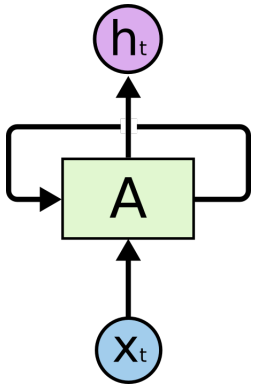
Recurrent Neural networks



Slides inspired by Abhishek Narwekar and Anusri Pampari at UIUC

Lecture Outline

1. Introduction
2. Learning Long Term Dependencies
3. Visualization for RNNs

Applications of RNNs

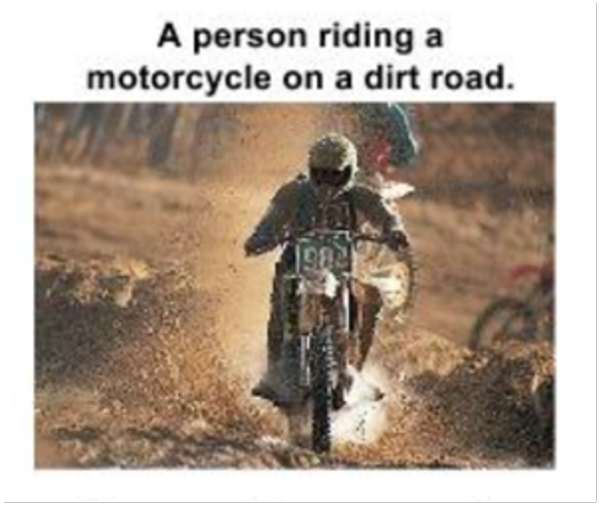
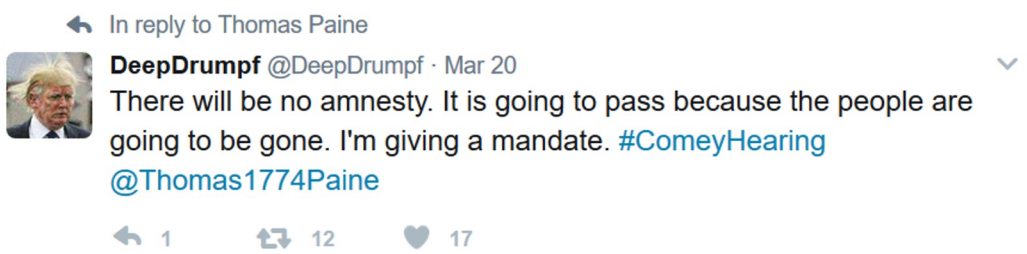


Image Captioning
[reference]
... and more!

VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;

Write like Shakespeare
[reference]



.. and Trump
[reference]

Applications of RNNs

Technically, an RNN models sequences

- Time series
- Natural Language, Speech

We can even convert non-sequences to sequences, eg: feed an image as a sequence of pixels!

Applications of RNNs

- RNN Generated TED Talks
  - YouTube Link
- RNN Generated Eminem rapper
  - RNN Shady
- RNN Generated Music
  - Music Link

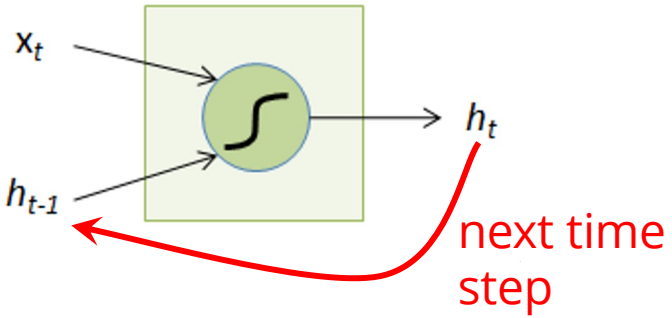
Why RNNs?

- Can model sequences having variable length
- **Efficient**: Weights shared across time-steps
- **They work**!
  - SOTA in several speech, NLP tasks
  - Not any more, now Transformers

    (See for image stuffs, https://openai.com/blog/dall-e/)

The Recurrent Neuron

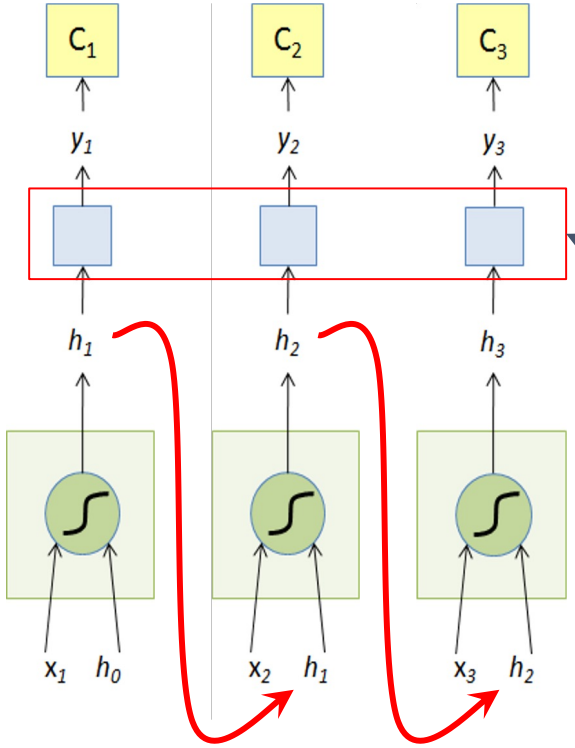
- x_t: Input at time t
- h_{t-1}: State at time t-1



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

Source: Slides by Arun

Unfolding an RNN

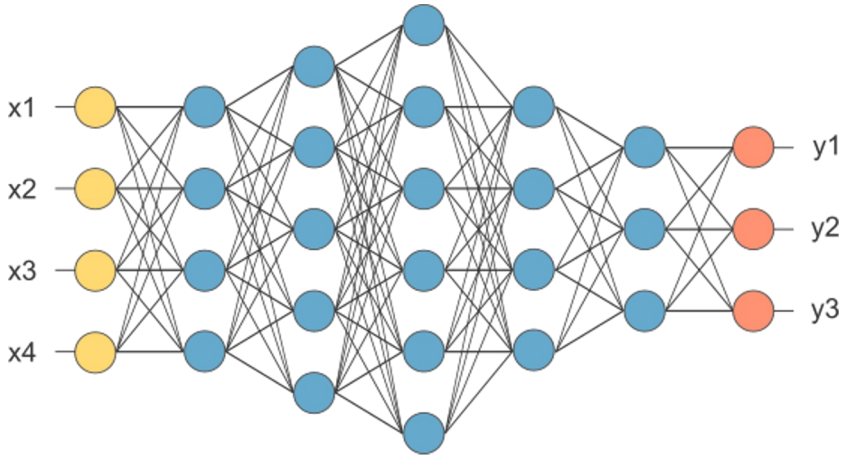


$$h_t = f(W_h h_{t-1} + W_x x_t)$$

Weights shared over time!

Source: Slides by Arun

Making Feedforward Neural Networks Deep



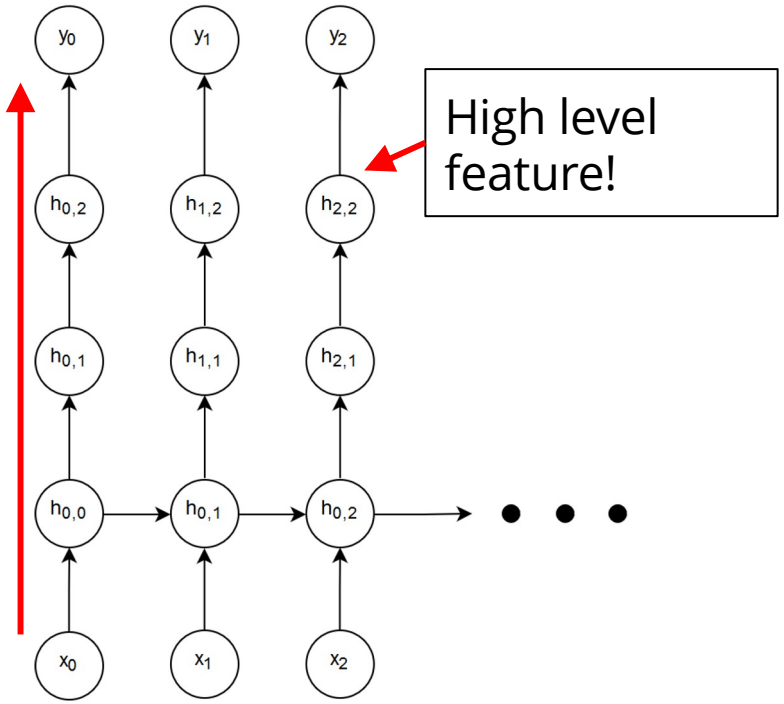
Source: http://www.opennn.net/images/deep_neural_network.png

Option 1: Feedforward Depth (d_f)

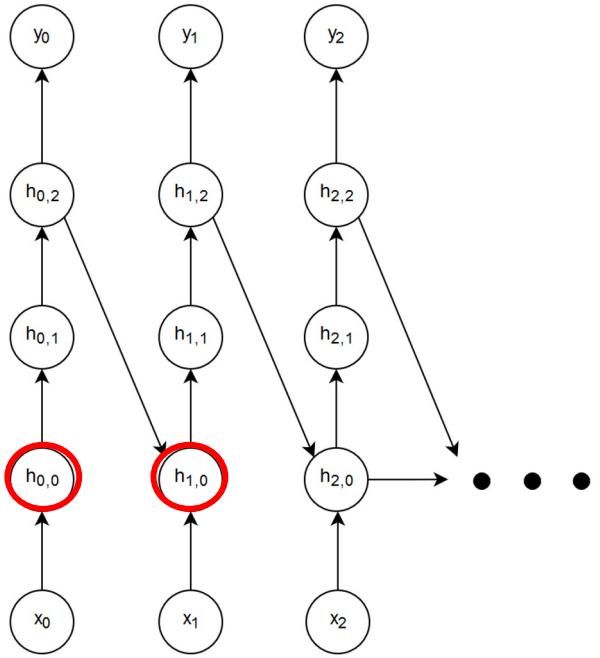
Notation: $h_{0,1} \Rightarrow$ time step 0, neuron #1

- **<u>Feedforward depth</u>**: **longest** path between an input and output at the **same timestep**

