Neural Language Model

Neural LMs, RNN, LSTM and GRU

Ramaseshan Ramachandran

CURSE OF DIMENSIONALITY

In the probabilistic language model, we took advantage of the idea - ***temporally closer words in the word sequence are statistically more dependent***

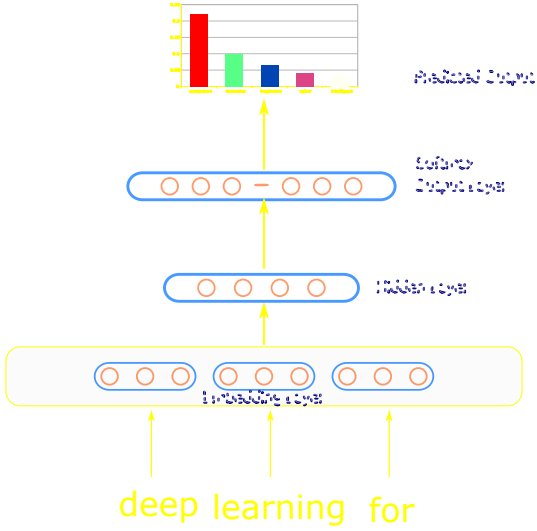
- A fundamental problem that makes language modelling and other learning problems difficult is the curse of dimensionality
- It is particularly obvious in the case when one wants to model the joint distribution between many discrete random variables
- If one wants to estimate the joint probability distribution of n-grams words in a language with a $|V|$ words as vocabulary, then we need to estimate a maximum of $|V|^n$ free parameters

DISADVANTAGES OF PLM

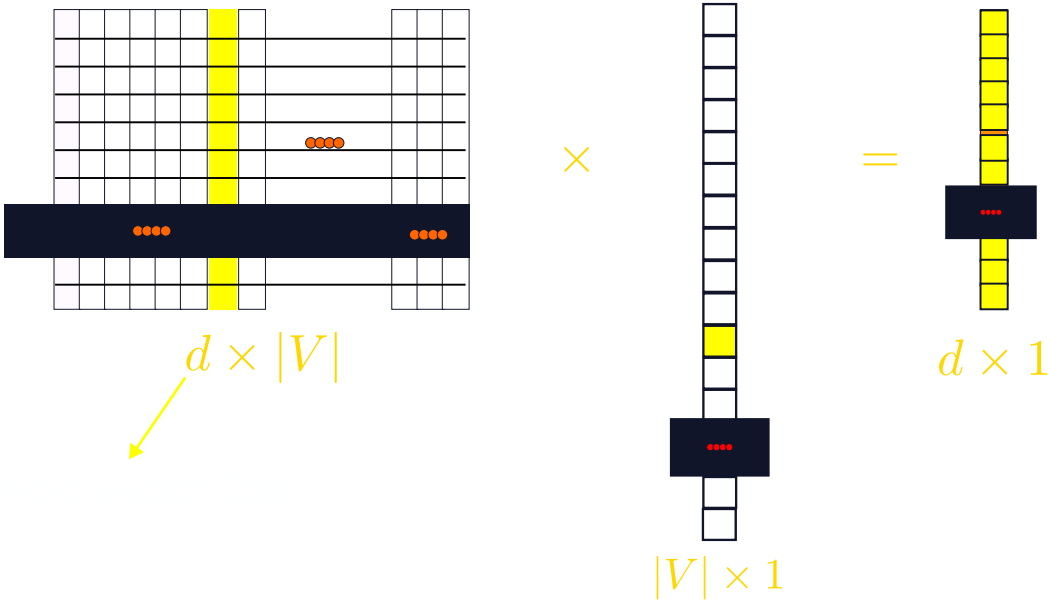
- Scalability wrt increase in context window size
- Markov assumption. It is difficult to predict a word in a longer sequence

  As he is from Tamil Nadu, it is most probable that his mother tongue is ....
- Free parameter size increases exponentially with the context window size
- The context words closer to the word to be estimated are given more importance
- It does not use any semantic or syntactic similarity for the estimation

ANN FOR LM



EMBEDDING LAYER CREATION



NEURAL LM MODEL

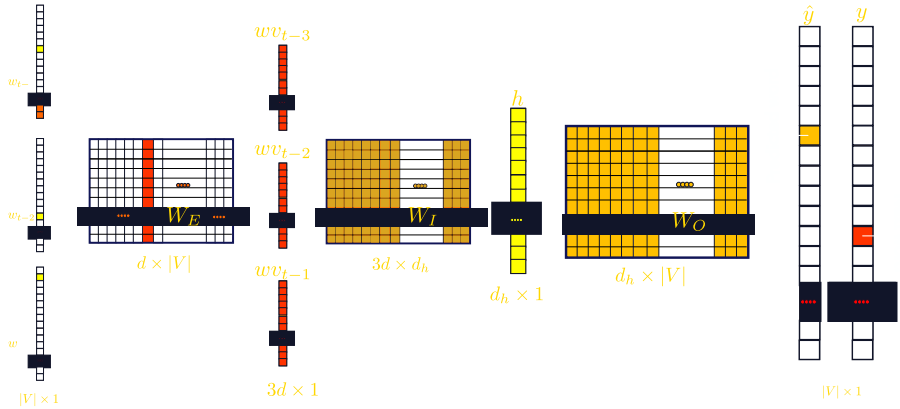


Figure: Adopted from https://web.stanford.edu/~jurafsky/slp3/7.pdf - Chapter 7 -[1]

FORWARD PASS

- Forward pass is used to compute the probability distribution over the possible outputs.
- Word embedding matrix is used
- One-hot vector is used to get the index of the input word to get the corresponding word vector from the embedding matrix
- Hidden layer values are computed using word vectors of the context words and the input weights W_{I}
- The output $\hat{\mathrm{y}}$ is computed using the hidden layer and the output weights W_{O}
- Embedding weights W_{E} is shared among all the context words
- h depends on W_{E} and W_{I}
- The parameters of this model are $W_{\mathrm{E}}, W_{\mathrm{I}}, \mathrm{h}, W_{\mathrm{O}}$

LOSS FUNCTION

The loss function measures how much the output $\hat{y}$ differs from the true observation, y. The loss function $\mathbb{L}$ is a maximum likelihood estimate with respect to the correct output y. The parameters of the model are chosen to maximize this probability given the input-output relationships.

$$\hat{y} = p(y|\text{context words}) \tag{1}$$

We use cross-entropy as a loss function when the estimated result and the target are probability distributions. If there is only one correct outcome (binary case), this is a Bernoulli distribution:

$$p(x) = \begin{cases} p & \text{if } x = 1 \\ 1-p & \text{if } x = 0 \end{cases} \tag{2}$$

Now, $p(y|\text{context words})$ in equation (1) can be written as

$$p(y|\text{context words}) = \hat{y}^y (1-\hat{y})^{1-y} \tag{3}$$

Take the negative log of equation 3 to get the cross-entropy loss:

$$\mathbb{L} = -(y \log \hat{y} + (1-y) \log(1-\hat{y})) \tag{4}$$

Taking log on both sides of equation(3)

$$\begin{aligned}\log(p(y|\text{context words})) &= \log\left(\hat{y}^{y}(1-\hat{y})^{1-y}\right) \\ &= y\log(\hat{y}) + (1-y)\log(1-\hat{y}) \end{aligned} \tag{5}$$

By incorporating equation(1), the loss function will be a cross entropy loss function which we need to minimize during the learning process

$$\mathscr{E}(W_E, W_I, h, W_O) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y}) \tag{6}$$

Replacing $\hat{y} = f(w_O.h + b)$ in equation we get

$$\mathscr{E}(W_E, W_I, h, W_O) = -y\log(f(w_O.h + b)) - (1-y)\log(1-f(w_O.h + b)) \tag{7}$$

SIGNIFICANCE OF THE CROSS-ENTROPY LOSS FUNCTION

As we are dealing with one-hot vector as the true output, the equation(6) becomes

$$\begin{aligned}\mathscr{E}(W_{\mathrm{E}}, W_{\mathrm{I}}, \mathrm{h}, W_{\mathrm{O}}) &= -\mathrm{y}\log(\hat{\mathrm{y}}) \\ &= -\mathrm{y}\log(\mathrm{f}(\mathrm{w}_{\mathrm{O}}.\mathrm{h} + \mathrm{b})) \text{ or} \\ \mathrm{E}(W_{\mathrm{E}}, W_{\mathrm{I}}, \mathrm{h}, W_{\mathrm{O}}) &= -\sum_{\mathrm{d}=1}^{|\mathrm{V}|} \mathrm{y}_{\mathrm{d}}\log(\hat{\mathrm{y}}_{\mathrm{d}}) \end{aligned} \tag{8}$$

