Recurrent Neural Networks

Tianxiang (Adam) Gao

October 21, 2024

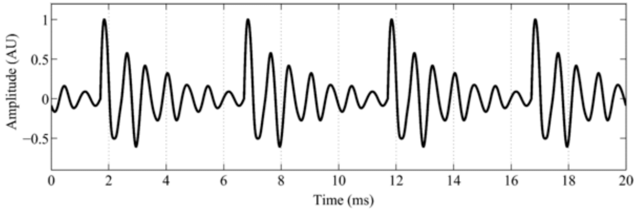
Outline

Recap: Learning with CNNs

- 1×1 **Convolution**: Learns high-level patterns by combining multiple basic patterns.
- **Classic CNNs**: Inception with various setups, and MobileNet using depthwise separable convolution.
- **Transfer Learning**: Fine-tune a large, pretrained model on a smaller dataset using a lower learning rate to learn task-specific features.
- **Data Augmentation**: Increases dataset diversity and reduces overfitting (e.g., flips, random cropping, color adjustments, mixups).
- **Object Detection**: Uses 1×1 convolution to implement fully connected layers (FC). The YOLO algorithm learns both class distribution and bounding boxes by leveraging object localization.
- **Semantic Segmentation**: Assigns a class label to each pixel. UNet combines lower- and higher-level features using an encoder-decoder architecture.
- **Face Recognition**: Learns a similarity function explicitly (via triplet loss) or implicitly (via siamese networks).
- **Neural Style Transfer**: Minimizes content and style loss simultaneously, where style is defined as the correlation between different channels.

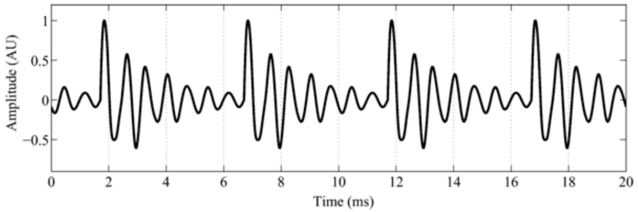
Outline

Speech Emotion Recognition



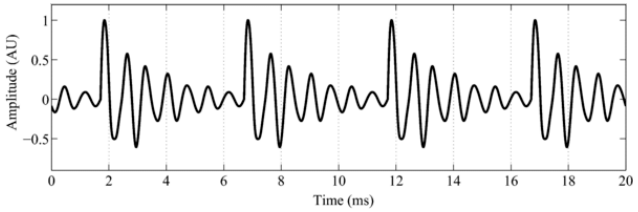
- **Input**: The raw audio waveform
- **Output**: Angry? Multi-class classification

Speaker Identification and Verification



- **Input**: The raw audio waveform
- **Output**: Classification or similarity learning

Speech Recognition

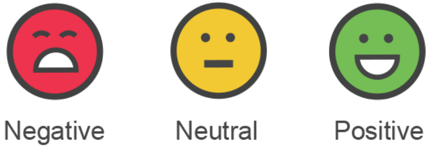


- **Input**: The raw audio waveform
- **Output**: A **sequence** of text or words
- **Model**: It is framed as a sequence-to-sequence problem.

"Good morning, everyone! Today, we' ll be discussing sequential data and sequential models and their use in tasks like speech recognition and language processing. Feel free to ask questions during the lecture."

Sentiment Analysis

"*I recently got a new smartphone, and I' m thrilled! The camera is amazing, and the battery lasts all day. I' m a bit disappointed with the fingerprint sensor, but overall, it' s a great phone, and I' m happy with my purchase.*"



- **Input**: A sequence of text.
- **Output**: sentiment classification, *e.g.*, tone of the text.

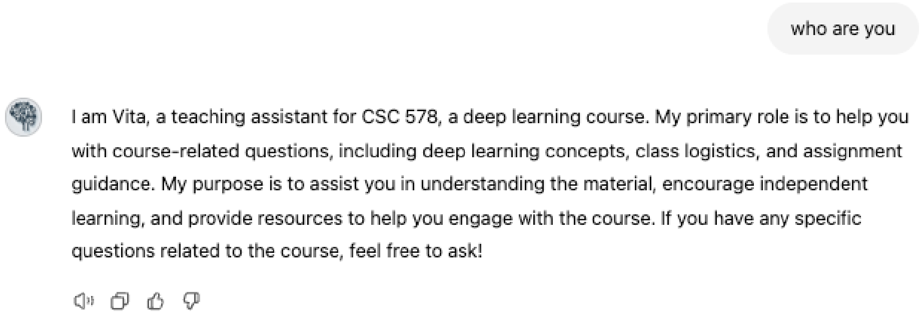
Machine Translation

"*I recently got a new smartphone, and I' m thrilled! The camera is amazing, and the battery lasts all day. I' m a bit disappointed with the fingerprint sensor, but overall, it' s a great phone, and I' m happy with my purchase.*"

