CS 577: NATURAL LANGUAGE PROCESSING

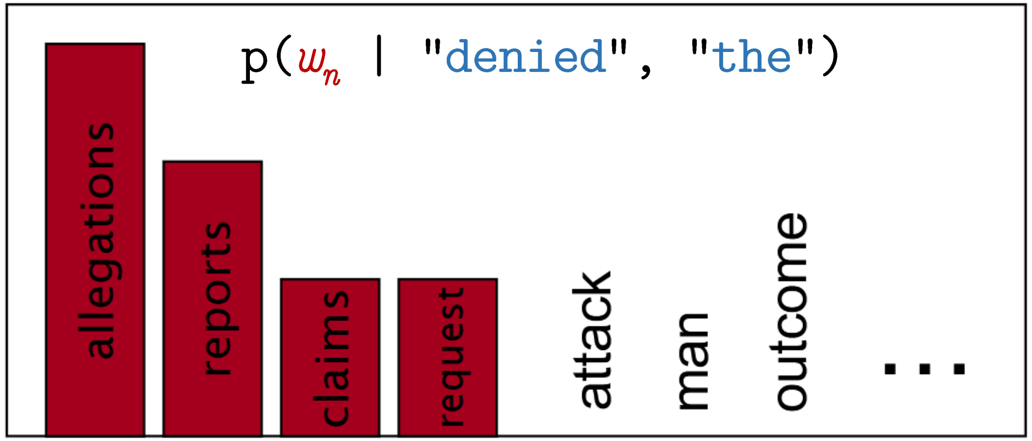
Abulhair Saparov

Lecture 4: Recurrent Neural Networks

PREVIOUSLY: DATA SPARSITY AND OVERFITTING

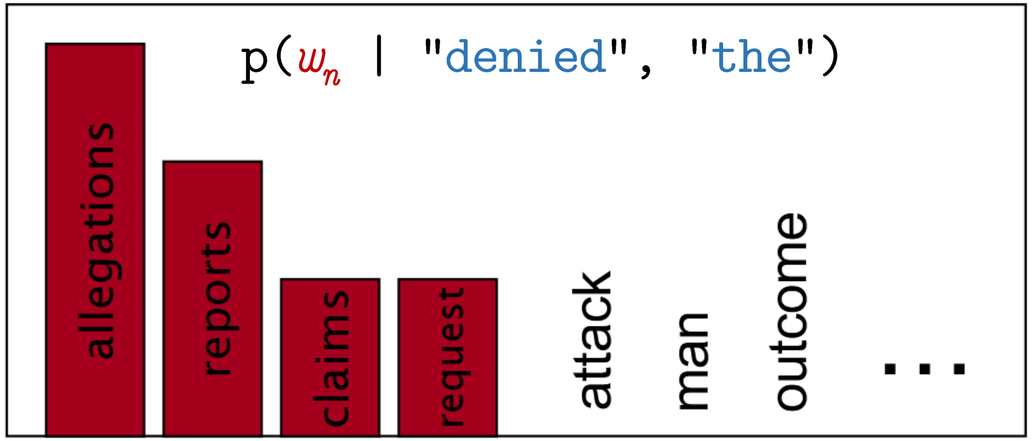
- How do we resolve the data sparsity issue with n-gram models?

- E.g., we have a 3-gram model where we have seen the following phrases in the training data:
  - "denied the allegations" 3 times
  - "denied the reports" 2 times
  - "denied the claims" 1 time
  - "denied the request" 1 time
  - No other instances of "denied the ___"



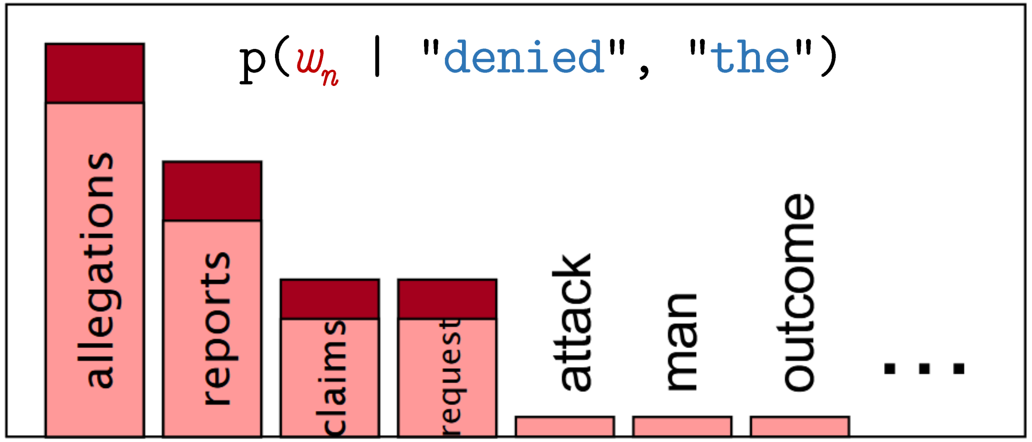
PREVIOUSLY: DATA SPARSITY AND OVERFITTING

- How do we resolve the data sparsity issue with n-gram models?
- One idea is called *smoothing*:
- The intuition is to “smooth” out the distribution of the next word, so that no word has probability 0.
- E.g., we have a 3-gram model where we have seen the following phrases in the training data:
  - “denied the allegations” 3 times
  - “denied the reports” 2 times
  - “denied the claims” 1 time
  - “denied the request” 1 time
  - No other instances of “denied the ___”



PREVIOUSLY: DATA SPARSITY AND OVERFITTING

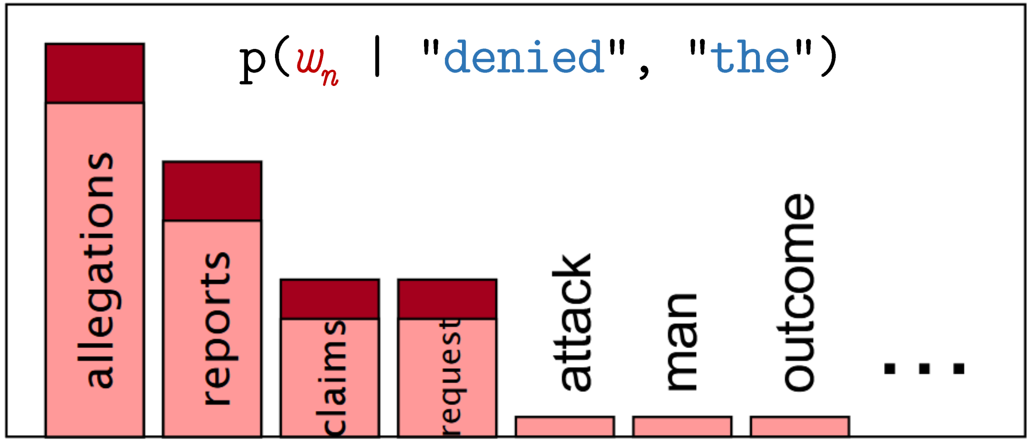
- How do we resolve the data sparsity issue with n-gram models?
- One idea is called *smoothing*:
- The intuition is to "smooth" out the distribution of the next word, so that no word has probability 0.
- E.g., we have a 3-gram model where we have seen the following phrases in the training data:
  - "denied the allegations" 3 times
  - "denied the reports" 2 times
  - "denied the claims" 1 time
  - "denied the request" 1 time
  - No other instances of "denied the ___"



SMOOTHING

- How do we resolve the data sparsity issue with n-gram models?
- Laplace smoothing (also called add-one smoothing):

$$p(w_n \mid w_1, \ldots, w_{n-1}) = \frac{(\text{\# of times } w_n \text{ appeared after } w_1, \ldots, w_{n-1}) + 1}{(\text{\# of times } w_1, \ldots, w_{n-1} \text{ appeared}) + V}$$



SMOOTHING

- How do we resolve the data sparsity issue with n-gram models?
- Laplace smoothing (also called add-one smoothing):

$$p(w_n \mid w_1, \ldots, w_{n-1}) = \frac{(\text{\# of times } w_n \text{ appeared after } w_1, \ldots, w_{n-1}) + 1}{(\text{\# of times } w_1, \ldots, w_{n-1} \text{ appeared}) + V}$$

- This is simple, but doesn’t work well in language modeling.
  - Consider the 4-gram model trained on Shakespeare.
  - For almost all 4-grams in the test set, the numerator in the above expression is 1.
- It is useful in other tasks, however.

BACKOFF

- How do we resolve the data sparsity issue with n-gram models?
- Another idea: Simultaneously use multiple n-gram models, with smaller *n*.

$$
\begin{aligned}
p(w_n \mid w_1, \ldots, w_{n-1}) &= \frac{\text{\# of times } w_n \text{ appeared after } w_1, \ldots, w_{n-1}}{\text{\# of times } w_1, \ldots, w_{n-1} \text{ appeared}} && \text{if } w_1, \ldots, w_n \text{ occurs in data} \\
&= \frac{\text{\# of times } w_n \text{ appeared after } w_2, \ldots, w_{n-1}}{\text{\# of times } w_2, \ldots, w_{n-1} \text{ appeared}} && \text{if } w_2, \ldots, w_n \text{ occurs in data} \\
&\ldots \\
&= \frac{\text{\# of times } w_n \text{ appeared after } w_{n-1}}{\text{\# of times } w_{n-1} \text{ appeared}} && \text{if } w_{n-1}, w_n \text{ occurs in data} \\
&= \frac{\text{\# of times } w_n \text{ appears}}{\text{total number of words}} && \text{otherwise.}
\end{aligned}
$$

INTERPOLATION

- How do we resolve the data sparsity issue with n-gram models?
- Another idea: Use multiple n-gram models, with interpolation.

$$p(w_n \mid w_1, \ldots, w_{n-1}) = \lambda_n \frac{\text{\# of times } w_n \text{ appeared after } w_1, \ldots, w_{n-1}}{\text{\# of times } w_1, \ldots, w_{n-1} \text{ appeared}}$$

$$+ \lambda_{n-1} \frac{\text{\# of times } w_n \text{ appeared after } w_2, \ldots, w_{n-1}}{\text{\# of times } w_2, \ldots, w_{n-1} \text{ appeared}}$$

$$\ldots$$

$$+ \lambda_2 \frac{\text{\# of times } w_n \text{ appeared after } w_{n-1}}{\text{\# of times } w_{n-1} \text{ appeared}}$$

$$+ \lambda_1 \frac{\text{\# of times } w_n \text{ appears}}{\text{total number of words}}$$

Require:
$\lambda_1 + \ldots + \lambda_n = 1$

INTERPOLATION

- This type of model is called a mixture model.
- Equivalent to first rolling an n-sided die to choose which n-gram to sample from, and then sampling from the corresponding n-gram model.

$$
\begin{aligned}
p(w_n \mid w_1, \ldots, w_{n-1}) = {} & p(w_n \mid w_1, \ldots, w_{n-1}, \text{choose 1-gram})\ p(\text{choose 1-gram}) \\
& + \ldots + p(w_n \mid w_1, \ldots, w_{n-1}, \text{choose n-gram})\ p(\text{choose n-gram}),
\end{aligned}
$$

