

Reminders

- Assignment 4 is out, due Wednesday
 - ▶ Graph neural nets
 - ▶ Policy gradient
 - ▶ Deep Q-Learning
- Assignment 5 (last!) posted Wednesday

Quiz 4

Quiz Summary

Section Filter ▾

Student Analysis

Item Analysis

Average Score

92%

High Score

100%

Low Score

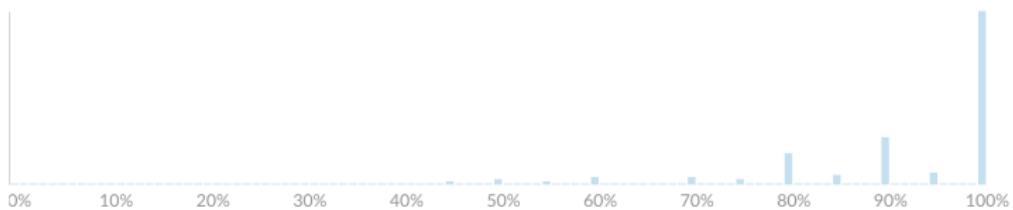
45%

Standard Deviation

1.2

Average Time

25:09



For Today

- Actor-critic: quick recap
- Sequence models
 - ▶ LMs, HMMs, KFs, RNNs
 - ▶ RNN examples

Recap: Actor-critic approaches

- Estimate policy and value function together
- Actor: policy used to select actions
- Critic: value function used to criticize actor
- Error signal from the critic drives all learning
- An on-policy approach



Actor-critic approaches

- After each selected action, critic evaluates new state:
 - ▶ Have things gone better or worse than expected?
- The error signal is used to update actor and value function

Actor-critic approaches

Error signal is

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

$\delta_t > 0$: action was better than expected

$\delta_t < 0$: action was worse than expected

Actor-critic approaches

Error signal is

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

Value function is updated as

$$V(s_t) \leftarrow V(s_t) + \alpha \delta_t$$

Actor-critic approaches

Error signal is

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

Used to update parameters of policy.

If δ_t is positive (negative), action a_t should become more (less) probable in state s_t

For example, with

$$\pi_\theta(a|s) = \text{Softmax}\left\{f_\theta(s, a_1), \dots, f_\theta(s, a_D)\right\}$$

parameters θ adjusted so $f_\theta(s_t, a_t)$ increases (decreases)

Actor-critic approaches

Error signal is

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

Used to update parameters of policy.

For example, with

$$\pi_\theta(a|s) = \text{Softmax}\left\{f_\theta(s, a_1), \dots, f_\theta(s, a_D)\right\}$$

parameters θ adjusted so $f_\theta(s_t, a_t)$ increases (decreases)

$$\theta \leftarrow \theta + \eta \delta_t \nabla_\theta \log \pi_\theta(a_t | s_t)$$

Before we leave RL...

Recent DeepMind work has used actor-critic RL in an attempt to discover faster matrix multiplication algorithms

Article

Discovering faster matrix multiplication algorithms with reinforcement learning

<https://doi.org/10.1038/s41586-022-05172-4>

Received: 2 October 2021

Accepted: 2 August 2022

Published online: 5 October 2022

Open access



Alhussein Fawzi^{1,2*}, Matej Balog^{1,3}, Aja Huang^{1,2}, Thomas Hubert^{1,2}, Bernardino Romera-Paredes^{1,3}, Mohammadamin Barekatain¹, Alexander Novikov¹, Francisco J. R. Ruiz¹, Julian Schrittweise¹, Grzegorz Swirszcz², David Silver¹, Demis Hassabis¹ & Pushmeet Kohli¹

Improving the efficiency of algorithms for fundamental computations can have a widespread impact, as it can affect the overall speed of a large amount of computations. Matrix multiplication is one such primitive task, occurring in many systems—from neural networks to scientific computing routines. The automatic discovery of algorithms using machine learning offers the prospect of reaching beyond human intuition and outperforming the current best human-designed algorithms. However, automating the algorithm discovery procedure is intricate, as the space of possible algorithms is enormous. Here we report a deep reinforcement learning approach based on AlphaZero¹ for discovering efficient and provably correct algorithms for the multiplication of arbitrary matrices. Our agent, AlphaTensor, is trained to play a single-player game where the objective is finding tensor decompositions within a finite factor space. AlphaTensor discovered algorithms that outperform the state-of-the-art complexity for many matrix sizes. Particularly relevant is the case of 4×4

