Lecture 6: Recurrent Neural Networks, Decoding

Andrei Arsene Simion

Columbia University

February 22, 2023

Outline

- Language Modeling
- Recurrent Neural Networks
- LSTM
- Decoding

Outline

- Language Modeling
- Recurrent Neural Networks
- LSTM
- Decoding

Outline

- Language Modeling is one of the oldest problems in NLP
- Nowadays, language models are usually termed "Foundation" Models since people are starting to realize that solving Language Modeling leads to solving other things
- At a high level, we'd like to be able to construct plausible sentences, sequences with $P(y_1, y_2, \ldots, y_T)$ high
- We saw before that we can do this with n-gram count based models (Lecture 1), CNNs (Lecture 5), and MLP (Lecture 1)

Outline

- All these models had a problem: limited (fixed) context
- Can we build models with the potential of having unlimited context?
- Unlike before, we do not want to approximate the equality

$$P(y_1, \ldots, y_T) = P(y_1)P(y_2|y_1) \ldots P(y_T|y_{T-1}, \ldots, y_1)$$

- Recall that, for the previously models, we had to assume that

$$P(y_t|y_{t-1}, \ldots, y_1) \sim P(y_t|y_{t-1}, \ldots, y_{t-k})$$

- The models we look at today do not need such approximations ...

Outline

- Language Modeling
- Recurrent Neural Networks
- LSTM
- Decoding

Recurrent Neural Networks: example data

- One way get training data for language modeling is to grab chunks of contiguous text $x = (x_1, x_2, \ldots, x_T)$ and try to predict the shifted version of this text $y = (x_2, x_3, \ldots, x_{T+1})$
- For example, you want to map "START The man walks" to "The man walks END"
- Here $x = (START, "The", "man", "walks")$ and $y = ("The", "man", "walks", END)$
- We'd like to have a model that feeds in each x_t in sequence and using this x_t (and past history) predicts y_t
- Some notes:
  - In the example above, the Loss should have 4 terms
  - Feeding data into this model should be sequential, from left to right
  - We need a way to pass in new data and carry forward older data
  - This is Self-Supervised Learning: we took some raw data and built x and y from it out of thin air

Recurrent Neural Networks

- In summary we want a model so that we can build $P(y_1, y_2, \dots, y_T)$
- Using Bayes' rule, this is equal to

$$P(y_1)P(y_2|y_1)\dots P(y_T|y_{T-1}, \dots, y_1)$$

- We'd like to NOT approximate this, we'd like to have in some sense a very large or an infinite context
- We want models that can truly express terms like

$$P(y_T|y_{T-1}, \dots, y_1)$$

- RNN's allow us to do this

Recurrent Neural Networks: Recursions

- Each time step of an RNN related h_t to h_{t-1} and x_t
  - At each time step, we pass in x_t
  - Additionally, also use the previous h_{t-1}
- We have

$$h_t = tanh(W_h h_{t-1} + W_x x_t + b_h)$$

- At each time step, we have $\hat{y}_t = F(W_y h_t + b_y)$ where F is usually a sigmoid or softmax (over $|V|$ possible words)
- We initialize h_0 appropriately
- At each time step, we have a Loss $L_t(\hat{y}_t, y_t)$ which is usually a Cross-Entropy
- The full loss per

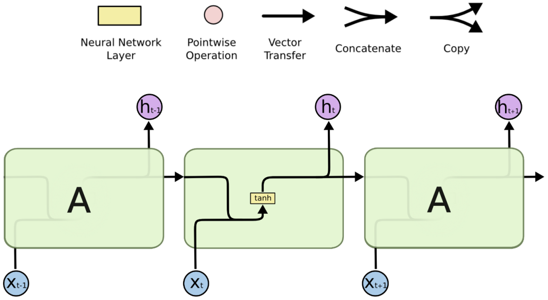
$$((x_1, x_2, \ldots, x_T), (y_1 = x_2, y_2 = x_3, \ldots, y_T = x_{T+1}))$$

