

CS11-747 Neural Networks for NLP

# Recurrent Neural Networks

Graham Neubig



**Carnegie Mellon University**

Language Technologies Institute

Site

<https://phontron.com/class/nn4nlp2018/>

# NLP and Sequential Data

- NLP is full of sequential data
  - Words in sentences
  - Characters in words
  - Sentences in discourse
  - ...

# Long-distance Dependencies in Language

- Agreement in number, gender, etc.

**He** does not have very much confidence in **himself**.

**She** does not have very much confidence in **herself**.

- Selectional preference

The **reign** has lasted as long as the life of the **queen**.

The **rain** has lasted as long as the life of the **clouds**.

# Can be Complicated!

- What is the referent of “it”?

The trophy would not fit in the brown suitcase because it was too **big**.

Trophy

The trophy would not fit in the brown suitcase because it was too **small**.

Suitcase

(from Winograd Schema Challenge:

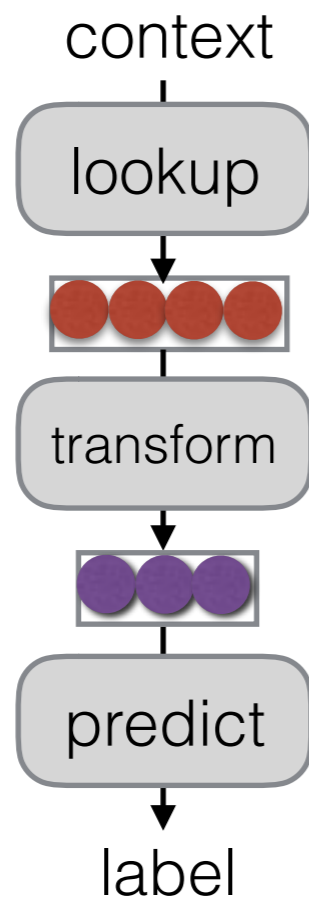
<http://commonsensereasoning.org/winograd.html>)

# Recurrent Neural Networks

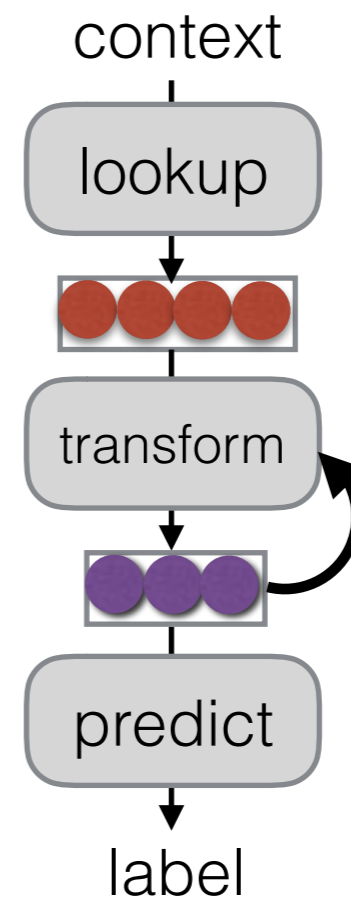
(Elman 1990)

