Section 12 Recurrent Neural Network

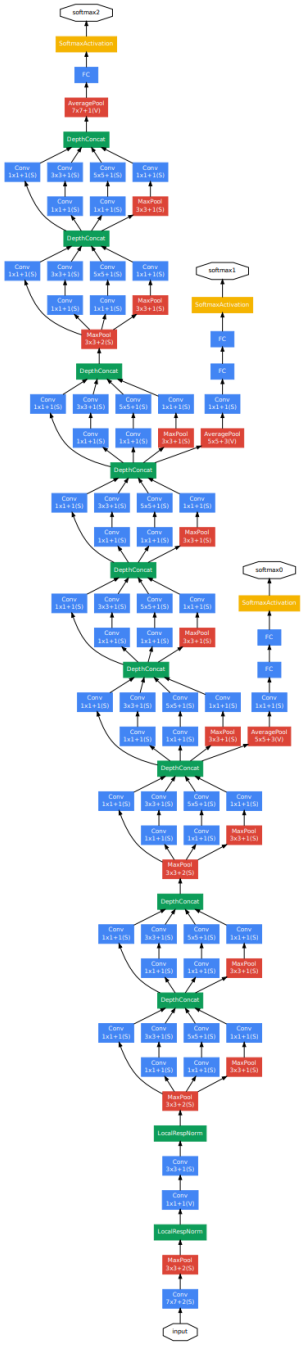
- Recurrent Neural Network
- Computational Graph and backpropagation
- Applications of RNN
- Long Short Term Memory (LSTM)

Types of Artificial Neural Networks

Artificial neural networks provide a powerful set of classifiers since for any input shape dim(X) and any output shape dim(Y) we have many **deep architectures** that may allows us to fit the network to the problem at hand.

The challenge of course is filling in the middle for each of these cases, while taking into consideration training time, data composition and computational power. The appeal of neural networks is that it's very easy to set a problem up computational power alone will solve it.

Sometimes, this is the case as with the MNIST data set and the perceptron networks. But often a clever solution requires more complex architecture than just densely connected layers.



Modern artificial neural networks can be sorted into three broad categories based on their structure:

Feed forward networks: A trained feed forward network acts like a function, taking in a set of data at one end and returning a new set of data at the other. Feed forward networks are the most common type, they take an input, process it through a series of operations, and return some output.

Feed forward networks can be labeling or regression, but they can also be generative networks (returning more data) or unsupervised, returning a dimensional reduction, clustering or other description of the data. However, once trained the weights θ are fixed for all prediction.

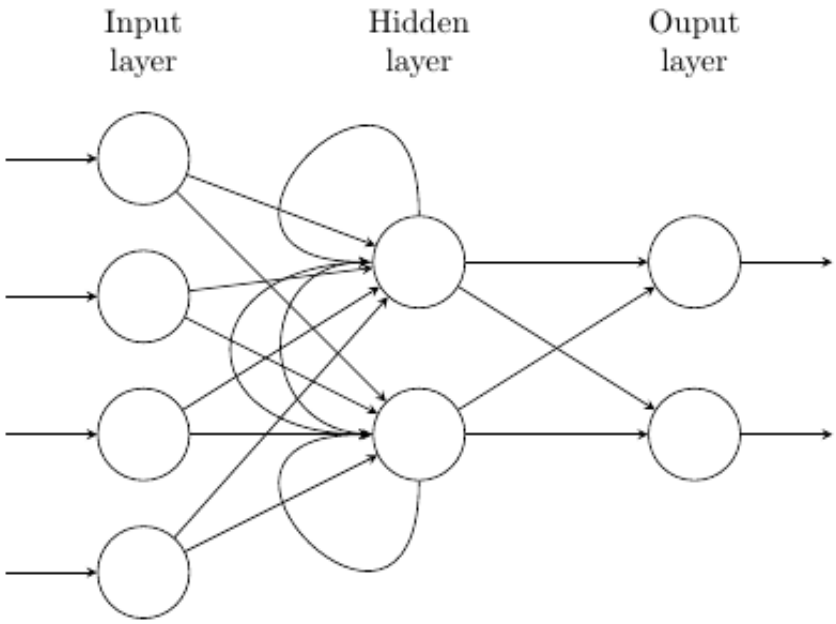
Recurrent neural networks: Trained recurrent networks are stateful. That is, RNN's take data and return an output but the remember the last M pieces of data sent through in an internal state. (We will study RNN in this lecture.)

Symmetrically Connected Networks: A trained SCN is a densely connected network with an update rule. For any initial value of the nodes, the function "updates", moving at each step towards a "lower energy state". The result is achieved when updating no longer changes the state.

Zoo of common architectures: https://www.asimovinstitute.org/author/fjodorvanveen/

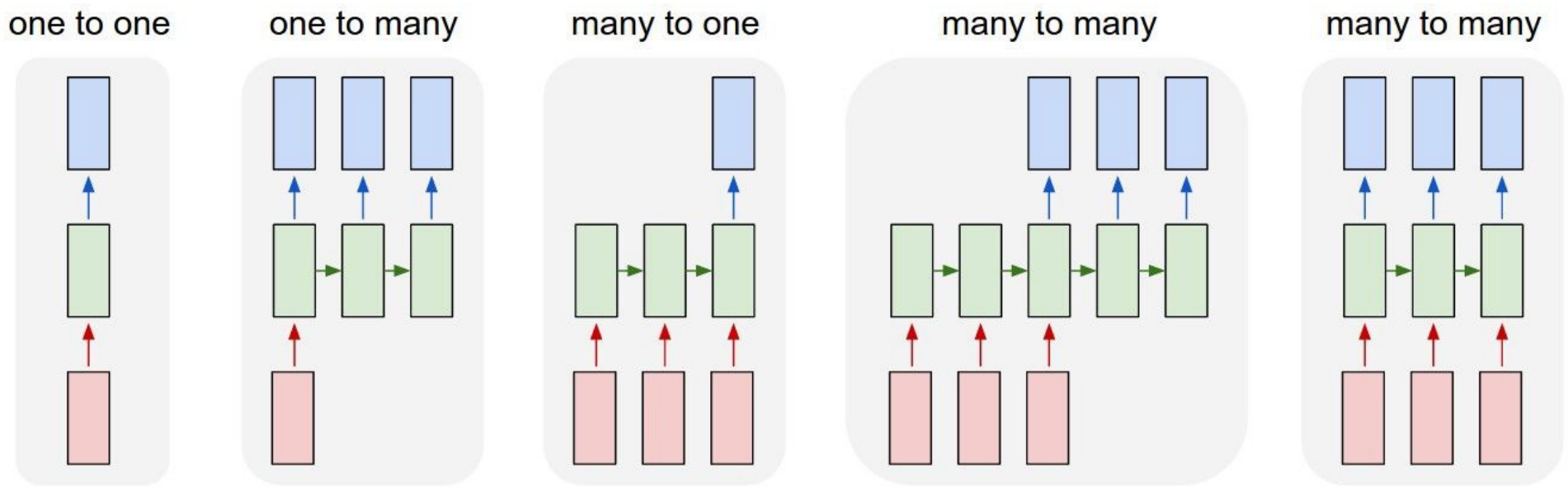
➢ Recurrent Neural Networks (RNN)

In RNN, each of the neurons in hidden layers receives an input with a specific delay *in time*. It allow previous outputs to be used as inputs while having hidden states. (RNN usually has a short-term memory. Long-term memory is also used in some research problems.)