Using on-policy reinforcement learning

- The *state* of the game is a tensor, initialized to $S_0 = T_n$, the tensor for multiplication of $n \times n$ matrices
- An *action* is a choice of a rank-1 tensor $u \otimes v \otimes w$
- The state is updated as

$$S_t \leftarrow S_{t-1} - u \otimes v \otimes w$$

with reward -1

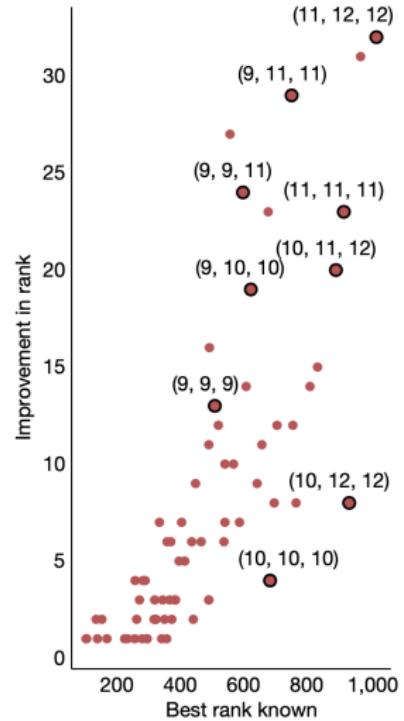
- Episode ends when $S_r \equiv 0$, yielding rank r decomposition

$$T_n = \sum_{t=1}^r u_t \otimes v_t \otimes w_t$$

Using RL

- Entries of tensors chosen from a finite set $\{0, \pm 1, \pm 2\}$
- State and action space is huge: $5^{3n^2} \approx 10^{34}$ for $n = 4$
- Sample from policy using a Monte Carlo scheme

Size (n, m, p)	Best method known	Best rank known	AlphaTensor rank Modular	AlphaTensor rank Standard
(2, 2, 2)	(Strassen, 1969) ²	7	7	7
(3, 3, 3)	(Laderman, 1976) ¹⁵	23	23	23
(4, 4, 4)	(Strassen, 1969) ² $(2, 2, 2) \otimes (2, 2, 2)$	49	47	49
(5, 5, 5)	$(3, 5, 5) + (2, 5, 5)$	98	96	98
(2, 2, 3)	$(2, 2, 2) + (2, 2, 1)$	11	11	11
(2, 2, 4)	$(2, 2, 2) + (2, 2, 2)$	14	14	14
(2, 2, 5)	$(2, 2, 2) + (2, 2, 3)$	18	18	18
(2, 3, 3)	(Hopcroft and Kerr, 1971) ¹⁶	15	15	15
(2, 3, 4)	(Hopcroft and Kerr, 1971) ¹⁶	20	20	20
(2, 3, 5)	(Hopcroft and Kerr, 1971) ¹⁶	25	25	25
(2, 4, 4)	(Hopcroft and Kerr, 1971) ¹⁶	26	26	26
(2, 4, 5)	(Hopcroft and Kerr, 1971) ¹⁶	33	33	33
(2, 5, 5)	(Hopcroft and Kerr, 1971) ¹⁶	40	40	40
(3, 3, 4)	(Smirnov, 2013) ¹⁸	29	29	29
(3, 3, 5)	(Smirnov, 2013) ¹⁸	36	36	36
(3, 4, 4)	(Smirnov, 2013) ¹⁸	38	38	38
(3, 4, 5)	(Smirnov, 2013) ¹⁸	48	47	47
(3, 5, 5)	(Sedoglavic and Smirnov, 2021) ¹⁹	58	58	58
(4, 4, 5)	$(4, 4, 2) + (4, 4, 3)$	64	63	63
(4, 5, 5)	$(2, 5, 5) \otimes (2, 1, 1)$	80	76	76



Summary of actor-critic RL

- Estimate policy and value function together
- Actor: policy used to select actions
- Critic: value function used to criticize actor
- Error signal from the critic drives learning
- Connections to neuroscience of behavior and reward

Language models

- A language model is a way of assigning a probability to any sequence of words (or string of text)

$$p(w_1, \dots, w_n)$$

Language models

- A language model is a way of assigning a probability to any sequence of words (or string of text)

$$p(w_1, \dots, w_n)$$

- By the basic rules of conditional probability we can factor this as

$$p(w_1, \dots, w_n) = p(w_1)p(w_2 | w_1) \dots p(w_n | w_1, \dots, w_{n-1})$$

Language models

- A language model is a way of *generating* any sequence of words

$$P(\text{"the whole forest had been anesthetized"}) =$$

$$\begin{aligned} & P(\text{"the"}) \times P(\text{"whole"} | \text{"the"}) \\ & \quad \times P(\text{"forest"} | \text{"the whole"}) \\ & \quad \times P(\text{"had"} | \text{"the whole forest"}) \\ & \quad \times P(\text{"been"} | \text{"the whole forest had"}) \\ & \quad \times P(\text{"anesthetized"} | \text{"the whole forest had been"}) \end{aligned}$$

Remixing Noon

Text generated from Channel Skin by Jeff Noon

"The whole forest had been anesthetised, her temples wired, her senses stimulated, her eyes were not on Eva, not on Eva, not on Eva, not on anybody in that same realm, the land of dreams and nightmares."

