NLP Notes

Natural Language Processing (University of Technology Sydney)

41043 Natural Language Processing Notes

1 Week 3 - Part of Speech Tagging

1.1 POS Tagging

Problem: Assigning a part-of-speech marker (tag) to each word in an input text.

Input: A sequence of tokenized words and a tagset.

Output: A sequence of tags, one per token.

Aim of POS Tagging: Tagging is ambiguous because words can have ≥ 1 possible part-of-speech. POS Tagging must resolve these ambiguities by choosing the correct tag for the context. **Accuracy** is calculated by the % of tags correctly labeled.

1.2 Hidden Markov Model (HMM)

1.2.1 HMM

The HMM is a probabilistic sequence model: Given a sequence of words, it computes probability distribution over possible sequences of labels and chooses the best label sequence.

Represent a sequence of **n** words as $w_1, ..., w_n$ or w_1^n. The probability of a word sequence is:

$$\begin{aligned} P(w_1, ..., w_n) &= P(w_1^n) \\ &= P(w_1) \times P(w_2|w_1) \times P(w_3|w_1^2) \times ... \times P(w_n|w_1^{n-1}) \\ &= \prod_{i=1}^{n} P(w_i|w_1^{i-1}) \end{aligned} \tag{1}$$

Bigram model based on Markov's assumption: P(next word) depend only on the previous word, thus: $P(w_n|w_1^{n-1}) := P(w_n|w_{n-1})$

The sentence probability sequence (1) becomes:

$$\begin{aligned} P(w_1^n) &= P(w_1) \times P(w_2|w_1) \times P(w_3|w_1^2) \times ... \times P(w_n|w_1^{n-1}) \\ &\approx P(w_1) \times P(w_2|w_1) \times P(w_3|w_2) \times ... \times P(w_n|w_{n-1}) = \prod_{i=1}^{n} P(w_i|w_{i-1}) \end{aligned} \tag{2}$$

Let w_1^n denote the word sequence of length n and t_1^n denote the tag sequence of length n. The Hidden Markov Model (HMM) aims to choose the tag sequence t_1^n that is most likely given the observed sequence of words w_1^n:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n|w_1^n) \tag{3}$$

Assumption 1: The probability of a word appearing depends only on its own tag and is independent of neighboring words and tags

$$P(w_1^n|t_1^n) \approx \prod_{i=1}^{n} P(w_i|t_i) \tag{4}$$

Assumption 2: The probability of a tag depends only on the previous tag, not the entire tag sequence

$$P(t_i^n) \approx \prod_{i=1}^{n} P(t_i|T-i-1) \tag{5}$$

Replacing (4) and (5) into (3) produces the equation for the most likely tag sequence:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}}\, P(t_1^n|w_1^n) = \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1}) \tag{6}$$

The HMM is made up of 2 components. Their maximum likelihoods is calculated by counting:

- $P(w_i|t_i)$ is the **emission probability**, expressing the probability of an observation (word) w_i being generated from a state (tag) t_i

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)} = \frac{\text{number of times } t_i \text{ is associated with given word } w_i}{\text{number of times } t_i \text{ is observed in the text}}$$

- $P(t_i|t_{i-1})$ is the **transition probability**, expressing the probability of moving from a state (tag) t_{i-1} to the next state (tag) t_i

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})} = \frac{\text{number of times } t_{i-1} \text{ is followed by } t_i}{\text{number of times } t_{i-1} \text{ is observed in the text}}$$

1.2.2 Challenges with HMM

- **Narrow window context:** Bigram HMM uses a 1-word window context to predict the current word's POS tag, which can limit the ability to capture context beyond the window size.
- **Sparsity:** POS tagging requires large amount of annotated training data. This can lead to sparsity issues, especially for low frequency words or rare POS tags
- **Unknown words:** HMM tends to assign low probability to unseen or out-of-vocab worse, leading to incorrect tagging for those words
- **Lack of lexical info:** HMM relies on transition probabilities between POS tags and emission probabilities of words based on given POS tags. It doesn't fully use lexical info or word meanings, especially in cases of ambiguity.

1.3 Maximum Entropy Markov Model (MEMM)

1.3.1 MEMM vs. HMM

In HMM, we compute the best tag sequence that maximizes ($P(t_1^n|w_1^n)$ by relying on Bayes' rule and the likelihood $P(w_1^n|t_1^n)$

$$\begin{aligned}\hat{t}_1^n &= \underset{t_1^n}{\operatorname{argmax}}\, P(t_1^n|w_1^n) = \underset{t_1^n}{\operatorname{argmax}}\, P(w_1^n|t_1^n)P(t_1^n) \\ &= \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_i|t_i) \prod_{i=1}^{n} P(t_i|t_{i-1})\end{aligned}$$

In MEMM, we compute $P(t_1^n|w_1^n)$ directly, conditioned on the previous state and the current observation:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n|w_1^n) = \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(t_i|w_i, t_{i-1})$$

1.3.2 Features in MEMMs

A MEMM POS tagger conditions on the observation word w_i it self, neighboring words, previous tags, and various combinations of them:

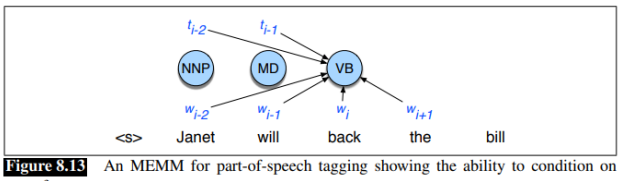


Figure 8.13 An MEMM for part-of-speech tagging showing the ability to condition on more features.

$$\langle t_i, w_{i-2}\rangle, \langle t_i, w_{i-1}\rangle, \langle t_i, w_i\rangle, \langle t_i, w_{i+1}\rangle, \langle t_i, w_{i+2}\rangle$$
$$\langle t_i, t_{i-1}\rangle, \langle t_i, t_{i-2}, t_{i-1}\rangle,$$
$$\langle t_i, t_{i-1}, w_i\rangle, \langle t_i, w_{i-1}, w_i\rangle \langle t_i, w_i, w_{i+1}\rangle,$$

To improve accuracy, we have to build and incorporate better features into our model, leading to the following strengths of the MEMM over HMM:

- **Broader window size:** MEMM allows for a window size broader (looking over the next words or more words from the past) than the bigram HMM, enabling it to capture longer-range context.
- **Inclusion of lexical information:** MEMM incorporates lexical information as features (upper/lowercases, prefixes, suffixes, word shapes)
- **Handling unknown words:** MEMM can handle unknown words better than HMM as they rely on features derived from the observed sequence rather than predefined emission probabilities

2 Week 4 - Name-entity Recognition

2.1 NER Task Definition

A subtask of Information Extraction: Extracting semantic content from unstructured text.

Aim of NER: To find all mentions of the predefined named entities (proper names) in a given text and label its type (e.g. person, organization, location, etc.)

2.2 NER as Sequence Labelling

The standard algorithm for NER is a word-by-word sequence labeling task called **(BIO tagging - beginning, inside, outside)**, in which the assigned tasks capture both the boundary and the type.

- **I- prefix:** The word is inside of the named entity chunk of text
- **O- prefix:** The word is outside of the named entity chunk of text, i.e. not a proper name
- **B- prefix**: The word marks the beginning of a named entity chunk of text. This helps distinguishing between 2 named entities placed next to each other in a sentence.