During the learning process, if

$$\mathscr{E} = \begin{cases} \text{very small, then the classifier is good - parameters are learned} \\ \text{large, then the classifier is bad or the entropy is large} \\ \text{- Lack of predictability - Needs more training} \end{cases}$$

Since $-\log(\hat{y}) \in (0, \infty)$ for $\hat{y} \in (0, 1)$, the loss function in (6), which is the cross-entropy between y and the estimate $\hat{y}$, ensures that the word to be predicted gets a high probability and the incorrect ones get low probability.
The idea is to

1. Learn the weights for all context and target words
2. Minimize the loss function $\mathscr{E}$
3. Generalize over all training samples (or maximize the input-output relationship)

DRAWBACKS OF TRADITIONAL ANN

- Memory-less and does not bother where the words and context come from
- Traditional Neural networks do not accept arbitrary input length
- Architecture is fixed with respect to the context window or input
- Every context is considered in isolation
- Some important tasks depend on the entire sequence of data
  $y(t+1) = f(x(t), x(t-1), x(t-2)...x(t-n))$

DRAWBACKS OF TRADITIONAL ANN

- Traditional Neural networks are not designed as a state machine
- Anything outside the context window has no impact on the decision being made
- Some NLP tasks require semantic modeling over the whole sentence
- Temporal dependencies are important and are not captured
- Do not or struggle to capture correlations between the temporal elements in sequences

SEQUENCE LEARNING

Sequence learning is the study of machine learning algorithms designed for applications that require sequential data or temporal data

- Helps in modeling complex temporal patterns in the data
- Captures the dependencies and correlations between the different elements in the sequence
- Maintains internal state that captures the context of the previous occurrences of words

OPTIMAL SEQUENCE LEARNING

- Associate each word in the vocabulary with a distributed feature vector.
- Allow the feature vectors to represent the joint probability of the n-gram sequences.
- Train the features as parameters to represent the sequence.

APPLICATIONS

- Named Entity Recognition
- Paraphrase detection - identifying semantically equivalent questions
- Language Generation
- Machine Translation
- Speech recognition
  - Wreck a nice beach or recognize speech
- Automatically generating subtitles for a video
- Spell Checking
- Predictive typing
- Chat-bots/Dialog understanding
- Generate missing text
- Correct Hand-written text

RECURRENT NEURAL NETWORK

- Sequential data prediction is considered as a key problem in machine learning and artificial intelligence
- Unlike images where we look at the entire image, we read text documents sequentially to understand the content.
- The likelihood of any sentence can be determined from everyday use of language.
- The earlier sequence of words (in terms of time) is important to predict the next word, sentence, paragraph or chapter
- If a word occurs twice in a sentence, but could not be accommodated in the sliding window, then the word is learned twice
- An architecture that does not impose a fixed-length limit on the prior context

RNN

- States are important in the reading exercise- the next state depends on the previous states
- In order to use the previous state, we need to store it or remember it
- Inherent ability to model sequential input
- Handle variable length inputs without the use of arbitrary fixed-sized windows
- Use its own output as input
- RNNs encode not only attributional similarities between words, but also similarities between pairs of words
  - Analogy - $\mathrm{Chennai} : \mathrm{Tamil} :: \mathrm{London} : \mathrm{English}$ or $\mathrm{go} : went :: \mathrm{run} : \mathrm{Ran}$ or $queen \approx \mathrm{king} - \mathrm{man} + woman$

SEQUENCE LEARNING

Sequence learning is the study of machine learning algorithms designed for applications that require sequential data or temporal data

- Helps in modeling complex temporal patterns in the data
- Captures the dependencies and correlations between the different elements in the sequence
- Maintains internal state that captures the context of the previous occurrences of words

RECURRENT NEURAL NETWORK

- Sequential data prediction is considered as a key problem in machine learning and artificial intelligence
- Unlike images where we look at the entire image, we read text documents sequentially to understand the content.
- The likelihood of any sentence can be determined from everyday use of language.
- The earlier sequence of words (in terms of time) is important to predict the next word, sentence, paragraph or chapter
- If a word occurs twice in a sentence, but could not be accommodated in the sliding window, then the word is learned twice
- An architecture that does not impose a fixed-length limit on the prior context

RNN

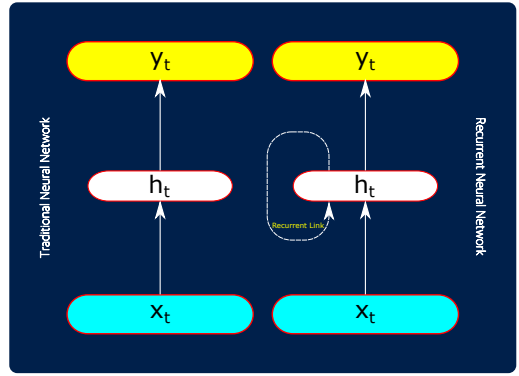
- States are important in the reading exercise- the next state depends on the previous states
- In order to use the previous state, we need to store it or remember it
- Inherent ability to model sequential input
- Handle variable length inputs without the use of arbitrary fixed-sized windows
- Use its own output as input
- RNNs encode not only attributional similarities between words, but also similarities between pairs of words
  - Analogy - $\mathrm{Chennai} : \mathrm{Tamil} :: \mathrm{London} : \mathrm{English}$ or $\mathrm{go} : went :: \mathrm{run} : \mathrm{Ran}$ or $queen \approx \mathrm{king} - \mathrm{man} + woman$

APPLICATIONS

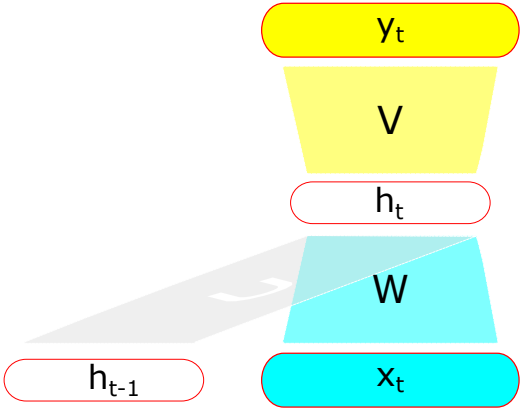
- Named Entity Recognition
- Paraphrase detection - identifying semantically equivalent questions
- Language Generation
- Machine Translation
- Speech recognition
  - Wreck a nice beach or recognize speech
- Automatically generating subtitles for a video
- Spell Checking
- Predictive typing
- Chat-bots/Dialog understanding
- Generate missing text
- Correct Hand-written text

RECURRENT NEURAL NETWORKS (RNNS)

- RNNs are designed to process sequential data.
- They maintain a hidden state that captures information about past inputs.
- The hidden state is updated at each time step.



RNN - AN EXTENSION OF A FEED-FORWARD NETWORK



- the memory includes the information from the start of the sentence with no imposition of window size
- The hidden weights U from the time-stamp h_{t-1} is the significant addition to RNN
- The past weights from the previous time-stamp determines memory of the network

h_t	$= f(h_{t-1}, x_t; \theta)$
y_t	$= Vh_t$
x_t	: Input at time t
h_{t-1}	: State of hidden weights at time $t-1$

RNN EQUATIONS

- Input at time step t: x_t (word embedding).
- Hidden state at time step t: h_t.
- Output at time step t: y_t.
- Equations:

$$h_t = f(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$
$$y_t = \text{softmax}(W_{hy}h_t + b_y)$$

- Where:
  - f is an activation function. In this case it is $\tanh$
  - W_{xh}, W_{hh}, W_{hy} are weight matrices.
  - b_h, b_y are bias vectors.

WHY tanh?

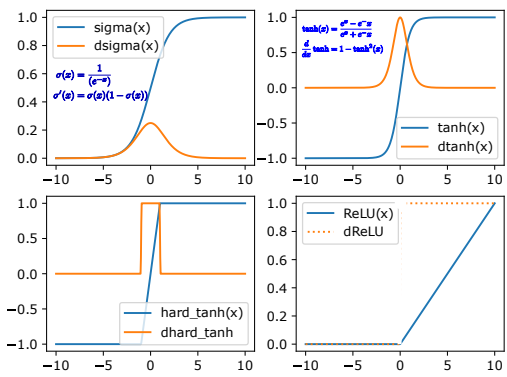
tanh is symmetrical around the origin.

