CS 490: NATURAL LANGUAGE PROCESSING

Dan Goldwasser, Abulhair Saparov

Lecture 7: Recurrent Neural Networks

MACHINE LEARNING FOR TEXT CLASSIFICATION

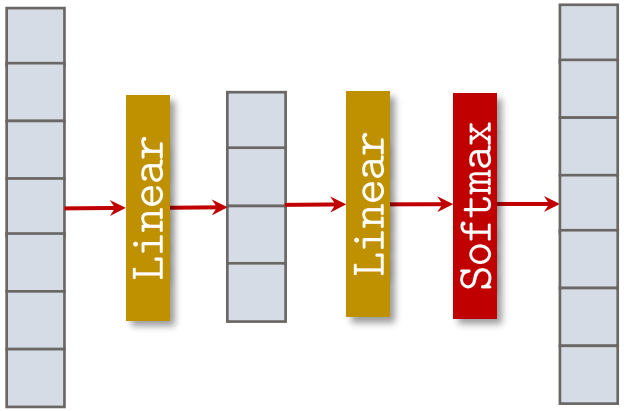
- We previously discussed NLP methods for text classification.
  - Perceptron
  - Naïve Bayes
  - Logistic regression
  - Multi-layer perceptron (MLP)
- Representation learning:
  - Instead of using simple (linear) models with engineered features,
  - We can train a complex (nonlinear) model with simple embeddings.
    - With the hope that the hidden layers in the complex model will automatically learn the complex features from data.

LEARNING REPRESENTATIONS OF WORDS

- Let's apply the ideas of representation learning that we previously discussed to learn good representations of words.
- Let's train a model to predict a word using information about the surrounding words.
- For example, pick some "window size", say 2.
- Given an input '`the cat sat on the mat`', and a target word '`sat`'.
  - We want the model to predict '`sat`' given the 2 preceding and 2 succeeding words:
    '`the cat ___ on the`'
- Use a bag-of-words embedding of the surrounding words.
- E.g., suppose our vocabulary consists of `{the, cat, sat, on, mat}`.
  - The bag-of-words embedding of the above input is:
    `[2, 1, 0, 1, 0]`
- This task is called masked language modeling. (goal: predict the masked word)

LEARNING REPRESENTATIONS OF WORDS

- Let's use the following neural network architecture:



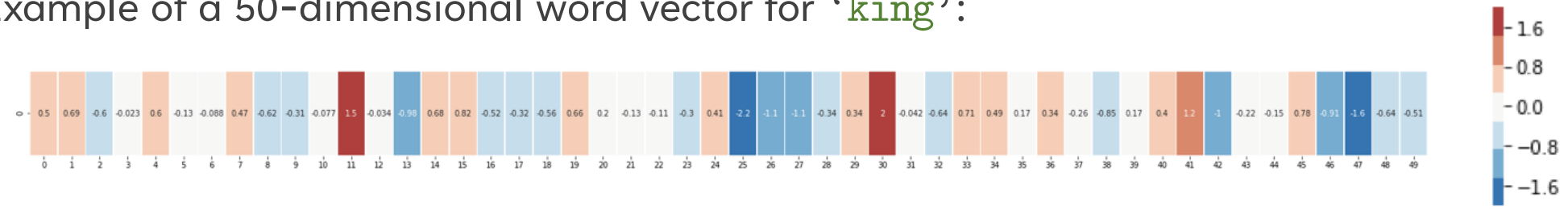
- The input is the bag-of-words encoding of the neighboring words.
- The output is the probability of the "masked" word (in the middle of the window).
- There is one hidden layer that is smaller than the input and output layers.
- This model is called the continuous bag-of-words (CBoW) model.

LEARNING REPRESENTATIONS OF WORDS

- In 2013, Mikolov et al. trained this model on data from Google News containing 1.6 billion words.
  - They used a window size of 4.
  - The vocab size is 1 million.
  - The hidden layer size (also called hidden dimension) is 600.
  - They minimized cross-entropy loss.
- After training the model, we inspect the weight matrix of the first layer.
  - What are its dimensions?
  - `h × V` (`V` is vocab size, and `h` is hidden dimension)
- The columns of this matrix provide an `h`-dimensional vector representation of every word.
  - This is called `word2vec`.
  - There are other ways to obtain vector representations of words (more generally called word vectors).

WORD VECTORS

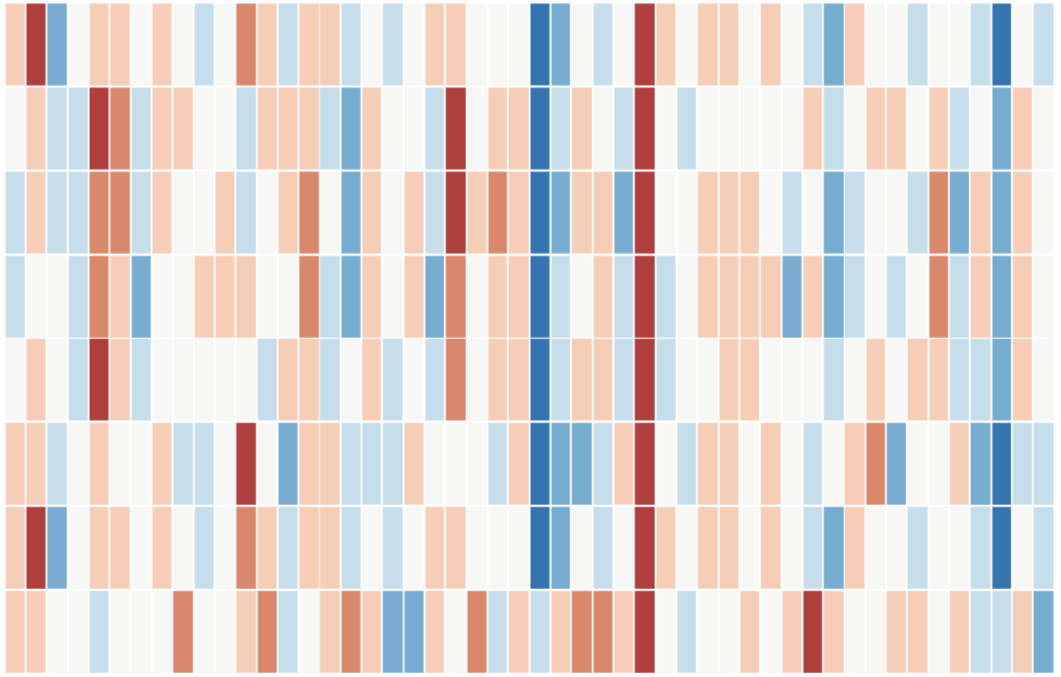
- How do we measure whether a learned representation is good?
  - I.e., How do we measure the quality of word vectors?
- What do word vectors look like?
- Example of a 50-dimensional word vector for ‘king’:



[Alammar, 2019, The Illustrated Word2vec]

WORD VECTORS

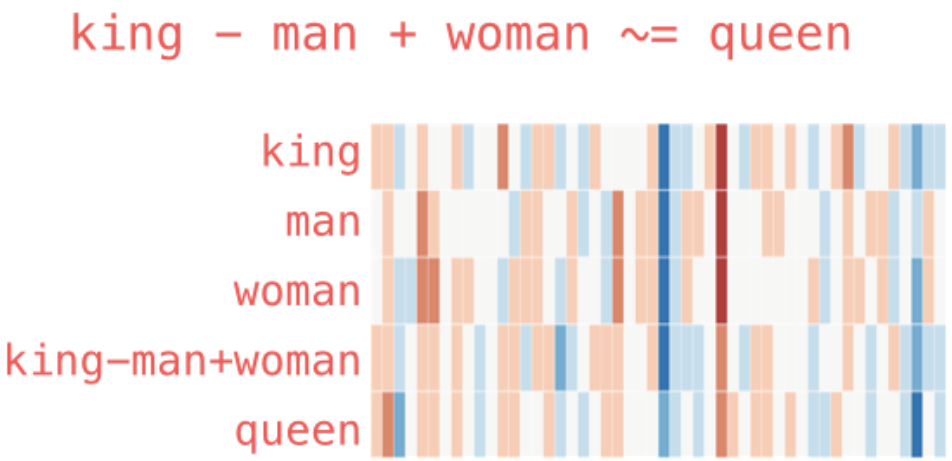
- How do we measure whether a learned representation is good?
  - I.e., How do we measure the quality of word vectors?
- What do word vectors look like?
- More examples:



[Alammar, 2019, The Illustrated Word2vec]

WORD VECTORS

- We can use word vectors to try to solve word analogy problems:
- ‘man’ is to ‘king’ as ‘woman’ is to what?
- Compute: ‘king’ – ‘man’ + ‘woman’
  - What is the word vector that is closest to the resulting vector?