2.3 NER Approaches

2.3.1 Lexical Rules and Gazetteers

Approach 1: Extract features and train a MEMM or CRF sequence model to label new sentences However,

identity of w_i, identity of neighboring words
embeddings for w_i, embeddings for neighboring words
part of speech of w_i, part of speech of neighboring words
base-phrase syntactic chunk label of w_i and neighboring words
presence of w_i in a **gazetteer**
w_i contains a particular prefix (from all prefixes of length ≤ 4)
w_i contains a particular suffix (from all suffixes of length ≤ 4)
w_i is all upper case
word shape of w_i, word shape of neighboring words
short word shape of w_i, short word shape of neighboring words
presence of hyphen

Figure 17.5 Typical features for a feature-based NER system.

this is not scalable as new named entities keep appearing, requiring gazeetteers to be regularly updated. Some entities are ambiguous (i.e. can fall into ≥ 1 types). Mispellings are not accounted.

2.4 Traditional Machine Learning

A 2-step Approach: Feature engineering then train a classifier (HHM, SVM, CRF)

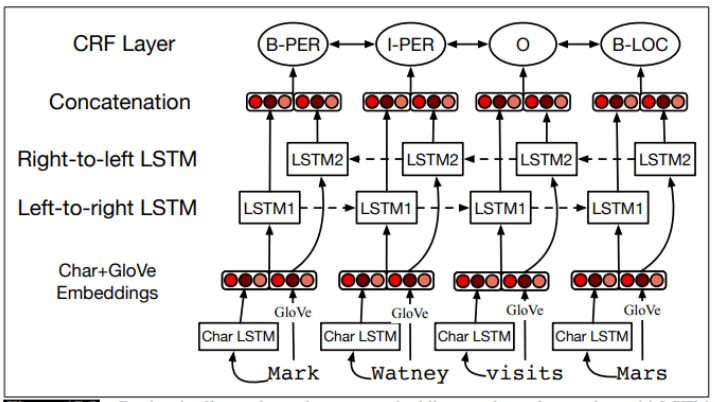


Figure 17.8 Putting it all together: character embeddings and words together a bi-LSTM sequence model. After Lample et al. (2016).

2.5 Conditional Random Field

2.5.1 CRF Introduction

Intuition

The HMM models the joint probability $P(Y, X)$ for input words $X = x_1, ..., x_n$ and output tags $Y = y_1, ..., y_n$. The joint probability measures the likelihood that 2 events happen at the same time. The assumptions for HMM are that: (1) each word is conditionally independent of other words given its tag, and (2) a tag depends only on its previous tag and not the entire sequence. In reality, there are many cases where features of words are highly correlated.

HMM Approach:

- In an HMM-based NER system, we model the joint probability of the observed words and the sequence of entity labels. The emission probabilities capture the likelihood of observing a particular word given the hidden entity label.
- Features related to word capitalization and presence of business-related keywords might be highly correlated. For example, if a word is capitalized, it's more likely to represent a named entity. Similarly, if a word contains a business-related keyword, it's more likely to represent a company name.
- However, if these features are highly correlated, the model might become overly reliant on one feature and fail to generalize well to unseen data. For instance, if many company names in the training data happen to be capitalized, the model might incorrectly assume that all capitalized words are company names.

In NER case, most of the time the words to be tagged are unknown words (proper names), so it's difficult for HMM to work well.

Conditional Random Field (CRF) is a discriminative model that computes the conditional probability $P(Y|X)$ of the entire

3 Week 5 - Topic Modelling

3.1 Task Description

Topic is a probability distribution over a fixed word vocabulary. In a topic model, each document is described as being composed of a **mixture of corpus-wide topics** rather than one topic alone. For each document, we find the **topic proportions** that maximize the probability that we would observe the words in that particular document.

A topic model simultaneously estimates two sets of probabilities

1. The probability of observing each word for each topic
2. The probability of observing each topic in each document

These quantities can then be used to organize documents by topic, assess how topics vary across documents, etc.

- Consider a vocabulary: gene, dna, genetic, data, number, computer
- When speaking about **genetics**, you will:
  - frequently use the words "gene", "dna" & "genetic"
  - infrequently use the words "data", "number" & "computer"
- When speaking about **computation**, you will:
  - frequently use the words "data", "number" & "computation"
  - infrequently use the words "gene", "dna" & "genetic"

Topic	gene	dna	genetic	data	number	computer
Genetics	0.4	0.25	0.3	0.02	0.02	0.01
Computation	0.02	0.01	0.02	0.3	0.4	0.25

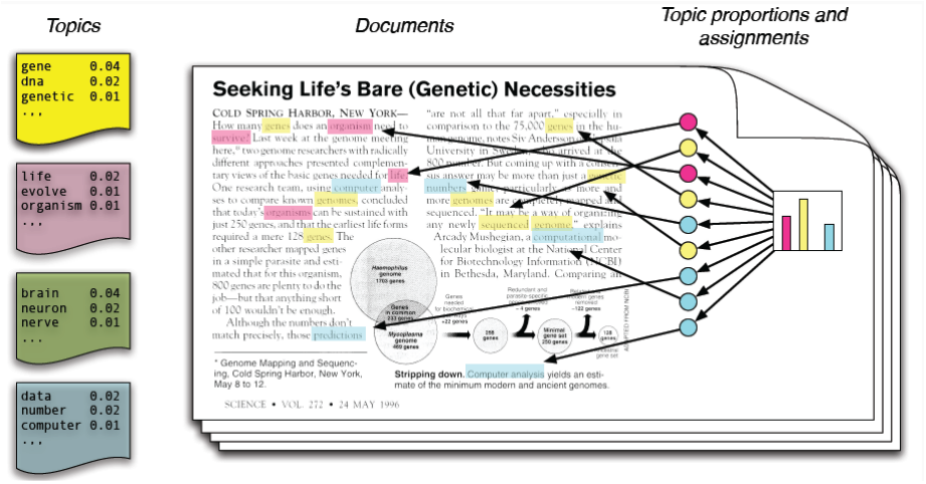
Imagine we have two documents with the following word counts

Table 2: Document word counts

Doc	gene	dna	genetic	data	number	computer
A	2	3	1	3	2	1
B	2	4	2	1	2	1

Table 3: Topic distributions

Topic	gene	dna	genetic	data	num
Genetics	0.4	0.25	0.3	0.02	0.02
Computation	0.02	0.01	0.02	0.3	0.4



- The researcher picks a number of topics, K.
- Each **topic** k is a distribution over words
- Each **document** d is a mixture of corpus-wide topics
- Each **word** j is drawn from one of those topics

In reality, only the document d is observed. The other structures are **hidden**, which we have to compute their distribution conditioned on the document.

3.2 Topic Models

HMM in Topic Modelling

Take an HMM, but give every document its own transition probabilities (rather than a global parameter of the corpus) $\longrightarrow$ This lets you specify that certain topics are more common in certain documents.

We also assume the hidden state of a token doesn't actually depend on the previous tokens.

The probability of a token is the join probability of the word and the topic label.

$$P(\text{word=Apple, topic=1}|\theta_d, \beta_1) = P(\text{word=Apple }|\text{topic=1, }\beta_1)P(\text{topic=1 }|\theta_d) \tag{7}$$

- θ_d: the distribution of all topics on document d
- β_1: the distribution of topic 1 over all words
- P(word=Apple |topic=1, β_1): the emission probability, which is global across all documents
- P(topic=1 |θ_d): the transition probability, which is local to each document

The probability of the word Apple and the topic is 1 given the distribution θ_d and β_1 = the probability of the word Apple given the topic is 1 $\times$ the probability of topic 1 given the distribution of all topics in document d.

- The probability of a document is the product of all of its token probabilities (tokens are independent)
- The probability of a corpus is the product of all of its document probabilities.

Need to estimate the parameters θ, β that maximize the likelihood of the observed data $\longrightarrow$ But we don't know the hidden topic assigned to each token.

3.3 Expectation Maximization

Initialization: Starting with some random guess for θ and β.

