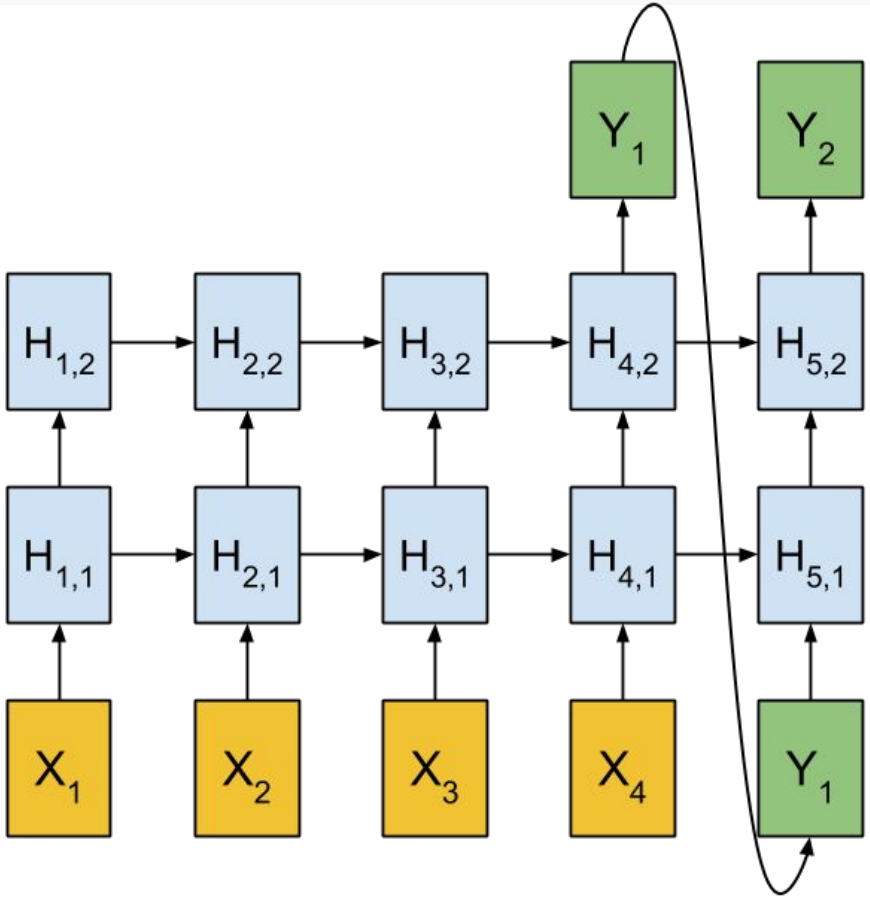
An Empirical Exploration of Recurrent Network Architectures