A RNN can take in a series of inputs and produce a series of outputs, as in predicting time series data like stock prices or traffic across a network.

Recurrent Neural Networks: Process Sequences (for sequence data.)

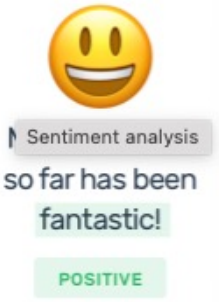


Vanilla Neural Networks

Image Captioning

A baseball player throws a ball

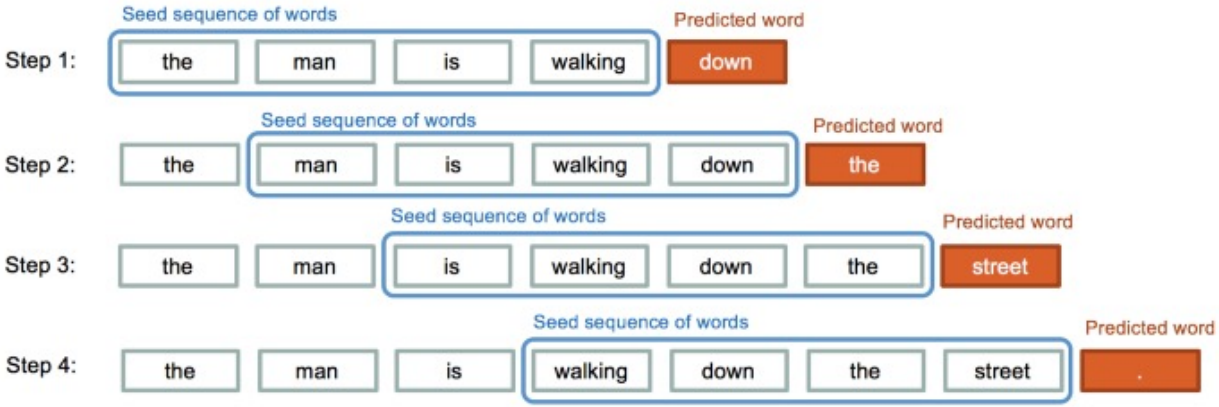
Sentiment Classification



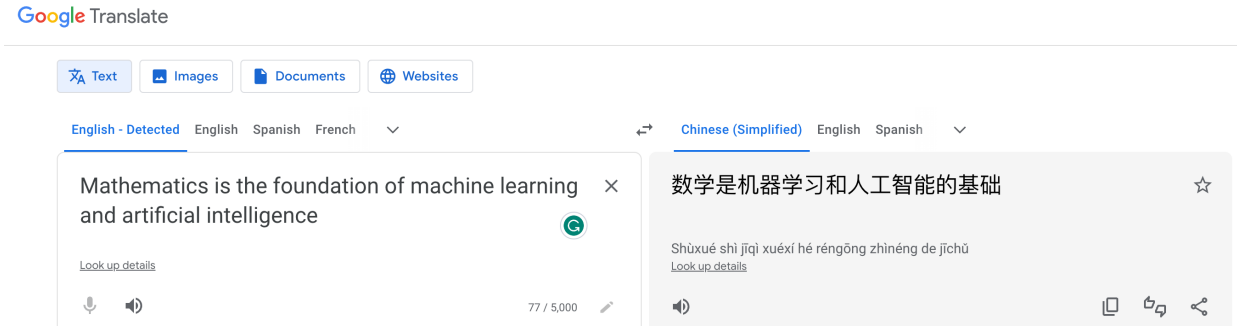
Machine Translation

Video classification on frame level

- Word prediction:



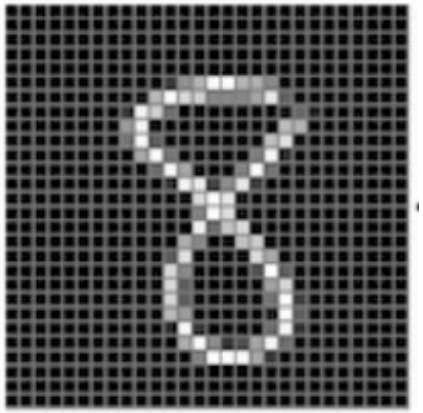
- Machine Translation:



- Sequential Processing of Non-Sequence Data.

1. Classify images by taking a series of “glimpses”

2. Generate images one piece at a time.



Ba, Mnih, and Kavukcuoglu, “Multiple Object Recognition with Visual Attention”, ICLR 2015.
Gregor et al, “DRAW: A Recurrent Neural Network For Image Generation”, ICML 2015

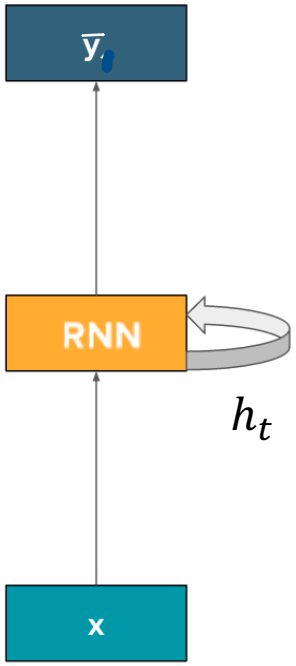
A **recurrent neural network** has **state variables** that can be changed at runtime and that persist between prediction runs.