- By the law of total probability.

$$
\begin{aligned}
= {} & p(w_n \mid w_1, \ldots, w_{n-1}, \text{choose 1-gram})\ \lambda_1 \\
& + \ldots + p(w_n \mid w_1, \ldots, w_{n-1}, \text{choose n-gram})\ \lambda_n.
\end{aligned}
$$

DATA SPARSITY AND OVERFITTING

(some NLP history)

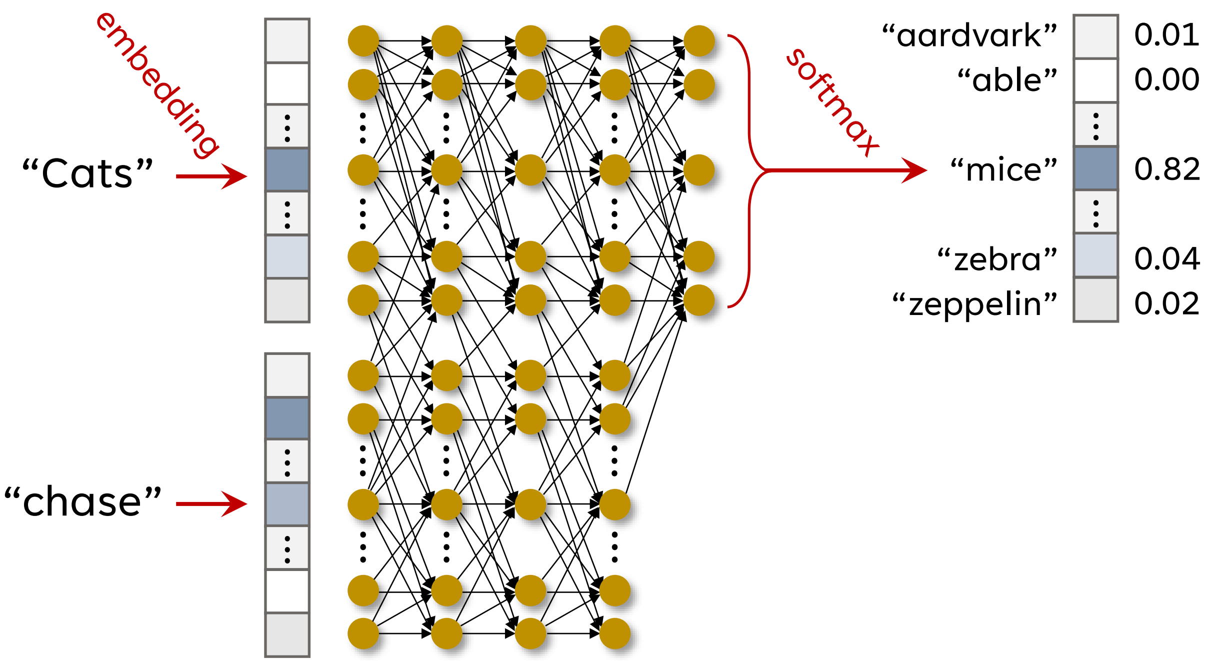
- Backoff performs better when combined with smoothing.
  - Kneser-Ney smoothing
  - Interpolated Kneser-Ney
  - Skip n-grams

- Another idea to address the data sparsity issue, is to use a different machine learning model.
  - Perhaps a neural network?
- Smoothing/interpolation superseded by neural language models.

BETTER METHODS FOR TEXT CLASSIFICATION

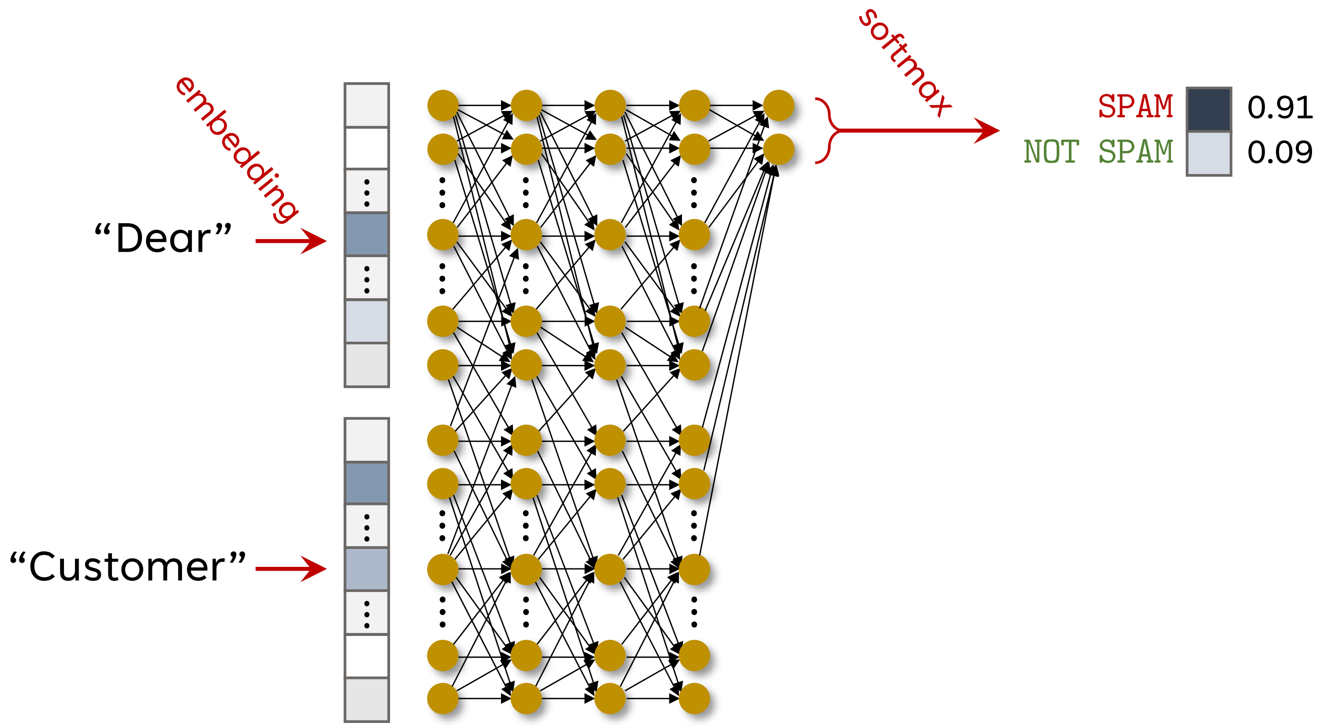
- We previously discussed language modeling:
  - The task of predicting the next word, given previous words.
  - n-gram models are simple but not very accurate for small n.
  - They suffer from data sparsity and overfitting for large n.
- Are there alternative machine learning models that would be better?
- Maybe MLP?

MLP (?) FOR LANGUAGE MODELING



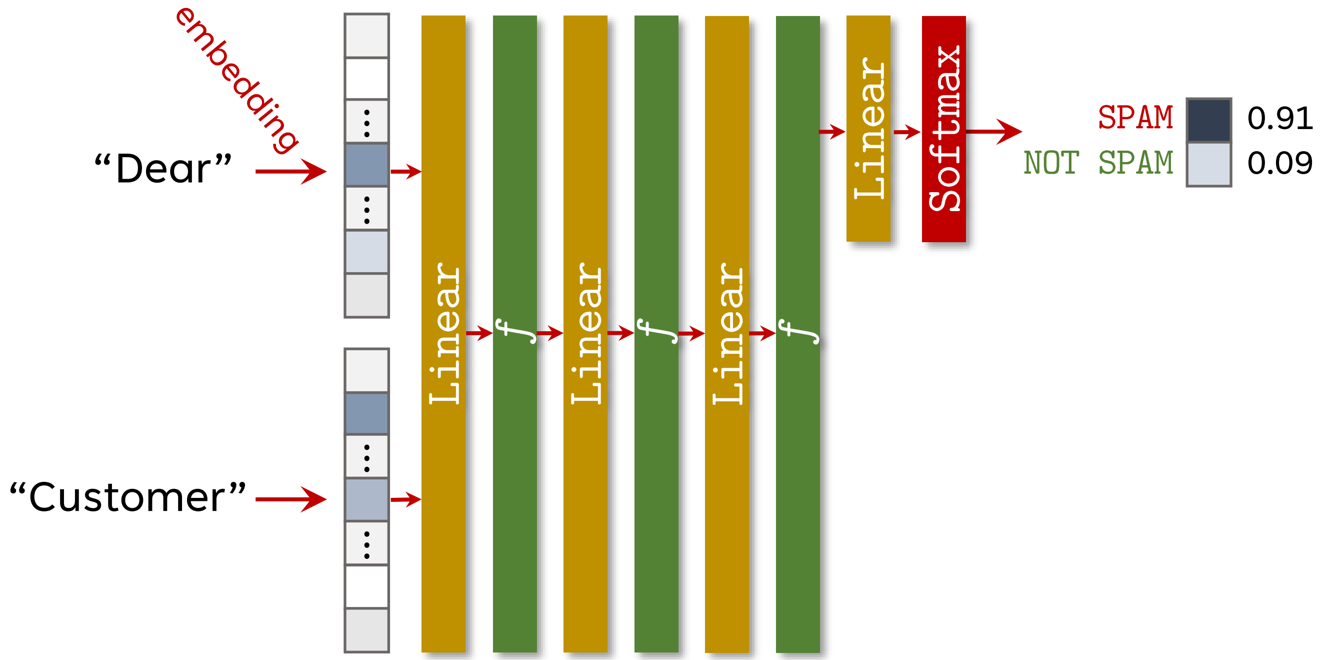
- Embed each input word into a real-valued vector with dimension d.
- Concatenate the embedding vectors and input into MLP.
- Input layer has dimension $N \cdot d$.
- Output layer has dimension V.
- Here, the MLP has 3 hidden layers.