Rafal Jozefowicz, Wojciech Zaremba, Ilya Sutskever

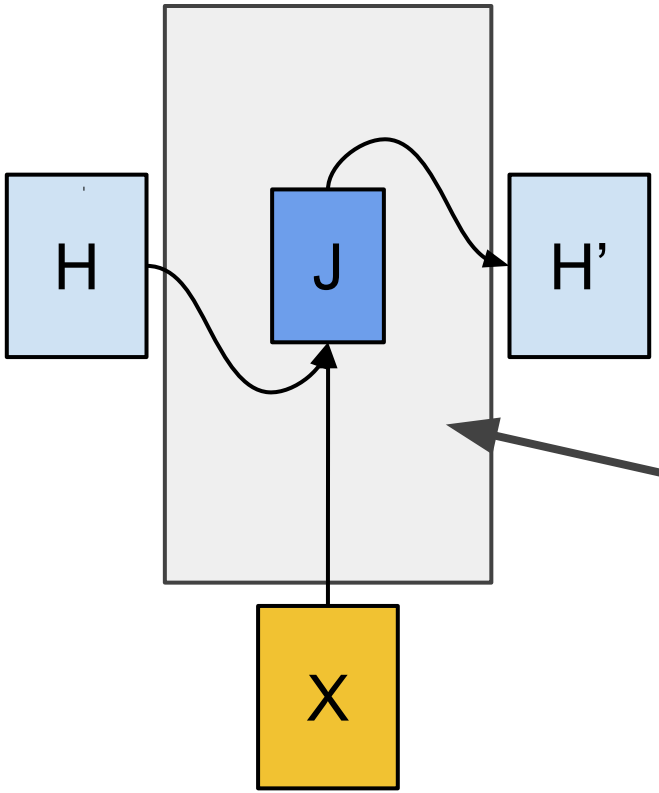
Recurrent Neural Networks

- Recurrent Neural Networks (RNNs) are very capable models for sequence modelling

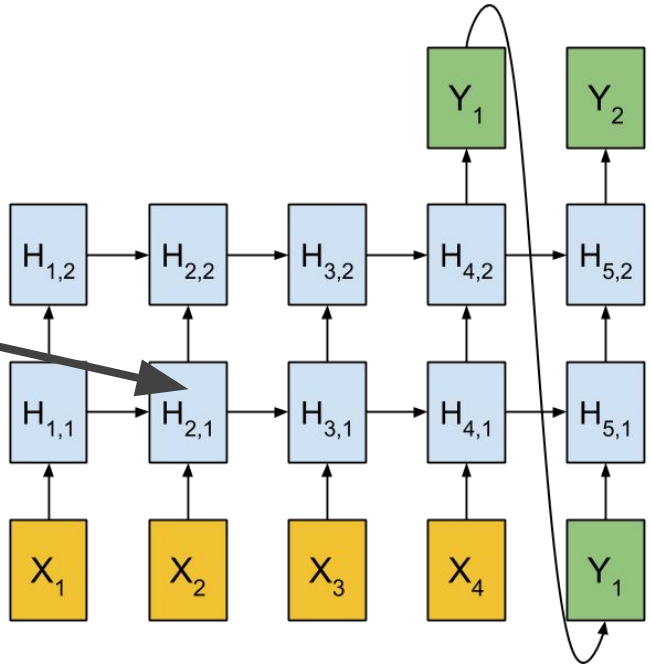
Recurrent Neural Networks



Standard RNN



$$H' = Tanh(W_X X + W_H H + b)$$



Exploding gradients problem

It is relatively easy to handle by simply shrinking gradients whose norms exceed a threshold (Gradient Clipping)

