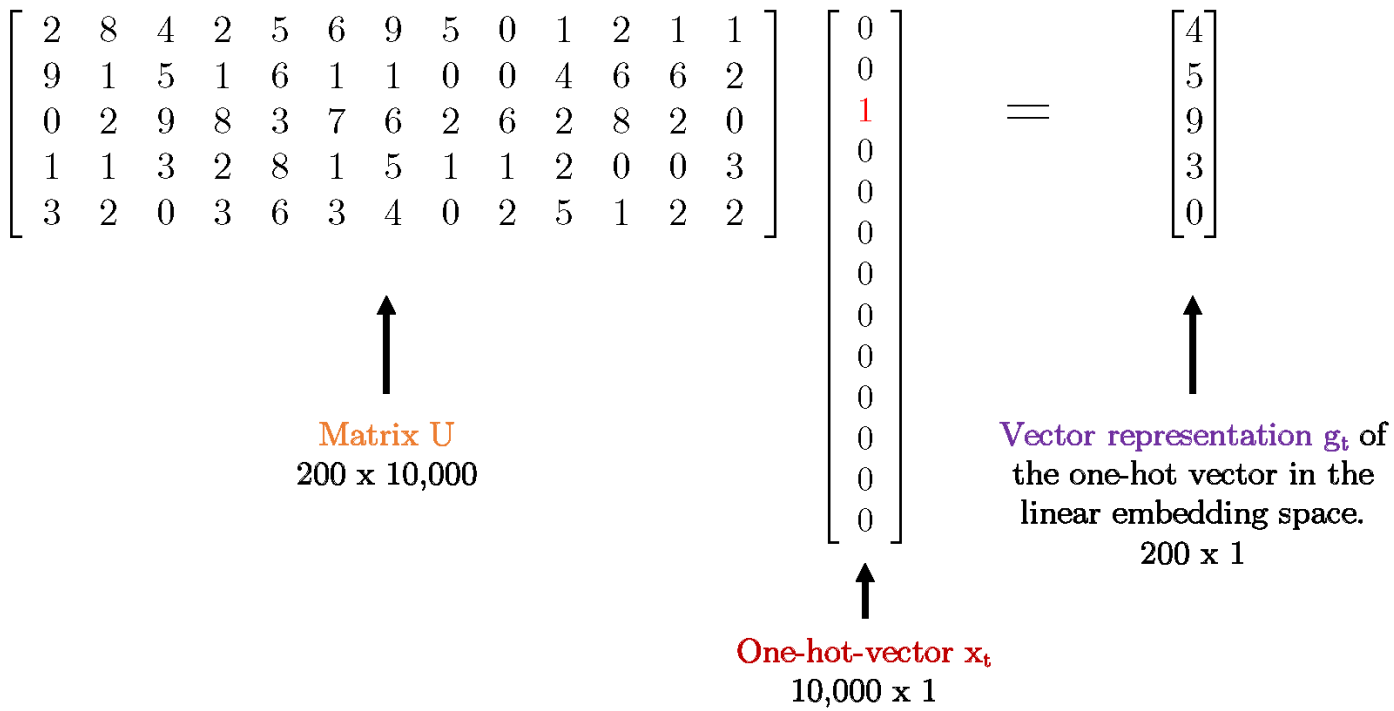
Word embedding

- A matrix U represents the linear embedding space :



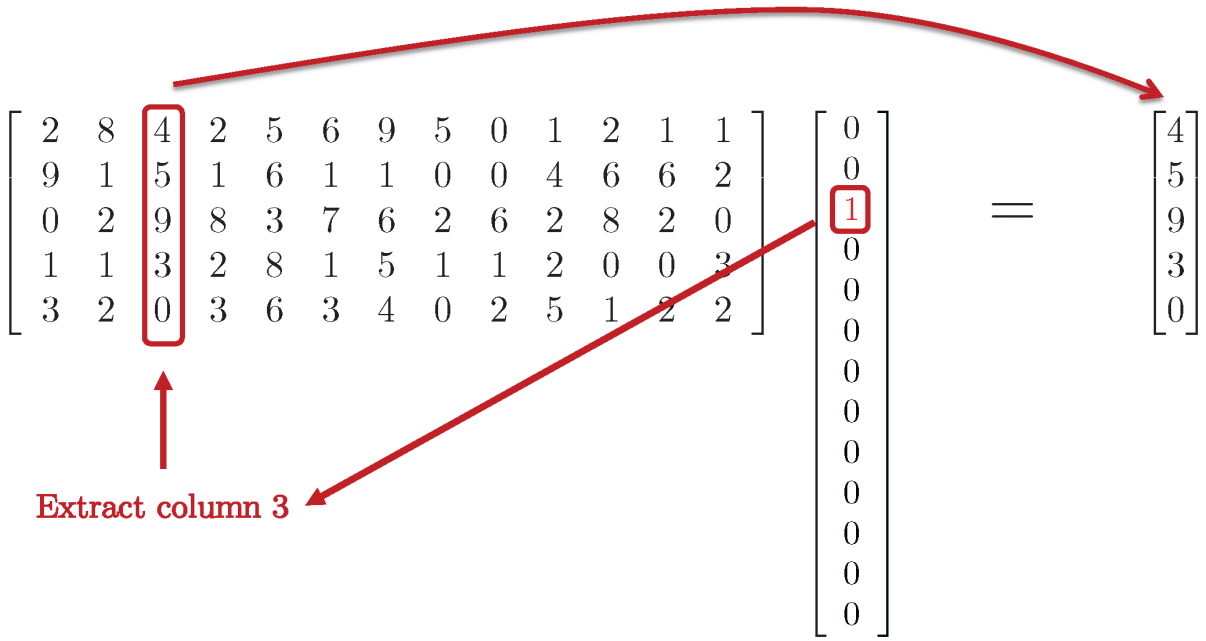
Word embedding

- A matrix U represents the linear embedding space :



Word embedding

- This matrix-one-hot-vector multiplication is a simple slicing operation :
  - This can be efficiently implemented.



Implementation

- PyTorch implementation :
  - World embedding can be implemented either with
    - **nn.Linear()**
    - **nn.Embedding()**

```
mod = nn.Linear(10000, 200, bias=False)
```

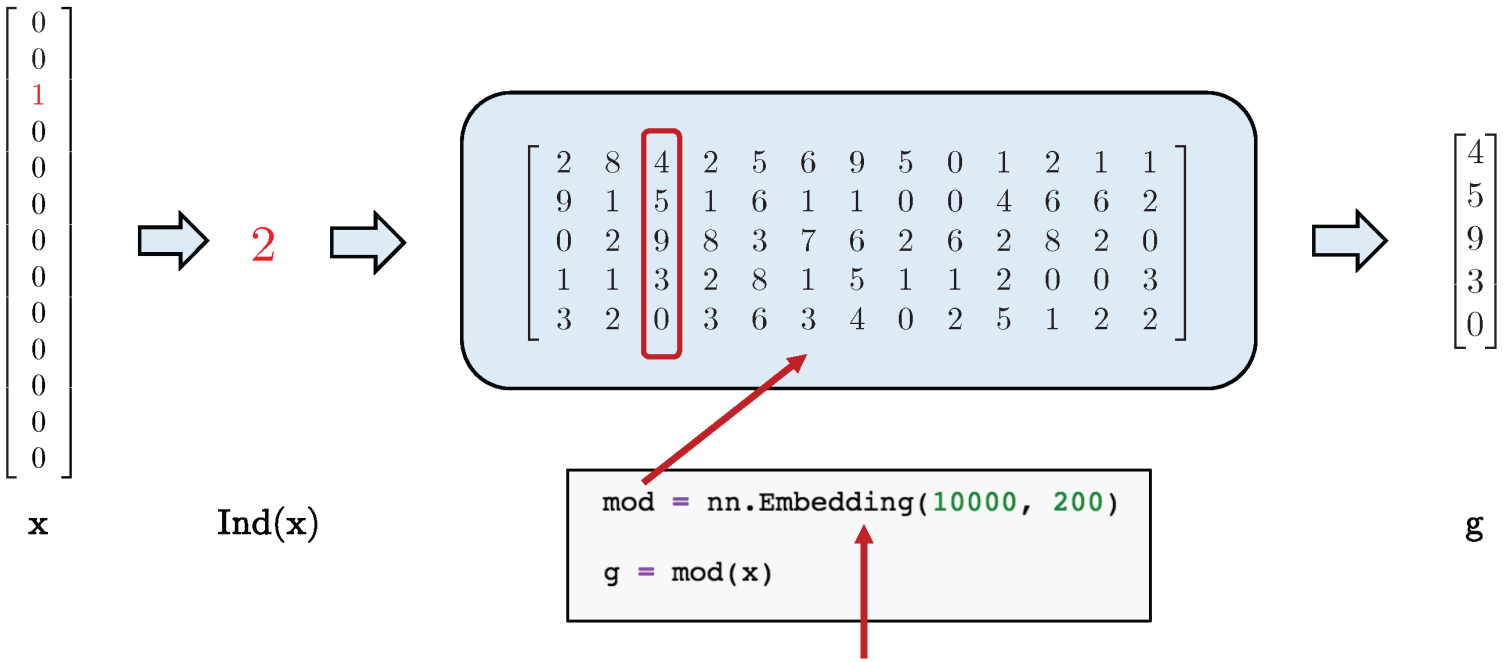
```
mod = nn.Embedding(10000, 200)
```

More efficient !

(requirement: inputs are integers representing the indices of non-zero one-hot-vectors)

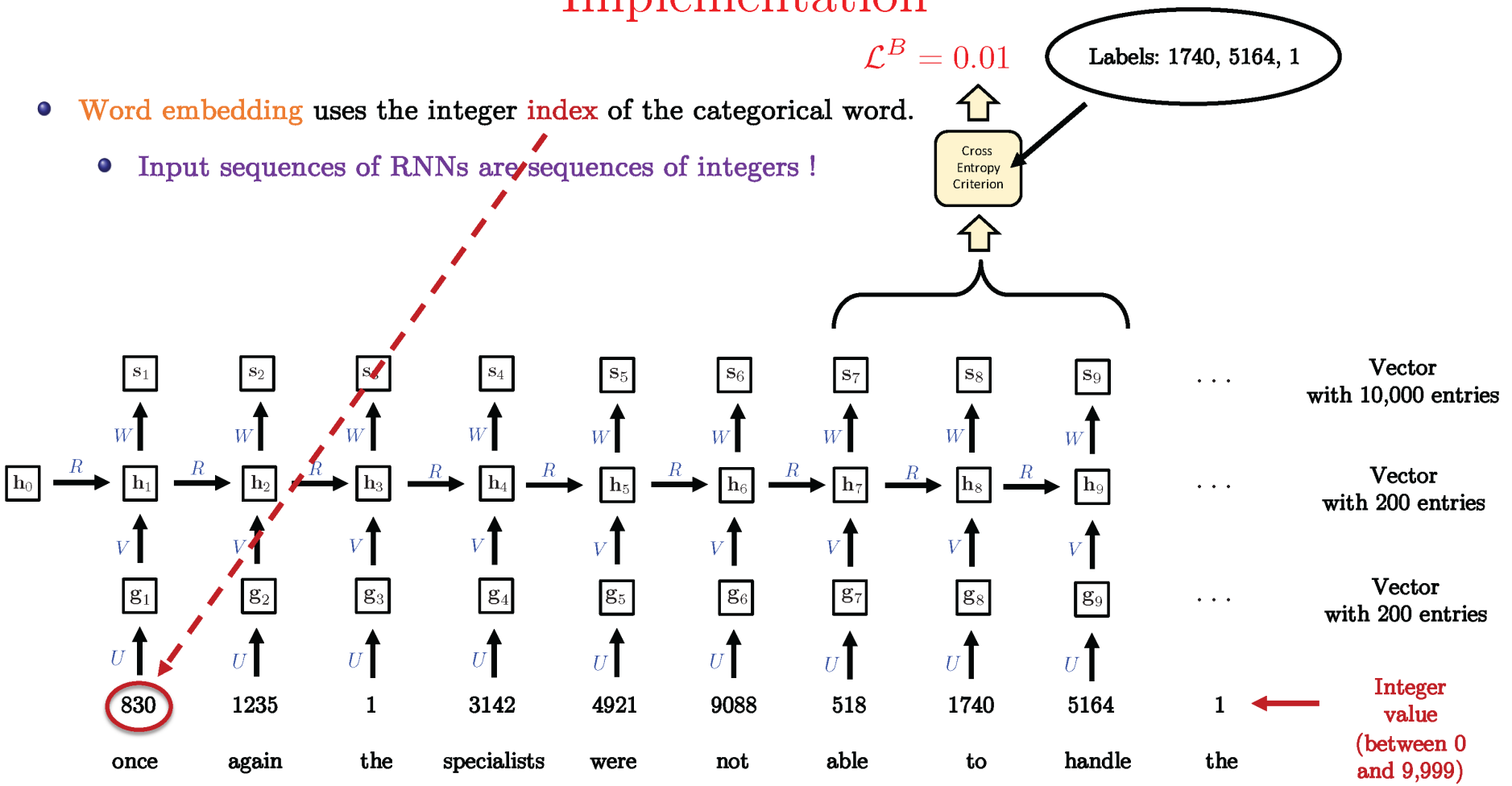
Implementation

- World embedding with nn.Embedding() :