pair is an average of the loss across all time steps

$$\frac{\sum_{t=1}^{T} L_t(\hat{y}_t, y_t)}{T}$$

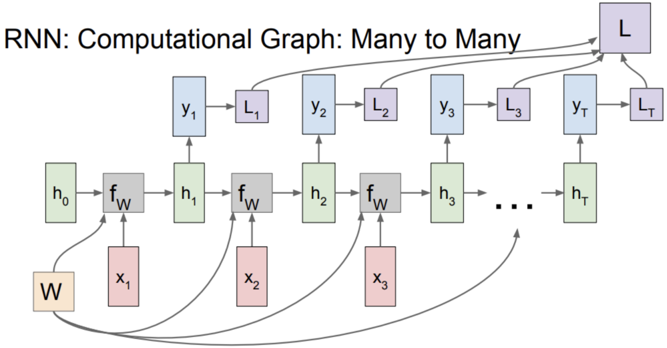
Recurrent Neural Networks: Recursions

- A picture of a RNN might look like this



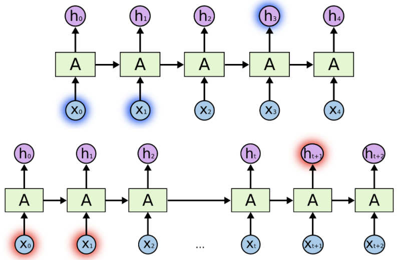
The repeating module in a standard RNN contains a single layer.

Recurrent Neural Networks: Computational Graph



Recurrent Neural Networks: Recursions

- As the data gets longer, an RNN can no longer propagate dependencies between things far in the future and elements earlier on



Why?

Recurrent Neural Networks: Recursions

- Let's do a small example where $W_h = a$, $W_x = b$ and $b_h = c$ are all scalars and we have the recursion $h_t = ah_{t-1} + bx_t + c$
- Assume also we have $\hat{y}_t = dh_t$ and the loss is $L_t(\hat{y}_t, y_t) = (\hat{y}_t - y_t)^2$
- Here, a, b, c, d are parameters of dimension 1
- What is $\frac{\partial L_1}{\partial a}$?
- Here we have $\frac{\partial L_1}{\partial a} = \frac{\partial L_1}{\partial \hat{y}_1} \frac{\partial \hat{y}_1}{\partial h_1} \frac{\partial h_1}{\partial a} = 2(\hat{y}_1 - y_1)dh_0$

Recurrent Neural Networks: Recursions

- Let's do a small example where $W_h = a$, $W_x = b$ and $b_h = c$ are all scalars and we have the recursion $h_t = ah_{t-1} + bx_t + c$
- Assume also we have $\hat{y}_t = dh_t$ and the loss is $L_t(\hat{y}_t, y_t) = (\hat{y}_t - y_t)^2$
- What is $\frac{\partial L_2}{\partial a}$?
- Here we have $\frac{\partial L_2}{\partial a} = \frac{\partial L_2}{\partial \hat{y}_2}\frac{\partial \hat{y}_2}{\partial h_2}\frac{\partial h_2}{\partial a}$
- This is actually $\frac{\partial L_2}{\partial \hat{y}_2}\frac{\partial \hat{y}_2}{\partial h_2}[\frac{\partial h_2}{\partial a} + \frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial a}]$
- This is $2(\hat{y}_2 - y_2)d(h_1 + ah_0)$
- Note the appearance of the a term

Recurrent Neural Networks: Recursions

- What is $\frac{\partial L_4}{\partial a}$?
- Here we have $\frac{\partial L_4}{\partial a} = \frac{\partial L_4}{\partial \hat{y}_4}\frac{\partial \hat{y}_4}{\partial h_4}\frac{\partial h_4}{\partial a}$
- This is actually

$$\frac{\partial L_4}{\partial a} = \sum_{t=1}^{3} \frac{\partial L_4}{\partial \hat{y}_4}\frac{\partial \hat{y}_4}{\partial h_4}\frac{\partial h_4}{\partial h_t}\frac{\partial h_t}{\partial a} + \frac{\partial L_4}{\partial \hat{y}_4}\frac{\partial \hat{y}_4}{\partial h_4}\frac{\partial h_4}{\partial a}$$

- Simplified, this is

$$2(\hat{y}_2-y_2)da^3h_0+2(\hat{y}_2-y_2)da^2h_1+2(\hat{y}_4-y_4)dah_2+2(\hat{y}_4-y_4)dh_3$$

- What happens when a is small or large?