[Alammar, 2019, The Illustrated Word2vec]

WORD VECTORS

- Examples of other learned relationships from `word2vec`:

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

- These are qualitative, not quantitative results, we have to interpret them with a grain of salt.
  - The authors may have cherry-picked them.
- There are some mistakes: ‘`France: tapas`’, ‘`uranium: plutonium`’, ‘`Google: Yahoo`’

[Mikolov et al. 2019]

WORD VECTORS

- Mikolov et al. (2019) did provide some quantitative results as well.

Dimensionality / Training words	24M	49M	98M	196M	391M	783M
50	13.4	15.7	18.6	19.1	22.5	23.2
100	19.4	23.1	27.8	28.7	33.4	32.2
300	23.2	29.2	35.3	38.6	43.7	45.9
600	24.0	30.1	36.5	40.8	46.6	50.4

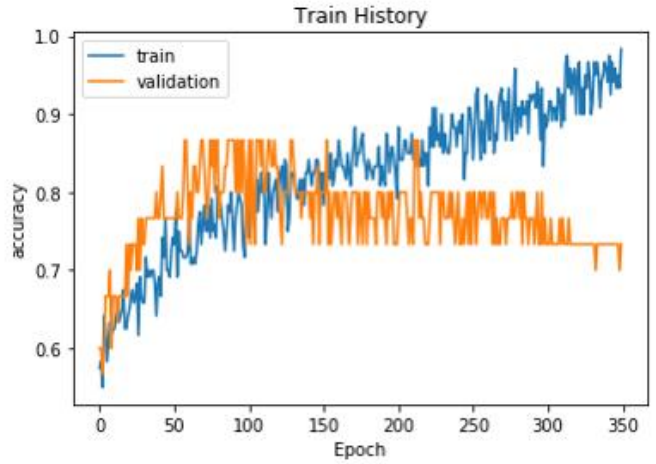
- The table shows accuracies.
- The rows correspond to different hidden layer sizes.
- The columns correspond to different training set sizes.
- Evidently, more data and larger models do better.
- But the largest model is far from perfect.
- Word vectors can still be useful! (e.g., as embeddings in a larger model)

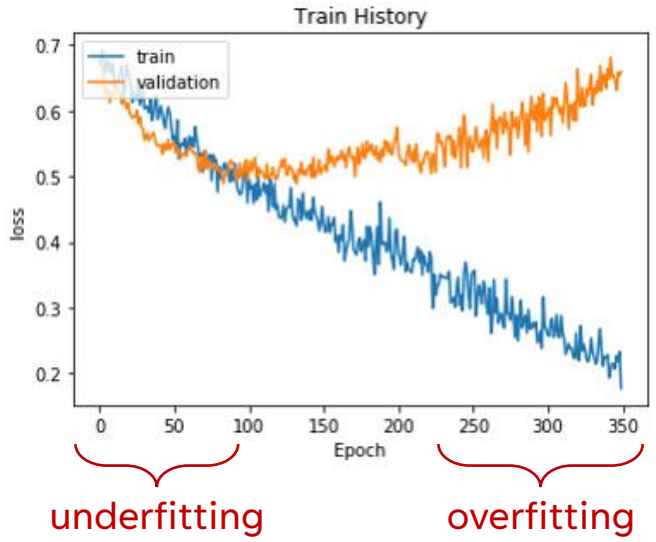
NEURAL NETWORK CAVEATS

- Neural networks such as MLPs are highly expressive.
  - They can learn non-linear decision boundaries.
- As a trade-off, they are highly prone to overfitting.
- How can we mitigate overfitting?
  - Regularization
    - L1, L2, etc.
  - More training data!
    - Data augmentation (create more data from existing data)
  - Reduce model complexity
    - E.g., reduce number of hidden neurons, number of layers, etc.

OVERFITTING MITIGATION

- Idea: Use a held-out portion of the data for validation.
  - This is called the validation dataset or validation set.
- We compute training error/accuracy and training loss on the training data.
  - We similarly compute validation error/accuracy and loss.
- Monitor training accuracy and loss for underfitting.
- Monitor validation accuracy and loss for overfitting.

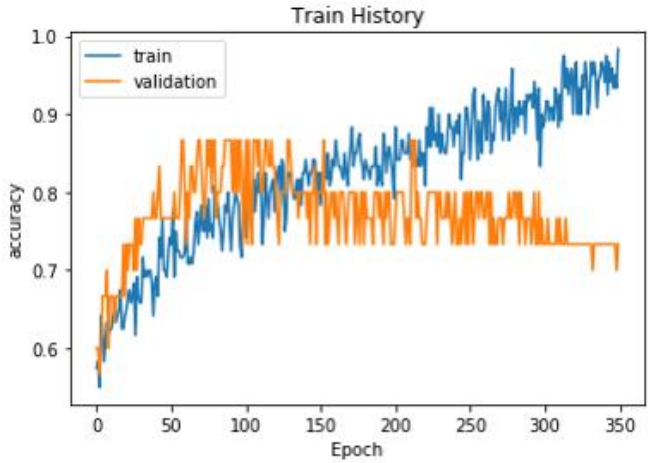


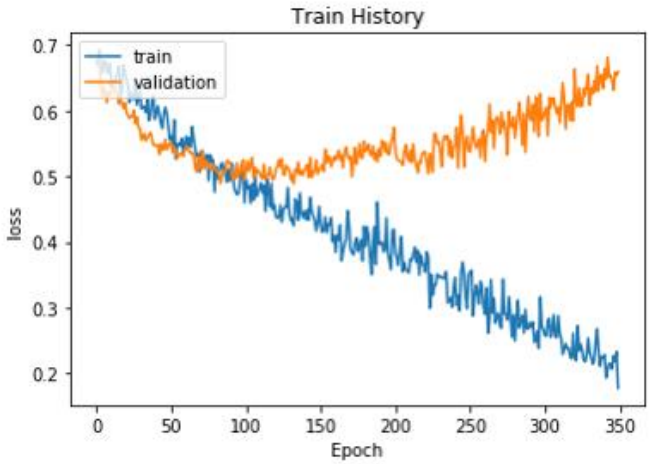


[stackoverflow.com/questions/61325681/keras-lstm-why-my-earlystopping-function-didnt-work]

OVERFITTING MITIGATION

- Cross-validation provides a lower-variance estimate of validation accuracy and loss.
- To avoid losing data to validation:
  - After determining the appropriate training duration using validation,
  - Re-run the training with the full data but only for the predetermined training duration.
- This idea is called early stopping.
  - Train until validation accuracy stops improving for some time, then stop training.





[stackoverflow.com/questions/61325681/keras-lstm-why-my-earlystopping-function-didnt-work]

WEIGHT DECAY

- At each step of gradient descent, multiply the current weights by $1 - \eta\alpha$,
  - Where α is the weight decay parameter, η is the learning rate.
- So the gradient update formula looks like:

$$w_{new} = (1 - \eta\alpha)w_{old} - \eta\nabla L$$

- Rearranging the terms:

$$w_{new} = w_{old} - \eta\alpha w_{old} - \eta\nabla L$$

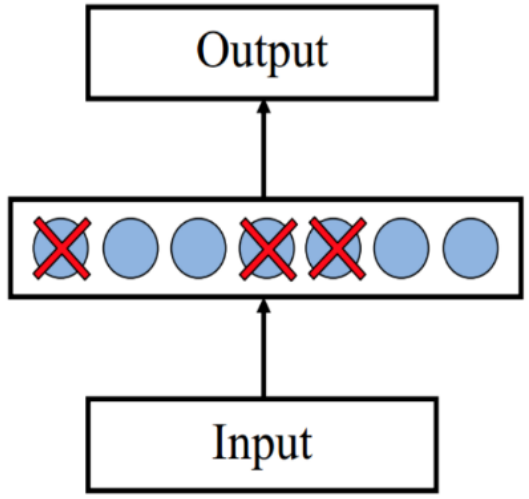
$$w_{new} = w_{old} - \eta(\nabla L + \alpha w_{old})$$

$$w_{new} = w_{old} - \eta\nabla(L + \frac{\alpha}{2}\|w_{old}\|^2)$$

- Interestingly, weight decay is equivalent to L2 regularization.
  - Which is equivalent to putting a Gaussian prior on the weights.