Viability: 0.000000326%

https://revdancatt.com/2017/03/01/markov_noon

Text generation

- Words generated one-by-one
- A word is chosen by sampling from a probability distribution
- Condition on previously generated text, as if “real”
- Result is purely synthetic text

Modern language models

Suppose a computer program assigns a “score” to possible next words:

$$f(v; \underbrace{s(w_1, \dots, w_n)}_{\text{state of word history}})$$

Modern language models

Suppose a computer program assigns a “score” to possible next words:

$$f(v; \underbrace{s(w_1, \dots, w_n)}_{\text{state of word history}})$$

Can convert this to a language model by the “softmax” operation:

$$p(w | w_1, \dots, w_n) = \frac{\exp(f(w; s(w_1, \dots, w_n)))}{\sum_{v \in V} \exp(f(v; s(w_1, \dots, w_n)))}$$

First things first

- In ChatGPT, the function $f(v; s(w_{1:n}))$ is learned on large amounts of text to predict the next word (unsupervised) using a type of deep neural network called a *transformer*
- We'll first discuss "classical" sequence models — HMMs, KFs, RNNs

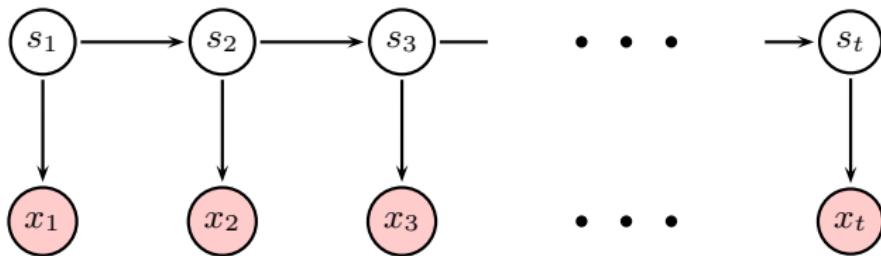
Hidden Markov Model

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 Model

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 Model

Probability of word sequence:

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

For topic models:

$$p(x_1, \dots, x_n) = \int p(\theta | \alpha) \sum_{s_1, \dots, s_n} \prod_{t=1}^n p(s_t | \theta) p(x_t | s_t) d\theta$$

Word order doesn't matter

Hidden Markov Model

Probability of word sequence:

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

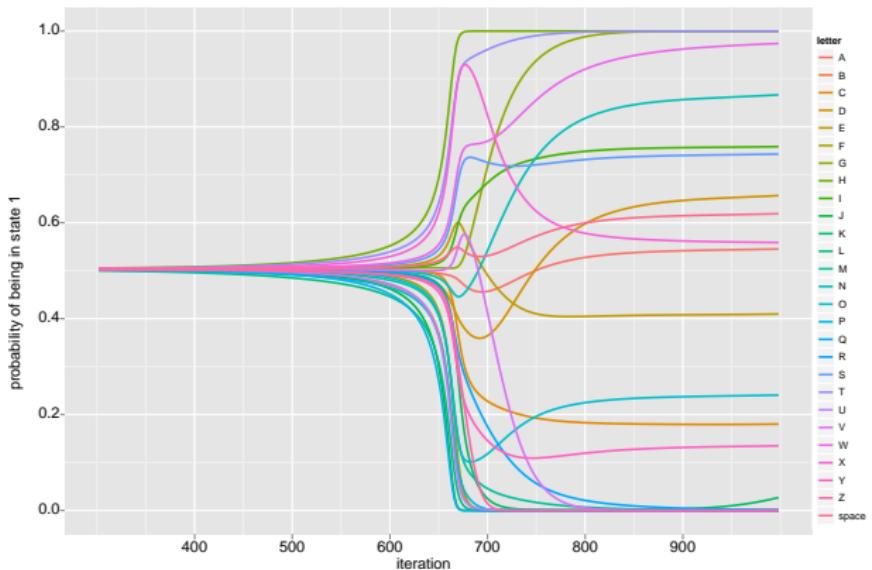
Can be efficiently computed using the “forward-backward algorithm” which is a type of dynamic program

Hidden Markov Models: Estimation

The usual algorithms for estimating the parameters are iterative:

