# (Neural) Language models

CMSC 723 / LING 723 / INST 725

Hal Daumé III [he/him]

10 Oct 2019

(many slides c/o Marine Carpuat or Graham Neubig)

# Announcements, logistics

- Class project, a few updates/edits
  - Team size: 4-8, expectations commensurate with team size
  - Pitches will be:
    - One slide, two minutes: answer **what** and **how**
    - One page PDF project description to answer also **why** and **how to measure success**
  - You will provide feedback both on the presentation & document

  - If you're working on a project from outside class, that's fine
    *just be sure to describe how it relates to what you've learned here*

# Last Time

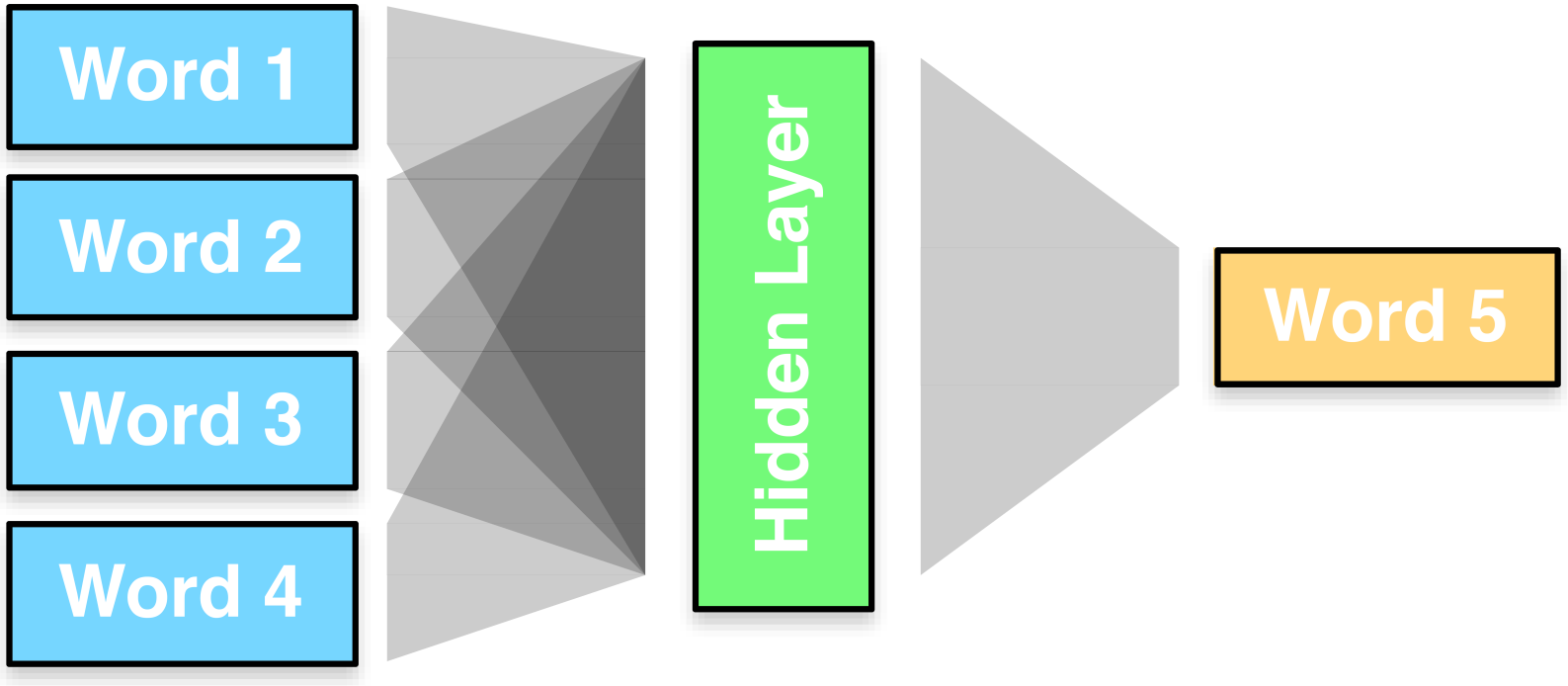
- Language modeling
  - What is it?
  - Ngrams for language modeling
  - Evaluation: perplexity
  - Smoothing

# Today

- Language modeling as a prediction problem
- Feed-forward neural language models (& embeddings)
- Recurrent neural language models

# Feed-forward neural language models

# Toward a Neural Language Model




Figures by Philipp Koehn (JHU)

# Count-based n-gram models vs. feedforward neural networks

- Pros of feedforward neural LM
  - Word embeddings capture generalizations across word typesq
- Cons of feedforward neural LM
  - Closed vocabulary
  - Training/testing is more computationally expensive
- Weaknesses of both types of model
  - Only work well for word prediction if the test corpus looks like the training corpus
  - Only capture short distance context

# Recurrent neural network language models

# 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

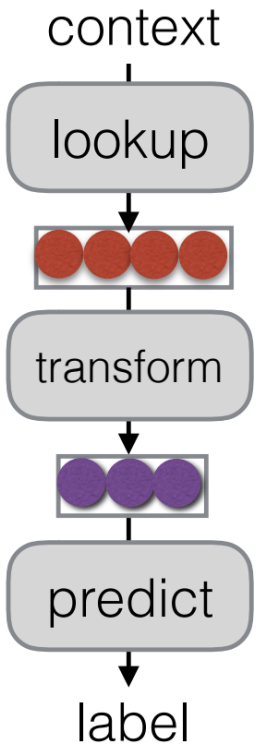
The **reign** has lasted as long as the life of the **queen**.
The **rain** has lasted as long as the life of the **clouds**.