Feedforward depth = 4



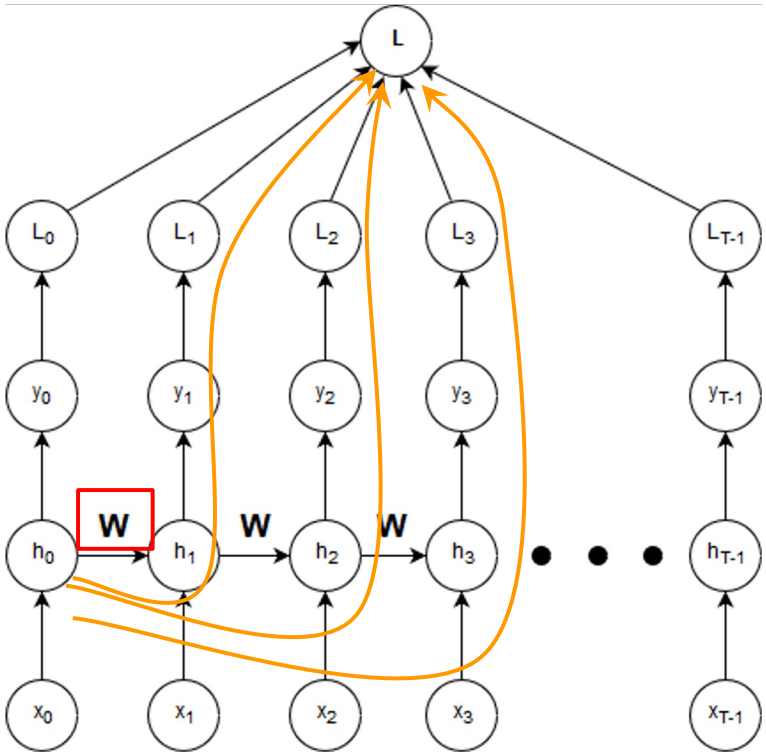
Option 2: Recurrent Depth (d_r)



Recurrent depth = 3

- **<u>Recurrent depth</u>**: Longest path between **same hidden state** in **successive timesteps**

Backpropagation Through Time (BPTT)



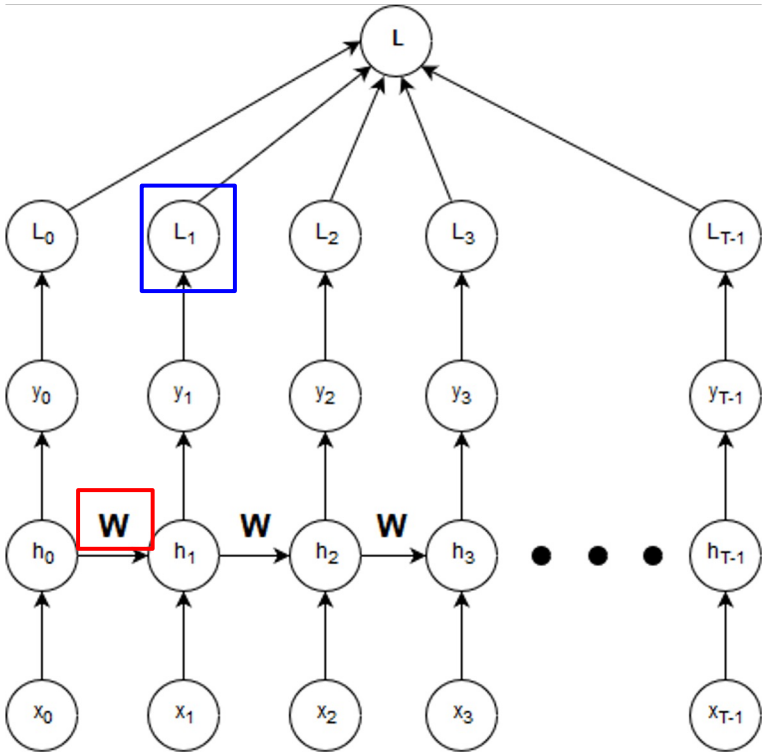
- Objective is to update the weight matrix:

$$\mathbf{W} \rightarrow \mathbf{W} - \alpha \frac{\partial L}{\partial \mathbf{W}}$$

- Issue: **W** occurs each timestep
- **Every** path from **W** to L is one dependency
- Find all paths from **W** to L!

(note: dropping subscript h from $\mathbf{W_h}$ for brevity)

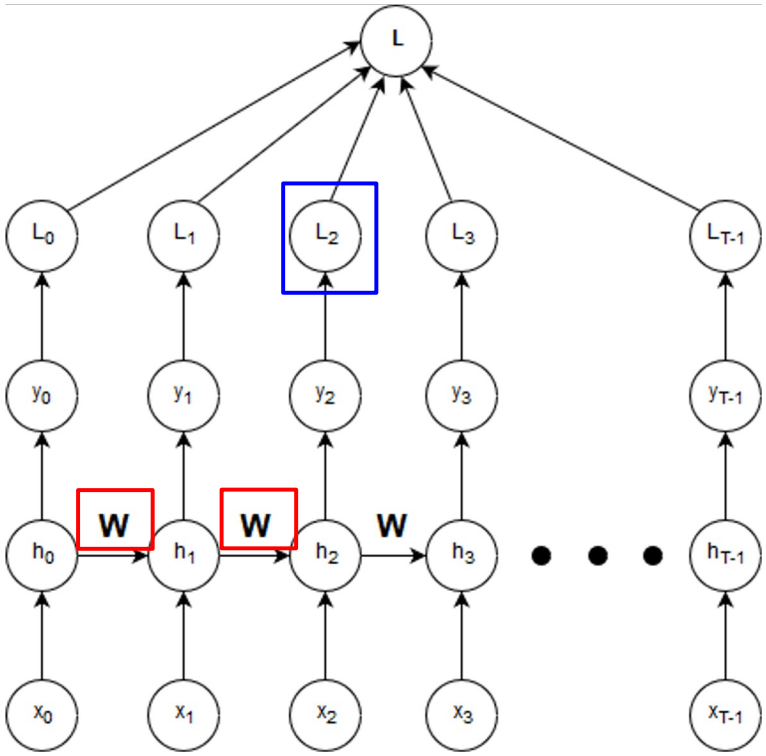
Systematically Finding All Paths



How many paths exist from W to L *through* L_1?

Just 1. Originating at h_0.

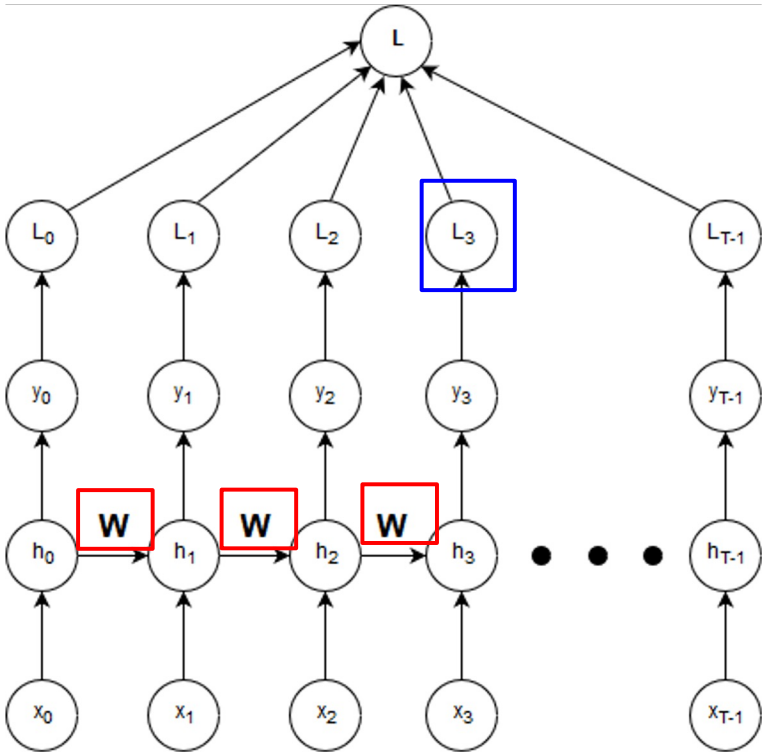
Systematically Finding All Paths



How many paths from **W** to L *through* L_2?

2. Originating at h_0 and h_1.

Systematically Finding All Paths



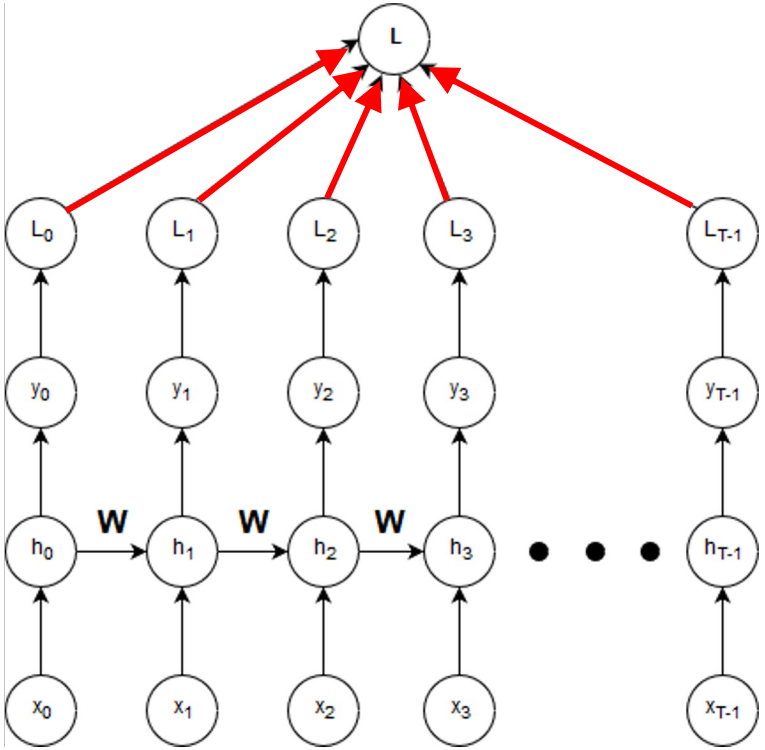
And 3 in this case.