The E Step: Estimate the posterior probability, i.e. the expected value of the variables, given the current model parameters θ and β.

$$P(\text{topic=1}|\text{word=Apple}, \theta_d, \beta_1) = \frac{P(\text{word=Apple, topic=1}|\theta_d, \beta_1)}{\sum_k P(\text{word=Apple, topic=}k|\theta_d, \beta_k)} \tag{8}$$

The M Step

Calculate new θ_{d1} by summing over each token i in document d.

$$\begin{aligned} \text{New } \theta_{d1} &= \frac{\#\text{ tokens in } d \text{ whose topic is 1}}{\#\text{ tokens in } d} = \frac{\text{Expected \# tokens with topic 1}}{\text{Expected \# tokens}} \\ &= \frac{\sum_{i \in d} P(\text{topic } i = 1|\text{word } i, \theta_d)}{\sum_k \sum_{i \in d} P(\text{topic } i = k|\text{word } i, \theta_d, \beta_k)} \end{aligned} \tag{9}$$

Calculate new β_{1w} by summing over each token i in the entire corpus.

$$\begin{aligned} \text{New } \beta_{1w} &= \frac{\#\text{ tokens with topic 1 and word } w}{\#\text{ tokens with topic 1}} = \frac{\text{Expected \# of times word } w \text{ belongs to topic 1}}{\text{Expected \# of all tokens belonging to topic 1}} \\ &= \frac{\sum_i I(\text{word } i = w) P(\text{topic } i = 1|\text{word } i = w, \theta_d, \beta_1)}{\sum_v \sum_i I(\text{word } i = v) P(\text{topic } i = 1|\text{word } i = v, \theta_d, \beta_1)} \end{aligned} \tag{10}$$

Then go back to E step.

3.4 Latent Dirichlet Allocation - LDA

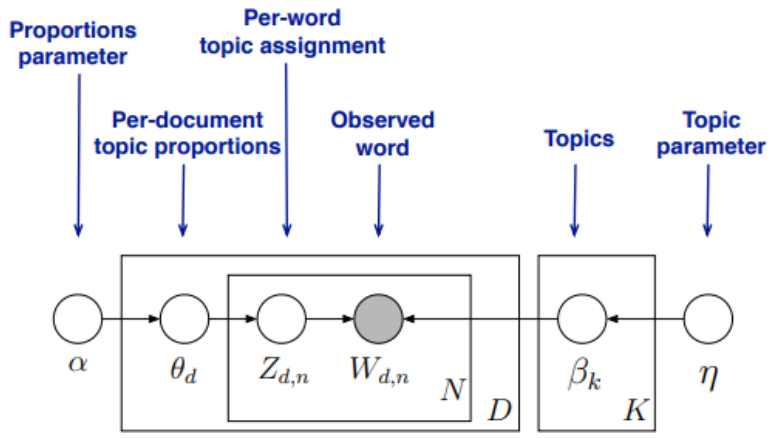
The parameters θ and β are random variables. Some values are more likely than others due to a prior distribution.

- $P(\theta|\alpha) = \text{Dirichlet}(\alpha)$
- $P(\beta|\eta) = \text{Dirichlet}(\eta)$

The Dirichlet distribution is a distribution over the simplex, which is a geometric object that generalizes the notion of a triangle to higher dimensions. A draw from a Dirichlet distribution returns a positive vector that sum to one, i.e. multinomial distribution.

α is a vector that gives the mean/variance of the distribution. Larger α means higher density around mean, and lower variance.

3.5 LDA



$$P(\beta_{1:K}, \theta_{1:D}, w_{1:D}) = \prod_{i=1}^{K} P(\beta_i) \prod_{d=1}^{D} P(\theta_d) \left(\prod_{n=1}^{N} P(z_{d,n}|\theta_d) P(w_{d,n}|\beta_{1:K}, z_{d,n}) \right) \tag{11}$$

Step 1: For each topic, draw a probability distribution over words

$\beta_k \sim \text{Dirichlet}(\eta)$, $\beta_k \in \{0, 1\}$ and $\sum_{j=1}^{J} \beta_{j,k} = 1 \longrightarrow$ Probability that word j occurs in topic k.

Step 2: For each document, draw a probability distribution over topics.

$\theta_d \sim \text{Dirichlet}(\alpha)$, $\theta_{d,k} \in \{0, 1\}$ and $\sum_{k=1}^{K} \theta_{d,k} = 1 \longrightarrow$ Probablity that topic k occurs in document d.

Step 3: For each word in each document:

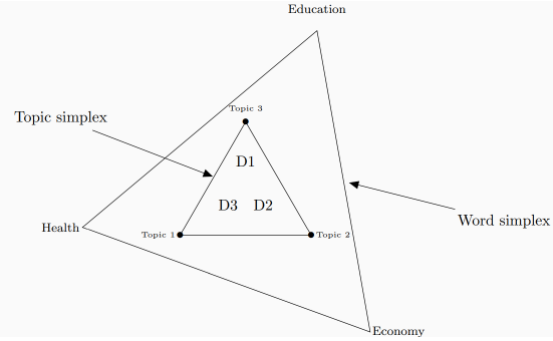
1. Draw one of K topics from **Step 2** (θ_d): $z_i \sim \text{Multinomial}(\theta_d) \longrightarrow$ Topic indicator of word i
2. Draw one of J words from **Step 1** (β_k): $w_i \sim \text{Multinomial}(\beta_{z_i}) \longrightarrow$ Actual word of word i

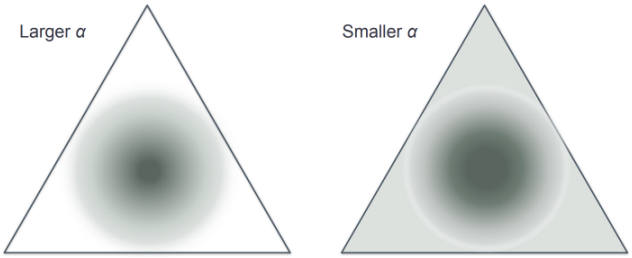
The Dirichlet is used twice in LDA:

- The topics (β_k) are a J dimensional Dirichlet (topics are a probability distribution over words)
- The topic proportion (θ_d) are a K dimensional Dirichlet (documents are a probability distribution over topics)

The parameters α or η controls the sparsity of the draws from the Dirichlet distribution.

- Large α: the probabilities will be more evenly spread across categories
- Small α: more probability mass will be allocated to particular categories





Why does LDA work? By trading off 2 goals, LDA can find groups of tightly occurring words

1. For each document, allocate its words to as few topics as possible (α)
2. For each topic, assign high probability to as few terms as possible (η)

Goal (1) of putting a document in a single topic makes Goal (2) hard: All of a document's words must have probability under that topic.

Goal (2) of putting very few words in each topic makes (1) hard: To cover a document's words, it must assign many topics to it.