# Recurrent Neural Networks

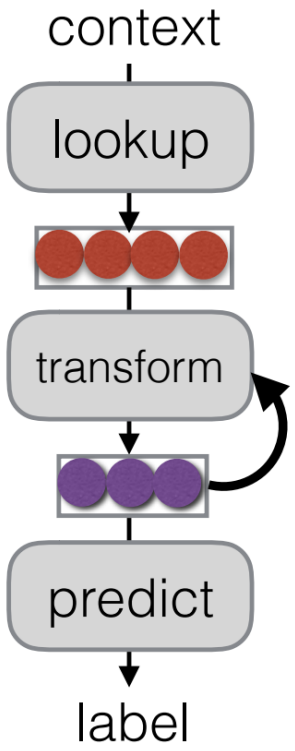
## (Elman 1990)

- Tools to “remember” information

Feed-forward NN

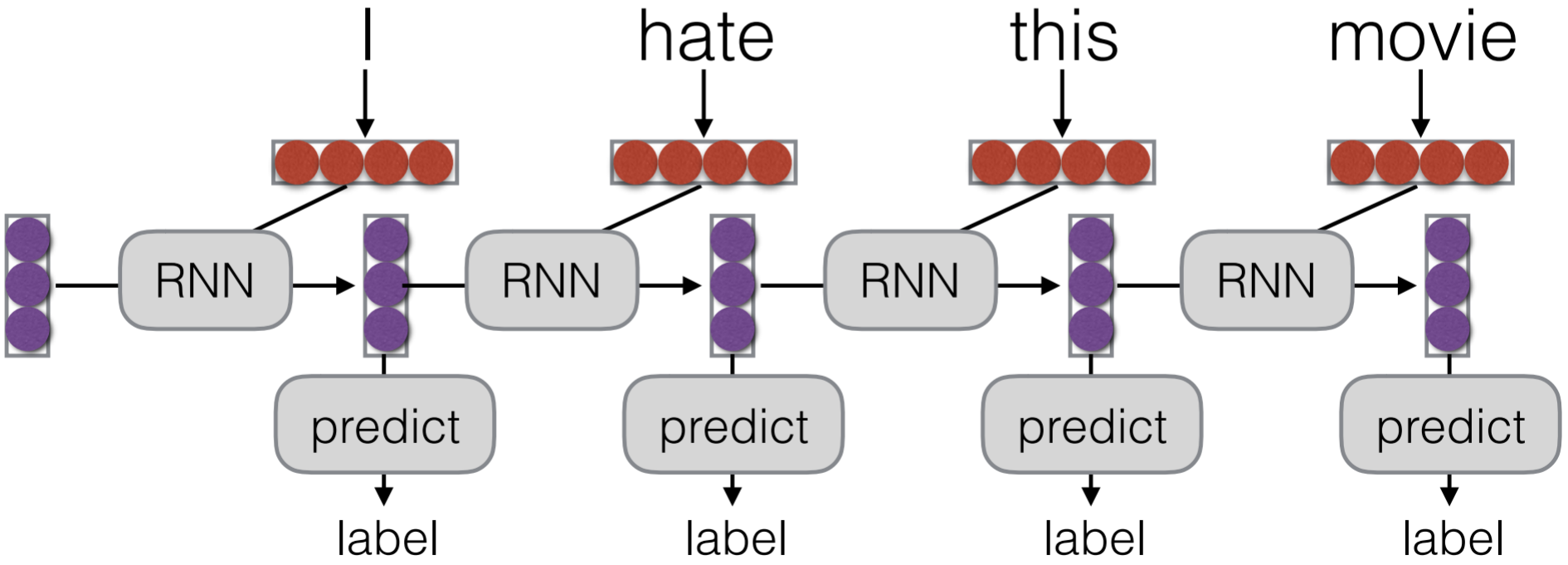



Recurrent NN

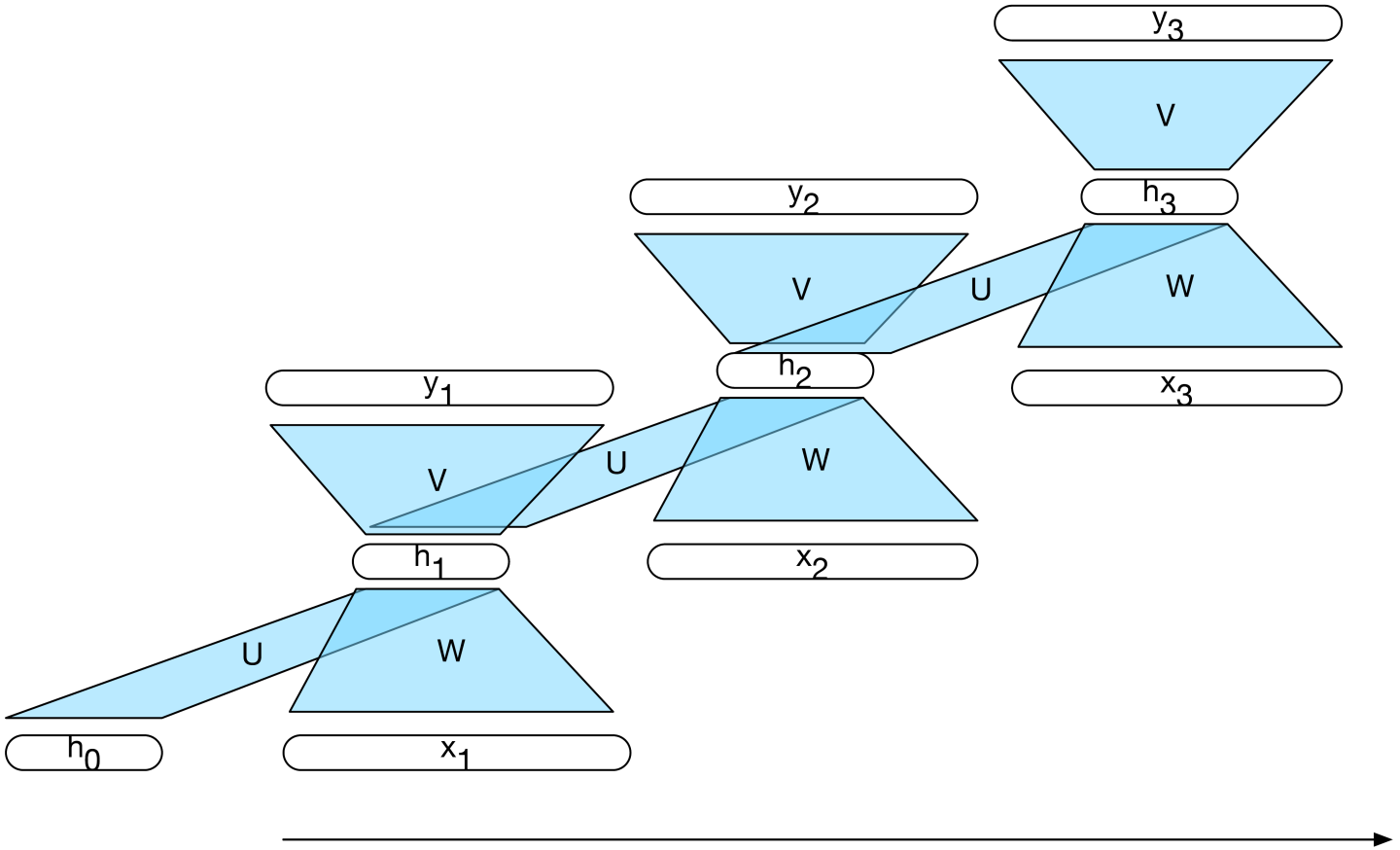


# Unrolling in Time

- What does processing a sequence look like?