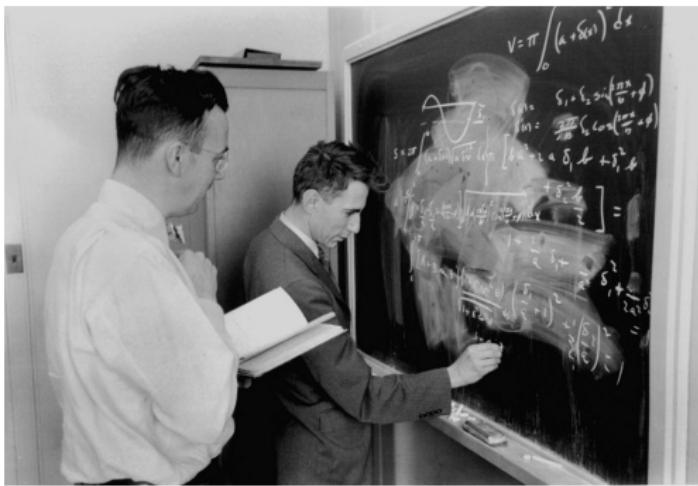
- Use the forward-backward algorithm to compute the probability that the outputs are generated from each state
- Treat the data as labeled, with these weights, and compute the relative frequencies
- Repeat
- Maximum likelihood (or MAP) estimation

Hidden Markov Models: Example



HMM has two states, emits one of the twenty-six letters A-Z. Trained on 20,000 characters of English text, initial parameters close to uniform. Curves show the posterior probability $\mathbb{P}(\text{state} = 1 \mid \text{letter}, \mathcal{D})$ of the probability each letter is generated from state one. “phase transition” at around 650 iterations.

Claude Shannon



Model underlying Kalman filters

A Gaussian HMM:

$$S_t | S_{t-1} \sim N(AS_{t-1}, \Gamma)$$

$$X_t | S_t \sim N(BS_t, \Sigma)$$

where A and B are matrices for the means; Γ and Σ for the covariances. Everything is linear.

- State is distributed (real vector)
- State evolves stochastically
- Schur complements and forward-backward used to compute conditional probabilities; similar to Gaussian processes

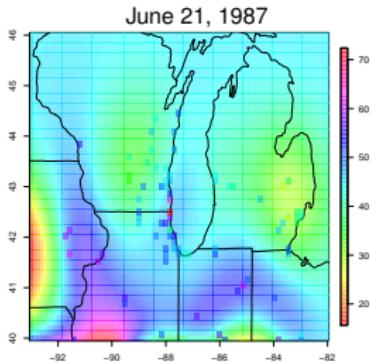
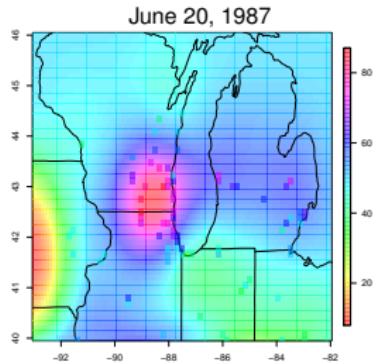
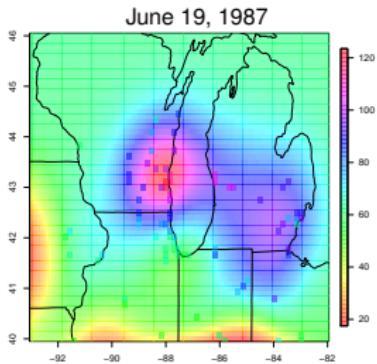
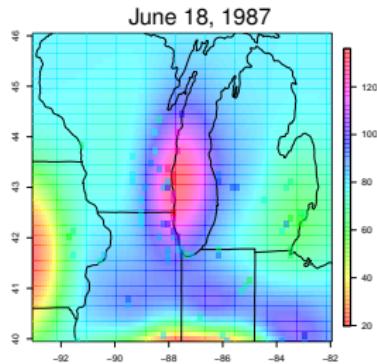
Example: Ozone

Data are daily maximum 8-hour ozone concentrations (in parts-per-billion) at 153 sites in the US midwest near Lake Michigan, for 89 days during the summer of 1987.

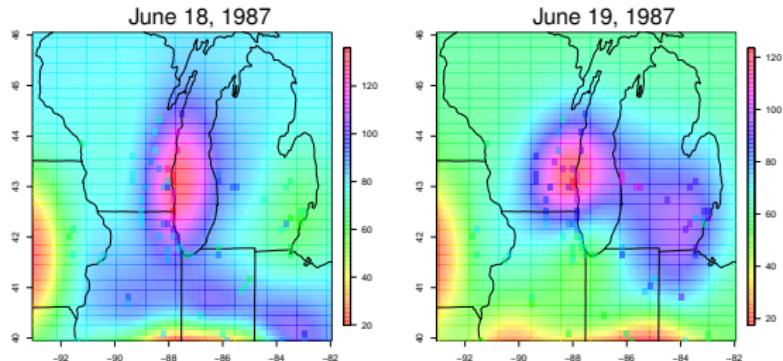
- `lon.lat`: longitudes and latitudes of the 153 stations;
- `y`: measurements at each stations on each day;
- `station.id` and dates.