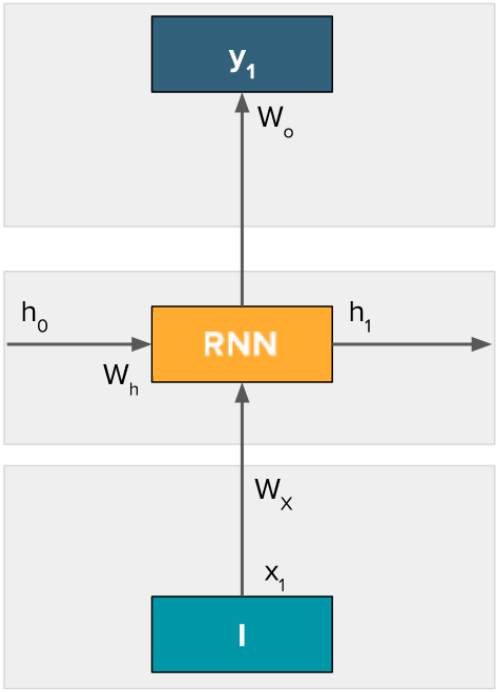
For example, in text prediction an RNN may predict one word at a time while "remembering" its previous predictions.

A **recurrent node** is often represented in “wrapped” form. RNN’s are used extensively in *time series prediction* and *natural language processing*.

Usually want to predict a vector at some time steps

State variables





Sequence of vectors $\vec{x}$ by applying a recurrence formula at every time step:

RNN new hidden state

$$h_t = f_W(h_{t-1}, \vec{x}_t)$$

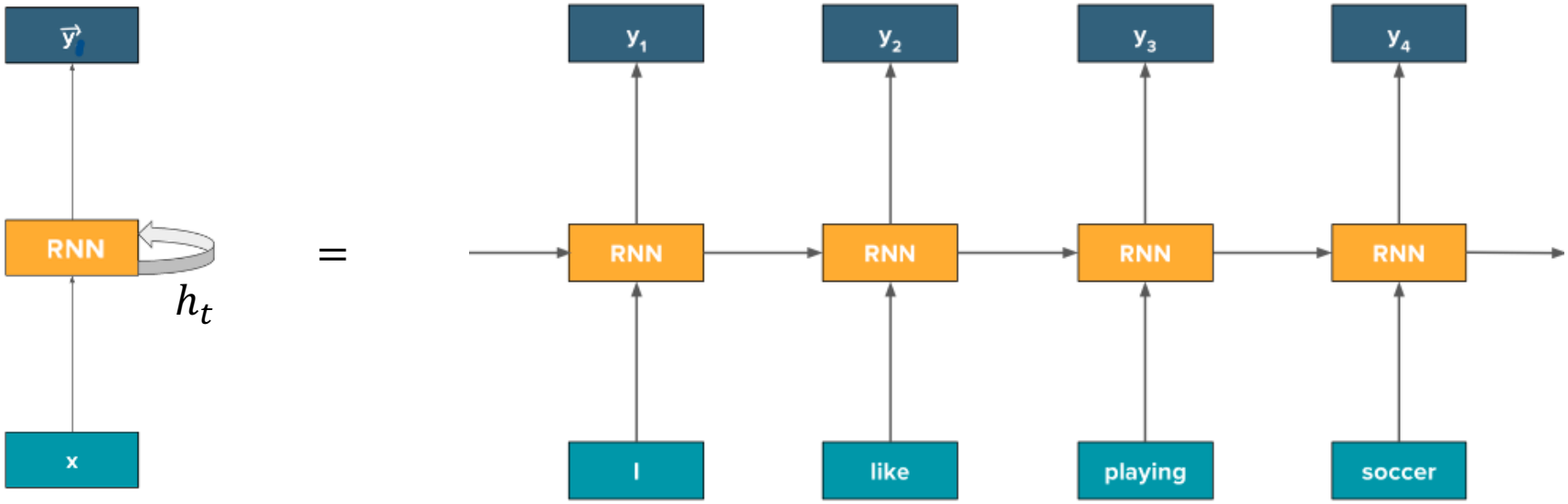
f_W : Function with parameters W.

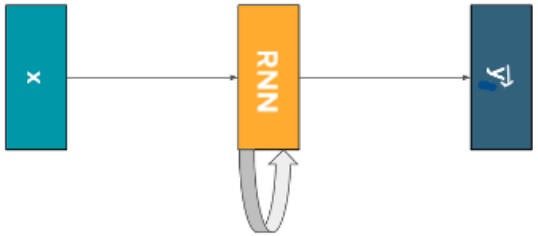
$\vec{x}_t$: input vector at some time step.

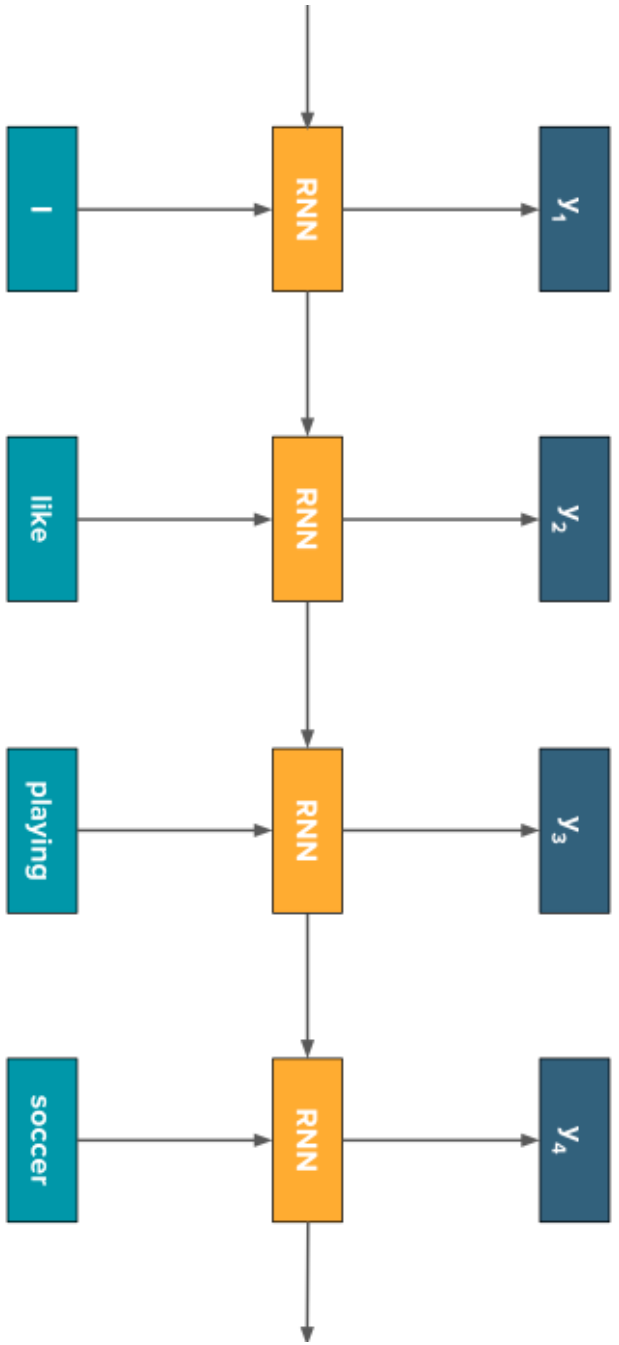
RNN output

$$y_t = g_{W_{hy}}(h_t)$$

g_W : Function with parameters W_{hy}.