DROPOUT REGULARIZATION

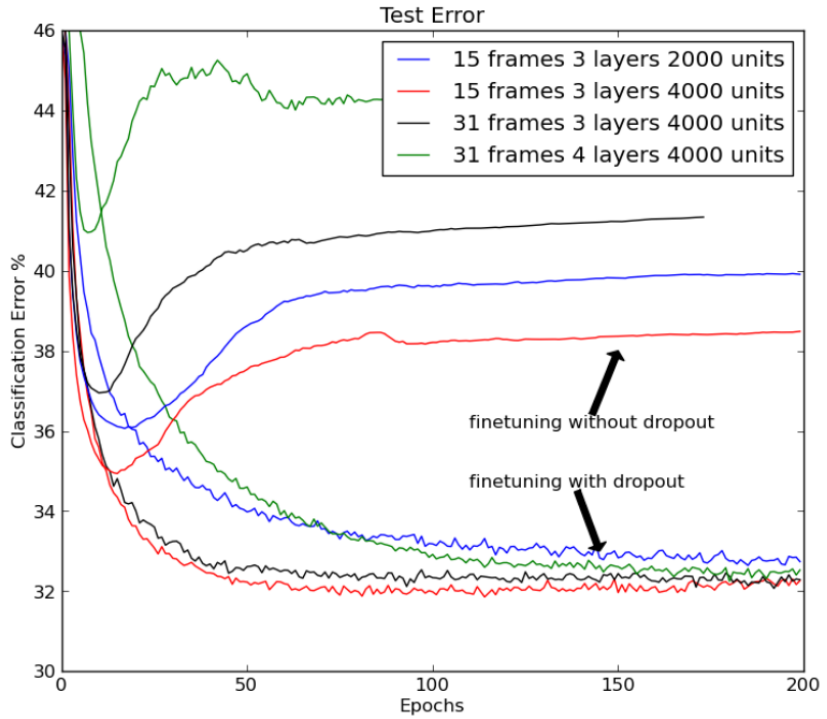
- Idea: During training, every time we perform a forward pass, randomly "dropout" some neurons with probability p.
  - p is the dropout parameter.
  - Dropping out a neuron just means to set its activation to zero.



- Proposed by Hinton et al. (2012).
- One intuition is to force the neural network to "spread out" its representation across more neurons.
  - Learn a more redundant representation.

DROPOUT REGULARIZATION

- Evaluate training a model with vs without dropout on a speech recognition task.
  - The model is trained on the TIMIT dataset.
- Here, they dropped out 50% of hidden neurons,
- And 20% of the input neurons.



[Hinton et al. 2012]

ALTERNATIVE NEURAL ARCHITECTURES

- In addition to MLPs, there is a 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?

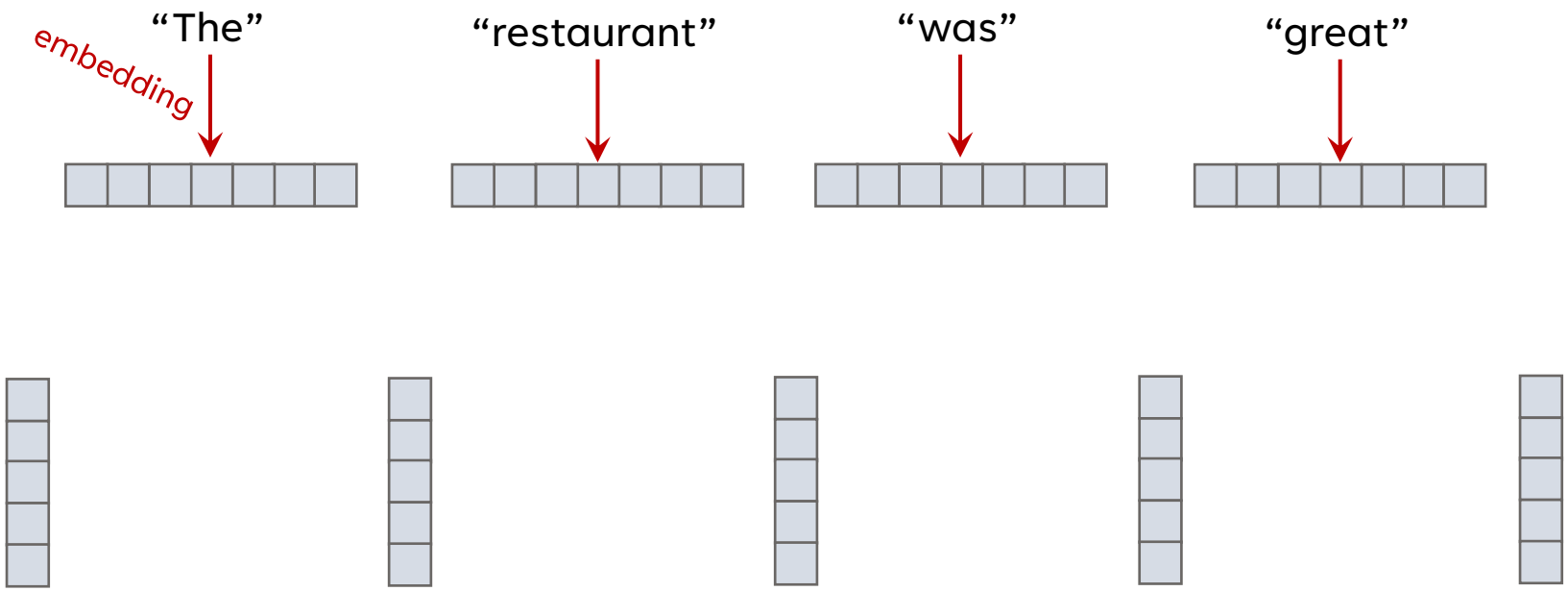
RECURRENT NEURAL NETWORKS

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

"The" "restaurant" "was" "great"