- Tools to “remember” information

Feed-forward NN

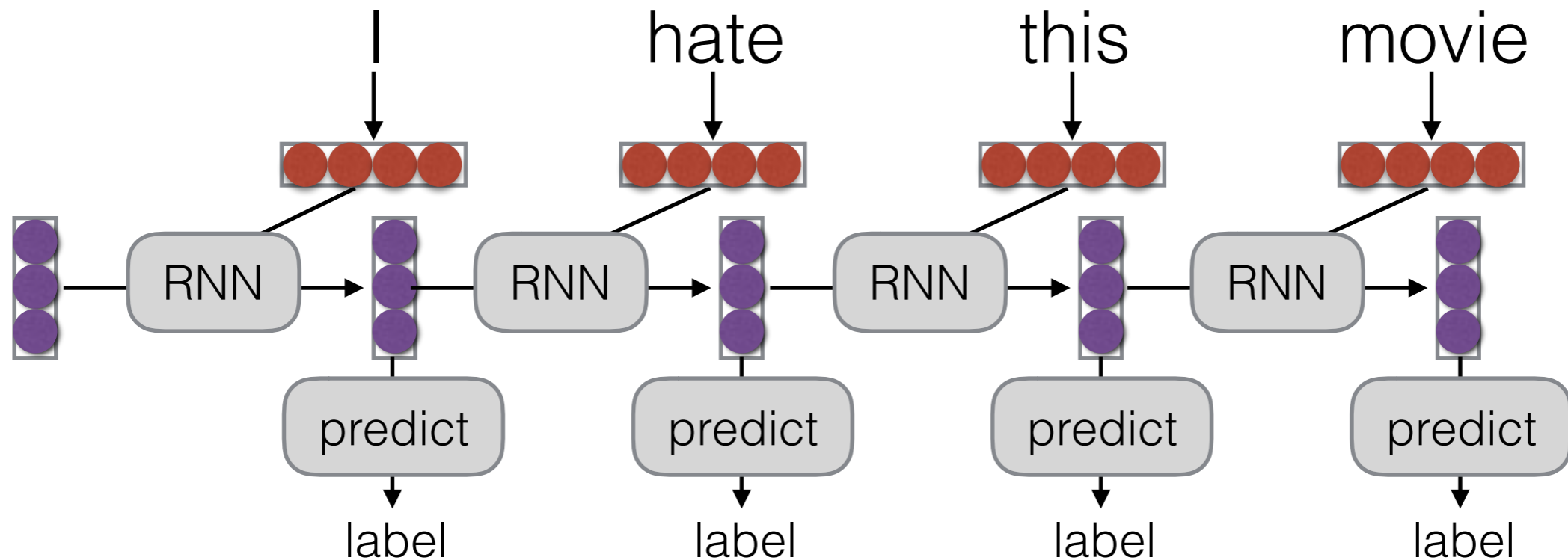


Recurrent NN

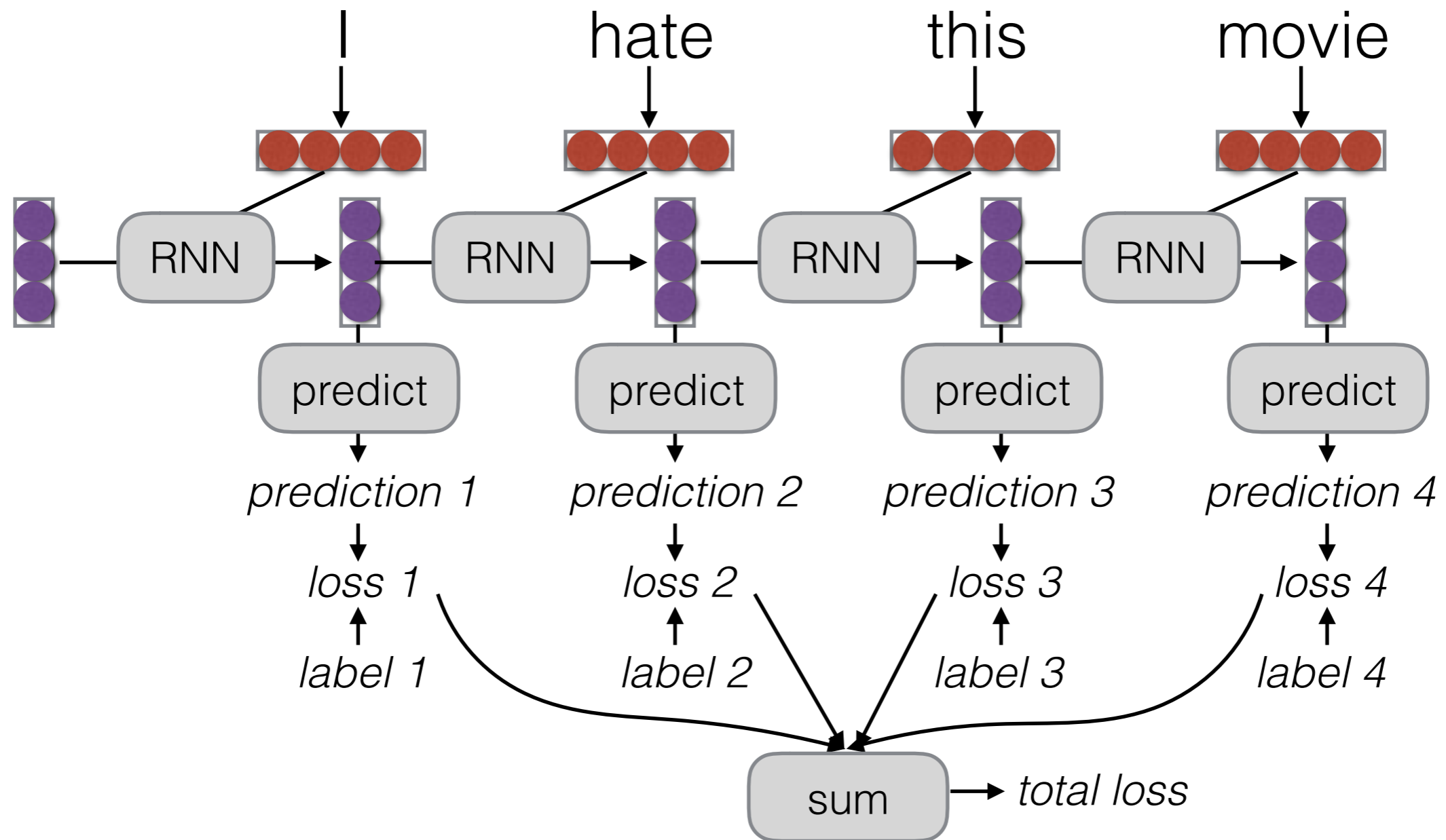


# Unrolling in Time

- What does processing a sequence look like?