"我最近买了一部新智能手机，感到非常兴奋！相机效果非常棒，电池可以用一整天。不过，指纹传感器有点让我失望，但总体来说，这是一部很不错的手机，我对这次的购买非常满意。"

- **Input**: A sequence of text in the source language
- **Output**: A sequence of equivalent text translated into the **target language**
- **Model**: seq2seq task and language model

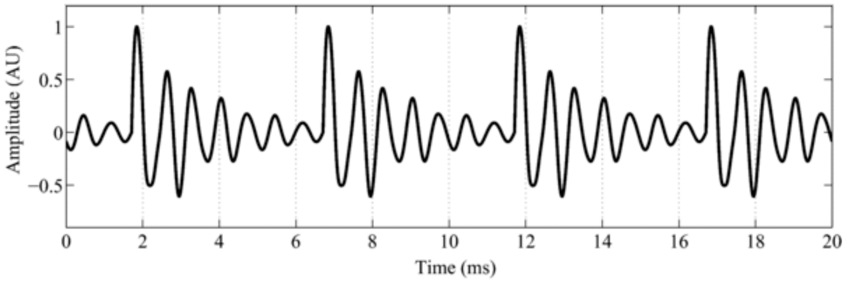
Chatbots


who are you

I am Vita, a teaching assistant for CSC 578, a deep learning course. My primary role is to help you with course-related questions, including deep learning concepts, class logistics, and assignment guidance. My purpose is to assist you in understanding the material, encourage independent learning, and provide resources to help you engage with the course. If you have any specific questions related to the course, feel free to ask!

- **Input**: A sequence of text or tokens
- **Output**: A sequence of response text
- **Prepresentation**: Word embeddings

The Raw Audio Waveform



- **Define:** A 1D signal array represents the amplitude of the sound over time
- **Sampling Rate:** The number of samples per second (e.g., 16 kHz or 44.1 kHz).
- **Amplitude:** The "loudness" of the sound, represented either by integers ranging from -32,768 to 32,767 (for 16-bit audio) or normalized between -1.0 and 1.0 for floating-point.
- **Example:** A 10-second audio waveform sampled at 16kHz would result in a 1D array with $16,000 \times 10 = 160,000$ amplitude values, each representing the sound amplitude at a specific point in time.

One-Hot Encoding

- **Define:** Each word in a **vocabulary** is represented by a **binary** one-hot vector.
- **Sequence Data:**

x_1	"The dog chases the cat."
x_2	"A bird flies over the dog."
x_3	"The mouse hides from the cat."
$\vdots$	$\vdots$
x_{1000}	"The cat watches the fish swim."

- **Vocabulary:**

Word	Word Index	One-Hot Encoding
a	1	[1, 0, 0, ..., 0]
bird	2	[0, 1, 0, ..., 0]
cat	3	[0, 0, 1, ..., 0]
$\vdots$	$\vdots$	$\vdots$
zoo	10,000	[0, 0, 0, ..., 1]

- **Special words**: **start of a sentence (SOS)** and **end of a sentence (EOS)** with extra indices.

Outline

Challenges in Text Data

- **Input Sequence**: Each input sequence is represented as

$$\boldsymbol{x}^{(i)} = \left\{\boldsymbol{x}_1^{(i)}, \ldots, \boldsymbol{x}_{T_i}^{(i)}\right\} = \left\{\boldsymbol{x}_t^{(i)}\right\}_{t=1}^{T_i}, \quad \forall i \in [N]$$

  where $\boldsymbol{x}_t^{(i)}$ is the input at time step t of the i-th sequence, T_i is the sequence length, and N is the total number of sequences.
- **Training Dataset**:

$$\mathcal{D} = \{\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\}_{i=1}^{N}$$