- Zero-centring of tanh allows faster convergence during the training process
- $\partial(\tanh)$ is steep at the origin or gradients are larger when compared to sigmoid function
- Allows efficient backpropagation of errors during the training
- Speeds up the learning process or updates weights faster than sigmoid
- $\partial(\sigma)$ has a range $[0, 0.25]$. Changes slowly at the origin - may result in very slow learning at around the origin
- $\partial(\tanh)$ should be able to capture small changes
- $\partial(\tanh)$ has a range $[0, 1]$. The rapid change may minimize the problem of vanishing gradient

Is Sigmoid function symmetrical around origin?

DERIVATIVES OF SIGMOID, TANH, HARD TANH, RELU

- The steeper derivative of `tanh` amplifies gradient signals during backpropagation.
- This can lead to faster convergence and a slight reduction in vanishing gradient problem compared to sigmoid
- Modern architectures largely use Rectified Linear Unit (ReLU) and its variants
- ReLU and its variants provide gradient flow and are computationally efficient



RELU

- ReLU is a simple and efficient activation function.
- Alleviates the vanishing gradient problem.
- Offers computational efficiency and sparsity.
- If a ReLU neuron's input is always negative, it becomes inactive.
- Gradients stop flowing through it.
- Can lead to "dead" neurons that never recover.
- Variations like Leaky ReLU and Parametric ReLU address this issue.
  - **Leaky ReLU:** $\mathrm{LeakyReLU}(x) = \max(\alpha x, x)$ (α is a small constant).
  - **Parametric ReLU (PReLU):** $\mathrm{PReLU}(x) = \max(\alpha x, x)$ (α is a learnable parameter).
  - **Exponential Linear Unit (ELU):** Smoothly transitions to negative values.

RNNS FOR LANGUAGE MODELING

- Input: Sequence of word embeddings.
- Output: Probability distribution over the vocabulary for the next word.
- At each time step t, the RNN predicts the next word w_{t+1} given the previous words $w_1, w_2, ..., w_t$.
- The output y_t represents the probability distribution:

$$P(w_{t+1}|w_1, w_2, ..., w_t)$$

TRAINING RNN LANGUAGE MODELS

- Objective: Minimize the cross-entropy loss between the predicted and actual next words.
- Loss function at time step t:

$$L_t = -\sum_{i=1}^{|V|} y_{t,i}^{\text{true}} \log(y_{t,i})$$

  Where $|V|$ is the vocabulary size, $y_{t,i}$ is the predicted probability of the i-th word, and $y_{t,i}^{\text{true}}$ is the true probability (1 for the correct word, 0 otherwise).
- Total loss:

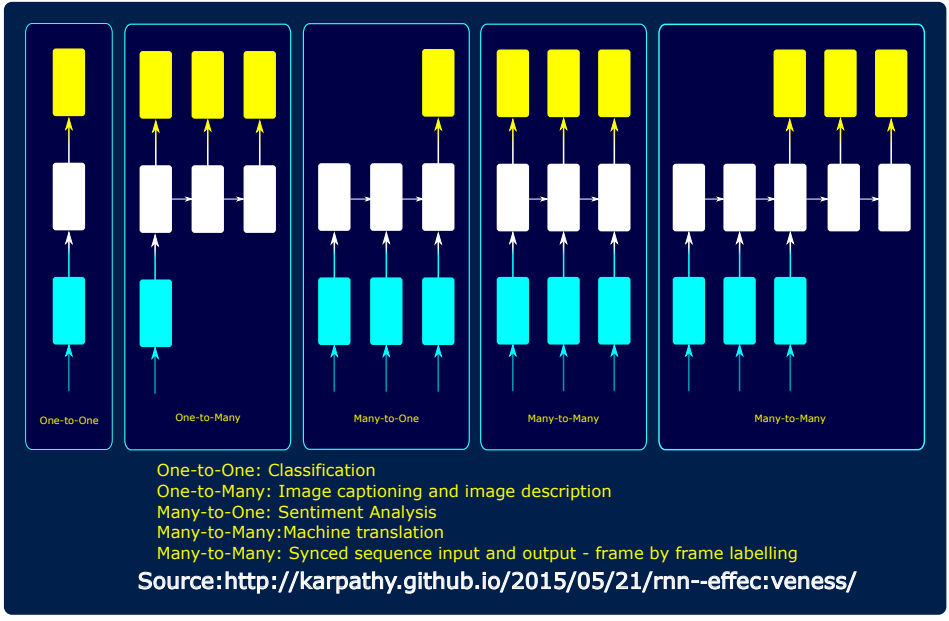
$$L = \sum_{t=1}^{T} L_t$$

- Optimization: Backpropagation through time (BPTT).

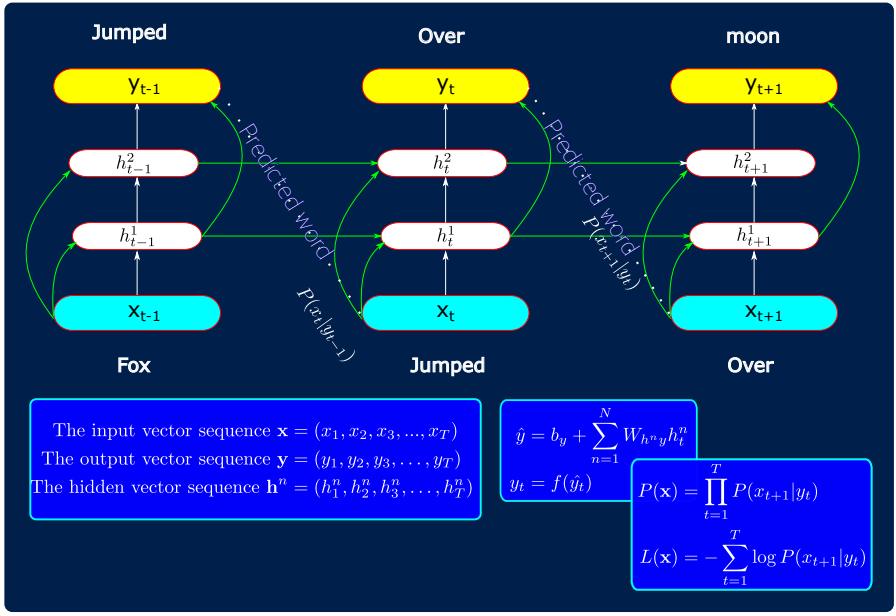
GENERATING TEXT USING RNN LANGUAGE MODELS

- Given a starting sequence of words.
- Sample the next word from the predicted probability distribution.
- Append the sampled word to the sequence.
- Repeat until a desired length or termination condition is met.

MULTIPLE ARCHITECTURES OF RNN



RNN FOR PREDICTING THE NEXT WORD



SHORT-TERM MEMORY OF RNN

Learning to store information over extended time intervals via recurrent takes a very long time, mostly due to insufficient, decaying error back flow [2]

- Vanilla RNNs are good at learning from sequential recency rather than from long term dependency
- The temporal evolution of the back-propagated error exponentially depends on the size of the weights
- The gradients tend to either (1) explode or (2) vanish:
  - If it explodes up, then the learning may lead to oscillating weights
  - If it vanishes, then either takes a lot of time to learn or fails