Implementation

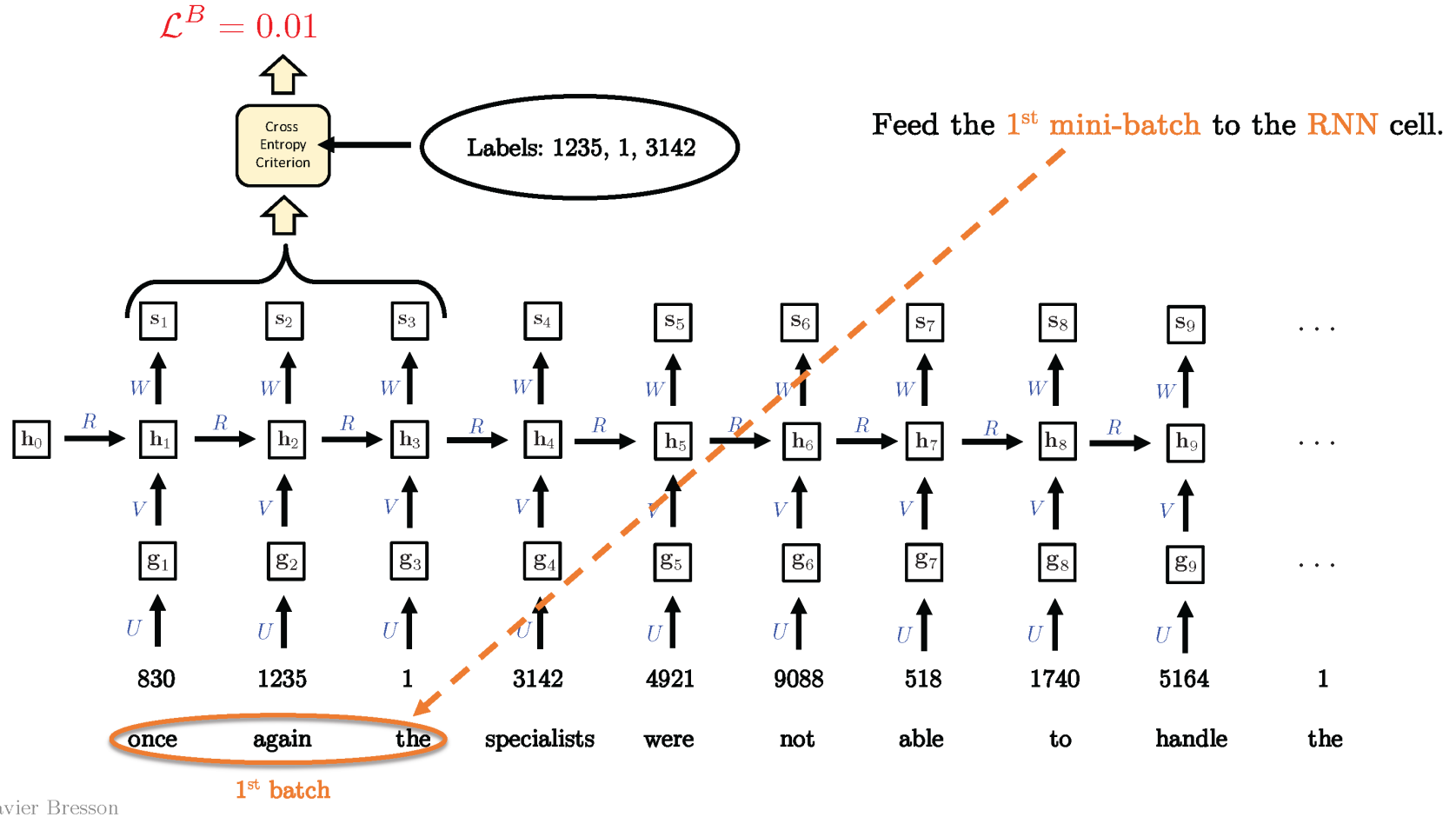
- Word embedding uses the integer index of the categorical word.
  - Input sequences of RNNs are sequences of integers !



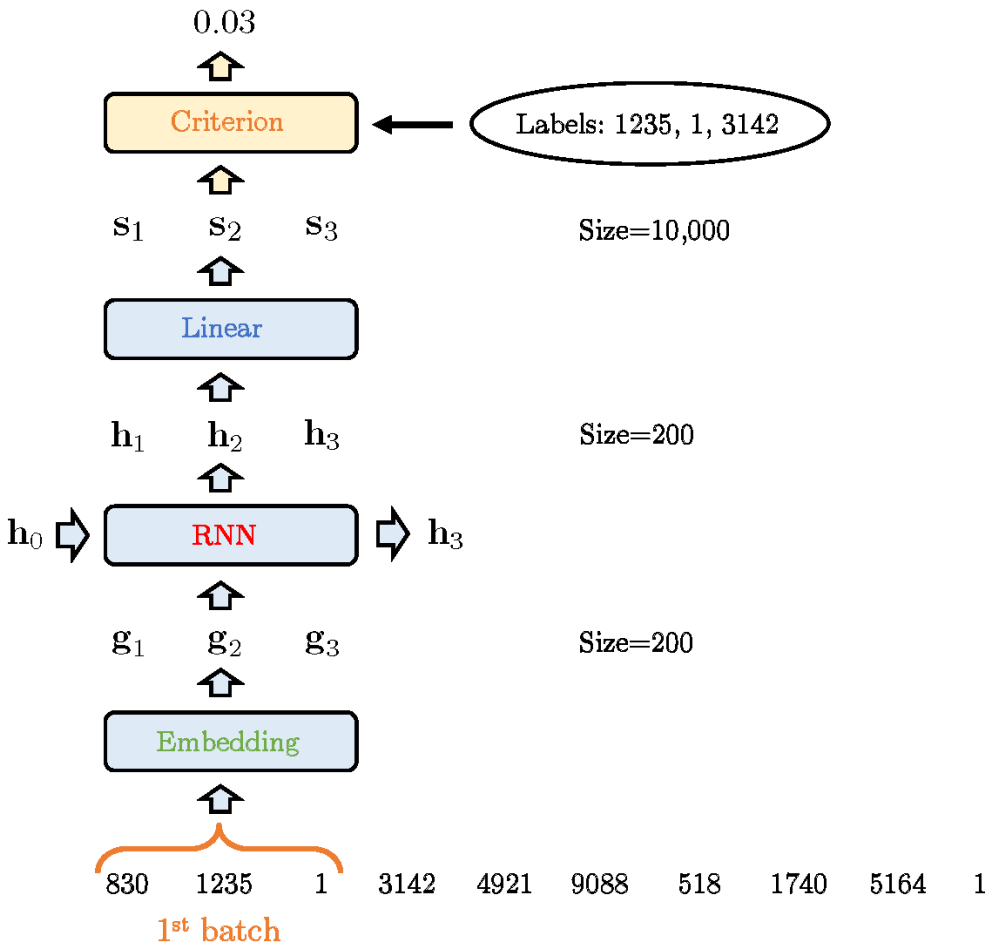
Outline

- Word embedding
- **Vanilla RNN implementation**
- Batch of documents
- Training loop of VRNNs
- Application to NLP
- LSTM
- Deep RNNs and RNN variants
- Conclusion

Vanilla RNN implementation



Vanilla RNN implementation



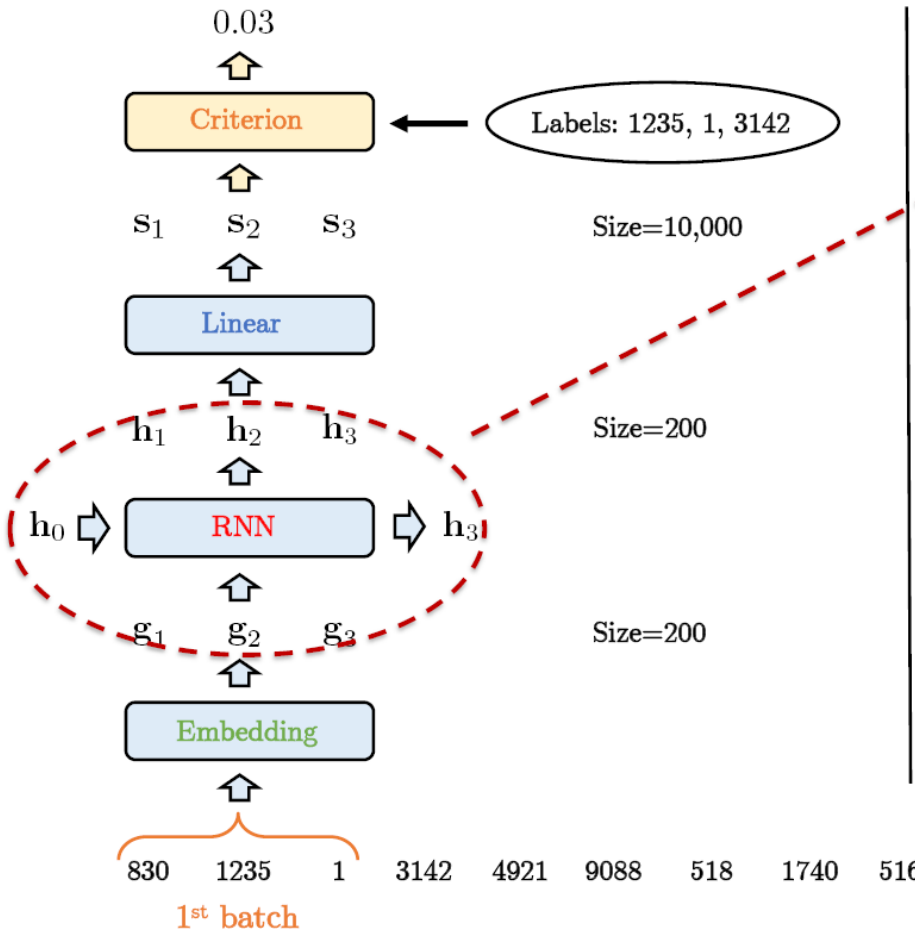
Criterion
Process the batch in parallel

Linear layer
Process the batch in parallel

Recurrent layer
Process the batch in serial

Embedding layer
Process the batch in parallel

Vanilla RNN implementation




```
mod = nn.RNN(200,200)
```

```
h_seq , h_final = mod( g_seq , h_init )
```

