

SEQUENCE

S&DS 365 / 665
Intermediate Machine Learning

RNNs, LSTMs, GRUs, and All That

November 15

Yale

Home stretch!

- Assignment 4 due tonight
- Assignment 5 (last!) posted today
 - ▶ Policy iteration
 - ▶ GRUs (today's topic)
- Quiz 5 (last!) November 30

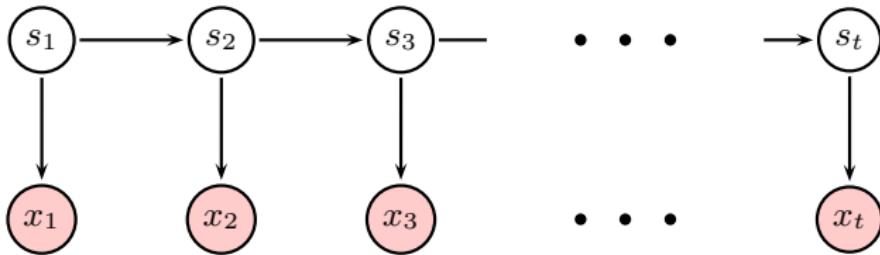
Recall: Hidden Markov Model

In an HMM, the next word is generated from a latent variable

- State is chosen stochastically (that is, probabilistically)
- Introduces dependence on earlier words that are further back than the previous time (non-Markov on observations)

Hidden Markov Model

The graphical model looks like this:



Here x_t is observable (word) at time t and s_t is unobserved state.

Hidden Markov Models

Probability of word sequence:

$$p(x_1, \dots, x_n) = \sum_{s_1, \dots, s_n} \prod_{t=1}^n p(s_t | s_{t-1}) p(x_t | s_t)$$

Hidden Markov Models

Probability of word sequence:

$$p(x_1, \dots, x_n) = \sum_{s_1, \dots, s_n} \prod_{t=1}^n p(s_t | s_{t-1}) p(x_t | s_t)$$

Probability of the next word:

$$p(x_t | x_1, \dots, x_{t-1}) = \sum_{s_t} p(s_t | x_1, \dots, x_{t-1}) p(x_t | s_t)$$

This is efficiently computed using dynamic programming

Hidden Markov Models

Probability of the next word:

$$p(x_t | x_1, \dots, x_{t-1}) = \sum_{s_t} p(s_t | x_1, \dots, x_{t-1}) p(x_t | s_t)$$

This is efficiently computed using dynamic programming:

$$p(s_t | x_1, \dots, x_{t-1}) \propto \sum_{s_{t-1}} p(s_{t-1} | x_1, \dots, x_{t-2}) p(x_{t-1} | s_{t-1}) p(s_t | s_{t-1})$$

Hidden Markov Model

- Next word x_t is predicted from the state s_t
- State is a summary of information from the past x_1, \dots, x_{t-1}
- State evolves stochastically

Hidden Markov Models: Estimation

The usual algorithms for estimating the parameters are iterative:

- Use the forward-backward algorithm to compute the probability that the outputs are generated from each state
- Treat the data as labeled, with these weights, and compute the relative frequencies
- Repeat
- Maximum likelihood (or MAP) estimation

Recurrent neural networks

- Can be thought of as a type of language model
- Similar to hidden Markov models, but the *state evolves deterministically*, not stochastically
- HMMs are mixture models; RNNs are not
- The state is distributed, not categorical as for HMMs
- We'll describe this using characters rather than words — could do either

HMMs vs RNNs

In an HMM:

$$p(x_t | x_1, \dots, x_{t-1}) = \sum_{s_t} p(s_t | x_1, \dots, x_{t-1}) p(x_t | s_t)$$

This is efficiently computed using dynamic programming

HMMs vs RNNs

In an HMM:

$$p(x_t | x_1, \dots, x_{t-1}) = \sum_{s_t} p(s_t | x_1, \dots, x_{t-1}) p(x_t | s_t)$$