Origin of path = basis for Σ

$$\frac{\partial L}{\partial \mathbf{W}}$$

The gradient has two summations:
1: Over L_j
2: Over h_k

Backpropagation as two summations

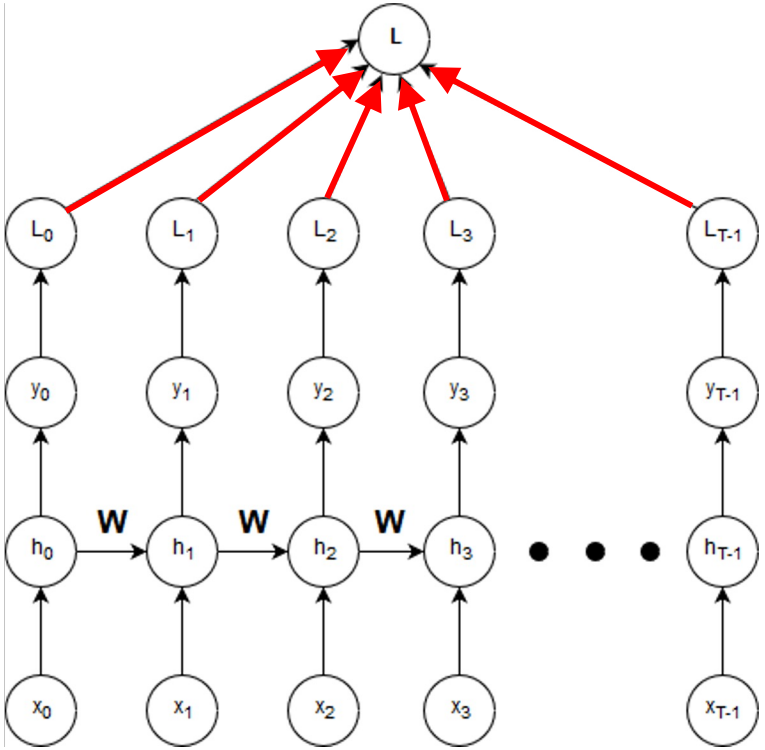


- **First summation over L**

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_{j=0}^{T-1} \frac{\partial L_j}{\partial \mathbf{W}} \frac{\partial L}{\partial L_j}$$

$$(L = L_1 + L_2 + \cdots L_{T-1})$$

Backpropagation as two summations

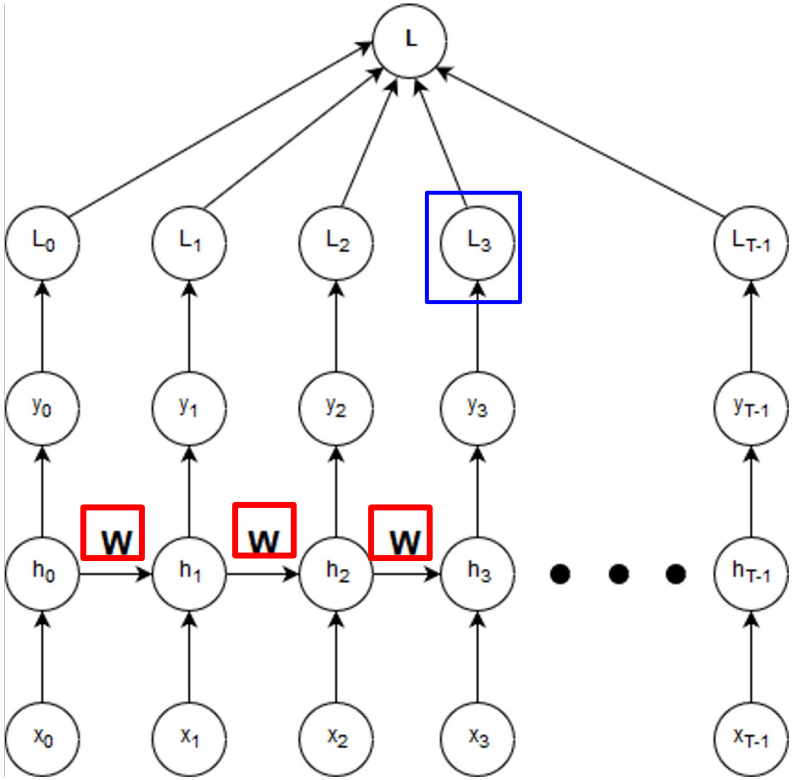


- **First summation over L**

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_{j=0}^{T-1} \frac{\partial L_j}{\partial \mathbf{W}} \cancel{\frac{\partial L}{\partial L_j}}$$

$$(L = L_1 + L_2 + \cdots L_{T-1})$$

Backpropagation as two summations

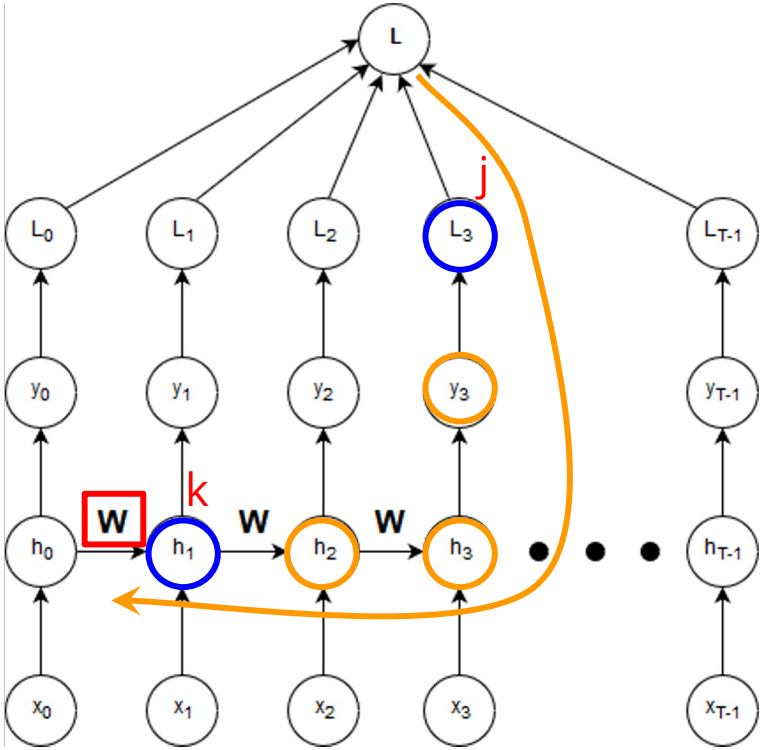


- **Second summation over h:** Each $\mathbf{L_j}$ depends on the weight matrices *before it*

$$\frac{\partial L_j}{\partial \mathbf{W}} = \sum_{k=1}^{j} \frac{\partial L_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$

L_j depends on all h_k before it.

Backpropagation as two summations

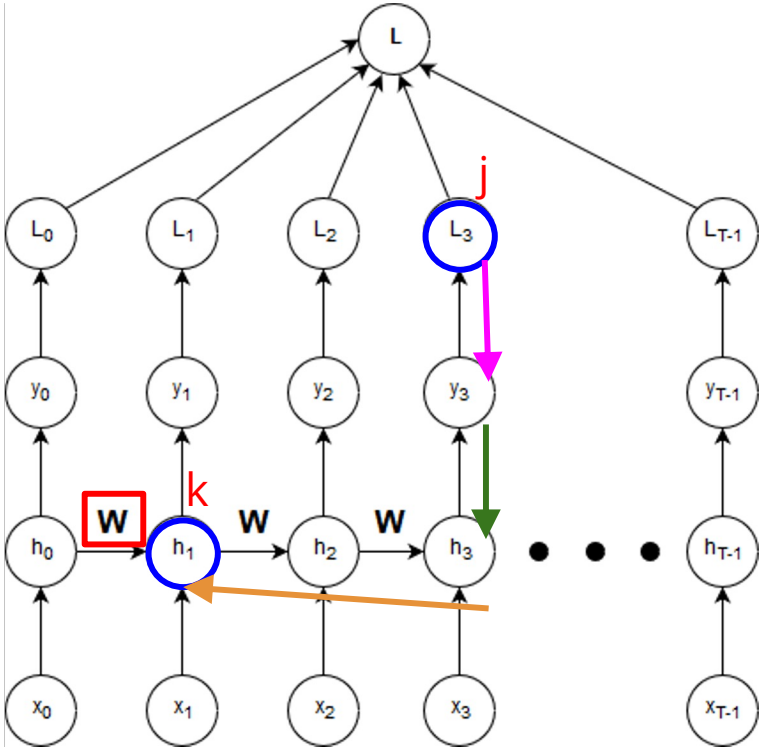


$$\frac{\partial L_j}{\partial W} = \sum_{k=1}^{j} \frac{\partial L_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$

- Use chain rule to fill missing steps

$$\frac{\partial L_j}{\partial \mathbf{W}} = \sum_{k=1}^{j} \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \frac{\partial h_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$

Backpropagation as two summations

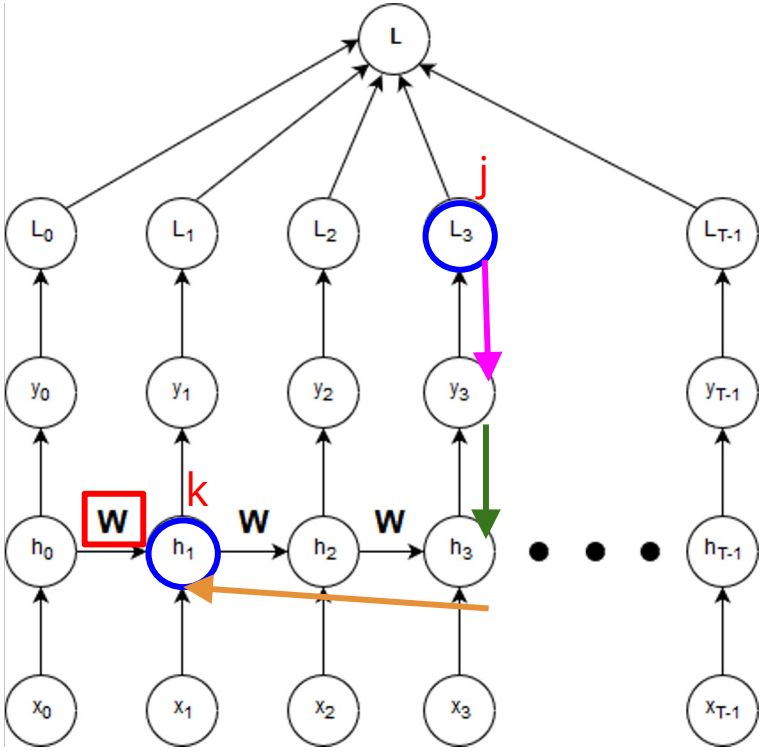


$$\frac{\partial L_j}{\partial W} = \sum_{k=1}^{j} \frac{\partial L_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$

- Use chain rule to fill missing steps

$$\frac{\partial L_j}{\partial \mathbf{W}} = \sum_{k=1}^{j} \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \frac{\partial h_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$

The Jacobian



$$\frac{\partial L_j}{\partial \mathbf{W}} = \sum_{k=1}^{j} \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \frac{\partial h_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$

Indirect dependency. One final use of the chain rule gives:

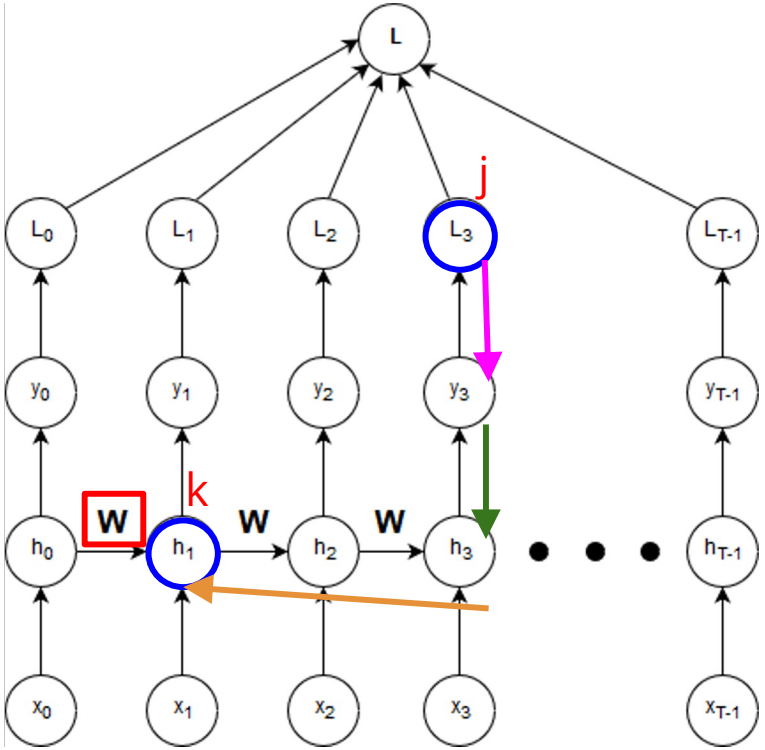
$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}}$$

"The Jacobian"

The Final Backpropagation Equation

$$\frac{\partial L}{\partial \mathbf{W_h}} = \sum_{j=0}^{T-1} \sum_{k=1}^{j} \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \left(\prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W_h}}$$

Backpropagation as two summations



$$\frac{\partial L}{\partial \mathbf{W_h}} = \sum_{j=0}^{T-1} \sum_{k=1}^{j} \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \left(\prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W_h}}$$

- Often, to reduce memory requirement, we truncate the network
- Inner summation runs from *j-p* to *j* for some *p* ==> truncated BP through time

Expanding the Jacobian

$$\frac{\partial L}{\partial W} = \sum_{j=0}^{T-1} \sum_{k=1}^{j} \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \left(\prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W}}$$

$$h_m = f(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)$$

$$\frac{\partial h_m}{\partial h_{m-1}} = \mathbf{W}_h^T \operatorname{diag}\left(f'(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)\right)$$

The Issue with the Jacobian

$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^{j} \mathbf{W}_h^T \operatorname{diag}\left(f'(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)\right)$$

Weight Matrix

Derivative of activation function

Repeated matrix multiplications leads to **vanishing and exploding gradients**.
How? Let's take a slight detour.

Eigenvalues and Stability

- Consider identity activation function
- If Recurrent Matrix $\mathbf{W_h}$ is a diagonalizable:

$$W_h = Q \quad * \Lambda * Q^{-1}$$

Q matrix composed of eigenvectors of W_h

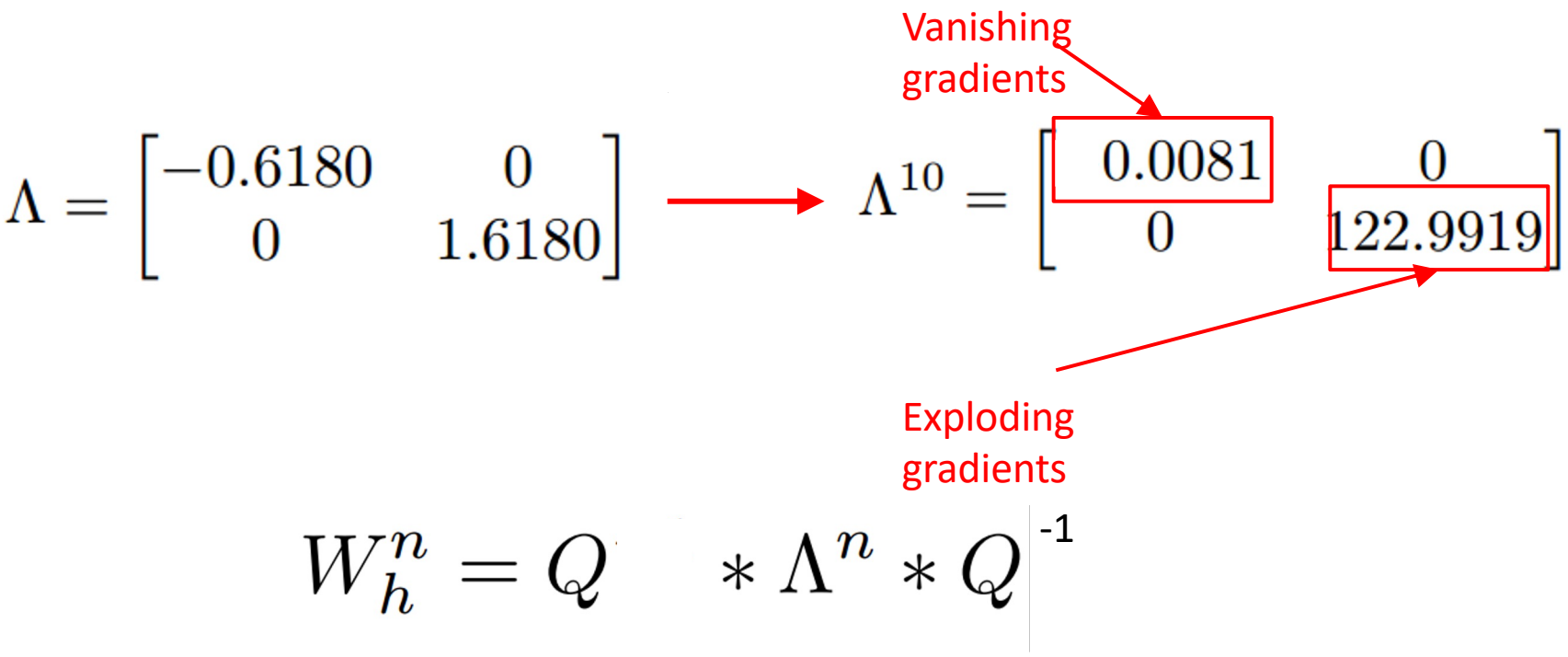
Λ is a diagonal matrix with eigenvalues placed on the diagonals

Computing powers of $\mathbf{W_h}$ is simple:

$$W_h^n = Q \quad * \Lambda^n * Q^{-1}$$

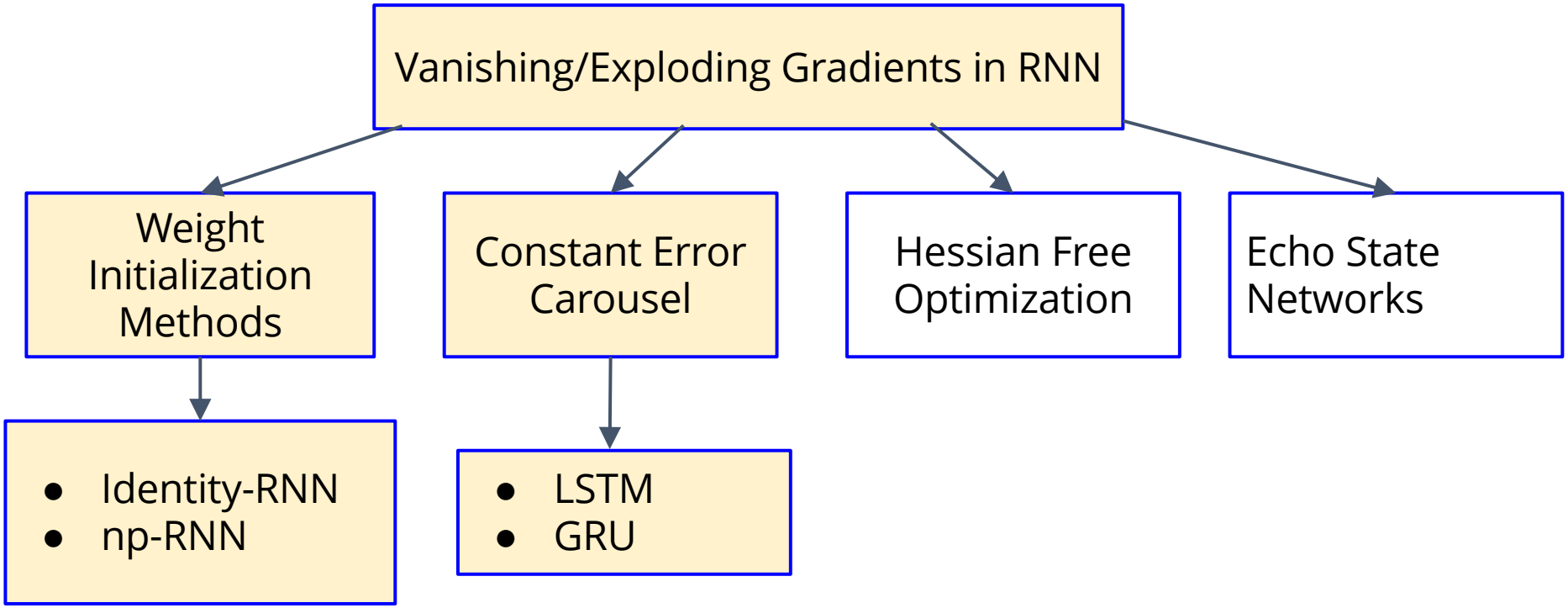
Bengio et al, "On the difficulty of training recurrent neural networks." (2012)