Tensor of size 3 x 200

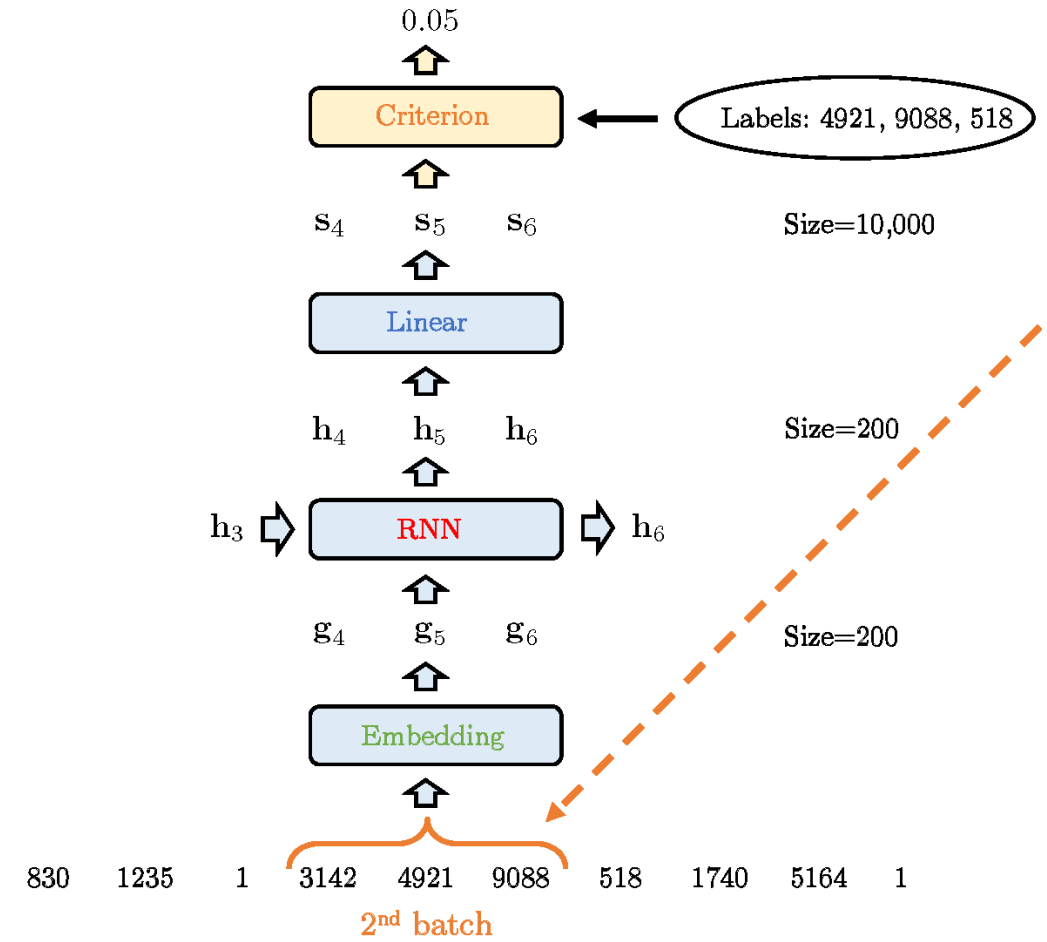
Tensor of size 200

Tensor of size 3 x 200

Tensor of size 200

- seq_length = 3 (batch of words)
- hidden_size = 200

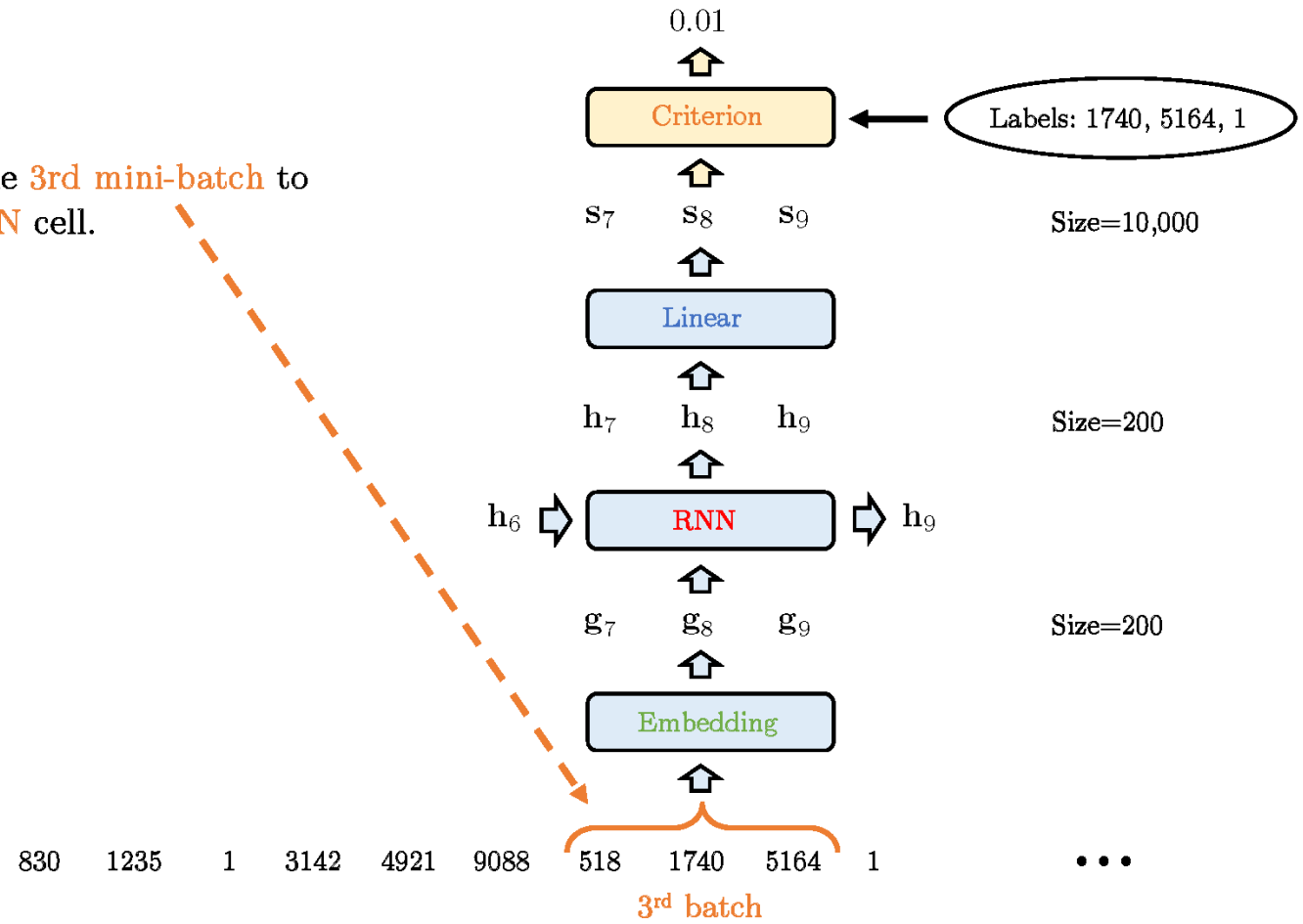
Vanilla RNN implementation



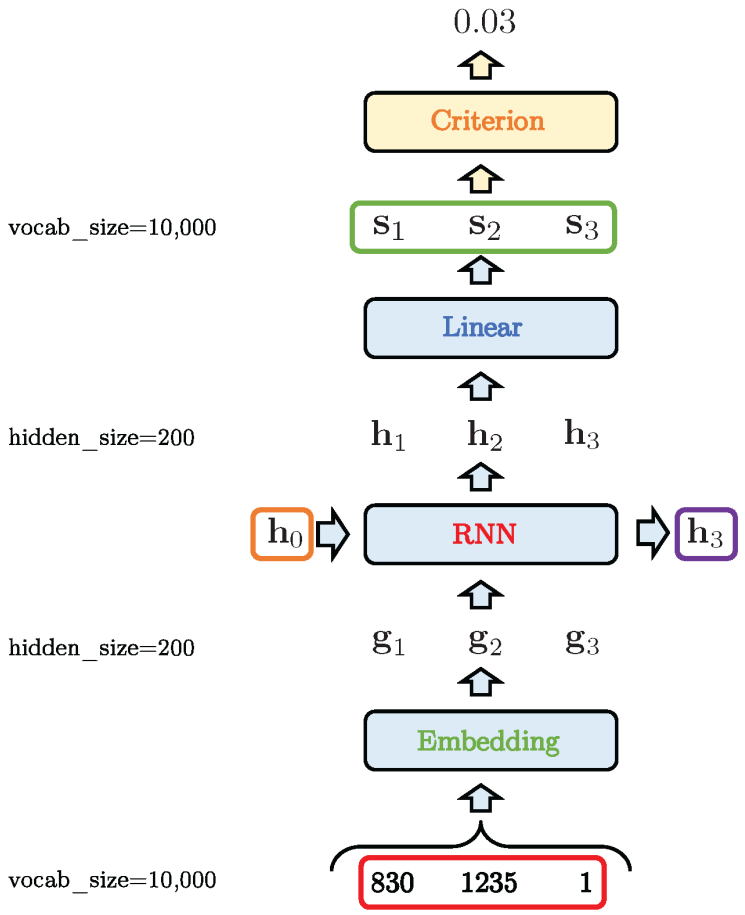
- Feed the 2nd mini-batch to the RNN cell.
- The same RNN cell is shared for all batches :
  - Weight sharing : U, V, W, R
- The RNN cell learns (with the recurrence equation and U, V, W, R) the dynamics of causal signals, here the sequence of words.

Vanilla RNN implementation

- Feed the 3rd mini-batch to the RNN cell.
- Etc.



Vanilla RNN implementation




```
mod = nn.RNN(200,200)
```

```
class three_layer_recurrent_net(nn.Module):

    def __init__(self, hidden_size):
        super(three_layer_recurrent_net, self).__init__()

        self.layer1 = nn.Embedding( vocab_size  , hidden_size  )
        self.layer2 = nn.RNN(       hidden_size , hidden_size  )
        self.layer3 = nn.Linear(    hidden_size , vocab_size   )


    def forward(self, word_seq, h_init ):

        g_seq                =   self.layer1( word_seq )
        h_seq , h_final      =   self.layer2( g_seq , h_init )
        score_seq            =   self.layer3( h_seq )

        return score_seq  h_final
```

Class definition of VRNN with PyTorch

Outline

- Word embedding
- Vanilla RNN implementation
- **Batch of documents**
- Training loop of VRNNs
- Application to NLP
- LSTM
- Deep RNNs and RNN variants
- Conclusion

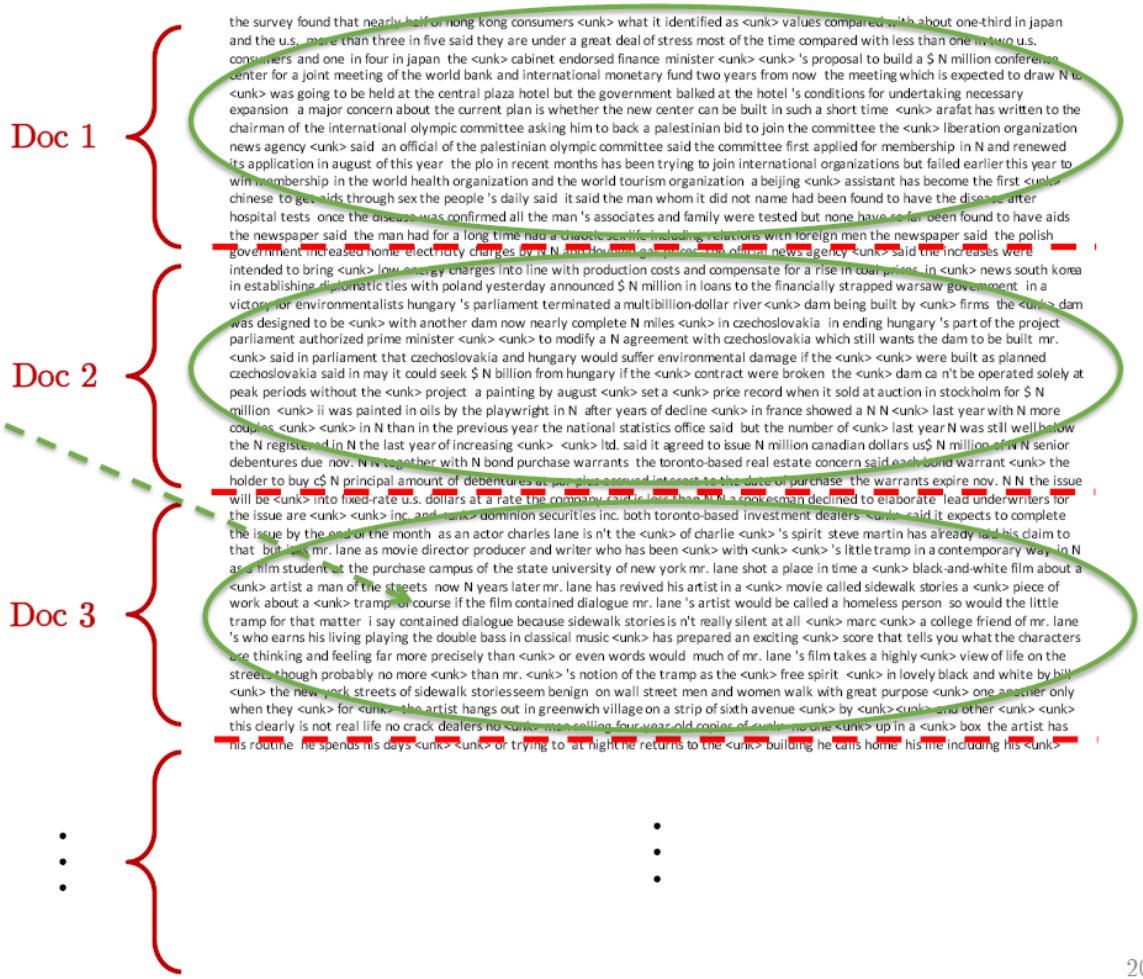
Batch of documents

- Penn Tree Bank :
  - A very large documents of 1 million words
- An epoch of 1 million words would take a lot of time :
  - We can split the full document into multiple sub-documents.

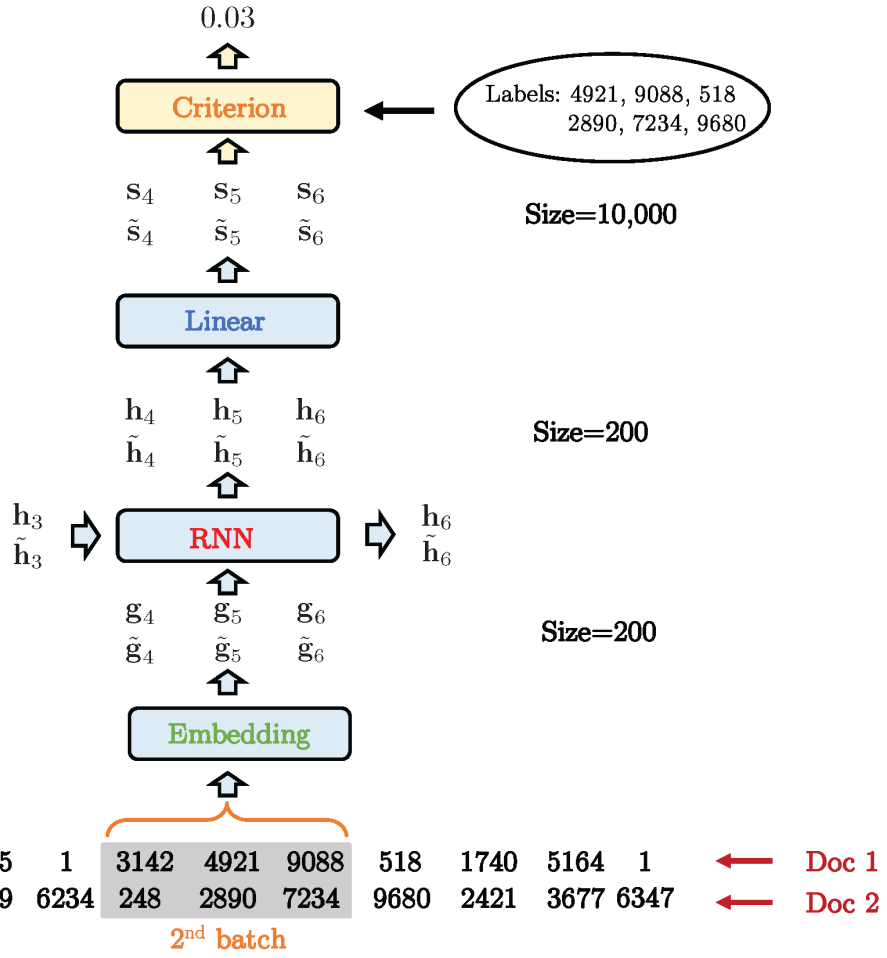
the survey found that nearly half of hong kong consumers <unk> what it identified as <unk> values compared with about one-third in japan and the u.s. more than three in five said they are under a great deal of stress most of the time compared with less than one in two u.s. consumers and one in four in japan the <unk> cabinet endorsed finance minister <unk> <unk> 's proposal to build a $ N million conference center for a joint meeting of the world bank and international monetary fund two years from now the meeting which is expected to draw N to <unk> was going to be held at the central plaza hotel but the government balked at the hotel 's conditions for undertaking necessary expansion a major concern about the current plan is whether the new center can be built in such a short time <unk> arafat has written to the chairman of the international olympic committee asking him to back a palestinian bid to join the committee the <unk> liberation organization news agency <unk> said an official of the palestinian olympic committee said the committee first applied for membership in N and renewed its application in august of this year the plo in recent months has been trying to join international organizations but failed earlier this year to win membership in the world health organization and the world tourism organization a beijing <unk> assistant has become the first <unk> chinese to get aids through sex the people 's daily said it said the man whom it did not name had been found to have the disease after hospital tests once the disease was confirmed all the man 's associates and family were tested but none have so far been found to have aids the newspaper said the man had for a long time had a chaotic sex life including relations with foreign men the newspaper said the polish government increased home electricity charges by N N and doubled gas prices the official news agency <unk> said the increases were intended to bring <unk> low energy charges into line with production costs and compensate for a rise in coal prices in <unk> news south korea in establishing diplomatic ties with poland yesterday announced $ N million in loans to the financially strapped warsaw government in a victory for environmentalists hungary 's parliament terminated a multibillion-dollar river <unk> dam being built by <unk> firms the <unk> dam was designed to be <unk> with another dam now nearly complete N miles <unk> in czechoslovakia in ending hungary 's part of the project parliament authorized prime minister <unk> <unk> to modify a N agreement with czechoslovakia which still wants the dam to be built mr. <unk> said in parliament that czechoslovakia and hungary would suffer environmental damage if the <unk> <unk> were built as planned czechoslovakia said in may it could seek $ N billion from hungary if the <unk> contract were broken the <unk> dam ca n't be operated solely at peak periods without the <unk> project a painting by august <unk> set a <unk> price record when it sold at auction in stockholm for $ N million <unk> ii was painted in oils by the playwright in N after years of decline <unk> in france showed a N N <unk> last year with N more couples <unk> <unk> in N than in the previous year the national statistics office said but the number of <unk> last year N was still well below the N registered in N the last year of increasing <unk> <unk> ltd. said it agreed to issue N million canadian dollars us$ N million of N N senior debentures due nov. N N together with N bond purchase warrants the toronto-based real estate concern said each bond warrant <unk> the holder to buy c$ N principal amount of debentures at par plus accrued interest to the date of purchase the warrants expire nov. N N the issue will be <unk> into fixed-rate u.s. dollars at a rate the company said is less than N N a spokesman declined to elaborate lead underwriters for the issue are <unk> <unk> inc. and <unk> dominion securities inc. both toronto-based investment dealers <unk> said it expects to complete the issue by the end of the month as an actor charles lane is n't the <unk> of charlie <unk> 's spirit steve martin has already laid his claim to that but it is mr. lane as movie director producer and writer who has been <unk> with <unk> <unk> 's little tramp in a contemporary way in N as a film student at the purchase campus of the state university of new york mr. lane shot a place in time a <unk> black-and-white film about a <unk> artist a man of the streets now N years later mr. lane has revived his artist in a <unk> movie called sidewalk stories a <unk> piece of work about a <unk> tramp of course if the film contained dialogue mr. lane 's artist would be called a homeless person so would the little tramp for that matter i say contained dialogue because sidewalk stories is n't really silent at all <unk> marc <unk> a college friend of mr. lane 's who earns his living playing the double bass in classical music <unk> has prepared an exciting <unk> score that tells you what the characters are thinking and feeling far more precisely than <unk> or even words would much of mr. lane 's film takes a highly <unk> view of life on the streets though probably no more <unk> than mr. <unk> 's notion of the tramp as the <unk> free spirit <unk> in lovely black and white by bill <unk> the new york streets of sidewalk stories seem benign on wall street men and women walk with great purpose <unk> one another only when they <unk> for <unk> the artist hangs out in greenwich village on a strip of sixth avenue <unk> by <unk> <unk> and other <unk> <unk> this dearly is not real life no crack dealers no <unk> men selling four-year-old copies of <unk> no one <unk> up in a <unk> box the artist has his routine he spends his days <unk> <unk> or trying to at night he returns to the <unk> building he calls home his life including his <unk>