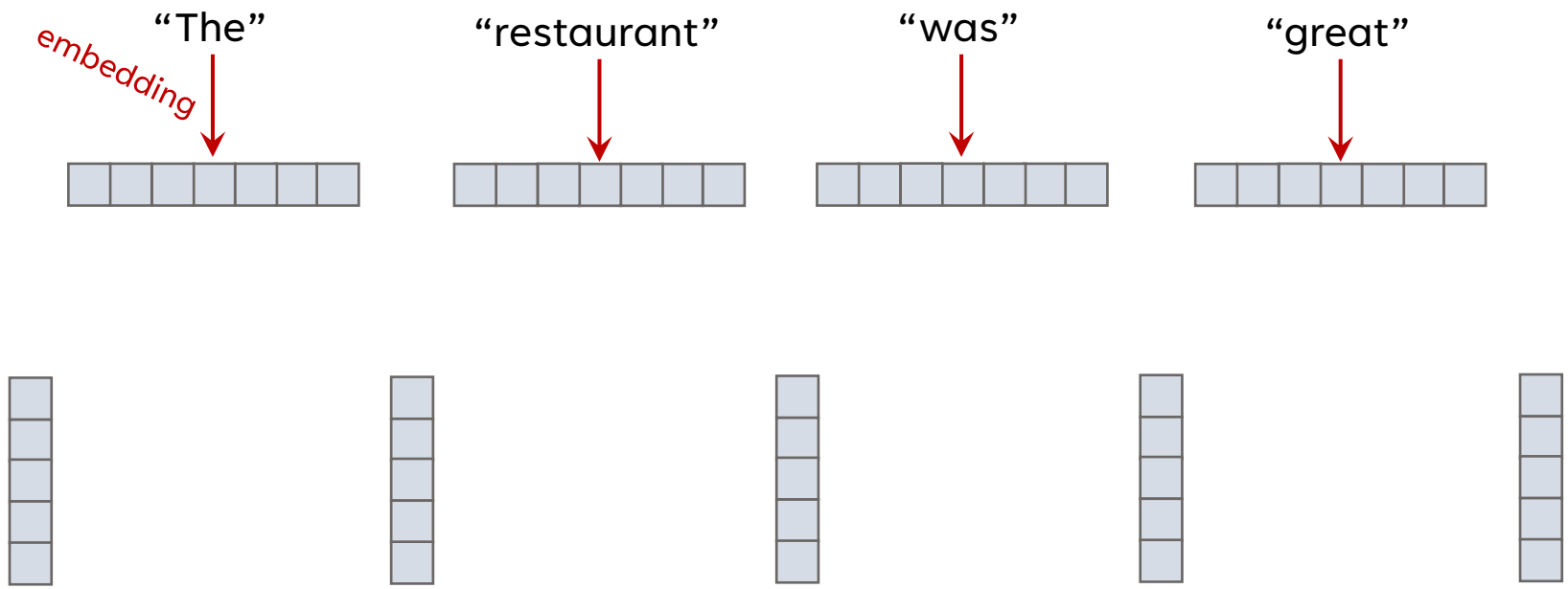
RECURRENT NEURAL NETWORKS

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



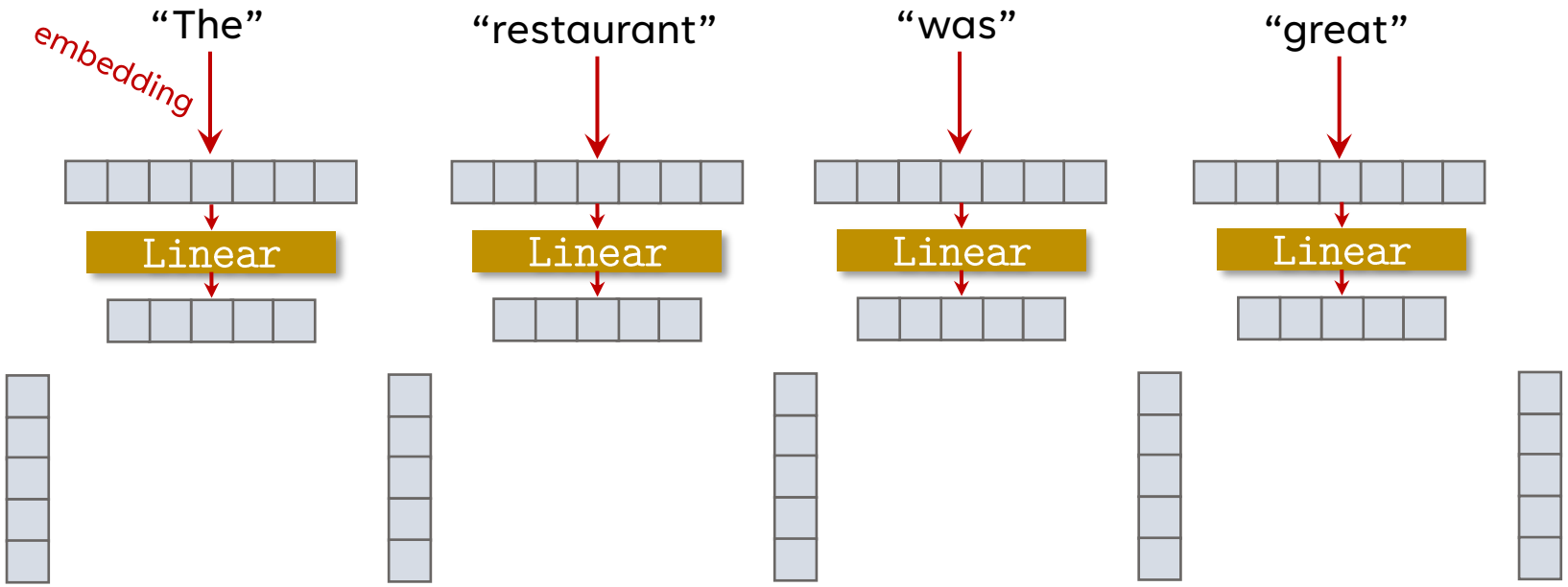
RECURRENT NEURAL NETWORKS

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



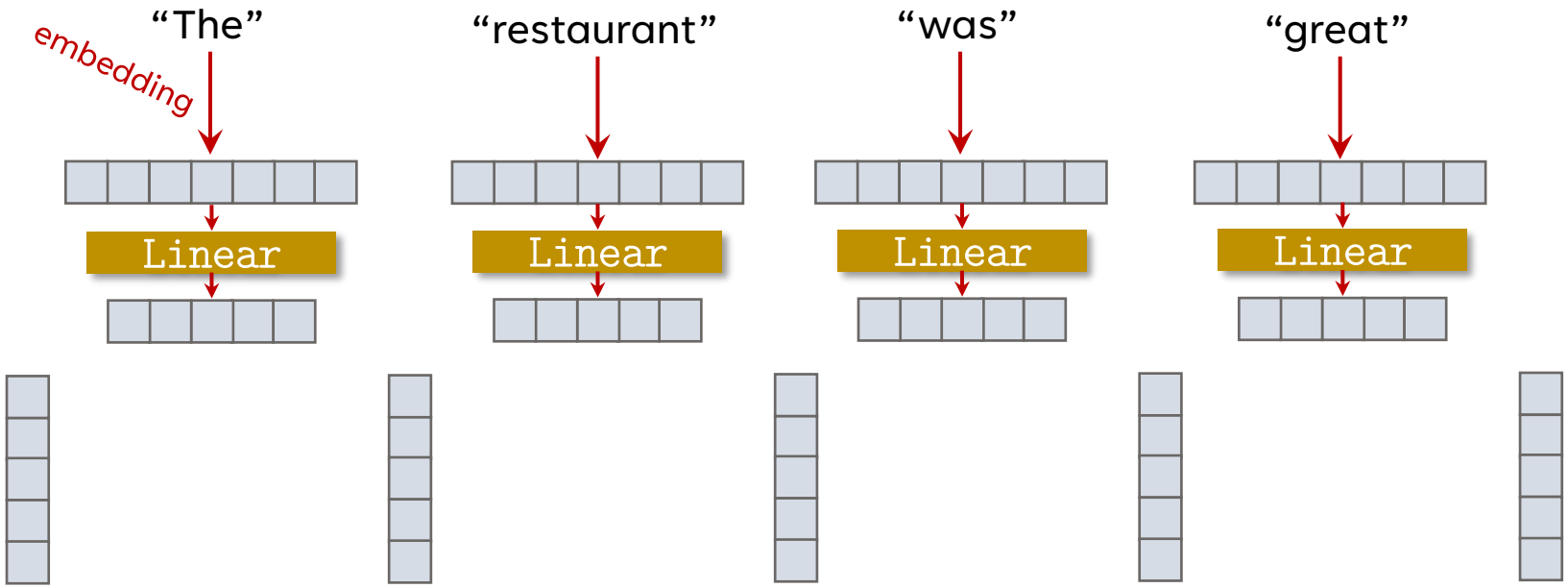
RECURRENT NEURAL NETWORKS

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



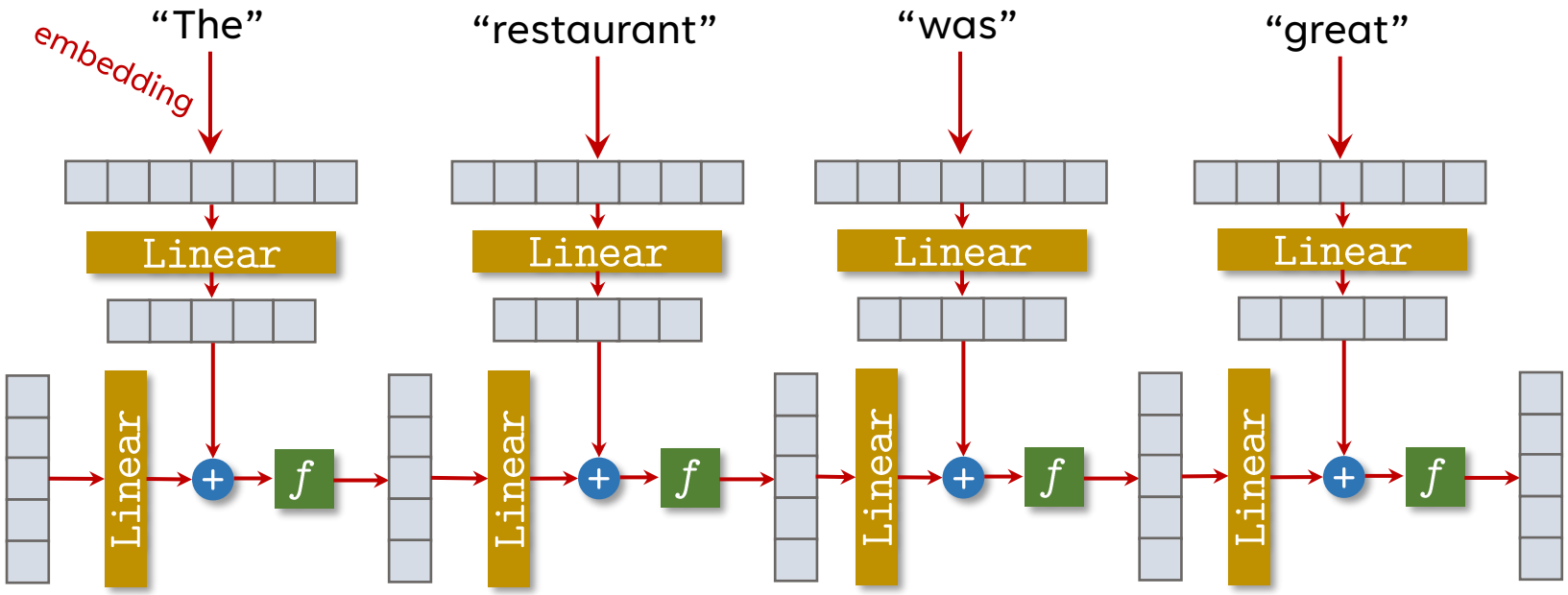
RECURRENT NEURAL NETWORKS

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



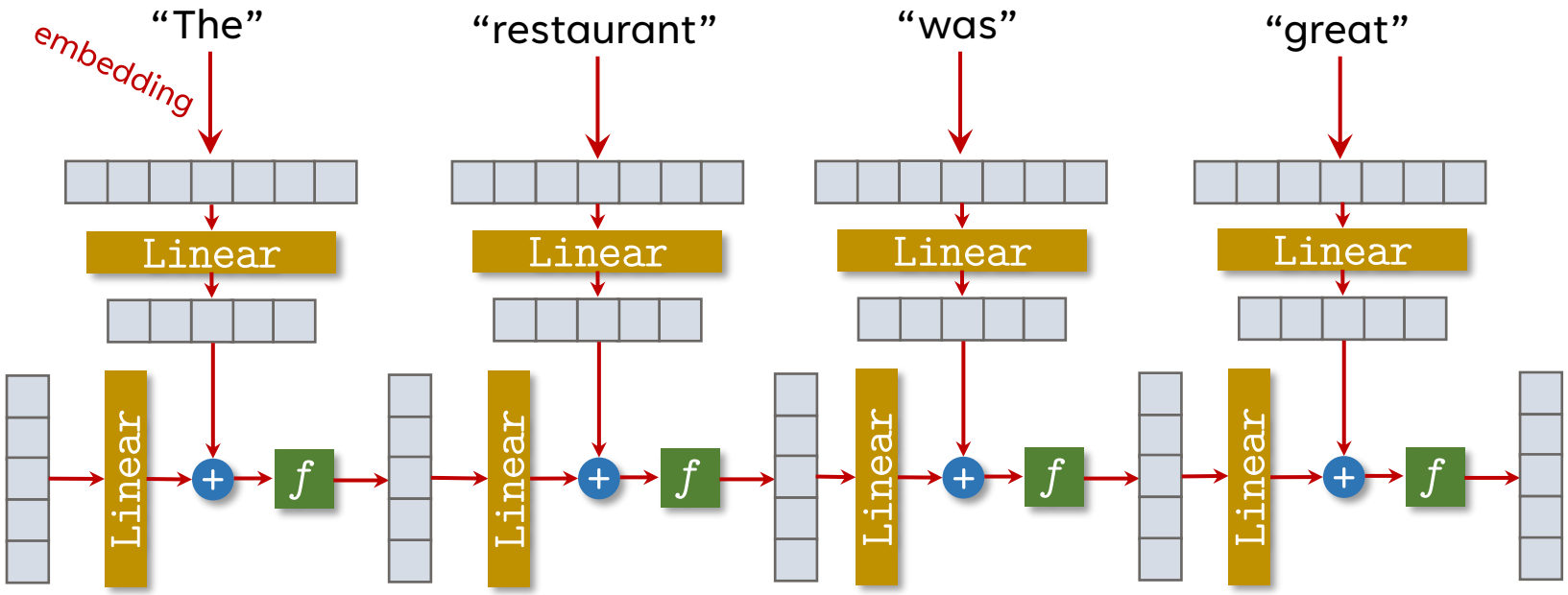