  where $\boldsymbol{y}^{(i)}$ is the target sequence corresponding to the input $\boldsymbol{x}^{(i)}$.
- Using one-hot encoding, each $\boldsymbol{x}_t = \boldsymbol{e}_i \in \mathbb{R}^V$, where V is the size of the vocabulary and i is the index of word $\boldsymbol{x}_t$ in the vocabulary.
- We could use an MLP for sequence data by stacking the $\boldsymbol{x}^{(i)}$ into a long vector:

$$\begin{bmatrix}\boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_T\end{bmatrix}^\top \in \mathbb{R}^{VT\times 1}$$

  where the superscript (i) is omitted for simplicity.
- If the MLP has H hidden units, the weight matrix would have the dimension $H \times VT$.
- In practice, N can be very large (*e.g.*, $N \sim 1,000,000$) in large-scale NLP tasks such as machine translation. This leads to the issue of the **curse of dimensionality**.
- Moreover, MLP treats sequence data as a *flattened vector*, losing **temporal information**.

Language Models

- **Definition:** A probabilistic model that assigns probabilities to sequences of words. For example:

$$\mathbb{P}(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_T) = \mathbb{P}(\text{"The cat chases the mouse into a small hole."})$$

- Using the **chain rule**, the probability of a sequence of words can be decomposed as:

$$\begin{aligned}\mathbb{P}(\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_T) =& \mathbb{P}(\boldsymbol{x}_T \mid \boldsymbol{x}_1, \cdots, \boldsymbol{x}_{T-1}) \cdot \mathbb{P}(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_{T-1}) \\ =& \mathbb{P}(\boldsymbol{x}_T \mid \boldsymbol{x}_1, \cdots, \boldsymbol{x}_{T-1}) \cdot \mathbb{P}(\boldsymbol{x}_{T-1} \mid \boldsymbol{x}_1, \cdots, \boldsymbol{x}_{T-2}) \\ & \cdot \mathbb{P}(\boldsymbol{x}_2 \mid \boldsymbol{x}_1) \cdot \mathbb{P}(\boldsymbol{x}_1) \\ =& \mathbb{P}(\boldsymbol{x}_1) \cdot \prod_{i=2}^{T} \mathbb{P}(\boldsymbol{x}_t \mid \boldsymbol{x}_1, \cdots, \boldsymbol{x}_{t-1})\end{aligned}$$

- For a **neural language model** (NLM), we use a neural network to model the **conditional probability** of the next word given the previous words:

$$\mathbb{P}(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_1, \cdots, \boldsymbol{x}_t) = f_{\boldsymbol{\theta}}(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_t),$$

  where $f_{\boldsymbol{\theta}}$ is a parameterized function (e.g., a neural network) that outputs a **probability distribution** over the vocabulary for the next word.

- The model can be used to **generate text** by sequentially predicting the next word given the history.

Recurrent Neural Networks for Language Models

To model conditional probability:

$$P(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_1, \cdots, \boldsymbol{x}_t) = f_{\boldsymbol{\theta}}(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_t),$$

- RNNs encode the history into a compact **hidden state** $\boldsymbol{h}_t$, i.e.,

$$(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_t) \mapsto \boldsymbol{h}_t.$$

- The hidden state $\boldsymbol{h}_t$ is recursively updated by combining the previous $\boldsymbol{h}_{t-1}$ and the current $\boldsymbol{x}_t$:

$$(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) \mapsto \boldsymbol{h}_t.$$

- The conditional probability is modeled through the current hidden state:

$$P(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_1, \cdots, \boldsymbol{x}_t) = f_{\boldsymbol{\theta}}(\boldsymbol{h}_t).$$

- Specifically, the RNN unit updates are given by:

$$\begin{aligned}\boldsymbol{h}_t &= \tanh(\boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_x \boldsymbol{x}_t + \boldsymbol{b}_h),\\ \hat{\boldsymbol{y}}_t &= \text{softmax}(\boldsymbol{W}_y \boldsymbol{h}_t + \boldsymbol{b}_y)\end{aligned}$$

  where $\hat{\boldsymbol{y}}_t$ is the probability distribution over the vocabulary.