- works well as long as gradient has small norm for majority of time

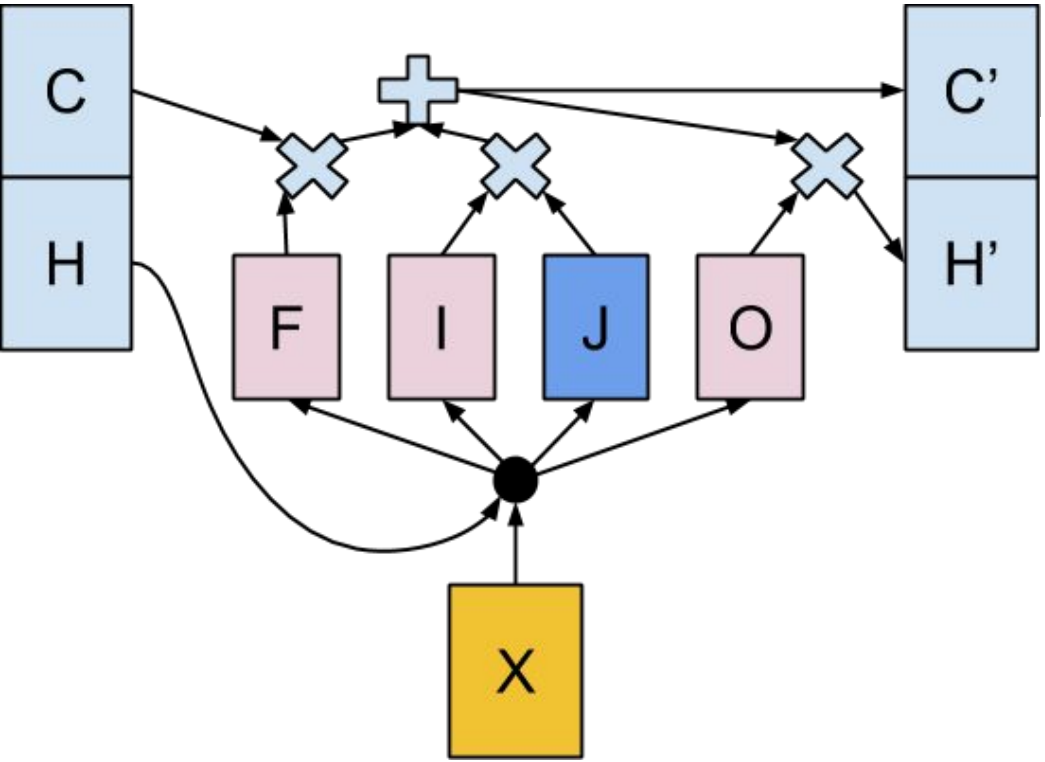
Vanishing gradients problem

- RNNs can easily learn short-term but not long-term dependencies
- The LSTM addresses this problem by reparameterization
  - Instead of directly computing new state (S_t) from the previous one (S_{t-1}), the LSTM computes the difference between them and adds to the previous value

    $S_t = S_{t-1} + \Delta S_t$ and $S_T = \Sigma_i \Delta S_i$

  - This way the gradients of the long-term dependencies cannot vanish

LSTM - Long Short-Term Memory



$$F = Sigmoid(W_{F_1}X + W_{F_2}H + b_F)$$

$$I = Sigmoid(W_{I_1}X + W_{I_2}H + b_I)$$

$$J = Tanh(W_{J_1}X + W_{J_2}H + b_J)$$

$$O = Sigmoid(W_{O_1}X + W_{O_2}H + b_O)$$

$$C' = C \times F + I \times J$$

$$H' = Tanh(C') \times O$$

LSTM(X, C, H) = [C', H']

$F = Lin_{FX}(X) + Lin_{FH}(H)$

$I = Lin_{IX}(X) + Lin_{IH}(H)$

$J = Lin_{JX}(X) + Lin_{JH}(H)$

$O = Lin_{OX}(X) + Lin_{OH}(H)$

where:

