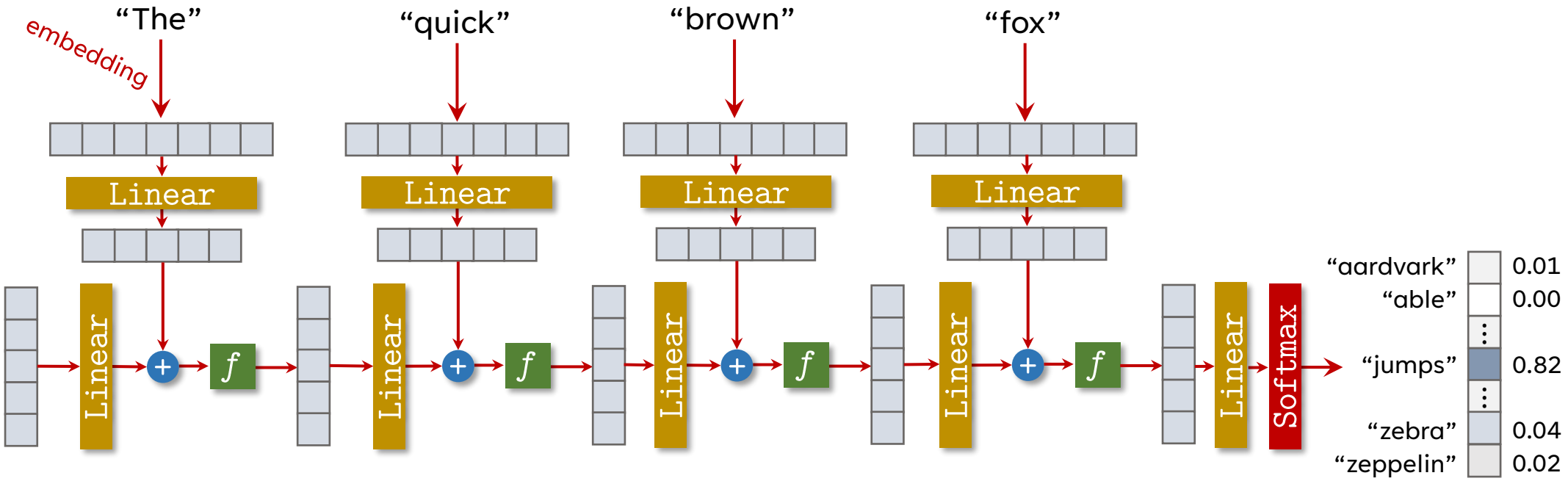
CS 577: NATURAL LANGUAGE PROCESSING

Abulhair Saparov

Lecture 5: LSTMs and GRUs

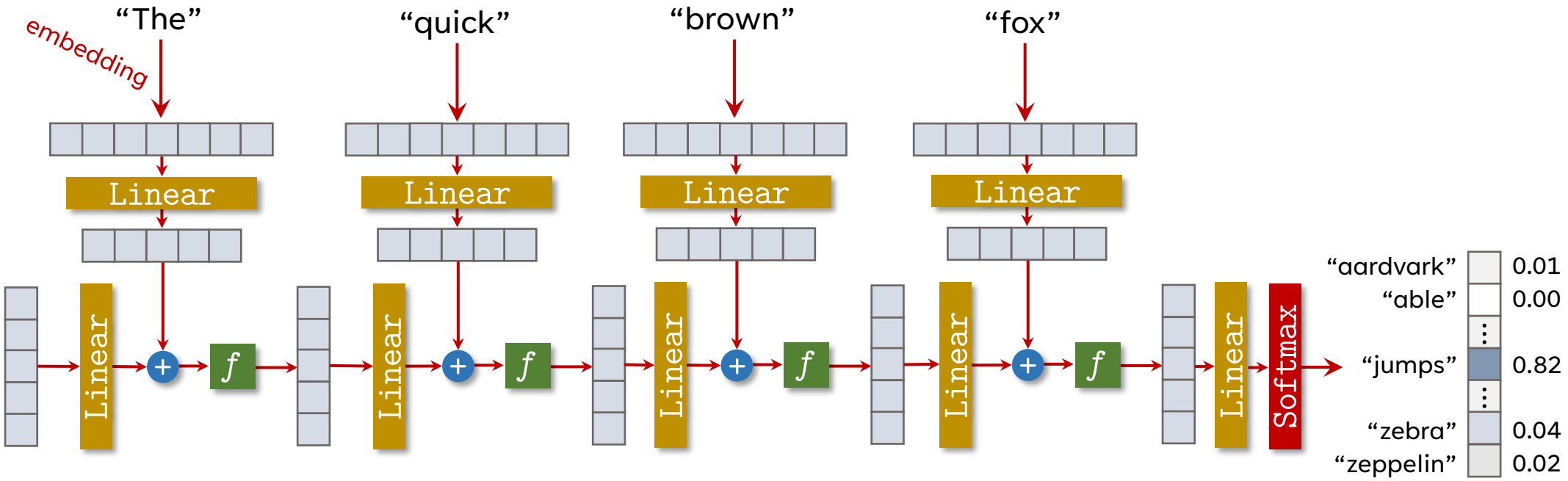
RECAP FROM PREVIOUS LECTURE

- We discussed the recursive neural network (RNN) architecture.



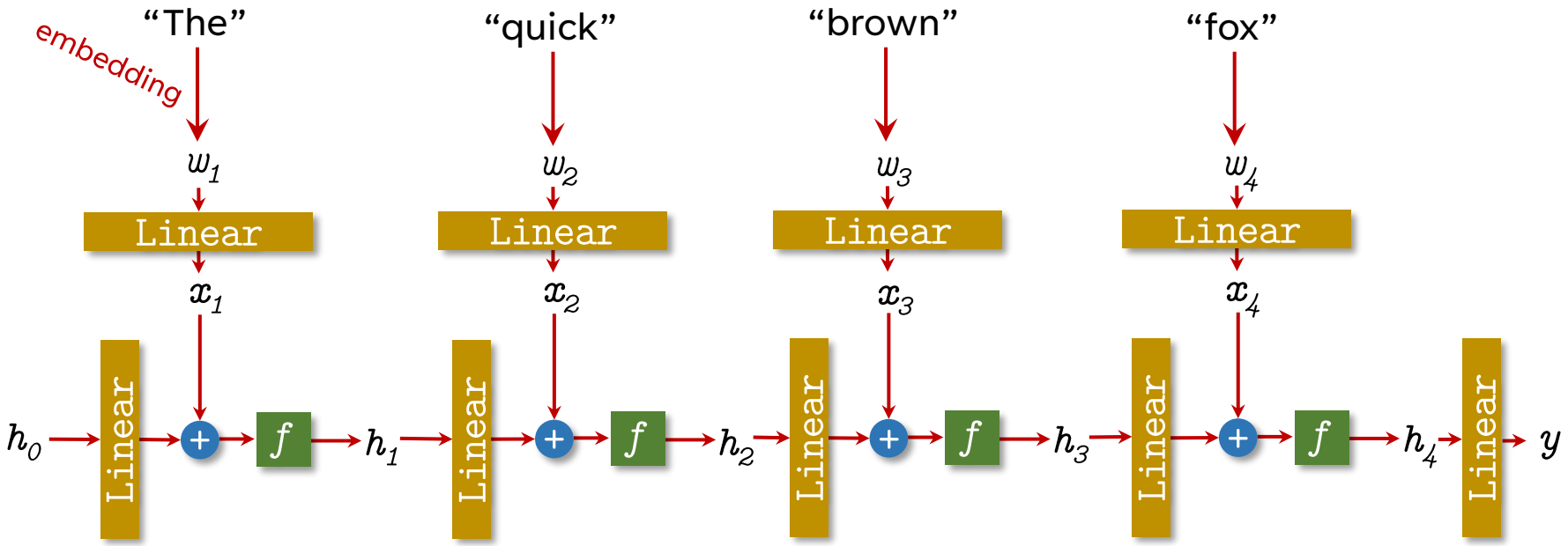
RECAP FROM PREVIOUS LECTURE

- Models the sequential (word-by-word) processing nature of human language processing.



RECAP FROM PREVIOUS LECTURE

- Models the sequential (word-by-word) processing nature of human language processing.