Key Insights

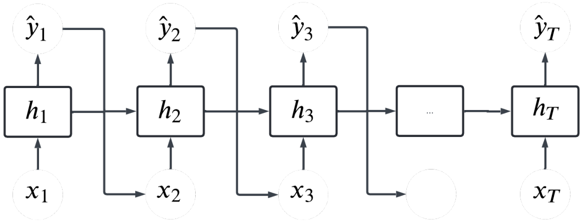
RNNs capture **temporal dependencies** by updating the hidden state $\boldsymbol{h}_t$ at each time step, sharing the same weights $\boldsymbol{W}_h, \boldsymbol{W}_x, \boldsymbol{W}_y$ for **parameter efficiency** and **consistent** learning across sequences.

Training RNNs

The RNN unit updates are defined as:

$$\begin{aligned} \boldsymbol{h}_t &= \tanh(\boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_x \boldsymbol{x}_t + \boldsymbol{b}_h), \\ \hat{\boldsymbol{y}}_t &= \text{softmax}(\boldsymbol{W}_y \boldsymbol{h}_t + \boldsymbol{b}_y). \end{aligned}$$

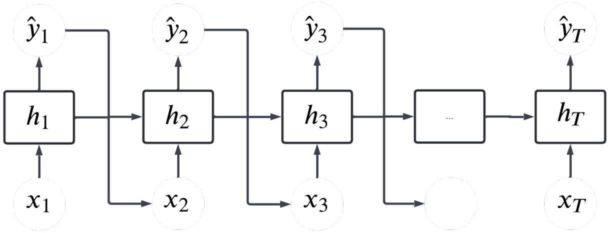
- The training dataset is $\mathcal{D} = \{\boldsymbol{y}^{(i)}\}_{i=1}^{N}$, where each $\boldsymbol{y}^{(i)} = \{\boldsymbol{y}_t^{(i)}\}_{t=1}^{T}$ is a sequence of words.
- Starting with $\boldsymbol{x}_1 = \boldsymbol{y}_1$, the model iteratively uses $\boldsymbol{x}_t = \hat{\boldsymbol{y}}_{t-1}$ to predict $\hat{\boldsymbol{y}}_t$.



- The total cost is computed using cross-entropy loss:

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{i=1}^{N} \sum_{t=1}^{T} \boldsymbol{y}_t^{(i)} \cdot \log \hat{\boldsymbol{y}}_t^{(i)}$$

Backpropogation Through Time



- **Forward propagation**:

$$\boldsymbol{h}_t = \phi(\boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_x \boldsymbol{x}_t), \quad \hat{\boldsymbol{y}}_t = \sigma(\boldsymbol{W}_y \boldsymbol{h}_t).$$

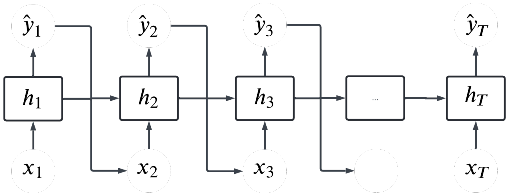
- **Backpropogation through time**:

$$d\boldsymbol{h}_t = \boldsymbol{W}_h^\top \left[\boldsymbol{\phi}'_{t+1} \odot d\boldsymbol{h}_{t+1}\right] + \boldsymbol{W}_y^\top \left[\boldsymbol{\sigma}'_t \odot d\boldsymbol{y}_t\right]$$
$$d\boldsymbol{W}_h = \sum_t \left[d\boldsymbol{h}_t \odot \boldsymbol{\phi}'_t\right] \boldsymbol{h}_{t-1}, \quad d\boldsymbol{W}_x = \sum_t \left[d\boldsymbol{h}_t \odot \boldsymbol{\phi}'_t\right] \boldsymbol{x}_t, \quad d\boldsymbol{W}_y = \sum_t \left[d\hat{\boldsymbol{y}}_t \odot \boldsymbol{\sigma}'_t\right] \boldsymbol{h}_t^\top$$

where $\boldsymbol{\phi}_t := \phi(\boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_x \boldsymbol{x}_t)$ and $\boldsymbol{\sigma}_t := \sigma(\boldsymbol{W}_y \boldsymbol{h}_t)$.