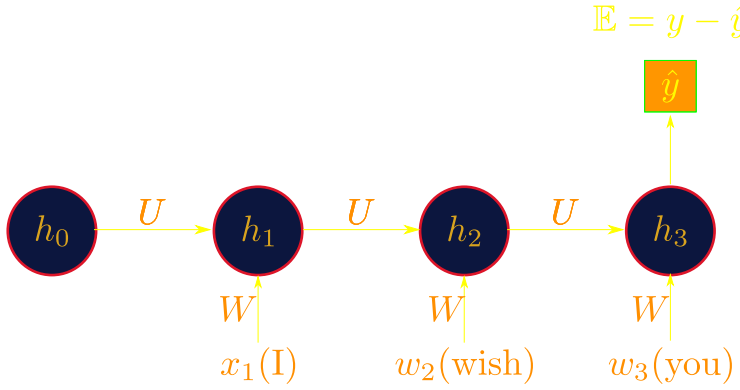
DERIVATIVES OF ACTIVATION FUNCTIONS

Activation Function	Derivative
$y = \left(\frac{1}{1+e^{-x}}\right)$	$\frac{dy}{dx} = \left(\frac{1}{1+e^{-x}}\right)\left(1-\frac{1}{1+e^{-x}}\right) = \sigma(x)(1-\sigma(x))$
$y = \tanh(x) = \frac{\sinh(x)}{\cosh(x)}$	$\frac{dy}{dx} = \frac{\cosh(x)\cosh(x)-\sinh(x)\sinh(x)}{\cosh^2(x)} = (1-\tanh^2(x))$
$y = \max(c,x)$	$\frac{dy}{dx} = \begin{cases} 1, & \text{if } x > c \\ 0, & \text{if } x < c \end{cases}$ For ReLU, $c = 0$; $\frac{dy}{dx}$ does not exist at $x = c$

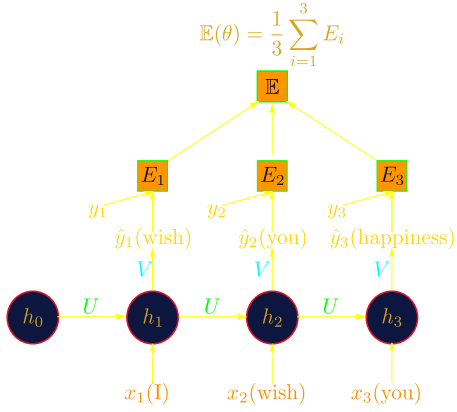
WHY RELU OR ITS EXTENSIONS?

- Sigmoid and Tanh functions saturate while RELU does not
- It is computationally inexpensive and produces real zero and not approximate one

ERROR COMPUTATION - 1



ERROR COMPUTATION - 2



–

We need to compute the following derivatives

FORWARD PASS I

Forward Pass

$$h_t = \tanh(Wx_t + Uh_{t-1}) \quad (9)$$

$$\hat{y}_t = \mathrm{softmax}(Vh_t) \quad (10)$$

$$E_t = -\sum_t y_t \log(\hat{y}_t) \quad (11)$$

FORWARD PASS II

If the corpus contains T words, then $(x_1, x_2, x_3, \ldots, x_T)$ are the corresponding word vectors

- $x_t \in R^{D_w}$ represents the input word at time t and D_w is the dimension of the word vector. If one-hot vector, it will be $x^{D_{|V|}}$
- $W \in R^{D_w \times D_h}$ is the weight matrix that conditions the input vector
- $U \in R^{D_h \times D_h}$ matrix that keeps the dependency of the word sequence
- $V \in R^{|V| \times R^{D_h}}$
- s_{t-1} is the output of the non-linear function $(\tanh)$ of the time step $t-1$
- $\hat{y_t}^t \in R^{|V|}$ is the probability distribution of the predicted word at time step t for the given context of $x_1, x_2, x_3, \ldots x_t$, where $|V|$ is the size of the vocabulary

BPTT - DERIVATIVE FOR V

$$h_t = \tanh(Wx_t + Uh_{t-1}) \tag{12}$$

$$\hat{y}_t = \text{softmax}(Vh_t) \tag{13}$$

$$E_t = -\sum_k y_{t,k} \log(\hat{y}_{t,k}) \tag{14}$$

$$\mathbb{L} = \sum_t E_t \tag{15}$$

$$\frac{\partial E_t}{\partial V} = \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial V} \tag{16}$$

$$\frac{\partial E_t}{\partial \hat{y}_t} = \hat{y}_t - y_t = \delta^t_{out} \tag{17}$$

$$\frac{\partial \hat{y}_t}{\partial V} = h_t^T \tag{18}$$

$$\frac{\partial E_t}{\partial V} = \delta^t_{out} h_t^T \tag{19}$$

Assumption - Cross-entropy loss with softmax

RNN: DERIVATIVE WITH RESPECT TO U

$$h_t = \tanh(Wx_t + Uh_{t-1}) \tag{20}$$

$$E_t = -\sum_k y_{t,k} \log(\hat{y}_{t,k}) \tag{21}$$

$$\frac{\partial E_t}{\partial U} = \frac{\partial E_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial U} \tag{22}$$

$$\frac{\partial E_t}{\partial \hat{y}_t} = \hat{y}_t - y_t \quad ; \frac{\partial \hat{y}_t}{\partial h_t} = V^\top$$

$$\frac{\partial h_t}{\partial U} = (1 - h_t^2)\, h_{t-1}^\top$$

$$\frac{\partial E_t}{\partial U} = \left[\left(V^\top(\hat{y}_t - y_t)\right) \odot (1 - h_t^2)\right] h_{t-1}^\top \tag{23}$$

Note: $\odot$ *denotes element-wise multiplication.*

RNN: DERIVATIVE FOR $\frac{\partial E_t}{\partial W}$

$$\frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial W} \tag{24}$$

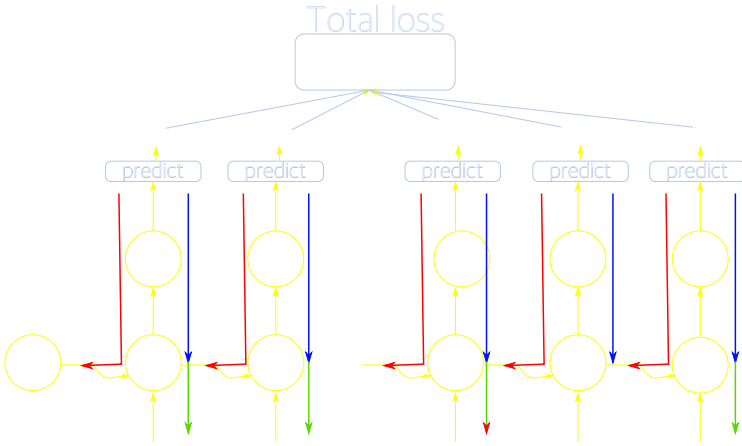
$$\frac{\partial E_t}{\partial \hat{y}_t} = \hat{y}_t - y_t \tag{25}$$

$$\frac{\partial \hat{y}_t}{\partial h_t} = V^\top; \quad \frac{\partial h_t}{\partial W} = (1 - h_t^2) x_t^\top$$

$$\frac{\partial E_t}{\partial W} = \left[(V^\top (\hat{y}_t - y_t)) \odot (1 - h_t^2) \right] x_t^\top \tag{26}$$

Note: $\odot$ denotes elementwise multiplication.

BPTT - UNROLLED RNN



The error for the entire duration of T for all the vocabulary is the sum of all the error across the layers

$$\mathbb{E}(\theta) = -\frac{1}{T}\sum_{1}^{T}\sum_{j=1}^{|V|} y_{t,j}\log(\hat{y}_{t,j}) \tag{27}$$

PERPLEXITY

Perplexity is a measurement of how well a model predicts a sample. Perplexity is defined as

$$\text{For bigram model, } \mathrm{PP}(W_N) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}} \tag{28}$$

$$\text{For trigram model } \mathrm{PP}(W_N) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1}w_{i-2})}} \tag{29}$$

A good model gives maximum probability to a sentence or minimum perplexity to a sentence

SIZE OF THE RNN NETWORK

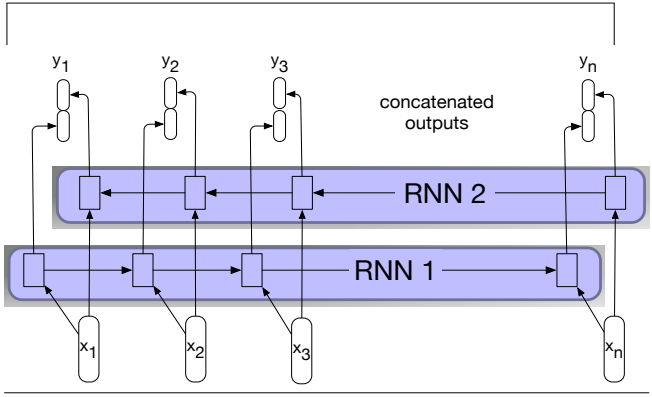
If we assume the size of the word vector as 100 and the number of the hidden neurons as 500, and $|V_{vocab}| = 10000$, then

Parameter	Size
Word Vector	100
W	500x100
h_t, s_t	500
U	500x500
V	500x10000
$\hat{y}_t$	10000

BI-DIRECTIONAL RNN

- Capture the word meaning as well as its contextual information to enable different word embeddings for the same word -contextualized word embedding
- The language model learns to predict not only the next word but also the previous word.