3.6 The Posterior Distribution

The probability of the hidden variables and parameters, give the observed data:

$$P(\beta_{1:K}, \theta_{1:D}, z_{1:D}|w_{1:D}) = \frac{P(\beta_{1:K}, \theta_{1:D}, z_{1:D}, w_{1:D})}{P(w_{1:D})} \text{ where } P(w_{1:D}) \text{ is a constant} \tag{12}$$

Using **Expectation Maximization** to find the MAP value (the parameters that maximize the posterior probability): E Step is the same. M Step is modified

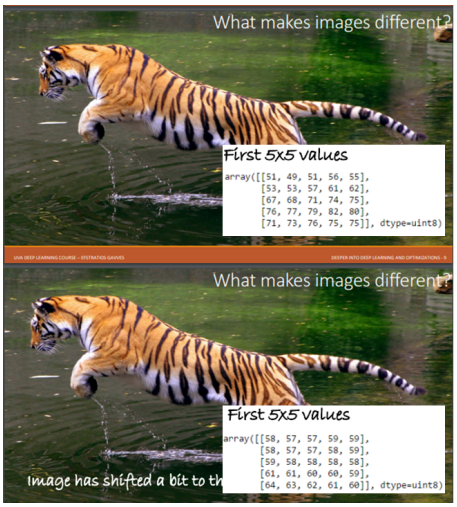
$$\text{New } \theta_{d1} = \frac{\alpha_1 - 1 + \sum_{i \in d} P(\text{topic } i = 1|\text{word } i, \theta_d, \beta_1)}{\sum_k (\alpha_k - 1 + \sum_{i \in d} P(\text{topic } i = k|\text{word } i, \theta_d, \beta_k)} \tag{13}$$

Advantages	Disadvantages
Automatically finds substantively interesting collections of words	Generated topics may not reflect substantive interest of researcher
Automatically labels documents in "meaningful" ways	Many estimated topics may be redundant for research question
Easily scaled to large corpora (millions of documents)	Requires extensive post-hoc interpretation of topics
Requires very little prior work (no manual labelling of texts/dictionary construction etc)	Sensitivity to number of topics selected (what is the best choice for K?)

4 Week 6 - Text CNNs

4.1 Convolutional Neural Network - CNN

Motivation Example: Showing that neighboring input variables are locally correlated



An image has **spatial structure (width, height, depth)**

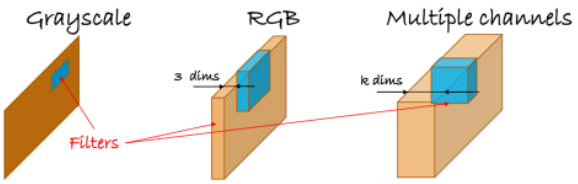
Images have huge dimensionality

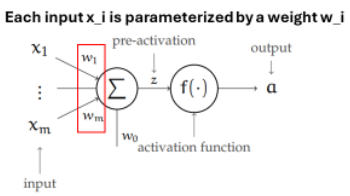
Image = stationary signals that share features

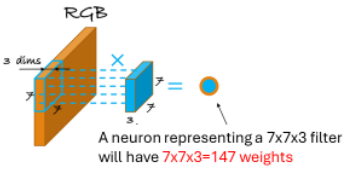
- After variance images are still meaningful
- Small visual changes (often invisible to naked eye) = big changes to input vector
- But semantics still remain
- Basic natural image statistics are the same

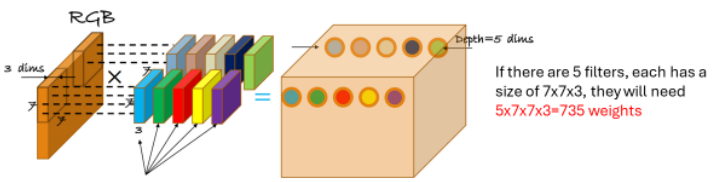
If images are k-D, parameters should be organized in k-D in order to:

- Learn the local correlations between input variables
- Exploit the spatial nature of images



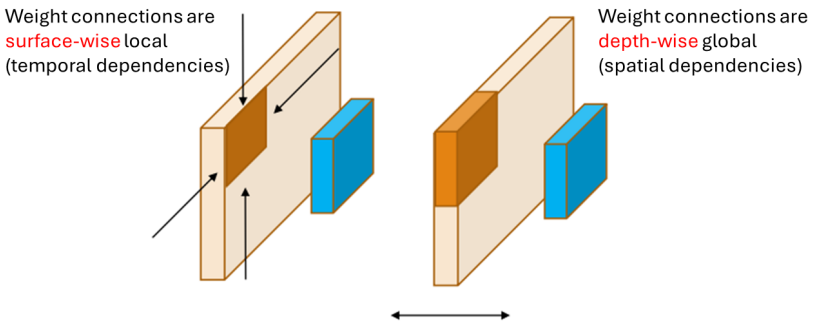




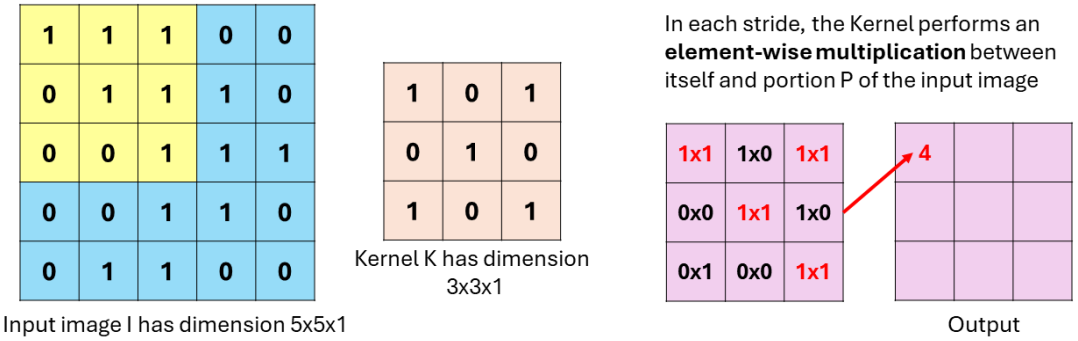


Convolutional Neural Network (CNN) is a neural network with some convolutional layers (and others), which have a number of filters that does convolutional operation.

A CNN can successfully capture the spatial and temporal dependencies in an image through the application of relevant filters. ⟶ A better fitting to the image dataset due to the reduction in the number of parameters involved and the reusability of weights.



Role of CNN: To reduce the images into a form that is easier to process, without losing features critical for getting a good prediction.

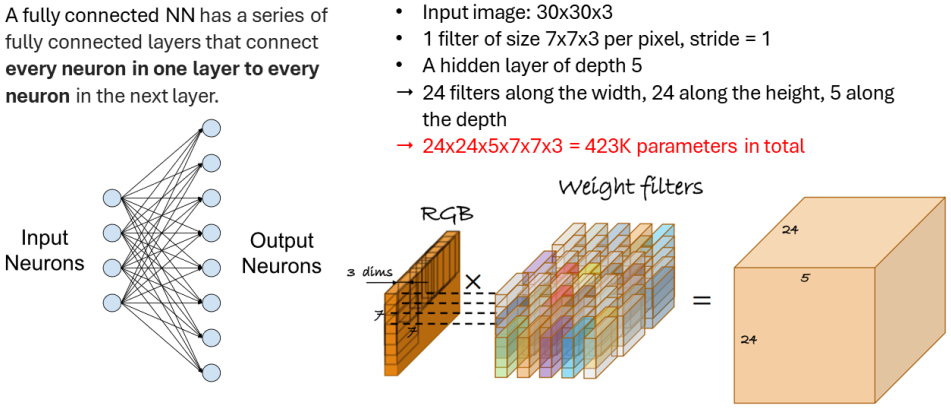


The filter moves to the right with a certain **stride value** till it parses the complete width. Then, it hops down to the beginning (left) of the image with the same **stride value** and repeats the process until the entire image is traversed.

≥ **1 convolution layer:** The first layer can capture lower-level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to higher-level features.

⟶ **Result:** High-level features are extracted from the input image.

4.2 Convolution vs. Fully Connected Layer



Problems with fully connected NN:

- Fitting a model with that many parameters is not easy $\longrightarrow$ Computationally expensive
- With high-dimensional data and small samples, such a model is prone to overfitting
- Are all these weights necessary?

4.3 Why Pooling

Intuition: Find local patterns in the early layers, then look for patterns in those patterns. Having a stride value ≥ 1 makes the images smaller in successive layers, but doesn't necessarily aggregate information over that spatial range.