Eigenvalues and stability

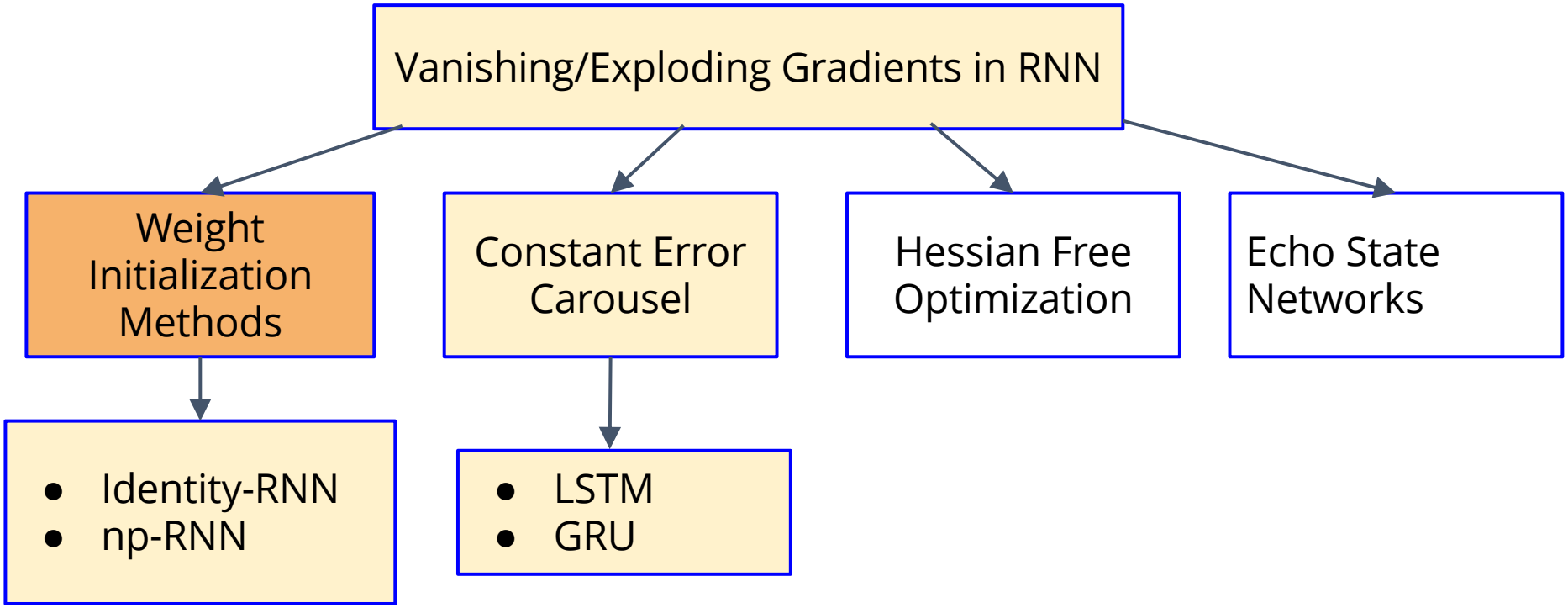


2. Learning Long Term Dependencies

Outline


Vanishing/Exploding Gradients in RNN
Weight Initialization Methods
Constant Error Carousel
Hessian Free Optimization
Echo State Networks
Identity-RNN
np-RNN
LSTM
GRU

Outline


Vanishing/Exploding Gradients in RNN
Weight Initialization Methods
Constant Error Carousel
Hessian Free Optimization
Echo State Networks
Identity-RNN
np-RNN
LSTM
GRU

Weight Initialization Methods

$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^{j} \mathbf{W}_h^T \operatorname{diag}\left(f'(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)\right)$$

- Activation function : ReLU

$$(\mathbf{W}_h^T)^n = Q \quad * \Lambda^n * Q^{-1}$$

Bengio et al,. "On the difficulty of training recurrent neural networks." (2012)

Weight Initialization Trick #1: IRNN

- Careful initialization of W_h with suitable eigenvalues

  ⇒ allows the RNN to learn in the initial epochs

  ⇒ hence can generalize well for further iterations

- W_h initialized to Identity
- Activation function: ReLU

Quoc Le et al., "A Simple Way to Initialize Recurrent Networks of Rectified Linear Units" arXiv 2015

Weight Initialization Trick #2: np-RNN

- W_h positive definite (+ve real eigenvalues)
- At least one eigenvalue is 1, others all less than equal to one
- Activation function: ReLU

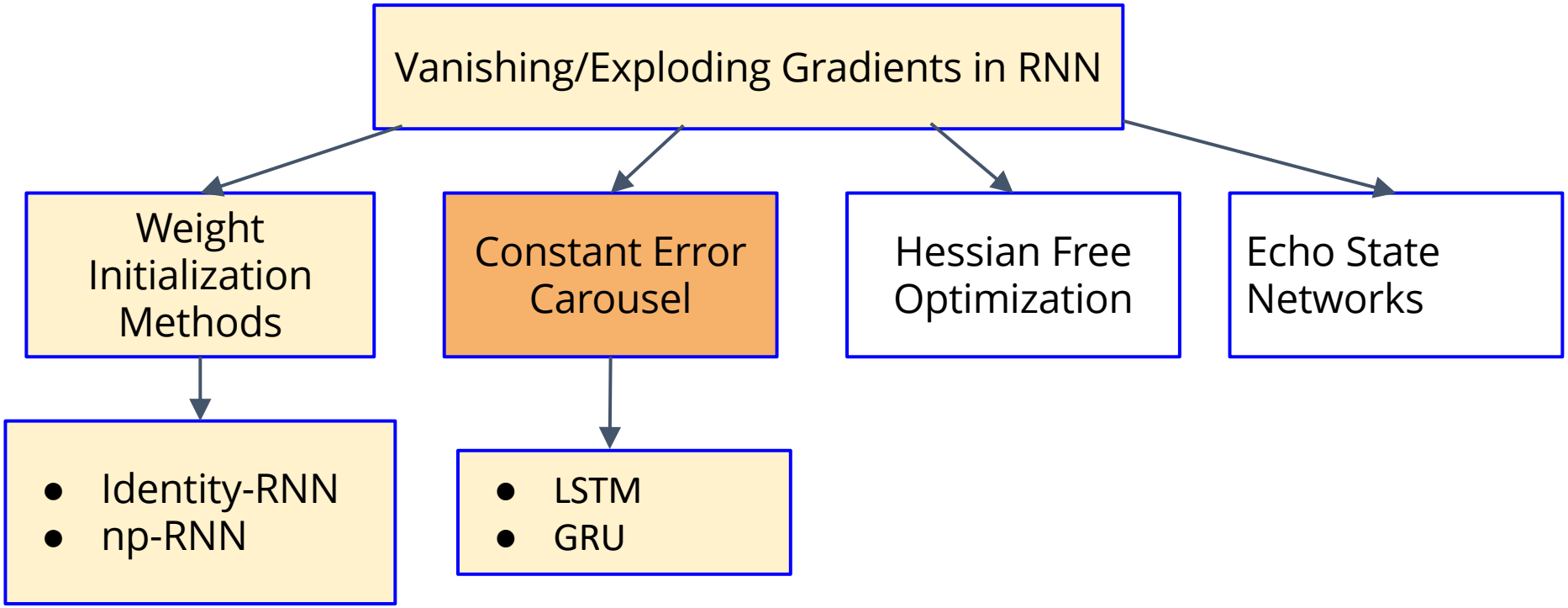
Talathi “Improving Performance of Recurrent Neural Network with ReLU nonlinearity” ICLR 2016

np-RNN vs IRNN

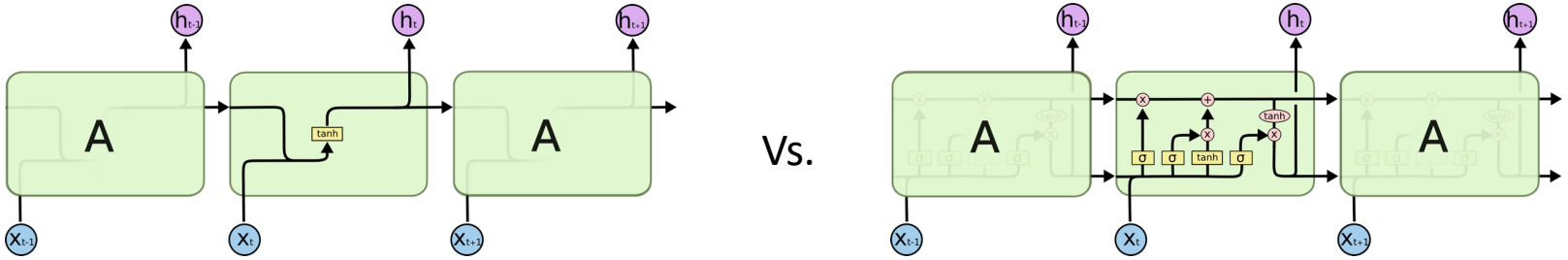
- Sequence Classification Task

RNN Type	Accuracy Test	Parameter Complexity Compared to RNN	Sensitivity to parameters
IRNN	67 %	x1	high
np-RNN	75.2 %	x1	low
LSTM **(???)**	78.5 %	x4	low

Outline

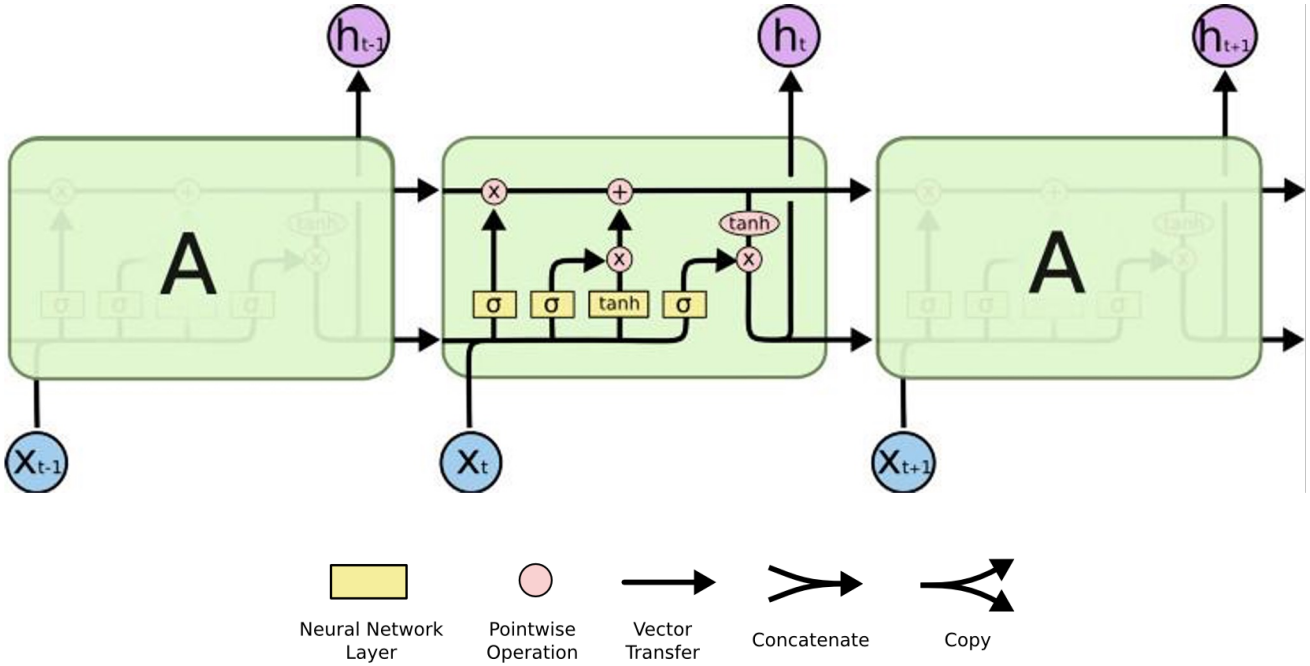

Vanishing/Exploding Gradients in RNN
Weight Initialization Methods
Constant Error Carousel
Hessian Free Optimization
Echo State Networks
Identity-RNN
np-RNN
LSTM
GRU