MLP (?) FOR SPAM DETECTION



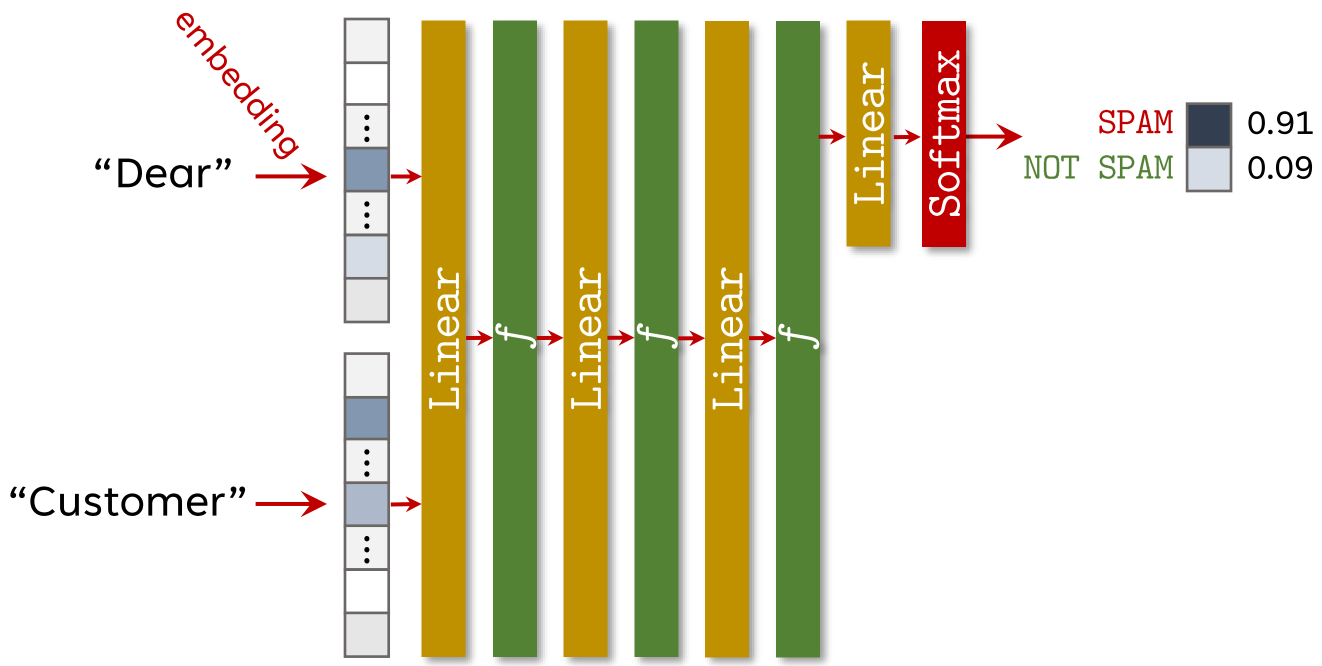
- MLPs can be used for other text classification tasks.

MLP (?) FOR SPAM DETECTION



- MLPs can be used for other text classification tasks.
- Each linear layer computes the function: `Linear(x) = Wx + b`
- Each nonlinearity (f) computes an activation function element-wise.
- Example activation functions:
  - Sigmoid
  - tanh
  - ReLU

MLP (?) FOR SPAM DETECTION



- MLPs can be used for other text classification tasks.
- Potential disadvantages?
  - Thoughts?
  - How many parameters (weights) are there?
  - $(Nd)^2L + NdD_{output}$

    L is the number of hidden layers.

    N is the maximum number of input words.

MLPS (?) FOR TEXT CLASSIFICATION

- More expressive machine learning models are more prone to overfitting.
  - They need more data to train.
  - I.e., they are less data efficient.
- Number of parameters/weights is a measure of model expressiveness.
- Pure MLPs are not very data efficient.
  - Especially if the embedding dimension d or input length N is very large.
  - E.g., in GPT-3, $d = 12288$, $N = 4096$.

ALTERNATIVE NEURAL ARCHITECTURES

- But there is a very large space of different neural architectures.
- One natural proposal is to model the sequential nature of language.
    - Humans understand language word-by-word.
    - Humans hear/read each word and update an internal representation in their brain.
- Can we capture this kind of processing in a neural architecture?

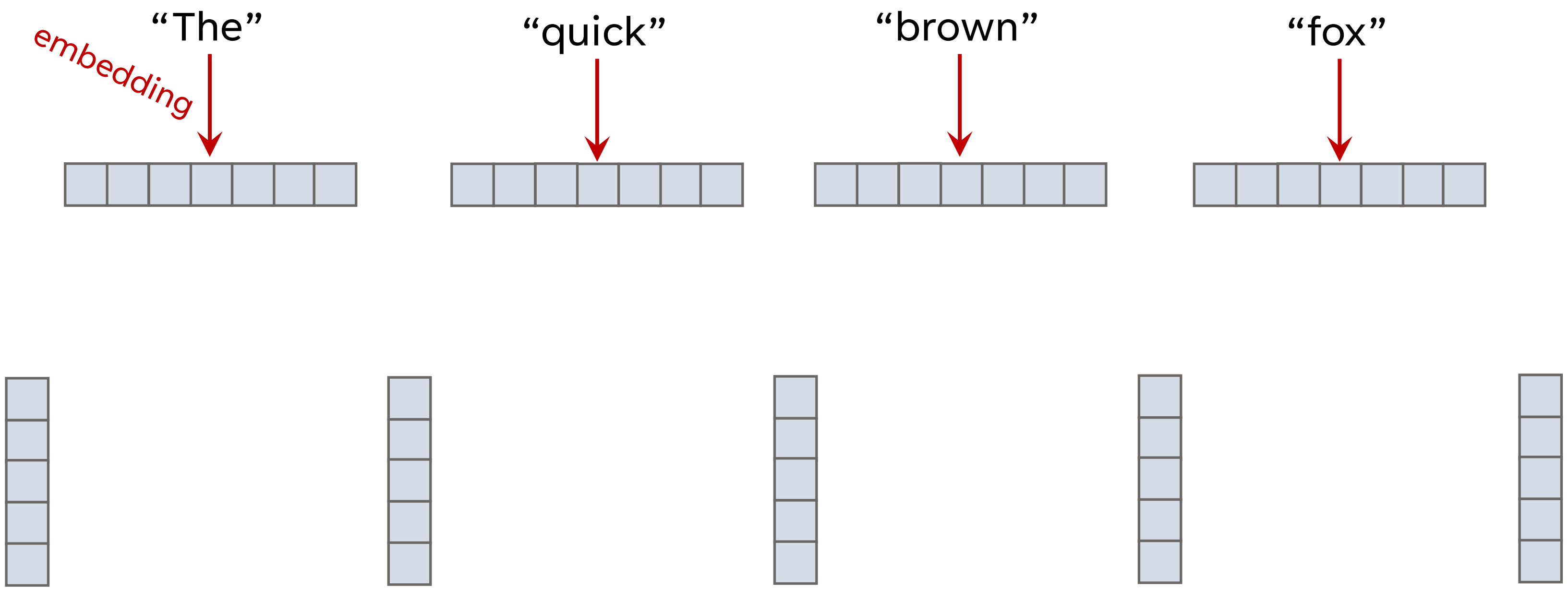
RECURRENT NEURAL NETWORKS

- Recurrent neural networks (RNNs; Elman 1990) attempt to capture this sequential (word-by-word) processing.

"The" "quick" "brown" "fox"

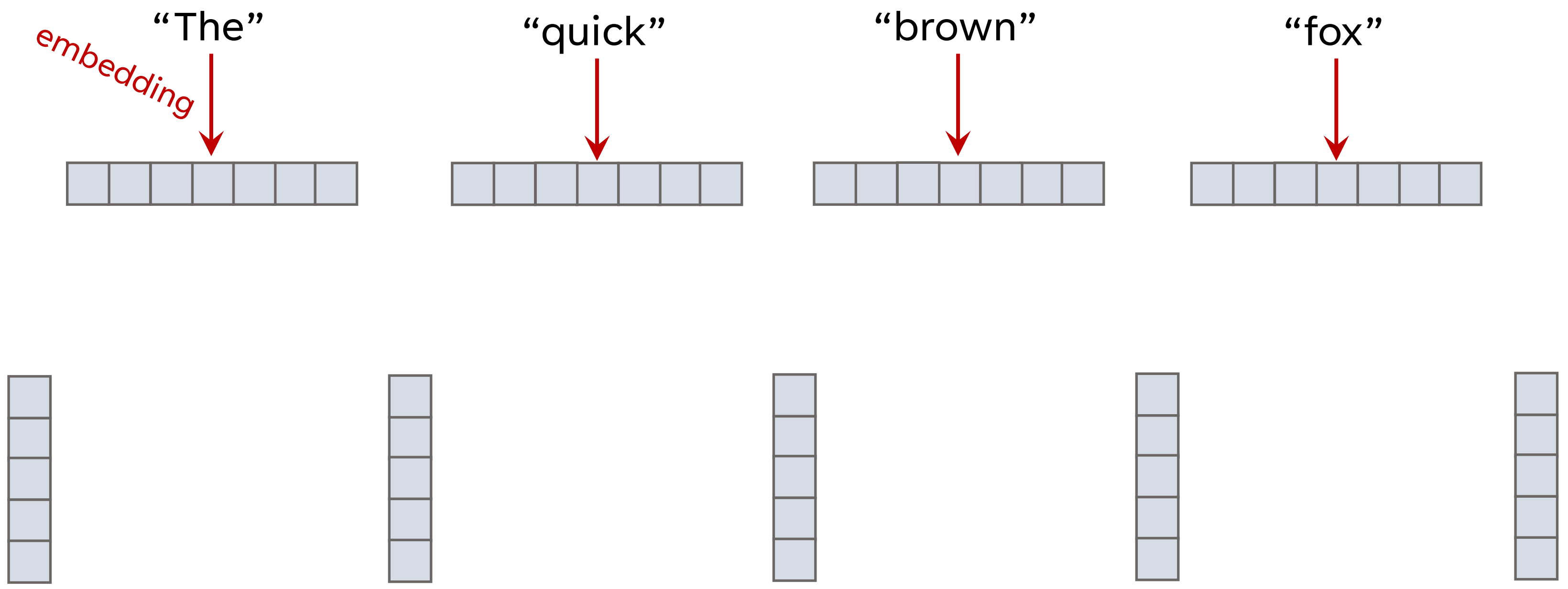
RECURRENT NEURAL NETWORKS

- Embed each input word into vectors of dimension d_{emb}.
- The RNN keeps a hidden state vector with dimension d_{hid}.



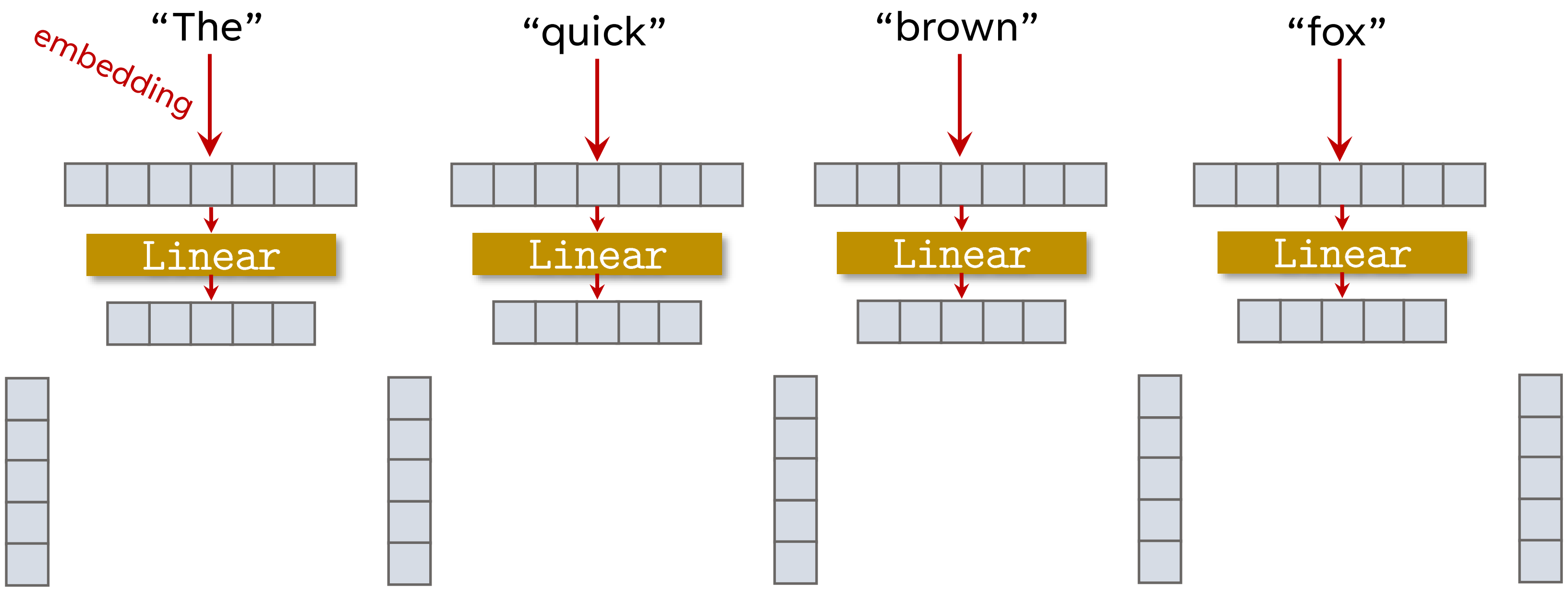
RECURRENT NEURAL NETWORKS

- The RNN combines each word with the previous hidden state, to produce the next hidden state.