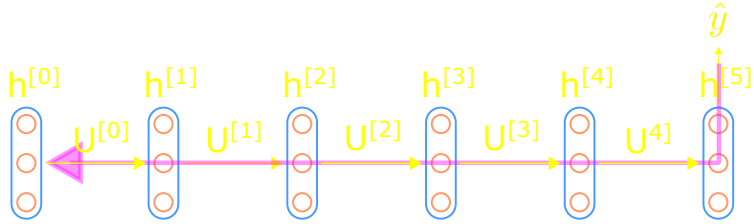
BIDIRECTIONAL RNN WITH MULTIPLE OUTPUT



WHAT IS THE PROBLEM WITH RNN?

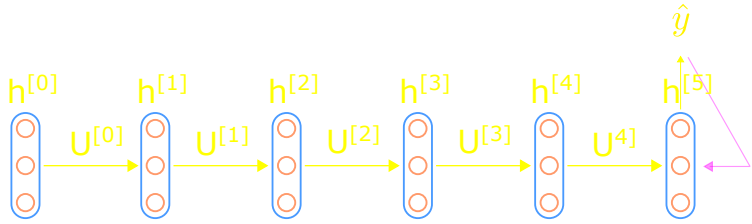
- Long-term behavior depends on the eigenvalues of the recurrent matrix U
- $Uv_i = \lambda_i v_i$
- Theoretically, it is possible to store all historical information in the RNN
- If the eigenvalues satisfy $|\lambda| > 1$, the gradients explode (grow very large)
- If the eigenvalues satisfy $|\lambda| < 1$, the gradients vanish quickly
- Vanishing gradient problem - The diminishing value of δ makes it difficult to capture the long term memory as we move down the memory lane or layers of hidden nodes
- What is the solution?

EXPLODING/VANISHING GRADIENT



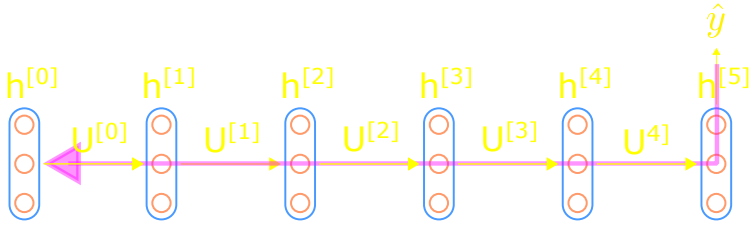
We know that $\hat{y_t} = f(U, h)$. We want to propagate the error through back propagation

EXPLODING/VANISHING GRADIENT



The error E at 5th time state depends on $h^{[5]}$. Hence we need to estimate $\frac{\partial E}{\partial h^{[5]}}$. $h^{[5]}$ depends on $h^{[4]}$, $h^{[4]}$ depends on $h^{[3]}$, $h^{[3]}$ depends on $h^{[2]}$, $h^{[2]}$ depends on $h^{[1]}$, and $h^{[1]}$ depends on $h^{[0]}$.

EXPLODING/VANISHING GRADIENT



$$\frac{\partial E^{[5]}}{\partial h^{[0]}} = \frac{\partial E^{[5]}}{\partial h^{[5]}} \times \frac{\partial h^{[5]}}{\partial h^{[4]}} \times \frac{\partial h^{[4]}}{\partial h^{[3]}} \times \frac{\partial h^{[3]}}{\partial h^{[2]}} \times \frac{\partial h^{[2]}}{\partial h^{[1]}} \times \frac{\partial h^{[1]}}{\partial h^{[0]}} = \frac{\partial E^{[5]}}{\partial h^{[5]}} \prod_{t=1}^{t=5} \frac{\partial h^{[t]}}{\partial h^{[t-1]}} \quad (30)$$

Generalizing

$$\frac{\partial E^{[\tau]}}{\partial h^{[0]}} = \frac{\partial E^{[\tau]}}{\partial h^{[\tau]}} \prod_{t=1}^{t=\tau} \frac{\partial h^{[t]}}{\partial h^{[t-1]}} \quad (31)$$

where τ represents depth of the layers or the size of the time series

EXPLODING/VANISHING GRADIENT

$$\frac{\partial E^{[\tau]}}{\partial h^{[0]}} = \frac{\partial E^{[\tau]}}{\partial h^{[\tau]}} \prod_{t=1}^{t<\tau} \frac{\partial h^{[t]}}{\partial h^{[t-1]}}$$

$$h^{[t]} = \sigma\left(WX^{[t]} + Uh^{[t-1]}\right) \tag{32}$$

$$\frac{\partial h^{[t]}}{\partial h^{[t-1]}} = \mathrm{diag}(\sigma'(WX^{[t-1]} + Uh^{[t-1]}))U \tag{33}$$

where σ' computes element-wise the derivative of σ

$$\frac{\partial h^{[t]}}{\partial h^{[t-1]}} \text{ is a Jacobian} \tag{34}$$

$$\therefore \frac{\partial E^{[\tau]}}{\partial h^{[0]}} = \frac{\partial E^{[\tau]}}{\partial h^{[\tau]}} U^{\tau} \prod_{t=1}^{t=\tau} \mathrm{diag}(\sigma'(WX^{[t-1]} + Uh^{[t-1]})) \tag{35}$$

Multiplication of the Jacobian is always a shrinking operation. Values of U become very small when the depth increases[1]

[1] Source: "On the difficulty of training ... recurrent neural networks", Pascanu et al., 2013

EXPLODING/VANISHING GRADIENT

Consider the following sentence:
Raj entered CoffeeDay to meet his partner Dru. Raj said "Hi Dru. In the next few hours they discussed their start-up and devised a plan to develop a product on knowledge management. After a the long discussion and fruitful discussion, Raj said goodbye to his _____(47^{th} word).

The target word is *partner*. If the long distance gradient (the gap between $U^{[7]}$ and $U^{[\$]}$ is large), then the target word is lost in the gradient as it would be to small to contribute

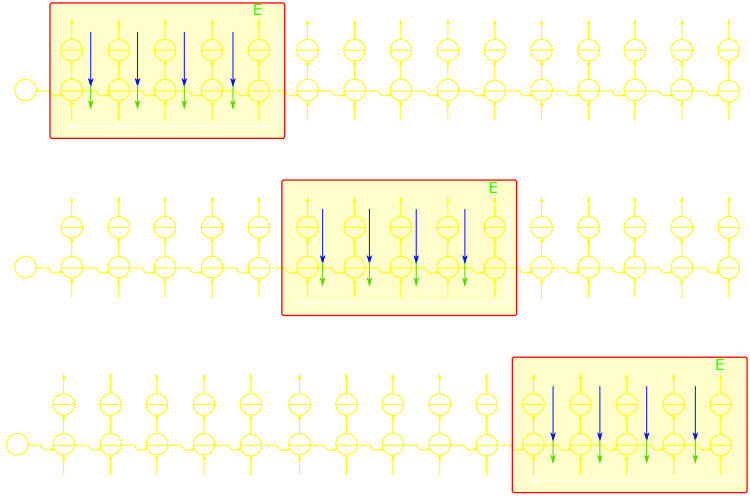
The decay in the gradient value is proportional to the depth of the network. The deeper the net network, the the chance of getting a smaller value of the gradient towards the end of the back propagation. If the some of the values are in the range of $[(0.01, 0.5), (0.03, 0.01)]$, then the the derivative would vanish to zero

$$0.01^{47} = 1.0e-94 \text{ and } 0.5^{47} = 7.1054274e-15 \text{ and } 1.2^{47} = 5266.45726273$$

GRADIENT CLIPPING

- The gradient is either very large or very small. This can cause the optimizer to converge slowly.
- To speed up training, clip the gradient at certain values
  - If $g < 1$, or if $, g > 1$, then $g = 1$
  - Or
  - If $\|g\| > \mathrm{threshold}$, then $g \leftarrow \frac{\mathrm{threshold}}{\|g\|} g$
- Clip the gradient if it exceeds a threshold

TBPTT



TRUNCATED BPTT

For applications with long sequences, the input is truncated into manageable fixed-sized segments. This approach is called Truncated Backpropagation Through Time (TBPTT).