VANISHING/EXPLODING GRADIENTS

- Suppose we have a long RNN (lots of tokens).

$$y = g_n(g_{n-1}(\ldots g_2(g_1(x_1, W_1))\ldots))$$

- Each g_i is an RNN "unit", written more simply.
- x_1 is the first word, and W_1 is the weight matrix in the linear layer after x_1.
- What is the gradient with respect to W_1?

$$\nabla_{W_1} y = g_n'(\ldots) \cdot \nabla_{W_1} g_{n-1}(\ldots)$$

$$= g_n'(\ldots) \cdot g_{n-1}'(\ldots) \cdot \nabla_{W_1} g_{n-2}(\ldots)$$

$$= \ldots = g_n'(\ldots) \cdot g_{n-1}'(\ldots) \cdot \ldots \cdot g_2'(\ldots) \cdot g_1'(\ldots) \cdot x_1^T$$

VANISHING/EXPLODING GRADIENTS

- Note that this is a product containing many terms.
- If the terms are > 1, their product will grow exponentially in n.
- If the terms are < 1, their product will shrink to 0 exponentially in n.
- This is called the exploding or vanishing gradient problem.
- This is also an issue for very deep networks (containing many layers).

$$\nabla_{W_1} y = g_n'(\ldots) \cdot \nabla_{W_1} g_{n-1}(\ldots)$$
$$= g_n'(\ldots) \cdot g_{n-1}'(\ldots) \cdot \nabla_{W_1} g_{n-2}(\ldots)$$
$$= \ldots = g_n'(\ldots) \cdot g_{n-1}'(\ldots) \cdot \ldots \cdot g_2'(\ldots) \cdot g_1'(\ldots) \cdot x_1^T$$

LONG-TERM DEPENDENCIES

- Suppose we have some NLP task where the input is a document or string of words.
- If the expected output for this task requires information about words that occur far from the end of the input,

  This is a long-term dependency (or a long-range dependency).
- Why are they important in natural language?

LONG-TERM DEPENDENCIES

- Consider the following sentences:
  - “The cat, which was very hungry, chased the mouse.”
  - “The children sitting under the tree know the farmers.”
- Suppose we ask questions such as:
  - Who chased the mouse?
  - Who knows the farmers?

LONG-TERM DEPENDENCIES

- Consider the following sentences:
  - `“Without her contributions would be impossible.”`
    - What is the subject of this sentence?
    - What is the main verb?
  - `“The old man the boat.”`
  - `“I convinced her children are noisy.”`
    - (I convinced her that children are noisy)
- These are called garden path sentences.

LONG-TERM DEPENDENCIES

- Long-term dependencies appear in almost every NLP task.
- Coreference resolution:
  - “I couldn’t fit the trophy in the bag because it was too big.”
- Translation:
  - Consider translating from an SVO language into an VSO language.
  - E.g., English into Arabic.
  - “The cat, which was very hungry, chased the mouse”
  - [“chased”] [“the cat, which was very hungry”] [“the mouse”]

LONG-TERM DEPENDENCIES

- It is possible for an RNN to compute long-term dependencies.
- For example, we can construct an RNN to memorize the input sequence:
  - Input: word embedding w_t and previous state h_{t-1}
  - Suppose hidden dimension is Nd
    - where N is the sequence length, and d the embedding dimension.
  - The linear layer on the word simply takes the embedding of w_t and appends many zeros to the end of it.
    - The first d elements of the output are the embedding of w_t, and the last $(N-1)d$ elements are zero.
  - The linear layer on the hidden state rotates the dimensions of h_{t-1} by d positions.

LONG-TERM DEPENDENCIES

- It is possible for an RNN to compute long-term dependencies.
- For example, we can construct an RNN to memorize the input sequence:
  - Input: word embedding w_t and previous state h_{t-1}
  - This construction is simply concatenating each word embedding into a long hidden state vector (which is initially zero).
  - Therefore, the hidden state is able to keep track of long-term dependencies.
    - It has memorized the entire input!

EXPRESSIVENESS VS LEARNABILITY

- It is possible for an RNN to compute long-term dependencies.
- In fact, RNNs are *Turing-complete* (Siegelmann 1995).
- A model is Turing-complete if for any Turing machine M (i.e., algorithm),

  There exists an instance of the model that implements M.

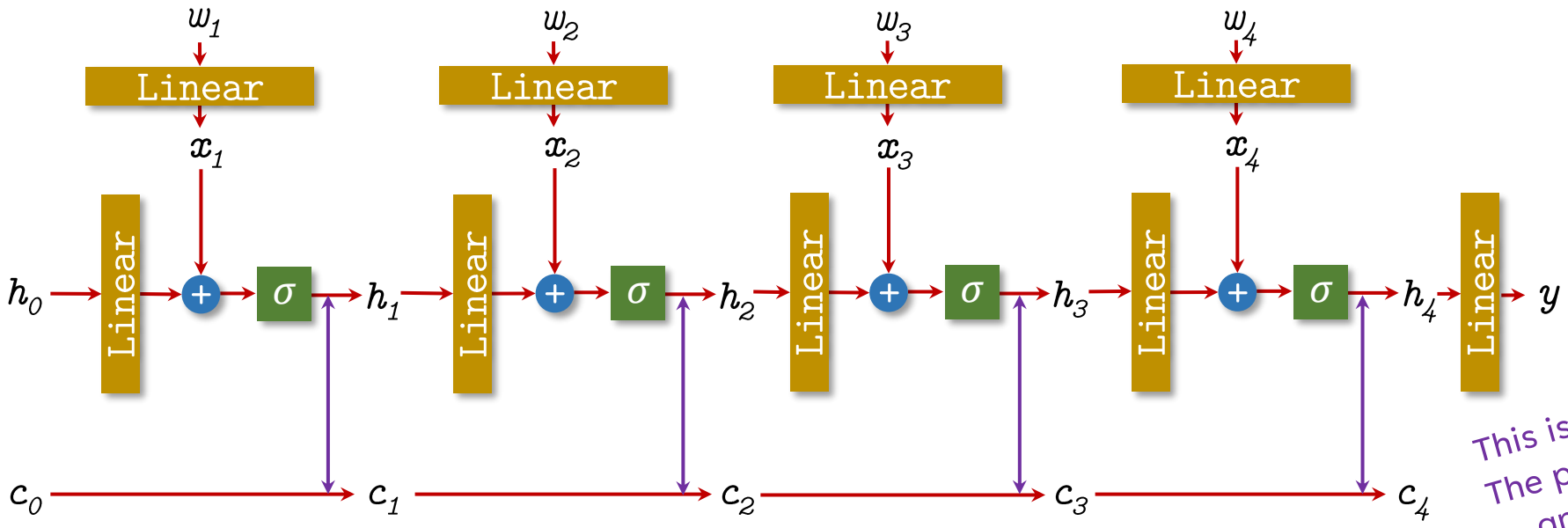
  I.e., there is a setting of the model weights such that it implements M.

EXPRESSIVENESS VS LEARNABILITY

- The problem is that this network is *difficult to learn*.
  - Due to the vanishing/exploding gradients problem.
- But this is a common problem with highly expressive models.
  - E.g., MLPs, RNNs, transformers
  - These models are able to express a vast number of algorithms.
  - But the question of whether they can learn all such algorithms from data is entirely separate.
  - Oftentimes, there are algorithms that these models can express, but have difficulty learning.

LONG SHORT-TERM MEMORY NETWORKS

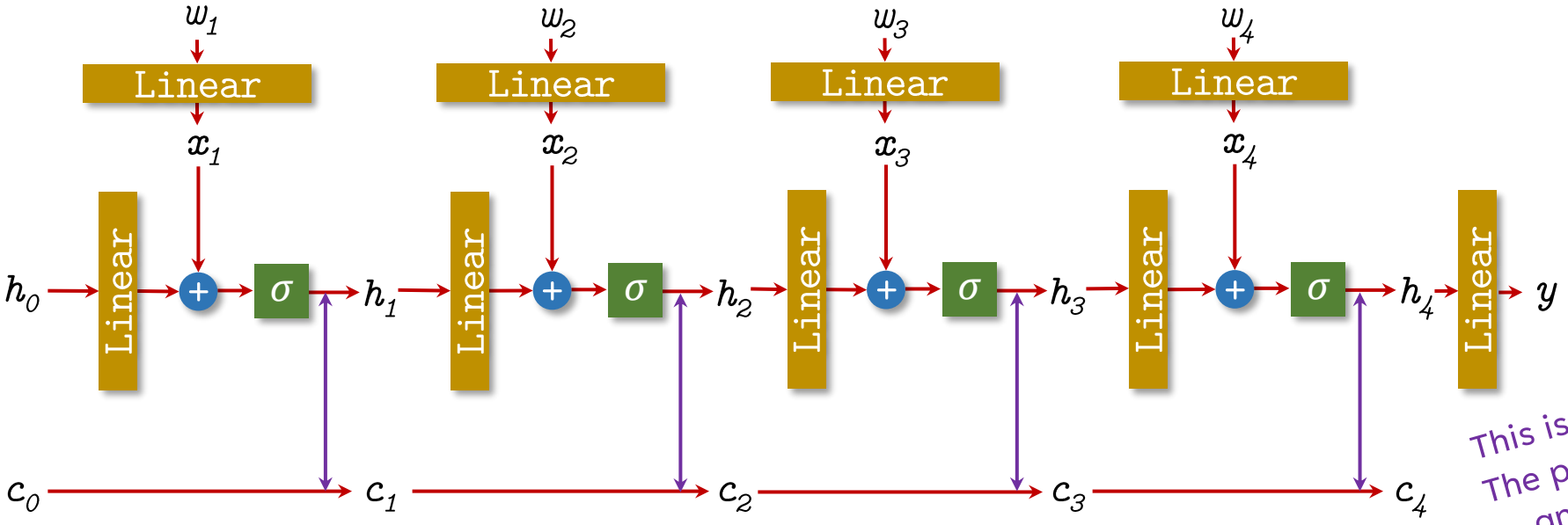
- To address this problem, LSTMs were developed (Hochreiter and Schmidhuber 1997).