Can use Kalman filtering/smoothing to infer ozone concentrations of the area

Kalman filtering: Example

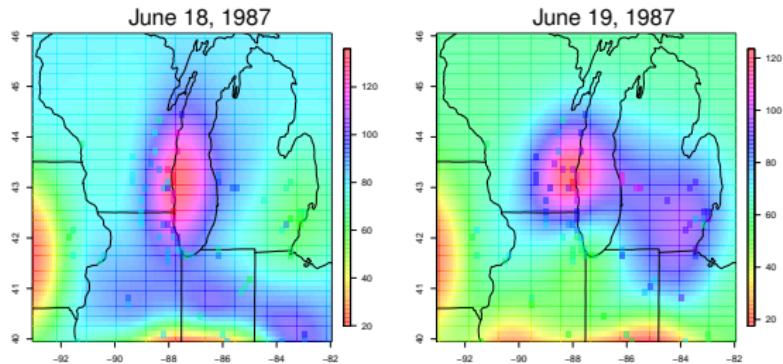


Kalman filtering: Example



- In this example, $A = I$ (just add noise)
- B_j = “average over locations near j ”.

Kalman filtering: Example



- The Kalman filter is a generative model
- You can “synthesize” ozone levels on a given day.
- Could use kernel smoothing (a discriminative model) or a Gaussian process

Kalman filtering: Discrete data

We can also work with state space models for discrete data, like documents.

$$S_t | S_{t-1} \sim N(AS_{t-1}, \Gamma)$$

$$W_t | S_t \sim \text{Softmax}(BS_t)$$

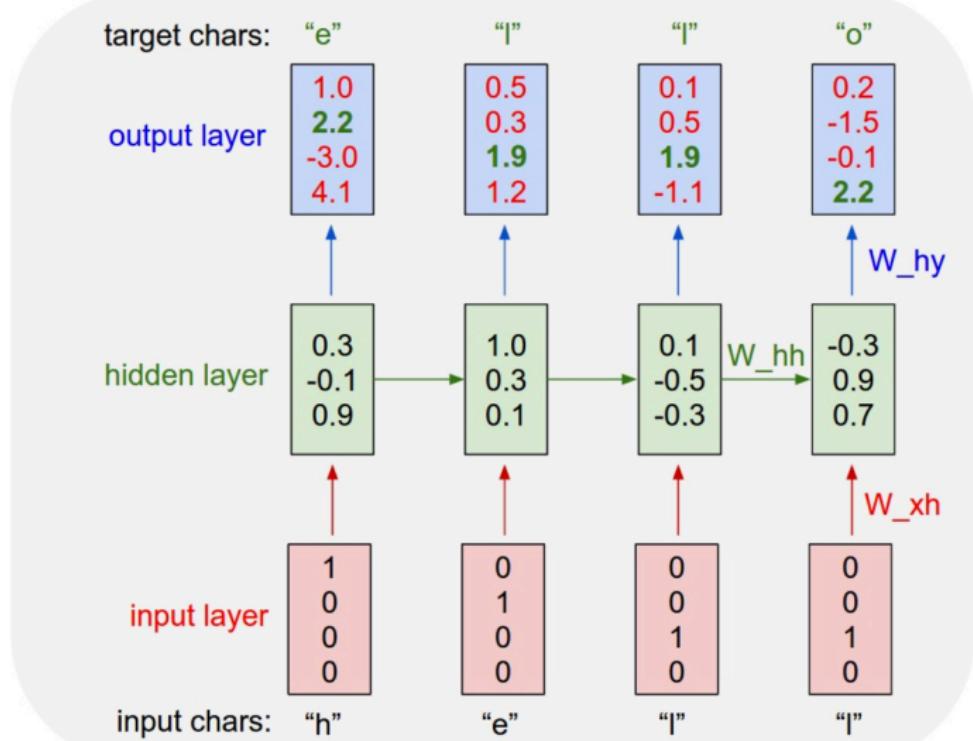
where

$$\text{Softmax}(BS_t)_j = \frac{\exp(B_j^T S_t)}{\sum_k \exp(B_k^T S_t)}$$

This is what's done with dynamic topic models, which use a Kalman filter as a variational approximation.
https://en.wikipedia.org/wiki/Dynamic_topic_model