This is efficiently computed using dynamic programming

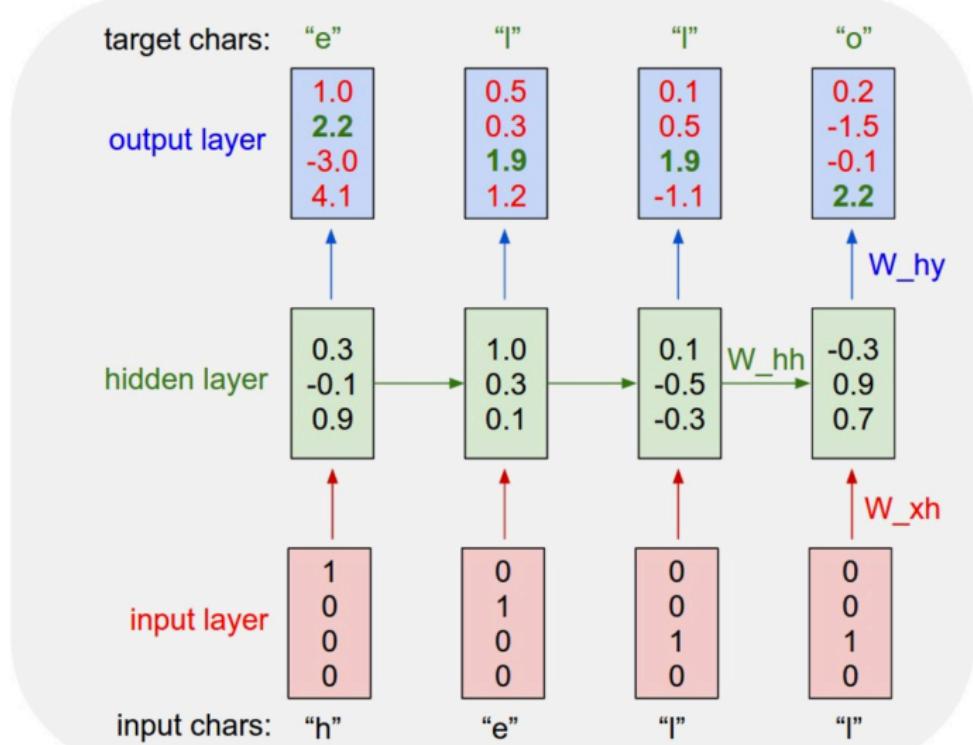
In an RNN:

$$p(x_t | x_1, \dots, x_{t-1}) = p(x_t | s_t)$$

where $s_t = s(x_1, \dots, x_{t-1})$ is a nonlinear function

RNNs

<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>



RNNs

This means

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

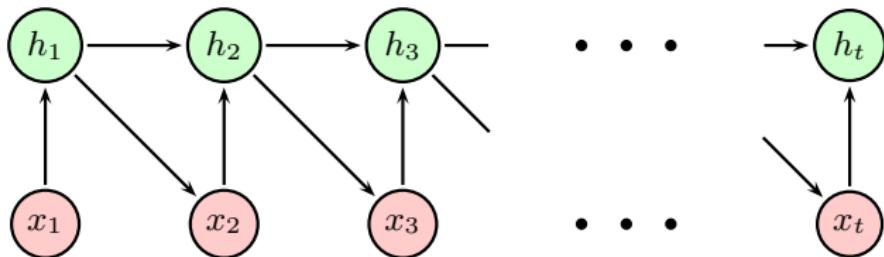
$$y_t = W_{hy}h_t + b_y$$

$$x_{t+1} | y_t \sim \text{Multinomial}(\pi(y_t))$$

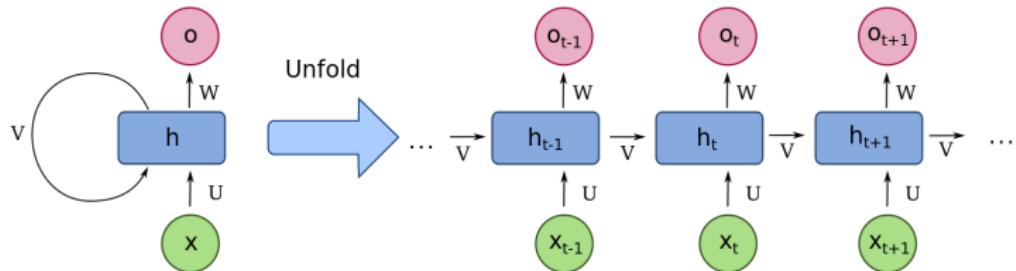
where $\pi(\cdot)$ is the soft-max function.

In this illustration, x_t is the “1-hot” representation of a character, $W_{xh} \in \mathbb{R}^{3 \times 4}$, $W_{hh} \in \mathbb{R}^{3 \times 3}$ and $W_{hy} \in \mathbb{R}^{4 \times 3}$.