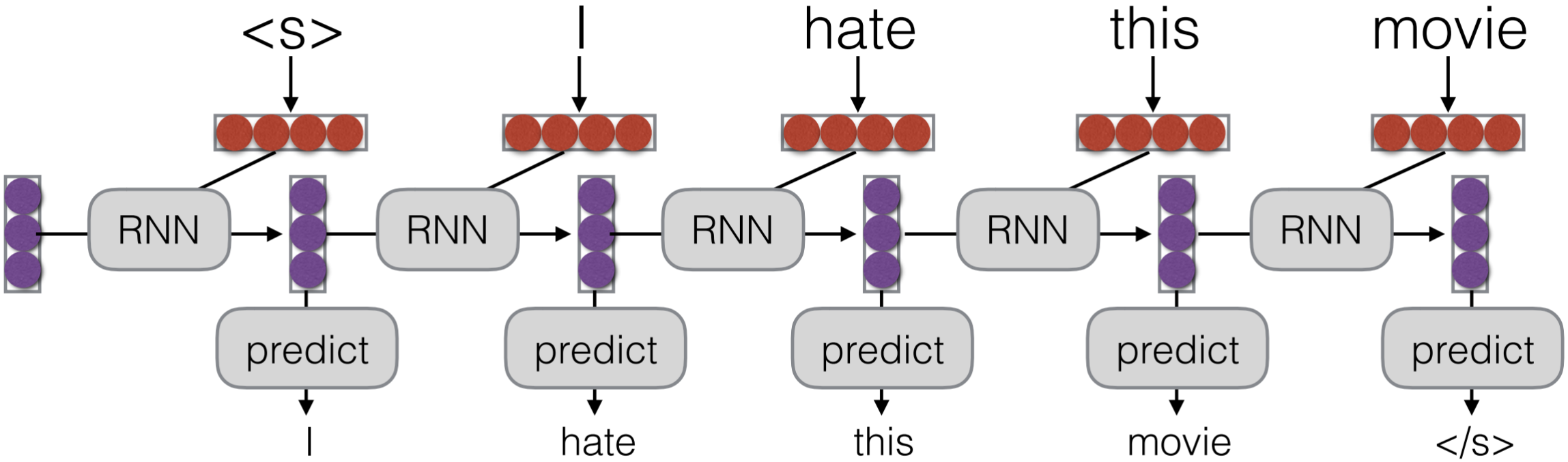


# Unrolled RNN illustrated




weights U, V, W are shared across all timesteps

# e.g. Language Modeling




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

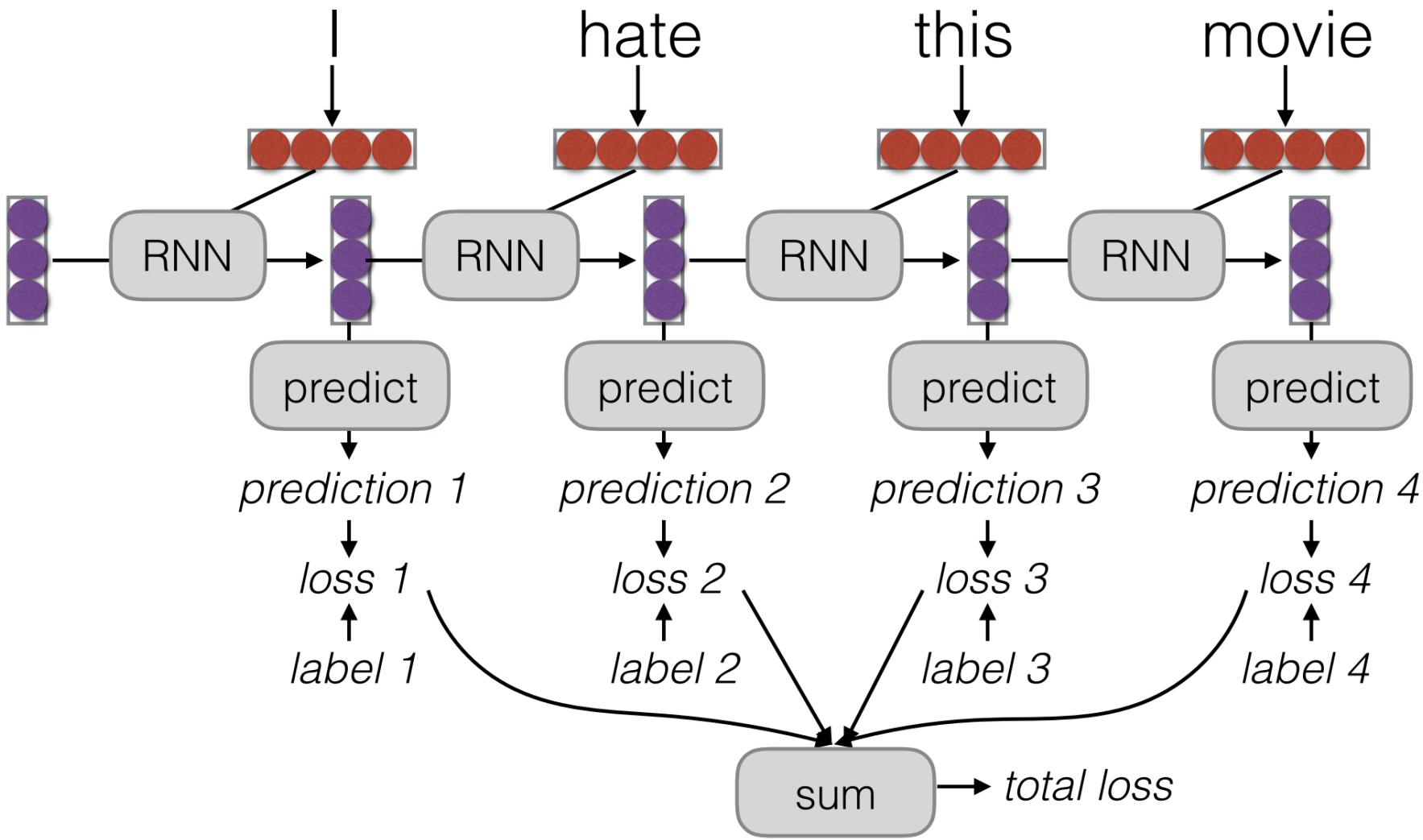
# Prediction/Inference with RNNs

**function** FORWARDRNN($x$, *network*) **returns** output sequence $y$

$h_0 \leftarrow 0$
**for** $i \leftarrow 1$ **to** LENGTH($x$) **do**
    $h_i \leftarrow g(U\ h_{i-1} + W\ x_i)$
    $y_i \leftarrow f(V\ \ h_i)$
**return** $y$

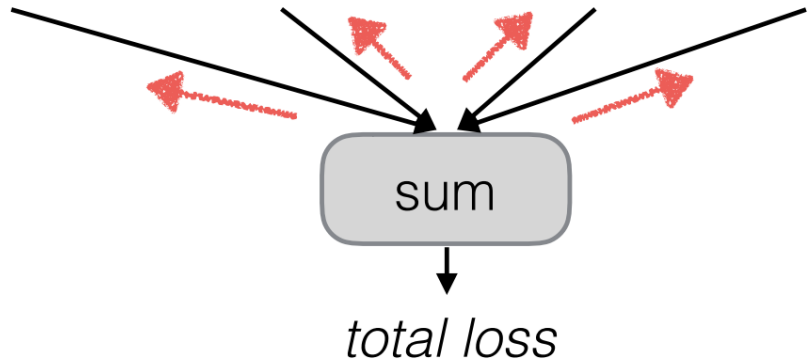
For language modeling, f = softmax function to provide normalized probability distribution over possible output classes