Recurrent Neural Networks: Gradients Issues

- We have the gradient $\frac{\partial L_4}{\partial a}$ was

  $$2(\hat{y}_2 - y_2)da^3 h_0 + 2(\hat{y}_2 - y_2)da^2 h_1 + 2(\hat{y}_4 - y_4)dah_2 + 2(\hat{y}_4 - y_4)dh_3$$

- If a is small, we have that the term measuring the dependency between the current time step and the one in the past is less and less
- This is called the vanishing gradient problem
- If a is too large, we'll have an infinite gradient, another problem
- This is called the exploding gradient problem

Recurrent Neural Networks: Solving Gradient Issues

- The exploding gradient problem can be solved by gradient clipping
- The vanishing gradient problem might be solved by other ways
  - Residual connections
  - Xavier Initialization
  - Momentum based optimizers (Adam)
- None of the above are principled, they "might" work
- In practice, vanilla RNNs do not perform as good as they theoretically can due to the vanishing gradient problem
- More complicated (LSTM, Transformers, Attention) models are used

Recurrent Neural Networks: Truncated BP

- Doing back propagation in RNN models can be expensive since you need to go all the way back to $t = 0$
- We generally have

$$\frac{\partial L_t}{\partial a} = \sum_{s=1}^{t} \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial h_s} \frac{\partial h_s}{\partial a}$$

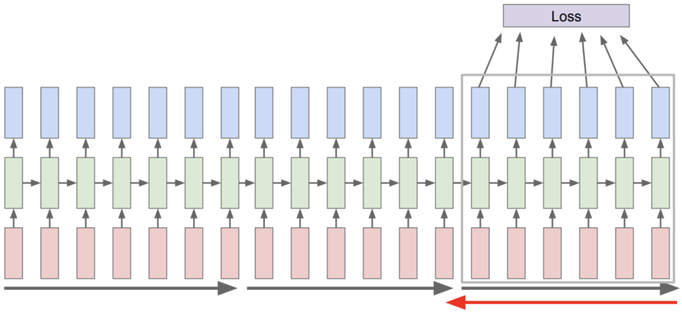
  - I use $\frac{\partial h_t}{\partial h_t} = 1$ for the case $s = t$
- One idea is to simplify the above to be

$$\frac{\partial L_t}{\partial a} = \sum_{s=t-k}^{t} \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial h_s} \frac{\partial h_s}{\partial a}$$

- Thus, for each term in the loss L_t, we only look back a fixed window when computing the gradients
- This works somewhat, but now we are back to the finite context case

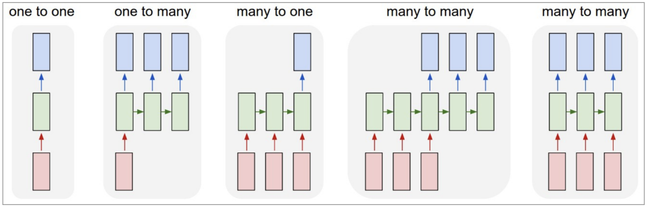
Recurrent Neural Networks: Truncated BP Visual

Truncated Backpropagation through time



Recurrent Neural Networks: Variants

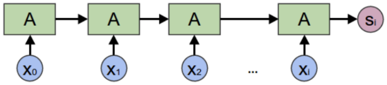
- Many data and modeling situations can be addressed via Recurrent Neural Networks

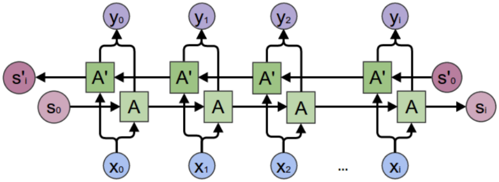


Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right: **(1)** Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). **(2)** Sequence output (e.g. image captioning takes an image and outputs a sentence of words). **(3)** Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). **(4)** Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French). **(5)** Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.