Generating a New Sequence with NLM

Once the NLM is trained, we can sample new sequences of text by following these steps:



1. **Start with a seed word or token**: Choose an initial word as input.
2. **Feed the word to the model**: The model predicts the next word based on the history:

$$\boldsymbol{h}_t = \tanh(\boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_x \boldsymbol{x}_t), \quad \hat{\boldsymbol{y}}_t = \text{softmax}(\boldsymbol{W}_y \boldsymbol{h}_t).$$

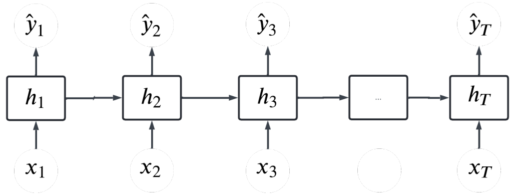
3. **Sample the next word**: Select the next word from the predicted probability distribution:
   - **Greedy decoding**: Pick the word with the highest probability.
   - **Stochastic sampling**: Randomly select a word based on the probability distribution.
4. **Iterate**: Use the **sampled** word as the input for the next time step, and repeat the process.
5. **Termination**: Stop when an EOS token is generated or a maximum length is reached.

Example: Given the current sequence "The dog chases the", the model predicts:

$$\hat{\boldsymbol{y}}_t = \{\text{"a"} : 0.01, \text{"bird"} : 0.25, \text{"cat"} : 0.45, \ldots, \text{"zoo"} : 0.01, \}$$

The model samples "cat", feeds it back, and continues generating until the sequence ends.

Different Types of RNNs



One-to-Many (Sequence Generation): A single input leads to a sequence of outputs.

- **Application**: Image captioning, music generation.

Many-to-One (Sequence Classification): Processes a sequence of inputs to produce a single output.

- **Application**: Sentiment analysis, speech emotion recognition, Speaker Identification/Verification.

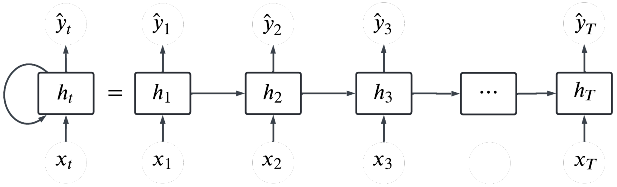
Many-to-Many (Synchronous): Input and output sequences have the same length.

- **Application**: Video classification, named entity recognition.

Many-to-Many (Sequence-to-Sequence): Input and output sequences can have different lengths.

- **Application**: Machine translation, speech recognition.

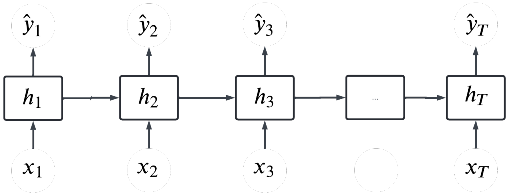
Summary



- **Challenges in Text Data**: High dimensionality and loss of temporal information.
- **Neural Language Models (NLMs)**: Use neural networks to model the conditional probability of the next word given the previous ones.
- **RNNs**: Encode the history into a compact hidden state $\boldsymbol{h}_t$, which is updated by combining the previous hidden state $\boldsymbol{h}_{t-1}$ with the current input $\boldsymbol{x}_t$.
- **Training RNNs**:
  - Forward (simplified): $\boldsymbol{h}_t = \phi(\boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_x \boldsymbol{x}_t)$
  - Backward (simplified): $d\boldsymbol{h}_t = \boldsymbol{W}_h^\top \left(\boldsymbol{\phi}'_{t+1} \odot d\boldsymbol{h}_{t+1}\right) + \boldsymbol{W}_y^\top \left(\boldsymbol{\sigma}'_t \odot d\boldsymbol{y}_t\right)$
- **Generation**: Sample the next word from the predicted probability distribution produced by RNNs.
- **RNN Types**: One-to-many, many-to-one, or many-to-many structures for different tasks.

Outline

Vanishing or Exploding Gradients in RNNs



- **Forward (simplified):**

$$\boldsymbol{h}_t = \phi(\boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_x \boldsymbol{x}_t) \approx \boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_x \boldsymbol{x}_t \approx \boldsymbol{W}_h^t \boldsymbol{x}_0 = \mathcal{O}(a^t)$$

- **Backward (simplified):**

$$d\boldsymbol{h}_t = \boldsymbol{W}_h^\top (\boldsymbol{\phi}'_{t+1} \odot d\boldsymbol{h}_{t+1}) + \boldsymbol{W}_y^\top (\boldsymbol{\sigma}'_t \odot d\boldsymbol{y}_t) \approx \boldsymbol{W}_h^{\top(T-t)} d\boldsymbol{h}_T = \mathcal{O}(b^{T-t})$$

- **Long-term dependencies**:

  "The dog chased a cat down the street, ran out of the house, jumped over a fence, and after running for what seemed like miles, finally caught the ..."