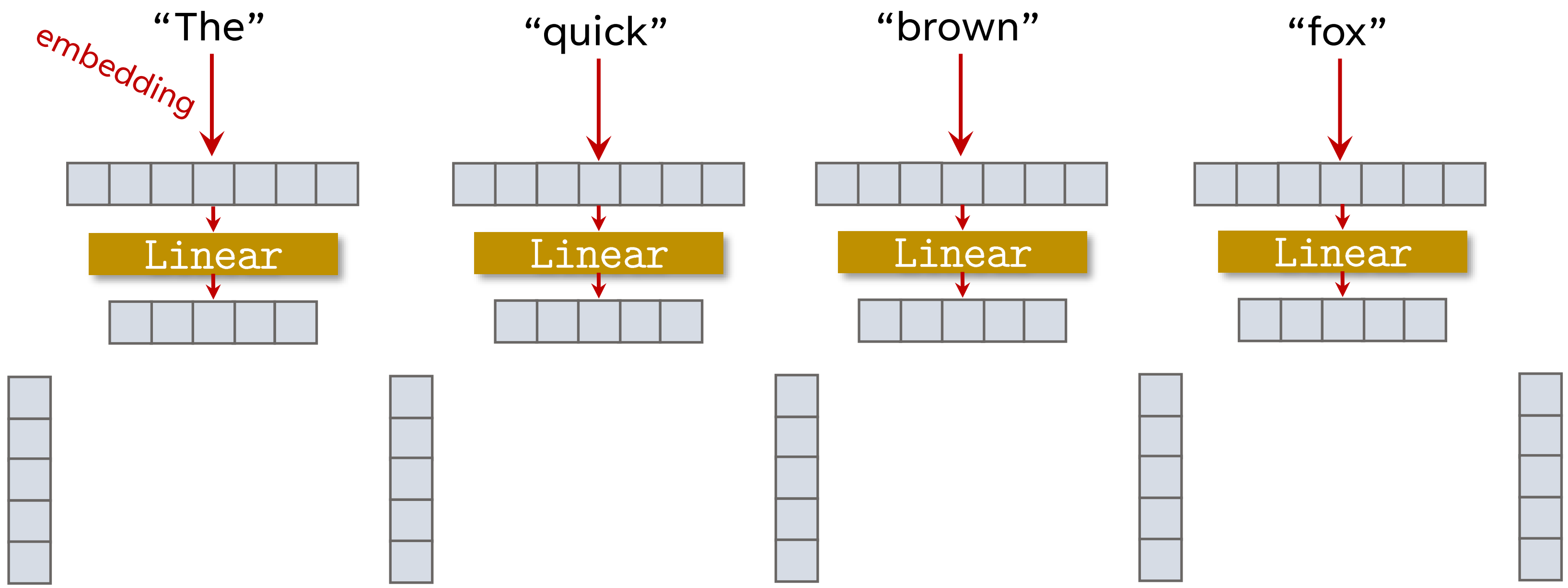
RECURRENT NEURAL NETWORKS

- To do so, we need to convert the embeddings into d_{hid}-dimensional vectors.
- We do this with a linear layer.



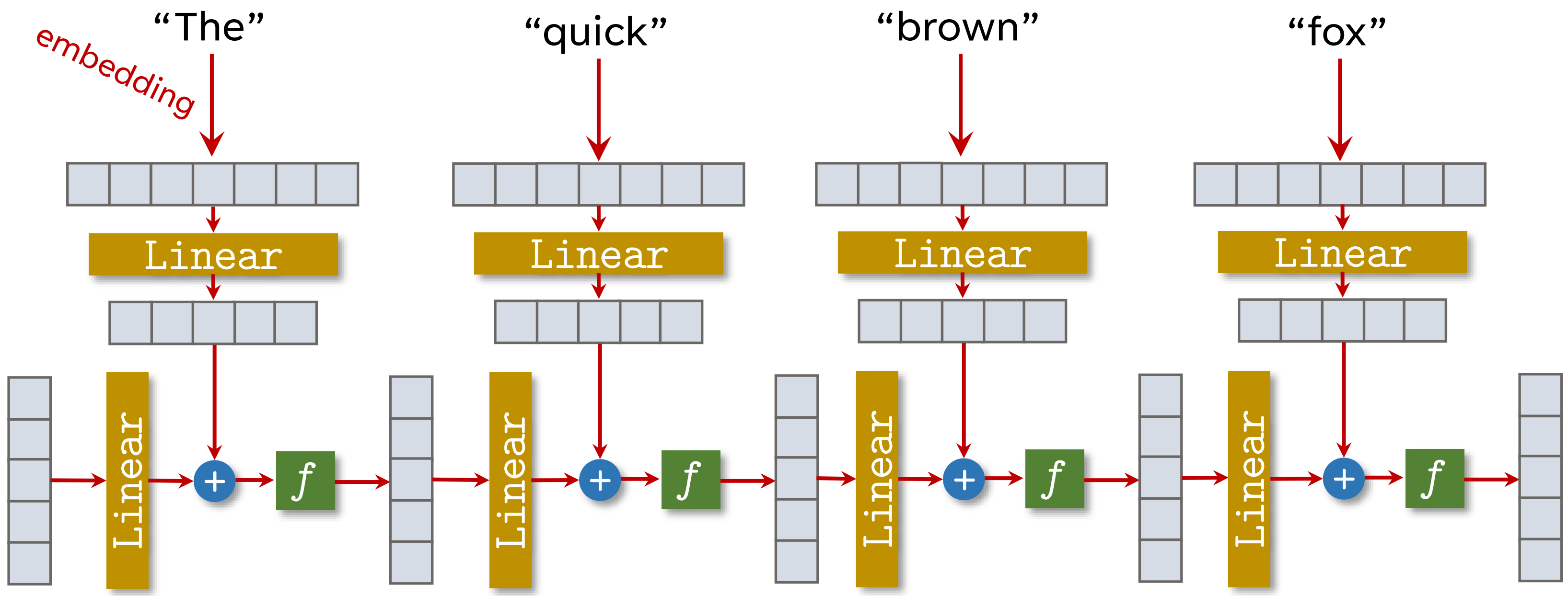
RECURRENT NEURAL NETWORKS

- Importantly, these linear layers are coupled.
- Each linear layer has the same weights as the other linear layers.



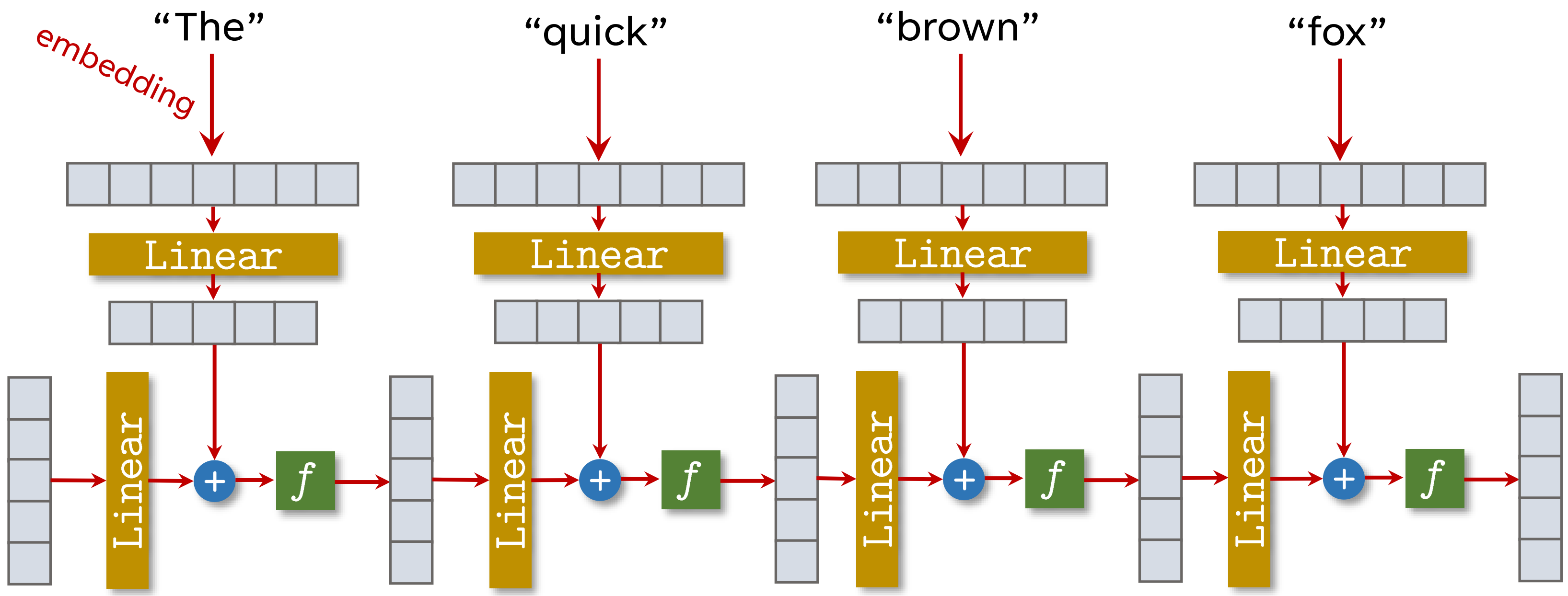
RECURRENT NEURAL NETWORKS

- Now the word vectors and hidden state have the same dimension, we combine them to produce the next hidden state.