RECURRENT NEURAL NETWORKS

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



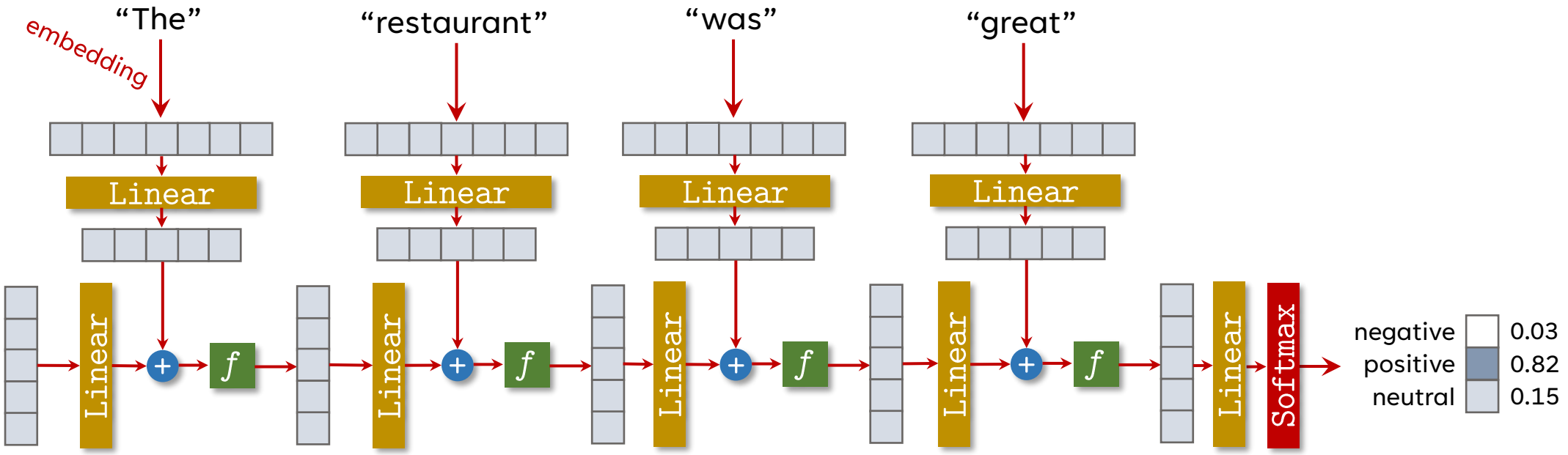
RECURRENT NEURAL NETWORKS

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



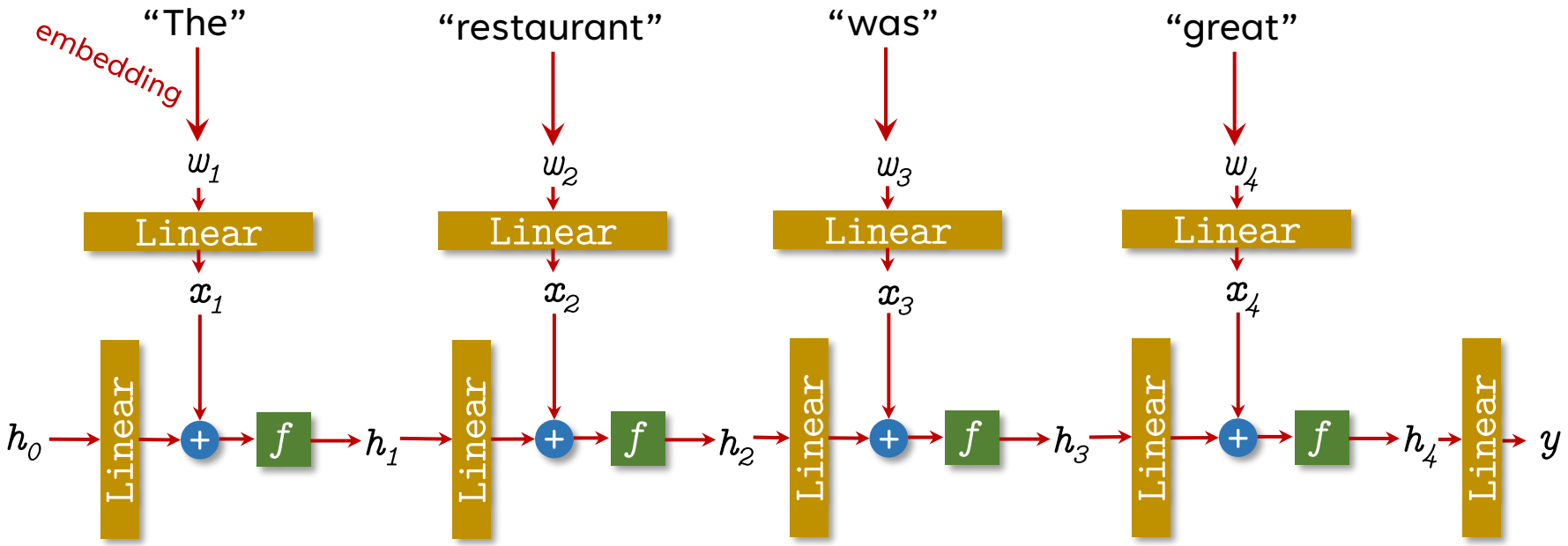
RECURRENT NEURAL NETWORKS

- Once we have the last hidden state, we can use it to make a prediction.
- In the example, we have a sentiment analysis task for a restaurant review.



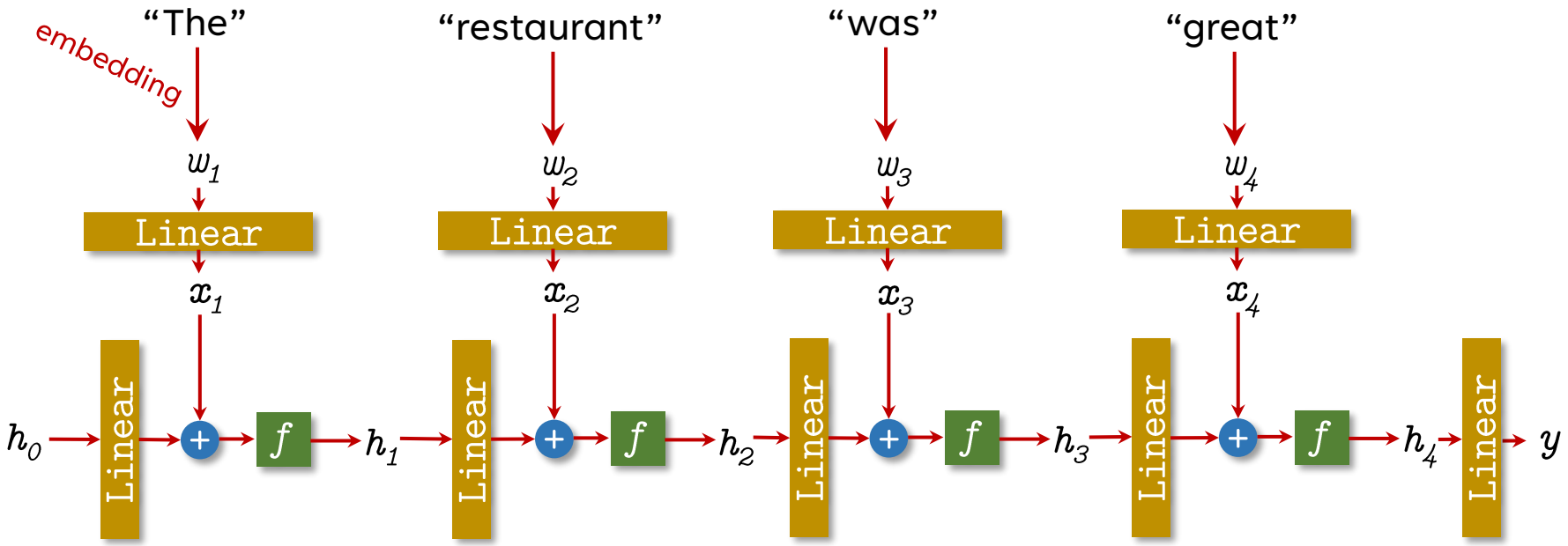
RECURRENT NEURAL NETWORKS

- 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.