$Lin_i(T) = Dot(W_i, T) + b_i$

Forget Gate

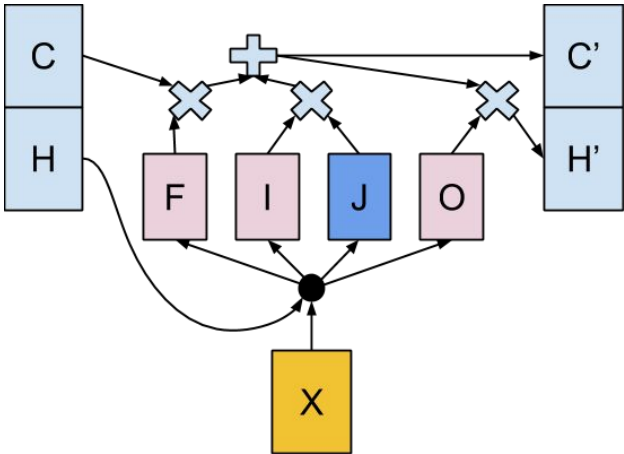
Input Gate

New Input

$C' = C\ Sigmoid(F) + Sigmoid(I)Tanh(J)$

$H' = Tanh(C')\ Sigmoid(O)$

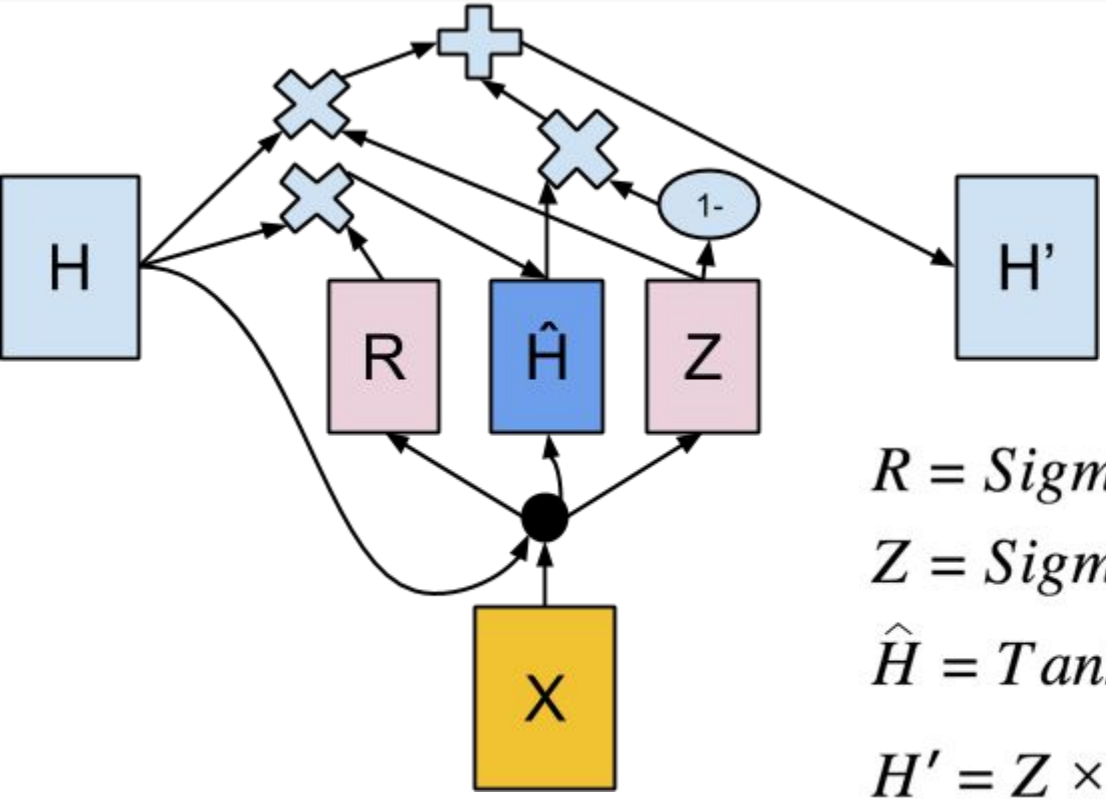
Output Gate



The problem

- The LSTM is complicated and is ad-hoc
- Does there exist a much better RNN architecture?
  - **Goal:** use exhaustive search to determine if there exists an architecture that is much better than the LSTM
- Are the various LSTM gates important?
  - **Goal:** determine what happens when we remove the various LSTM gates