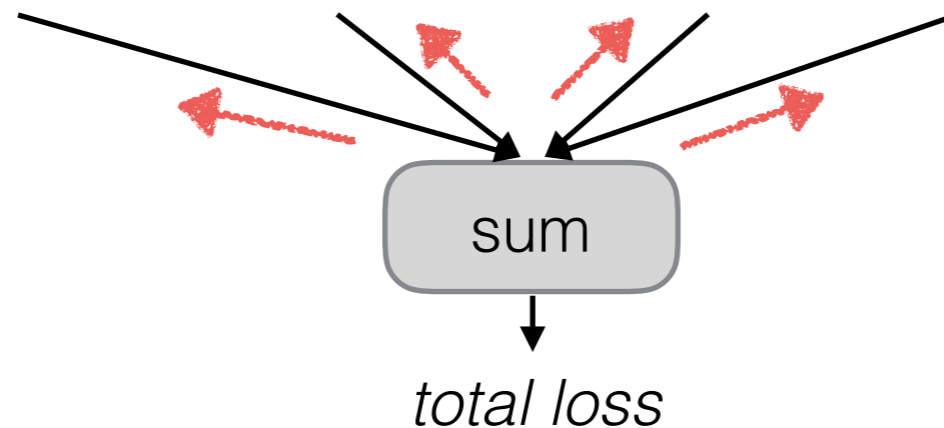


# Training RNNs



# RNN Training

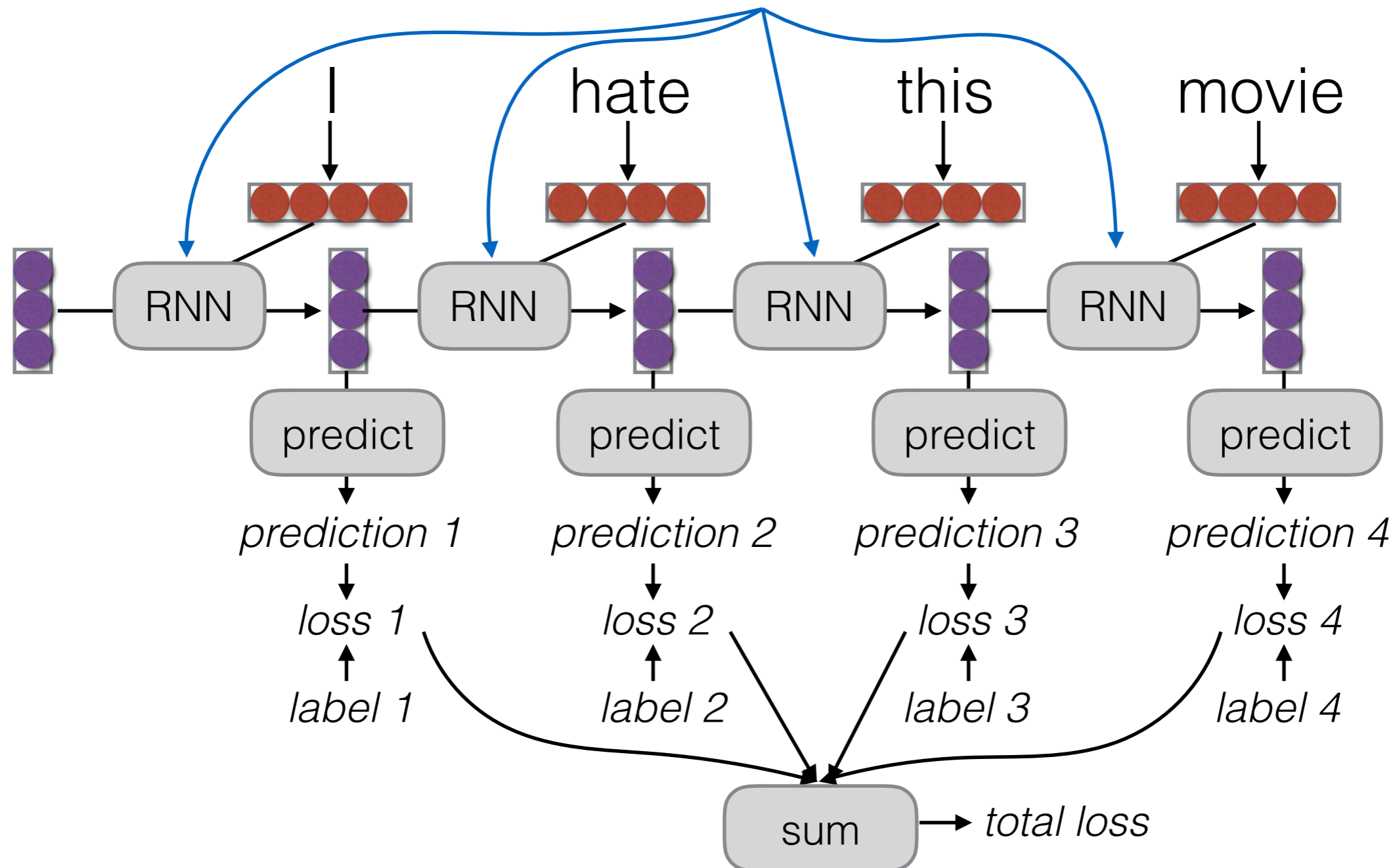
- The unrolled graph is a well-formed (DAG) computation graph—we can run backprop



- Parameters are tied across time, derivatives are aggregated across all time steps
- This is historically called “backpropagation through time” (BPTT)

# Parameter Tying

Parameters are shared! Derivatives are accumulated.