This is just a sketch!
The precise interaction between c_i and h_i will be explained later.

LONG SHORT-TERM MEMORY NETWORKS

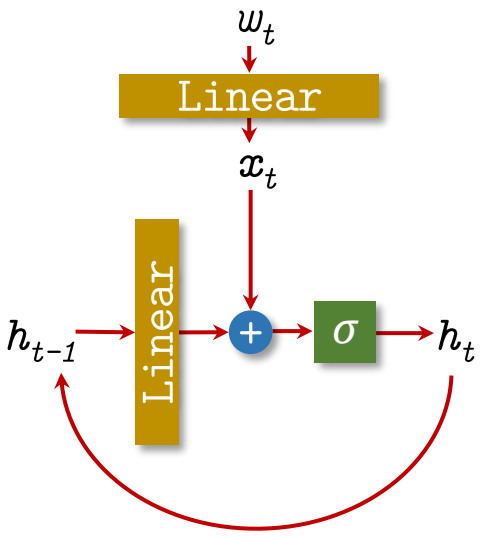
- Key idea is that the updates to the c_i stream are additive.
- So gradients of c_i do not get very large or very small with increasing n.
- How are c_i and h_i computed?



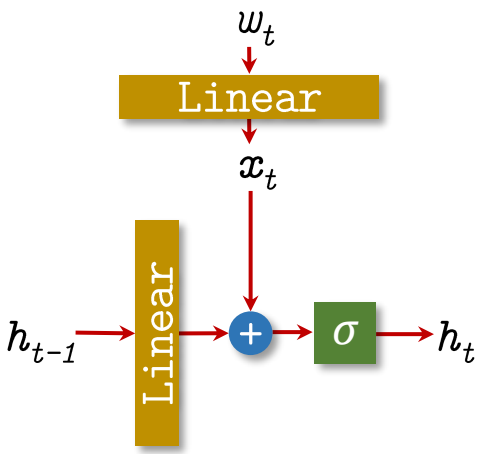
This is just a sketch!
The precise interaction between c_i and h_i will be explained later.

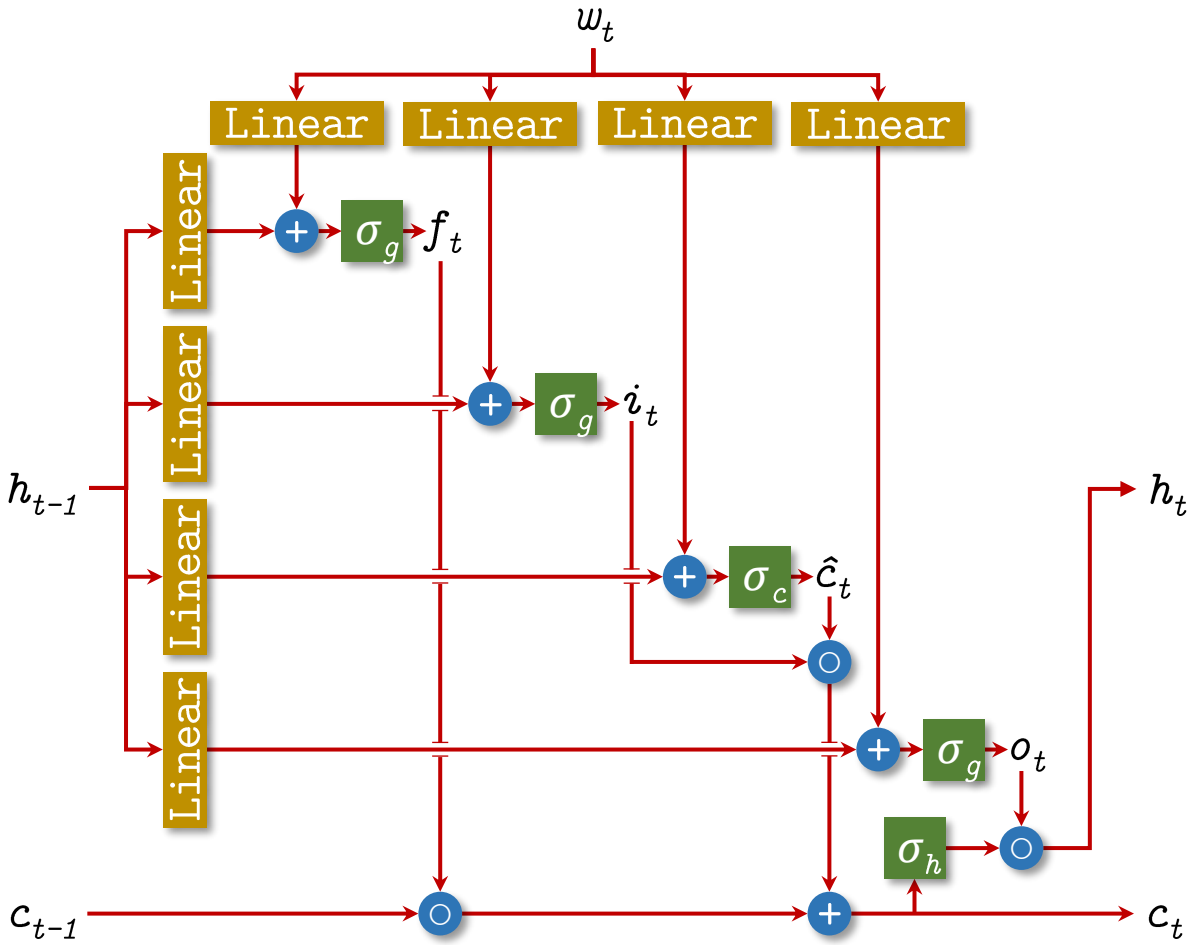
LONG SHORT-TERM MEMORY NETWORKS

- Key idea is that the updates to the c_i stream are additive.
- So gradients of c_i do not get very large or very small with increasing n.
- How are c_i and h_i computed?