RNN vs. LSTM



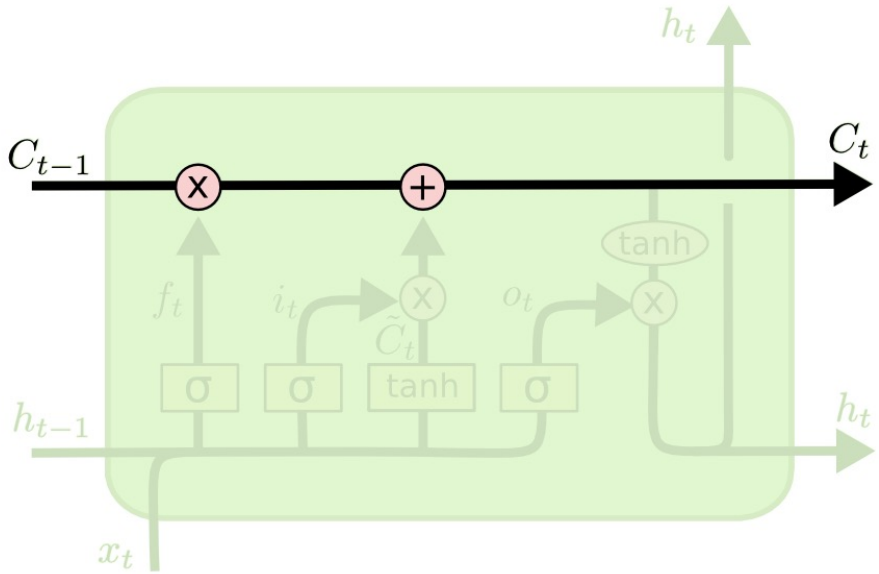
Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

The LSTM Network (Long Short-Term Memory)

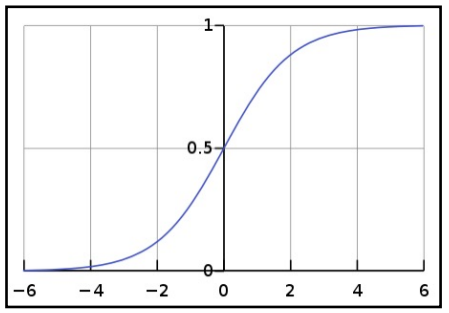


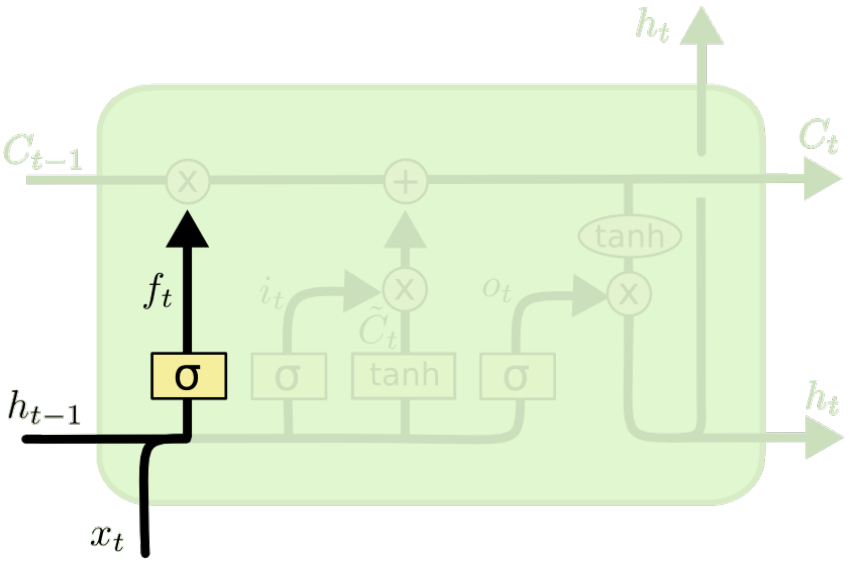
Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Avoid vanishing gradients



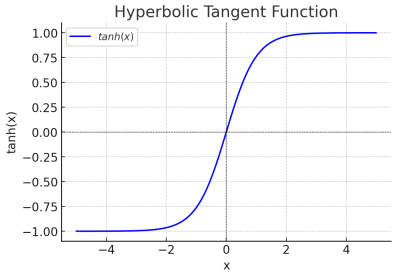
Decide “what to forget”

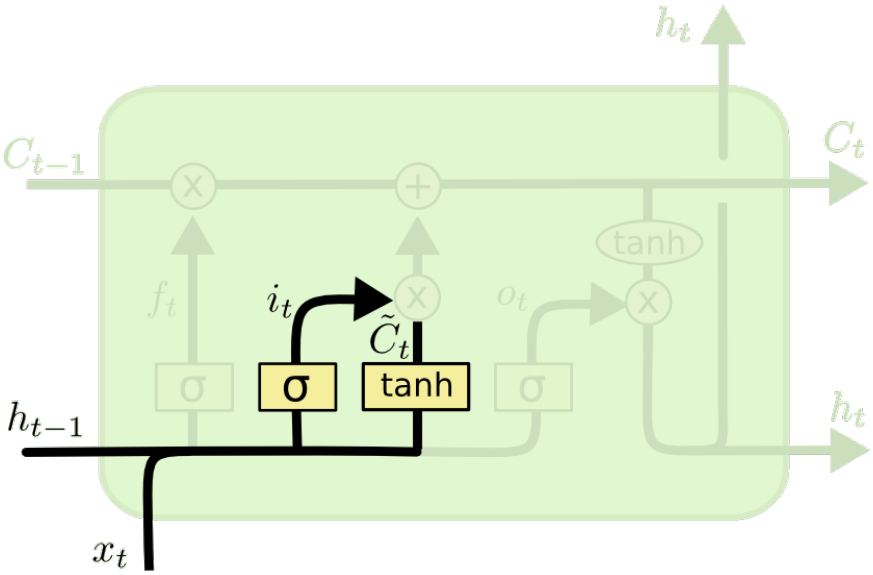




$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Decide “what to add”