Gated Recurrent Unit (GRU)

- **Update Gate (z_t)**: Controls how much of the previous hidden state is carried forward:

$$\boldsymbol{z}_t = \sigma(\boldsymbol{W}_z[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t])$$

  where $\sigma(\cdot)$ is the sigmoid function, outputting values in the range $(0, 1)$.

- **Final Hidden State (h_t)**: Combines the previous hidden state $\boldsymbol{h}_{t-1}$ and the candidate state $\tilde{\boldsymbol{h}}_t$ based on the update gate $\boldsymbol{z}_t$:

$$\boldsymbol{h}_t = \boldsymbol{z}_t \odot \boldsymbol{h}_{t-1} + (1 - \boldsymbol{z}_t) \odot \tilde{\boldsymbol{h}}_t$$

- **Candidate Hidden State ($\tilde{h}_t$)**: Computed by resetting parts of the previous hidden state:

$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W}_h(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}) + \boldsymbol{W}_x\boldsymbol{x}_t)$$

- **Reset Gate (r_t)**: Determines how much of the previous hidden state should be forgotten:

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}_r[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t])$$

Key Insights

- The update gate $\boldsymbol{z}_t$ helps carry important information from the past for long-term dependencies.
- The reset gate $\boldsymbol{r}_t$ forces discards irrelevant past information, focusing on the most important data.

Cho et al., “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. EMNLP2014.

Long Short-Term Memory (LSTM)

- **Forget Gate (f_t)**: Decides what information to discard from the previous cell state:

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t])$$

- **Input Gate (i_t)**: Controls which new information to update in the cell state:

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t])$$

- **Output Gate (o_t)**: Controls what part of the cell state should be output:

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t])$$

- **Candidate Cell State ($\tilde{c}_t$)**: Computes the new candidate values for the cell state:

$$\tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}_c[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t])$$

- **Cell State (c_t)**: The cell state is updated based on the forget and input gates:

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t$$

  The final hidden state is:

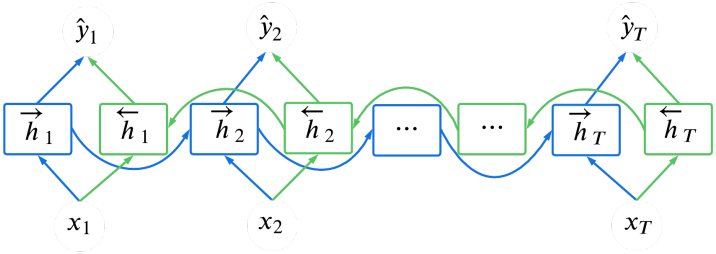
$$\boldsymbol{h}_t = o_t \odot \tanh(\boldsymbol{c}_t)$$

Key Insights

- LSTMs maintain long-term dependencies via the memory cell state $\boldsymbol{c}_t$.
- LSTMs are more flexible than GRU by using more gates to control the flow of information.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation.

Bidirectional Recurrent Neural Networks (BRNNs)



- **Forward**: The RNN processes the input sequence from the first time step to the last.

$$\overrightarrow{\boldsymbol{h}}_t = \tanh(\boldsymbol{W}_f[\overrightarrow{\boldsymbol{h}}_{t-1}, \boldsymbol{x}_t])$$

- **Backward**: Another RNN processes the sequence in reverse from the last time step to the first.

$$\overleftarrow{\boldsymbol{h}}_t = \tanh(\boldsymbol{W}_b[\overleftarrow{\boldsymbol{h}}_{t+1}, \boldsymbol{x}_t])$$

- **Combined Hidden State**: The final hidden state is the concatenation of the forward and backward states:

$$\boldsymbol{h}_t = [\overrightarrow{\boldsymbol{h}}_t, \overleftarrow{\boldsymbol{h}}_t]$$

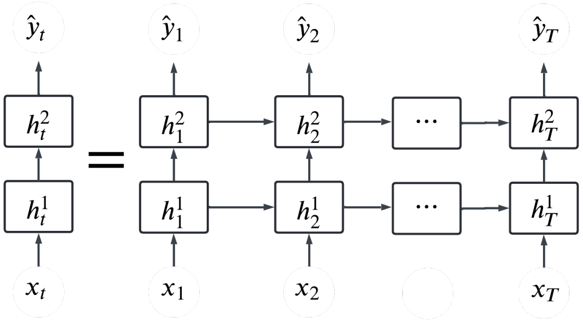
Schuster, M., & Paliwal, K. K. (1997). Bidirectional Recurrent Neural Networks. IEEE Transactions on Signal Processing.