Batch of documents

- Batch of documents :
  - We break the original document into 20 sub-documents of 50 thousands words.
  - Efficiency: We can process these 20 sub-documents in parallel !
  - All sub-documents are (quasi-)independent.



Batch of documents




```
mod = nn.RNN(200,200)
```

```
h_seq , h_final = mod( g_seq , h_init )
```

Tensor of size
3 x 2 x 200

Tensor of size
2 x 200

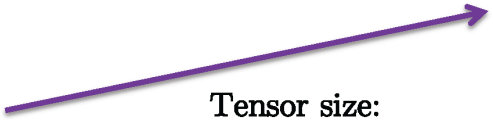
Tensor of size
3 x 2 x 200

Tensor of size
2 x 200

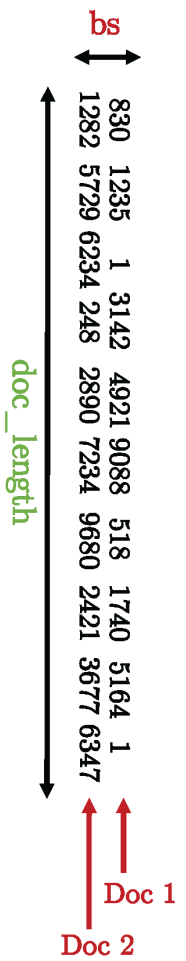
- bs = 2 (batch of sub-documents)
- seq_length = 3 (batch of words)
- hidden_size = 200

Batch of documents

- The batch of documents is shaped as :
  - doc_length x bs, where
    - doc_length : the length of the sub-documents.
    - bs : the number of sub-documents.
- Here is the view of the data representation :



Tensor size:
doc_length x bs



- This data representation allows to directly slice mini-batch of words (PyTorch makes easy to slice w.r.t. the first dimension).

```
minibatch_data = test_data[ count : count + seq_length ]
```

Batch of documents

- The PTB dataset is prepared as follows :
  - Train set is a tensor of size 46,479 x 20.
  - Test set is a tensor of size 4,121 x 20.
  - Sanity check : (46,479 + 4,121) x 20 = 1 million words

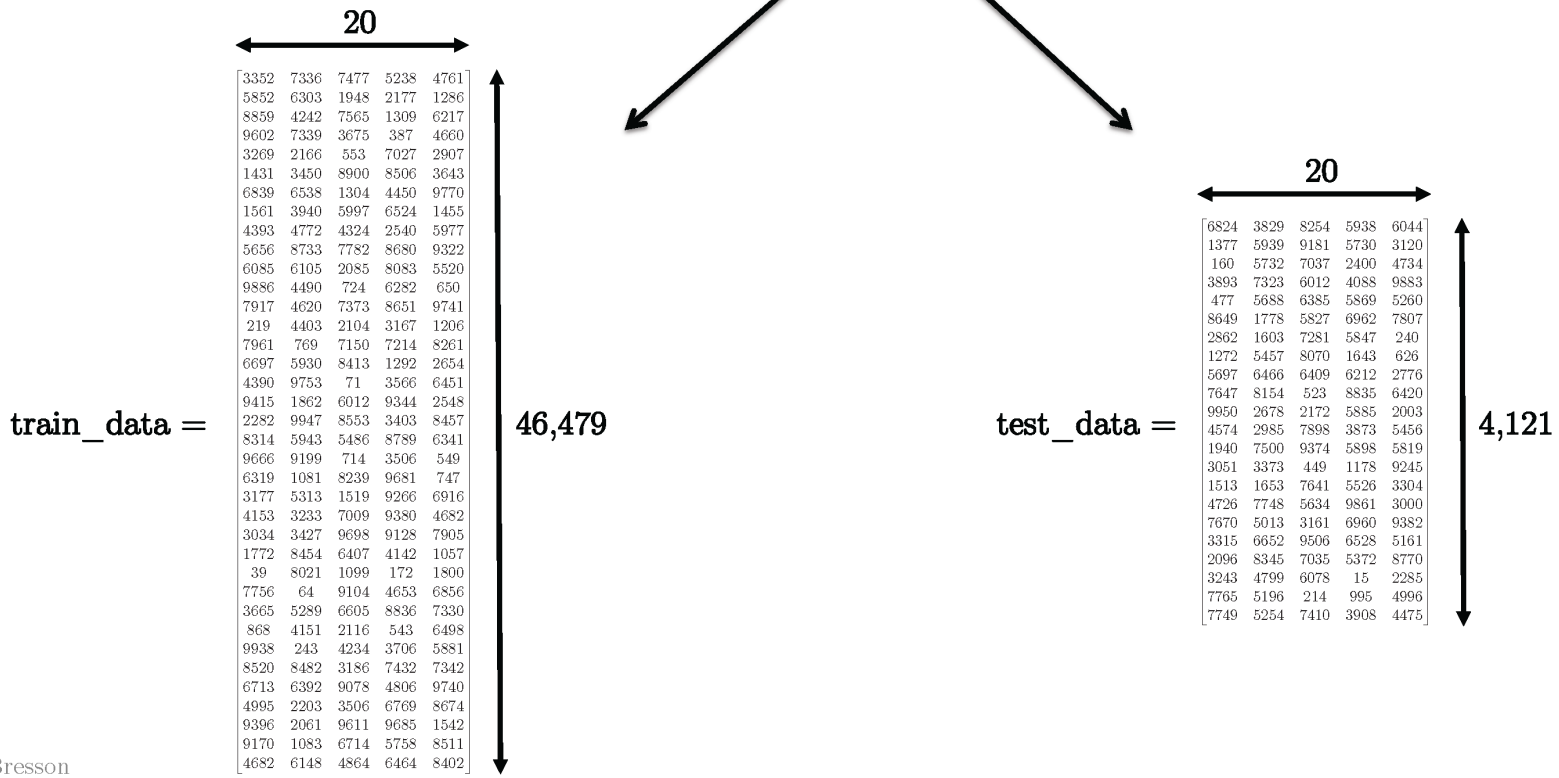
- Load the PTB datasets with PyTorch :

```
train_data  =  torch.load(data_path+'ptb/train_data.pt')
test_data   =  torch.load(data_path+'ptb/test_data.pt')

print(  train_data.size()  )
print(  test_data.size()   )
```

```
torch.Size([46479, 20])
torch.Size([4121, 20])
```

Batch of documents

```
train_data  =  torch.load(data_path+'ptb/train_data.pt')
test_data   =  torch.load(data_path+'ptb/test_data.pt')
```



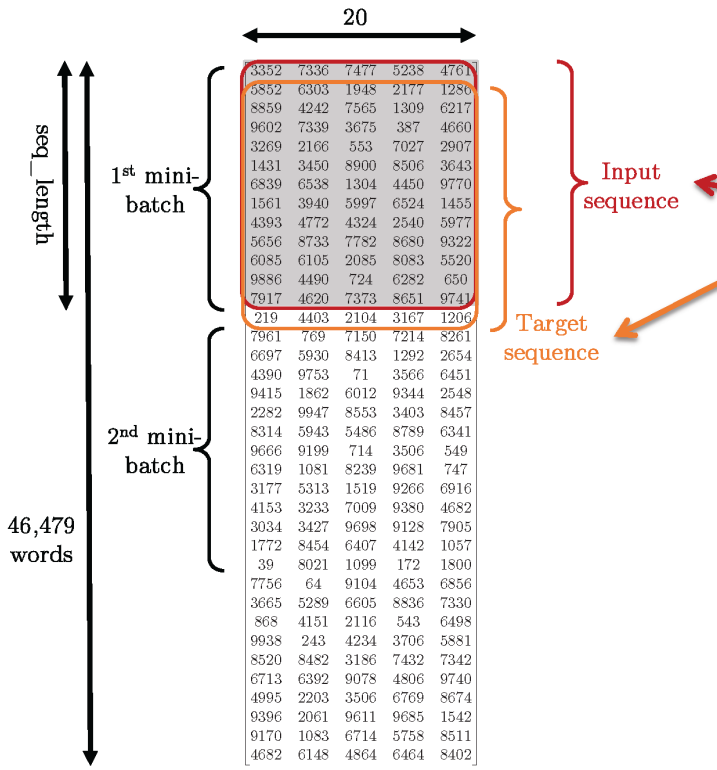
Outline

- Word embedding
- Vanilla RNN implementation
- Batch of documents
- **Training loop of VRNNs**
- Application to NLP
- LSTM
- Deep RNNs and RNN variants
- Conclusion

Training loop of VRNNs

The input sequence is a mini-batch/sequence of words.

The target sequence is simply the same sequence shifted by one word.




```
for count in range( 0 , 46478-seq_length ,  seq_length):

    optimizer.zero_grad()

    minibatch_data =  train_data[ count    : count+seq_length    ]
    minibatch_label = train_data[ count+1 : count+seq_length+1 ]

    minibatch_data=minibatch_data.to(device)
    minibatch_label=minibatch_label.to(device)

    h=h.detach()
    h=h.requires_grad_()

    scores , h  =  net( minibatch_data, h )

    scores          =            scores.view(  bs*seq_length , vocab_size)
    minibatch_label =   minibatch_label.view(  bs*seq_length )

    loss = criterion(  scores ,  minibatch_label )

    loss.backward()

    utils.normalize_gradient(net)
    optimizer.step()
```

Training loop of VRNNs

Send data to the CPU or GPU device.

```
for count in range( 0 , 46478-seq_length ,  seq_length):

    optimizer.zero_grad()

    minibatch_data =  train_data[ count    : count+seq_length    ]
    minibatch_label = train_data[ count+1 : count+seq_length+1 ]

    minibatch_data=minibatch_data.to(device)
    minibatch_label=minibatch_label.to(device)

    h=h.detach()
    h=h.requires_grad_()

    scores , h  =  net( minibatch_data, h )

    scores          =            scores.view(  bs*seq_length , vocab_size)
    minibatch_label =   minibatch_label.view(  bs*seq_length )

    loss = criterion(  scores ,  minibatch_label )

    loss.backward()

    utils.normalize_gradient(net)
    optimizer.step()
```

Training loop of VRNNs

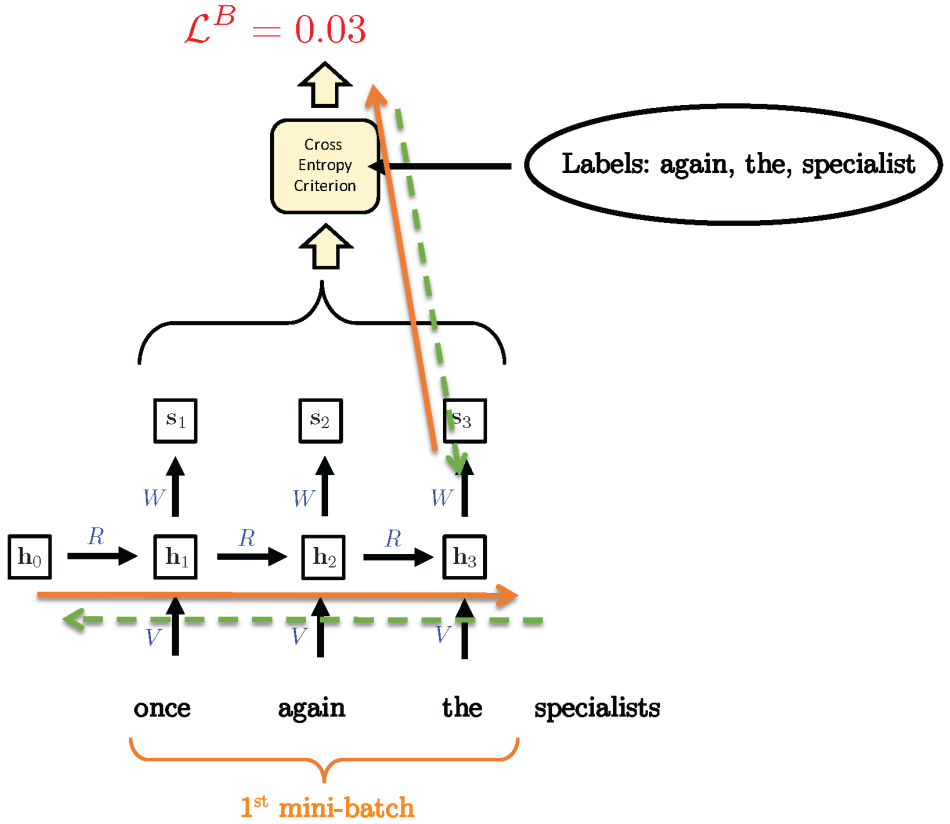
Stop backpropagation to go beyond the current mini-batch.