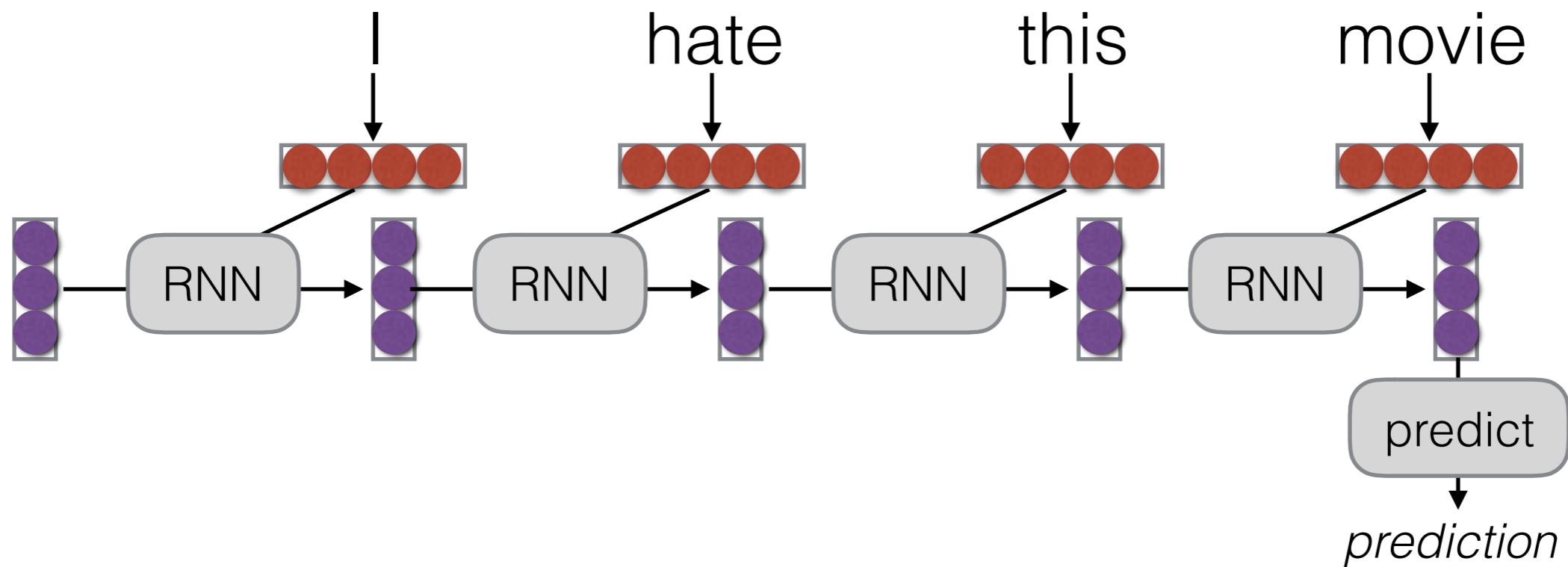


# Applications of RNNs

# What Can RNNs Do?

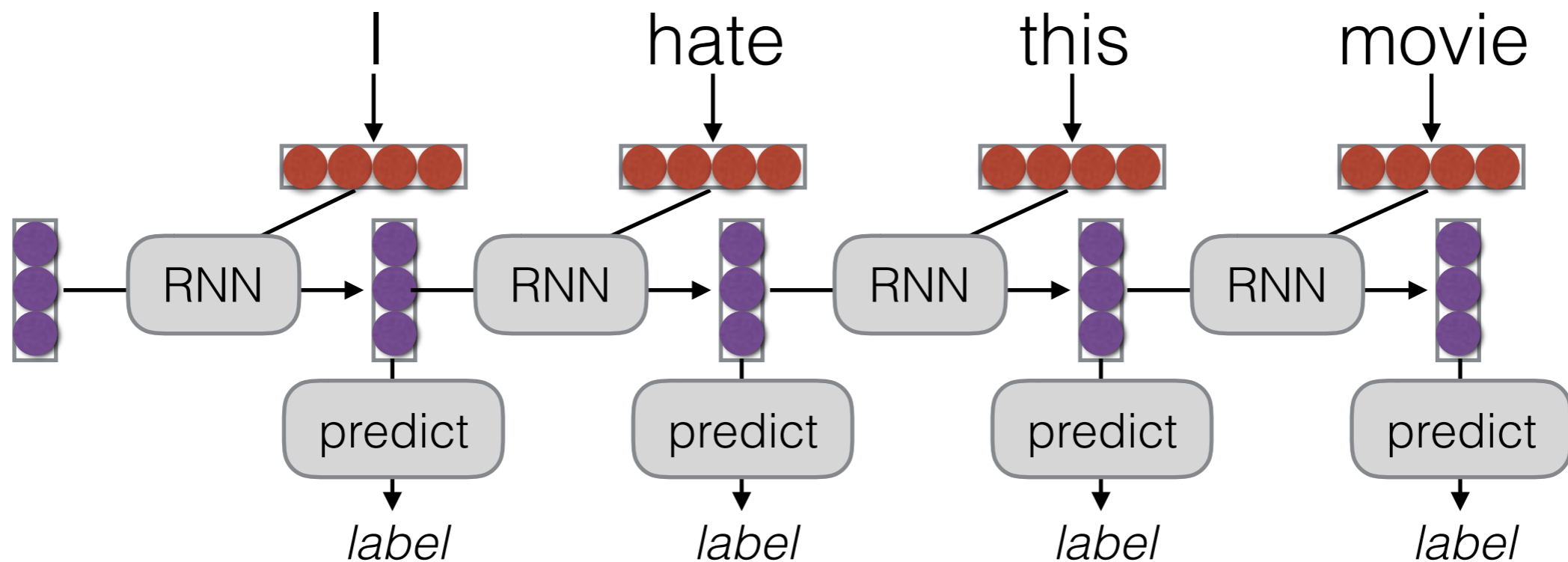
- Represent a sentence
  - Read whole sentence, make a prediction
- Represent a context within a sentence
  - Read context up until that point

# Representing Sentences



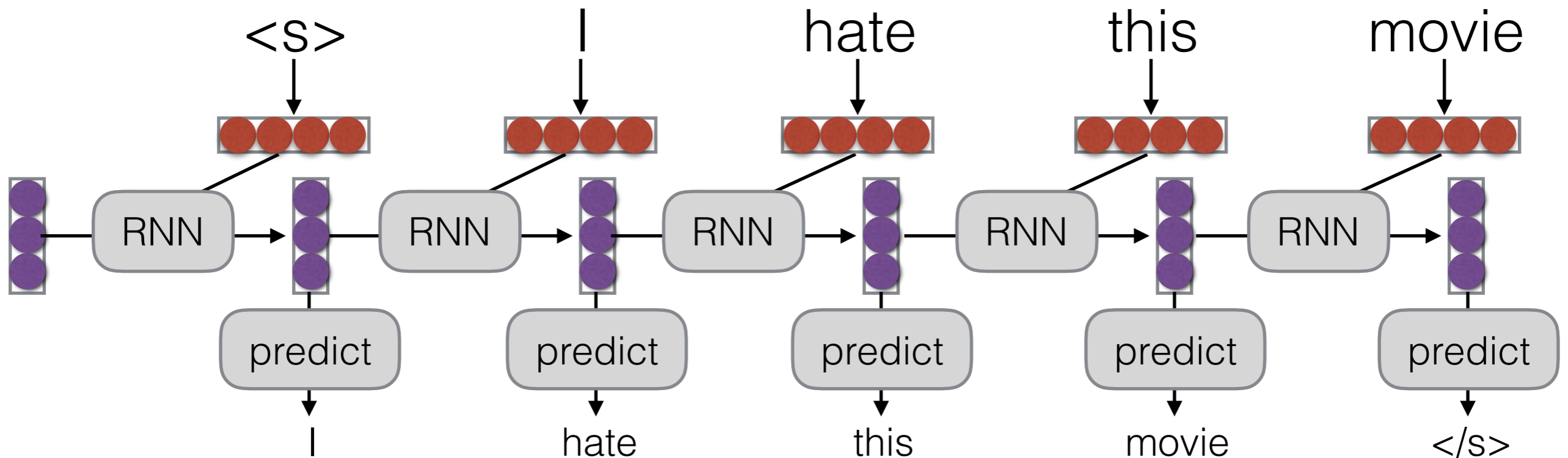
- Sentence classification
- Conditioned generation
- Retrieval

# Representing Contexts



- Tagging
- Language Modeling
- Calculating Representations for Parsing, etc.