Example

Consider a sequence of 5000 samples. We could split this in to 50 sequences of 100 samples each, and the BPTT is computed for each sequence. This works most of the time, but it is blind to temporal dependencies that may span across two sequences. One way to solve this is to have a sentence separator as the conditional BPTT.

Hyper-parameter

- *batch* - the number of samples or subset of the training dataset
- *Epoch* - complete pass through the entire training dataset

TRADITIONAL RNN LIMITATIONS

- The component of the gradient in directions that correspond to long-term dependencies is small[2]
- Gradients shrink over time, making it hard to learn long-term dependencies
- $$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \left(\prod_{k=t+1}^{T} \frac{\partial h_k}{\partial h_{k-1}} \right) \frac{\partial E}{\partial h_t} \frac{\partial h_t}{\partial W}$$
- The component of the gradient in directions that correspond to short-term dependencies is large
- As a result, RNNs can easily learn the short-term but not the long-term dependencies
- Short-Term Memory: Effectively remembers information for a few time steps

[2] An empirical exploration of recurrent network architectures - http://dl.acm.org/citation.cfm?id=3045118.3045367

GATING MECHANISM

We require a slowly-decaying error propagation. In other words, the update of weights should enhance/retain distributed properties of a sequence.

- Control the flow of information
- Should the new/old information be allowed or dropped?
- Gates are capable of interrupting, or allowing, the passage of activation values among neurons in the hidden layer
- The ability to manage the flow of information may play a key role arresting or setting up a well-behaved gradient during the back-propagation
- Multiplicative gates provide the protection of memory contents from decaying
- Multiplicative input and output gates protect contents from perturbations

WHAT IS LSTM?

- Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN).
- Designed to overcome the limitations of traditional RNNs, particularly with long-term dependencies.
- The memory cell acts like a conveyor belt for information flow, allowing it to maintain information for long periods.

LSTM I

- An LSTM network[2] is the same as a standard RNN, except that the summation units in the hidden layer are replaced by memory blocks
- Instead of computing h_t from h_{t-1} directly with a linear combination followed by a nonlinearity ($f(W, x_t, h_{t-1})$), the LSTM directly computes Δh_t, which is then added to h_{t-1} to obtain h_t
- The multiplicative gates allow LSTM memory cells to store and access information over long periods of time, thereby mitigating the vanishing gradient problem[3]
- Along with the hidden state vector, h_t, LSTM maintains a memory vector C_t

LSTM II

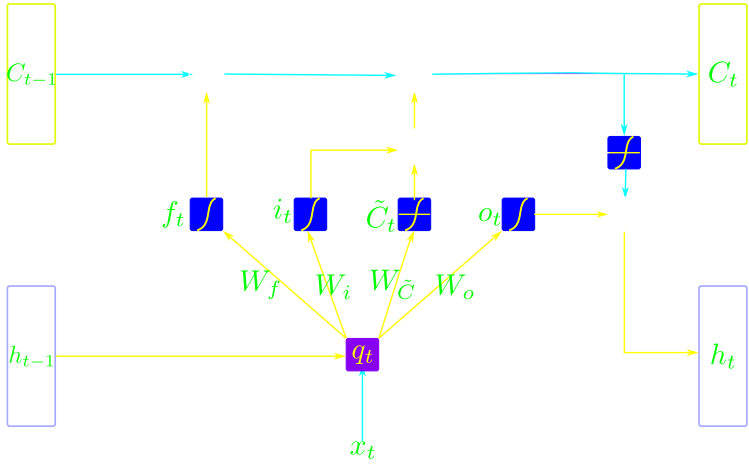
- At each time step the LSTM can choose to read from, write to, or reset the cell using explicit gating mechanisms
- LSTM computes well behaved gradients by controlling the values using the gates
- LSTM turns multiplication into addition
- LSTM uses gates to control how much information to add/erase or include/forget
- LSTM doesnt guarantee that there will be no vanishing/exploding gradient, but it provides a simple way to learn long-distance dependencies

[3] http://dblp.uni-trier.de/db/journals/corr/corr1506.html#KarpathyJL15

LSTM CELL STRUCTURE

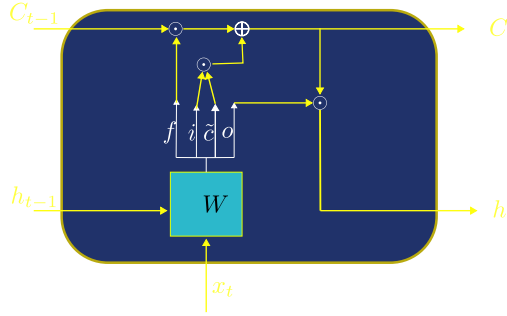
- Cell State (C_t): Main pathway for information flow.
- Forget Gate (f_t): Decides what to discard from the cell state.
- Input Gate (i_t): Determines how much new information to add.
- Candidate Cell State ($\tilde{C}_t$): New values that could be added.
- Output Gate (o_t): Controls what parts of the cell state to output.

LSTM CELL

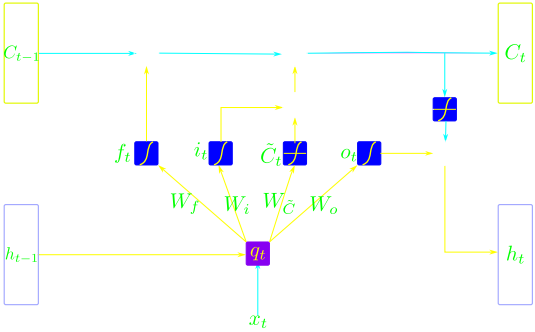


SIMPLE REPRESENTATION

$$\begin{pmatrix} i \\ f \\ \tilde{C} \\ o \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \tanh \\ \sigma \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \quad (36)$$



LSTM - FORWARD PASS



$$f_t = \sigma(W_f q_t + b_f) \quad (37)$$

$$i_t = \sigma(W_i q_t + b_i) \quad (38)$$

$$\tilde{C}_t = \tanh(W_{\tilde{C}_t} q_t) \quad (39)$$

$$C_t = (f_t \otimes C_{t-1}) \oplus (i_t \otimes \tilde{C}_t) \quad (40)$$

$$o_t = \sigma(W_o q_t + b_o) \quad (41)$$

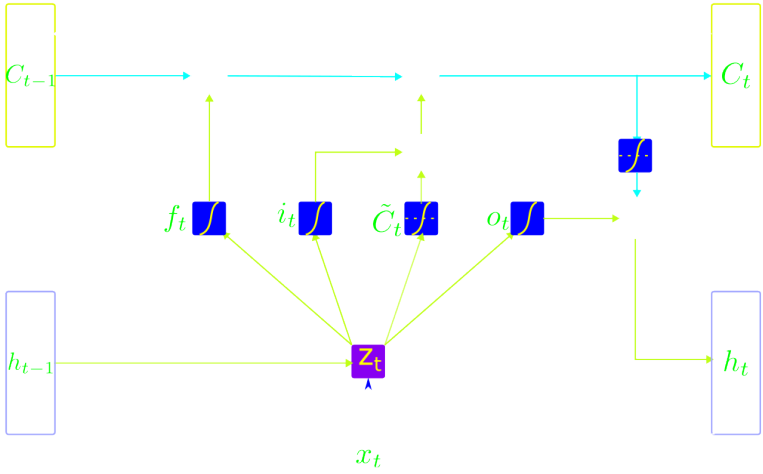
$$h_t = o_t \otimes \tanh(C_t) \quad (42)$$

$$s_t = \tanh(h_t) \quad (43)$$

$$z_t = V s_t \quad (44)$$

$$\hat{y}_t = \mathrm{softmax}(z_t) \quad (45)$$

BACK PROPAGATION THROUGH TIME



FORGET GATE

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

- f_t: Forget gate output.
- σ: Sigmoid function.
- W_f: Weight matrix for the forget gate.
- $[h_{t-1}, x_t]$: Concatenation of previous hidden state and current input.
- b_f: Bias vector for the forget gate.
- Output Values: - 0 means "completely forget". - 1 means "completely retain".
- Learning Process: Through training, the network learns what information to forget or retain.

