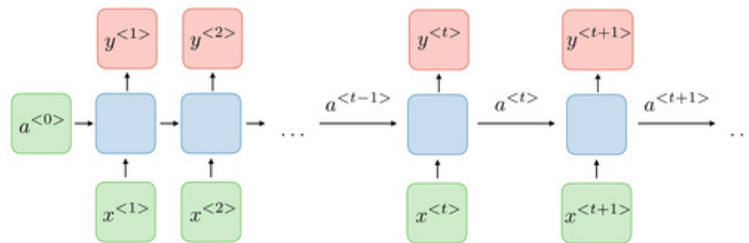


VIP Cheatsheet: Recurrent Neural Networks



Overview

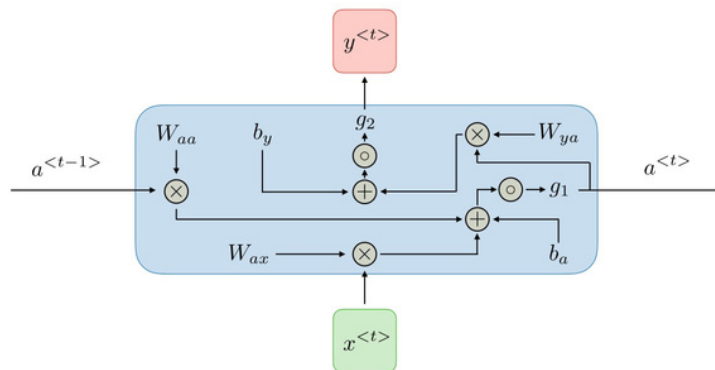
Architecture of a traditional RNN – Recurrent neural networks, also known as RNNs, are a class of neural networks that allow previous outputs to be used as inputs while having hidden states. They are typically as follows:



For each timestep t , the activation $a^{<t>}$ and the output $y^{<t>}$ are expressed as follows:

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \text{ and } y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

where $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients that are shared temporally and g_1, g_2 activation functions



The pros and cons of a typical RNN architecture are summed up in the table below:

Advantages	Drawbacks
- Possibility of processing input of any length	- Computation being slow
- Model size not increasing with size of input	- Difficulty of accessing information historical information
- Computation takes into account from a long time ago	- Cannot consider any future input
- Weights are shared across time for the current state	

Applications of RNNs – RNN models are mostly used in the fields of natural language processing and speech recognition. The different applications are summed up in the table below:

Type of RNN	Illustration	Example
One-to-one Traditional neural network $T_x = T_y = 1$		
One-to-many Music generation $T_x = 1, T_y > 1$		
Many-to-one Sentiment classification $T_x > 1, T_y = 1$		
Many-to-many Name entity recognition $T_x = T_y$		
Many-to-many Machine translation $T_x = 6, T_y = 6$		

Loss function – In the case of a recurrent neural network, the loss function

L of all time

steps is defined based on the loss at every time step as follows:

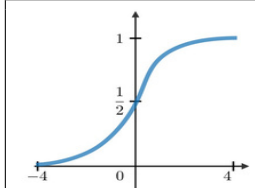
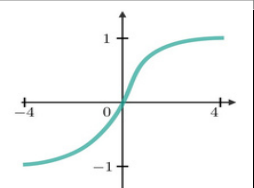
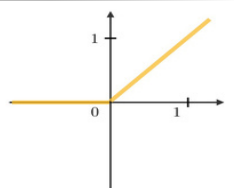
$$L(\hat{y}, y) = \sum_{t=1}^T L(\hat{y}_{<t>, y_{<t>})$$

Backpropagation through time – Backpropagation is done at each point in time. At timestep T , the derivative of the loss L with respect to weight matrix W is expressed as follows:

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

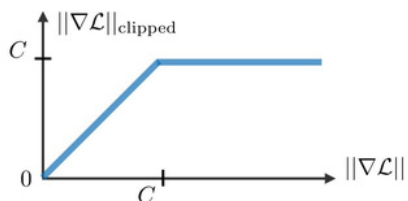
Handling long term dependencies

Commonly used activation functions – The most common activation functions used in RNN modules are described below:

	Sigmoid Tanh RELU	
$\sigma(z) = \frac{1}{1 + e^{-z}}$	$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$\text{ReLU}(z) = \max(0, z)$
		

Vanishing/exploding gradient – The vanishing and exploding gradient phenomena are often encountered in the context of RNNs. The reason why they happen is that it is difficult to capture long term dependencies because of multiplicative gradient that can be exponentially decreasing/increasing with respect to the number of layers.