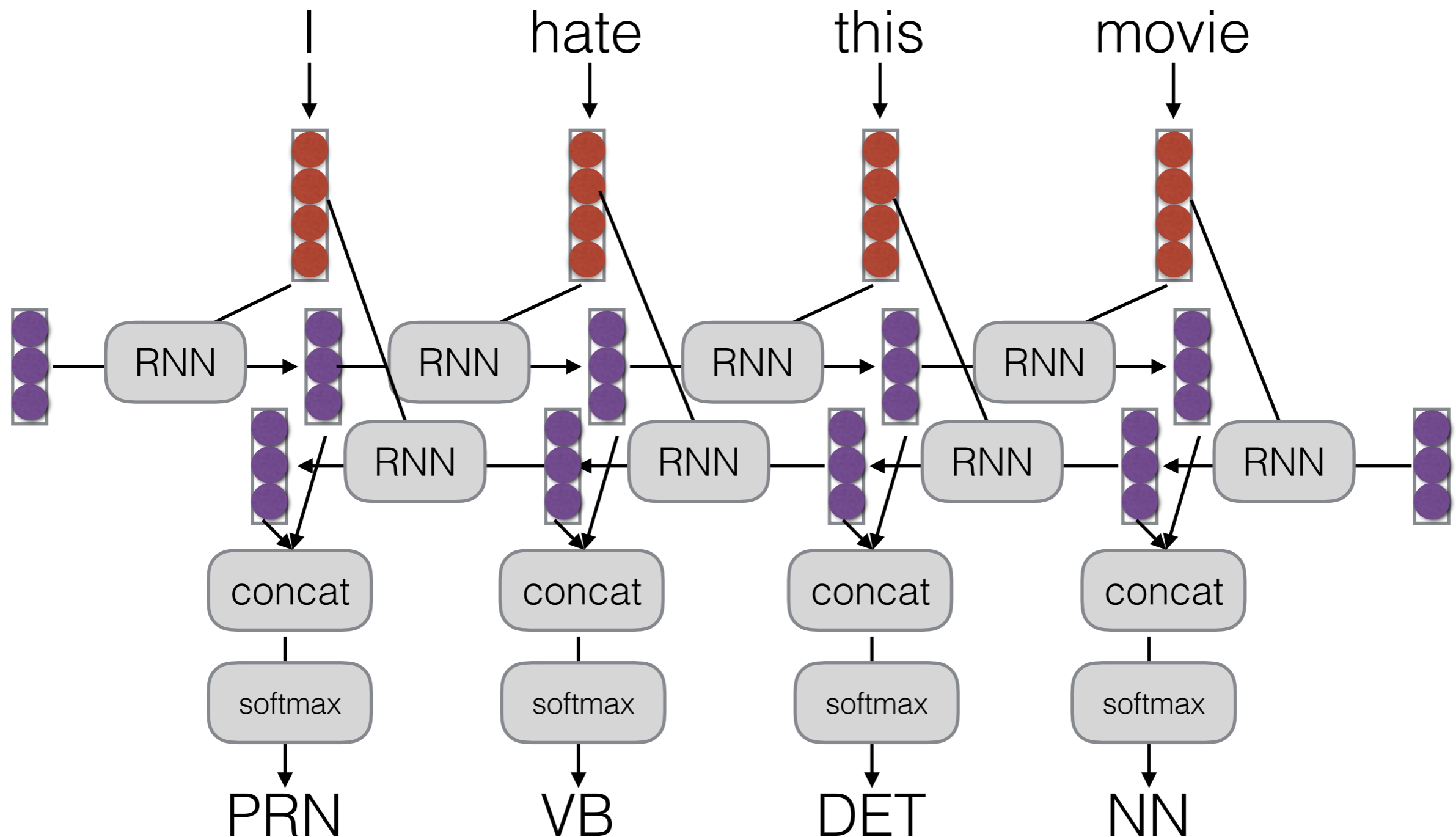
# e.g. Language Modeling



- Language modeling is like a tagging task, where each tag is the next word!

# Bi-RNNs

- A simple extension, run the RNN in both directions



Let's Try it Out!

# Recurrent Neural Networks in DyNet

- Based on “\*Builder” class (\*=SimpleRNN/LSTM)
- Add parameters to model (once):

```
# LSTM (layers=1, input=64, hidden=128, model)  
RNN = dy.SimpleRNNBuilder(1, 64, 128, model)
```

- Add parameters to CG and get initial state (per sentence):

```
s = RNN.initial_state()
```

- Update state and access (per input word/character):

```
s = s.add_input(x_t)  
h_t = s.output()
```

# RNNLM Example: Parameter Initialization

```
# Lookup parameters for word embeddings
WORDS_LOOKUP = model.add_lookup_parameters((nwords, 64))

# Word-level RNN (layers=1, input=64, hidden=128, model)
RNN = dy.SimpleRNNBuilder(1, 64, 128, model)

# Softmax weights/biases on top of RNN outputs
W_sm = model.add_parameters((nwords, 128))
b_sm = model.add_parameters(nwords)
```

# RNNLM Example: Sentence Initialization

```
# Build the language model graph
def calc_lm_loss(wids):
    dy.renew_cg()

    # parameters -> expressions
    W_exp = dy.parameter(W_sm)
    b_exp = dy.parameter(b_sm)

    # add parameters to CG and get state
    f_init = RNN.initial_state()

    # get the word vectors for each word ID
    wembs = [WORDS_LOOKUP[wid] for wid in wids]

    # Start the rnn by inputting "<s>"
    s = f_init.add_input(wembs[-1])
```

...

# RNNLM Example: Loss Calculation and State Update

...

```
# process each word ID and embedding
losses = []
for wid, we in zip(wids, wembs):

    # calculate and save the softmax loss
    score = W_exp * s.output() + b_exp
    loss = dy.pickneglogsoftmax(score, wid)
    losses.append(loss)

    # update the RNN state with the input
    s = s.add_input(we)

# return the sum of all losses
return dy.esum(losses)
```