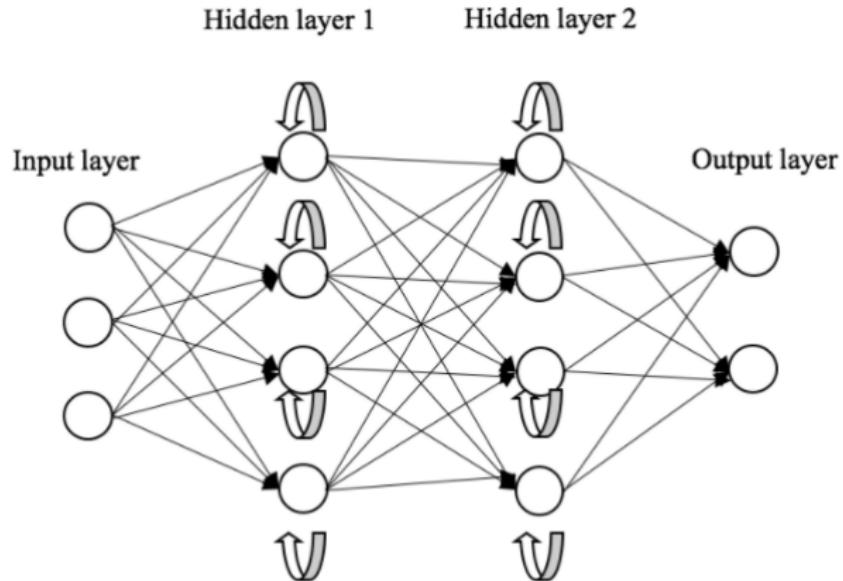
RNN graphical structure: Different views