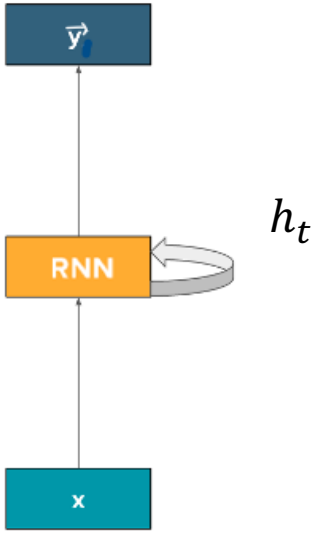


Notice: the same function and the same set of parameters are used at every time step.

Vanilla Recurrent Neural Networks

State space equations in feedback dynamical systems.

The state consists of a single “hidden” vector h:



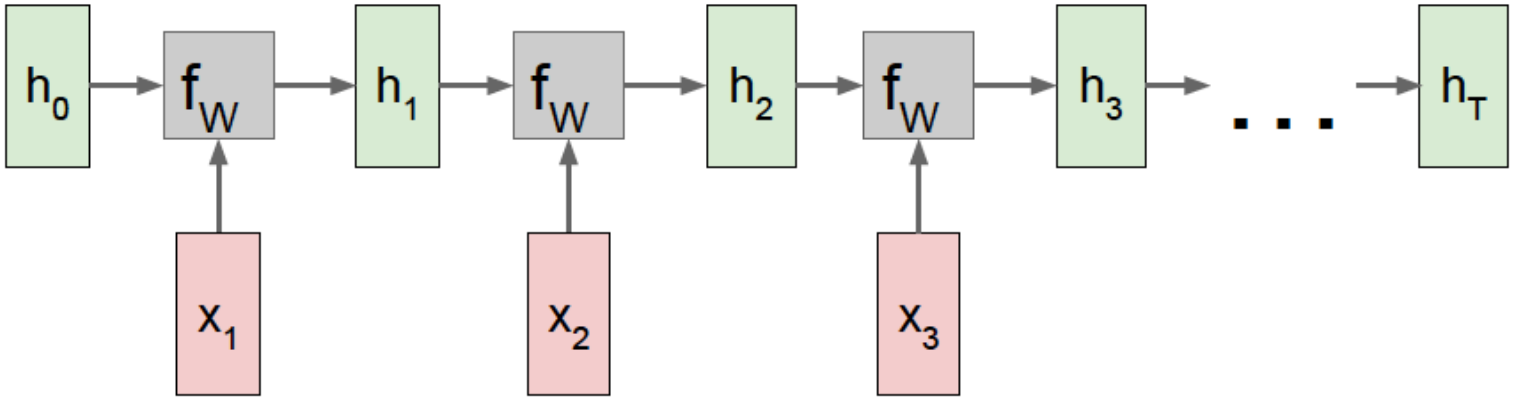
h_t

$$h_t = f_W(h_{t-1}, x_t)$$

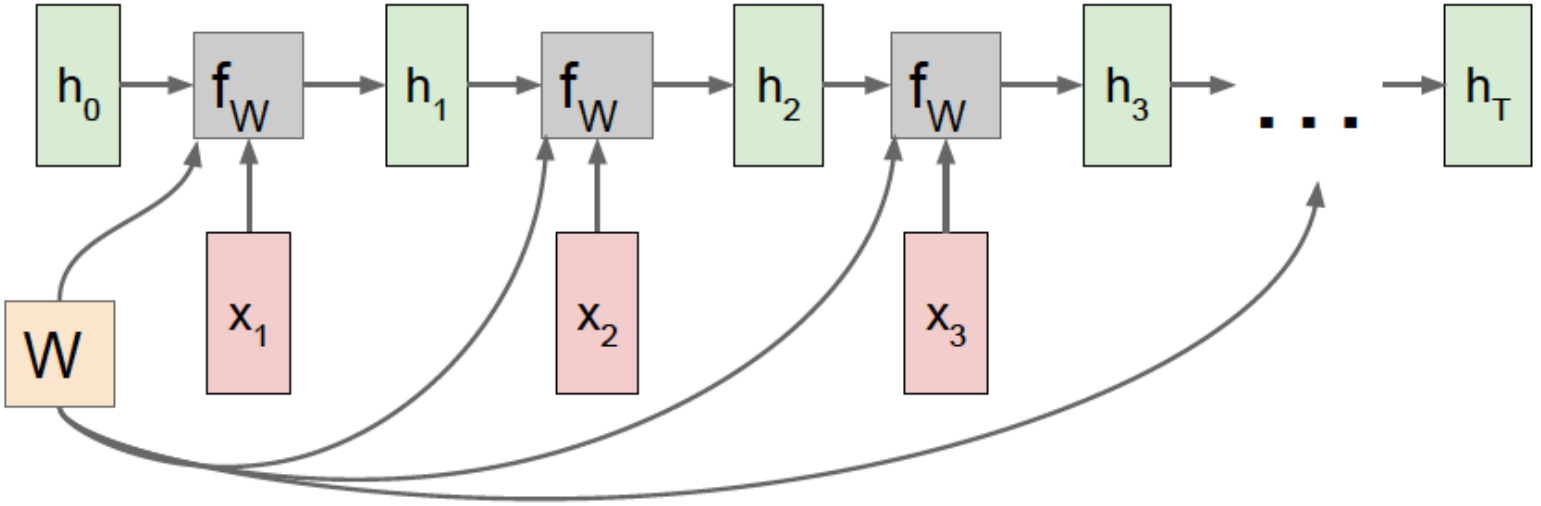
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t \qquad \text{Or } y_t = \text{softmax}\,(W_{hy}h_t)$$