# Training RNNs



# RNN Training

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

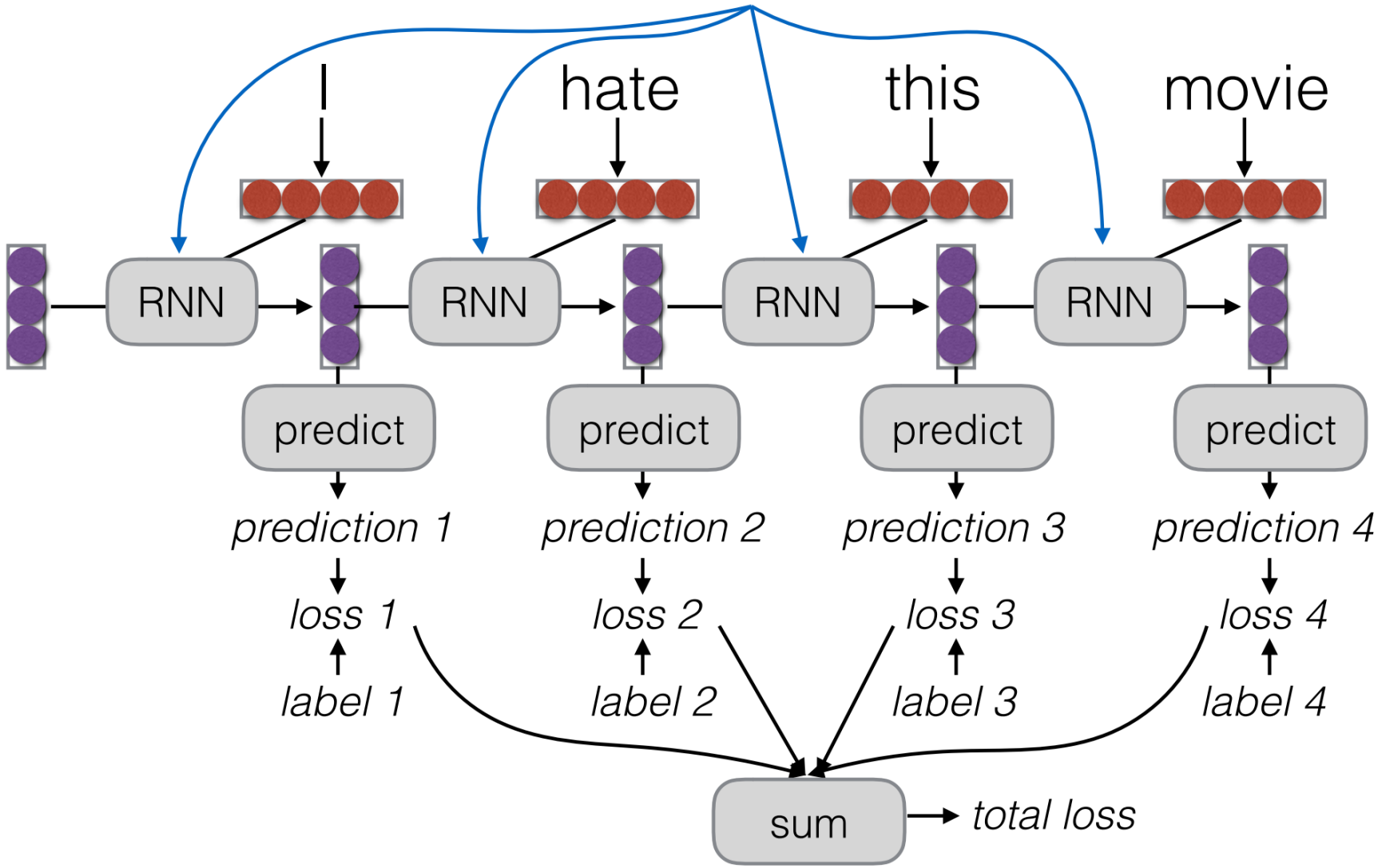



  - Parameters are tied across time, derivatives are aggregated across all time steps

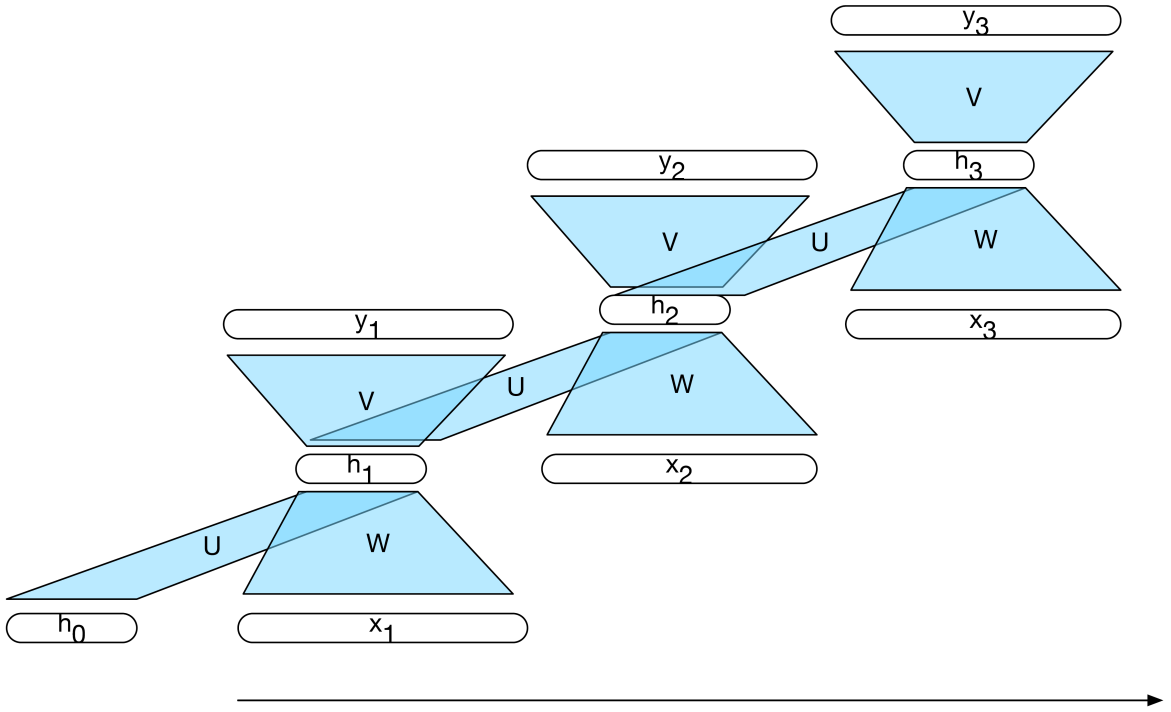
  - This is historically called “backpropagation through time” (BPTT)

# Parameter Tying

Parameters are shared! Derivatives are accumulated.