RNN graphical structure: Different views

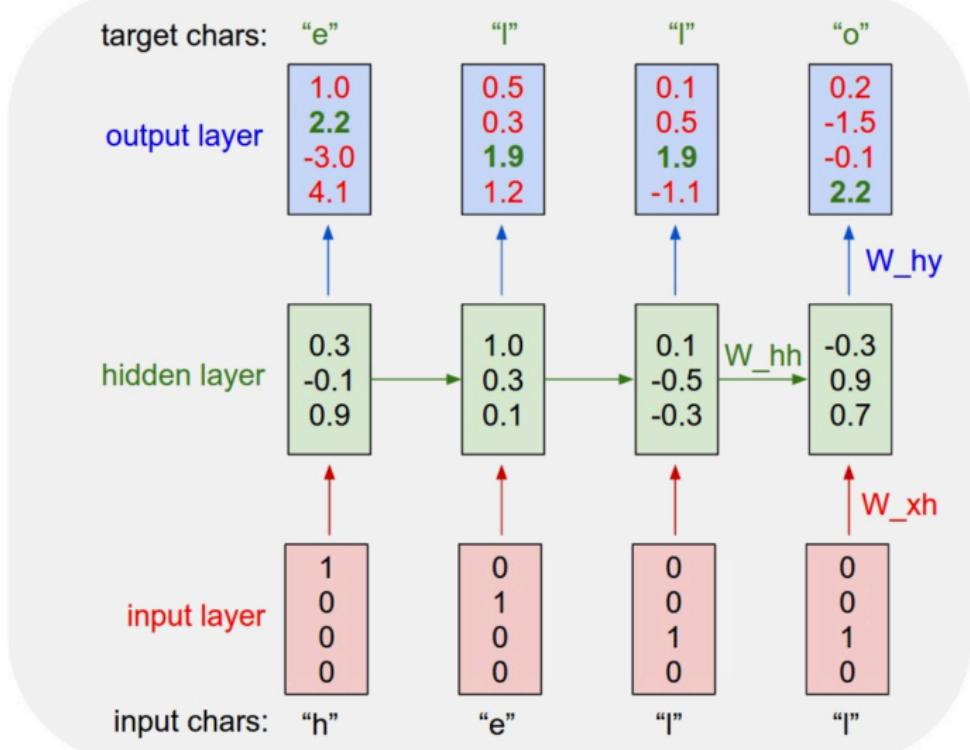


RNN graphical structure: Different views



General Form of RNNs

RNN architecture



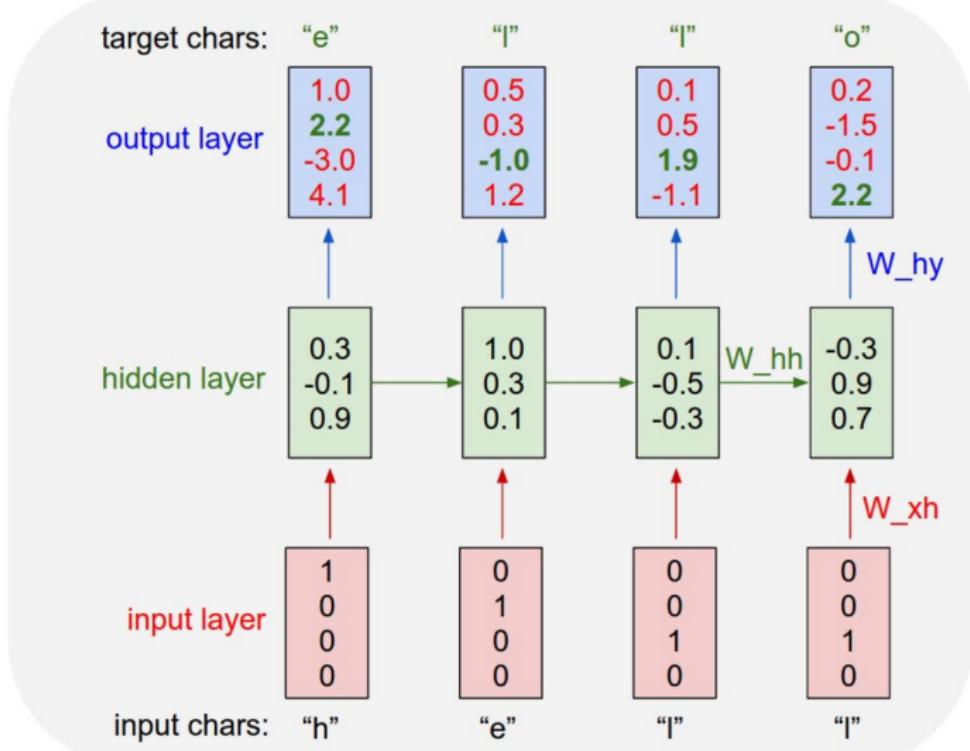
RNN Python Code

```
class RNN:  
    # ...  
    def step(self, x):  
        # update the hidden state  
        self.h = np.tanh(np.dot(self.W_hh, self.h) + np.dot(self.W_xh, x))  
        # compute the output vector  
        y = np.dot(self.W_hy, self.h)  
        return y
```

One hidden layer:

```
rnn = RNN()  
y = rnn.step(x) # x is an input vector, y is the RNN's output vector
```

One hidden layer



RNN Python Code

```
class RNN:  
    # ...  
    def step(self, x):  
        # update the hidden state  
        self.h = np.tanh(np.dot(self.W_hh, self.h) + np.dot(self.W_xh, x))  
        # compute the output vector  
        y = np.dot(self.W_hy, self.h)  
        return y
```

Two hidden layers:

```
y1 = rnn1.step(x)  
y = rnn2.step(y1)
```

Loss function

The model is trained to assign high probability to the word (or character) that appears next:

$$\text{Loss} = -\frac{1}{T} \sum_{t=1}^T \log p(x_t | h_t)$$

If the last layer assigns a score to word v as

$$f(v, h_t) = \beta_v^T h_t$$

then this is given by

$$\text{Loss} = -\frac{1}{T} \sum_{t=1}^T \left\{ \beta_{x_t}^T h_t - \log \left(\sum_{v \in V} e^{\beta_v^T h_t} \right) \right\}$$

In terms of our earlier notation, β_v^T would be the v -th row of the matrix W_{hy} , and the intercept would be zero

Hallucinated Shakespeare

PANDARUS:

Alas, I think he shall be come approached and the day
When little strain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Second Lord:

They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:

Come, sir, I will make did behold your worship.

VIOLA:

I'll drink it.