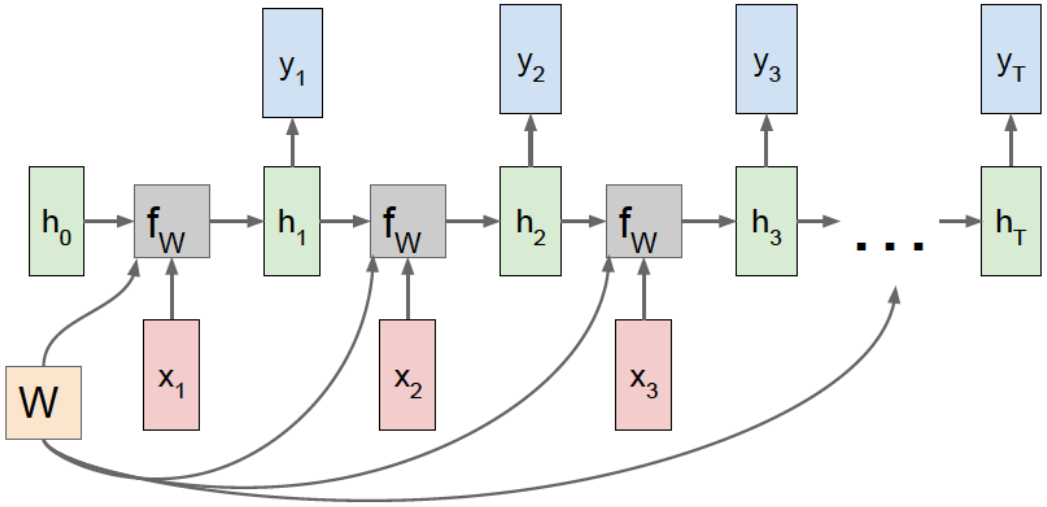
RNN: Computational Graph



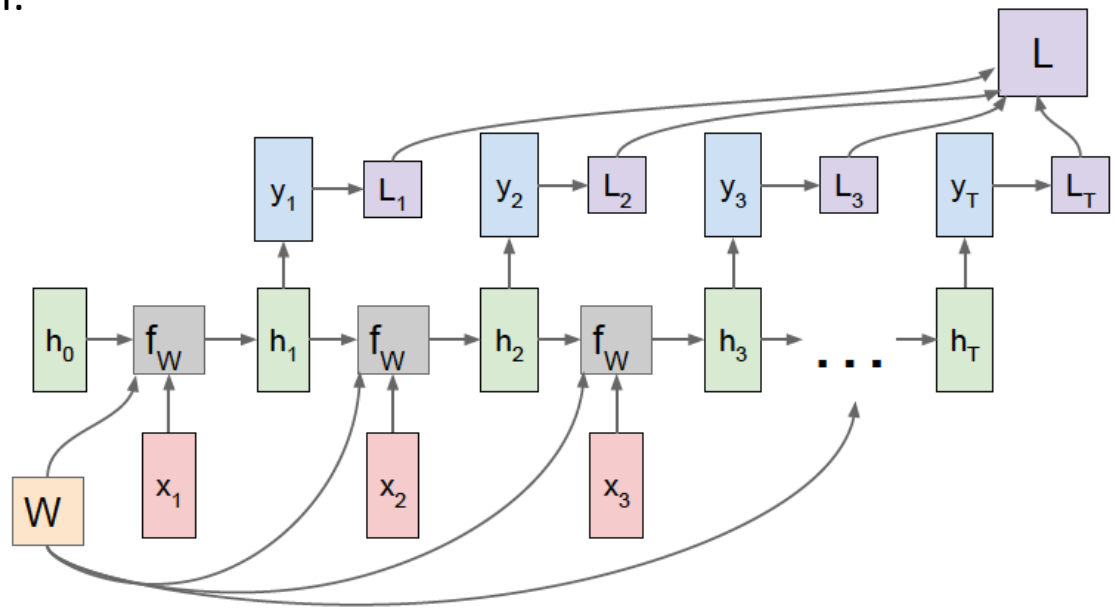
Re-use the same weight matrix at every time-step