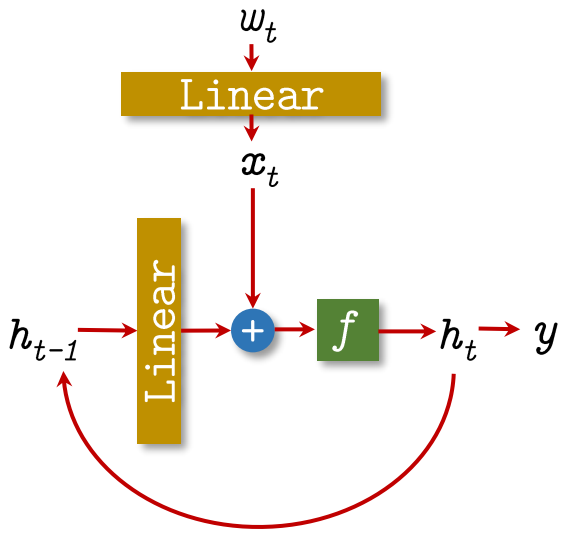
RECURRENT NEURAL NETWORKS

- Why “recurrent”?



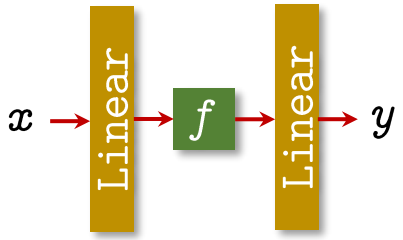
RECURRENT NEURAL NETWORKS

- 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)

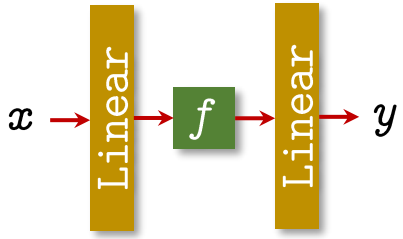


$$y = b_2 + W_2 \cdot f(b_1 + W_1 x)$$

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.

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



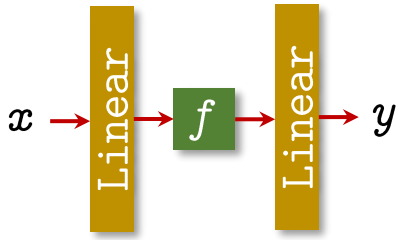
$$y = b_2 + W_2 \cdot f(b_1 + W_1 x)$$

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

[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



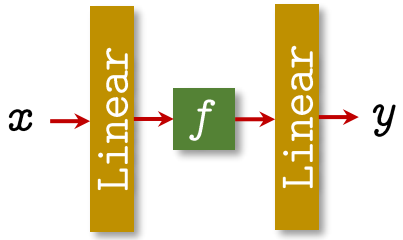
$$y = b_1 + W_1 \cdot f(b_1 + W_1 x)$$

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

[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.



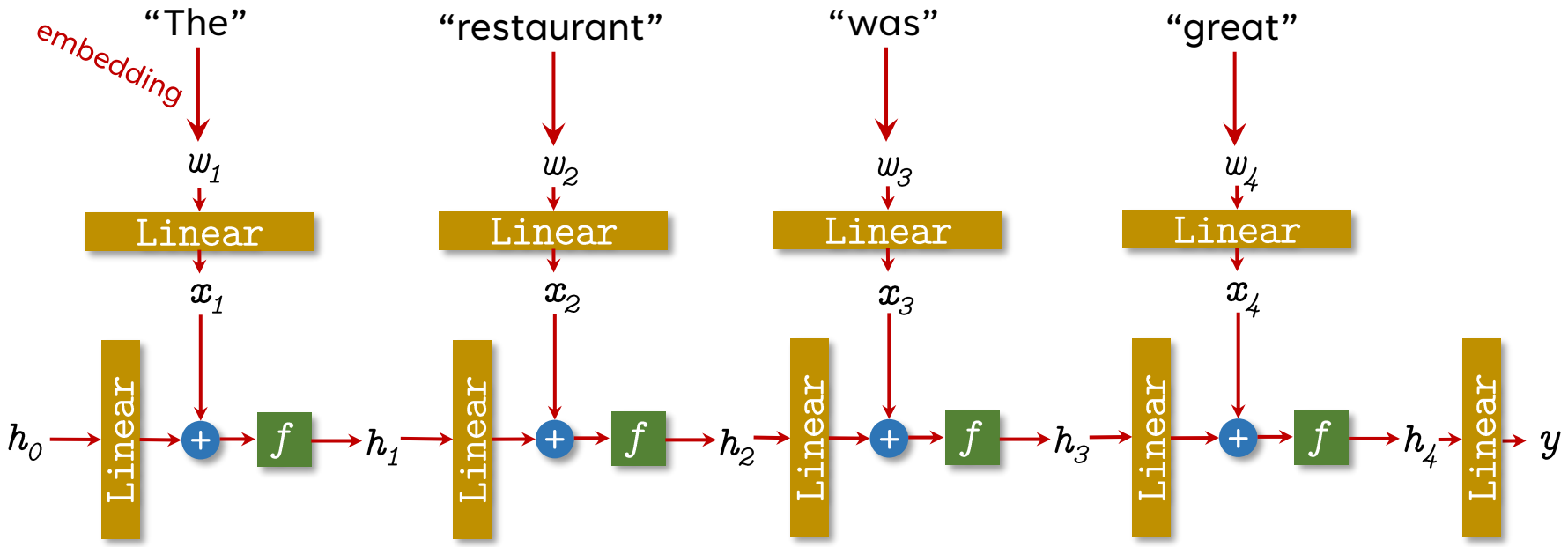
$$y = b_1 + W_1 \cdot f(b_1 + W_1 x)$$

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

[en.wikipedia.org/wiki/Matrix_calculus]