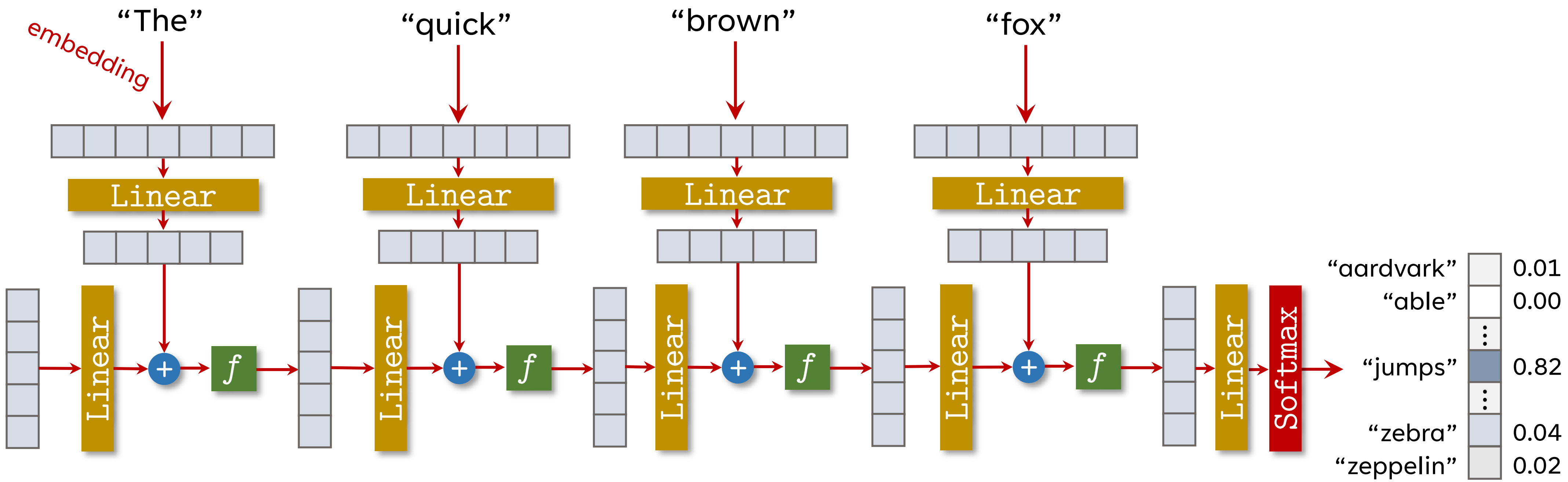
RECURRENT NEURAL NETWORKS

- The linear layers acting on the hidden states are also coupled.



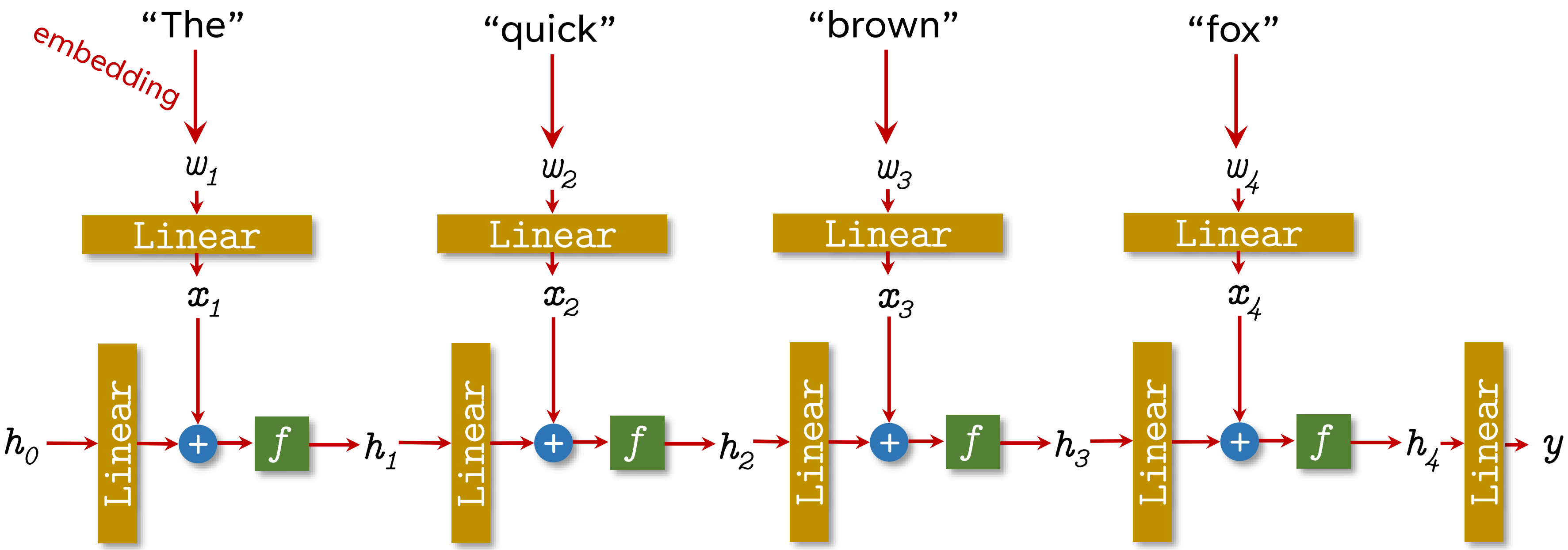
RECURRENT NEURAL NETWORKS

- Once we have the last hidden state, we can use it to make a prediction.
- In the example, we have a language modeling/next-word prediction task.



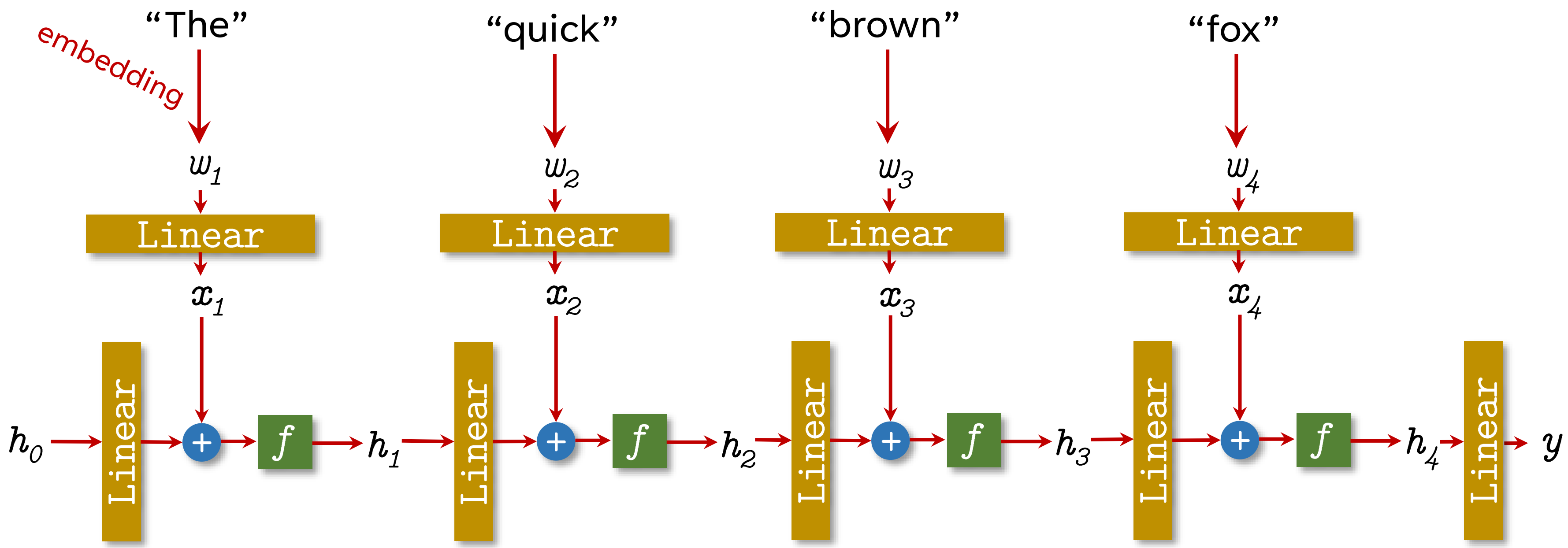
RECURRENT NEURAL NETWORKS

- It's often easier to depict neural architectures symbolically.
- Note h_0 is often set to a vector of zeros, but it can also be learned.



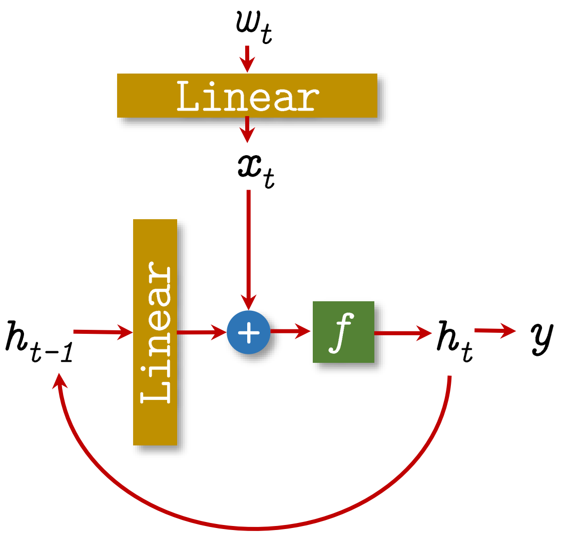
RECURRENT NEURAL NETWORKS

- Why “recurrent”?



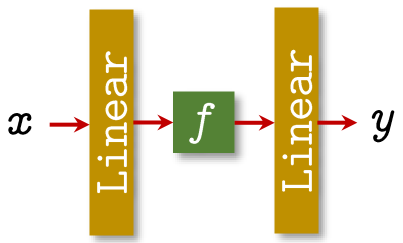
RECURRENT NEURAL NETWORKS

- Why “recurrent”?
- Converting from this into the feedforward (i.e., directed acyclic) form is called “unfolding in time”.