Interpreting RNNs

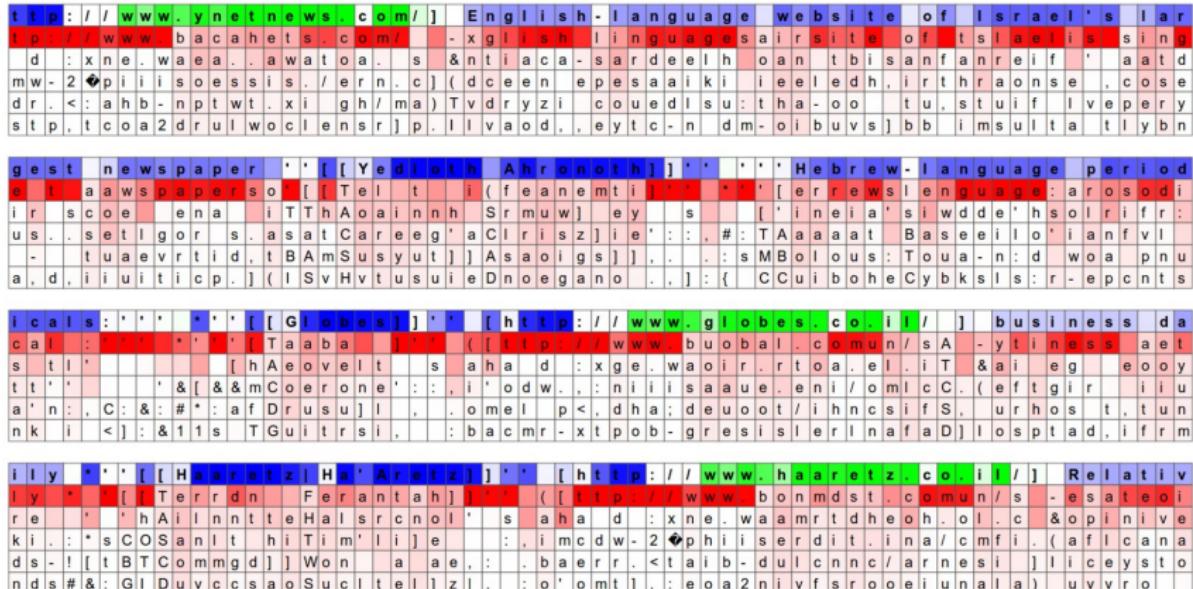
Each component is a tanh-thresholded linear function of the previous states and current input—a “neuron”

If the j th neuron h_{tj} is close to ± 1 then it is very “excited.” This is colored green in the following illustration. If it is near zero, then it is “not excited” or cool—this is colored blue.

Below the neuron color, the five most probable characters c are shown — ordered in decreasing order of y_{tc} and hence $\pi(y_t)_c$.

Three layers, with 512 neurons on each level. Most will be completely uninterpretable. Recall—it’s generally even hard to interpret coefficients in a least squares linear regression model!

Interpretation



The neuron highlighted in this image seems to get very excited about URLs and turns off outside of the URLs. The LSTM is likely using this neuron to remember if it is inside a URL or not.

Interpretation

''''[[Jerusalem Report]]'''	[http://www.jrep.com/]	Left-of-center En
''[[To a usual maoz gur ()]]'''	(http://www.bsinioom/)	-ial affenter (ng
[[['[Cassmene] Beaonds s a a d : xne. waaaaoca. s &ato- nf h hisum- ouc		
.'s mFurnls i etallsa': ., i' c dw- 2tpiisoeg. er/. a](oseswr- ciddrs [mt		
: AqDenebiutn Cipre e], . b1emr. 9: ahb- npumughnmp) Teiretu: eoseodsald		
#T&TfSi wrp e] al uvelru s: - mprts<#moa2deyshil r] c. Augl, 1p, larc : fae		
glish [[weekly newspaper]]'''	[YNNet News]]'''	[http://www.ynetnews.co
lish c[Caaky]cawspaper]]'''	[hTaA at]]'''	(http://www.bacahets.co
iaci-lhSoipli sec lenps . . . [Co wess] s a a d : xne. waea.. awatoa		
ena, pCietnedloox] gicill s [sAmFe Sahonjt': ., imomw- 2#piisoessi. /er		
syz . spenn alrue lrra . '#: oDuFreiuep. . : b1edr. <: ahb- nptwt. xigh		
a dpeamArbdeorpi tee] dts- T{ [BaAvTp oSwao, . . oacstp, tcoa2drulwocliens		
om] English-language website of Israel's largest newspaper'''	[Yed	
ml -xglish lingua gesair site of tsraelis singl aawspaper[[Tel		
. s &ntiaca-sardeelh oantbisanfanreif' aatdir scoe enai iTThAoai		
n.c) (dceen epesaai ki ieledh,irthraonse, coseus. setigors. asatCare		
/ma) Tvdryzi couedlsu: tha- oo tu, stuif lvevery - tuaevrtid, tBAmSusy		
r] p. lvaod, . eytc- n dm- oibuys] bb imsulta tiybna, d. iiuiticp.] (ISvHvtu		
loth Ahronoth]]'''	Hebrew-language periodicals:'''	[[Globes]]'''
t i (feanemti)]'''	[errewslengua: arosodical]]'''	[Taaba]]'''
nh Srmuw] ey s [ineia siwddesoirifr: stl'		[hAeovelt s
eg' aCrlisz] ie': ., #: TAaaaat Baseeilo'ianf vltt' . . & [&&mCoerone': :		
ut]] Asa oigs], . . s MBolous: Toua- n: d woapnu a'n:, C: & #: afDrusu]] ,		
suie Dnoegano, .,]:{ C Cuiboh e Cybksls: r- epcnts nk i <]: & 11s T Guitrsi,		

The highlighted neuron here gets very excited when the RNN is inside the [[]] markdown environment and turns off outside of it.