Pooling traits:

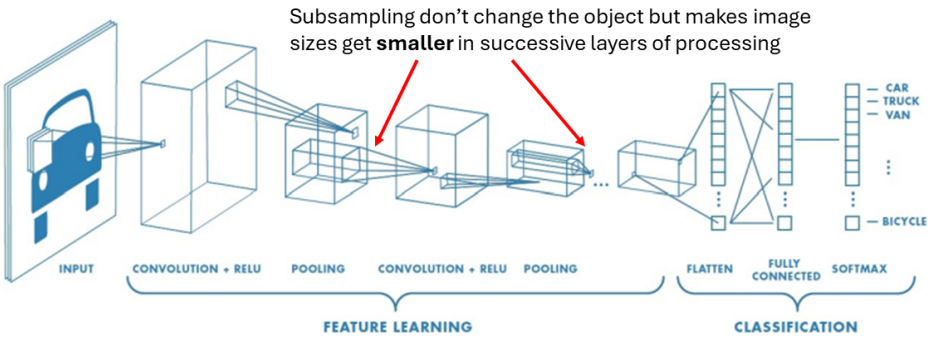
- stride ≥ 1: the resulting image is smaller than the input image
- filter size $\geq$ stride: the whole image is covered

Max Pooling returns the maximum value from the portion of the image covered by the Kernel. MP also performs as a noise suppressant. It discards the noisy activations altogether along with dimensionality reduction.

Roles of Pooling:

- Reduce the spatial size of the convolved feature
- Decrease the computational power required to process the data through dimensionality reduction
- Extract dominant features which are rotational and positional invariant, thus maintaining the effective training the model.

Therefore, **CNN > Fully connected NN** because CNN reduces the number of connections and pooling further reduces complexity.



After each convolution filter layer, there is generally a ReLU layer, then a max pooling layer. Repeat the layers until the output is down to a relatively small size. This output image is then flattened, leading to a fully connected layer, to an activation function (e.g. softmax) that produces the final output.

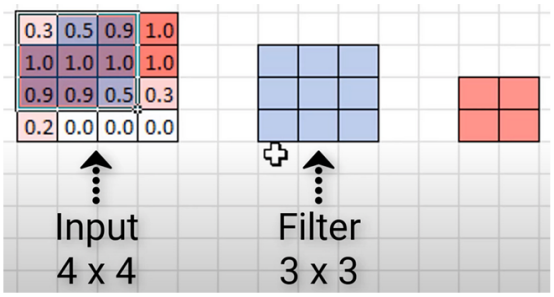
4.4 CNN in Text Classification

Main CNN idea:

- Compute vectors for every possible word subsequence of a certain length?
- Example: "tentative deal reached to keep government open" computes vectors for:
  - tentative deal reached, deal reached to, reached to keep, to keep government, keep government open
- Do not need to worry whether the phrase is grammatical
- Do not need to be linguistically or cognitively plausible
- Then group them

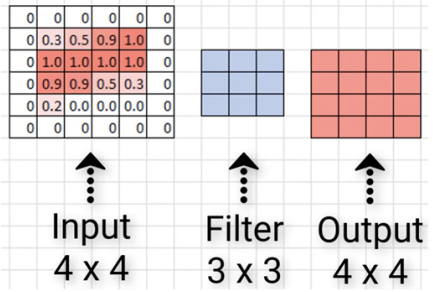
Zero Padding occurs when we add a border of pixels all with value zero around the edges of our input.

Input size: n x n
Filter size: f x f
Output size = (n – f + 1) x (n – f + 1)



Losing valuable info around the edges of the input because the filter is not convolving those edges nearly as much as its convolving the inner pieces of the input

By adjusting the amount of zero-padding, one can **control the size of the output** feature maps.



Padding **preserves boundary info** → Helpful in cases where objects are located near the edges.

4.4.1 Single layer CNN

Representation of sentences

There are n words in a sentence, each is denoted $x_i \in \mathbb{R}^k$ so x_i is a k-dimensional word vector. A sentence is represented by:

$$X_{1:n} = X_1 \oplus X_2 \oplus ... \oplus X_n \text{ where } \oplus \text{ is the concatenation operator} \tag{14}$$

Convolutional layer

A filter is denoted as $w \in \mathbb{R}^h k$. w has h rows and k columns $\longrightarrow$ h is the number of words covered in a stride at a time, k is simply the size of the word vectors.

A feature map $c = [c_1, c_2, ..., c_{n-h+1}]$ can be generated by:

$$c_i = f(w \times x_{i:i+h-1} + b) \tag{15}$$

where w is the filter, x_{h-h+1} represent all the possible windows of length h starting from position i to position $i+h-1$, b is the bias term, f is the ReLU activation operator.

The result is c, a vector whose length equals the number of times the filter shifts through the sentence.

Max pooling

From the feature map $\mathbf{c} = [c_1, c_2, c, ..., c_{n-h+1}]$, max pool a single number $\hat{c} = max(\mathbf{c})$. This selects the most important feature from each feature map. Therefore, each best feature $\hat{c}$ is generated by one filter w.

Fully connected layer

Each $\hat{c}$ generated by each filter is flattened into a single vector $\mathbf{Z} = [\hat{c_1}, \hat{c_2}, ..., \hat{c_j}]$.

5 Week 7 - RNN and LSTM

5.1 Motivation

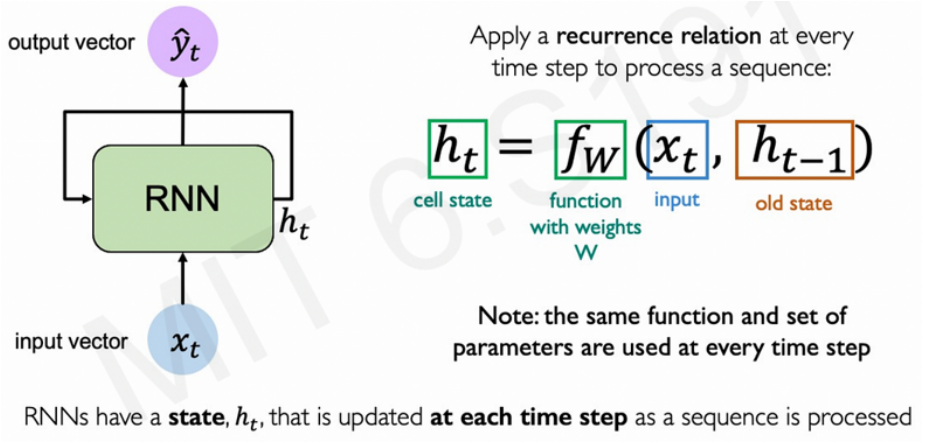
Temporal nature of language: Language is a sequence that unfolds over time (e.g. conversation flow, news feed, twitter stream, etc.) $\longrightarrow$ We don't always have access to the full input text.

Algorithms like HMM, neural networks allow access to all aspects of the input at a time.

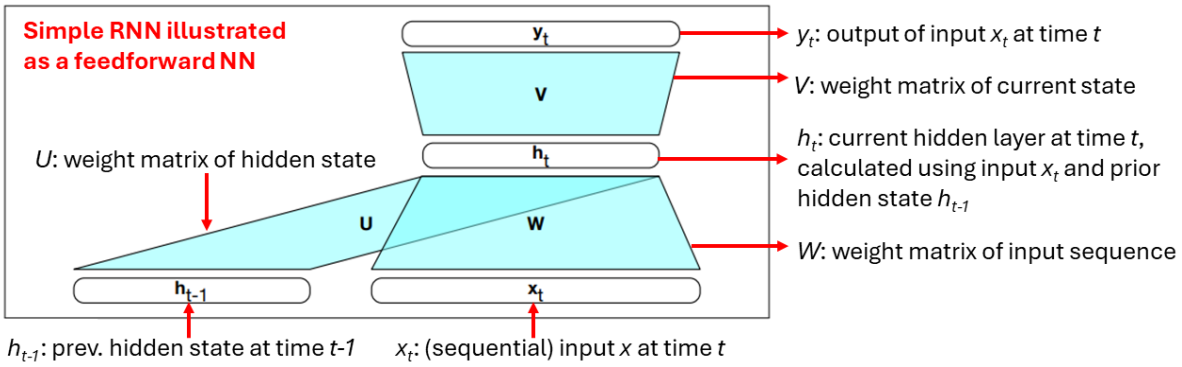
$\longrightarrow$ Recurrent neural networks (RNN) and its variant LSTM can deal directly with the sequential language without having to use fixed-size windows.