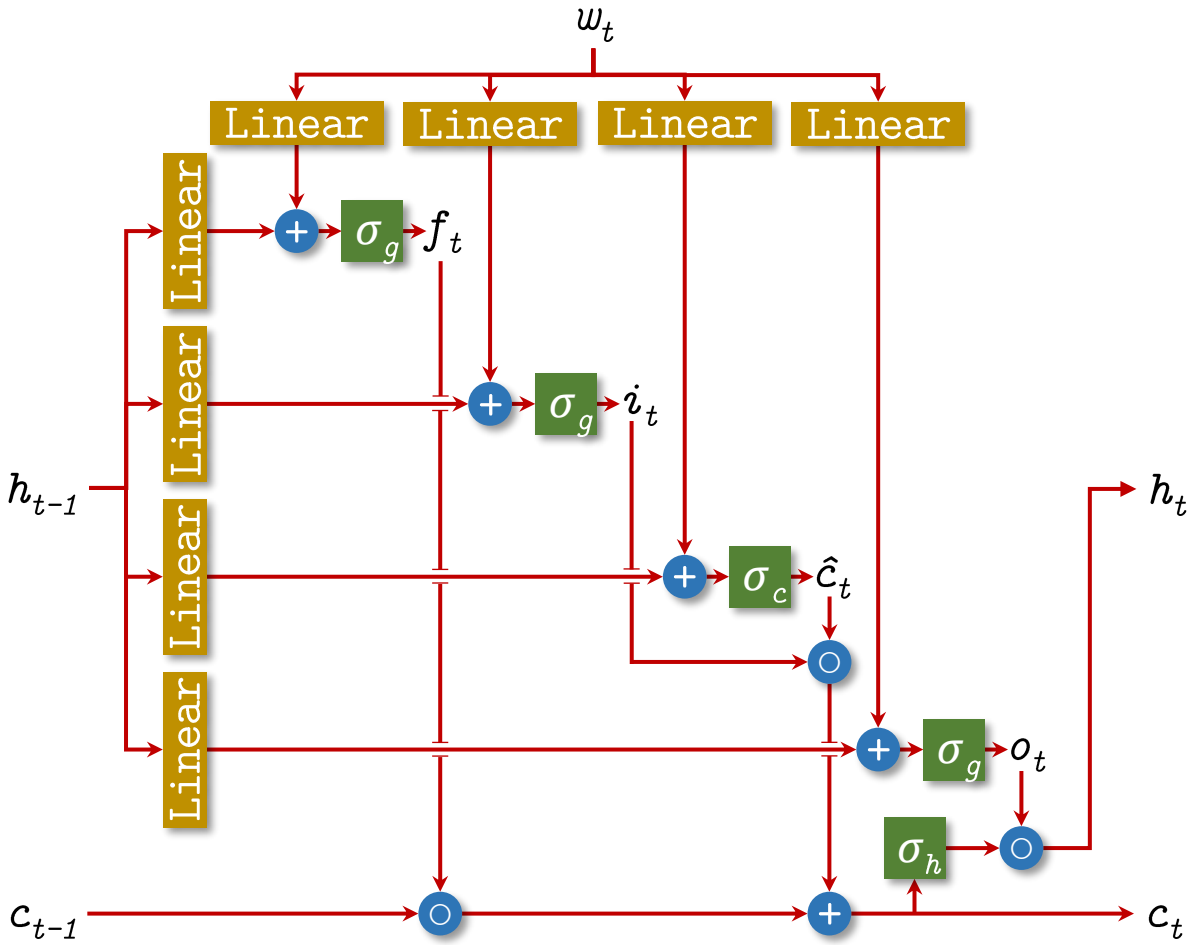
LONG SHORT-TERM MEMORY NETWORKS





LONG SHORT-TERM MEMORY NETWORKS

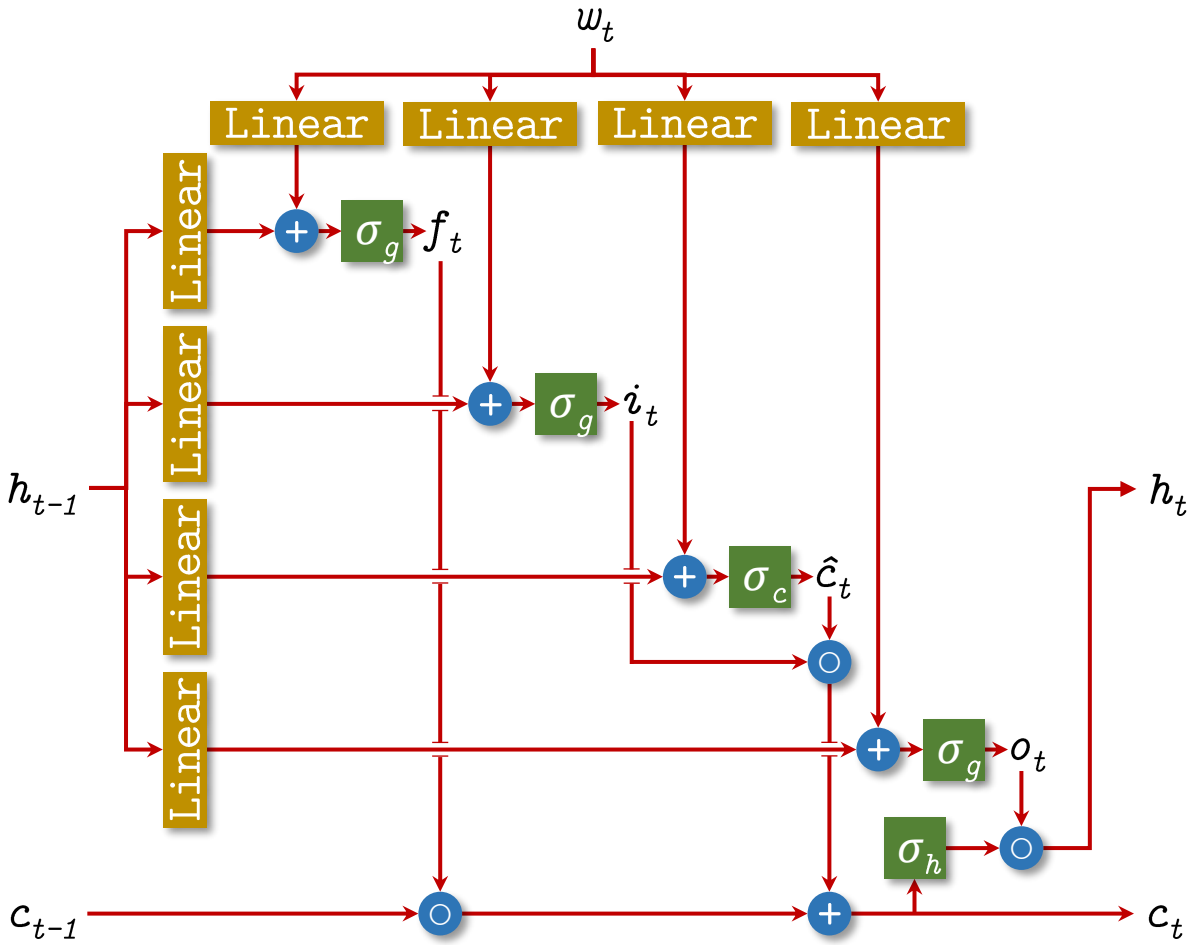
- Don't worry: You don't need to memorize this circuit diagram.
- It's easy enough to lookup.
- But the key idea is important:
- Updates to c_t are additive.
- Gradients with respect to c_t won't vanish or explode.



LONG SHORT-TERM MEMORY NETWORKS

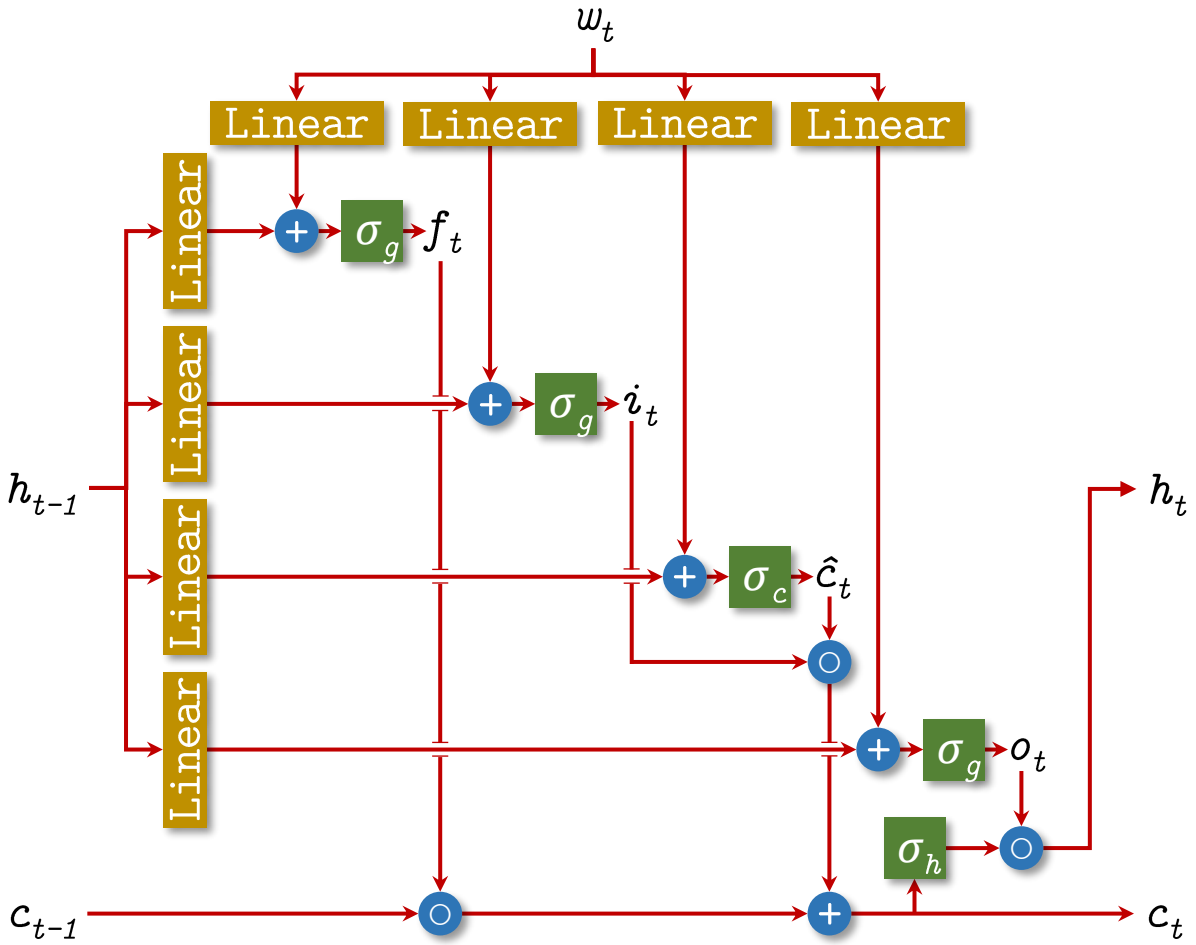
$$f_t = \sigma_g(W_f w_t + U_f h_{t-1} + b_f)$$
$$i_t = \sigma_g(W_i w_t + U_i h_{t-1} + b_i)$$
$$o_t = \sigma_g(W_o w_t + U_o h_{t-1} + b_o)$$
$$\hat{c}_t = \sigma_c(W_c w_t + U_c h_{t-1} + b_c)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \hat{c}_t$$
$$h_t = o_t \circ \sigma_h(c_t)$$

Traditionally, σ_g is a sigmoid, σ_c and σ_h are *tanh*.