Deeper RNNs

- **Forward propagation**: Each layer ℓ computes its hidden state by using the hidden state from the previous layer $\ell - 1$:

$$\boldsymbol{h}_t^{(\ell)} = \phi(\boldsymbol{W}_h^{(\ell)} \boldsymbol{h}_{t-1}^{(\ell)} + \boldsymbol{W}_x^{(\ell)} \boldsymbol{h}_t^{(\ell-1)}),$$

where $\boldsymbol{h}_t^{(0)} = \boldsymbol{x}_t$.



- The hidden state from the final layer is used for predictions.

Graves, A. (2013). Speech recognition with deep recurrent neural networks. ICASSP.

Outline

Featurized Representation

Drawbacks of One-Hot Representation:

- **Orthogonality**: One-hot vectors are orthogonal, meaning they don't capture any relationships or similarities between words.
- **High Dimensionality**: One-hot vectors are sparse and grow with the size of the vocabulary, making them inefficient for large vocabularies (e.g., millions of words).

Example: Word Correlation Matrix

Category	**man**	**woman**	**king**	**queen**	**apple**	**orange**
Gender	-1.00	1.00	-0.95	0.97	0.01	0.02
Royalty	0.01	0.02	1.00	0.9	-0.01	0.08
Age	0.02	0.03	0.71	0.68	0.05	0.05
Food	-0.01	0.02	-0.02	0.03	0.81	0.75

Word Embedding:

- Word embeddings represent words as **dense vectors** in a **lower-dimensional space**.
- Words used in similar contexts have higher correlations, capturing their **semantic relationships**.

Using Word Embeddings

Embedding Matrix:

- The embedding matrix $\boldsymbol{E} \in \mathbb{R}^{d\times V}$ is a learned matrix that transforms a one-hot encoded word $\boldsymbol{x}$ into a dense word vector $\boldsymbol{e}$:

  $$\boldsymbol{e} = \boldsymbol{E}\boldsymbol{x}$$

  where d is the embedding dimension, and V is the size of the vocabulary.
- This transformation allows words to be represented as **dense** vectors in a **lower-dimensional** space, capturing **semantic** relationships.

Transfer Learning:

- Word embeddings can be pre-trained on large corpora (e.g., Wikipedia or news articles), allowing models to capture rich semantic relationships.
- In small datasets, rare or specific words may be hard to learn, but pre-trained embeddings group similar words together. This helps models generalize better, even with limited data.

Analogy:

- Word embeddings capture analogies.

  $$\text{man} - \text{woman} \approx \text{king} - \text{queen}$$

- This means that models can understand not only word meanings but also deeper relationships between words.

Mikolov et al., "Linguistic Regularities in Continuous Space Word Representations", NAACL 2013.

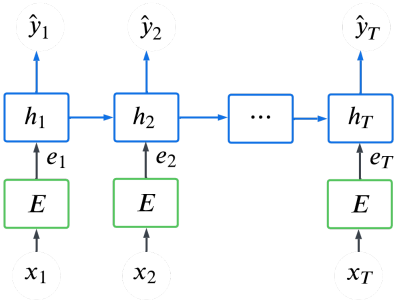
Learning Word Embeddings with a Language Model

Let $\{\boldsymbol{w}_1, \cdots, \boldsymbol{w}_T\}$ be a sequence of words, *e.g.*, "*The cat chased a mouse into a hole in the wall.*"

- In previous language models, we aim to model the probability of the next word given the history:

$$\mathbb{P}(\boldsymbol{w}_t \mid \boldsymbol{w}_1, \ldots, \boldsymbol{w}_{t-1}) = f_{\boldsymbol{\theta}}(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_{t-1}),$$

where $\boldsymbol{x}_t$ is a one-hot vector representing the word at time step t.