Start recording the new computational graph for backpropagation.

```
for count in range( 0 , 46478-seq_length ,  seq_length):

    optimizer.zero_grad()

    minibatch_data =  train_data[ count    : count+seq_length    ]
    minibatch_label = train_data[ count+1 : count+seq_length+1 ]

    minibatch_data=minibatch_data.to(device)
    minibatch_label=minibatch_label.to(device)

    h=h.detach()
    h=h.requires_grad_()

    scores , h  =  net( minibatch_data, h )

    scores          =            scores.view(  bs*seq_length , vocab_size)
    minibatch_label =   minibatch_label.view(  bs*seq_length )

    loss = criterion(  scores ,  minibatch_label )

    loss.backward()

    utils.normalize_gradient(net)
    optimizer.step()
```

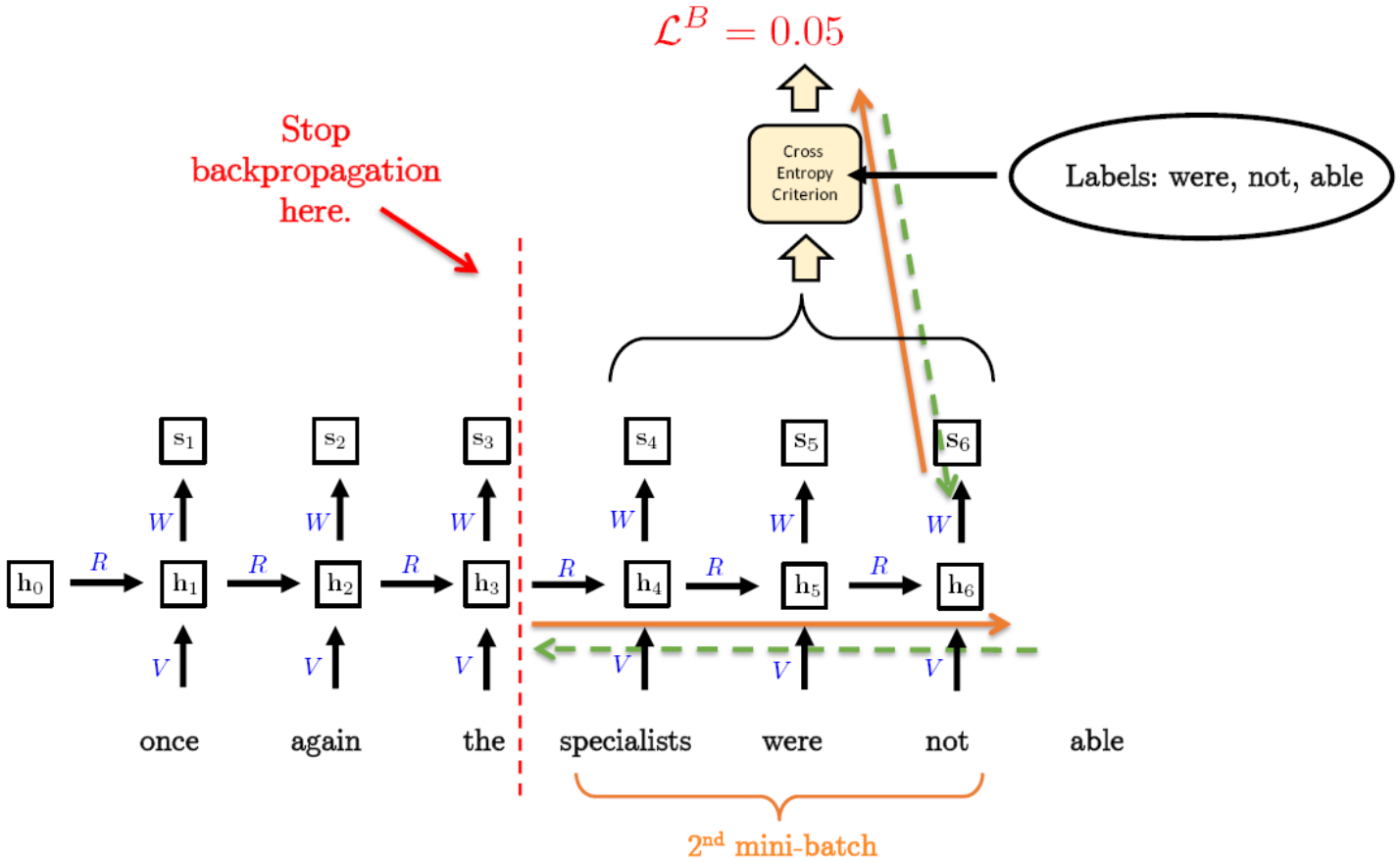
Training loop of VRNNs

- With **detach()** :



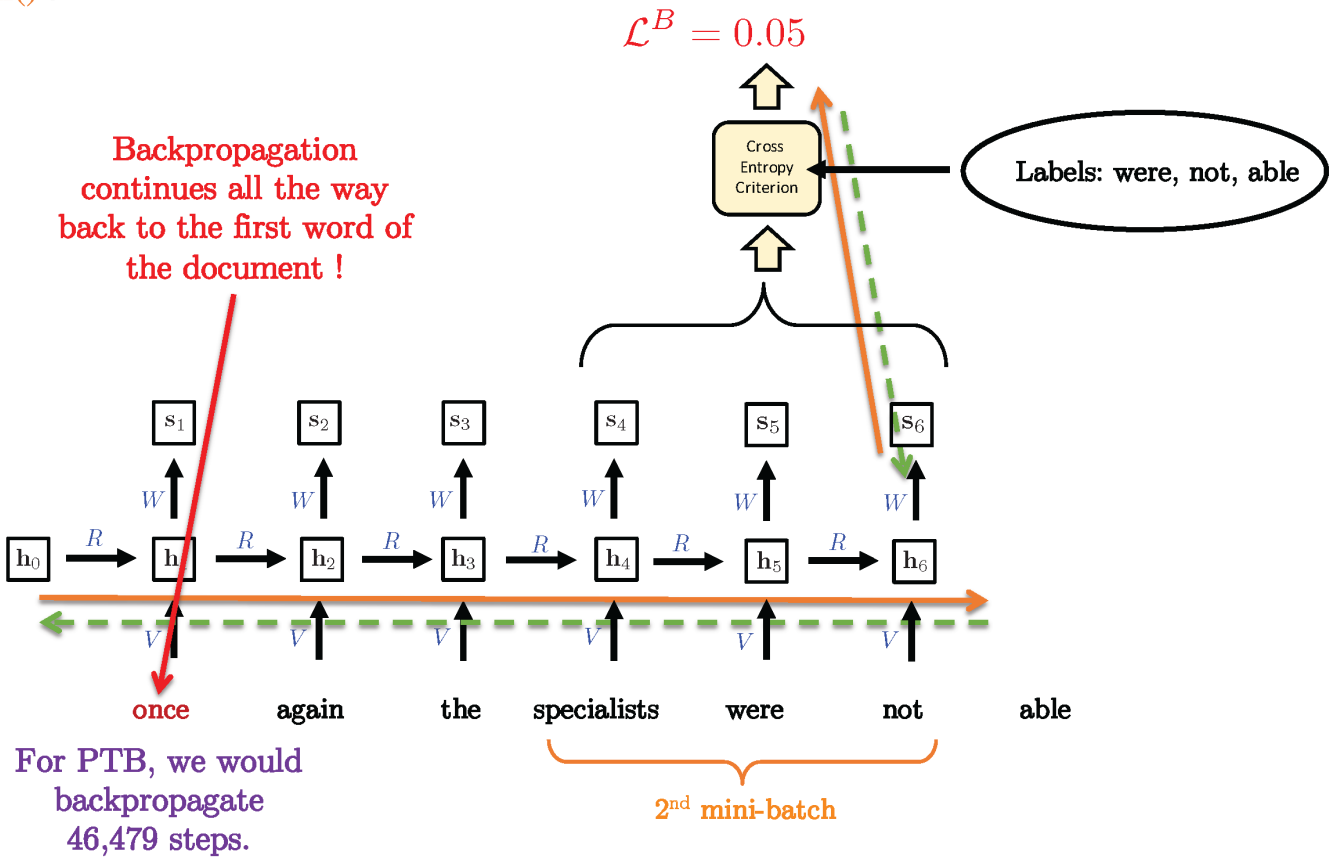
Training loop of VRNNs

- With **detach()** :

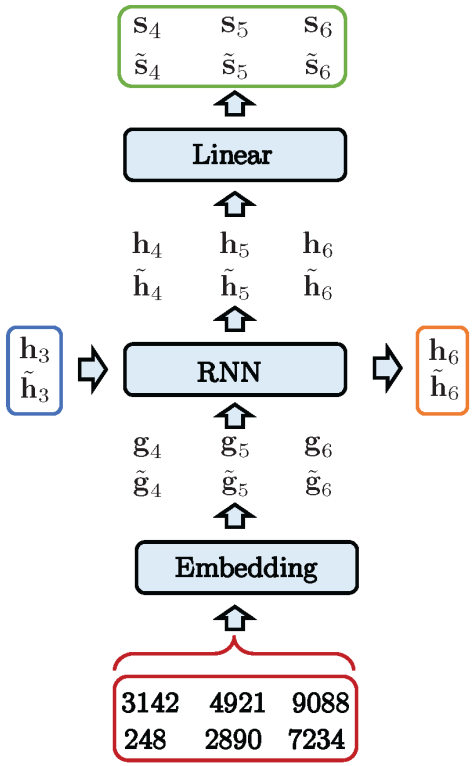


Training loop of VRNNs

- Without detach() :



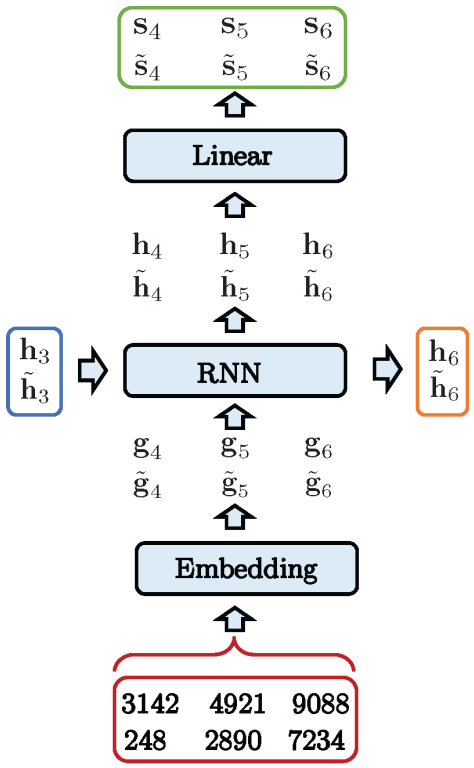
Training loop of VRNNs




```
for count in range( 0 , 46478-seq_length ,  seq_length):

    optimizer.zero_grad()

    minibatch_data =  train_data[ count    : count+seq_length    ]
    minibatch_label = train_data[ count+1 : count+seq_length+1 ]

    minibatch_data=minibatch_data.to(device)
    minibatch_label=minibatch_label.to(device)

    h=h.detach()
    h=h.requires_grad_()

    scores , h = net( minibatch_data, h )

    scores           =           scores.view(  bs*seq_length , vocab_size)
    minibatch_label =  minibatch_label.view(  bs*seq_length )

    loss = criterion(  scores ,  minibatch_label )

    loss.backward()

    utils.normalize_gradient(net)
    optimizer.step()
```

net is an instantiation of the VRNN class.