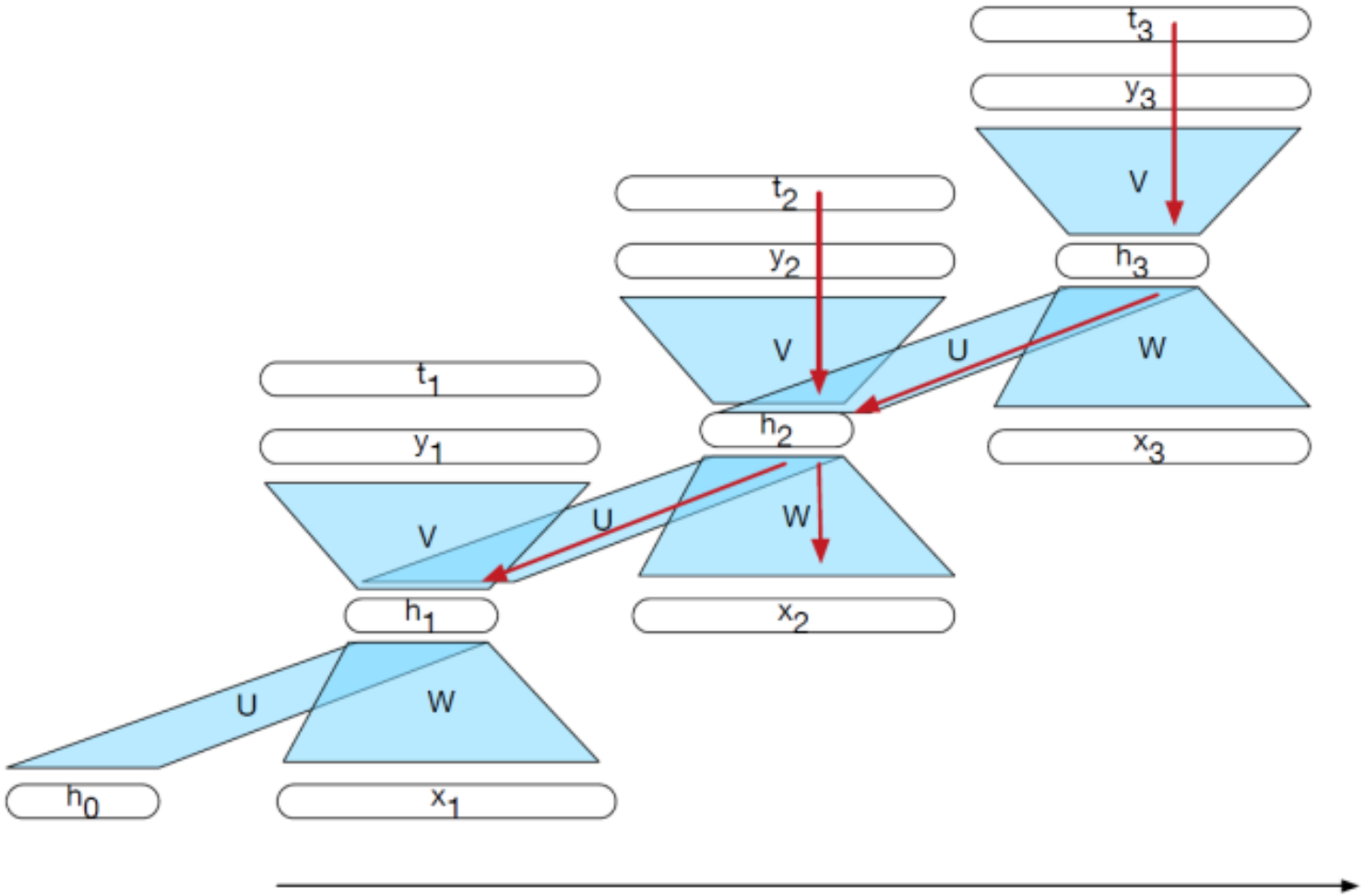


# Training RNNs with backpropagation




- Training goal: estimate parameter values for U, V, W
- Use same loss as for feedforward language models
- Given unrolled network, run forward and backpropagation algorithms as usual