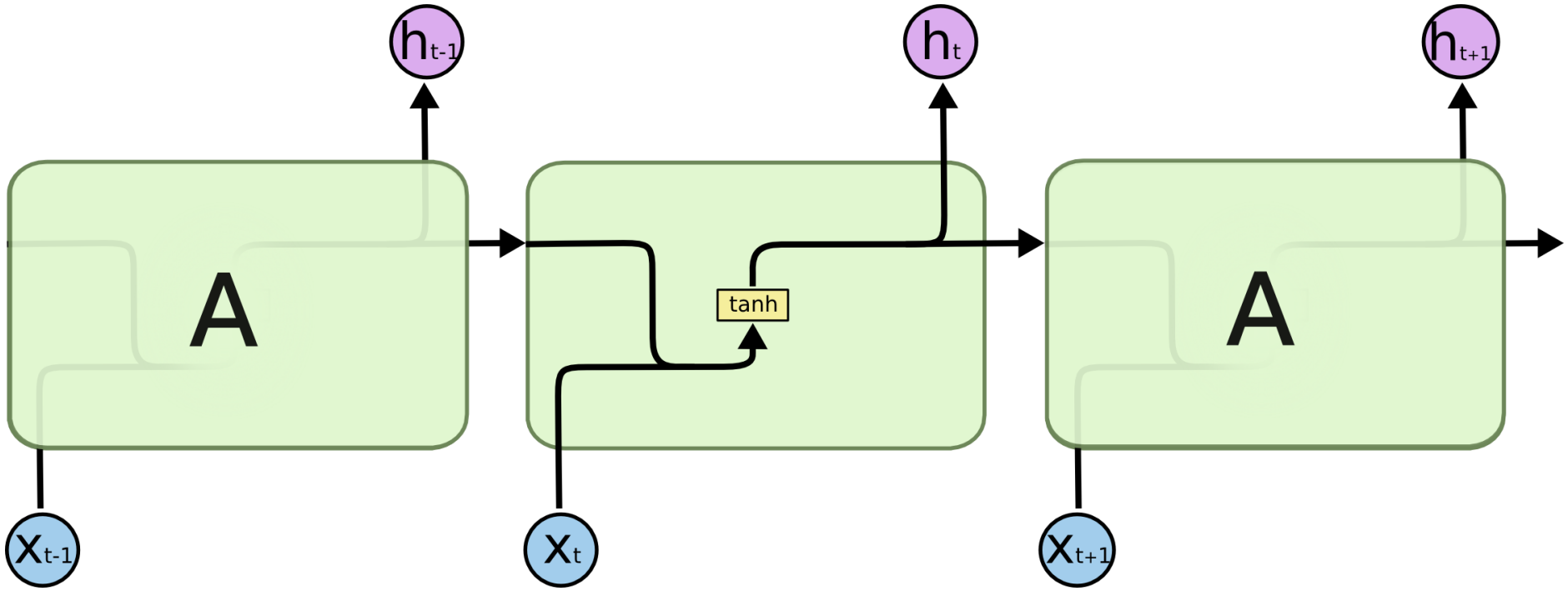


LONG SHORT-TERM MEMORY NETWORKS

- Subcircuits in the LSTM are referred to as “gates”.
- For example, the subcircuit involving f_t is called the forget gate.
- The subcircuit involving i_t is the input gate.
- The subcircuit involving o_t is the output gate.

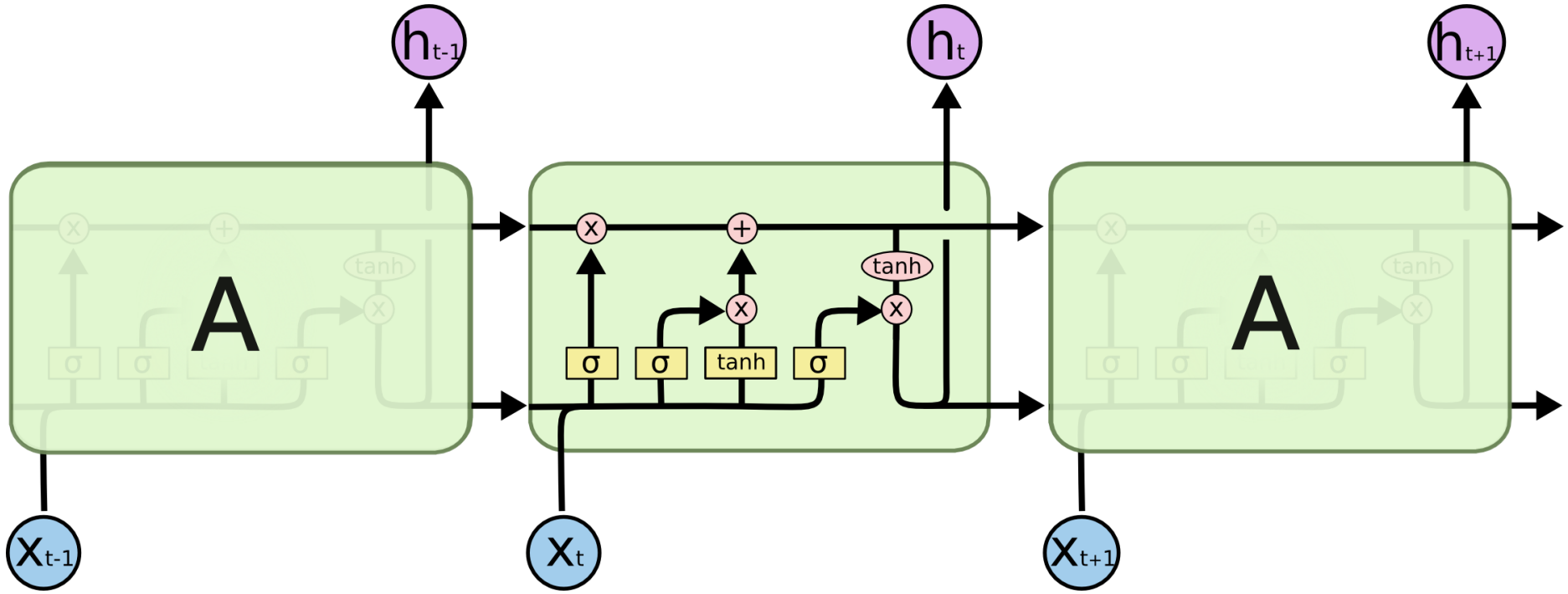


LONG SHORT-TERM MEMORY NETWORKS



[Christopher Olah, Understanding LSTM Networks]

LONG SHORT-TERM MEMORY NETWORKS



[Christopher Olah, Understanding LSTM Networks]

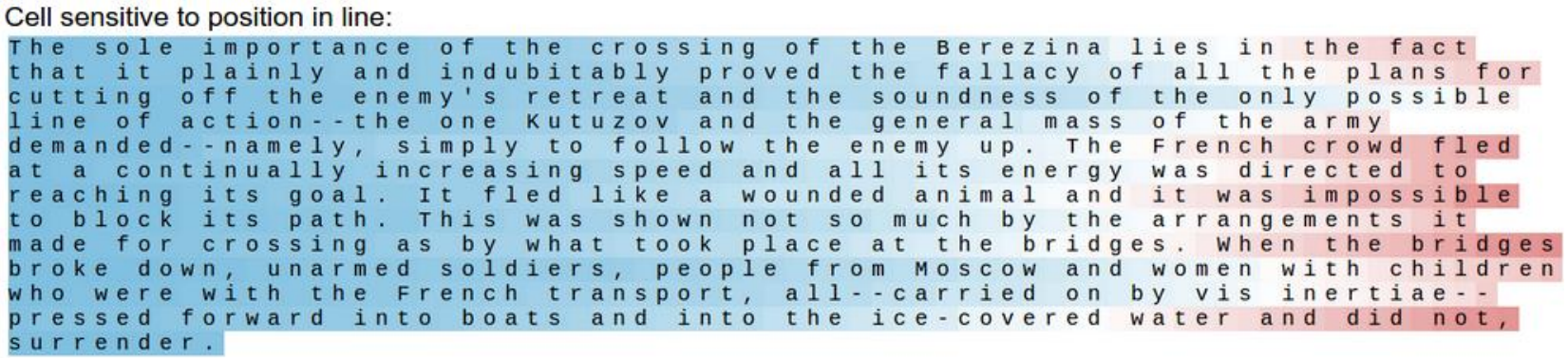
INTERPRETATION OF TRAINED LSTMS

- Train character-level LSTM on the language modeling task.
  - I.e., predict the next character given a sequence of previous characters.
  - Train on two datasets: War and Peace, and Linux kernel source code.
- We can inspect the activations in the cell state vector c_t (i.e., activations of individual neurons).