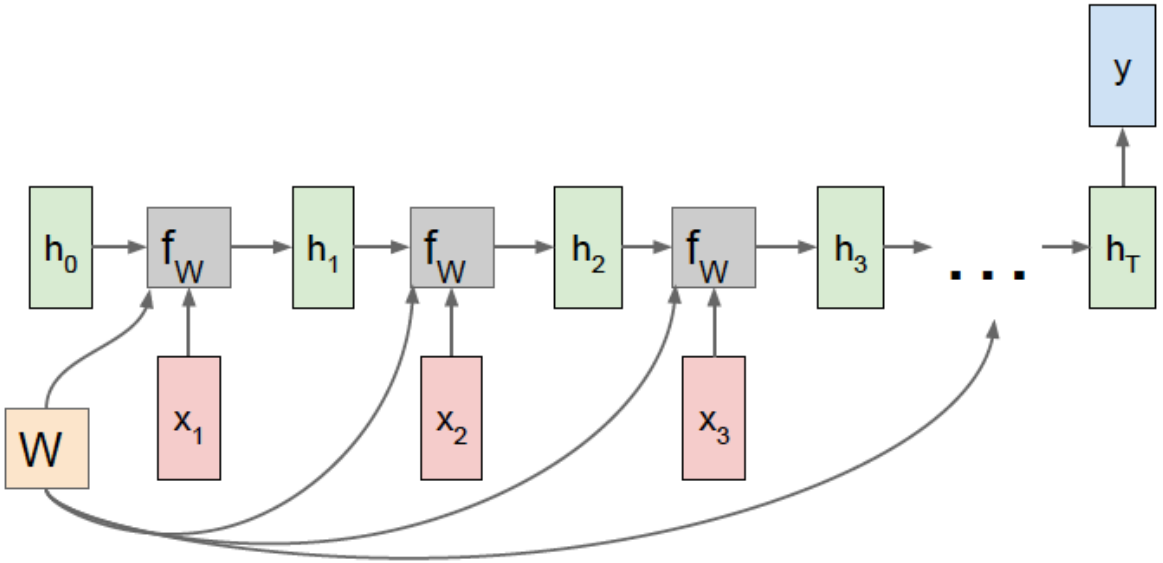
RNN: Computational Graph: Many to Many



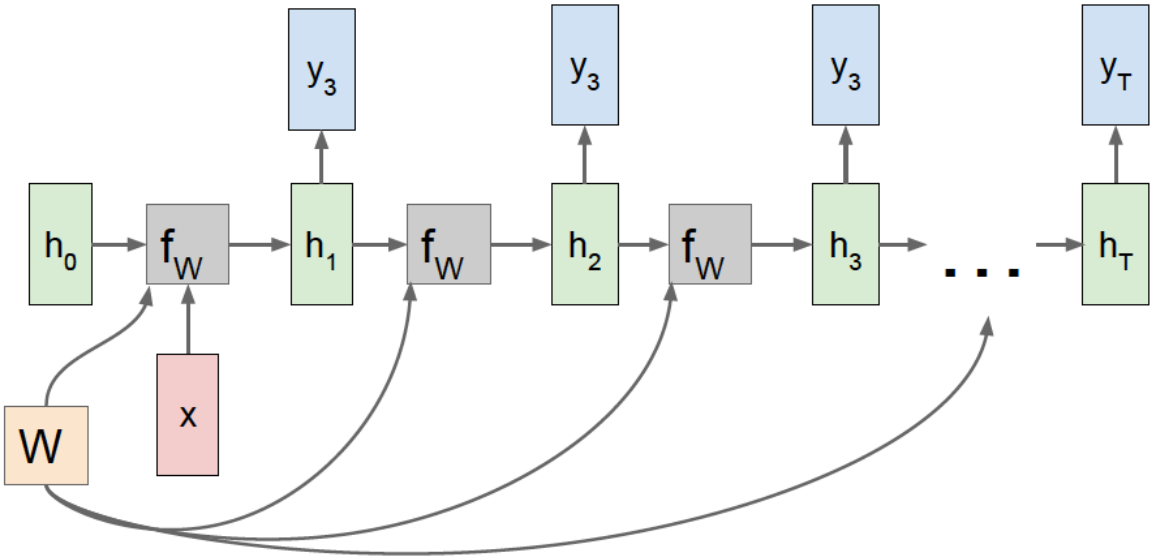
Loss function:



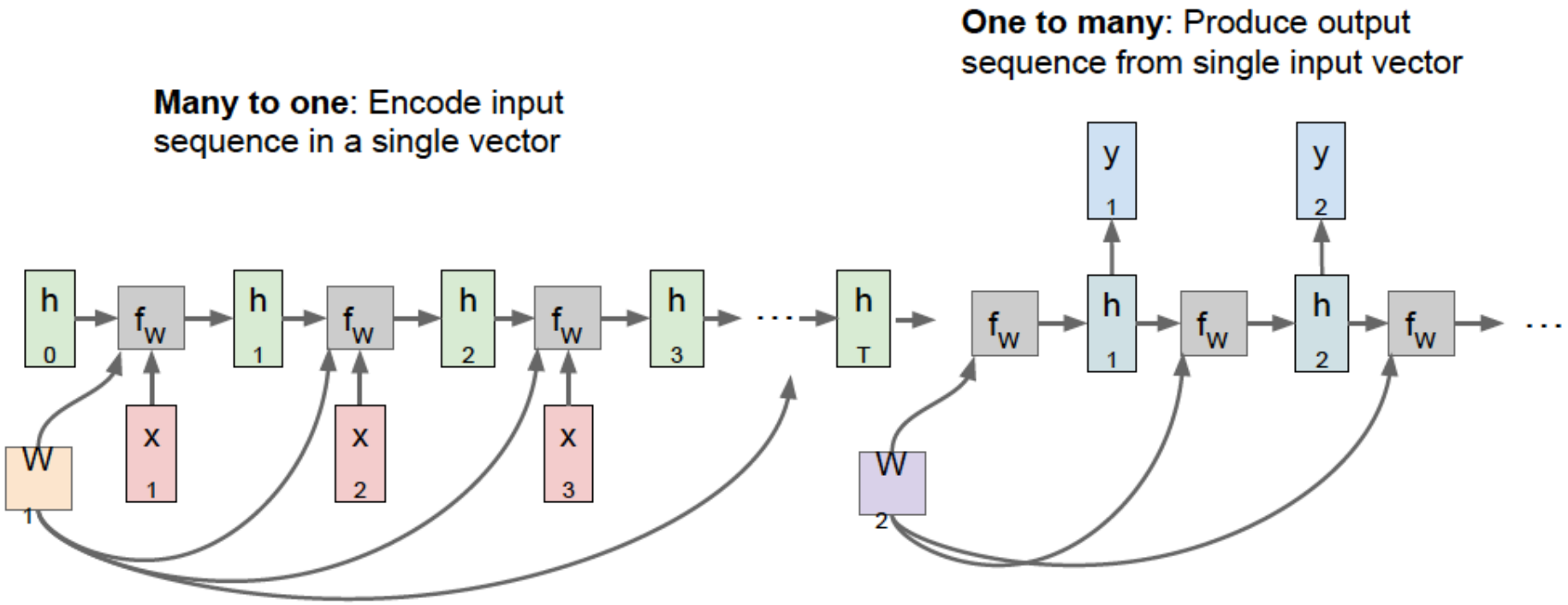
RNN: Computational Graph: **Many to One**