Next: Recurrent neural networks

- Can be thought of as a type of language model
- Similar to hidden Markov models and Kalman filters, but the hidden layer is not stochastic
- The state is distributed (a vector), as in Kalman filters, not categorical as for HMMs



RNNs

This means

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

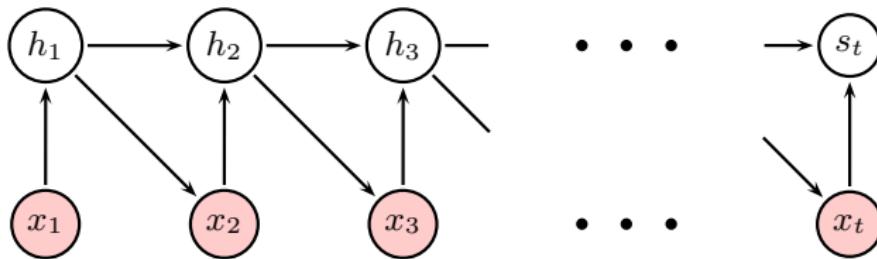
$$y_t = W_{hy}h_t$$

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

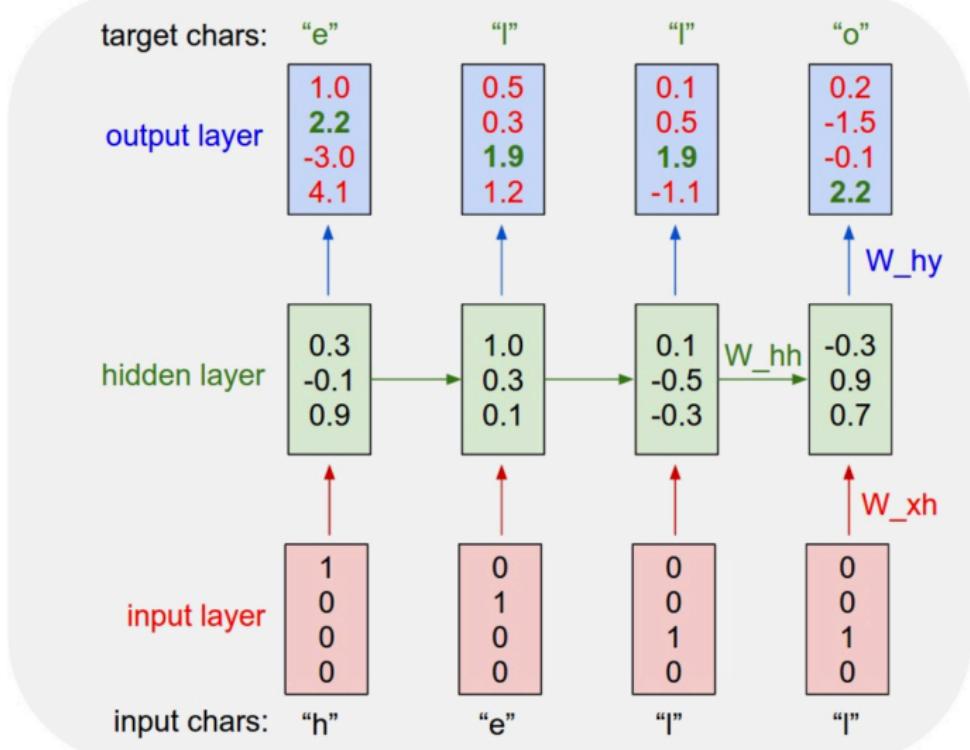
where $\pi(\cdot)$ is the soft-max function.