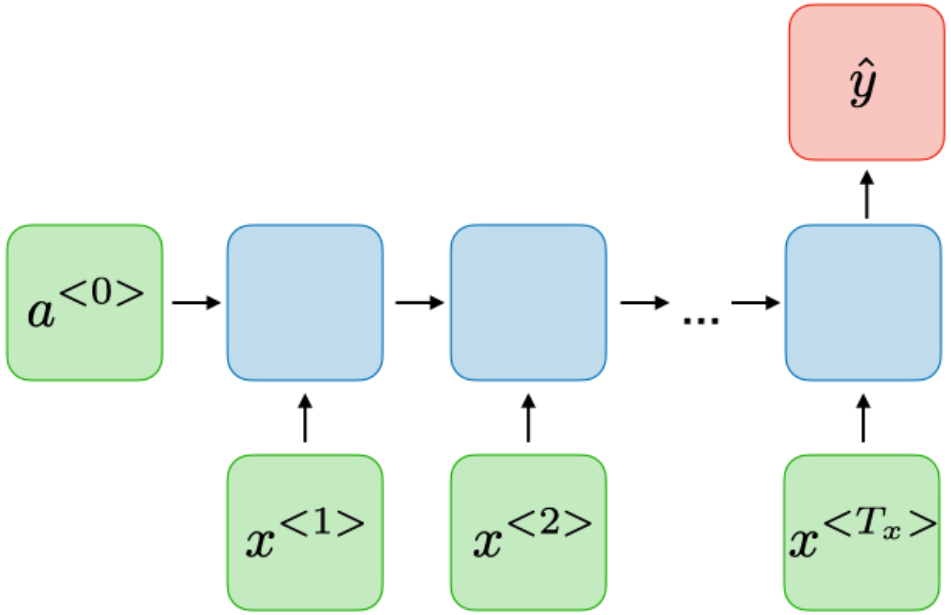
RNN APPLICATIONS

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



RNN APPLICATIONS

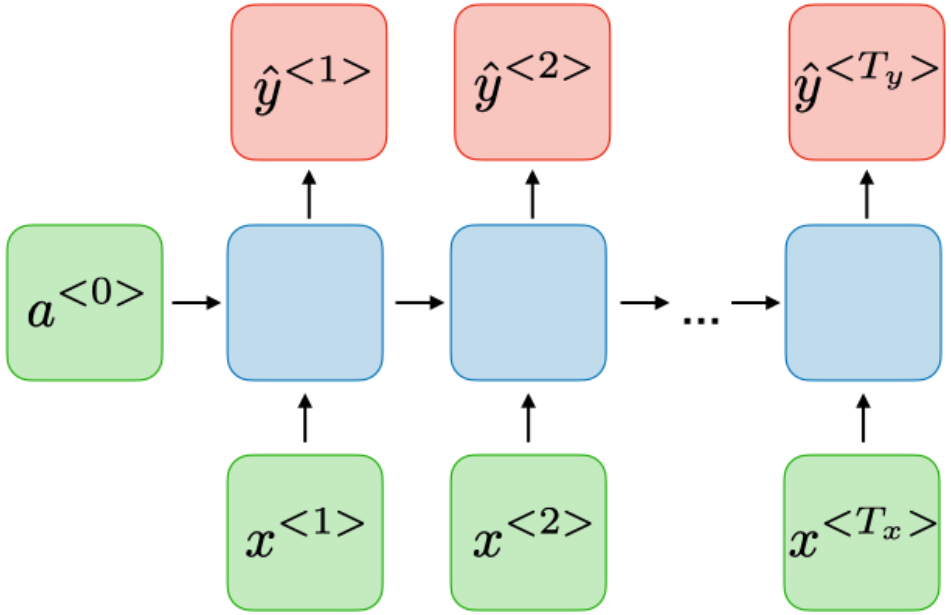
- RNNs have a very wide variety of applications.
- Beyond simple text classification.



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

RNN APPLICATIONS

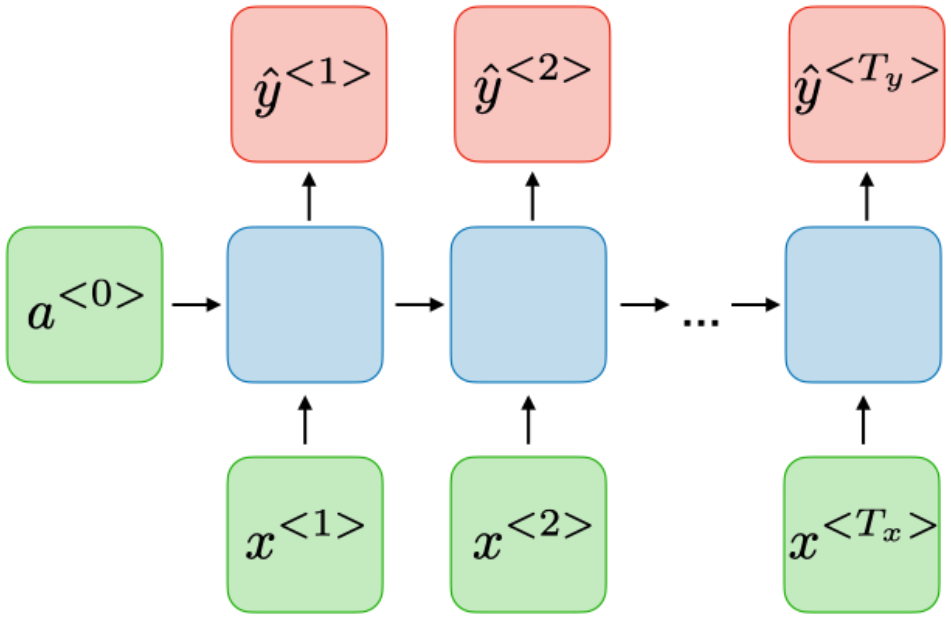
- We can make more than one prediction.
- For example, we can make a prediction per input word.



[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.

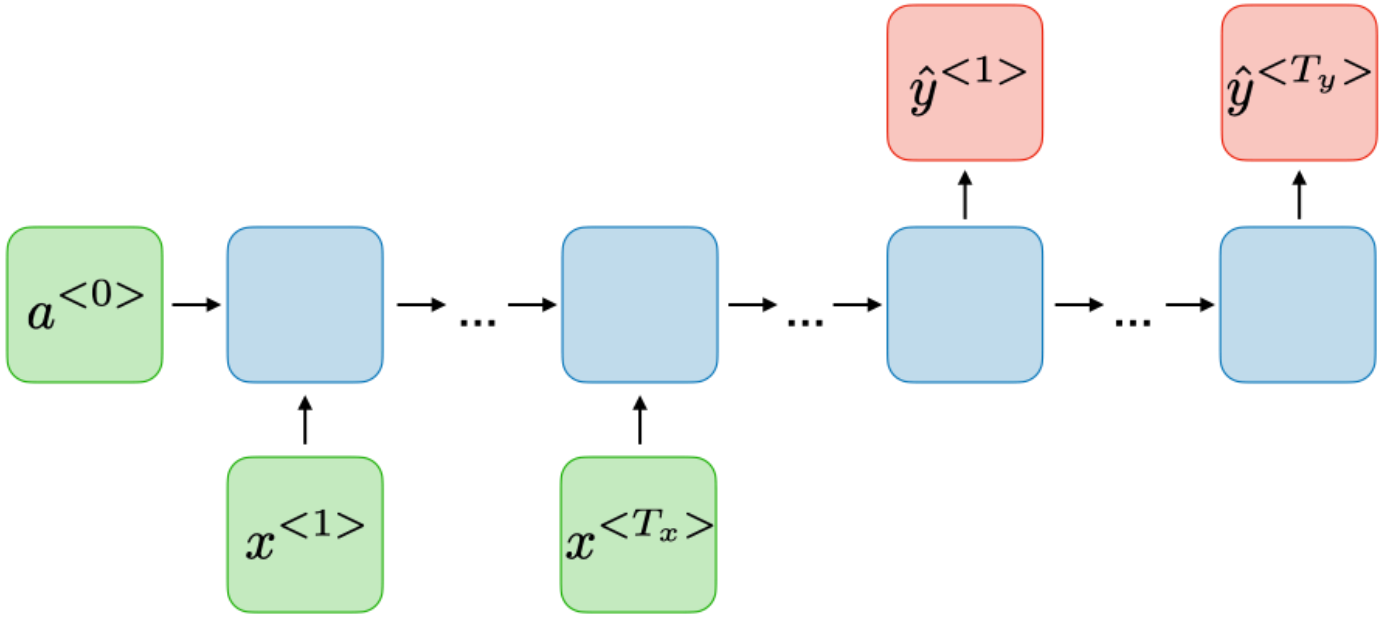
Example prediction:
Input: The quick brown fox jumped.
Prediction: DET ADJ NN NN V

$$Total\ loss = L(\text{DET},\text{DET}) + L(\text{ADJ},\text{ADJ}) + L(\text{ADJ},\text{NN}) + L(\text{NN},\text{NN}) + L(\text{V},\text{V})$$

[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.

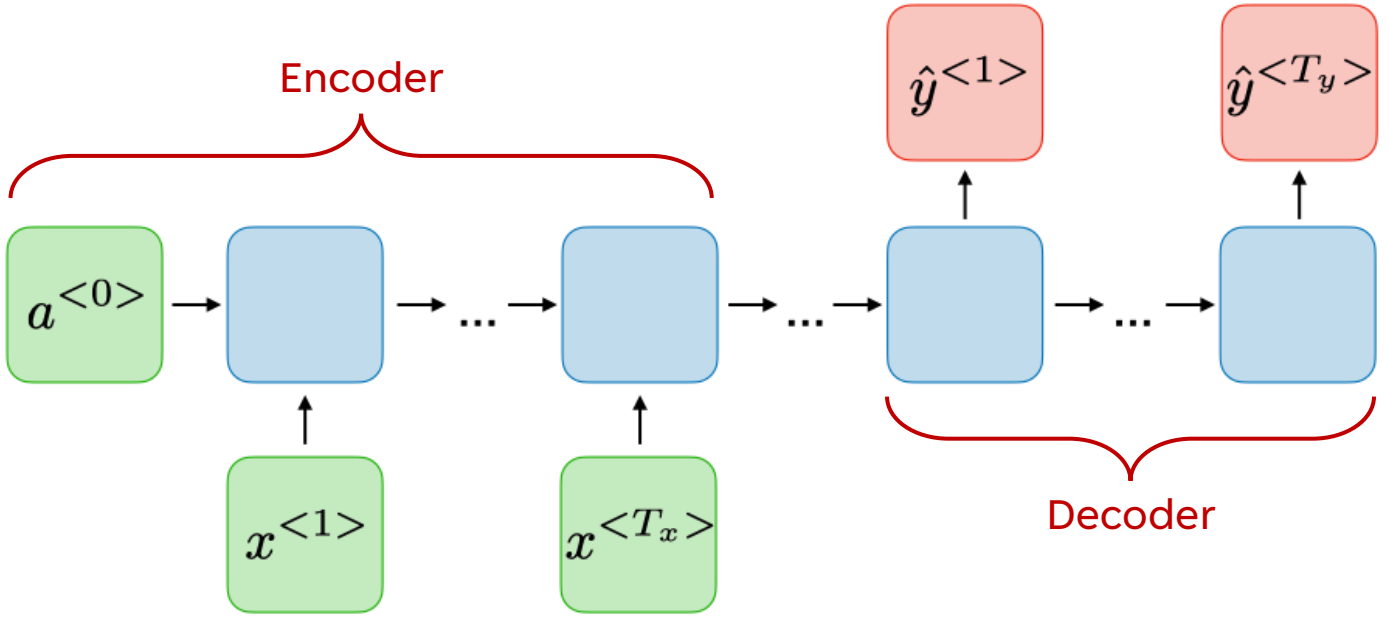


[Afshine Amidi and Shervine Amidi, Recurrent Neural Networks cheatsheet]