Gated Recurrent Unit - a recently proposed LSTM alternative



$$R = Sigmoid(W_R X + W_R H + b_R)$$

$$Z = Sigmoid(W_Z X + W_Z H + b_Z)$$

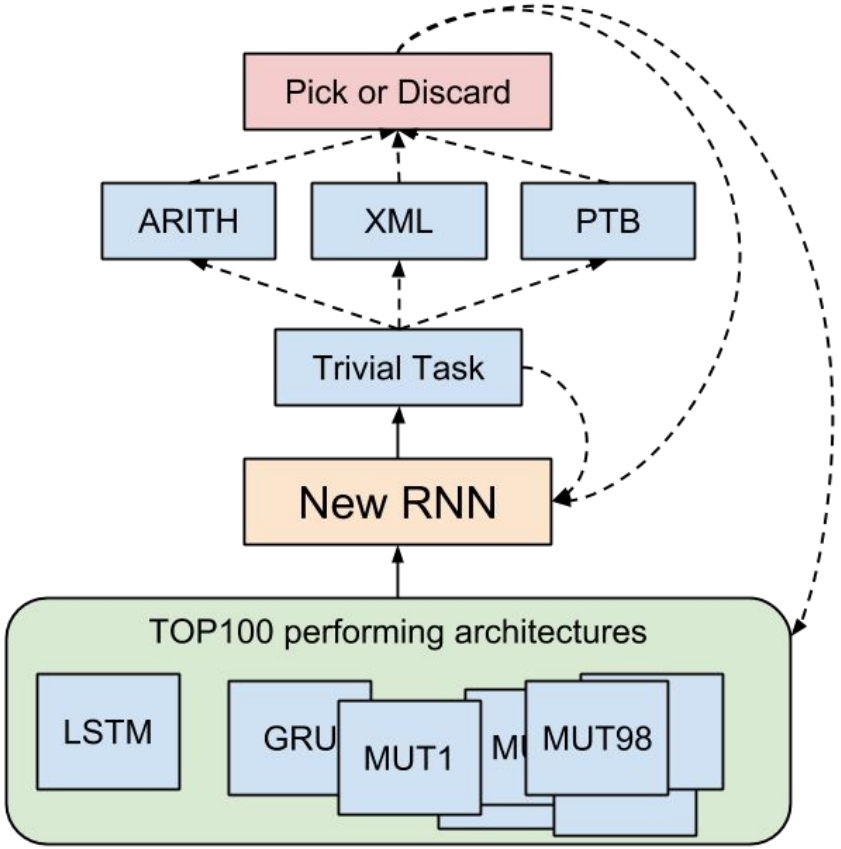
$$\hat{H} = Tanh(W_{\hat{H}} X + W_{\hat{H}}(R \times H) + b_{\hat{H}})$$

$$H' = Z \times H + (1 - Z) \times \hat{H}$$

Experiments

- Our main experiment is an extensive architecture search
  - We evaluated over 10,000 architectures
  - We optimized the hyperparameters of the promising architectures
- We also performed an ablative study of different LSTM variations

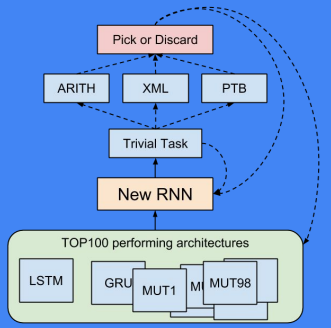
Architecture Search



Repeat the following steps:

- Pool of promising candidates
- Choosing an architecture to test
- Evaluate on a trivial task
- Evaluate on 3 challenging problems
- Update promising candidate pool

Architecture Search - candidate pool

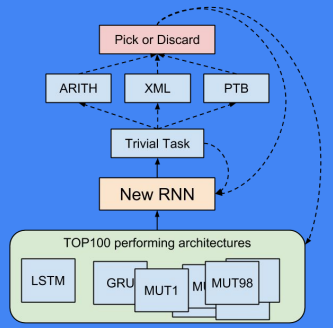


- We maintain a list of 100 best performing architectures so far
- The pool is initialized with only the LSTM and the GRU
- The architectures are ranked with a simple metric:

$$min_{task} \frac{architecture's\ best\ accuracy\ on\ task}{GRU's\ best\ accuracy\ on\ task}$$

- By selecting the minimum over all tasks, we make sure that the search procedure will look for an architecture that works well on every task

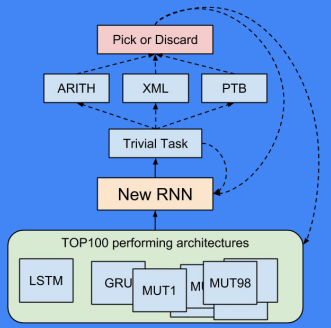
Architecture Search - candidate selection



Either:

- Randomly pick an architecture from the pool, evaluate it on a new hyperparameter settings and update its performance estimate
- Propose a new architecture by choosing one from the pool and mutating it