IMPORTANCE OF FORGET GATE BIAS IN LSTM INITIALIZATION

- A critical yet often overlooked aspect of LSTMs is the initialization of the forget gate bias $\mathrm{b_f}$.
- Standard LSTM initialization typically uses small random weights, which can introduce challenges with long-term dependencies.

EFFECT OF DEFAULT FORGET GATE INITIALIZATION

- Default initialization of LSTM weights often leads to the forget gate bias $\mathrm{b_f}$ being close to 0.5.
- This results in a vanishing gradient with a decay factor of approximately 0.5 per time step.
- Problems with long-term dependencies are especially affected by this vanishing gradient, as seen in [2] and [3]

ADJUSTING THE FORGET GATE BIAS

- To mitigate the vanishing gradient problem, set the forget gate bias $\mathrm{b_f}$ to a higher value, such as 1 or 2.
- Initializing $\mathrm{b_f}$ to a large value ensures the forget gate is close to 1, enabling better gradient flow.
- This initialization strategy was originally suggested by Gers et al. [4].

$$\mathrm{b_f} \approx 1 \text{ or } 2 \tag{46}$$

RISKS OF INCORRECT FORGET GATE INITIALIZATION

- Without proper initialization of $\mathrm{b_f}$, the LSTM might appear incapable of learning tasks with long-range dependencies
- This is a misconception. Appropriate initialization of the forget gate enables the LSTM to manage long-term information
- Initializing the forget gate bias $\mathrm{b_f}$ to a high value is crucial for effective learning of long-term dependencies in LSTMs
- This adjustment prevents vanishing gradients and enhances LSTMs performance on tasks requiring long-range memory
- Reemphasizing this technique is valuable, as it is often overlooked but significantly impacts the performance of LSTMs

INPUT GATE AND CANDIDATE CELL STATE

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

where i_t and $\tilde{C}_t$ are the Input gate output and the Candidate cell state, respectively

CELL STATE UPDATE

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

where C_t is the Current cell state and $\odot$ is the Element-wise multiplication operator

OUTPUT GATE

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \odot \tanh(C_t)$$

where o_t is the Output gate and h_t is the output of the Hidden state

GRADIENT FLOW THROUGH CELL STATE

$$\frac{\partial E}{\partial C_{t-1}} = f_t \odot \frac{\partial E}{\partial C_t}$$

where E is the loss function and f_t is the forget-gate value at time t.

IMPACT ON LEARNING

- Long-Term Dependencies: Cell state allows gradients to flow back many steps if necessary.
- Mitigating Vanishing Gradients: Helps in keeping information unchanged for many steps.

- LSTMs use forget gates to manage information flow, solving short-term memory issues in traditional RNNs.
- Back propagation through the cell state allows for effective learning of long-term dependencies.

WHAT IS GRU?

- Gated Recurrent Unit (GRU) is a type of recurrent neural network (RNN) designed to address the vanishing gradient problem
- Introduced as a simpler alternative to Long Short-Term Memory (LSTM) units

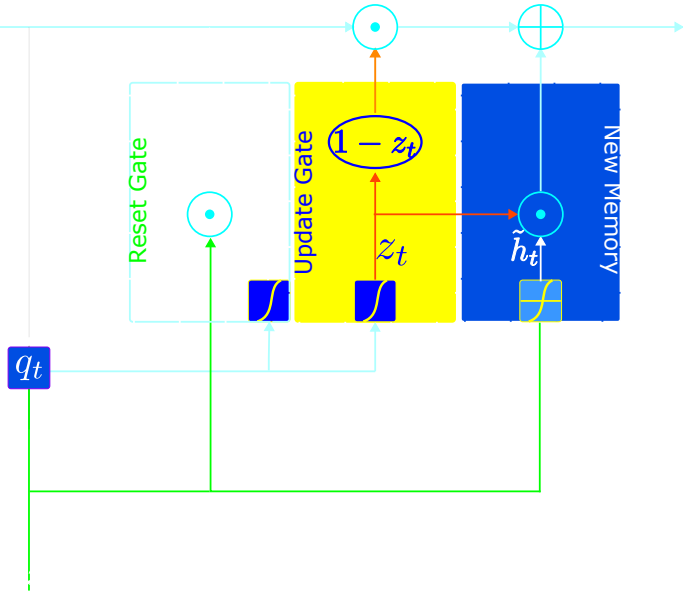
GRU ARCHITECTURE

- Reset Gate (r_t):
  - Controls how much of the previous information to forget.
  - $r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$
- Update Gate (z_t):
  - Determines how much of the previous information to retain in the current state.
  - $z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$
- Candidate Activation ($\tilde{h}_t$):
  - Combines reset gate information to produce a candidate activation.
  - $\tilde{h}_t = \tanh(W \cdot [r_t \odot h_{t-1}, x_t] + b)$
- Hidden State Update (h_t):
  - Updates the hidden state based on the update gate and candidate activation.
  - $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$

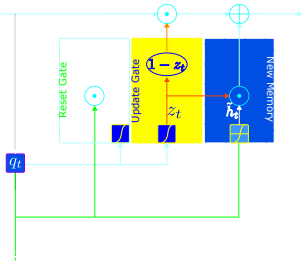
COMPONENTS OF GRU

- Reset Gate (r_t): Decides how much of the past information to forget
- Update Gate (z_t): Determines how much of the previous memory to keep or update.
- Candidate Activation ($\tilde{h}_t$): New memory content

GRU ARCHITECTURE



GRU FORWARD PASS



$$q_t = [h_{t-1}, x_t] \tag{47}$$

$$z_t = \sigma(W_z q_t) \tag{48}$$

$$r_t = \sigma(W_r q_t) \tag{49}$$

$$\tilde{h}_t = \tanh(W[r_t \otimes h_{t-1}, x_t]) \tag{50}$$

$$h_t = (1 - z_t) \otimes h_{t-1} \oplus (z_t \otimes \tilde{h}_t) \tag{51}$$

$$s_t = \tanh(h_t) \tag{52}$$

$$\hat{y}_t = softmax(V s_t) \tag{53}$$

Intuition

If the reset gate values $\rightarrow 0$, previous memory states are faded and new information is stored. If the z_t is close to 1, the information is copied and retained thereby adjusting the gradient to be alive for the next time step, thereby long-term dependency is stored. BPTT decides the learning of the reset and update gate.

STRUCTURAL DIFFERENCES

- GRU: Combines the forget and input gates into a single update gate, and merges the cell state and hidden state
- LSTM: Has separate gates for forgetting, inputting, and outputting, with an explicit cell state

COMPUTATIONAL COMPLEXITY

- GRU: Fewer parameters and operations, making it computationally lighter
- LSTM: More parameters and operations, potentially leading to better performance on complex tasks but at higher computational cost

PERFORMANCE AND USE CASES

- GRU: Often performs comparably to LSTM on many tasks, especially when computational efficiency is crucial
- LSTM: Generally preferred for tasks requiring longer-term memory or when the additional complexity can be justified by performance gains

WHAT IS BPTT?

- BPTT is a training algorithm for recurrent neural networks (RNNs) like GRUs
- It unfolds the RNN over time, treating each time step as a layer in a deep neural network

GRU FORWARD PASS EQUATIONS

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

$$\tilde{h}_t = \tanh(W \cdot [r_t \odot h_{t-1}, x_t] + b)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

BPTT STEPS

1. Forward pass through all time steps.
2. Compute loss at the final time step.
3. Backpropagate through the unfolded network.
4. Update weights considering temporal dependencies.

Loss Function:

$$L = \text{Loss}(h_T, y)$$

where h_T is the final hidden state, and y is the target output.

Initial Derivatives:

$$\frac{\partial L}{\partial h_T} = \text{computed from loss function}$$

BACKPROPAGATION THROUGH TIME (BPTT) IN GRUS

- Like RNNs and LSTMs, GRUs are trained using Backpropagation Through Time (BPTT).
- BPTT involves unrolling the network across time steps and applying backpropagation to compute gradients.
- In GRUs, we must compute gradients for both the Update Gate and Reset Gate.
- Our goal is to compute partial derivatives with respect to each parameter in order to update them during training.