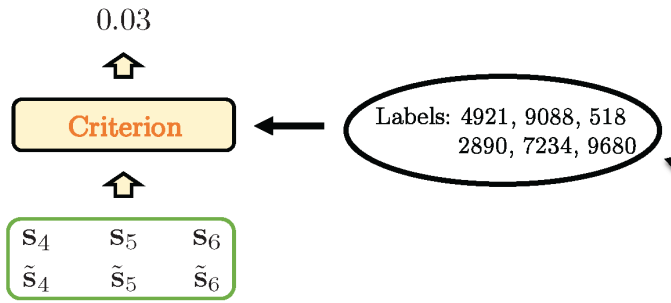
Training loop of VRNNs



Class definition of a VRNN cell.

```
class three_layer_recurrent_net(nn.Module):

    def __init__(self, hidden_size):
        super(three_layer_recurrent_net, self).__init__()

        self.layer1 = nn.Embedding( vocab_size  , hidden_size  )
        self.layer2 = nn.RNN(       hidden_size , hidden_size  )
        self.layer3 = nn.Linear(    hidden_size , vocab_size   )


    def forward(self, word_seq, h_init ):

        g_seq                =   self.layer1( word_seq )
        h_seq , h_final      =   self.layer2( g_seq , h_init )
        score_seq            =   self.layer3( h_seq )

        return score_seq, h_final
```

Training loop of VRNNs



Labels: 4921, 9088, 518
2890, 7234, 9680

```
for count in range( 0 , 46478-seq_length ,  seq_length):

    optimizer.zero_grad()

    minibatch_data =  train_data[ count    : count+seq_length    ]
    minibatch_label = train_data[ count+1 : count+seq_length+1 ]

    minibatch_data=minibatch_data.to(device)
    minibatch_label=minibatch_label.to(device)

    h=h.detach()
    h=h.requires_grad_()

    scores , h  =  net( minibatch_data, h )

    scores          =           scores.view(  bs*seq_length , vocab_size)
    minibatch_label =   minibatch_label.view(  bs*seq_length )

    loss = criterion(  scores ,  minibatch_label )

    loss.backward()

    utils.normalize_gradient(net)
    optimizer.step()
```

Tensor size of scores for PyTorch cross_entropy must be N x voc_size.

Tensor size for labels must be N. Values are integers between 0 and voc_size-1.

Training loop of VRNNs

Do backpropagation.

```
for count in range( 0 , 46478-seq_length ,  seq_length):

    optimizer.zero_grad()

    minibatch_data =  train_data[ count    : count+seq_length    ]
    minibatch_label = train_data[ count+1 : count+seq_length+1 ]

    minibatch_data=minibatch_data.to(device)
    minibatch_label=minibatch_label.to(device)

    h=h.detach()
    h=h.requires_grad_()

    scores , h  =  net( minibatch_data, h )

    scores          =            scores.view(  bs*seq_length , vocab_size)
    minibatch_label =   minibatch_label.view(  bs*seq_length )

    loss = criterion(  scores ,  minibatch_label )

    loss.backward()

    utils.normalize_gradient(net)
    optimizer.step()
```

Training loop of VRNNs

```
for count in range( 0 , 46478-seq_length ,  seq_length):

    optimizer.zero_grad()

    minibatch_data =  train_data[ count    : count+seq_length    ]
    minibatch_label = train_data[ count+1 : count+seq_length+1 ]

    minibatch_data=minibatch_data.to(device)
    minibatch_label=minibatch_label.to(device)

    h=h.detach()
    h=h.requires_grad_()

    scores , h  =  net( minibatch_data, h )

    scores          =            scores.view(  bs*seq_length , vocab_size)
    minibatch_label =   minibatch_label.view(  bs*seq_length )

    loss = criterion(  scores ,  minibatch_label )

    loss.backward()

    utils.normalize_gradient(net)
    optimizer.step()
```

Prevent gradient explosion in RNNs.

Update weights.

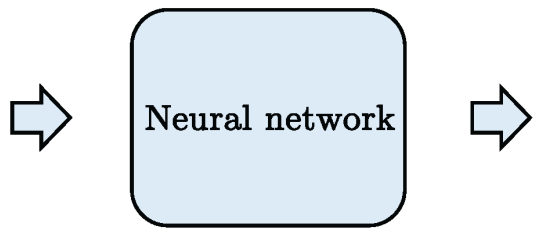
Outline

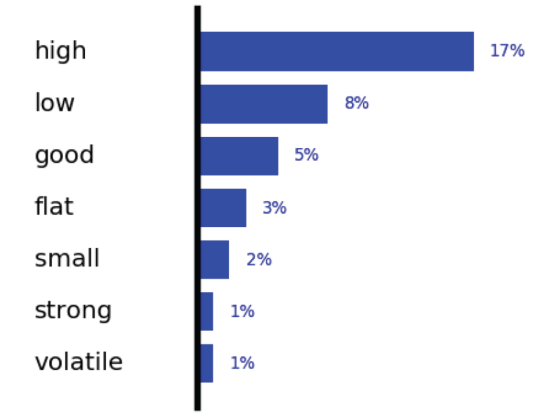
- Word embedding
- Vanilla RNN implementation
- Batch of documents
- Training loop of VRNNs
- **Application to NLP**
- **LSTM**
- **Deep RNNs and RNN variants**
- **Conclusion**

Application to NLP

- NLP task:
  - Learn the dynamics of sequences of words.
  - At test time, predicting the next word:

"some analysts expect oil prices to remain relatively ..."





Input:

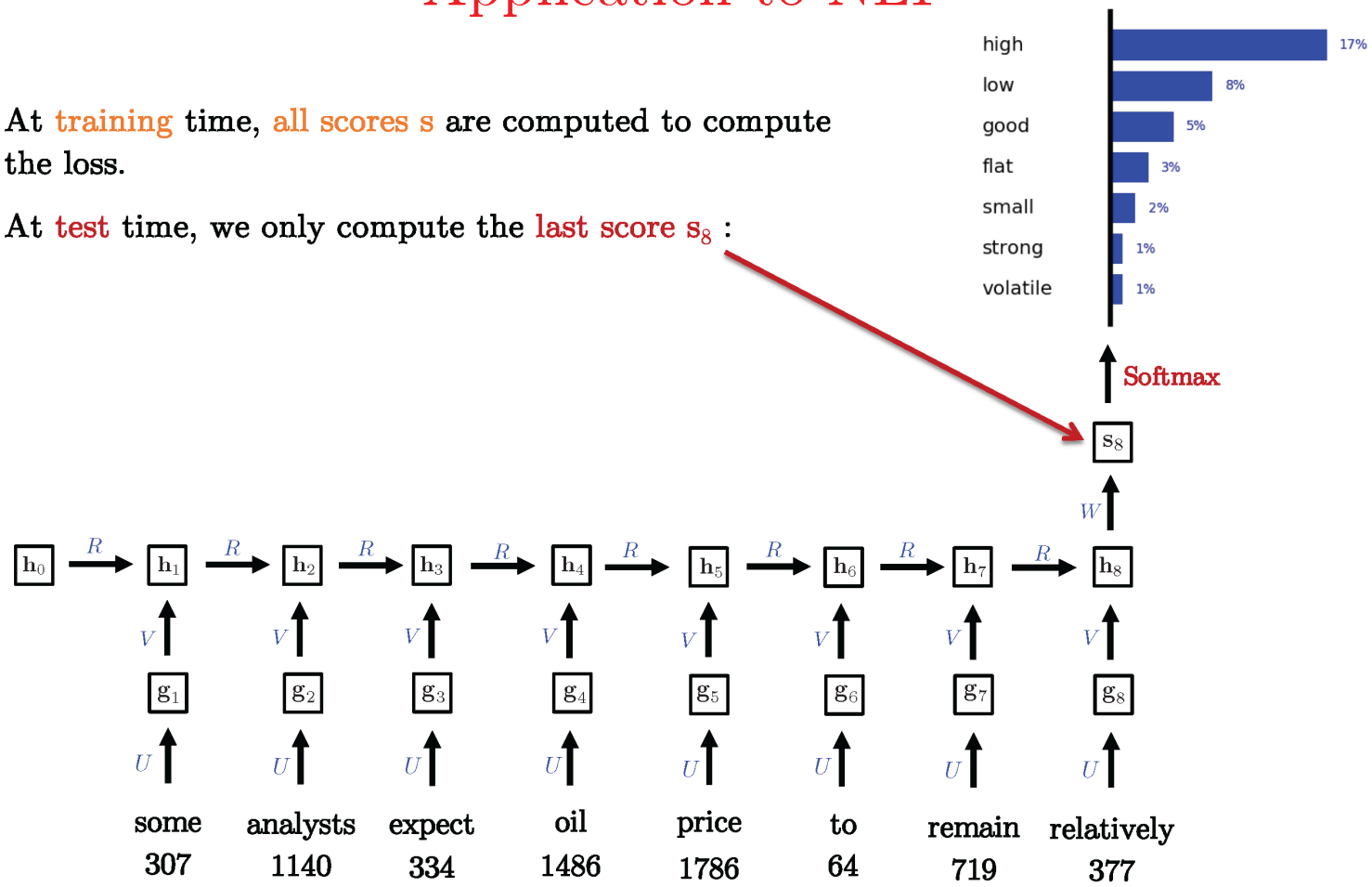
Sequence of words

Output:

Probability distribution over the dictionary/vocabulary

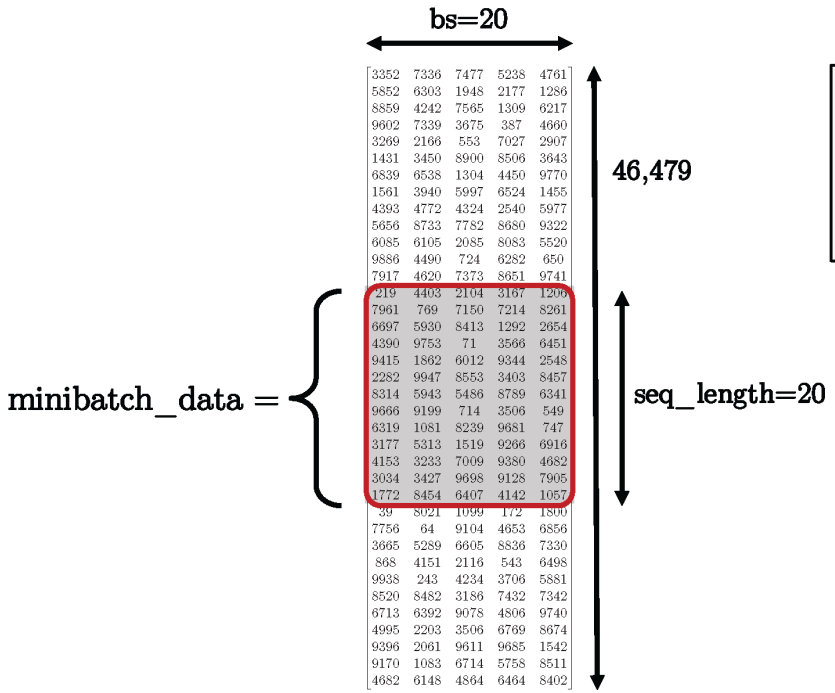
Application to NLP

- At training time, all scores s are computed to compute the loss.
- At test time, we only compute the last score s_8 :



Application to NLP

- Input data during training time :



1 word

bs documents

```
h = torch.zeros(1, bs, hidden_size)

minibatch_data =  train_data[ count : count+seq_length ]

scores, h  = net( minibatch_data, h )
```

Application to NLP

- Input data at test time :

```
h = torch.zeros(1, 1, hidden_size)

minibatch_data = utils.sentence2vector(mysentence)

scores, h  = net( minibatch_data, h )
```

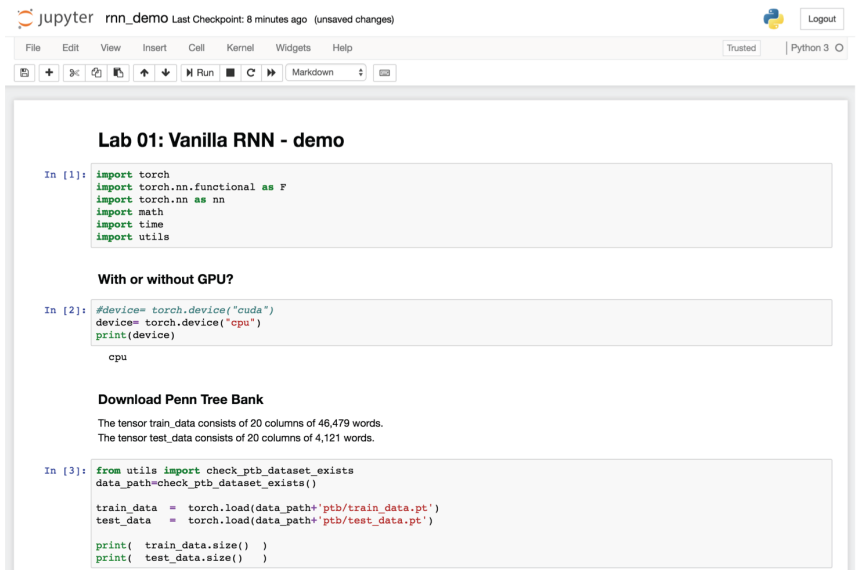
"some analysts expect oil prices to remain relatively ..."

bs=1
(1 document)

$$\begin{bmatrix} 307 \\ 1140 \\ 334 \\ 1486 \\ 1786 \\ 64 \\ 719 \\ 377 \end{bmatrix}$$

Lab 01

- **Vanilla RNN - demo**



Outline

- Word embedding
- Vanilla RNN implementation
- Batch of documents
- Training loop of VRNNs
- Application to NLP
- **LSTM**
- Deep RNNs and RNN variants
- Conclusion