Recurrent Neural Networks: Bidirectional RNN

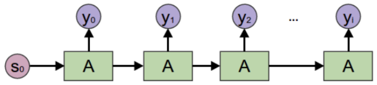
- You can also have models which have context on both sides
- Imagine the sentiment analysis of "The great movie was actually really just expensive and bad."
- You'd like to know that "great" has a bit of irony in it

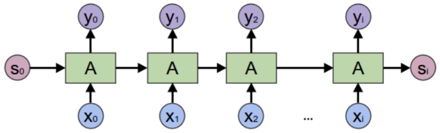




Recurrent Neural Networks: Bidirectional RNN

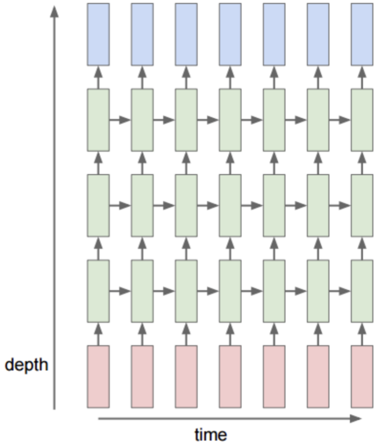
- When you are generating text, you can't use a bidirectional RNN since you are cheating
- Sentiment analysis is one example where a bidirectional RNN might be a good idea: each word has the best context of those in the sentence!





Recurrent Neural Networks: Multilayer RNN

- Multilayer RNN is also possible, but now you have many more parameters
- Note that more layers = vertical whereas more time = horizontal



Outline

- Language Modeling
- Recurrent Neural Networks
- LSTM
- Decoding

LSTM

- RNN suffer from vanishing gradient problems
- Because of this, people have tried to come up with variants that resolve this issue
- LSTM (1997 !) are one of the most popular variants
- Empirically, an LSTM has been shown to have a memory of about 100 time steps, whereas a RNN has about just 7

LSTM

- Dynamics of the LSTM are as follows
- We still have data $X = (x_1, x_2, \ldots, x_T)$ but now we have a more complicated interaction
- Specifically, each time step not only has h_t, but now there is a cell state c_t that is carried forward
- We have

$$(f_t, i_t, o_t, g_t) = W_h h_{t-1} + W_x x_t + b$$

- Where the state is

$$c_t = \sigma(f_t) \odot c_{t-1} + \sigma(i_t) \odot tanh(g_t)$$

- And the hidden state is

$$h_t = \sigma(o_t) \odot tanh(c_t)$$

LSTM

- The key new feature here is

$$c_t = \sigma(f_t) \odot c_{t-1} + \sigma(i_t) \odot tanh(g_t)$$

  - $\sigma(f_t)$ is the forget gate
  - $\sigma(i_t)$ is the update gate
- This cell state is a combination of old information and new information: the sole purpose of this state is to help the network not forget!
- And the hidden state is $h_t = \sigma(o_t) \odot tanh(c_t)$ is a function of new information and the new/old information in the cell state
- As we will show, this affects the gradients: there is less chance of vanishing gradient phenomenon and a current loss update can use information from far out in the past!

LSTM: Parameters

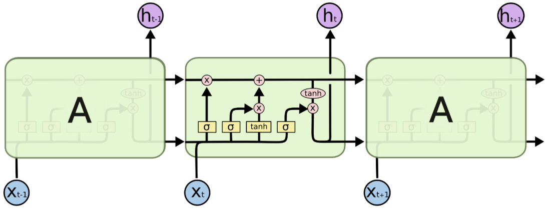
- In the above
- $W_h \in \mathbb{R}^{d_h \times 4d_h}$
- $W_x \in \mathbb{R}^{d_x \times 4d_h}$
- $b \in \mathbb{R}^{4d_h}$
- $h_0, c_0 \in \mathbb{R}^{d_h}$
- σ is the sigmoid
- $\odot$ is the Hadamard product, $[A \odot B]_{ij} = A_{ij} B_{ij}$

LSTM: Parameters

- So we can be more specific with the parameters
- Note that in reality we can think of $W_x = [W_{fx}, W_{ix}, W_{ox}, W_{gx}]$
- We can thus break up the weight matrix into blocks, with a specific block being for a specific parameter
- A similar idea applied to W_h and $b = [b_f, b_i, b_o, b_g]$

LSTM: Pictures

- Here is a nice picture of what is going on in an LSTM



The repeating module in an LSTM contains four interacting layers.