Interestingly, the neuron can't turn on right after it sees the character "[", it must wait for the second "[" and then activate. This task of counting whether the model has seen one or two "[" is likely done with a different neuron.

Interpretation

Cell sensitive to position in line:

The sole importance of the crossing of the Berezina lies in the fact that it plainly and indubitably proved the fallacy of all the plans for cutting off the enemy's retreat and the soundness of the only possible line of action--the one Kutuzov and the general mass of the army demanded--namely, simply to follow the enemy up. The French crowd fled at a continually increasing speed and all its energy was directed to reaching its goal. It fled like a wounded animal and it was impossible to block its path. This was shown not so much by the arrangements it made for crossing as by what took place at the bridges. When the bridges broke down, unarmed soldiers, people from Moscow and women with children who were with the French transport, all--carried on by vis inertiae--pressed forward into boats and into the ice-covered water and did not, surrender.

Cell that turns on inside quotes:

"You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."

Cell that robustly activates inside if statements:

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
    siginfo_t *info)
{
    int sig = next_signal(pending, mask);
    if (sig) {
        if (current->notifier) {
            if (sigismember(current->notifier_mask, sig)) {
                if (!!(current->notifier)(current->notifier_data)) {
                    clear_thread_flag(TIF_SIGPENDING);
                    return 0;
                }
            }
        }
        collect_signal(sig, pending, info);
    }
    return sig;
}
```

rnn-demo.ipynb: numpy version



We'll train an "np-complete" RNN on the complete works of William Shakespeare, downloaded from [Project Gutenberg](#) (specifically, [this link](#)).

This uses a simple, direct implementation of backpropagation for RNNs, paralleling what we did for simple feedforward networks.

```
In [ ]: def lossFun(inputs, targets, hprev):
    """
    inputs,targets are both list of integers.
    hprev is Hx1 array of initial hidden state
    returns the loss, gradients on model parameters, and last hidden state
    """
    xs, hs, ys, ps = {}, {}, {}, {}
    hs[-1] = np.copy(hprev)
    loss = 0

    # forward pass
    for t in range(len(inputs)):
        xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
        xs[t][inputs[t]] = 1
        hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
        ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
        ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
        loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)

    # backward pass: compute gradients going backwards
    dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
    dbh, dby = np.zeros_like(bh), np.zeros_like(by)
    dhnext = np.zeros_like(hs[0])
    for t in reversed(range(len(inputs))):
        dy = np.copy(ps[t])
        dy[targets[t]] -= 1 # backprop into y. see http://cs231n.github.io/neural-networks-case-stu
        dWhy += np.dot(dy, hs[t].T)
        dby += dy
        dh = np.dot(Why.T, dy) + dhnext # backprop into h
        ddraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
        dbh += ddraw
        dWxh += np.dot(ddraw, xs[t].T)
        dWhh += np.dot(ddraw, hs[t-1].T)
        dhnext = np.dot(Whh.T, ddraw)

    for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
        np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients

    return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
```

Tensorflow implementation

Text generation with an RNN

[Run in Google Colab](#)[View source on GitHub](#)[Download notebook](#)

This tutorial demonstrates how to generate text using a character-based RNN. You will work with a dataset of Shakespeare's writing from Andrej Karpathy's [The Unreasonable Effectiveness of Recurrent Neural Networks](#). Given a sequence of characters from this data ("Shakespear"), train a model to predict the next character in the sequence ("e"). Longer sequences of text can be generated by calling the model repeatedly.



Note: Enable GPU acceleration to execute this notebook faster. In Colab: *Runtime > Change runtime type > Hardware accelerator > GPU*. If running locally make sure TensorFlow version ≥ 1.11 .

This tutorial includes runnable code implemented using `tf.keras` and `eager execution`. The following is sample output when the model in this tutorial trained for 30 epochs, and started with the string "Q":

QUEENE:

I had thought thou hadst a Roman; for the oracle,
Thus by All bids the man against the word,
Which are so weak of care, by old care done;
Your children were in your holy love,
And the precipitation through the bleeding throne.

BISHOP OF ELY:

Marry, and will, my lord, to weep in such a one were prettiest;
Yet now I was adopted heir
Of the world's lamentable day,
To watch the next way with his father with his face?

ESCALUS:

The cause why then we are all resolved more sons.

Memory circuits

A variant called “Long Short-Term Memory” RNNs has a special hidden layer that “includes” or “forgets” information from the past.

Intuition: In language modeling, may be useful to remember/forget gender or number of subject so that personal pronouns (“he” vs. “she” vs. “they”) can be used appropriately.

Useful for things like matching parentheses, etc.

A simpler alternative to the LSTM circuit is called the
Gated Recurrent Unit (GRU)

Vanilla RNNs

In principle the state h_t can carry information from far in the past.

In practice, the gradients vanish (or explode) so this doesn't really happen

We need other mechanisms to “remember” information from far away and use it to predict future words

Vanilla RNNs

State is updated according to

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

We'll modify this with two types of "neural circuits" for storing information to be used downstream

LSTMs and GRUs

Both LSTMs and GRUs have longer-range dependencies than vanilla RNNs.

We'll go through this in detail for GRUs, which are simpler, more efficient, and more commonly used.

(And this is what you'll use on Assignment 5!)

Hadamard product

We'll need to use pointwise products. This is given the fancy name "Hadamard product" and written \odot :

$$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \odot \begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix} = \begin{pmatrix} 4 \\ 10 \\ 18 \end{pmatrix}$$

Hadamard product

We'll need to use pointwise products. This is given the fancy name "Hadamard product" and written \odot :

$$\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} \odot \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ 0 \\ 4 \end{pmatrix}$$

Gated recurrent units (GRUs)



High level idea:

- Learn when to update hidden state to “remember” important pieces of information
- Keep them in memory until they are used
- Reset or “forget” this information when no longer useful

GRUs

GRUs make use of “gates” denoted by Γ (Greek G for “Gate”)

$\Gamma = 1$: “the gate is open” and information flows through

$\Gamma = 0$: “the gate is closed” and information is blocked