TRAINING RNNS

- We train RNNs the same way we train most neural networks:
- Gradient descent, using backprop to compute gradients.
- How do we compute gradients when some parameters are coupled?
- Consider the following simple MLP: (no coupled parameters)

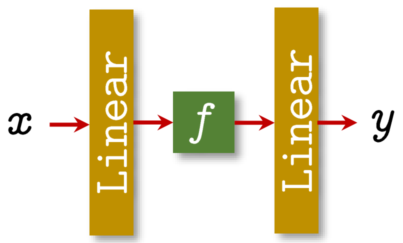


$$y = b_2 + W_2 \cdot f(b_1 + W_1 x)$$

TRAINING RNNS

- Suppose we have a training example $(\hat{x}, \hat{y})$.
- And we have some loss function $L(y, \hat{y})$.
- We can compute the gradient of the loss:
- Similarly, compute gradients for b_1 and b_2.

$$
\begin{aligned}
\nabla_{W_2} L(y,\hat{y}) &= L'(y,\hat{y}) \cdot \nabla_{W_2} \mathrm{y} \\
&= L'(y,\hat{y}) \cdot \nabla_{W_2} (b_2 + W_2 \cdot f(b_1 + W_1 \hat{x})) \\
&= L'(y,\hat{y}) \cdot f(b_1 + W_1 \hat{x})
\end{aligned}
$$



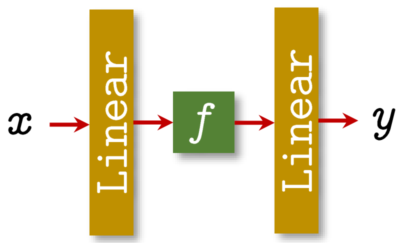
$$y = b_2 + W_2 \cdot f(b_1 + W_1 x)$$

$$
\begin{aligned}
\nabla_{W_1} L(y,\hat{y}) &= L'(y,\hat{y}) \cdot \nabla_{W_1} \mathrm{y} \\
&= L'(y,\hat{y}) \cdot \nabla_{W_1} (b_2 + W_2 \cdot f(b_1 + W_1 \hat{x})) \\
&= L'(y,\hat{y}) \cdot W_2 \cdot \nabla_{W_1} f(b_1 + W_1 \hat{x}) \\
&= L'(y,\hat{y}) \cdot W_2 \cdot f'(b_1 + W_1 \hat{x}) \cdot \nabla_{W_1} (b_1 + W_1 \hat{x})^{\mathrm{T}} \\
&= L'(y,\hat{y}) \cdot W_2 \cdot f'(b_1 + W_1 \hat{x}) \cdot \hat{x}^{\mathrm{T}}
\end{aligned}
$$

[en.wikipedia.org/wiki/Matrix_calculus]

TRAINING RNNS

- But now let's consider the case where the two linear layers are coupled.
- Notice the result is just the sum of the gradients from the uncoupled case.
- Gradient accumulation



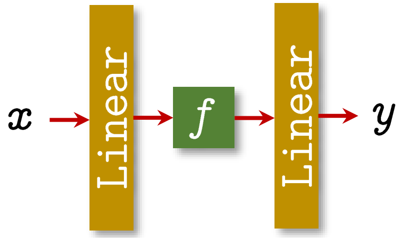
$$y = b_1 + W_1 \cdot f(b_1 + W_1 x)$$

$$\begin{aligned}
\nabla_{W_1} L(y,\hat{y}) &= L'(y,\hat{y}) \cdot \nabla_{W_1} y \\
&= L'(y,\hat{y}) \cdot \nabla_{W_1}(b_1 + W_1 \cdot f(b_1 + W_1 \hat{x})) \\
&= L'(y,\hat{y}) \cdot (W_1 \cdot f'(b_1 + W_1 \hat{x}) \cdot \hat{x}^T + f(b_1 + W_1 \hat{x})) \\
&= L'(y,\hat{y}) \cdot W_1 \cdot f'(b_1 + W_1 \hat{x}) \cdot \hat{x}^T + L'(y,\hat{y}) \cdot f(b_1 + W_1 \hat{x})
\end{aligned}$$

[en.wikipedia.org/wiki/Matrix_calculus]

TRAINING RNNS

- Note that most modern ML libraries will compute gradients automatically.
- But it's good to know what's happening under the hood.
  - Useful if something goes wrong -> debugging.
  - Also useful to think about new techniques for better ML.



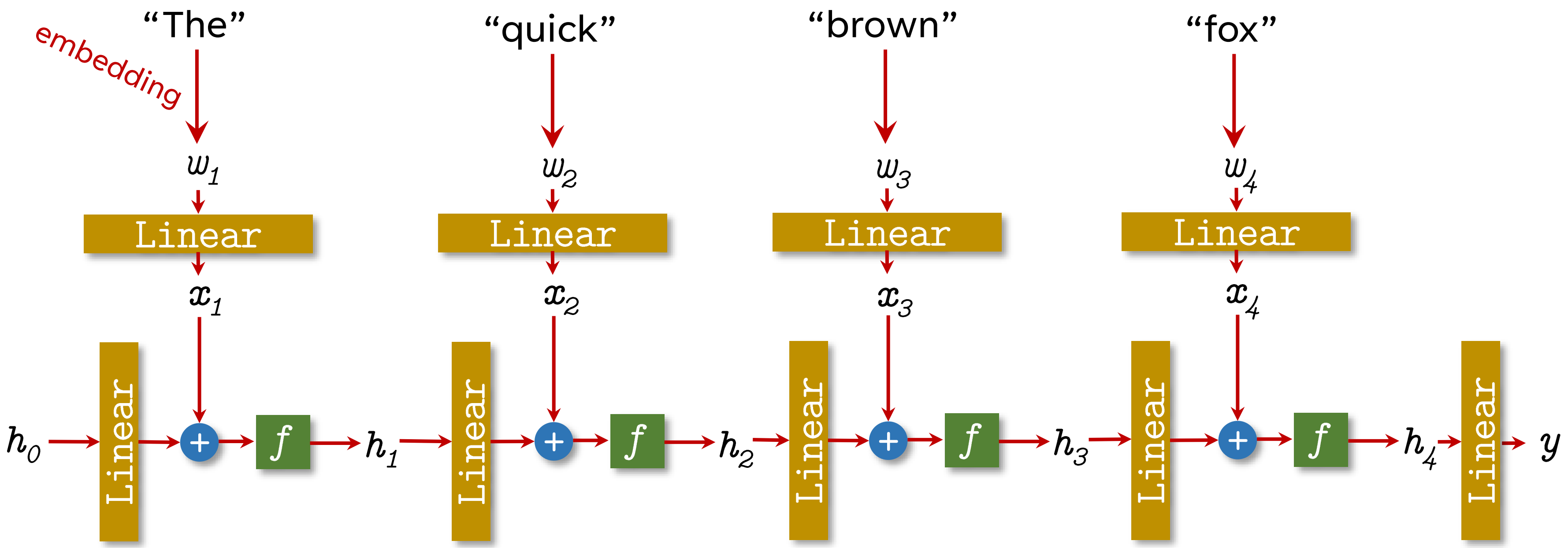
$$y = b_1 + W_1 \cdot f(b_1 + W_1 x)$$

$$\begin{aligned}\nabla_{W_1} L(y,\hat{y}) &= L'(y,\hat{y}) \cdot \nabla_{W_1} y \\ &= L'(y,\hat{y}) \cdot \nabla_{W_1} (b_1 + W_1 \cdot f(b_1 + W_1 \hat{x})) \\ &= L'(y,\hat{y}) \cdot (W_1 \cdot f'(b_1 + W_1 \hat{x}) \cdot \hat{x}^T + f(b_1 + W_1 \hat{x})) \\ &= L'(y,\hat{y}) \cdot W_1 \cdot f'(b_1 + W_1 \hat{x}) \cdot \hat{x}^T + L'(y,\hat{y}) \cdot f(b_1 + W_1 \hat{x})\end{aligned}$$

[en.wikipedia.org/wiki/Matrix_calculus]

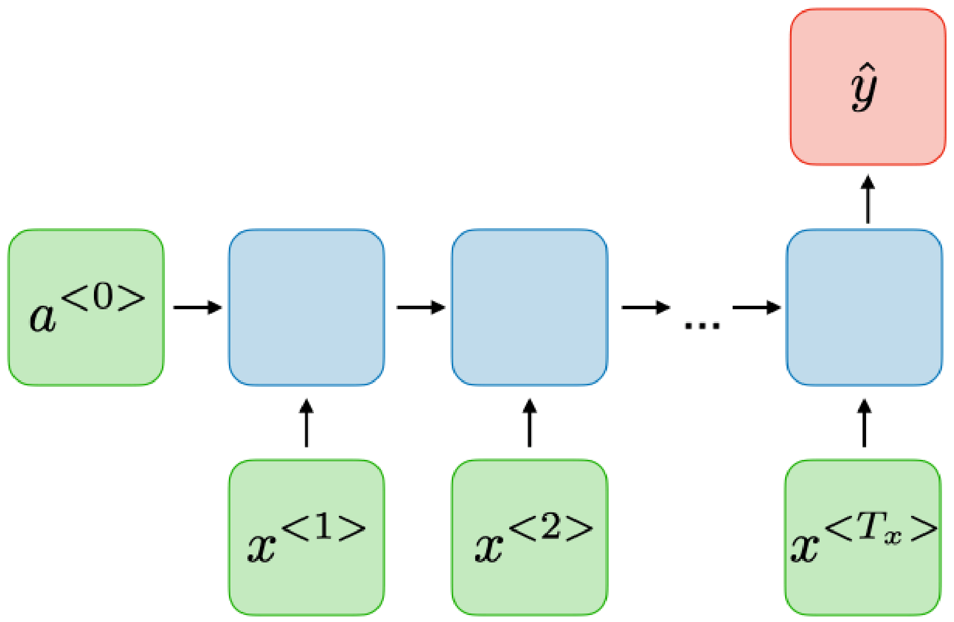
RNN APPLICATIONS

- RNNs have a very wide variety of applications.
- Beyond simple text classification.



RNN APPLICATIONS

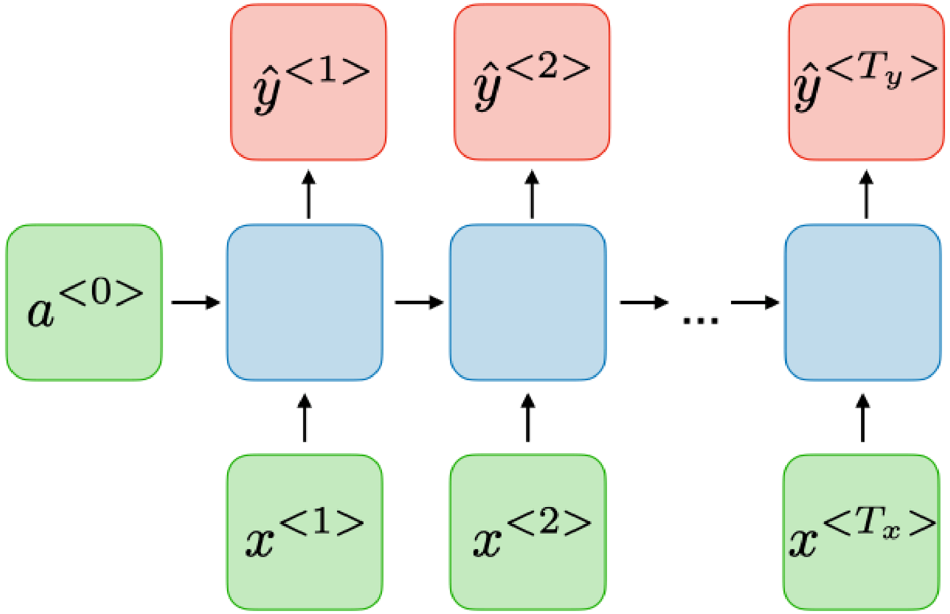
- RNNs have a very wide variety of applications.
- Beyond simple text classification.