LSTM: Idea

- The big idea of LSTM networks is that the added c_t allows the networks to remember things
- Put another way, if we write out gradients, there is a greater chance of long term dependencies
- Why?
- Recall, terms of the type $\frac{\partial h_t}{\partial h_{t-k}}$ are critical to gradient estimates for the RNN, let's focus on them since they are a (critical) part of all gradients
- For an LSTM, we also have $\frac{\partial c_t}{\partial c_{t-k}}$ to consider

RNN: Gradients

- Consider again an RNN and simplify it a bit so that $h_t = \sigma(W_h h_{t-1})$
- Recall $\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z))$
- If we do a gradient computation we get that

$$\frac{\partial h_t}{\partial h_{t-k}} = \prod_{s=t-k}^{t-1} W_h \sigma(W_h h_s)(1 - \sigma(W_h h_s))$$

- This is

$$\frac{\partial h_t}{\partial h_{t-k}} = W_h^k \prod_{s=t-k}^{t-1} \sigma(W_h h_s)(1 - \sigma(W_h h_s))$$

RNN: h_t Gradients

- This is

$$\frac{\partial h_t}{\partial h_{t-k}} = W_h^k \prod_{s=t-k}^{t-1} \sigma(W_h h_s)(1 - \sigma(W_h h_s))$$

- When $W_h h_s$ is large or small, we get vanishing gradients
- Even when $W_h h_s = 0$, the maximum point of $\sigma(x)(1 - \sigma(x))$, we get $1/4$, so we get a small product
- The term W_h^k can also cause vanishing or exploding gradients, mainly due to bad initialization
- Upshot: A lot of things can go wrong, and usually do

LSTM: c_t Gradients

- Now look at the LSTM cell
  $c_t = \sigma(f_t) \odot c_{t-1} + \sigma(i_t) \odot tanh(g_t)$
- What is $\frac{\partial c_t}{\partial c_{t-k}}$?
- Ignoring terms that have to do with x and b_f, this gives
- $$\frac{\partial c_t}{\partial c_{t-k}} = \frac{\partial c_t}{\partial c_{t-1}} \dots \frac{\partial c_{t-k+1}}{\partial c_{t-k}} = \prod_{s=t-k}^{t-1} \sigma(W_{fh} h_s + \dots)$$
- In this case, gradients only vanish if $W_{fh} h_s$ is very small: we have much less things that can go wrong!
  - Also, since the σ function squashes, the gradient will not explode because $W_{fh} h_s$ is very large we'll get 1
  - We don't have a bad "W_h^k" term like we had before

LSTM: h_t Gradients

- Now look at the LSTM hidden unit $h_t = \sigma(o_t) \odot tanh(c_t)$
- Keep in mind the LSTM cell
  $c_t = \sigma(f_t) \odot c_{t-1} + \sigma(i_t) \odot tanh(g_t)$
- Recall $\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z))$ and $\frac{dtanh(z)}{dz} = 1 - tanh^2(z)$
- What about the term $\frac{\partial h_t}{\partial h_{t-k}} = \frac{\partial h_t}{\partial h_{t-1}} \dots \frac{\partial h_{t-k+1}}{\partial h_{t-k}}$?

LSTM: h_t Gradients

- If we focus just on $\frac{\partial h_t}{\partial h_{t-1}}$ we have to simplify

$$\partial(\sigma(o_t) \odot tanh(c_t))/\partial h_{t-1}$$

- This is

$$\frac{\partial \sigma(o_t)}{\partial h_{t-1}} tanh(c_t) + \sigma(o_t) \frac{\partial tanh(c_t)}{\partial h_{t-1}}$$

- Which is

$$\sigma(o_t)(1 - \sigma(o_t)) tanh(c_t) W_{oh} +$$
$$\sigma(o_t)(1 - tanh^2(c_t))[\frac{\partial \sigma(f_t)}{\partial h_{t-1}} c_{t-1} + \frac{\partial(\sigma(i_t) \odot tanh(g_t))}{\partial h_{t-1}}]$$

LSTM: h_t Gradients

- We want to simplify the term

$$\frac{\partial \sigma(f_t)}{\partial h_{t-1}} c_{t-1} + \frac{\partial(\sigma(i_t) \odot tanh(g_t))}{\partial h_{t-1}}$$

- The left term simplifies to

$$\frac{\partial \sigma(f_t)}{\partial h_{t-1}} c_{t-1} = \sigma(f_t)(1 - \sigma(f_t)) W_{fh} c_{t-1}$$

- Th right term simplifies to

$$\begin{aligned} \frac{\partial(\sigma(i_t) \odot tanh(g_t))}{\partial h_{t-1}} &= \sigma(i_t)(1 - \sigma(i_t)) W_{ih} tanh(g_t) \\ &+ (1 - tanh^2(g_t))\sigma(i_t) W_{gh} \end{aligned}$$