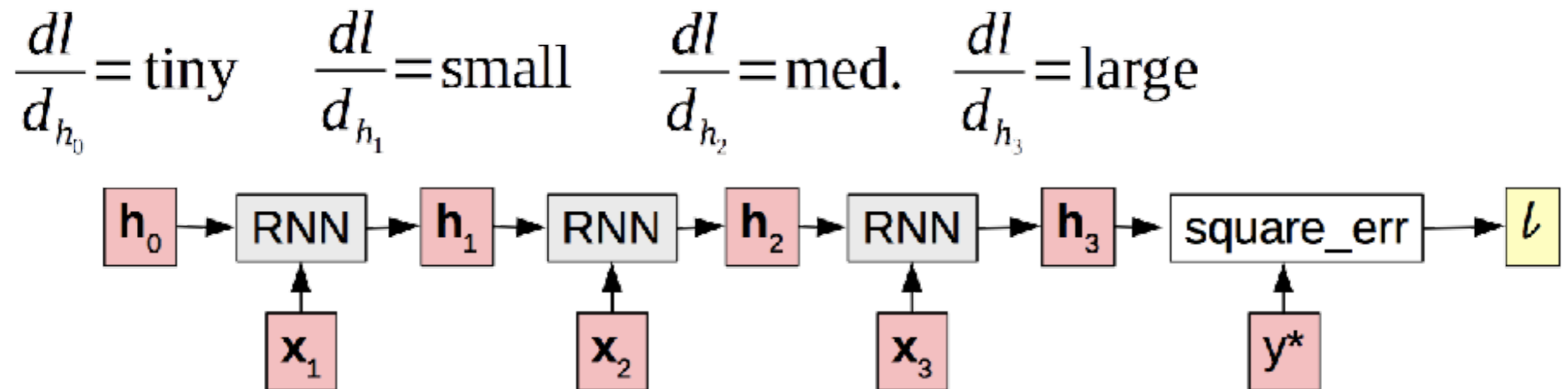
# Code Examples

`sentiment-rnn.py`

# RNN Problems and Alternatives

# Vanishing Gradient

- Gradients decrease as they get pushed back



- Why? “Squashed” by non-linearities or small weights in matrices.

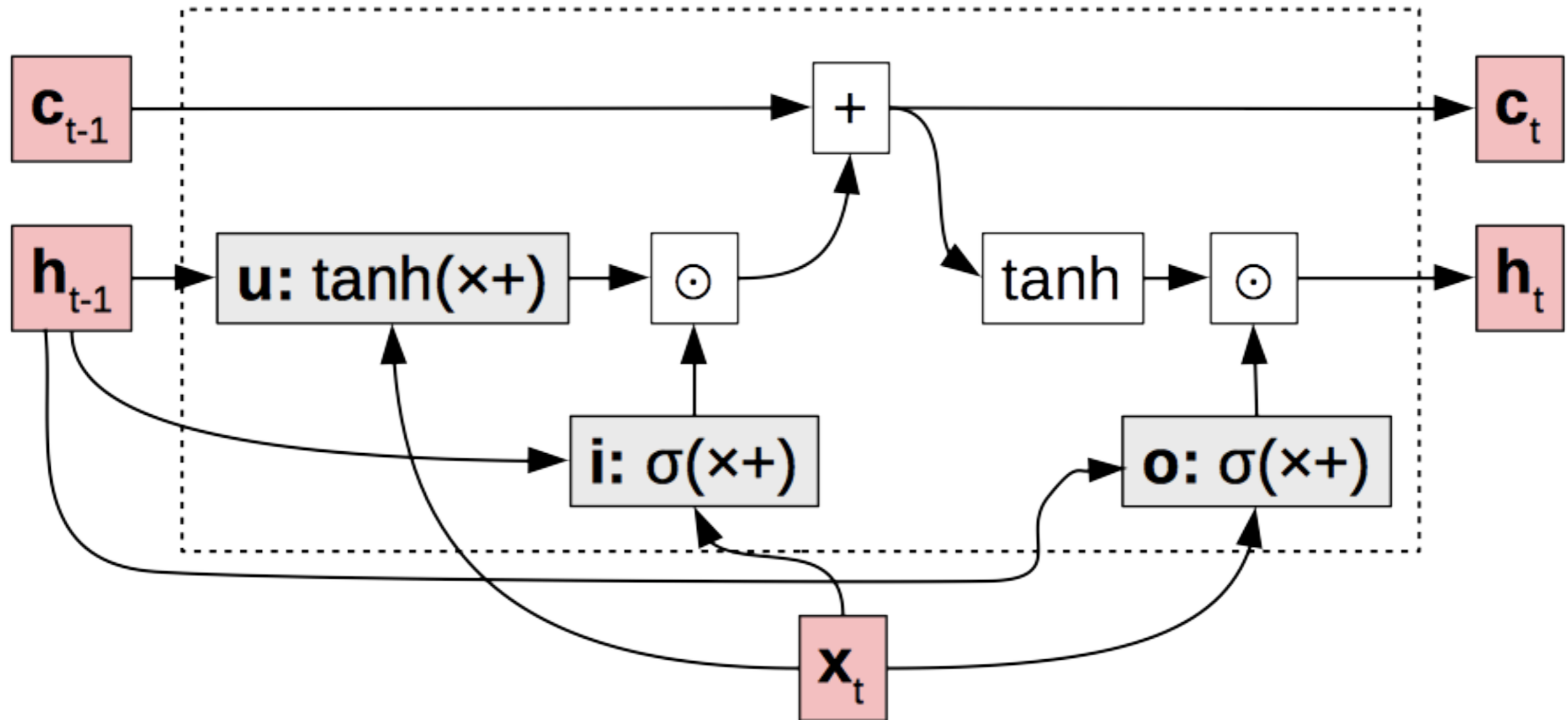
# A Solution:

## Long Short-term Memory

(Hochreiter and Schmidhuber 1997)

- **Basic idea:** make additive connections between time steps
- Addition does not modify the gradient, no vanishing
- Gates to control the information flow

# LSTM Structure



update **u**: what value do we try to add to the memory cell?  
input **i**: how much of the update do we allow to go through?  
output **o**: how much of the cell do we reflect in the next state?