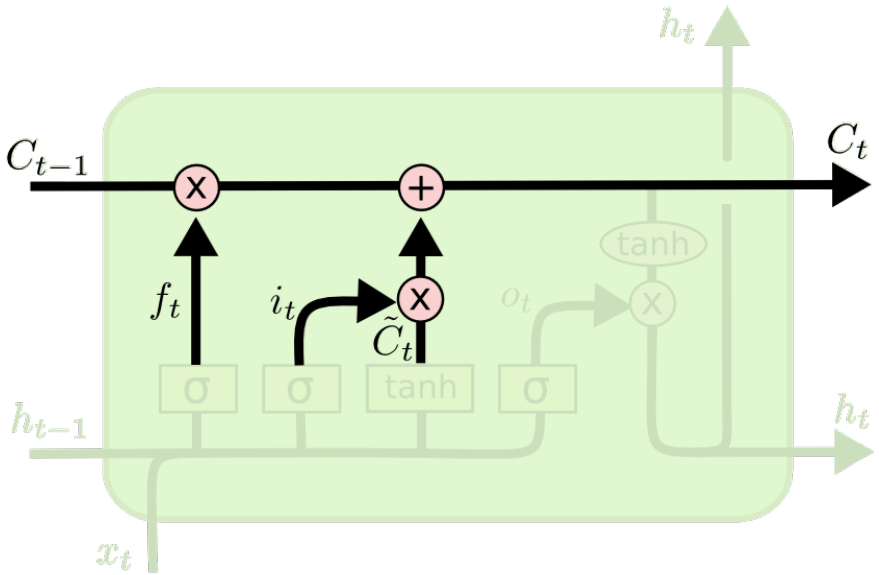




$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

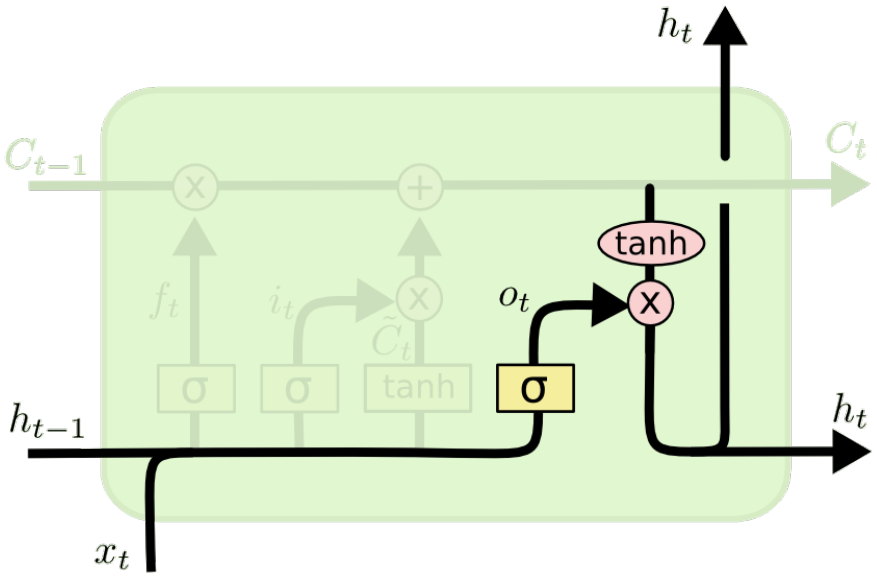
Update the state



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

Prepare output

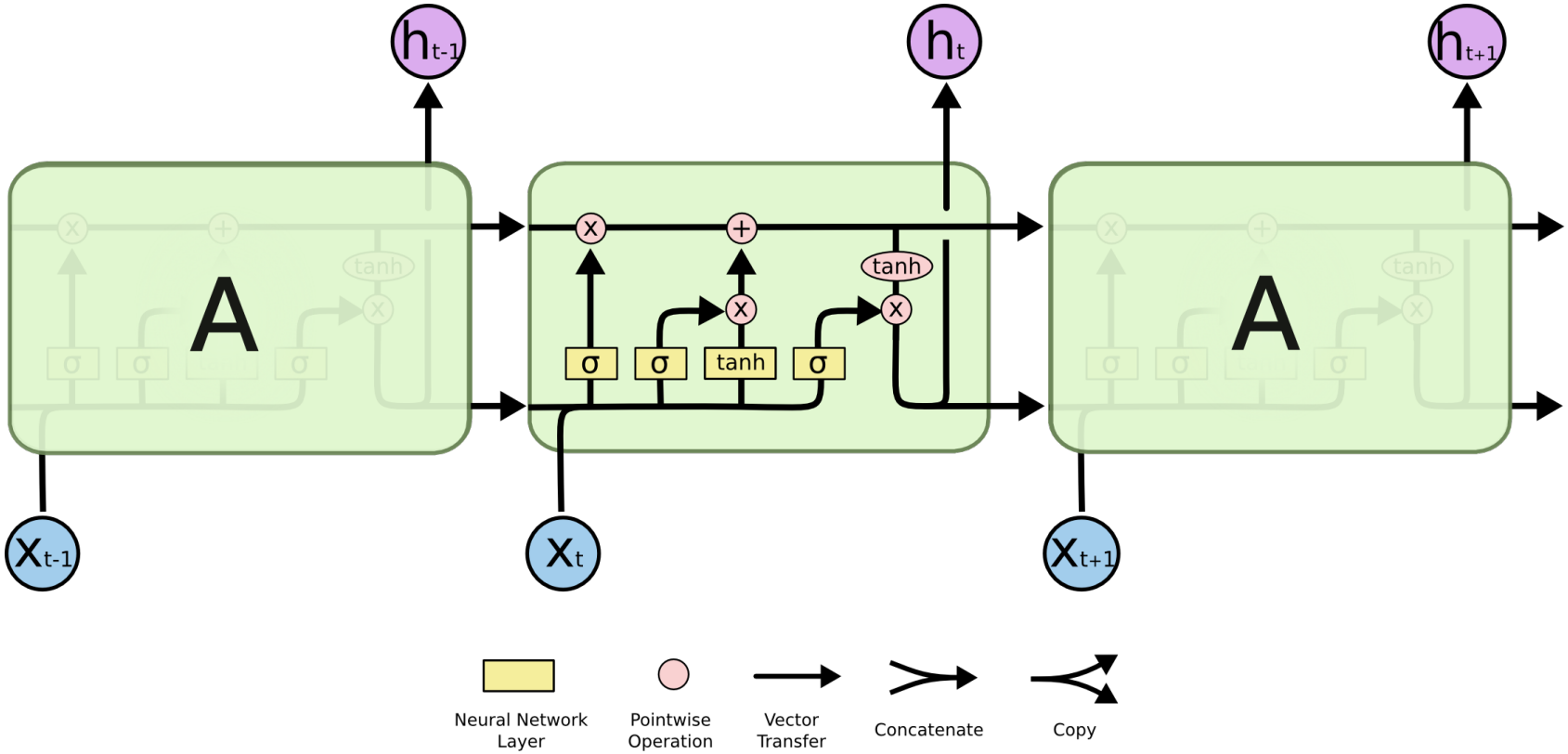
- Tanh re-normalizes output, and gate determines what to let through



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

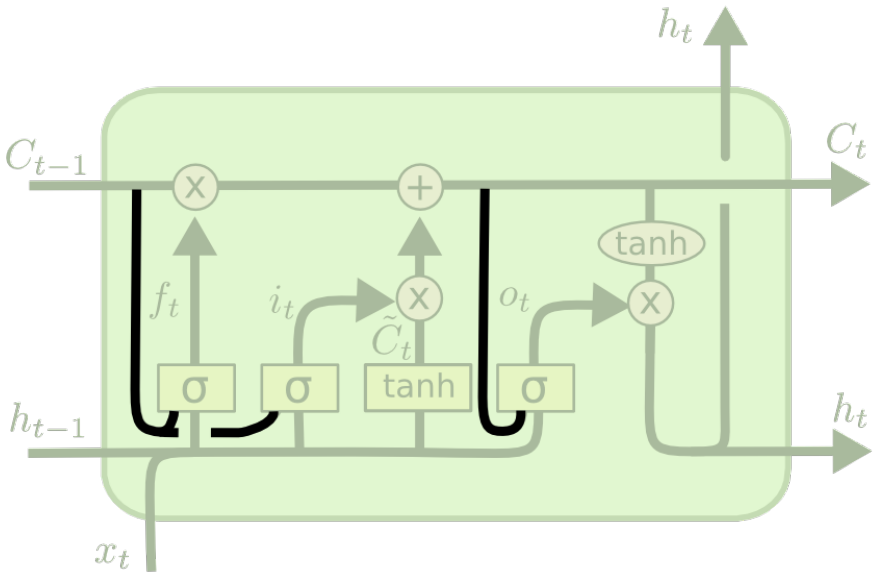
$$h_t = o_t * \tanh\left(C_t\right)$$

LSTM recap


h_{t-1}
h_t
h_{t+1}
A
A
tanh
σ
σ
tanh
σ
X_{t-1}
X_t
X_{t+1}
Neural Network Layer
Pointwise Operation
Vector Transfer
Concatenate
Copy

LSTM variants (“peephole connections”)

- Let the gate layers “peep” at the cell state



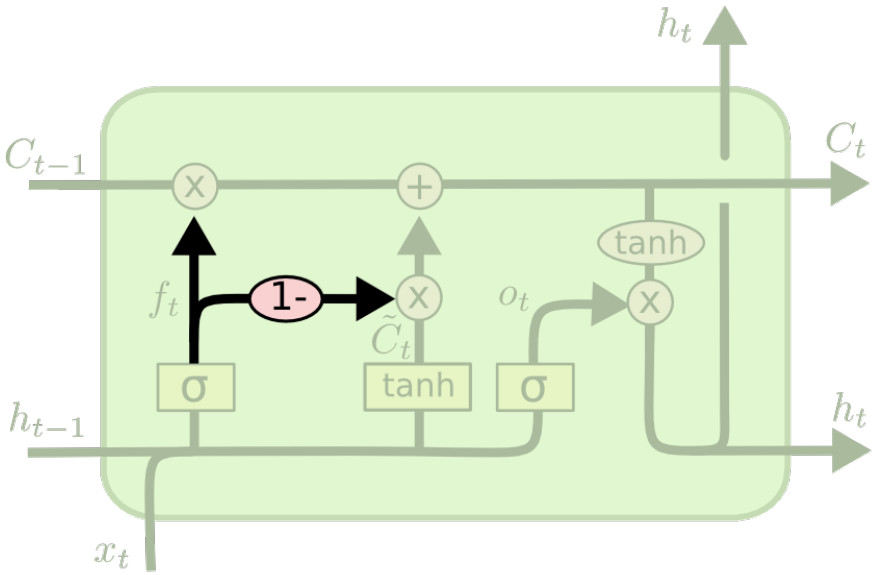
$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$

$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$

$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

LSTM variants (coupled gate)

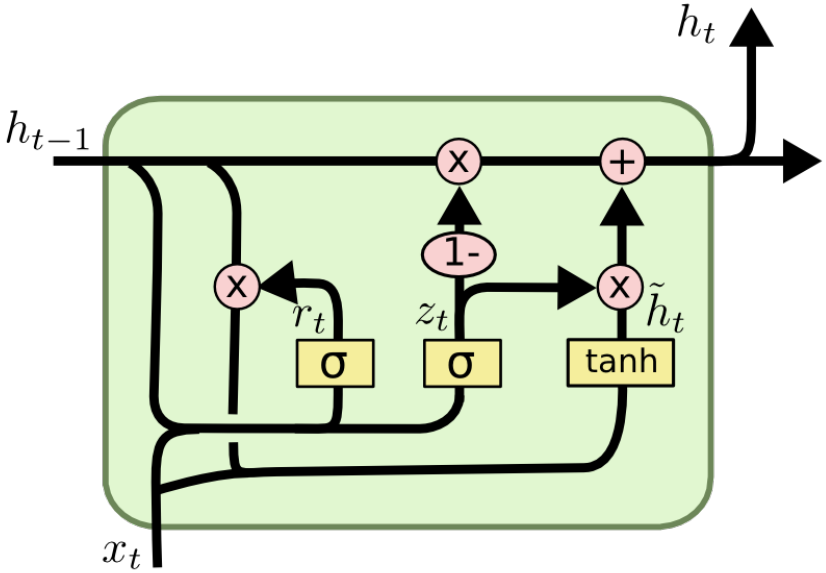
- Joint decision as to what delete/let through
  (forget when we're going to input something in its place)



$$C_t = f_t * C_{t-1} + (1 - \boldsymbol{f_t}) * \tilde{C}_t$$

GRU (Gated Recurrent Units)

- Combines the forget and input gates into a single “update” gate



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

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

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

Comparing GRU and LSTM

- Both **GRU and LSTM better than RNN with tanh**

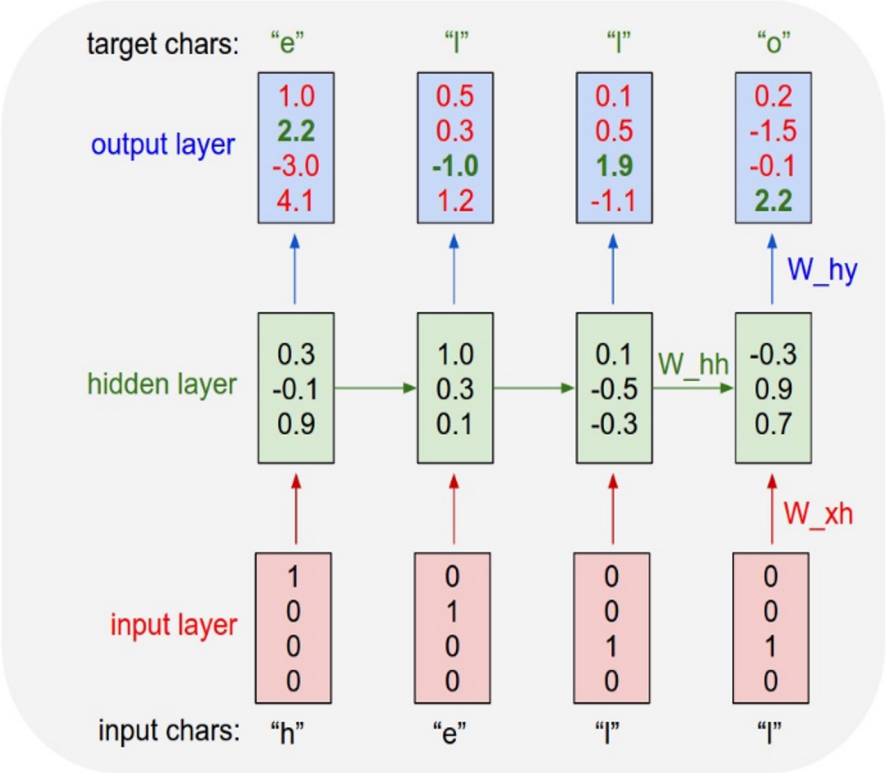
  (on music and speech modeling)
- GRU performs comparably to LSTM
- No clear consensus between GRU and LSTM

Source: Empirical evaluation of GRUs on sequence modeling, 2014

Section 3: Visualizing and Understanding Recurrent Networks

Character Level Language Modelling

Task: Predicting the next character given the current character



Andrej Karpathy, Blog on "Unreasonable Effectiveness of Recurrent Neural Networks"