GRADIENT OF LOSS WITH RESPECT TO OUTPUT

- Let the loss at time t be L_t.
- We aim to compute $\frac{\partial L}{\partial h_t}$ for the current time step.
- Using the chain rule, the total gradient of the loss over time will be:

$$\frac{\partial L}{\partial h_t} = \frac{\partial L_t}{\partial h_t} + \sum_{k=t+1}^{T} \frac{\partial L_k}{\partial h_k} \frac{\partial h_k}{\partial h_t}$$

GRADIENT OF HIDDEN STATE UPDATE h_t

- Recall the hidden state update: $h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$.
- The gradient with respect to h_t becomes:

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial h_t} + \frac{\partial L_t}{\partial h_t}$$

- Expanding $\frac{\partial h_t}{\partial h_{t-1}}$:

$$\frac{\partial h_t}{\partial h_{t-1}} = (1 - z_t) + z_t \cdot r_t \cdot \frac{\partial \tilde{h}_t}{\partial h_{t-1}}$$

GRADIENT OF UPDATE GATE z_t

- The gradient with respect to z_t is crucial for updating the hidden state:

$$\frac{\partial L}{\partial z_t} = \frac{\partial L}{\partial h_t} \cdot (\tilde{h}_t - h_{t-1})$$

- Applying the sigmoid derivative:

$$\frac{\partial z_t}{\partial W_z} = z_t(1 - z_t) \cdot \frac{\partial L}{\partial z_t}$$

GRADIENT OF RESET GATE r_t

- For the reset gate r_t: $\frac{\partial L}{\partial r_t} = \frac{\partial L}{\partial \tilde{h}_t} \cdot h_{t-1}$
- Using the chain rule, we derive: $\frac{\partial r_t}{\partial W_r} = r_t(1 - r_t) \cdot \frac{\partial L}{\partial r_t}$

GRADIENT OF CANDIDATE ACTIVATION $\tilde{h}_t$

- The candidate activation gradient $\frac{\partial L}{\partial \tilde{h}_t}$ is calculated as:

$$\frac{\partial L}{\partial \tilde{h}_t} = \frac{\partial L}{\partial h_t} \cdot z_t$$

- Expanding further, with the $\tanh$ activation derivative:

$$\frac{\partial \tilde{h}_t}{\partial W} = (1 - \tilde{h}_t^2) \cdot \frac{\partial L}{\partial \tilde{h}_t}$$

INTERSECTION OF NEUROSCIENCE AND MACHINE LEARNING

- When discussing the biology of forgetting, we enter an intriguing intersection of neuroscience and machine learning.
- Computational models like GRU (Gated Recurrent Unit) and LSTM (Long Short-Term Memory) provide a framework for understanding how information is retained or forgotten.
- Lets explore how these models mimic biological processes in handling forgetting and retaining long-term relationships.

MECHANISMS OF BIOLOGICAL FORGETTING

Biological forgetting can occur due to two primary mechanisms:

- Decay:
  - Over time, memories weaken if they are not rehearsed or revisited.
  - This is analogous to how neural network information might degrade if not reinforced.
- Interference:
  - New information can interfere with old memories (retroactive interference).
  - Old memories can also interfere with new learning (proactive interference).
  - Similar to how new inputs in RNNs may overwrite or interfere with previous states if not managed properly.

LONG-TERM MEMORY IN BIOLOGY

- Memories that are deemed important or frequently accessed are consolidated into long-term memory (LTM).
- LTM consolidation involves changes in synaptic strength and neural connections.
- This process is analogous to how LSTMs maintain a cell state to retain information over extended periods.

LSTM: MECHANISMS FOR FORGETTING AND RETENTION

- Forget Gate:
  - Explicitly decides what information to discard from the cell state.
  - Closely mimics biological forgetting, where irrelevant or less important information is discarded.
- Cell State:
  - Acts as a form of long-term memory, allowing information to persist across many time steps.
  - Mirrors how humans retain frequently accessed information in LTM.
- Input and Output Gates:
  - Control what new information is added to the cell state and what information is output.
  - Analogous to how humans selectively remember or recall information based on context.

GRU: MECHANISMS FOR FORGETTING AND RETENTION

- Update Gate:
  - Combines the functionality of LSTMs input and forget gates.
  - Decides how much of the previous memory to retain or update with new information.
  - Less granular than LSTM but effective for many tasks.
- Reset Gate:
  - Controls how much past information to forget.
  - Although not as explicit as LSTMs forget gate, it allows a form of resetting past states.

COMPARISON: COMPLEXITY AND EFFICIENCY

- LSTM:
  - More complex with separate gates for forgetting, input, and output.
  - Better at handling long-term dependencies, though with more parameters and computational overhead.
- GRU:
  - Simpler, with combined gates, which reduces computational cost.
  - Efficient for shorter sequences but may not capture very long-term dependencies as effectively as LSTM.

ROLE OF MUSASHI PROTEINS ROLES IN MEMORY PROCESSES

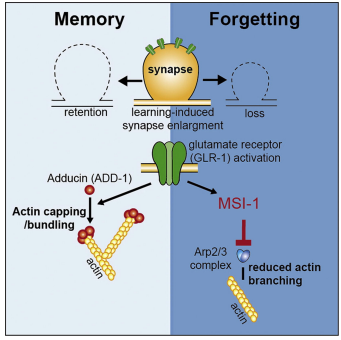


Figure: Musashi protein

- Repetition of events
- Primacy and recency
- Surprise
- Emotional Impact
- Positive or negative actions
- Hypocrisy
- ...

Memory length is regulated cooperatively through the activation of adducin (add-1) and by the inhibitory effect of msi-1[5]. Brain forgets unimportant information in order to remain efficient Can RNN be trained to simulate the actions of adducin and musashi proteins?

MUSHASHI VS RNNS

- Selective Processing: Both Musashi proteins and LSTM/GRU gates selectively process information
- In biological terms - this might mean which mRNAs are translated into proteins
- In neural networks - which information is retained or passed forward.
- Regulation Over Time: Both systems deal with changes over time
- Musashi's role in stem cell maintenance involves long-term regulation, similar to how LSTM maintains cell state over long sequences.

BIOLOGICAL ANALOGY: GRU VS. LSTM

- LSTM:
  - LSTMs multiple gates mimic biological systems, where different mechanisms control memory retention, consolidation, and forgetting.
  - The explicit *forget gate* mirrors selective forgetting in the human brain.
- GRU:
  - GRU has a simpler architecture, combining some functions, which could be seen as analogous to more streamlined or generalized memory functions in simpler neural systems.
  - While efficient, it lacks the granularity of the forget gate, making it more limited in mimicking biological forgetting.

SUMMARY I

- GRU simplifies LSTM by reducing the number of gates, making it more efficient but potentially less expressive for very complex temporal dependencies
- The choice between GRU and LSTM often depends on the specific requirements of the task, computational resources, and desired model complexity
- Back propagation in GRUs involves calculating partial derivatives for each gate and updating them based on gradients.
- The BPTT algorithm handles temporal dependencies by summing gradients over time.
- GRUs simplified structure (with two gates) generally leads to fewer parameters, making back propagation slightly more efficient compared to LSTMs
- Both LSTM and GRU offer models for understanding forgetting and retention.
- LSTM aligns more closely with complex biological systems due to its multiple gates.

SUMMARY II

- GRU provides a computationally efficient alternative, balancing simplicity with effective memory retention.
- This comparison highlights how machine learning models draw inspiration from biological processes, enhancing our understanding of memory in both fields.

REFERENCES I

[1] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd. Online manuscript released August 24, 2025. 2025. URL: https://web.stanford.edu/~jurafsky/slp3/.

[2] Sepp Hochreiter and Jürgen Schmidhuber. *Long Short-Term Memory*. Cambridge, MA, USA, Nov. 1997. DOI: 10.1162/neco.1997.9.8.1735. URL: https://www.bioinf.jku.at/publications/older/2604.pdf.

[3] Ilya Sutskever et al. "On the importance of initialization and momentum in deep learning". In: *International Conference on Machine Learning*. 2013. URL: https://api.semanticscholar.org/CorpusID:10940950.

[4] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. "Learning precise timing with LSTM recurrent networks". In: *Journal of machine learning research* 3.Aug (2002), pp. 115–143.

REFERENCES II

[5] Nils Hadziselimovic et al. “Forgetting Is Regulated via Musashi-Mediated Translational Control of the Arp2/3 Complex.”. In: *Cell* 156.6 (Mar. 2014), pp. 1153–1166. ISSN: 1097-4172. URL: http://view.ncbi.nlm.nih.gov/pubmed/24630719.