LSTM: h_t Gradients

- Putting this together, we get

$$\begin{aligned}
&tanh(\bullet)\sigma(\bullet)(1-\sigma(\bullet))W_{oh}+\\
&\qquad\sigma(\bullet)(1-tanh^2(\bullet))[c_{t-1}\sigma(\bullet)(1-\sigma(\bullet))W_{fh}+\\
&\qquad\sigma(\bullet)(1-tanh^2(\bullet))W_{gh}+tanh(\bullet)\sigma(\bullet)(1-\sigma(\bullet))W_{ih}]
\end{aligned}$$

- Let $\gamma(\bullet)$ be any generic term like σ or *tanh*, which are all bounded and assume $\gamma^2(\bullet)=\gamma(\bullet)$
- We get

$$\gamma(\bullet)W_{oh}+\gamma(\bullet)[c_{t-1}\gamma(\bullet)W_{fh}+\gamma(\bullet)W_{gh}+\gamma(\bullet)W_{ih}]$$

LSTM: h_t Gradients

- Using $\gamma^2 = \gamma$, this is further reduced to

$$\gamma(\bullet)W_{oh} + c_{t-1}\gamma(\bullet)W_{fh} + \gamma(\bullet)W_{gh} + \gamma(\bullet)W_{ih}$$

- So we have

$$\frac{\partial h_t}{\partial h_{t-k}} = \prod_{s=t-k}^{t-1} [\gamma(\bullet)W_{oh} + c_{t-1}\gamma(\bullet)W_{fh} + \gamma(\bullet)W_{gh} + \gamma(\bullet)W_{ih}]$$

- So, although $\frac{\partial c_t}{\partial c_{t-k}}$ has no exponential products, $\frac{\partial h_t}{\partial h_{t-k}}$ does, we can still have vanishing gradients for this term
- We must still be careful with initialization, and we might need to do gradient clipping
- However, h_t being unsafe from vanishing gradients is not as critical here as it is for a standard RNN; c_t can be used to pass on long range dependencies - the model has this capacity and h_t can use that

LSTM: Peephole Connected LSTM

- We can generalize the LSTM model further, for example we can modify the equations above
- We now pass the old state c_{t-1} in:

$$(f_t, i_t) = W_h h_{t-1} + W_x x_t + W_c c_{t-1} + b$$

- $g_t = W_{gh} h_{t-1} + W_{gx} x_t + b_g$ (same as before)
- $c_t = \sigma(f_t) \odot c_{t-1} + \sigma(i_t) \odot tanh(g_t)$ (same as before)
- We now have c_t is used for the output: $o_t = W_{oh} h_{t-1} + W_{ox} x_t + W_{oc} c_t + b_o$
- The idea of these models is to fix the case when $\sigma(o_t) \sim 0$ because then $h_t = \sigma(o_t) \odot tanh(c_t) \sim 0$
- Allowing the gates to have access to the cell state has been shown empirically to fix this issue

GRU: Simpler LSTM

- We can generalize the LSTM model further, for example we can modify the equations above
- A GRU defines reset and update gate logits via

$$(r_t, z_t) = W_h h_{t-1} + W_x x_t + b$$

- For a GRU, we have $g_t = W_{gh}(\sigma(r_t) h_{t-1}) + W_{gx} x_t + b_g$
- Now, $h_t = \sigma(z_t) \odot tanh(g_t) + (1 - \sigma(z_t)) \odot h_{t-1}$
  - GRU have a single state
  - They combine the gates $\sigma(f_t)$ and $\sigma(i_t)$ into one gate $\sigma(z_t)$
  - The reset gate $\sigma(r_t)$ allows the GRU to use or ignore the previous state when computing the next candidate g_t
- GRU models offer less parameters than a LSTM and have been shown to do about as well

Outline

- Language Modeling
- Recurrent Neural Networks
- LSTM
- Decoding

Decoding: Example

- Once we train a model, we have the problem of decoding: How do we generate text using the model?
- Usually, we start with a context like "The man walks" or a special start token "*START*" and begin decoding with the model
- One very simple variant: greedy decoding
- This is one of the oldest and easiest heuristics