[Andrej Karpathy, The Unreasonable Effectiveness of Recurrent Neural Networks; slide text from Greg Durrett]

INTERPRETATION OF TRAINED LSTMS

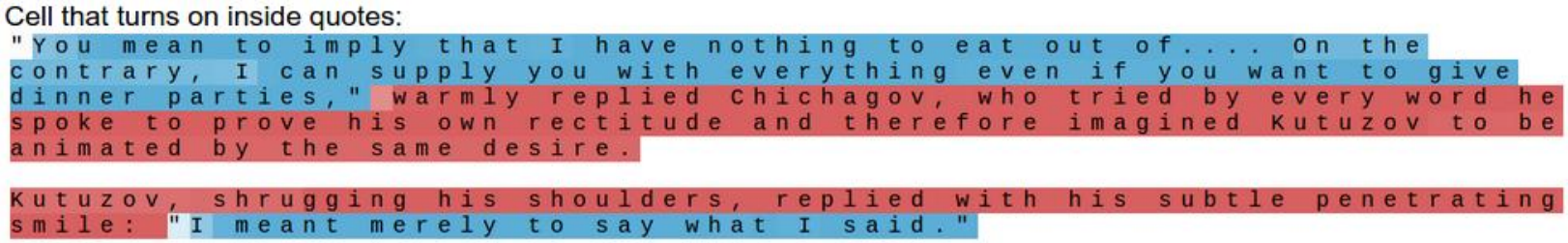
- One neuron seems to predict the length of the current line.
- This is important when, e.g., you need to know when to predict \n.
- Takeaway: LSTMs can model length.



[Andrej Karpathy, The Unreasonable Effectiveness of Recurrent Neural Networks; slide text from Greg Durrett]

INTERPRETATION OF TRAINED LSTMS

- One neuron seems to keep track of when we're inside a quotation.
- This is important when, e.g., you need to know when to predict ", or when to predict sentences in first or second person.
- Seems to work even with very long quotations.



[Andrej Karpathy, The Unreasonable Effectiveness of Recurrent Neural Networks; slide text from Greg Durrett]

INTERPRETATION OF TRAINED LSTMS

- One neuron seems to keep track of when we're inside an if condition.
- This is useful to remember to close the if statement, and to predict comparisons rather than assignments.

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;
}
```

[Andrej Karpathy, The Unreasonable Effectiveness of Recurrent Neural Networks; slide text from Greg Durrett]

INTERPRETATION OF TRAINED LSTMS

- One neuron seems to keep track of when we're inside a comment or a string.
- Useful to know to predict regular text rather than code.

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;
}
```

[Andrej Karpathy, The Unreasonable Effectiveness of Recurrent Neural Networks; slide text from Greg Durrett]

INTERPRETATION OF TRAINED LSTMS

- One neuron seems to keep track of the depth of the code.
- Useful to know how much to indent, or how many closing braces we need.

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;
}
```

[Andrej Karpathy, The Unreasonable Effectiveness of Recurrent Neural Networks; slide text from Greg Durrett]

INTERPRETATION OF TRAINED LSTMS

- One neuron seems to predict the ends of some statements and expressions (but not all), and not the ends of comments.
- This neuron is more difficult to interpret.

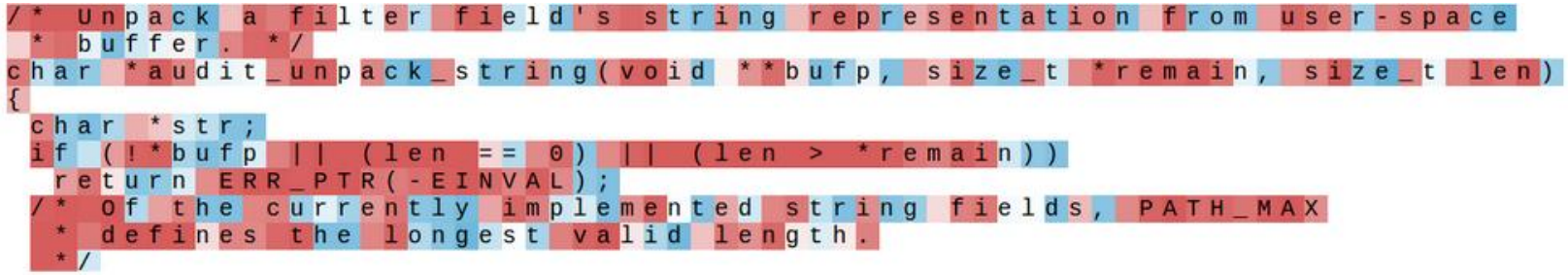
Cell that might be helpful in predicting a new line. Note that it only turns on for some ")":

```
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;
}
```

[Andrej Karpathy, The Unreasonable Effectiveness of Recurrent Neural Networks; slide text from Greg Durrett]

INTERPRETATION OF TRAINED LSTMS

- But *most* neurons are not interpretable.
- There is no reason for models to be easily interpretable after training,
- Unless we specifically design the neural network otherwise.