Architecture Search - Mutation

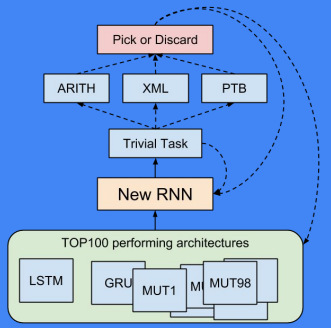


Given a parent architecture, we are allowed to modify it in following ways:

- Insert/Replace/Remove Activation Node:
  Tanh(x), Relu(x), Sigmoid(x), Lin(x), Lin(x)+0.9, Lin(x)+1, Lin(x)+1.1
- Insert/Replace/Remove Elementwise Binary Operator (Add/Mul/Sub)
- Replace Node with one of its ancestors (dependencies) - graph reduction

We pick 1-3 such transformations and apply it to each node with a randomly chosen probability.

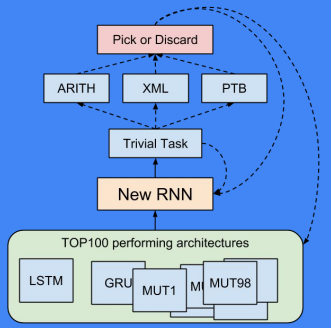
Architecture Search - Trivial Task



Simple memorization problem to filter unpromising architectures

- 5 symbols in sequence are to be read and then reproduced in the same order (alphabet size 26)
- Discard architecture if its accuracy is below 95% with teacher forcing
- Might re-examine the architecture in a future on a different set of hyperparameters if it’s randomly picked again at some point

Architecture Search - Evaluation

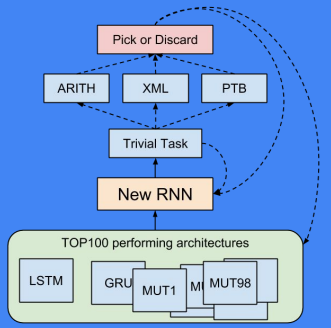


If an architecture passes the first stage, we evaluate it on the first task on a set of 20 new random hyperparameters

- if the parent is heavily evaluated, 80% of hyperparameters comes from best 100 parameters of the parent and the rest is uniform
- otherwise 33% comes from LSTM, 33% from GRU and the rest is uniform

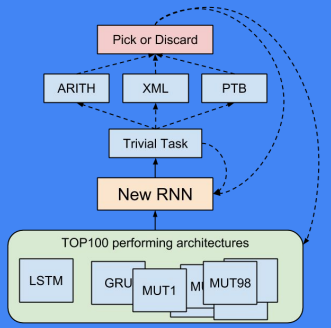
If the results were within 90% of best GRU's results, we'd evaluate the architecture on the second and then on the third task.

Architecture Search - Stats



- Overall, we evaluated more than 10,000 architectures
- 1,000 of them succeeded on the trivial task
- They were evaluated on average 220 times on 3 problems

Hyperparameter ranges



- The **initialization scale** is in {0.3, 0.7, 1, 1.4, 2, 2.8} and the weights are initialized uniformly in U[-x, x], where x = scale / Sqrt(#units in layer))
- The **learning rate** is in {0.1, 0.2, 0.3, 0.5, 1, 2, 5} divided by minibatch size
- The **maximal gradient norm** was set to {1, 2.5, 5, 10, 20}
- The **number of layers** was chosen from {1, 2, 3, 4}
- The **number of parameters** was also a hyperparameter (task-dependent)
    - But we observed only slight performance gains for the larger models

Evaluation Tasks: ARITHMETIC

- Arithmetic addition and subtraction of up to 8 digit numbers
- The input is given one character at the time
- To make the task more difficult we introduced distractor symbols between the successive input characters.

A typical instance:

3e**36**d**9**-h**1**h**39**f**94**eeh**43**keg**3**c**=-13991064.**

which represents

3369-13994433=-13991064.