Generated Text:

- Remembers to close a bracket
- Capitalize nouns
- 404 Page Not Found! :P The LSTM hallucinates it.

```
Naturalism and decision for the majority of Arab countries' capitalide was grounded
by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated
with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal
in the [[Protestant Immineners]], which could be said to be directly in Cantonese
Communication, which followed a ceremony and set inspired prison, training. The
emperor travelled back to [[Antioch, Perth, October 25|21]] to note, the Kingdom
of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known
in western [[Scotland]], near Italy to the conquest of India with the conflict.
Copyright was the succession of independence in the slop of Syrian influence that
was a famous German movement based on a more popular servicious, non-doctrinal
and sexual power post. Many governments recognize the military housing of the
[[Civil Liberalization and Infantry Resolution 265 National Party in Hungary]],
that is sympathetic to be to the [[Punjab Resolution]]
(PTS)[http://www.humah.yahoo.com/guardian.
cfm/7754800786d1751963s89.htm Official economics Adjoint for the Nazism, Montgomery
was swear to advance to the resources for those Socialism's rule,
was starting to signing a major tripad of aid exile.]]
```

Andrej Karpathy, Blog on “Unreasonable Effectiveness of Recurrent Neural Networks”

100 th iteration

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

300 th iteration

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

700 th iteration

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

2000 th iteration

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

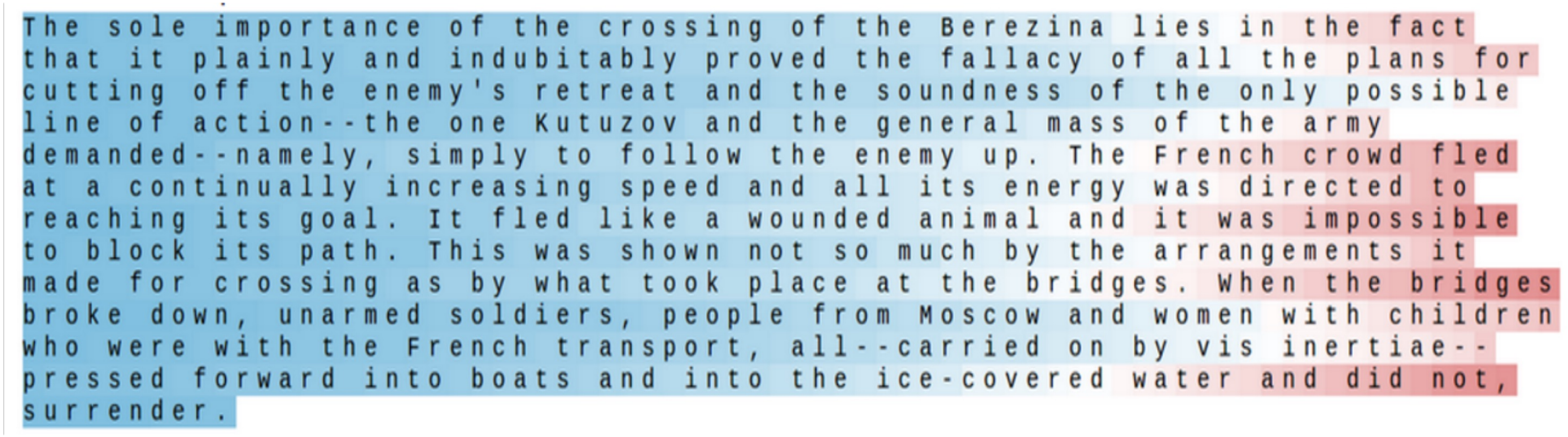
Andrej Karpathy, Blog on “Unreasonable Effectiveness of Recurrent Neural Networks”

Features RNN Captures in Common Language ?

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

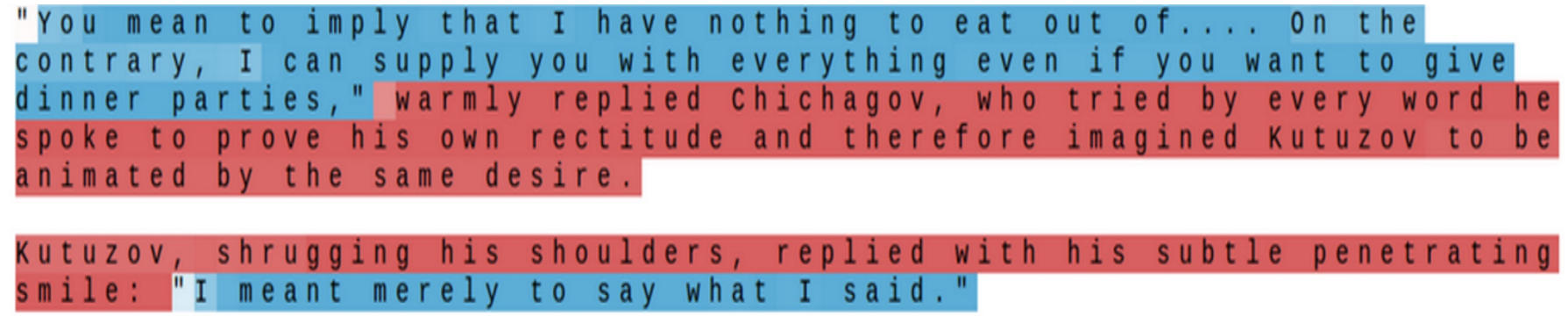
Cell Sensitive to Position in Line

- Can be interpreted as tracking the line length



Andrej Karpathy, Blog on “Unreasonable Effectiveness of Recurrent Neural Networks”

Cell That Turns On Inside Quotes



Andrej Karpathy, Blog on “Unreasonable Effectiveness of Recurrent Neural Networks”

Features RNN Captures in C Language?

```
for (i = 0; i < 16; i++) {
  if (k & (1 << 1))
    pipe = (in_use & UMXTHREAD_UNCCA) +
      ((count & 0x00000000ffffffff8) & 0x000000f) << 8;
  if (count == 0)
    sub(pid, ppc_md.kexec_handle, 0x20000000);
  pipe_set_bytes(i, 0);
}
/* Free our user pages pointer to place camera if all dash */
```

Andrej Karpathy, Blog on “Unreasonable Effectiveness of Recurrent Neural Networks”

Cell That 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;
}
```

Andrej Karpathy, Blog on “Unreasonable Effectiveness of Recurrent Neural Networks”

Cell That Is Sensitive To Indentation

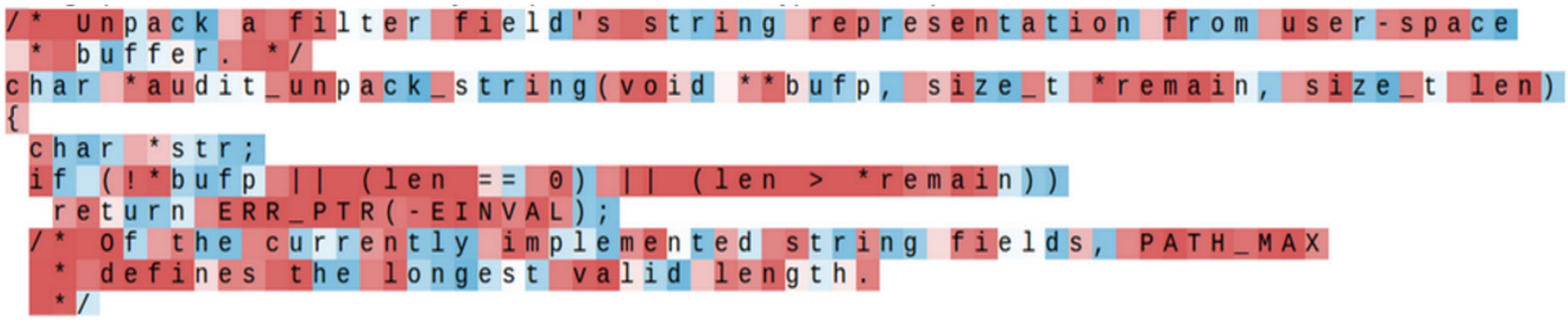
- Can be interpreted as tracking indentation of code.
- Increasing strength as indentation increases

```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
  int i;
  if (classes[class]) {
    for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
      if (mask[i] & classes[class][i])
        return 0;
  }
  return 1;
}
```

Andrej Karpathy, Blog on “Unreasonable Effectiveness of Recurrent Neural Networks”

Non-Interpretable Cells

- Only 5% of the cells show such interesting properties
- Large portion of the cells are **not interpretable** by themselves



Andrej Karpathy, Blog on “Unreasonable Effectiveness of Recurrent Neural Networks”

Key Takeaways

- **Recurrent neural network for sequential data processing**
  - Feedforward depth
  - Recurrent depth
- **Long term dependencies** are a major problem in RNNs. Solutions:
  - Intelligent weight initialization
  - LSTMs / GRUs
- **Visualization** helps
  - Analyze finer details of features produced by RNNs

References

- Survey Papers
  - Lipton, Zachary C., John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning, arXiv preprint arXiv:1506.00019 (2015).
  - Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Chapter 10: Sequence Modeling: Recurrent and Recursive Nets. MIT Press, 2016.
- Training
  - Semeniuta, Stanislau, Aliaksei Severyn, and Erhardt Barth. Recurrent dropout without memory loss. arXiv preprint arXiv:1603.05118 (2016).
  - Arjovsky, Martin, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. arXiv preprint arXiv:1511.06464 (2015).
  - Le, Quoc V., Navdeep Jaitly, and Geoffrey E. Hinton. A simple way to initialize recurrent networks of rectified linear units. arXiv preprint arXiv:1504.00941 (2015).
  - Cooijmans, Tim, et al. Recurrent batch normalization. arXiv preprint arXiv:1603.09025 (2016).

References (contd)

- Architectural Complexity Measures
  - Zhang, Saizheng, et al, Architectural Complexity Measures of Recurrent Neural Networks. Advances in Neural Information Processing Systems. 2016.
  - Pascanu, Razvan, et al. How to construct deep recurrent neural networks. arXiv preprint arXiv:1312.6026 (2013).
- RNN Variants
  - Zilly, Julian Georg, et al. Recurrent highway networks. arXiv preprint arXiv:1607.03474 (2016)
  - Chung, Junyoung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks, arXiv preprint arXiv:1609.01704 (2016).
- Visualization
  - Karpathy, Andrej, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. arXiv preprint arXiv:1506.02078 (2015).
  - Hendrik Strobelt, Sebastian Gehrmann, Bernd Huber, Hanspeter Pfister, Alexander M. Rush. LSTMVis: Visual Analysis for RNN, arXiv preprint arXiv:1606.07461 (2016).