```
/* 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.
   */
```

[Andrej Karpathy, The Unreasonable Effectiveness of Recurrent Neural Networks; slide text from Greg Durrett]

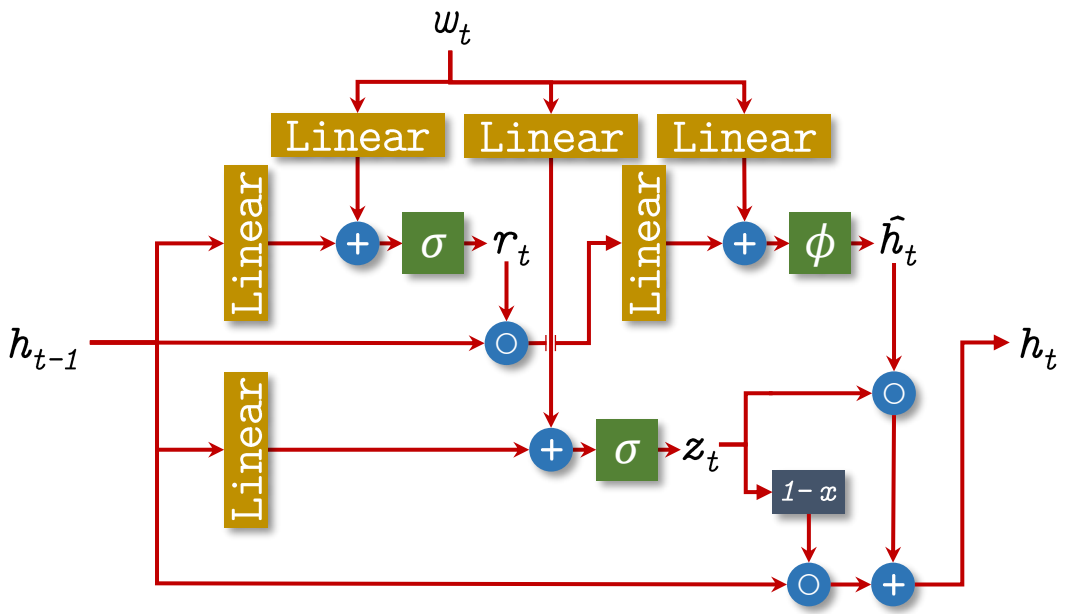
MECHANISTIC INTERPRETABILITY

- Mechanistic interpretability is the study of the internal computation of neural networks.
  - What algorithm has the network learned to solve the task?
  - "Reverse-engineering" the algorithm/mechanism learned by the network.
- Understanding neuron activations is akin to interpreting "intermediate variables" in the network's computation.
- But more work is needed to determine how these "intermediate variables" are related to each other.
  - How are they computed from the input/other intermediate variables?
  - How are they used to compute the output?

GATED RECURRENT UNITS

- A more recent variant of LSTMs is called the gated recurrent unit (GRU; Cho et al 2014).

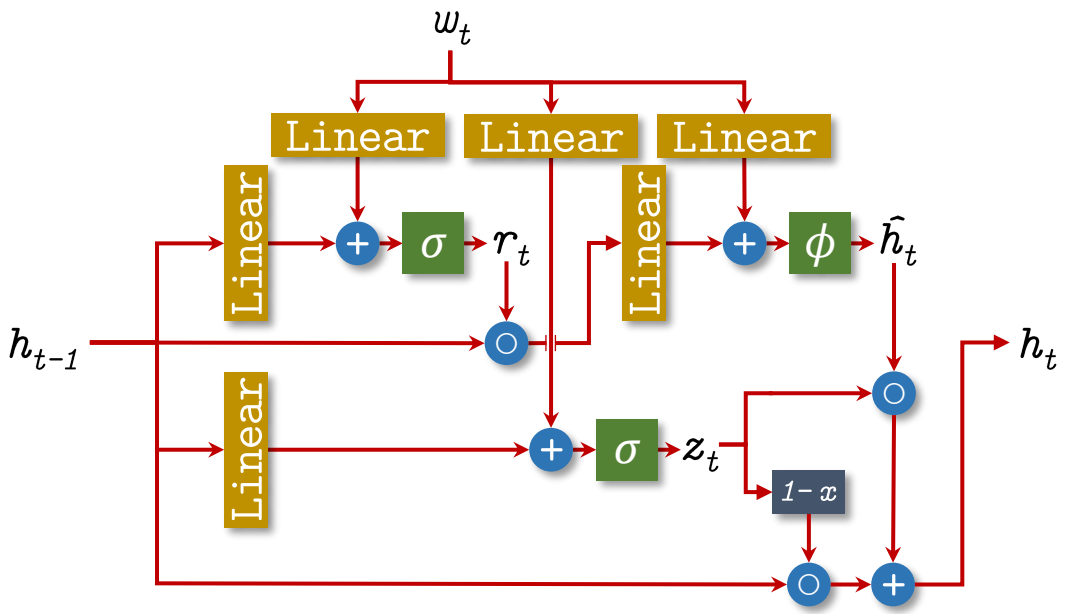
- Simpler than the LSTM:
- Instead of having a separate "cell" state, perform linear updates to the hidden state.
- 2 gates (rather than 4).



GATED RECURRENT UNITS

$$r_t = \sigma(W_r w_t + U_r h_{t-1} + b_r)$$
$$z_t = \sigma(W_z w_t + U_z h_{t-1} + b_z)$$
$$\hat{h}_t = \phi(W_h w_t + U_h(r_t \circ h_{t-1}) + b_h)$$
$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \hat{h}_t$$

σ is a sigmoid, and ϕ is *tanh*.



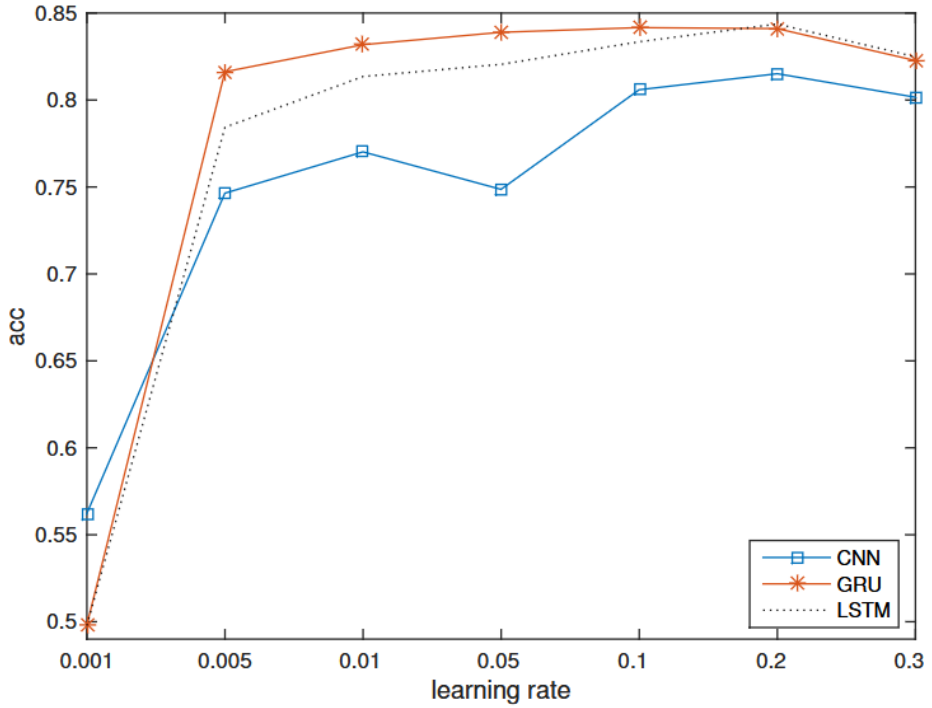
z_t is between 0 and 1, so h_t is just a convex combination of the old state h_{t-1}, and the candidate state $\hat{h}_t$.

LSTM VS GRU

- Which one is better?
- There is no one clear winner.
- It depends on the task, data, hyperparameters, etc.
- Let's take a look at some examples.
- Yin et al. 2017 trained three models on two tasks:
  - Models: LSTM, GRU, and CNN (convolutional neural network)
  - Tasks: Sentiment analysis, multiple-choice question answering