# Other Alternatives

- Lots of variants of LSTMs (Hochreiter and Schmidhuber, 1997)
- Gated recurrent units (GRUs; Cho et al., 2014)
- All follow the basic paradigm of “take input, update state”

# Code Examples

`sentiment-lstm.py`

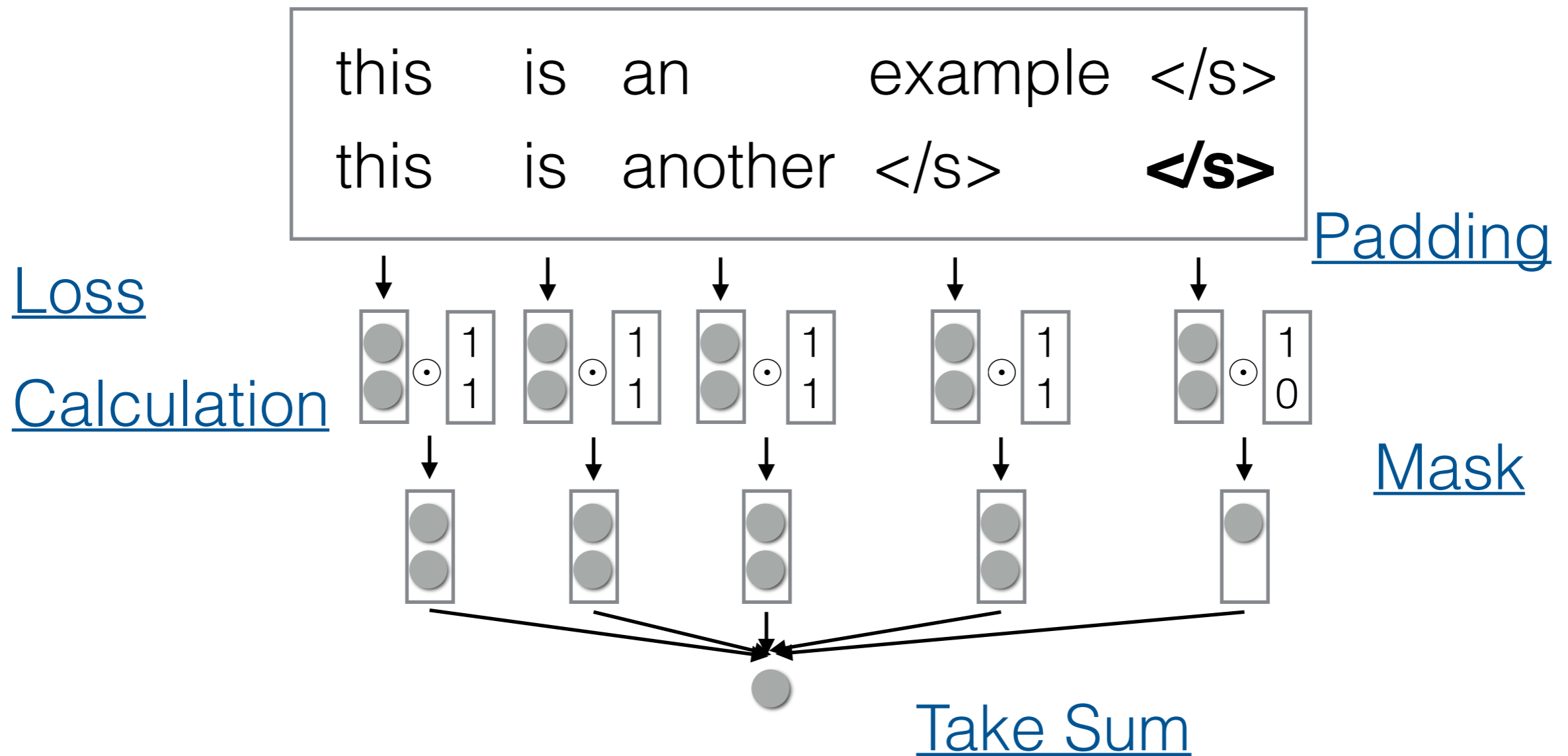
`lm-lstm.py`

# Efficiency/Memory Tricks

# Handling Mini-batching

- Mini-batching makes things much faster!
- But mini-batching in RNNs is harder than in feed-forward networks
  - Each word depends on the previous word
  - Sequences are of various length

# Mini-batching Method



(Or use DyNet automatic mini-batching,  
much easier but a bit slower)

# Bucketing/Sorting

- If we use sentences of different lengths, too much padding and sorting can **result in decreased performance**
- To remedy this: **sort sentences** so similarly-lengthed sentences are in the same batch

# Code Example

`lm-minibatch.py`

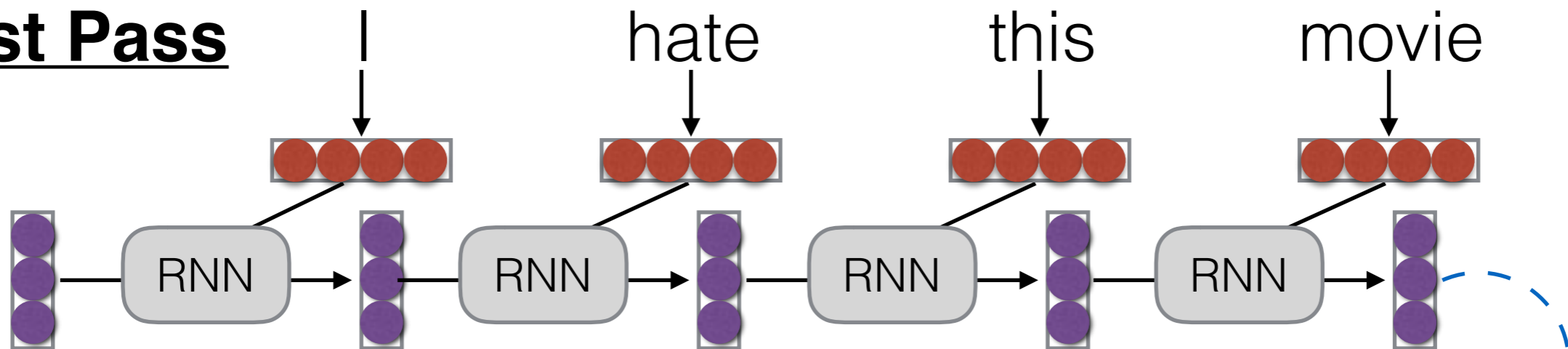
# Handling Long Sequences

- Sometimes we would like to capture long-term dependencies over long sequences
- e.g. words in full documents
- However, this may not fit on (GPU) memory