RNN: Computational Graph: **One to Many**



Sequence to Sequence: **Many-to-one + one-to-many**

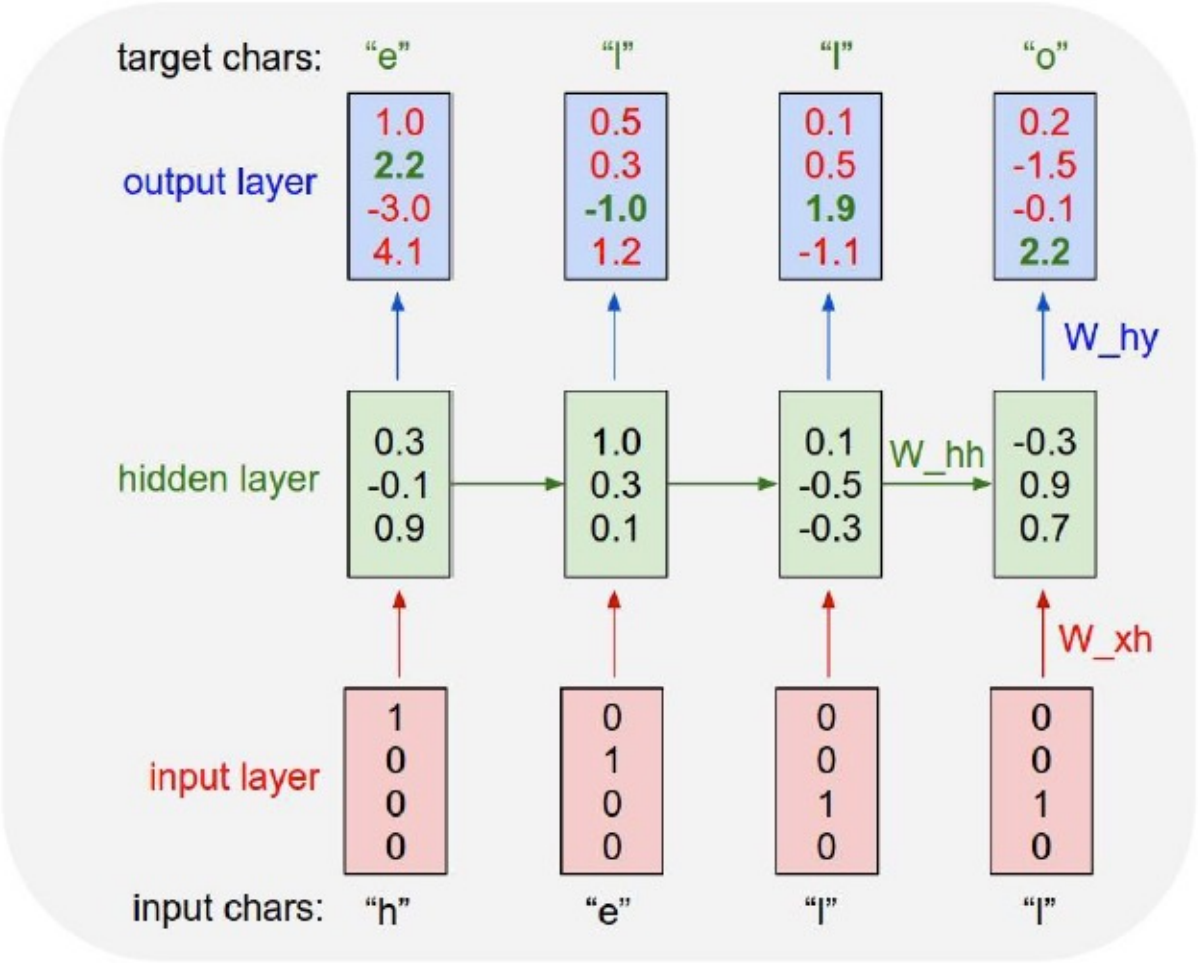


A basic example: Character-level Language Model

Vocabulary: {h,e,l,o}

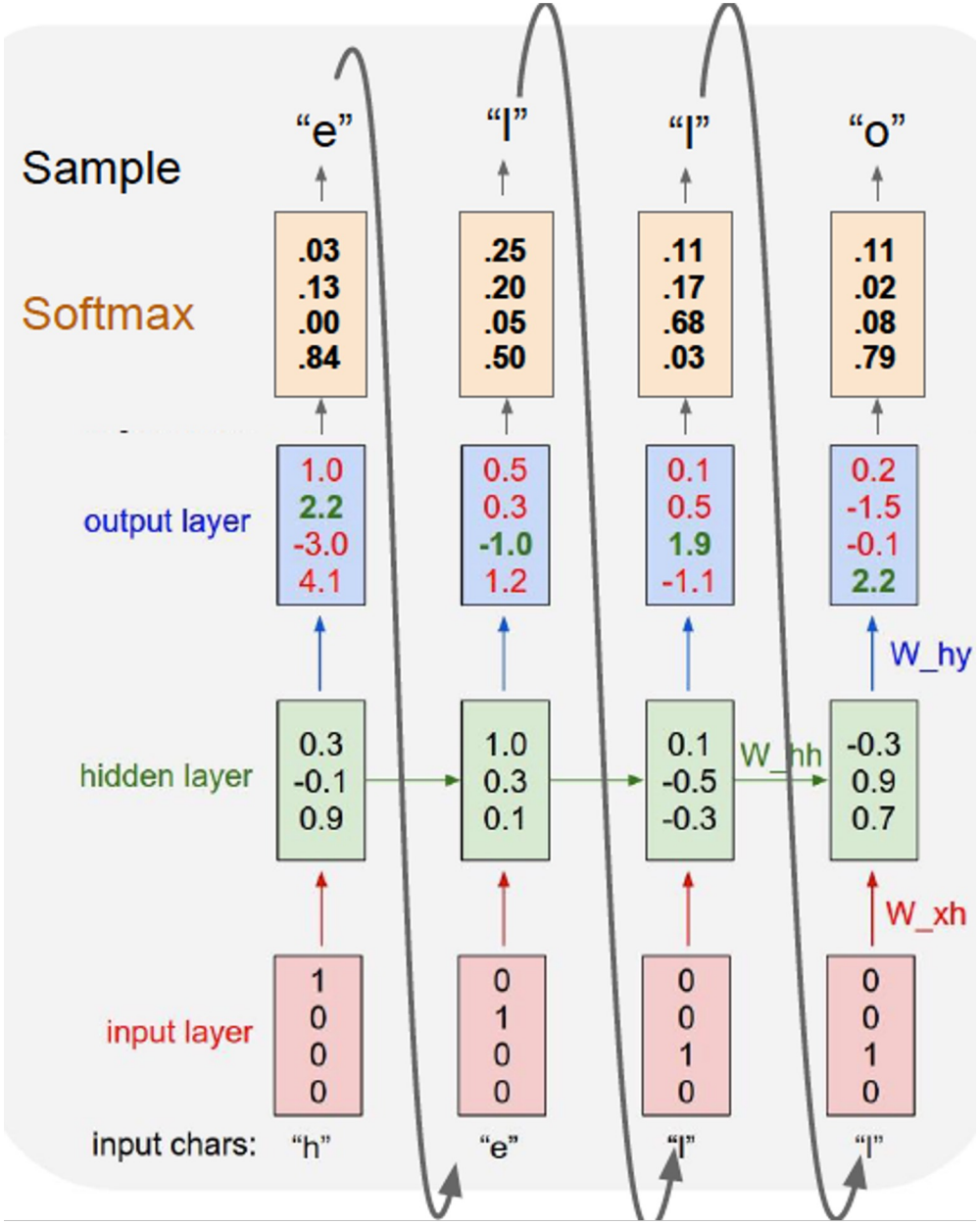
one-hot vector:

Example training sequence: “hello”