LSTM VS GRU

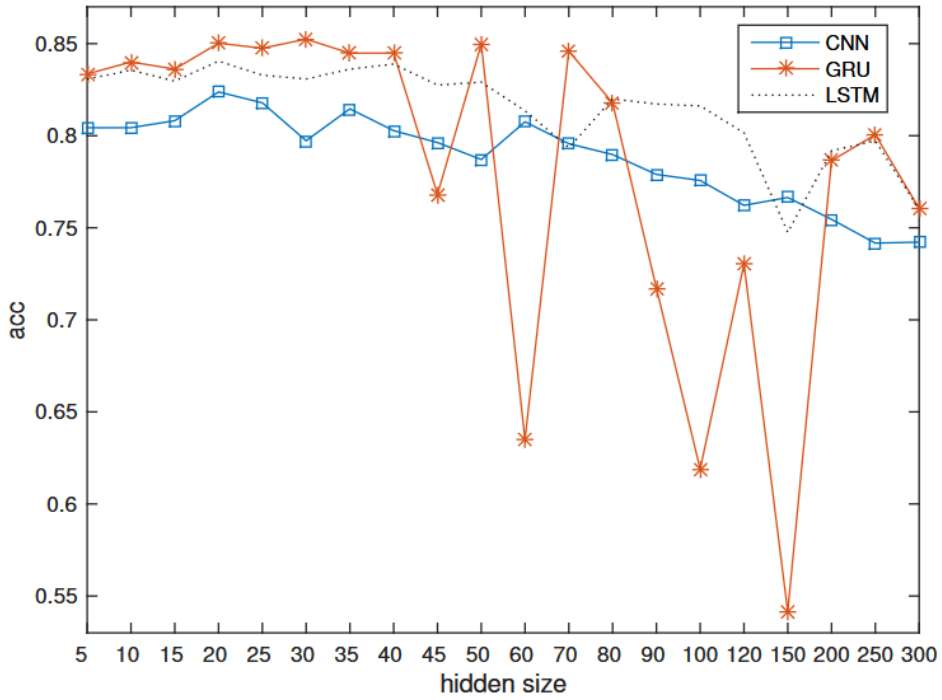
- Accuracy vs learning rate in sentiment analysis task.



[Yin, Kann, Yu, and Schütze 2017]

LSTM VS GRU

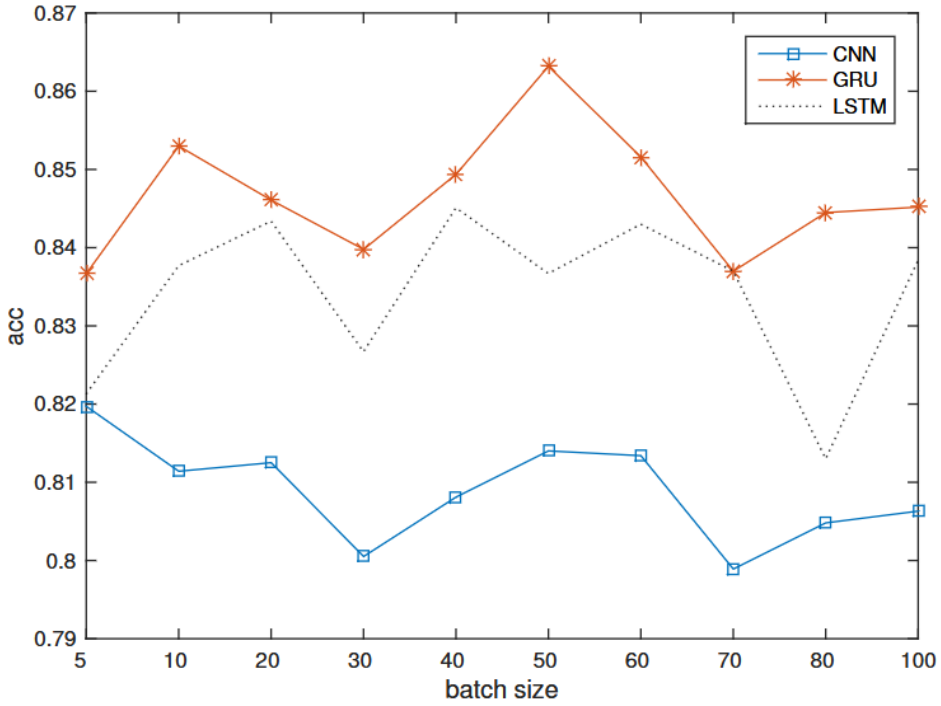
- Accuracy vs hidden state dimension in sentiment analysis task.



[Yin, Kann, Yu, and Schütze 2017]

LSTM VS GRU

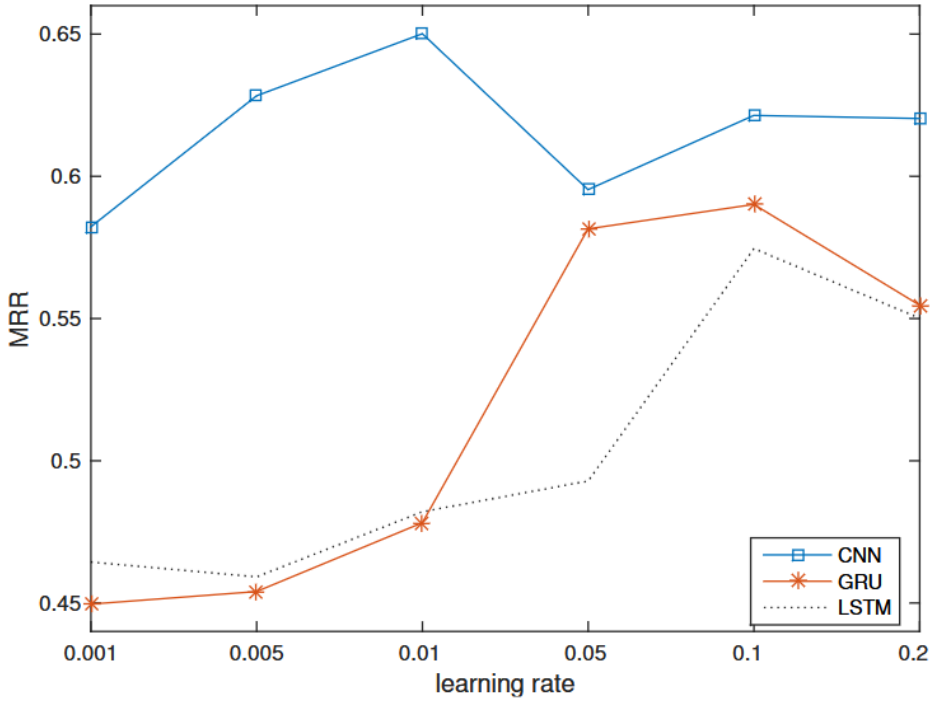
- Accuracy vs batch size in sentiment analysis task.



[Yin, Kann, Yu, and Schütze 2017]

LSTM VS GRU

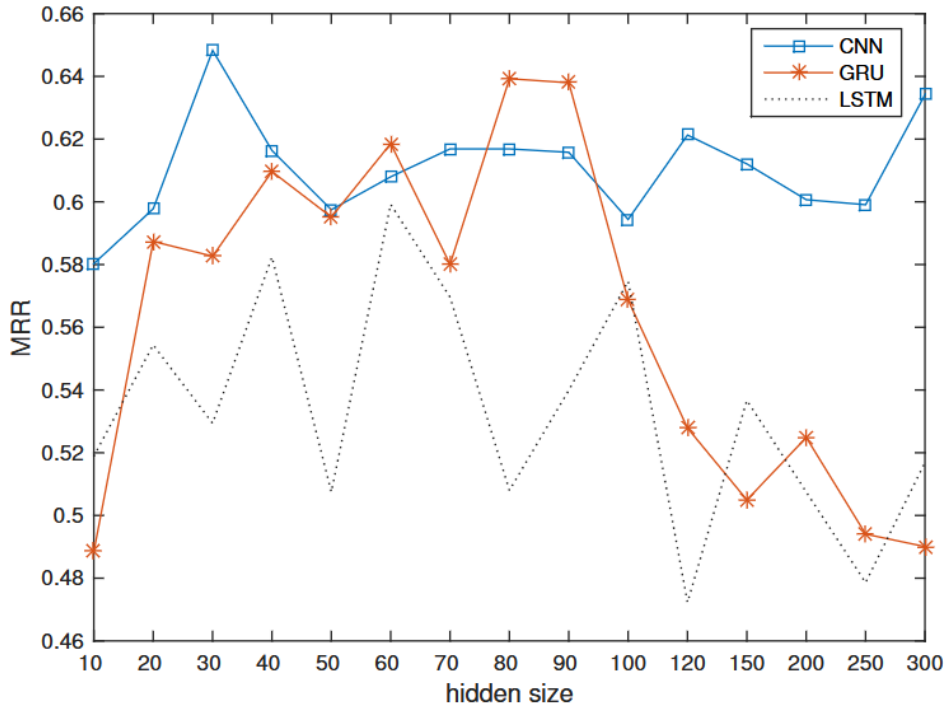
- Mean reciprocal rank vs learning rate in multiple-choice QA task.



[Yin, Kann, Yu, and Schütze 2017]

LSTM VS GRU

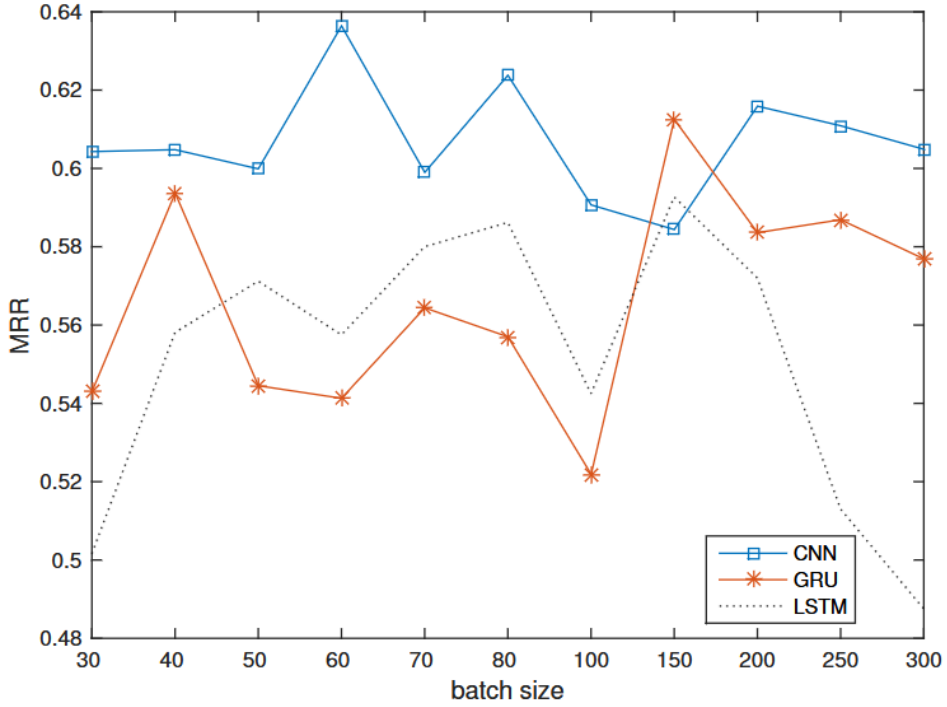
- Mean reciprocal rank vs hidden state dimension in multiple-choice QA task.



[Yin, Kann, Yu, and Schütze 2017]

LSTM VS GRU

- Mean reciprocal rank vs batch size in multiple-choice QA task.



[Yin, Kann, Yu, and Schütze 2017]

RNN SUMMARY

- GRUs can achieve comparable performance to LSTMs despite being simpler.
- Some terminology clarification:
  - GRUs are often considered a special case of LSTMs.
  - Both GRUs and LSTMs are often considered special cases of RNNs.
  - So "RNN" can be used as an umbrella term to include any method that processes sequential information one element at a time.
    - I.e., word-by-word, character-by-character

RNN DISADVANTAGES

- What are some disadvantages of RNNs (including LSTMs, GRUs)?
- They are not easily parallelizable.
  - If I want to compute the n-th hidden state, I need to process n tokens sequentially.
  - This is made even worse with deep RNNs.
    - The hidden state of the next layer is dependent on the hidden state of the previous layer.
- This limits their ability to scale to very large datasets. (but can still be done)
- Next lecture, we will look at transformers, which do not have this limitation.

QUESTIONS?