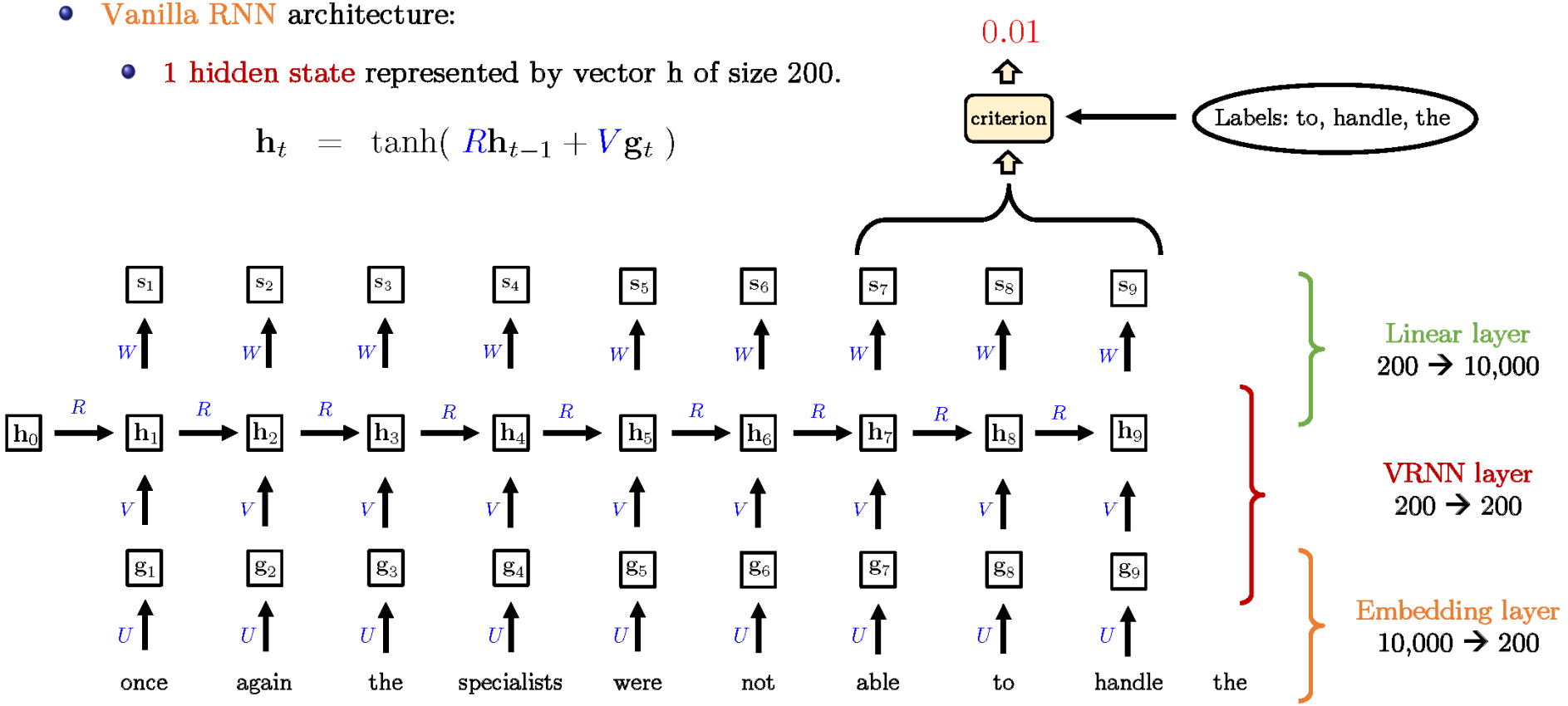
Long Short-Term Memory Networks (LSTM)

- Vanilla RNNs suffer from the vanishing gradient problem :
  - They cannot learn long-term dependencies (beyond 5-10 words).
- LSTM networks are less prone to the vanishing problem by design :
  - These RNNs use a short-term memory state (like VRNNs) and a long-term memory state.
- LSTM are the most used RNNs :
  - Applications : Translation, speech recognition, image captioning, etc

Vanilla RNN

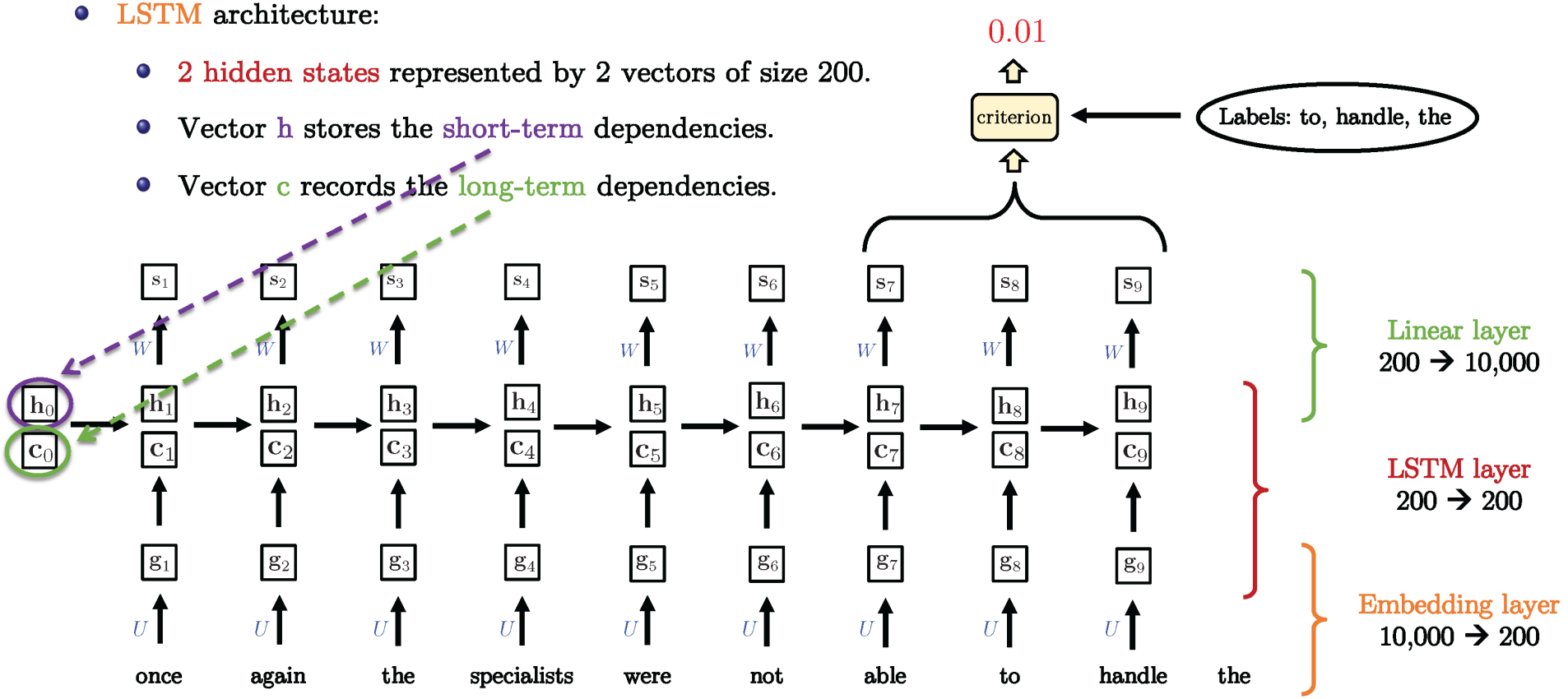
- Vanilla RNN architecture:
  - 1 hidden state represented by vector h of size 200.

$$\mathbf{h}_t = \tanh(R\mathbf{h}_{t-1} + V\mathbf{g}_t)$$



LSTM

- LSTM architecture:
  - 2 hidden states represented by 2 vectors of size 200.
  - Vector h stores the short-term dependencies.
  - Vector c records the long-term dependencies.



LSTM

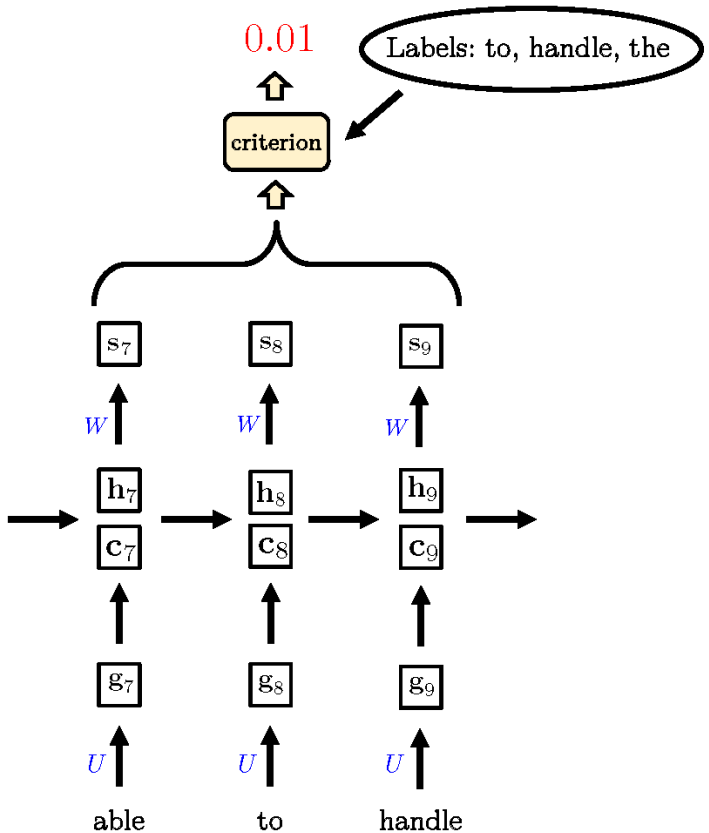
- LSTM recurrence formula:

$$\begin{aligned}
\hat{h}_t &= \tanh(\, R\mathbf{h}_{t-1} + V\mathbf{g}_t \,) \\
\mathbf{c}_t &= \theta_t \odot \mathbf{c}_{t-1} \;+\; \eta_t \odot \hat{h}_t \\
\mathbf{h}_t &= \psi_t \odot \tanh(\mathbf{c}_t)
\end{aligned}$$

Component-wise multiplication

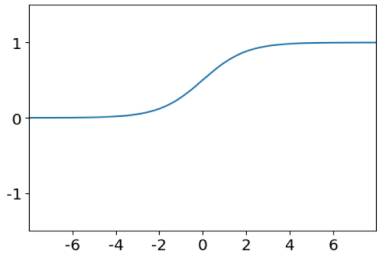
$$\begin{aligned}
\theta_t &= \mathrm{sigm}(A\mathbf{h}_{t-1} + B\mathbf{g}_t) \quad \text{“forget gate”} \\
\text{with} \quad \eta_t &= \mathrm{sigm}(C\mathbf{h}_{t-1} + D\mathbf{g}_t) \quad \text{“input gate”} \\
\psi_t &= \mathrm{sigm}(E\mathbf{h}_{t-1} + F\mathbf{g}_t) \quad \text{“output gate”}
\end{aligned}$$

- Let us analyze this recurrence formula.



Gating mechanism

- A gate/switch is defined as:
  - Closed : $\theta_t \approx 0$
  - In-between : $0 < \theta_t < 1$
  - Open : $\theta_t \approx 1$



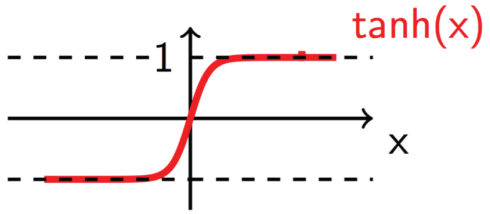
$$\text{sigm}(x) = \frac{1}{1+e^{-x}}$$

- An LSTM cell has three gates :

$$\begin{aligned}
\theta_t &= \text{sigm}(A\mathbf{h}_{t-1} + B\mathbf{g}_t) && \text{“forget gate”} \\
\eta_t &= \text{sigm}(C\mathbf{h}_{t-1} + D\mathbf{g}_t) && \text{“input gate”} \\
\psi_t &= \text{sigm}(E\mathbf{h}_{t-1} + F\mathbf{g}_t) && \text{“output gate”}
\end{aligned}$$

Flow of information

- A flow (of information) is defined be:
  - Positive
  - Negative



$$\hat{\boldsymbol{h}}_t = \tanh(R\mathbf{h}_{t-1} + V\mathbf{g}_t)$$

$$\mathbf{c}_t = \theta_t \odot \mathbf{c}_{t-1} + \eta_t \odot \hat{\boldsymbol{h}}_t$$

$$\mathbf{h}_t = \psi_t \odot \tanh(\mathbf{c}_t)$$

Long-term memory

- Formula :
  - At every time step, the vector c make a choice between :
    - Updating its value according to the new input.
    - Ignoring the new input and not changing.

$$\mathbf{c}_t = \underbrace{\theta_t \odot \mathbf{c}_{t-1}}_{} + \underbrace{\eta_t \odot \hat{h}_t}_{}$$

If θ=0, the memory c is wiped out, and the new memory is given from the input.

Forget gate

Input gate

$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 1.0 \\ 1.0 \\ 0.0 \\ 0.4 \end{bmatrix} \odot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} + \begin{bmatrix} 0.0 \\ 0.0 \\ 1.0 \\ 0.5 \end{bmatrix} \odot \begin{bmatrix} \hat{h}_1 \\ \hat{h}_2 \\ \hat{h}_3 \\ \hat{h}_4 \end{bmatrix}$$

New memory cell — Old memory cell — Freshly computed $\hat{h}$

Long-term memory

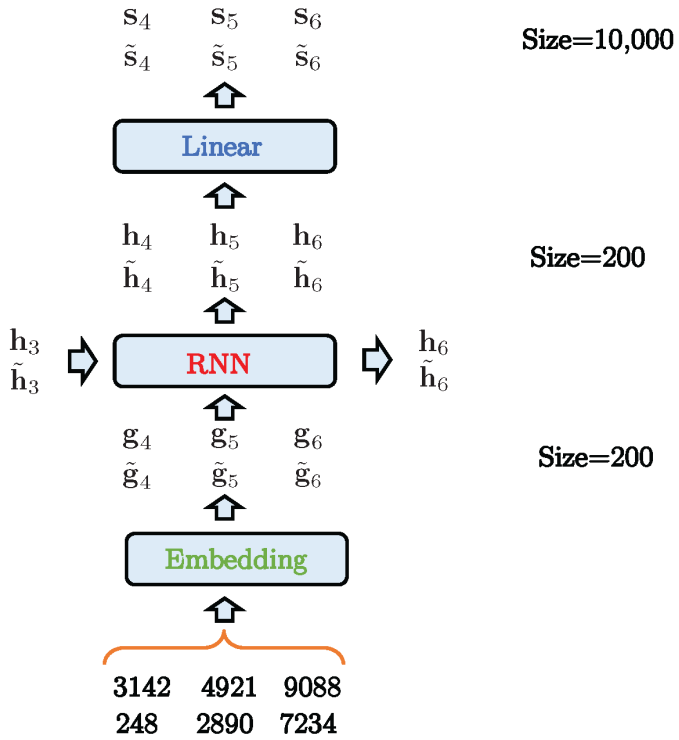
- Formula :

$$\mathbf{c}_t = \underbrace{\theta_t \odot \mathbf{c}_{t-1}} + \eta_t \odot \hat{\boldsymbol{h}}_t$$

  - If θ=1, the new memory c_t is directly copied from the old memory c_{t-1}.
  - The gradient does not loose any amplitude during backpropagation and the information can flow from the 1st word to the 1,000,000th word – That is theoretically long-term memory/dependencies.
  - VRNNs do not have this property.