Gradient clipping – It is a technique used to cope with the exploding gradient problem sometimes encountered when performing backpropagation. By capping the maximum value for the gradient, this phenomenon is controlled in practice.



Types of gates – In order to remedy the vanishing gradient problem, specific gates are used in some types of RNNs and usually have a well-defined purpose. They are usually noted Γ and are equal to:

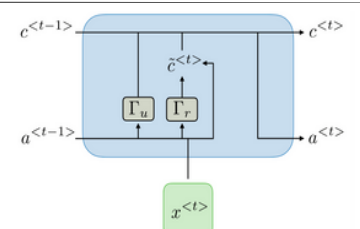
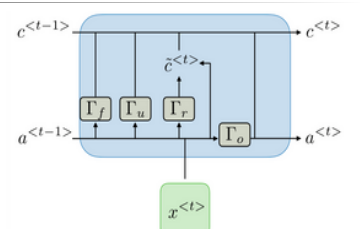
$$\Gamma = \sigma(Wx_{<t>} + Ua_{<t-1>} + b)$$

where W, U, b are coefficients specific to the gate and σ is the sigmoid function. The main ones are summed up in the table below:

	Type of gate	Role Used in
Update gate Γ_u	How much past should matter now?	GRU, LSTM
Relevance gate Γ_r	Drop previous information?	GRU, LSTM
Forget gate Γ_f	Erase cell or not?	LSTM
Output gate Γ_o	How much to reveal of a cell?	LSTM

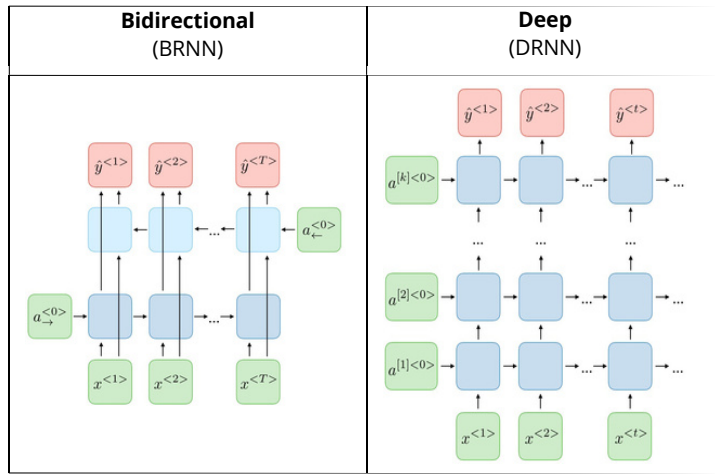
GRU/LSTM – Gated Recurrent Unit (GRU) and Long Short-Term Memory units (LSTM) deal with the vanishing gradient problem encountered by traditional RNNs, with LSTM being a generalization of GRU. Below is a table summing up the characterizing equations of each architecture:

Gated Recurrent Unit Long Short-Term Memory

(GRU)	(LSTM)
$\tilde{c}_{<t>} = \tanh(W[\Gamma_u a_{<t-1>} + c_{<t-1>} + \Gamma_r a_{<t>} + x_{<t>}] + b)$	$\tilde{c}_{<t>} = \tanh(W_c[\Gamma_u a_{<t-1>} + c_{<t-1>} + \Gamma_r a_{<t>} + x_{<t>}] + b_c)$
$c_{<t>} = \Gamma_u \tilde{c}_{<t>} + (1 - \Gamma_u) c_{<t-1>}$	$c_{<t>} = \Gamma_u \tilde{c}_{<t>} + (1 - \Gamma_u) c_{<t-1>}$
$a_{<t>} = \Gamma_o c_{<t>}$	$a_{<t>} = \Gamma_o c_{<t>}$
	

Remark: the sign \odot denotes the element-wise multiplication between two vectors.

Variants of RNNs – The table below sums up the other commonly used RNN architectures:



Learning word representation

In this section, we note V the vocabulary and $|V|$ its size.