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



RNN architecture



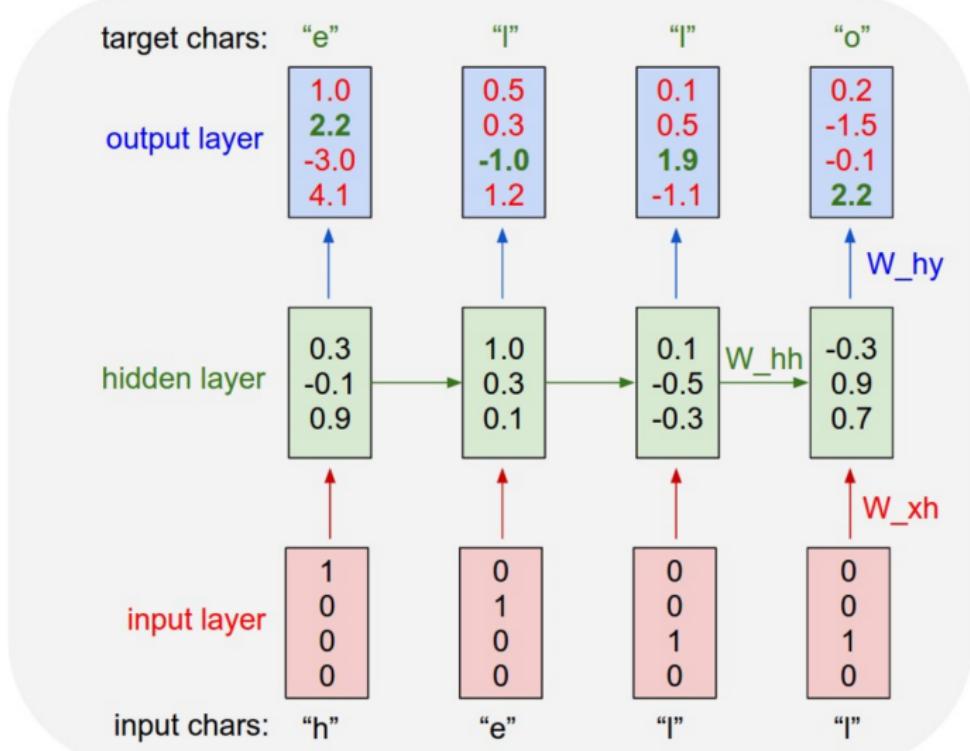
RNN Python Code

```
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)
```

RNN examples

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

LSTMs

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)

Synthetic 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.

Topics in scientific texts

Quantum physics	spin energy field electron magnetic state states hamiltonian
Particle physics	higgs neutrino coupling decay scale masses mixing quark
Astrophysics	mass gas star stellar galaxies disk halo radius luminosity
Relativity	black metric hole schwarzschild gravity holes einstein
Number theory	prime integer numbers conjecture integers degree modulo
Graph theory	graph vertex vertices edges node edge number set tree
Linear algebra	matrix matrices vector basis vectors diagonal rank linear
Optimization	problem optimization algorithm function solution gradient
Probability	random probability distribution process measure time
Machine learning	layer word image feature sentence model cnn lstm training

Latent topic vector θ included as additional state in an RNN for generating mathematical equations.

Topic modeling for equations

Topic	Generated Equations
Quantum physics	<ul style="list-style-type: none">$E = \hbar \frac{\partial^2 S}{\partial t^2} \left(\frac{\partial \varphi}{\partial c} \right) - \frac{k}{\hbar^2} \frac{\partial B}{\partial t} (t + \partial_t \delta).$$\Psi_{\text{pr}} = \sum_{\mathbf{l}} (\psi_{\mathbf{r}+\uparrow} - \psi_{\mathbf{r}\downarrow}^\dagger) + \sum_{\mathbf{r}'} (\psi_{\mathbf{r}',\uparrow}^\dagger - \psi_{\mathbf{r}'\downarrow} \sigma^\dagger).$
Particle physics	<ul style="list-style-type: none">$\mathcal{H} = \frac{1}{4} (\partial_\mu \phi)^2 + 2m\phi_\nu(\phi) + \frac{1}{2}m^2(\phi)(1-\phi^2)^2.$$m_{\text{eff}}(M) = 1.4 \cdot 10^{-13} \text{ GeV}.$
Relativity	<ul style="list-style-type: none">$\mathcal{M} = \frac{1}{2} g^{\mu\nu} (f_{\mu\nu,\mu} - g_{\mu\nu,\nu} + g_{\nu\nu,b} f_{\mu,\nu}) + \frac{1}{2} g^{\mu\nu}.$$T_{\mu\nu} = \int_0^\infty ds_{\mu\nu} ds^2 + a_\mu^2 dr^2 + r^2 d\Omega^2.$

LSTM with two hidden layers, 512 neurons in each layer, trained on 40,000 scientific articles, using \LaTeX source. The generated \LaTeX compiles about 80% of the time; otherwise needs minor tweaks.

Topic modeling for equations

Number theory	<ul style="list-style-type: none">• $(2^k)^k + (1^n + 1)(1 + p^k) = 1.$
Linear algebra	<ul style="list-style-type: none">• $\text{tr}(E_\varepsilon X^*) = U^\top(\text{tr}(V_\varepsilon X)).$• $\phi_h(\theta, y) = \left\{ X \in \text{Span} \left(P_c(\mathbf{T}[x, \mathbf{x}]) \right) \right\}.$
Optimization	<ul style="list-style-type: none">• $\min_p p(x)$ subject to $\ p^x - y\ _2 \leq m_p.$• $w^+ = w_t + g_t \ u_t - \nabla \mathbf{u}^*\ _2^2.$
Probability	<ul style="list-style-type: none">• $\mathbb{P}(r_\tau < t) = \mathbb{E}_{\tau_{\text{twist}}}(N_\tau).$• $T^*(t) = \lim_{t \rightarrow \infty} \mathbb{E}[N(t) + \mathbb{E}[\varphi_t(x)]^*]$

LSTM with two hidden layers, 512 neurons in each layer, trained on 40,000 scientific articles, using \LaTeX source. The generated \LaTeX compiles about 80% of the time; otherwise needs minor tweaks.