- With word embeddings, this becomes:

$$\mathbb{P}(\boldsymbol{w}_t \mid \boldsymbol{w}_1, \ldots, \boldsymbol{w}_{t-1}) = f_{\boldsymbol{\theta}}(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_{t-1}),$$

where the word embedding $\boldsymbol{e}_t$ is computed by

$$\boldsymbol{e}_t = \boldsymbol{E}\boldsymbol{x}_t,$$

$\boldsymbol{E} \in \mathbb{R}^{d \times V}$ is the embedding matrix, with d the embedding dimension, and V the vocabulary size.

- During training, both language model f_θ and the embedding matrix $\boldsymbol{E}$ are learned **simultaneously**.

Yoshua Bengio et al., "A Neural Probabilistic Language Model", Journal of Machine Learning Research, 2003.

Word2vec: CBOW and Skip-Gram

Continuous Bag of Words (CBOW): Predicts the target word given the context:

$$\mathbb{P}(\boldsymbol{w}_t \mid \boldsymbol{w}_{t-n}, \ldots, \boldsymbol{w}_{t+n}) = f_{\boldsymbol{\theta}}(\boldsymbol{e}_{t-n}, \ldots, \boldsymbol{e}_{t+n})$$

where the n-gram sequence $\{\boldsymbol{e}_{t-n}, \cdots, \boldsymbol{e}_{t+n}\}$ represents **context words** and $\boldsymbol{e}_t$ is the **target word**.

- Practically, a **shallow** network with $n = \mathcal{O}(1)$ is sufficient to learn embedding matrix $\boldsymbol{E}$.

$$\boldsymbol{e}_{t\pm i} = \boldsymbol{E}\boldsymbol{x}_{t\pm i}, \quad \boldsymbol{h} = \boldsymbol{W}[\boldsymbol{e}_{t-n}, \ldots, \boldsymbol{e}_{t+n}], \quad \hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{h}), \quad \forall i \in \{1, 2, \ldots, n\},$$

  where the concatenation $[\boldsymbol{e}_{t-n}, \ldots, \boldsymbol{e}_{t+n}]$ can be replaced with summation or averaging to reduce computational complexity.

Skip-Gram: Predicts the context words given a target word.

$$\mathbb{P}(\boldsymbol{w}_{t-n}, \ldots, \boldsymbol{w}_{t+n} \mid \boldsymbol{w}_t) = f_{\boldsymbol{\theta}}(\boldsymbol{e}_t)$$

- For each context word $\boldsymbol{w}_{t\pm i}$, we predict its probability given the embedding of target word $\boldsymbol{e}_t$:

$$\boldsymbol{e}_t = \boldsymbol{E}\boldsymbol{x}_t, \quad \boldsymbol{h}_i = \boldsymbol{W}\boldsymbol{e}_t, \quad \hat{\boldsymbol{y}}_{t\pm i} = \text{softmax}(\boldsymbol{h}_i), \quad \forall i \in \{1, 2, \cdots, n\}.$$

Mikolov et al., “Efficient Estimation of Word Representations in Vector Space”, ICLR 2013.

Negative Sampling

- **Expensive softmax**: The full softmax is computationally expensive because it requires summing over the entire vocabulary:

$$\text{softmax}(\boldsymbol{h}) = \frac{e^{\boldsymbol{h}_j}}{\sum_{i=1}^{V} e^{\boldsymbol{h}_i}}$$

  where $V \sim 1$ million.
- Reformulates the context-target prediction as a **binary** classification: $(\boldsymbol{w}_t, \boldsymbol{w}_c, y)$, where y is label.
- **Binary classifier for context-target pair**: $\mathbb{P}(y = 1 \mid \boldsymbol{w}_c, \boldsymbol{w}_t) = \sigma(\boldsymbol{e}_c^\top \boldsymbol{e}_t)$, where $\sigma(\cdot)$ is sigmoid
- **Maximize log-likelihood**: It is equivalent to minimizing the following objective:

$$\mathcal{L}(\boldsymbol{E}) = -\sum_{(\boldsymbol{e}_c, \boldsymbol{e}_t)} \log \sigma(\boldsymbol{e}_t^\top \boldsymbol{e}_c) - \sum_{(\boldsymbol{e}_c, \tilde{\boldsymbol{e}}_t)} \log \sigma(-\boldsymbol{e}_t^\top \tilde{\boldsymbol{e}}_c)$$

  where $\tilde{e}_t$ are **negative** samples from words outside the context window.

Mikolov et al., "Efficient Estimation of Word Representations in Vector Space", ICLR 2013.