Representation techniques – The two main ways of representing words are summed up in the table below:

1-hot representation Wordembedding

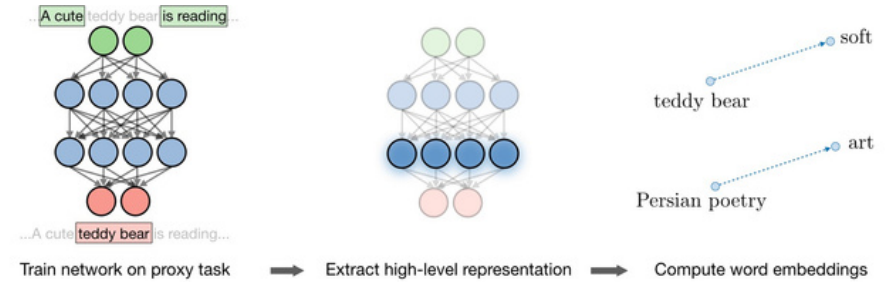
-Notedew - Naive approach, no similarity information	- Takes into account words similarity

Embedding matrix – For a given word w , the embedding matrix E is a matrix that maps its 1-hot representation ow to its embedding ew as follows:

$$ew = Eow$$

Remark: learning the embedding matrix can be done using target/context likelihood models.

Word2vec – Word2vec is a framework aimed at learning word embeddings by estimating the likelihood that a given word is surrounded by other words. Popular models include skip-gram, negative sampling and CBOW.



Skip-gram – The skip-gram word2vec model is a supervised learning task that learns word embeddings by assessing the likelihood of any given target word t happening with a context word c . By noting θ_t a parameter associated with t , the probability $P(t|c)$ is given by:

$$\sum \exp(\theta_t e_c) P(t|c) = \frac{e_c^T \sum \exp(\theta_t e_c)}{\sum \exp(\theta_t e_c)} = 1$$

Remark: summing over the whole vocabulary in the denominator of the softmax part makes this model computationally expensive. CBOW is another word2vec model using the surrounding words to predict a given word.

Negative sampling – It is a set of binary classifiers using logistic regressions that aim at assessing how a given context and a given target words are likely to appear simultaneously, with the models being trained on sets of k negative examples and 1 positive example. Given a context word c and a target word t , the prediction is expressed by:

$$P(y = 1 | c, t) = \sigma(\theta^T e_c e_t)$$

Remark: this method is less computationally expensive than the skip-gram model.

GloVe – The GloVe model, short for global vectors for word representation, is a word embedding technique that uses a co-occurrence matrix X where each X_{ij} denotes the number of times that a target i occurred with a context j . Its cost function J is as follows:

$$\sum |V| J(\theta) = \frac{1}{2} \sum_i \sum_j (X_{ij} + b_i^2 + b_j^2 - \log(X_{ij}))^2$$

here f is a weighting function such that $X_{ij} = 0 \Rightarrow f(X_{ij}) = 0$.

$$\Rightarrow f(X_{ij}) = 0$$

Given the symmetry that and play in (final) e_{θ} this model, the final word embedding ew is given by:

$$ew = \frac{1}{\sum_i X_{ij}} \sum_i X_{ij} e_i$$

Remark: pretable.

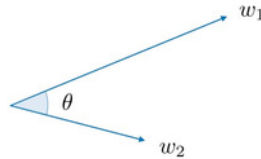
the individual components of the learned word embeddings are not necessarily inter-

Comparing words

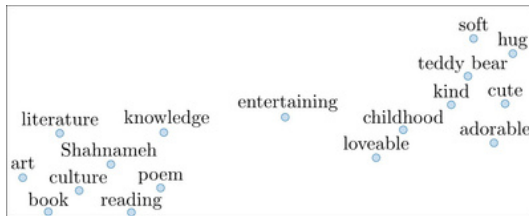
r Cosine similarity – The cosine similarity between words w_1 and

$$\text{similarity} = \frac{w_1 \cdot w_2}{\|w_1\| \|w_2\|} = \cos(\theta)$$

Remark: θ is the angle between words w_1 and w_2 .



r t-SNE – t-SNE (t-distributed Stochastic Neighbor Embedding) is a technique aimed at reducing high-dimensional embeddings into a lower dimensional space. In practice, it is commonly used to visualize word vectors in the 2D space.