5.2 Recurrent Neural Network

RNN is a neural network where each iteration produces a memory state in addition to the output. The memory state is fed into the input of the next iteration as a continued memory which will affect the output.



The hidden layer acts as a **memory** that contains the encoding of earlier processing back to the beginning of the input sequence. This can inform decisions made by the model later on in time **without imposing a fixed-length limit** on the prior context.



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

$\mathbf{h}_0 \leftarrow 0$
for $i \leftarrow 1$ **to** LENGTH(**x**) **do**
  $\mathbf{h}_i \leftarrow g(\mathbf{U}\mathbf{h}_{i-1} + \mathbf{W}\mathbf{x}_i)$
  $\mathbf{y}_i \leftarrow f(\mathbf{V}\mathbf{h}_i)$
return y

Let the input, hidden, and output layer have dimensions of d_{in}, d_h, and d_{out} respectively

Step 1: The initial state, h_0, is initialized to 0.

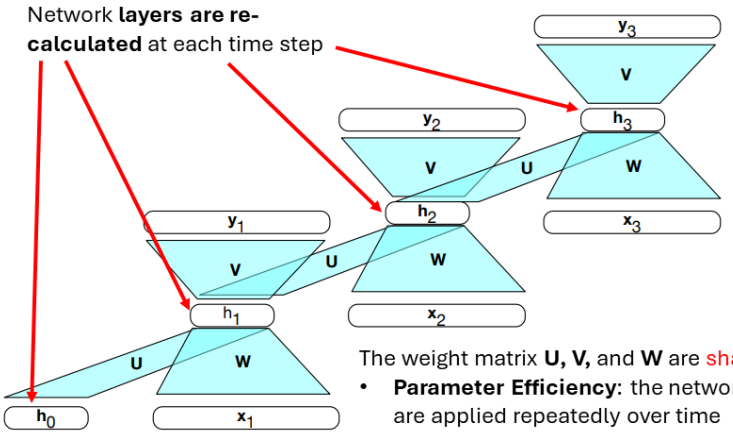
Step 2: For every element x_i at time step i of the input sequence x:

1. Update the current state $\mathbf{h_i}$ using $\mathbf{h_i} \longleftarrow g(\mathbf{U}\mathbf{h}_{i-1} + \mathbf{W}\mathbf{x}_i)$, where:
   - $\mathbf{U} \in \mathbb{R}^{d_h \times d_h}$ is the weight matrix governing the transformation of the previous hidden state $\mathbf{h_{i-1}}$ into the current internal state $\mathbf{h_i}$
   - $\mathbf{W} \in \mathbb{R}^{d_h \times d_{in}}$ is the weight matrix governing the transformation of the current input x_i into the new internal state $\mathbf{h_i}$
   - g is an activation function (tanh/ReLU) applied element-wise to the result of the weighted sum.
2. Compute the output at time step i using $\mathbf{y_i} \longleftarrow f(\mathbf{V}\mathbf{h}_i)$ where:
   - $\mathbf{V} \in \mathbb{R}^{d_o ut \times d_h}$ is the weight matrix governing the transformation of the current state $\mathbf{h_i}$ into the output $\mathbf{y_i}$
   - f is an activation function applied element-wise to $\mathbf{h_i}$. The choice of activation function depends on the task (but typically softmax).

RNN Use Cases:

- Many-to-many (each hidden state h_i corresponds to an input sequence x_i, which combine to produce a corresponding output y_i): Translation, speech recognition
- Many-to-one (input sequence x_i at each hidden state h_i, the final hidden state at the final time step h_t produces a single output y): Name-entity recognition, document classification, sentiment analysis
- One-to-many (only one input x, combined with different hidden states h_i to produce corresponding outputs y_i): Image captioning, video description

RNN reusing weight vectors



The weight matrix **U, V,** and **W** are shared across time:

- **Parameter Efficiency**: the network only learns a single set of params that are applied repeatedly over time
- **Modeling Variable-Length Sequences**: the model can handle sequences of various lengths without changing its architecture or # of params.
- **Efficient Backpropagation Through Time**: weight sharing simplifies the computation of gradients during backpropagation through time (BPTT), making training RNNs more tractable and effective

Advantages	Disadvantages
Suited for modeling sequential data with temporal dependencies, e.g. time series, text, audio	Basic RNN have limited memory, reducing their ability to capture dependencies in sequences with large time gaps
Weight sharing reduces # of parameters in the model and allows efficient learning of temporal patterns, leading to parameter efficiency and better generalization	Can suffer from the vanishing and exploding gradient problems (gradients become too small or too large during training), causing difficulties in learning long-term dependencies.
Can handle input sequences whose length vary	RNNs can be computationally expensive to train
Can retain information from previous time steps and incorporate it into the current prediction or output	

5.3 Long Short-term Memory (LSTM)

5.3.1 Limitations of RNN

#1: Hidden layers performing 2 tasks simultaneously	#2: Vanishing gradients
The hidden layers and the weights determining the values in the hidden layers have to perform 2 tasks: • Provide information useful for the current situation • Update & carry forward information required for future decisions → Weight sharing → The same transformations are applied at each step. The internal state is updated recursively based on the prev. state and current input → The ability to retain memory is constrained	The hidden layer at time t contributes to the loss at the next time step t+1 since it takes part in that calculation → During the backward pass of training, the hidden layers are subjected to repeated multiplications → Gradients are gradually driven to 0 → Vanishing gradient = useful info is not carried forward

5.3.2 RNN Gradient Flow

Let W_{hh} be the weight matrix controlling the transition from the previous hidden state h_{i-1} to the current hidden state h_i. This is similar to the matrix $\mathbf{U}$ in the prior section.

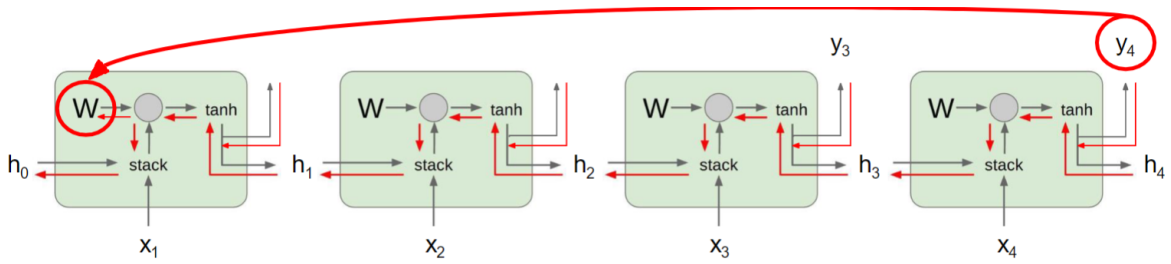
Let W_{xh} be the weight matrix controlling the transition from the current input x_i to the current hidden state h_i. This is similar to the matrix $\mathbf{W}$ in the prior section.