Evaluation Tasks: XML

- The goal is to predict the next character in the synthetic XML dataset
- To perform this task successfully, the network needs to learn how to stack memory

A short sample:

<etdomp><pegshmnaj><zbhbmg></zbhbmg></pegshmnaj><autmh><autmh></etdomp>

Evaluation Tasks: PTB

- Word-level language modeling task on the Penn TreeBank dataset
- 1M words with a vocabulary of size 10,000
- During the architecture search we monitored accuracy for early stopping

Evaluation Tasks: MUSIC

- Polyphonic music datasets from Boulanger-Lewandowski et al. (2012)
- Each timestep is a binary vector representing the played notes
- It’s different than other tasks as we’re predicting binary vectors

We didn’t use these datasets during the architecture search, but we used them to evaluate the learned models

Additional experiments

- The LSTM has three gates: the input gate *j*, the forget gate *f*, and the output gate *o*
- We removed each of the gates and evaluated the resulting architecture
  - We optimized the hyperparameters for each of the architectures

LSTM(X, C, H) - Forget Gate Bias

Most applications of LSTMs simply initialize weights with small random values

- This effectively sets the forget gate to 0.5 and the vanishing gradient with a factor of 0.5 per timestep

Forget Gate

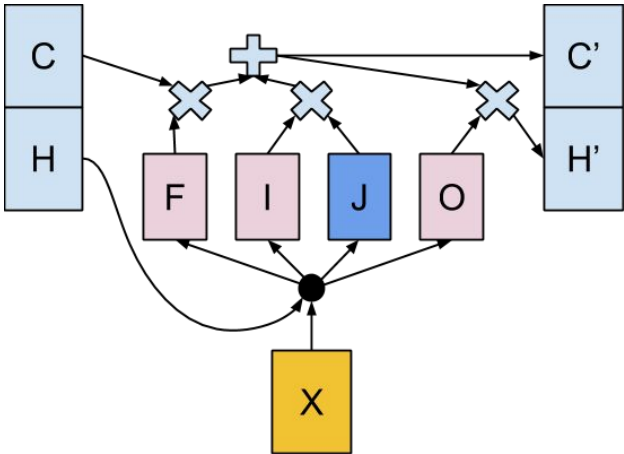
Input Gate

New Input

$$C' = C\,\text{Sigmoid}(F) + \text{Sigmoid}(I)\text{Tanh}(J)$$

$$H' = \text{Tanh}(C')\,\text{Sigmoid}(O)$$

Output Gate



LSTM(X, C, H) - Forget Gate Bias

Most applications of LSTMs simply initialize weights with small random values

- This effectively sets the forget gate to 0.5 and the vanishing gradient with a factor of 0.5 per timestep
- The problem can be addressed by initializing the forget gates biases to a large value like 1 or 2

Forget Gate

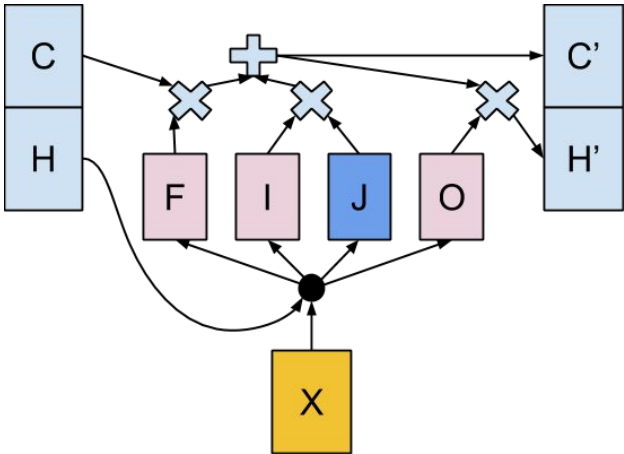
Input Gate

New Input

$$C' = C\,\text{Sigmoid}(F + \mathbf{1}) + \text{Sigmoid}(I)\text{Tanh}(J)$$

$$H' = \text{Tanh}(C')\,\text{Sigmoid}(O)$$

Output Gate



Best Architectures found during the search -- similar to GRU

MUT1

$$
\begin{aligned}
z &= \mathrm{sigm}(W_{\mathrm{xz}} x_t + W_{\mathrm{hz}} h_t + b_{\mathrm{z}}) \\
r &= \mathrm{sigm}(x_t + W_{\mathrm{hr}} h_t + b_{\mathrm{r}}) \\
h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + W_{xh} x_t + b_{\mathrm{h}}) \odot z \\
&+ h_t \odot (1 - z)
\end{aligned}
$$