Topic modeling for equations

Context words	Inferred Topics	Generated Equations
star gravity einstein mass galaxies	58% Astrophysics 36% Relativity 3% Quantum physics	<ul style="list-style-type: none">$\left(\frac{m_b}{M_\odot}\right)^{b_\nu} \ll g\sqrt{\frac{\tilde{\Phi}_0}{\eta_{\text{eff}}}}$.$G(r) = \int_{r_0}^{r_0} dr \sqrt{\log(r)(\bar{r} + r_0(w))}$.
data training likelihood model gradient	62% Machine learning 21% Statistics 15% Optimization	<ul style="list-style-type: none">$L = -\frac{1}{N} \sum_{i=1}^N \mathcal{R}_{R_i}(\mathbf{r}_i) + V^r(\mathbf{r}_i)$.$\operatorname{argmax}_{\mathcal{U}} \mathbb{E}_{W \sim \Psi} \log \exp [\Lambda(\widetilde{W}) - H]$.

LSTM with two hidden layers, 512 neurons in each layer, trained on 40,000 scientific articles, using \LaTeX source. The generated \LaTeX compiles about 80% of the time; otherwise needs minor tweaks.

Fake ν s

Which is fake?

$$L = \int_0^{e^{T-1}} \psi \sin^2 \theta d\lambda \sin^2 \theta d\theta^2 - C^{2/\sigma}$$

$$L = \int_0^L dz \|x_z\| = \int_0^L dz \sqrt{x_z \cdot x_z} \equiv \int_0^L dz \sqrt{g}$$

Fake ν s

$$L = \int_0^{e^{T-1}} \psi \sin^2 \theta d\lambda \sin^2 \theta d\theta^2 - C^{2/\sigma}$$

$$L = \int_0^L dz \|x_z\| = \int_0^L dz \sqrt{x_z \cdot x_z} \equiv \int_0^L dz \sqrt{g}$$

Fake vs

Which is fake?

$$(xy^{-1})_{N(y)} = s^\epsilon(sxy^{-1})_{N(y)} = s^\epsilon(sx)_\emptyset y_\emptyset^{-1}$$

$$(xy^2 - x^2)(y(x) - Ty(x)) \leq \int_0^T e^{-x't(w)} dt$$

Fake vs

$$(xy^{-1})_{N(y)} = s^\epsilon(sxy^{-1})_{N(y)} = s^\epsilon(sx)_\emptyset y_\emptyset^{-1}$$

$$(xy^2 - x^2)(y(x) - Ty(x)) \leq \int_0^T e^{-x't(w)} dt$$

Fake ν s with topics

“optimization”

$$X_L = -\frac{1}{\sum_i y_i} \pi(y_i, x_i, x > 0) \text{ subject to } x_i^T X^H x_i \leq \epsilon^2$$

$$\text{minimize} \quad - \sum_{i=1}^K \sum_{c=1}^Z a_{mij} \|g_i\|_2^2$$

Fake ν s with topics

“physics”

$$\hat{b}_{\text{prc}} = \frac{\hbar^2}{R} - \mu_{\text{sit}} F_d$$

$$\frac{\partial \delta C}{\partial r} Re_a a_\mu - \int_{V_{bor,quiet}} \langle K^{\alpha\beta} \rangle_\alpha^\beta = \mathcal{H} Er + \beta_{\alpha\mu\nu}^\top R_{\text{conseds}}^{\mu\lambda} \partial_\mu$$

$$\rho_{deg} \propto 11eV [L(003B)/20 \pm 10ifs.1.8V(10^{-2}ergs^{-1})/5; 1.13neV]$$

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
        dhy = np.dot(Why.T, dy) + dhnext # backprop into h
        ddraw = (1 - hs[t] * hs[t]) * dhy # 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.

Summary

- HMMs and Kalman Filters are classical models for sequences
- HMMs have discrete states, KFs have distributed states and Gaussian transitions/emissions
- Recurrent neural networks are also used for sequential data, like language
- As usual, linear mappings followed by activation functions.
- RNNs are unsupervised, generative models, able to generate realistic looking sequences when sufficiently well trained.
- LSTMs and GRUs model longer range dependencies (next time)