Using tanh as the activation function, the formula for the current hidden state h_t of time step t is:

$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ &= \tanh\left(\begin{pmatrix} W_{hh} & W_{xh} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \end{aligned} \tag{16}$$

Thus, in the backpropagation process, the derivative of h_t with respect to h_{t-1} is:

$$\frac{\partial h_t}{\partial h_{t-1}} = \tanh'\left(W_{hh}h_{t-1} + W_{xh}x_t\right) W_{hh} \text{ , applying the chain rule to (16)} \tag{17}$$



Backprogating by taking the derivative of the output L (corresp. to input x_i) with respect to W.

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W} \tag{18}$$

Continue the process. Applying the chain rule to take the derivative of the last output at time step T with respect to W:

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_t}{\partial h_T} \times \frac{\partial h_t}{\partial h_{t-1}} \times \ldots \times \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial h_1}{\partial W} \tag{19}$$

Substitute (17) into (19):

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_t}{\partial h_T} \left(\prod_{t=2}^{T} \tanh'(W_{hh}h_{t-1} + W_{xh}x_t) \right) W_{hh}^{T-1} \times \frac{\partial h_1}{\partial W} \tag{20}$$

Since the tanh activation function always squashes the value between the range -1 and 1, this leads to the problem of **vanishing gradient** where the gradients become vanishingly small as they are propagated backward through early training steps.

$\longrightarrow$ Gradients can't provide meaningful information for parameter updates. The RNN will have difficulties learning long-term dependencies in the data.

5.3.3 LSTM

Central Idea: A **memory cell** which can maintain its state over time, consisting of explicit memory (aka **cell state vector**) and **gating units** which regulate information flow into and out of the memory.

Order of Operation

Input Gate i:

The input gate determines **how much new information should be stored in the memory cell at the current time step**. The formula for the input gate vector is:

$$i_t = \sigma(U_i h_{t-1} + W_i x_t) \tag{21}$$

The input gate takes in the current input x_t, previous hidden state h_{t-1}. It uses a sigmoid σ activation function, and outputs a vector i_1 whose values are in $[0, 1]$, and each represents the degree to which the corresponding element in the cell state should be updated.

Gate gate g:

The gate gate g_t contains the proposed new information to be added to the memory cell at the current time step. The formula for g_t is:

$$i_t = \tanh(U_g h_{t-1} + W_g x_t) \tag{22}$$

The gate gate takes in the current input x_t, previous hidden state h_{t-1}, and uses a tanh activation function to squash the output between $[-1, 1]$. The purpose of the g_t vector is to provide a candidate update that could potentially be added to the memory cell. However, whether this is actually added to the cell state depends on the input gate's decision.

Forget gate

The forget gate f_t determines which information in the memory cell should be retained or forgotten from the previous time step. The formula for f_t is:

$$f_t = \sigma(U_f h_{t-1} + W_f x_t) \tag{23}$$

The forget gate takes in the current input x_t, previous hidden state h_{t-1}. It uses a sigmoid σ activation function, and outputs a vector i_1 whose values are in $[0, 1]$. Each value represents the degree to which the corresponding element in the cell state should be retained or forgotten.

Element-wise multiplications

f_t: the forget vector, which information in the memory cell should be retained or forgotten from the previous time step.

c_{t-1}: the previous cell state, which contains information from the previous time step.

$\longrightarrow f_t \odot c_{t-1}$: which elements of the previous cell state should be retained or forgotten based on the forget gate's decision. Elements of the c_{t-1} that correspond to f_t values close to 1 will be retained, while elements corresponding to f_t values close to 0 will be forgotten.

g_t: proposed information update vector, which represents the proposed new information to be added to the memory cell at the current time step.

i_t: input vector, which determines how much of the new information proposed by the candidate update vector should be added to the memory cell at the current time step.

$\longrightarrow g_t \odot i_t$: which elements of the candidate update vector should be added to the memory cell based on the input gate's decision. Elements of g_t that correspond to i_t values close to 1 will have a larger influence on the cell state c_t, while elements corresponding to i_t values close to 0 will have a smaller influence.

Update the current memory cell c_t

$$c_t = g_t \odot i_t + f_t \odot c_{t-1} \tag{24}$$

c_t combines the selective retention or forgetting of information from the previous state with the selective incorporation of new information proposed by the input gate.

$\longrightarrow c_t$ can retain relevant information from the previous state and incorporate new information based on the input gate's decision.

Output gate o_t:

$$o_t = \sigma(U_o h_{t-1} + W_o x_t) \tag{25}$$

The output gate vector o_t controls how much information from the cell state should be passed to the output at the current time step. It uses a sigmoid activation function to produce values between $[0, 1]$. Each element of this vector represents the degree to which the corresponding element in c_t should be output.

The current state h_t:

$$h_t = o_t \odot \tanh(c_t) \tag{26}$$

h_t is the current internal state of the LSTM cell, which may also serve as the final output of the LSTM layer or be passed as input to next cells in the network.

This process allows the LSTM cell to selectively pass relevant information from the cell state to the output, producing meaningful outputs based on the input and previous state.

Aspect	Advantages	Disadvantages
Long-Term Dependencies	Good at capturing long-term dependencies in sequential data. Suitable for tasks involving contexts or dependencies over a long timespan.	May still struggle with capturing very long-term dependencies
Mitigating Vanishing	Address the vanishing gradient problem by introducing the memory cell and gate mechanisms	May still face issues related to exploding gradients, but this is less common
Information Retention	Can selectively retain/forget information over time through gate mechanisms, allowing LSTM to maintain relevant information.	The forget gate is good at retaining relevant information, but may sometimes forget important details.
Versatility	Can be applied to a wide range of NLP tasks: time series analysis, speech recognition, other sequential data tasks.	Training and fine-tuning LSTMs may be computationally expensive
Handling Irregular	Can handle irregular time intervals in sequential data. Suitable for tasks where the timing of events varies	Designing and training LSTMs for irregularities may require careful preprocessing of temporal patterns

6 Week 8 - Transformer and BERT

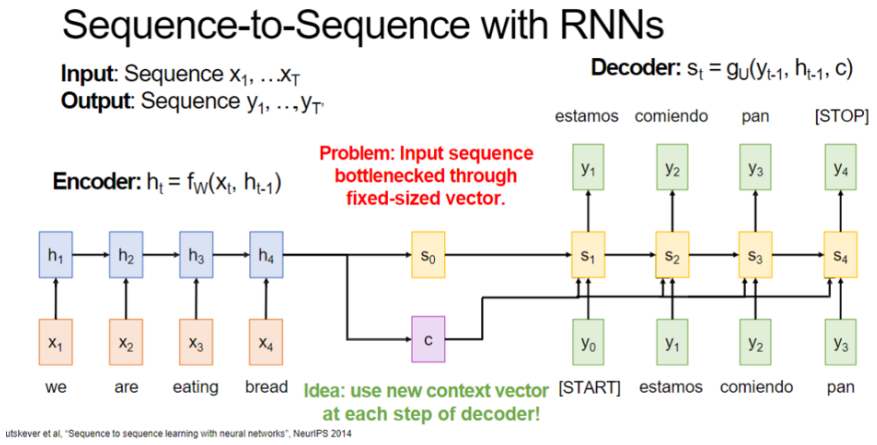
6.1 Sequence to Sequence

The input token x_i are read one at a time. The final hidden state of the cell will become c.

Problem is that it's difficult to compress an arbitrary-length sequence into a single fixed-size vector.

Solution is that the encoder will usually consist of stacked LSTMs: a series of LSTM "layers" where each layer's outputs are the input sequence to the next layer. The final layer's hidden state will be used as c.

The Seq2Seq encoder will **process the input sequence in reverse**: The last thing that the encoder sees will (roughly) correspond to the first thing that the model outputs. $\longrightarrow$ easier for the decoder to start on the output.