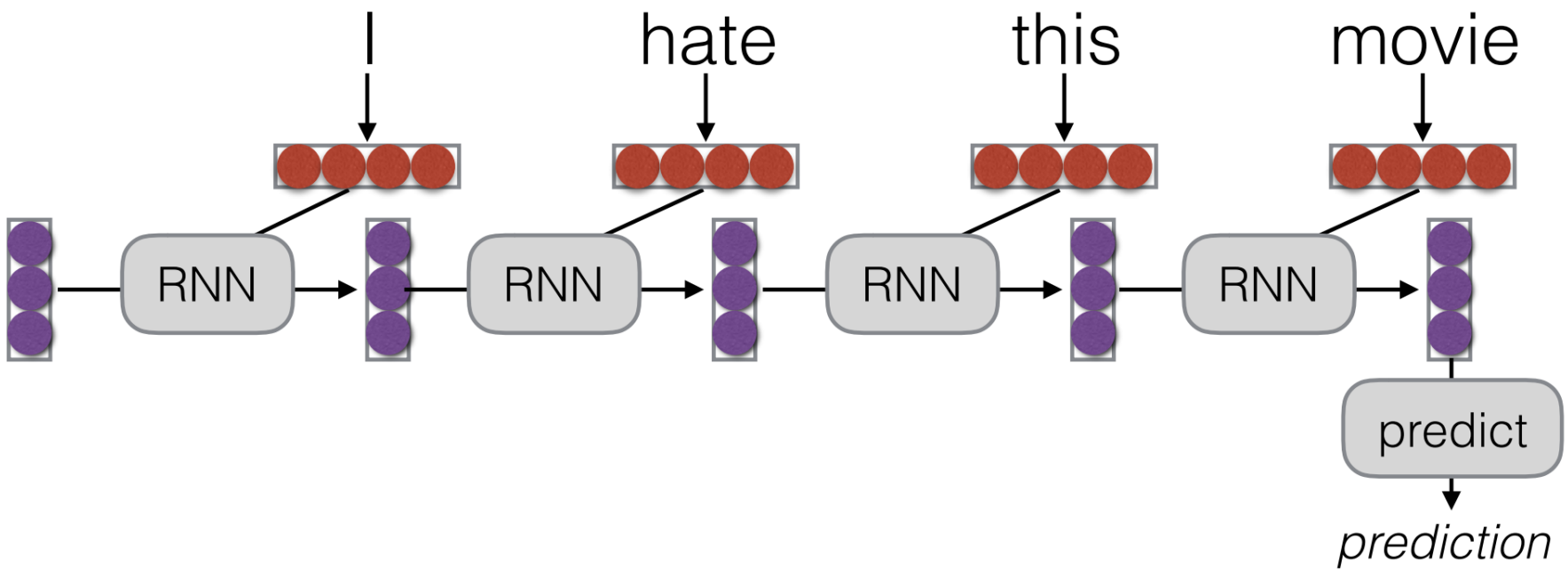
# Training RNNs with backpropagation



# What Can RNNs Do?

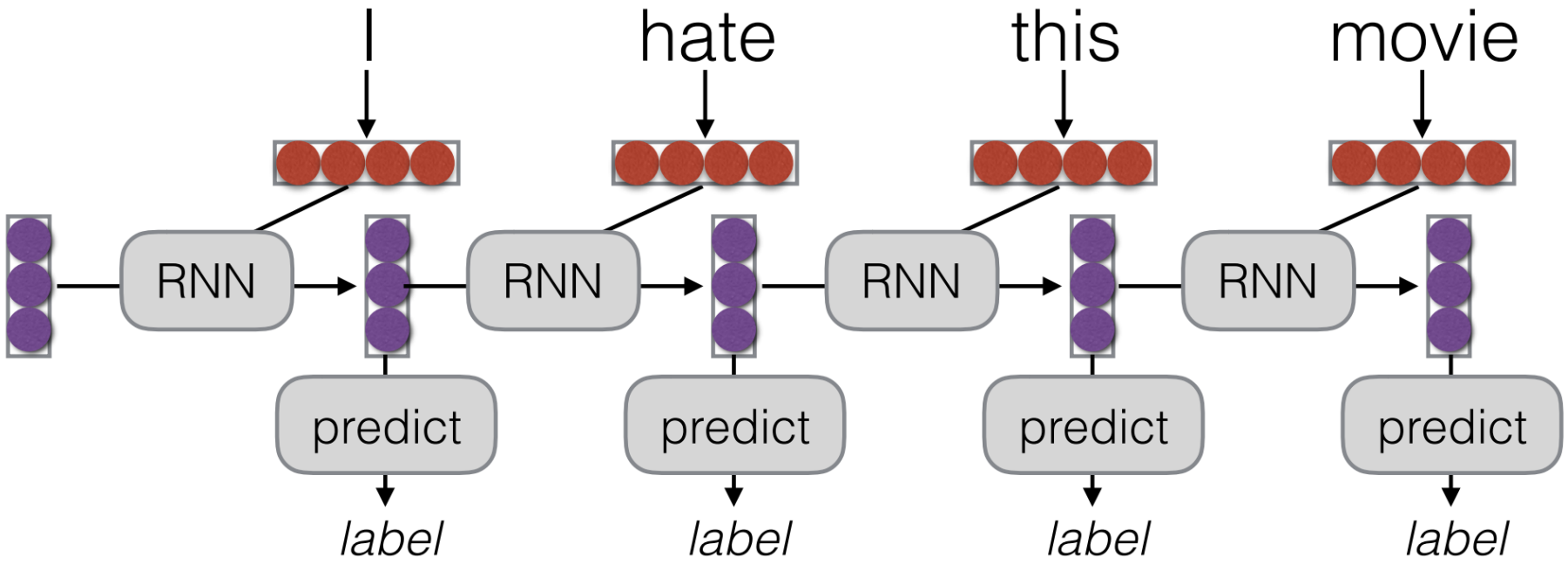
- Represent a sentence
  - Read whole sentence, make a prediction
- Represent a context within a sentence
  - Read context up until that point

# Representing Sentences




- Sentence classification
- Conditioned generation
- Retrieval

# Representing Contexts




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

# N-grams vs RNNs

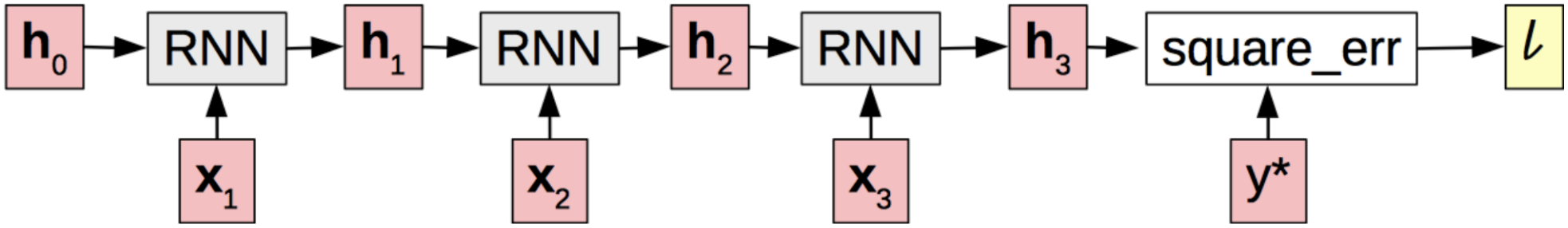
- The Unreasonable Effectiveness of Recurrent Neural Networks
  Andrej Karpathy
  http://karpathy.github.io/2015/05/21/rnn-effectiveness/