# Truncated BPTT

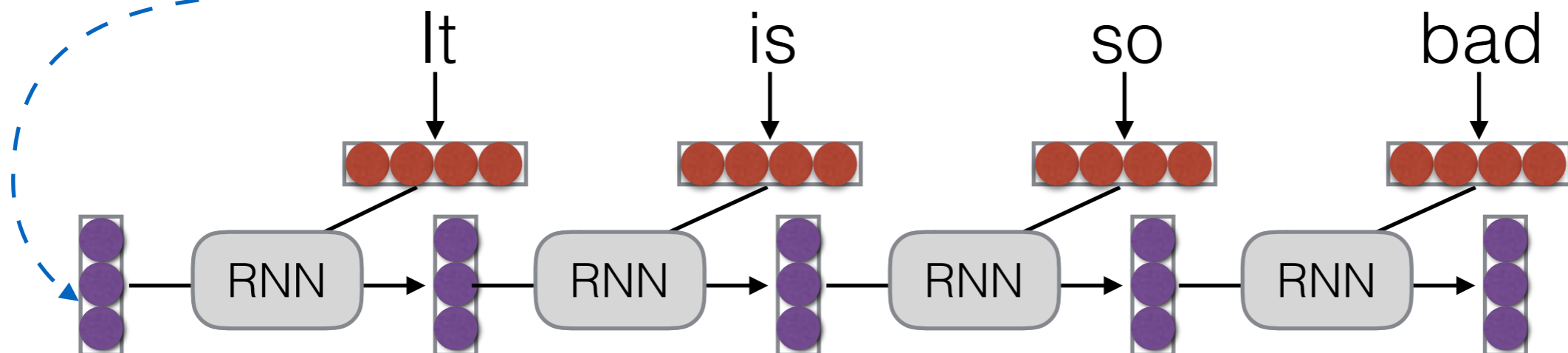
- Backprop over shorter segments, initialize w/ the state from the previous segment

## 1st Pass



## 2nd Pass

state only, no backprop



# Pre-training/Transfer for RNNs

# RNN Strengths/Weaknesses

- RNNs, particularly deep RNNs/LSTMs, are **quite powerful and flexible**
- But they **require a lot of data**
- Also have trouble with **weak error signals** passed back from the end of the sentence

# Pre-training/Transfer

- Train for one task, solve another
- **Pre-training task:** Big data, easy to learn
- **Main task:** Small data, harder to learn

# Example:

## LM -> Sentence Classifier

(Luong et al. 2015)

- Train a **language model first**: lots of data, easy-to-learn objective
- **Sentence classification**: little data, hard-to-learn objective
- Results in much better classifications, competitive or better than CNN-based methods

# Why Pre-training?

- The model learns consistencies in the data (Karpathy et al. 2015)

Cell sensitive to position in line:

```
The sole importance of the crossing of the Beresina lies in the fact that it plainly and indubitably proved the fallacy of all the plans for cutting off the enemy's retreat and the soundness of the only possible line of action--the one Kutuzov and the general mass of the army demanded--namely, simply to follow the enemy up. The French crowd fled at a continually increasing speed and all its energy was directed to reaching its goal. It fled like a wounded animal and it was impossible to block its path. This was shown not so much by the arrangements it made for crossing as by what took place at the bridges. When the bridges broke down, unarmed soldiers, people from Moscow and women with children who were with the French transport, all--carried on by vis inertiae--pressed forward into boats and into the ice-covered water and did not surrender.
```

Cell that turns on inside quotes:

```
"You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties." Warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.
```

```
Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."
```

Cell that robustly activates inside if statements:

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask, siginfo_t *info)
{
    int sig = next_signal(pending, mask);
    if (sig) {
        if (current->notifier) {
            if (sigismember(current->notifier_mask, sig)) {
                if (! (current->notifier)(current->notifier_data)) {
                    clear_thread_flag(TIF_SIGPENDING);
                    return 0;
                }
            }
        }
        collect_signal(sig, pending, info);
    }
    return sig;
}
```

A large portion of cells are not easily interpretable. Here is a typical example:

```
/* Unpack a filter field's string representation from user-space
 * buffer. */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
    char *str;
    if (!*bufp || (len == 0) || (len > *remain))
        return ERR_PTR(-EINVAL);
    /* Of the currently implemented string fields, PATH_MAX
     * defines the longest valid length.
     */
```

Cell that turns on inside comments and quotes:

```
/* Duplicate LSM field information. The lsm_rule is opaque, so
 * re-initialized. */
static inline int audit_dupe_lsm_field(struct audit_field *df,
                                     struct audit_field *sf)
{
    int ret = 0;
    char *lsm_str;
    /* Our own copy of lsm_str */
    lsm_str = kstrdup(sf->lsm_str, GFP_KERNEL);
    if (unlikely(!lsm_str))
        return -ENOMEM;
    df->lsm_str = lsm_str;
    /* Our own (refreshed) copy of lsm_rule */
    ret = security_audit_rule_init(df->type, df->op, df->lsm_str,
                                   (void **) &df->lsm_rule);
    /* Keep currently invalid fields around in case they
     * become valid after a policy reload. */
    if (ret == -EINVAL) {
        pr_warn("audit rule for LSM '%s' is invalid\n",
                df->lsm_str);
        ret = 0;
    }
    return ret;
}
```

Cell that is sensitive to the depth of an expression:

```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
    int i;
    if (classes[class]) {
        for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
            if (mask[i] & classes[class][i])
                return 0;
    }
    return 1;
}
```

Cell that might be helpful in predicting a new line. Note that it only turns on for some "Y":

```
char *audit_unpack_string(void **bufp, size_t *remain, si
{
    char *str;
    if (!*bufp || (len == 0) || (len > *remain))
        return ERR_PTR(-EINVAL);
    /* Of the currently implemented string fields, PATH_MAX
     * defines the longest valid length.
     */
    if (len > PATH_MAX)
        return ERR_PTR(-ENAMETOOLONG);
    str = kmalloc(len + 1, GFP_KERNEL);
    if (unlikely(!str))
        return ERR_PTR(-ENOMEM);
    memcpy(str, *bufp, len);
    str[len] = 0;
    *bufp += len;
    *remain -= len;
    return str;
}
```

- Model learns syntax (Shi et al. 2017) or semantics (Radford et al. 2017)

Questions?