GRUs

Two types of gates are used:

Γ^u : When open, information from long-term memory is propagated.
When closed, information from local state is used.

Γ' : When closed, the local state is reset. When open, the state is updated as in a “vanilla” RNN.

Note: These are usually called the “update” and “reset” gates. I find this terminology super confusing. More suggestive names might be the long distance memory gate (Γ^u) and the short distance memory gate (Γ').

GRUs

The state evolves according to

$$c_t = \tanh(W_{xh}x_t + \Gamma_t^r \odot W_{hh}h_{t-1} + b_h)$$

$$h_t = (1 - \Gamma_t^u) \odot c_t + \Gamma_t^u \odot h_{t-1}$$

GRUs

The state evolves according to

$$c_t = \tanh(W_{xh}x_t + \Gamma_t^r \odot W_{hh}h_{t-1} + b_h)$$

$$h_t = (1 - \Gamma_t^u) \odot c_t + \Gamma_t^u \odot h_{t-1}$$

- c_t is the “candidate state” computed using the usual “vanilla RNN” state, after possibly resetting some components.
- When the long-term memory gate is open ($\Gamma^u = 1$), the information gets sent through directly. *This deals with vanishing gradients.*
- The gates are multi-dimensional, applied componentwise
- Prediction of the next word is made using h_t .

GRUs

Everything needs to be differentiable, so the gate is actually “soft” and between zero and one.

The gates are computed as

$$\Gamma_t^u = \sigma(W_{ux}x_t + W_{uh}h_{t-1} + b_u)$$

$$\Gamma_t^r = \sigma(W_{rx}x_t + W_{rh}h_{t-1} + b_r)$$

where σ is the sigmoid function.

Putting it together

GRU state update equations

$$\Gamma_t^u = \sigma(W_{ux}x_t + W_{uh}h_{t-1} + b_u)$$

$$\Gamma_t^r = \sigma(W_{rx}x_t + W_{rh}h_{t-1} + b_r)$$

$$c_t = \tanh(W_{xh}x_t + \Gamma_t^r \odot W_{hh}h_{t-1} + b_h)$$

$$h_t = (1 - \Gamma_t^u) \odot c_t + \Gamma_t^u \odot h_{t-1}$$

There are minor variants on this architecture that are sometimes used.

Putting it together

The reset gate is sometimes moved inside the linear map:

GRU state update equations

$$\Gamma_t^u = \sigma(W_{ux}x_t + W_{uh}h_{t-1} + b_u)$$

$$\Gamma_t^r = \sigma(W_{rx}x_t + W_{rh}h_{t-1} + b_r)$$

$$c_t = \tanh(W_{xh}x_t + W_{hh}(\Gamma_t^r \odot h_{t-1}) + b_h)$$

$$h_t = (1 - \Gamma_t^u) \odot c_t + \Gamma_t^u \odot h_{t-1}$$

There are minor variants on this architecture that are sometimes used.

GRUs: Example

Example:

The leaves, as the weather turned cold in New Haven, fell silently from the trees in November.

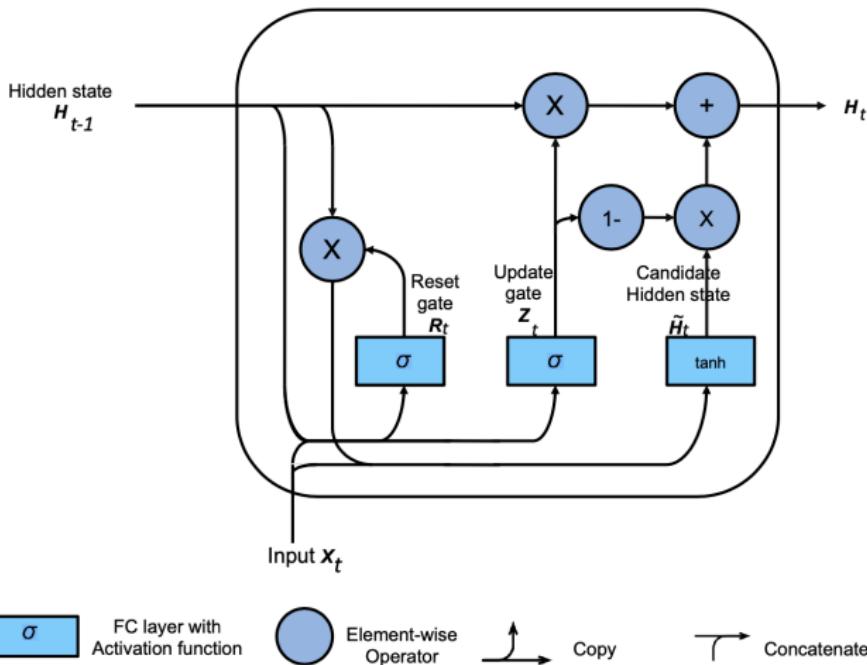
We want to keep `leaves` in memory. It's the subject of the sentence, and plural — syntax

It also has a “foliage” meaning that is relevant when we predict the words `trees` and `November` — semantics

Let's step through on the blackboard how the GRU handles this.

GRU Diagram

(using slightly different notation)



LSTMs

A variant called “Long Short-Term Memory” RNNs has a special context/hidden layer that “includes” or “forgets” information from the past.

$$\text{forget: } F_t = \sigma(W_{fh}h_{t-1} + W_{fx}x_t + b_f)$$

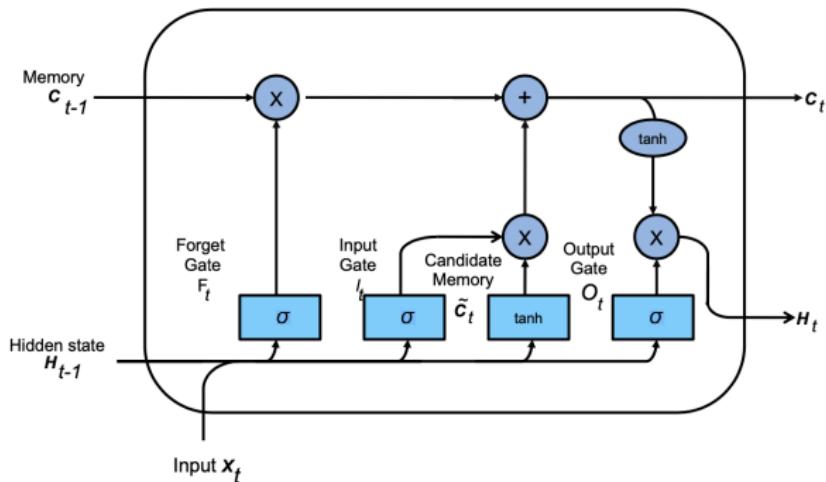
$$\text{include: } I_t = \sigma(W_{ih}h_{t-1} + W_{ix}x_t + b_i)$$

“Memory cell” or “context” c_t evolves according to

$$c_t = F_t \odot c_{t-1} + I_t \odot \tilde{c}_t$$

$$\tilde{c}_t = \tanh(W_{ch}h_{t-1} + W_{cx}x_t + b_c)$$

LSTM Diagram (using slightly different notation)



FC layer with
Activation function



Element-wise
Operator

↑
Copy

→ Concatenate

Summary

- Recurrent neural networks are used for sequential data
- LSTMs and GRUs model longer range dependencies through special gates that control what moves in and out of memory
- RNNs are unsupervised, generative models, able to generate realistic looking sequences when sufficiently well trained.