- The unreasonable effectiveness of Character-level Language Models (and why RNNs are still cool)
  Yoav Goldberg
  https://nbviewer.jupyter.org/gist/yoavg/d76121dfde2618422139

# Vanishing Gradient

- Gradients decrease as they get pushed back

$$\frac{dl}{d_{h_0}} = \text{tiny} \qquad \frac{dl}{d_{h_1}} = \text{small} \qquad \frac{dl}{d_{h_2}} = \text{med.} \qquad \frac{dl}{d_{h_3}} = \text{large}$$

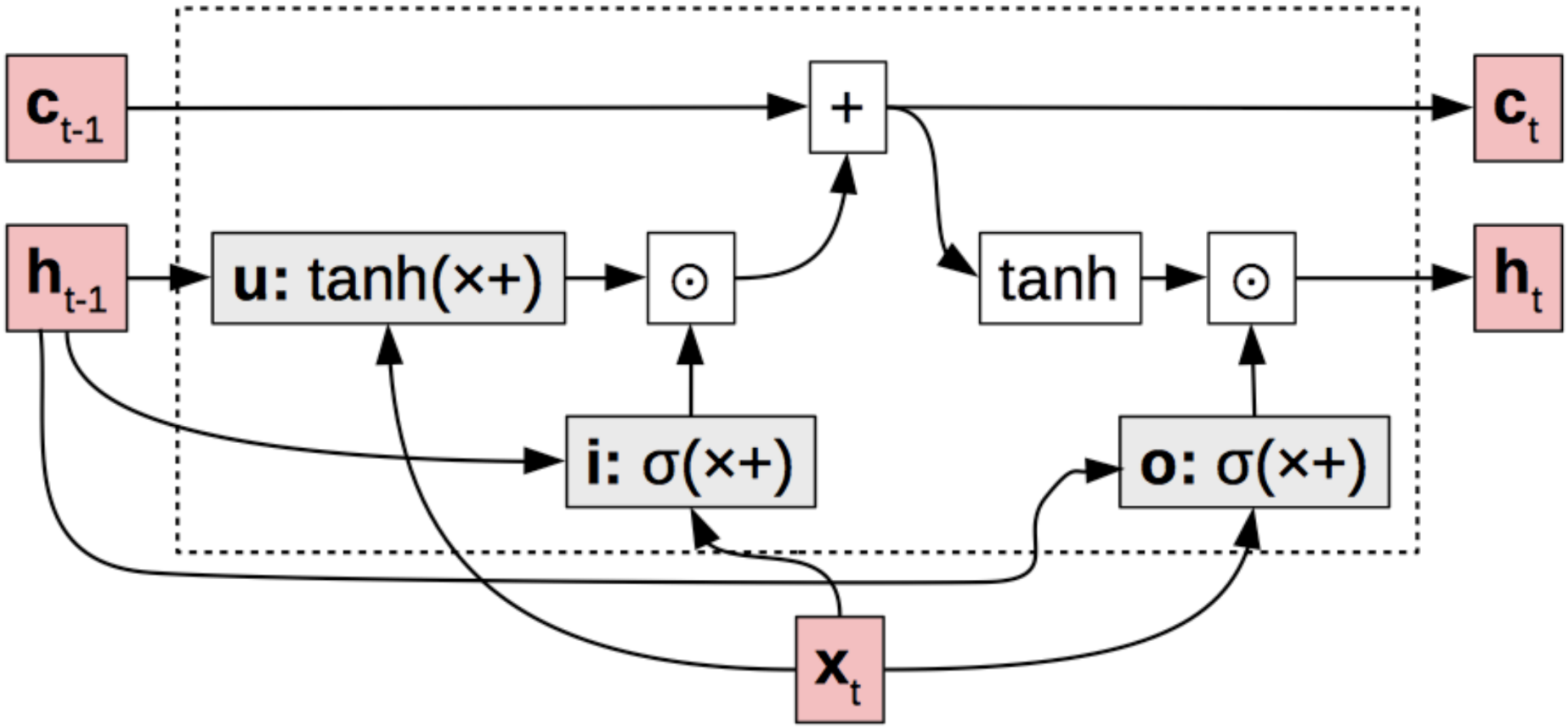



- Why? “Squashed” by non-linearities or small weights in matrices.

# 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?

# What can LSTMs Learn? (1)

## (Karpathy et al. 2015)

- Additive connections make single nodes surprisingly interpretable

Cell sensitive to position in line:

```
The sole importance of the crossing of the Berezina 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 ")":

```
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;
}
```

# Today

- Feed-forward neural language models
  - Turn language modeling into an explicit classification problem
  - (Yet) Another way to get word embeddings
    - Train from scratch
    - Use offline
    - Fine-tune

- Recurrent neural networks / language models
  - Unlimited history (in principle)
  - Shared parameters
  - Vanishing gradient -> LSTMs

# PROJECT TIME!