MUT2

$$
\begin{aligned}
z &= \mathrm{sigm}(W_{\mathrm{xz}} x_t + b_{\mathrm{z}}) \\
r &= \mathrm{sigm}(W_{\mathrm{xr}} x_t + W_{\mathrm{hr}} h_t + b_{\mathrm{r}}) \\
h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + \tanh(x_t) + b_{\mathrm{h}}) \odot z \\
&+ h_t \odot (1 - z)
\end{aligned}
$$

MUT3

$$
\begin{aligned}
z &= \mathrm{sigm}(W_{\mathrm{xz}} x_t + W_{\mathrm{hz}} \tanh(h_t) + b_{\mathrm{z}}) \\
r &= \mathrm{sigm}(W_{\mathrm{xr}} x_t + W_{\mathrm{hr}} h_t + b_{\mathrm{r}}) \\
h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + W_{xh} x_t + b_{\mathrm{h}}) \odot z \\
&+ h_t \odot (1 - z)
\end{aligned}
$$

Architecture Search Results (Accuracy)

Arch.	Arith.	XML	PTB
Tanh	0.29493	0.32050	0.08782
LSTM	0.89228	0.42470	0.08912
LSTM-f	0.29292	0.23356	0.08808
LSTM-i	0.75109	0.41371	0.08662
LSTM-o	0.86747	0.42117	0.08933
LSTM-b	0.90163	0.44434	0.08952
GRU	0.89565	0.45963	0.09069
MUT1	**0.92135**	**0.47483**	0.08968
MUT2	0.89735	**0.47324**	0.09036
MUT3	0.90728	0.46478	**0.09161**

Final evaluation on the MUSIC datasets (NLL)

Arch.	N	N-dropout	P
Tanh	3.612	3.267	6.809
LSTM	3.492	3.403	6.866
LSTM-f	3.732	3.420	6.813
LSTM-i	3.426	**3.252**	6.856
LSTM-o	3.406	**3.253**	6.870
LSTM-b	3.419	3.345	6.820
GRU	3.410	3.427	6.876
MUT1	**3.254**	3.376	**6.792**
MUT2	3.372	3.429	6.852
MUT3	3.337	3.505	6.840

Final evaluation on the PTB dataset (NLL)

Arch.	5M-tst	10M-v	20M-v	20M-tst
Tanh	4.811	4.729	4.635	4.582 (97.7)
LSTM	4.699	4.511	4.437	4.399 (81.4)
LSTM-f	4.785	4.752	4.658	4.606 (100.8)
LSTM-i	4.755	4.558	4.480	4.444 (85.1)
LSTM-o	4.708	4.496	4.447	4.411 (82.3)
LSTM-b	4.698	4.437	4.423	**4.380 (79.83)**
GRU	4.684	4.554	4.559	4.519 (91.7)
MUT1	4.699	4.605	4.594	4.550 (94.6)
MUT2	4.707	4.539	4.538	4.503 (90.2)
MUT3	4.692	4.523	4.530	4.494 (89.47)

Summary

- The GRU outperformed the LSTM on all tasks with the exception of language modeling
- MUT1 matched the GRU's performance on language modeling and outperformed it on all other tasks
- The LSTM significantly outperformed other architectures on PTB when dropout was allowed
- Adding large forget gate bias greatly improves the LSTM performance
- The LSTM's forget gate is the most important one while the output gate is relatively unimportant

Thanks!