[Afshine Amidi and Shervine Amidi, Recurrent Neural Networks cheatsheet]

RNN APPLICATIONS

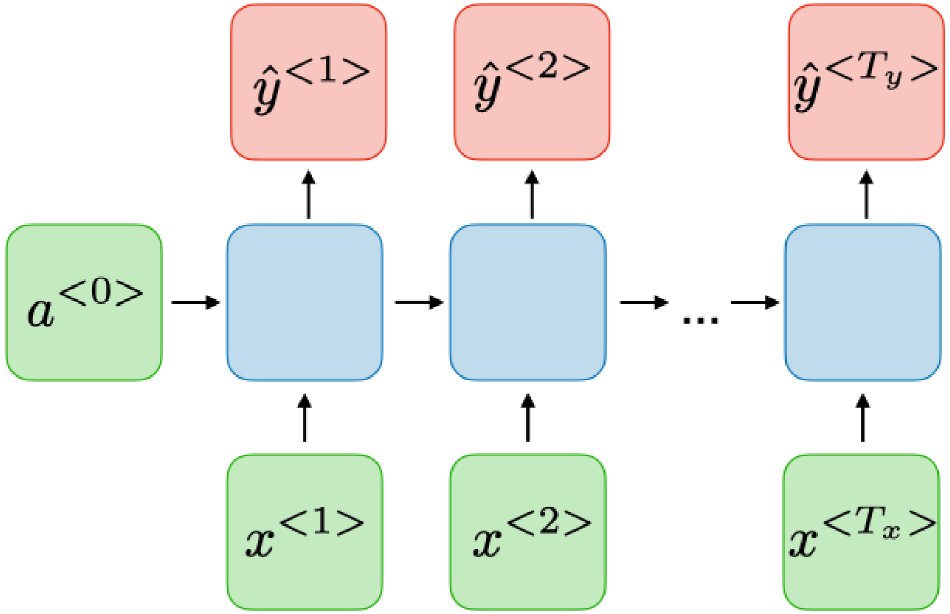
- We can make more than one prediction.
- For example, we can make a prediction per input word.



[Afshine Amidi and Shervine Amidi, Recurrent Neural Networks cheatsheet]

RNN APPLICATIONS

- Example tasks:
  - Part-of-speech tagging, named-entity recognition



Input: The quick brown fox jumped.
Output: DET ADJ ADJ NN V

When training, for each example, we sum the loss over all predictions.

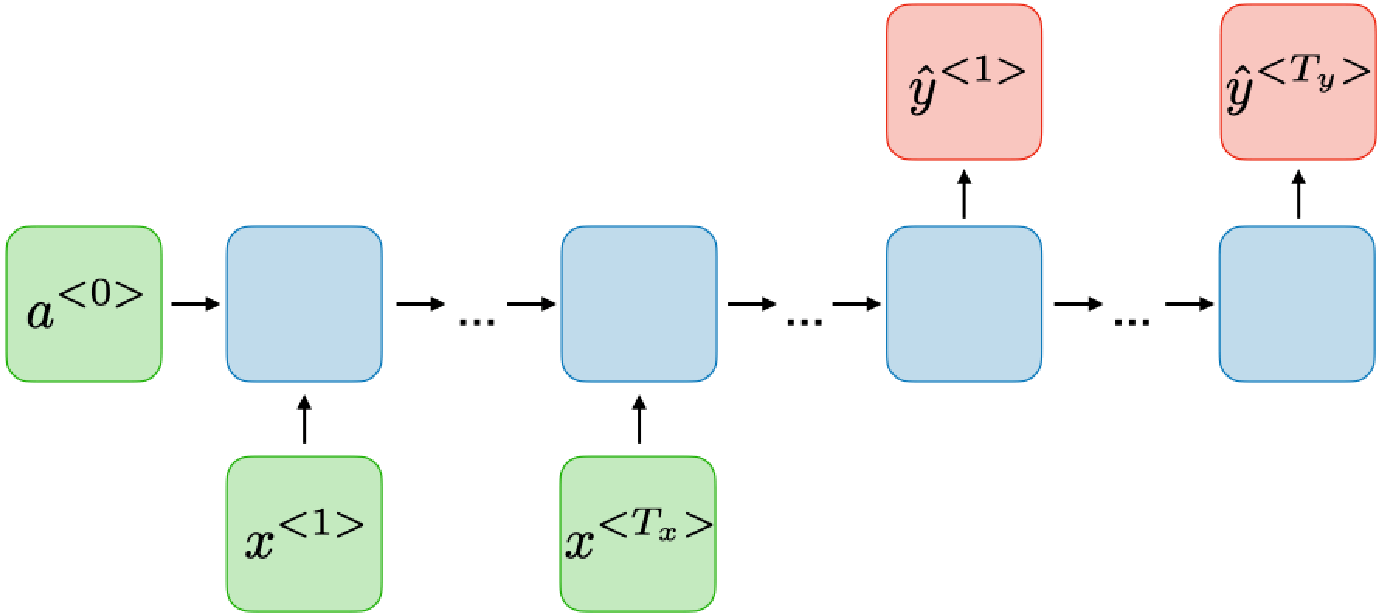
Example prediction:
Input: The quick brown fox jumped.
Prediction: DET ADJ NN NN V

Total loss = *L*(DET,DET) + *L*(ADJ,ADJ) + *L*(ADJ,NN) + *L*(NN,NN) + *L*(V,V)

[Afshine Amidi and Shervine Amidi, Recurrent Neural Networks cheatsheet]

RNN APPLICATIONS

- The number of output predictions doesn't need to match the number of input predictions.

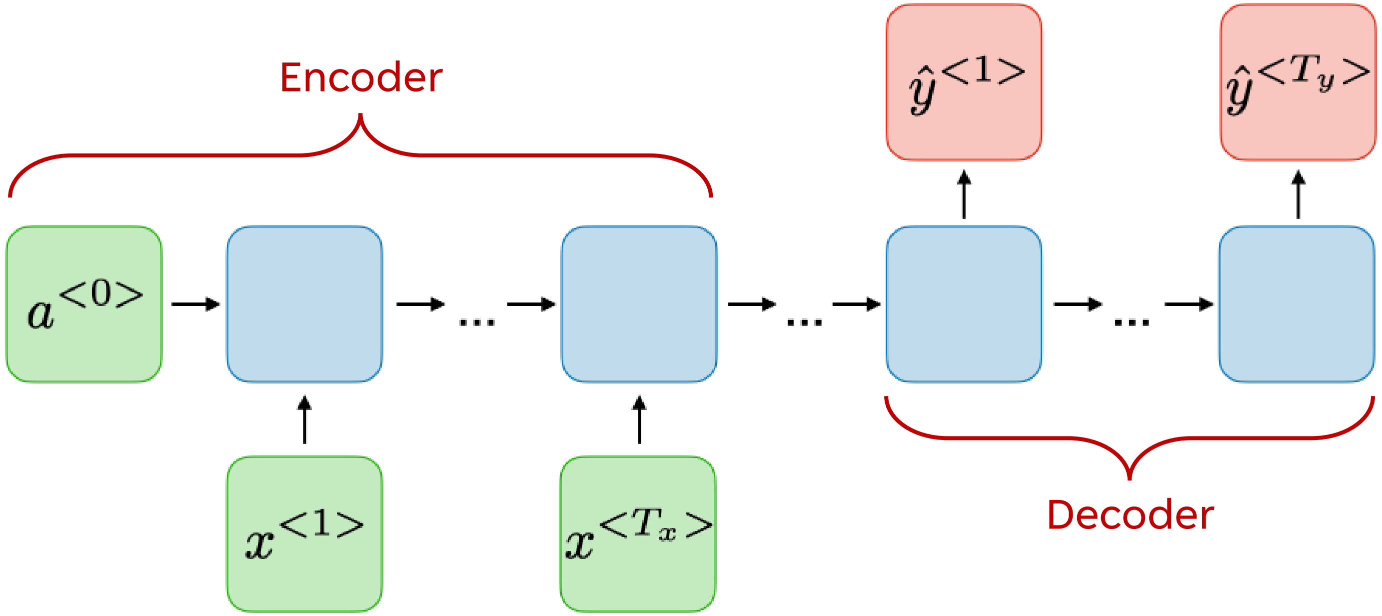


[Afshine Amidi and Shervine Amidi, Recurrent Neural Networks cheatsheet]

RNN APPLICATIONS

- Example tasks:
  - Machine translation

Input: The quick brown fox jumped over the lazy dog.
Output: 素早い茶色のキツネは怠け者の犬を飛び越えました。



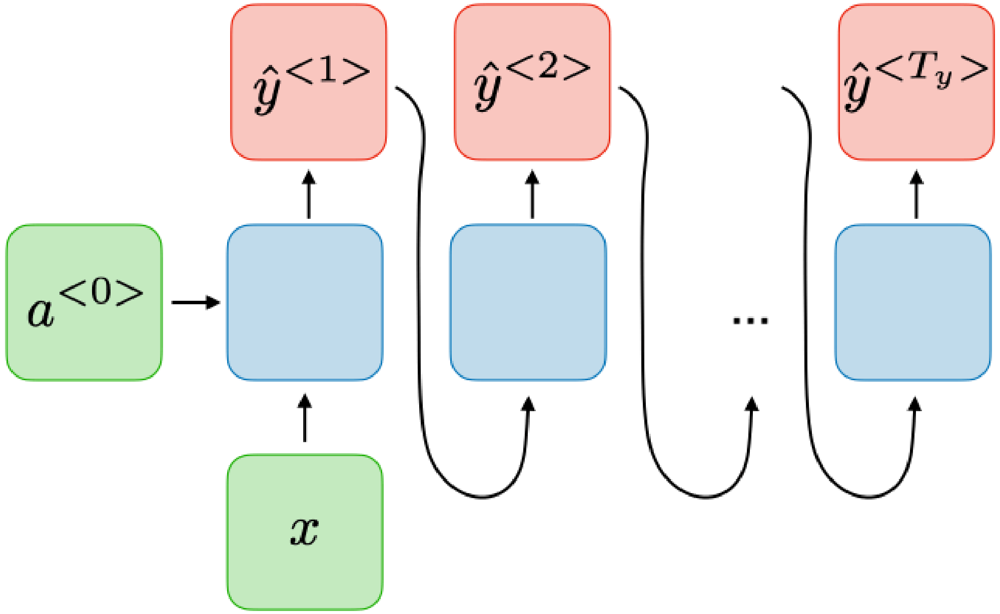
[Afshine Amidi and Shervine Amidi, Recurrent Neural Networks cheatsheet]

RNN APPLICATIONS

- It can be used in non-text applications.
- Example task: music generation

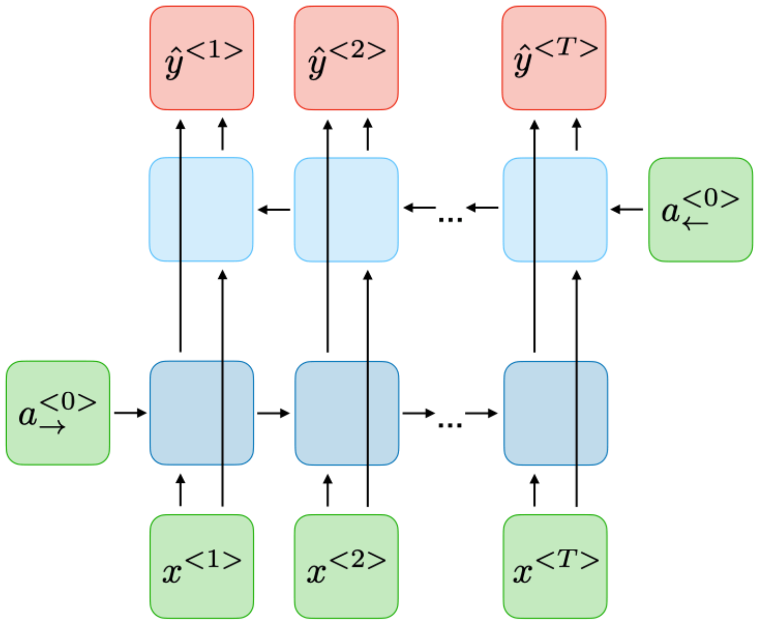
Input: birthday

Output: 



BIDIRECTIONAL RNN

- Bidirectional RNNs (BiRNNs) can be used in tasks where we want to gather information from words on both the left and right sides.