Decoding: Example

- Basic idea: Always pick the highest probability next word and use that as your next word
- Once picked, add the word to your context and repeat until some terminal condition happens
- Big idea is that, for example, we have

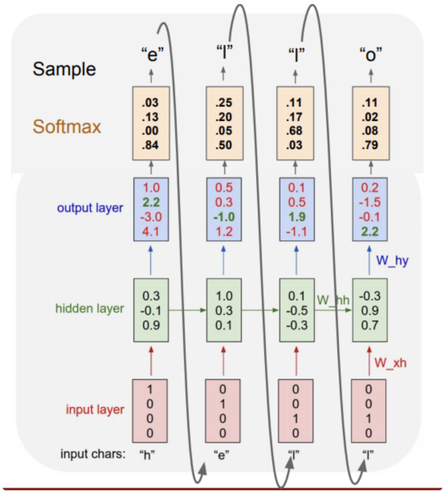
$$P(The, man, walks, y) = P(y|The, man, walks)P(The, man, walks)$$

- We can just focus on $P(y|The, man, walks)$ if we want to maximize $P(The, man, walks, y)$

Decoding: Greedy Search Example

- So suppose the context is "*The man walks*"
- Get $argmax_y P(y|The, man, walks)$ and suppose this is "*down*"
  - Note that technically we don't work with strings, but the integers that each string represent
- Now the context is "The man walks down"
- Get $argmax_y P(y|The, man, walks, down)$ and suppose this is "*the*"
- Now the context is "*the*"
- Etc. Repeat this until the context is done or until a special end "*END*" token is generated
- Once picked, add the word to your context and repeat over and over

Decoding: Greedy Search Example



Decoding: Greedy is not Optimal

- But remember we really want to solve

$$\max_{y_1, y_2, \ldots, y_T} \left(P(y_1, y_2, \ldots, y_T)\right)$$

- Greedy, generally, will not give the optimal solution - we keep just one choice!
- Also, human generated text is not generated this way: you don't always pick the word with the next highest probability to generate the things you say
- Sometimes, you make a lower probability choice but in the long run this leads to higher probability choices overall
- "The Himalayan Mountains are very nice"
- Another idea: Beam Search

Decoding: Beam Search

- Instead of picking the next highest probability word to add to your context, store a beam of size k as your set of candidates
- Example: Assume $k = 2$ and your context is again "*The man walks*"
- When we look at

$$argmax_y P(y|The, man, walks)P(The, man, walks)$$

  we don't just pick $y =$ "*down*", we also pick "*fast*", which has the second highest probability
- Now you have two context, "*The man walks down*" and "*The man walks fast*"
- Next time, consider

$$P(y|The, man, walks, down)P(The, man, walks, down)$$

  and

$$P(y|The, man, walks, fast)P(The, man, walks, fast)$$

  and get the $k = 2$ maximum probability tokens

Decoding: Beam Search

- Note that now we have to consider $k * |V|$ scores and we want to just select $k = 2$
- Note that we might have that among

  $$P(y|The, man, walks, down)P(The, man, walks, down)$$

  and

  $$P(y|The, man, walks, fast)P(The, man, walks, fast)$$

  the first case has the highest probability next words
- I.e. We might have that

  $$P(The, man, walks, down, the)$$

  and

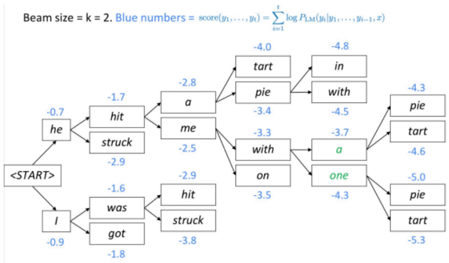
  $$P(The, man, walks, down, into)$$

  are the highest probabilities

Decoding: Beam Search

- Effectively, the new context becomes "*The man walks down the*" and "*The man walks down into*" so that "*The man walks fast*" is dropped
- Usually, Beam Search uses the trick below since it avoids the product of small numbers

$$\log P(y_1, y_2, \ldots, y_T) = \sum_{t=1}^{T} \log P(y_t | y_{t-1}, \ldots, y_1)$$



References

- Backpropogating an LSTM: A Numerical Example
- Understanding LSTM Networks
- Character Aware Neural Language Models
- Neural Machine Translation
- Andrej Karpathy's BLOG on RNNs
- Machine Learning Notebook