Vanilla RNNs

- Implementation :




```
mod = nn.RNN(200,200)
```

```
h_seq , h_final = mod( g_seq , h_init )
```

Tensor of size
2 x 3 x 200

Tensor of size
2 x 3 x 200

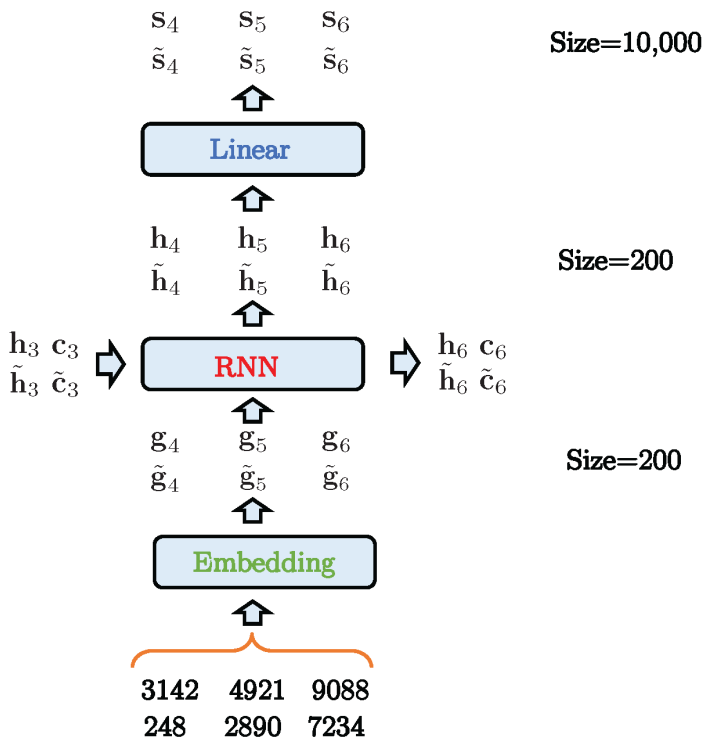
Tensor of size
2 x 200

Tensor of size
2 x 200

- bs = 2 (batch of documents)
- seq_length = 3 (batch of words)
- hidden_size = 200

LSTM

- Implementation :




```
mod = nn.LSTM(200,200)
```

```
h_seq , (h_final,c_final)  =  mod( g_seq , (h_init,c_init) )
```

Tensor of size
2 x 3 x 200

Tensor of size
2 x 3 x 200

2 tensors of size
2 x 200

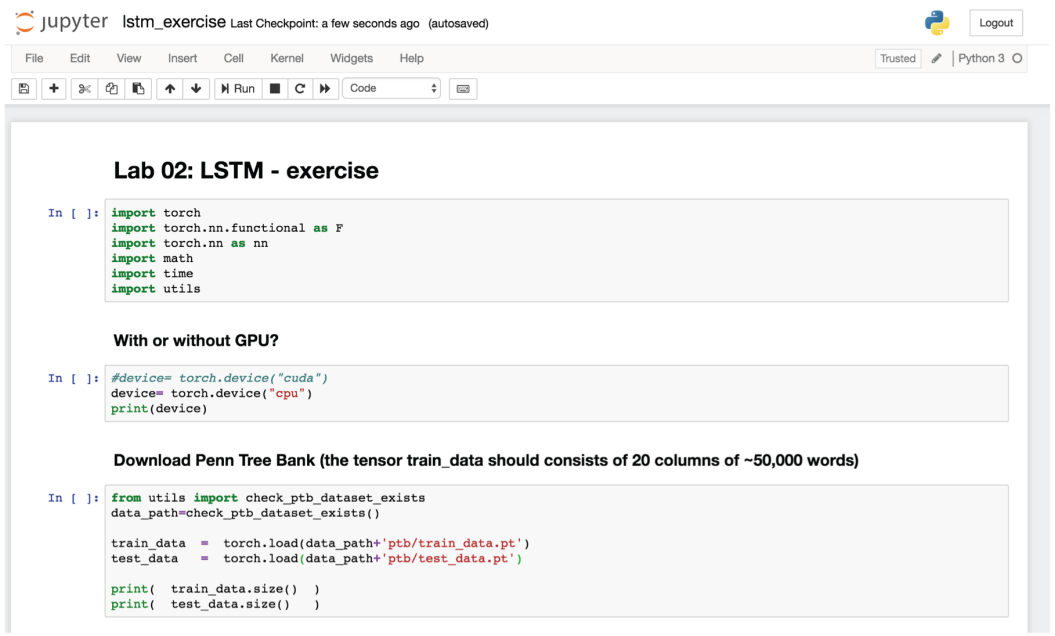
2 tensors of size
2 x 200

- bs = 2 (batch of sub-documents)
- seq_length = 3 (batch of words)
- hidden_size = 200

Intermediate long-term memory vectors c are not returned because they are not needed (unlike h).

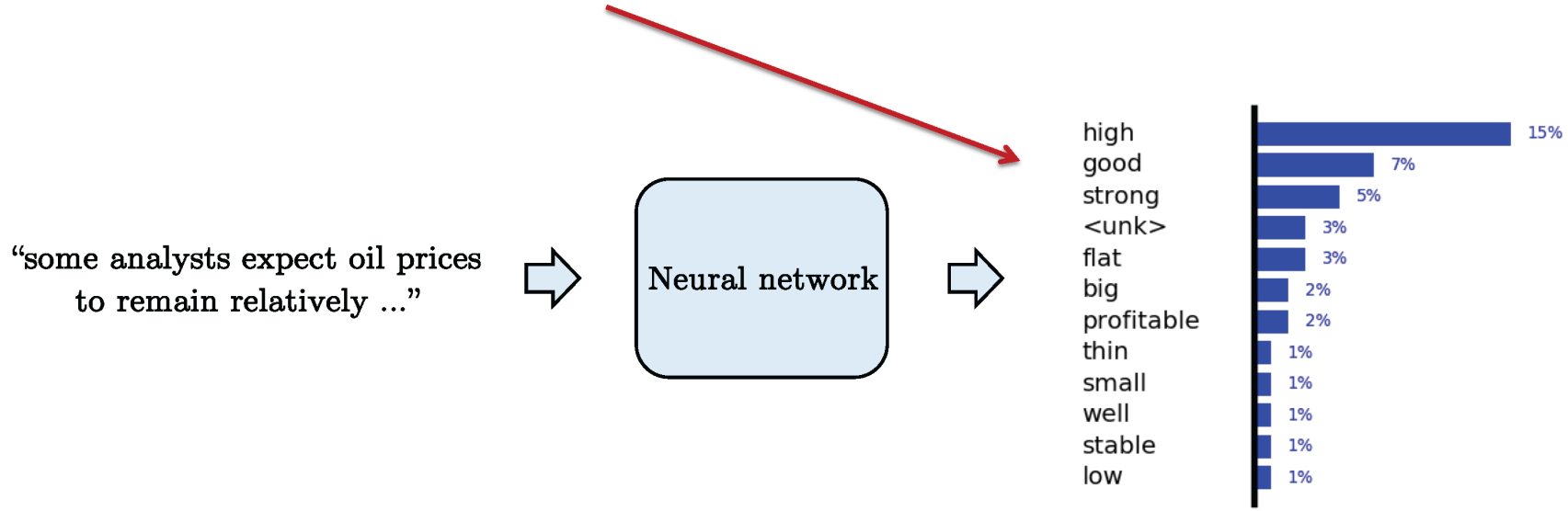
Lab 02

- LSTM implementation :



Lab 02

- Numerical results :
  - The solution should look like this, but each run may give a slightly different result.



Lab 02

- Numerical results :
  - Vanilla RNNs: exp(loss) = 135 — 3 million parameters
  - LSTM: exp(loss) = 107 — 7 million parameters
  - Current state-of-the-art: exp(loss) = 50 — 50 millions parameters

LSTM + many tricks

Perplexity = exp(loss)

Measure how much bits are needed to represent the distribution of words in PTB.

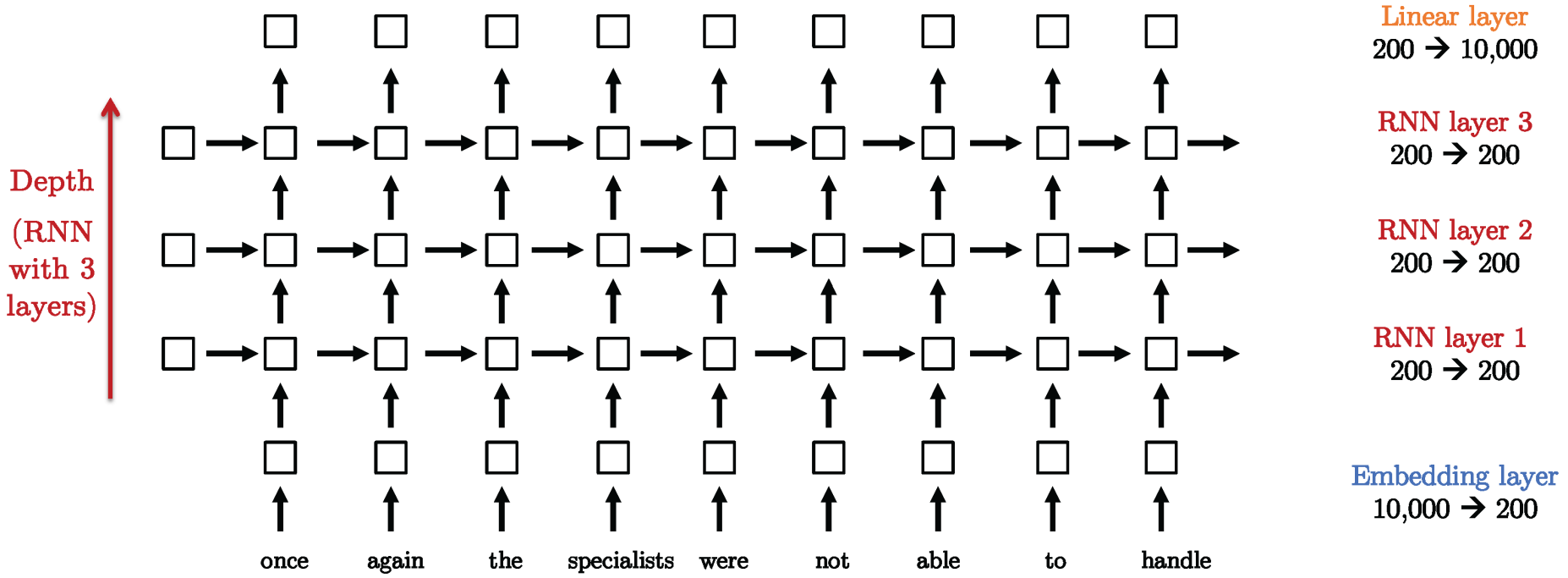
Note that we are still optimizing the loss, and not the perplexity (only use for performance comparison.)

Outline

- Word embedding
- Vanilla RNN implementation
- Batch of documents
- Training loop of VRNNs
- Application to NLP
- LSTM
- **Deep RNNs and RNN variants**
- Conclusion

Deep RNNs

- Stacking recurrent layers on the top of each other :



Application: Google Translation machine, since Nov. 2016

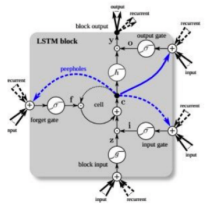
RNN Variants

- Several variants of RNNs exist.
  - Gated Recurrent Unit (GRU), 2014

$$\tilde{\mathbf{h}}_t = \tanh(E(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + F\mathbf{x}_t)$$
$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \tilde{\mathbf{h}}_t$$

with
$$\mathbf{r}_t = \text{sigm}(A\mathbf{h}_{t-1} + B\mathbf{x}_t)$$
$$\mathbf{z}_t = \text{sigm}(C\mathbf{h}_{t-1} + D\mathbf{x}_t)$$

- LSTM from 1997 still provides the best performances over many possible experimental conditions !



[*LSTM: A Search Space Odyssey*, Greff et al., 2015]

GRU [*Learning phrase representations using rnn encoder-decoder for statistical machine translation*, Cho et al. 2014]

[*An Empirical Exploration of Recurrent Network Architectures*, Jozefowicz et al., 2015]

MUT1:

$$\begin{aligned} z &= \text{sigm}(W_{xz}x_t + b_z) \\ r &= \text{sigm}(W_{xr}x_t + W_{hr}h_t + b_r) \\ h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + \tanh(x_t) + b_h) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

MUT2:

$$\begin{aligned} z &= \text{sigm}(W_{xz}x_t + W_{hz}h_t + b_z) \\ r &= \text{sigm}(x_t + W_{hr}h_t + b_r) \\ h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + W_{xh}x_t + b_h) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

MUT3:

$$\begin{aligned} z &= \text{sigm}(W_{xz}x_t + W_{hz}\tanh(h_t) + b_z) \\ r &= \text{sigm}(W_{xr}x_t + W_{hr}h_t + b_r) \\ h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + W_{xh}x_t + b_h) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

Outline

- Word embedding
- Vanilla RNN implementation
- Batch of documents
- Training loop of VRNNs
- Application to NLP
- LSTM
- Deep RNNs and RNN variants
- **Conclusion**

Conclusion

- RNNs offer lots of flexibility for designing NN architectures, and consequently they can be applied to many tasks.
- RNNs are not able to learn very long-term dependencies (even LSTM).