Language model

r Overview – A language model aims at estimating the probability of a sentence $P(y)$.

r n-gram model – This model is a naive approach aiming at quantifying the probability that an expression appears in a corpus by counting its number of appearance in the training data.

r Perplexity – Language models are commonly assessed using the perplexity metric, also known as PP, which can be interpreted as the inverse probability of the dataset normalized by the number of words T . The perplexity is such that the lower, the better and is defined as follows:

$$PP = \frac{1}{T} \sum_{t=1}^T \frac{1}{P(y_t | y_{1:t-1})}$$

Remark: PP is commonly used in t-SNE.

Machine translation

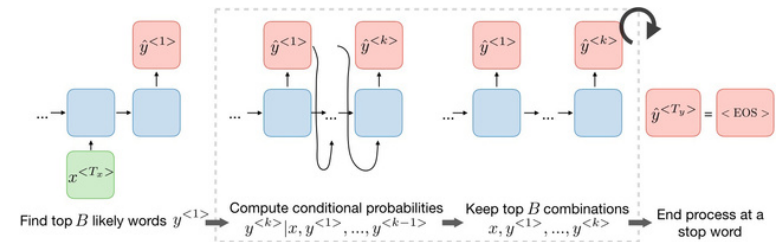
r Overview – A machine translation model is similar to a language model except it has an encoder network placed before. For this reason, it is sometimes referred as a conditional language model. The goal is to find a sentence y such that:

$$y = \arg \max_{y < 1 > \dots, y < T_y >} P(y < 1 > \dots, y < T_y > | x)$$

w_2 is expressed as follows:

r Beam search – It is a heuristic search algorithm used in machine translation and speech recognition to find the likeliest sentence y given an input x .

- Step 1: Find top B likely words $y < 1 >$
- Step 2: Compute conditional probabilities $y < k >$
 $| x, y < 1 > \dots, y < k-1 >$
- Step 3: Keep top B combinations $x, y < 1 > \dots, y < k >$



Remark: if the beam width is set to 1, then this is equivalent to a naive greedy search.

r Beam width – The beam width B is a parameter for beam search. Large values of B yield to better result but with slower performance and increased memory. Small values of B lead to worse results but is less computationally intensive. A standard value for B is around 10.

r Length normalization – In order to improve numerical stability, beam search is usually applied on the following normalized objective, often called the normalized log-likelihood objective, defined as:

$$\sum_{t=1}^T \log p(y_t | x, y_{1:t-1})$$

Remark: the parameter α can be seen as a softener, and its value is usually between 0.5 and 1.

r Error analysis – When obtaining a predicted translation

\hat{y} that is bad, one can wonder why we did not get a good translation y^* by performing the following:

Case	$P(y^* x) > P(\hat{y} x)$		
Remedies	Increase beamwidth - Regularize - Get more data		

r Bleu score – The bilingual evaluation understudy (bleu) score quantifies how good a machine translation is by computing a similarity score base

(don't use n-gram) precision. It is defined as follows: $\text{bleu score} = \exp(\sum_{n=1}^4 p_n)$

where p_n is the bleu score on n -gram only defined as follows:

countclip(n-gram)

= n-gram $\in \hat{y}^n$

count(n-gram)

n-gram

$\in \hat{y}$

Σ

Σ

Remark: a brevity penalty may be applied to short predicted translations to prevent an artificially inflated bleu score.

Attention

r Attention model – This model allows an RNN to pay attention to specific parts of the input that is considered as being important, which improves the performance of the resulting model in practice. By noting $\alpha_{\langle t, t' \rangle}$ the amount of attention that the output $y_{\langle t \rangle}$ should pay to the activation $a_{\langle t' \rangle}$ and $c_{\langle t \rangle}$ the

$\Sigma_{\text{context at time } t} \text{ we have: } c_{\langle t \rangle} = \alpha_{\langle t, t' \rangle} a_{\langle t' \rangle} \Sigma_{\alpha_{\langle t, t' \rangle} = 1} t'$

Remark: the attention scores are commonly used in image captioning and machine translation.



A cute teddy bear is reading Persian literature



A cute teddy bear is reading Persian literature

r Attention weight – The amount of attention that the output $y_{\langle t \rangle}$ should pay to the activation is $\alpha_{\langle t \rangle}$ given by $\alpha_{\langle t \rangle}$ computed as follows:

$\alpha_{\langle t, t' \rangle}$

$\Sigma e^{\langle t' \rangle} = \exp(\langle t \rangle) T x e^{\langle t, t' \rangle} \exp(\langle t \rangle) t' = 1$

Remark: computation complexity is quadratic with respect to Tx .

? ? ?