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。



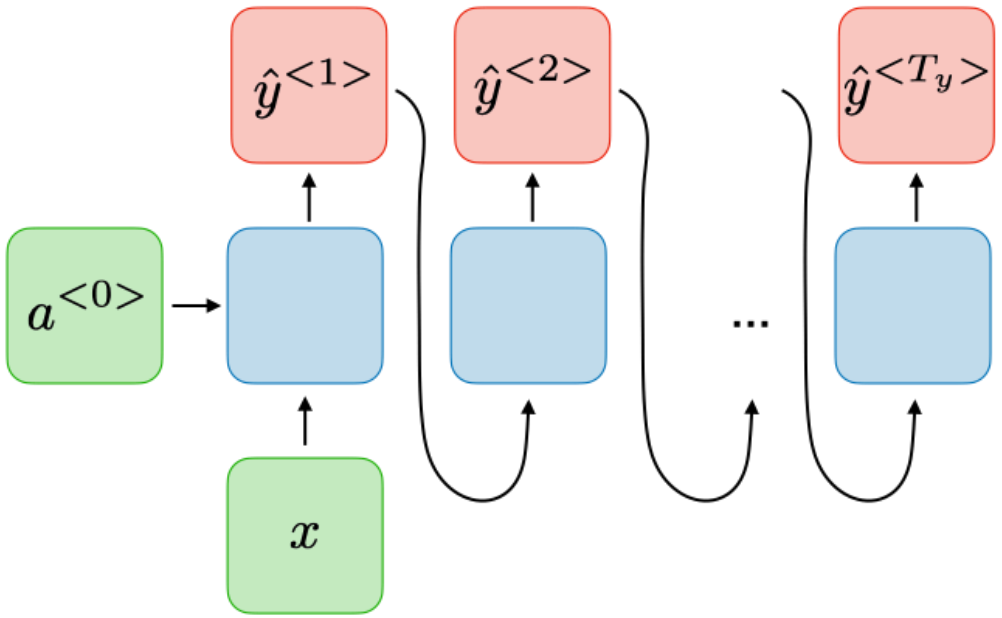
[Afshine Amidi and Shervine Amidi, Recurrent Neural Networks cheatsheet]

RNN APPLICATIONS

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

Input: birthday

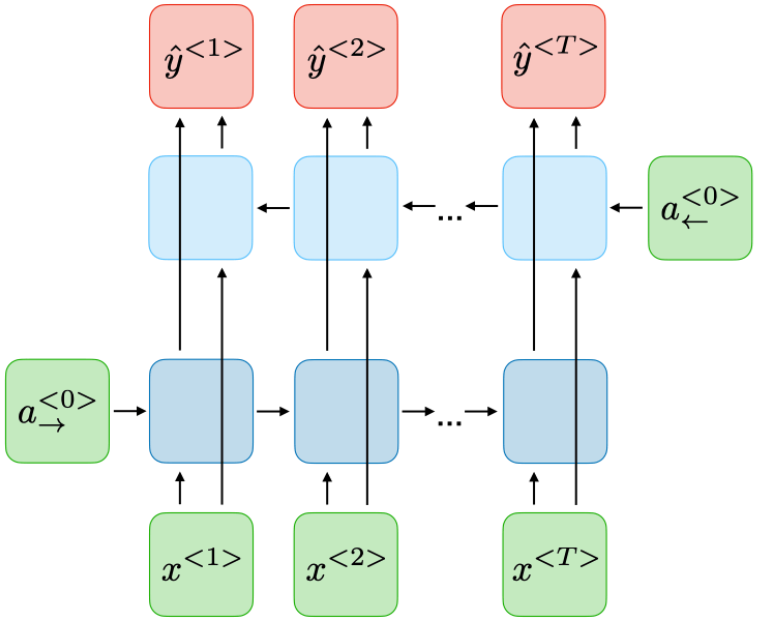
Output:



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

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.

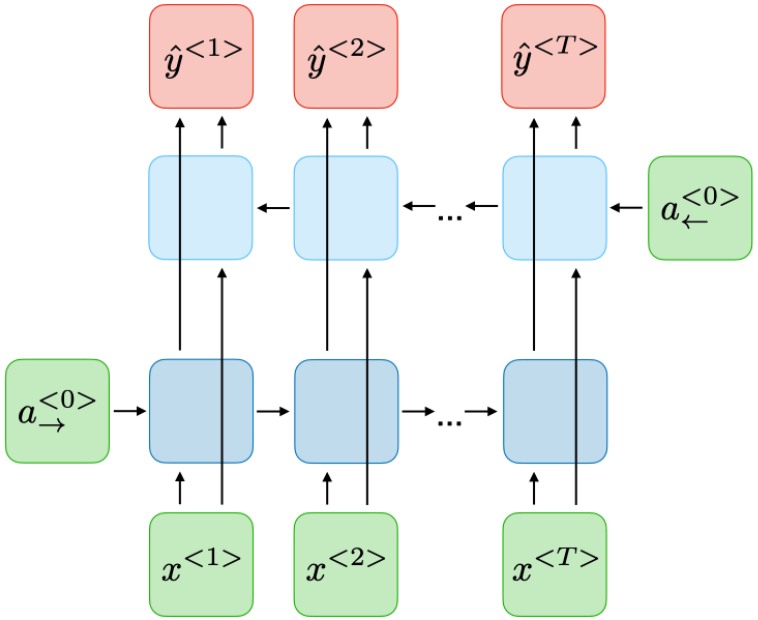


[Afshine Amidi and Shervine Amidi, Recurrent Neural Networks cheatsheet]

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

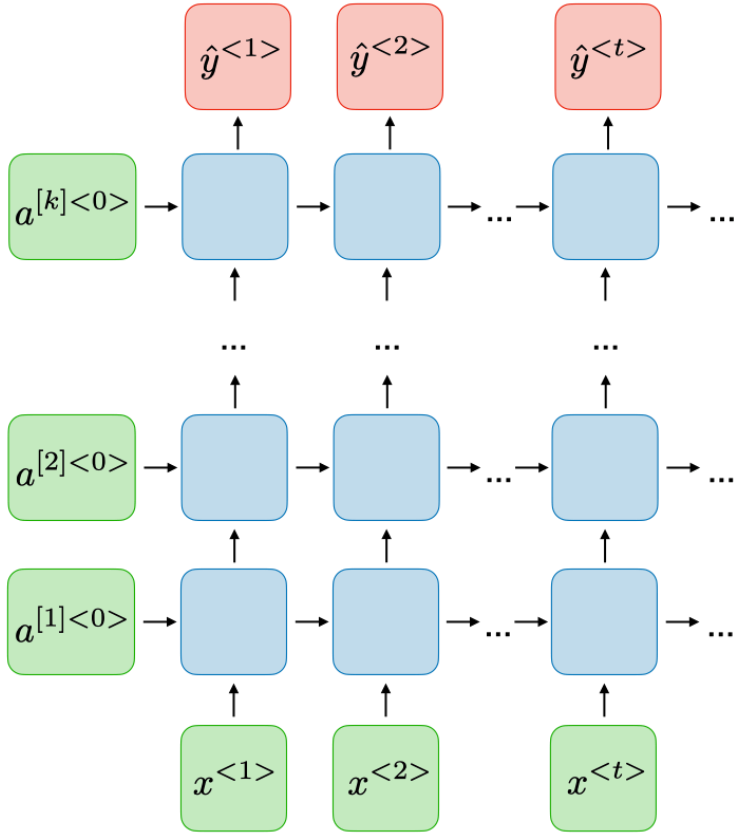
Input: I am ___ hungry.
Output: so

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).

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

- 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?

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

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

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

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}.

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

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

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

VANISHING/EXPLODING GRADIENTS

- Another solution is to change the architecture.
- There are more sophisticated versions of RNNs that attempt to solve this.
  - E.g., long short-term memory (LSTM; Hochreiter and Schmidhuber 1997)
- Attention and transformers (Vaswani et al. 2017) also attempt to solve this.
  - We will cover these methods in the next few lectures.

QUESTIONS?