Decoder will initialize the hidden state of its first layer with the context vector c from encoder and use the context of the input to generate an output.

A special token, usually $< EOS >$, will signify the start of output generation. The RNNs layers are run, one after the other, following up with an activation on the final layer's output to generate the first output word. This word is passed to the first layer, then generation is repeated.

6.2 Sequence to Sequence with Attention

Problem with using the final RNN hidden state as the single context vector for Seq2Seq:

Often, different parts of an input have different levels of significance. Moreover, different parts of the output may even consider different parts of the input "important."

Attention mechanisms make use of this observation by providing the decoder network with a look at the entire input sequence at every decoding step. The decoder can then decide what input words are important at any point in time.

6.2.1 Encoder

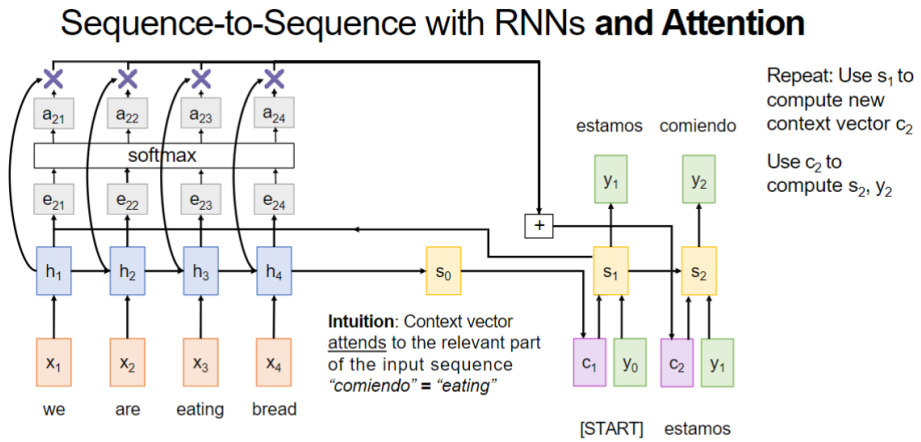
Input is a sequence of words $x_1, .., x_n$. Target is a sequence of words $y_1, ...y_m$.

$h_1, ..., h_n$ is the hidden vectors representing the input sentence. These capture the contextual representation of each word in the input sentence.

$$h_t = f_W(x_t, h_{t-1}) \tag{27}$$

The final hidden state h_n produces the initial decoder state s_0.

6.2.2 Decoder



Goal: To compute the hidden state s_i using a formula of the form:

$$s_t = g_U(s_{t-1}, y_{t-1}, c_t) \tag{28}$$

where s_{t-1} is the previous hidden vector, y_{t-1} s the generated word at the previous step, c_t is the context vector that captures the context from the original sentence relevant for the i-th decoding time step (different from standard Seq2Seq where there's only 1 context vector).

For each hidden vector from the original sentence h_i, compute a score:

$$e_{t,i} = f_{att}(s_{t-1}, h_i) \tag{29}$$

where f_{att} is any function with values in $\mathbb{R}$ like a layer of fully connected NN or an MLP. Now we have a sequence of scalar values $e_{t,1}, ..., e_{t,n}$. These compute the compatibility between the decoder hidden state s_t and the encoder hidden state h_i.

Normalize these scores into a vector $\alpha_{t,i} = (\alpha_{t,1}, ..., \alpha_{t,n})$ using a softmax layer.

The softmax function normalizes the scores across all encoder hidden states, ensuring that the attention weights sum up to 1 and represent a valid probability distribution. These attention weights indicate how much attention should be assigned to each encoder hidden state when computing the context vector.

$$\alpha_{t,i} = softmax(e_{t,i}) \tag{30}$$

Then, compute the context vector c_t as the weighted average of the hidden vectors from the original sentence:

$$c_t = \sum_{i=1}^{n} \alpha_{t,i} h_i \tag{31}$$

This vector c_t captures the relevant context information from the original sentence for the t time step of the decoder that the decoder should focus on when generating the next output token. It is a weighted combination of the encoder hidden states, with the weights determined by the attention mechanism.

6.3 Transformer

6.3.1 Self-attention

Attention so far (in seq2seq architectures):
In the ***decoder*** (which has access to the complete input sequence), compute attention weights over ***encoder*** positions that depend on each ***decoder*** position

Self-attention:
If the ***encoder*** has access to the complete input sequence, we can also compute attention weights over ***encoder*** positions that depend on each ***encoder*** position
self-attention:
For each ~~*decoder*~~ ***encoder*** position t...,
...Compute an attention weight for each *encoder* position s
...Renormalize these weights (that depend on t) w/ softmax to get a new weighted avg. of the input sequence vectors

Aspect	Hidden Markov Model (HMM)	Maximum Entropy Markov Model (MEMM)	Conditional Random Field (CRF)
Type of Model	Generative Model	Discriminative Model	Discriminative Model
State Representation	Hidden states represent latent variables	Hidden states represent observed variables	Hidden states represent observed variables
Transition Probabilities	Governed by transition probabilities	Modeled explicitly using feature weights	Modeled explicitly using feature weights
Observation Probability	Modeled as emission probabilities	Modeled directly from input features	Modeled directly from input features
Feature Representation	Does not incorporate explicit feature information	Features are explicitly defined and incorporated	Features are explicitly defined and incorporated
Training Algorithm	Baum-Welch algorithm (Expectation-Maximization)	Maximum likelihood estimation (e.g., gradient descent)	Maximum likelihood estimation (e.g., gradient descent)
Decoding Algorithm	Viterbi algorithm	m search or other decoding algorithms	Various decoding algorithms (e.g., Viterbi,
Sequential Dependencies	Accounts for sequential dependencies naturally	Explicitly models sequential dependencies	Explicitly models sequential dependencies
Scalability	May struggle with large feature spaces or complex dependencies	Can handle large feature spaces and complex dependencies effectively	Can handle large feature spaces and complex dependencies effectively
Memory Usage	Typically requires less memory	May require more memory due to feature representation	May require more memory due to feature representation
Usage	Commonly used in speech recognition, part-of-speech tagging, etc.	Widely used in natural language processing tasks such as named entity recognition, chunking, etc.	Widely used in natural language processing tasks such as named entity recognition, sequence labeling, etc.

Aspect	Recurrent Neural Network (RNN)	Long Short-Term Memory (LSTM)	Transformer
Architecture	Sequential processing with recurrent connections	Sequential processing with gated recurrent connections	Attention-based architecture with self-attention mechanism
Handling of Sequential Data	Suitable for sequential data with short-range dependencies	Effective for long-range dependencies and mitigates vanishing gradient problem	Highly effective for capturing long-range dependencies and parallel computation

Continued on next page

(Continued)

Memory Management	Limited memory due to vanishing gradient problem	Mitigates vanishing gradient problem, maintains long-term memory	Captures long-range dependencies efficiently, limited only by memory capacity
Parallelization	Challenging to parallelize due to sequential nature	Partially parallelizable through mini-batch processing	Fully parallelizable computation, efficient on hardware accelerators
Learning Representations	Captures temporal dependencies in sequential data	Learns to retain and update information over long sequences	Learns global dependencies through self-attention mechanism
Training and Inference	Slower training and inference, especially on long sequences	Efficient training and inference, suitable for long sequences	Fast training and inference, scalable to long sequences
Handling of Variable-Length Sequences	Handles variable-length sequences naturally	Handles variable-length sequences, maintains context over time	Handles variable-length sequences efficiently, maintains global context
Application	Sequential data modeling (e.g., time series prediction, language modeling)	Sequential tasks with long dependencies (e.g., speech recognition, text generation)	Natural language processing tasks (e.g., machine translation, text classification, language modeling)