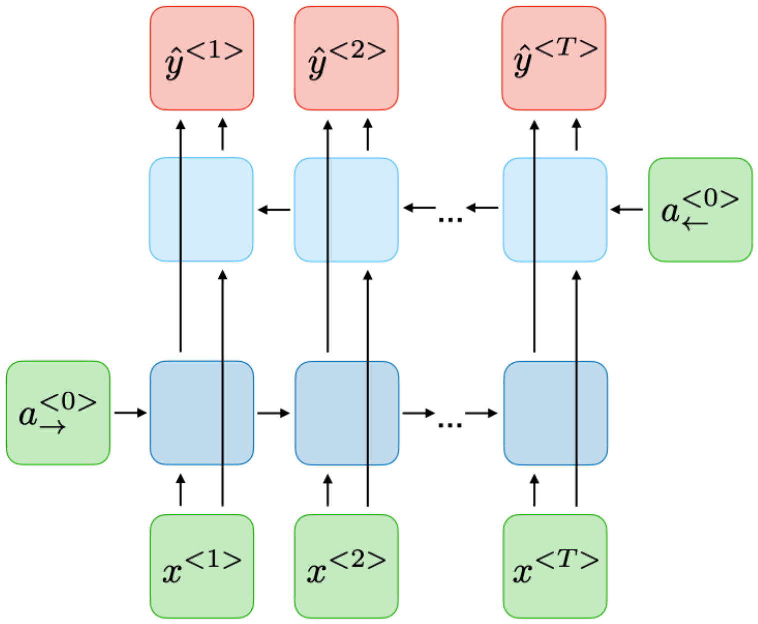


[Afshine Amidi and Shervine Amidi, Recurrent Neural Networks cheatsheet]

BIDIRECTIONAL RNN

- This is useful in the masked language modeling task.

  (a *good* unidirectional RNN could also solve this task)



Input: The quick brown ___ jumped.
Output: fox

Input: I am ___.
Output: running

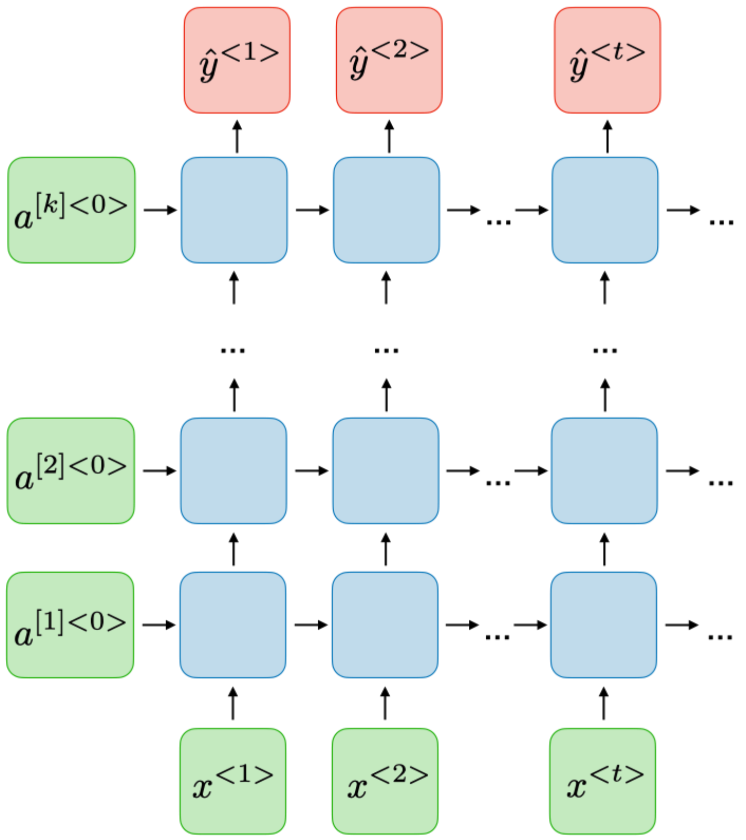
Input: I am ___ hungry.
Output: so

Input: I am ___ hungry; I just ate.
Output: not

[Afshine Amidi and Shervine Amidi, Recurrent Neural Networks cheatsheet]

DEEP RNN

- We can stack many layers of RNNs.



[Afshine Amidi and Shervine Amidi, Recurrent Neural Networks cheatsheet]

RNN GRADIENTS

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

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

- 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?

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

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

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

RNN 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).

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

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

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

VANISHING/EXPLODING GRADIENTS

- How do we solve this problem?
- Pick activation functions whose derivatives are 1 (ReLU).
- Gradient clipping:

  If the gradient vector v has magnitude larger than v_{max},

  divide it by $||v||/v_{max}$, so that its magnitude is at most v_{max}.

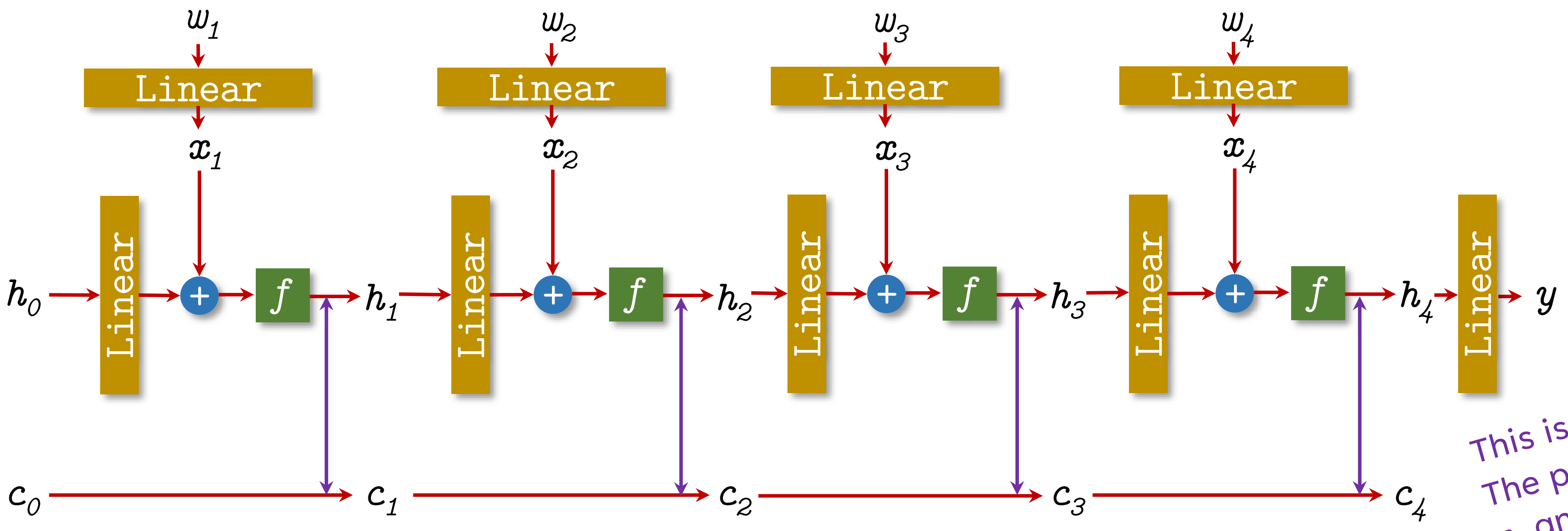
$$\nabla_{W_1} y = g_n'(\ldots) \cdot \nabla_{W_1} g_{n-1}(\ldots)$$

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

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

VANISHING/EXPLODING GRADIENTS

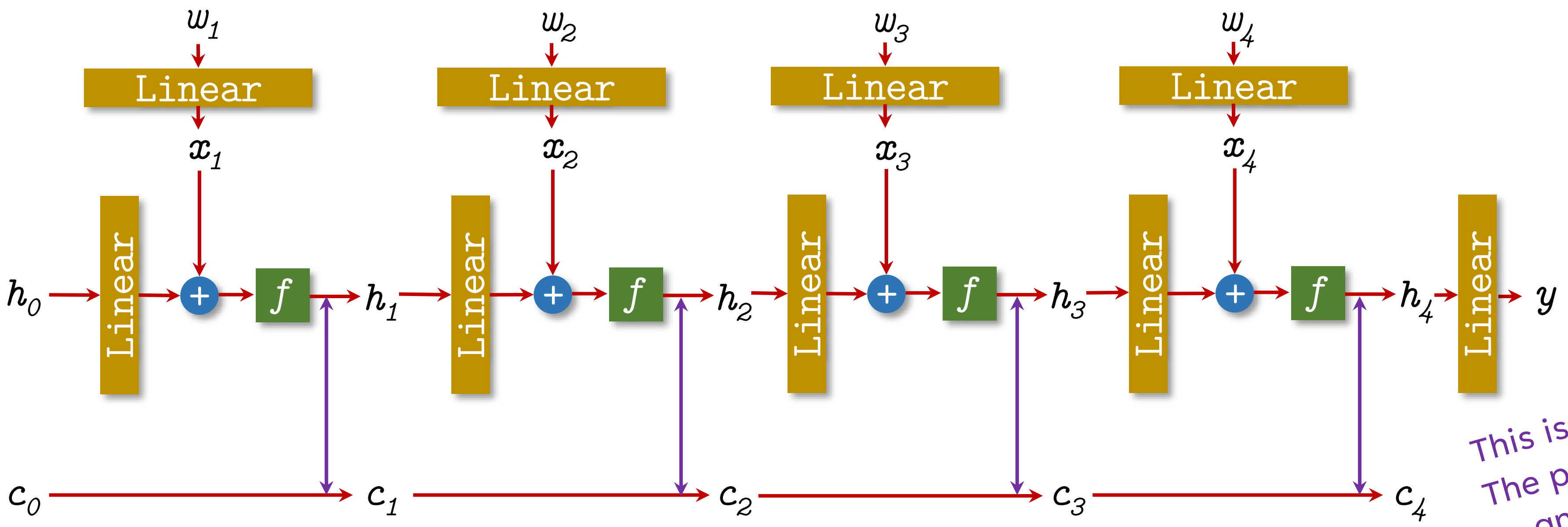
- Another solution is to change the architecture.
- Long short-term memory (LSTM; Hochreiter and Schmidhuber 1997)



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

LONG SHORT-TERM MEMORY

- 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.
- We will go into further detail next lecture.



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

QUESTIONS?