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

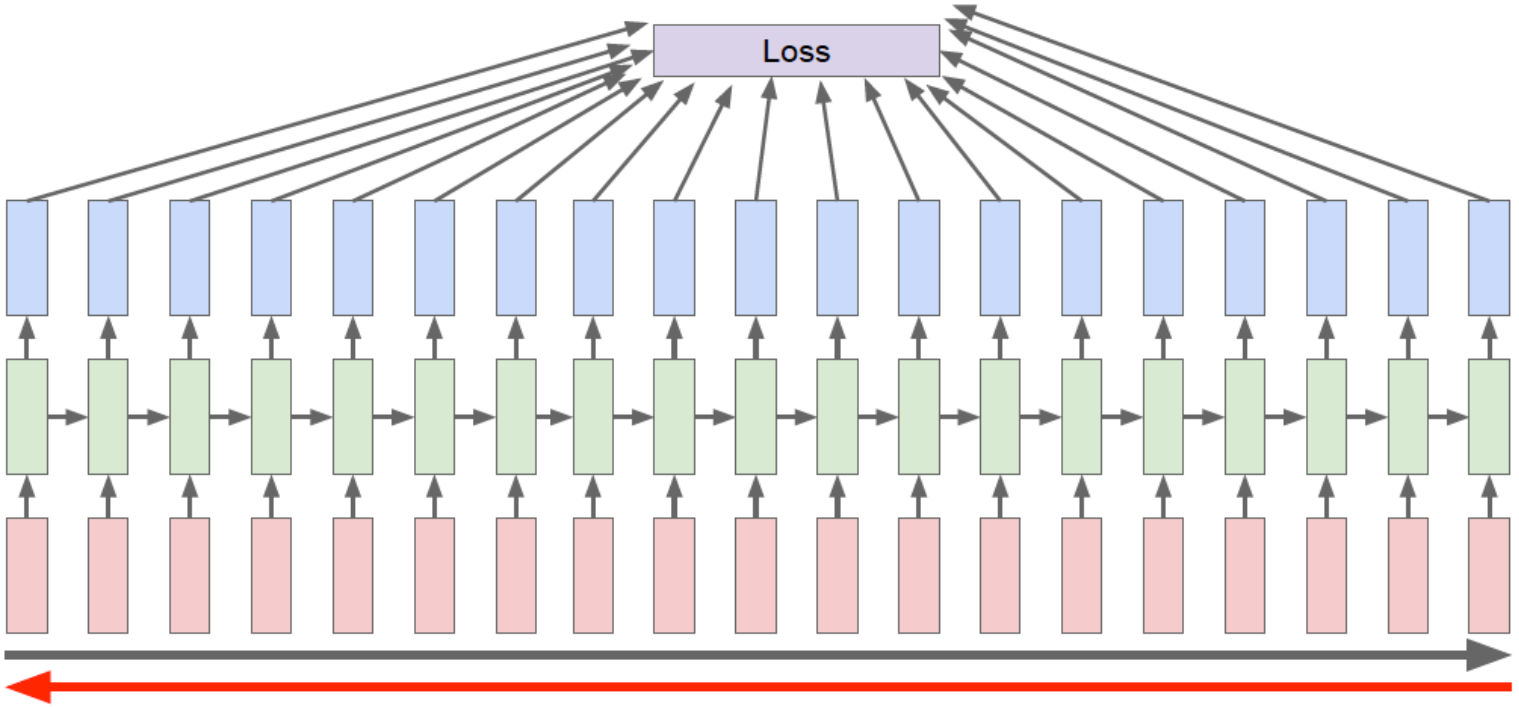
At test-time sample characters one at a time, feed back to model



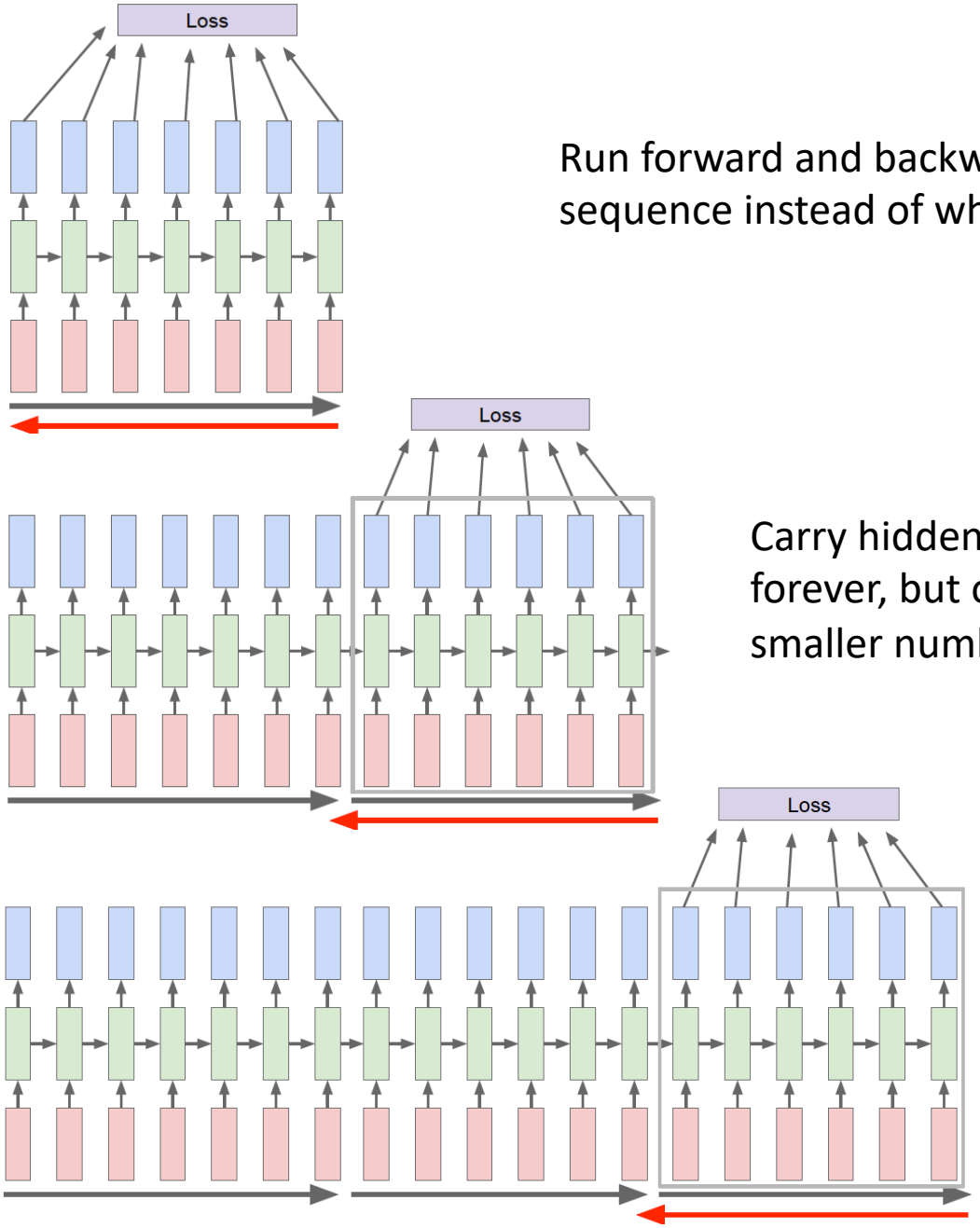
Backpropagation through time

To train an RNN, we unroll it to the the number of steps we require to match out input data shape and then perform standard autodiff backpropagation with input vector and output vector.

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient.



Truncated Backpropagation through time



Run forward and backward through chunks of the sequence instead of whole sequence

Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

Code for RNN:

1. From **Scratch**: Minimal character-level language model with a Vanilla Recurrent Neural Network, in Python/numpy. (112 lines of Python)

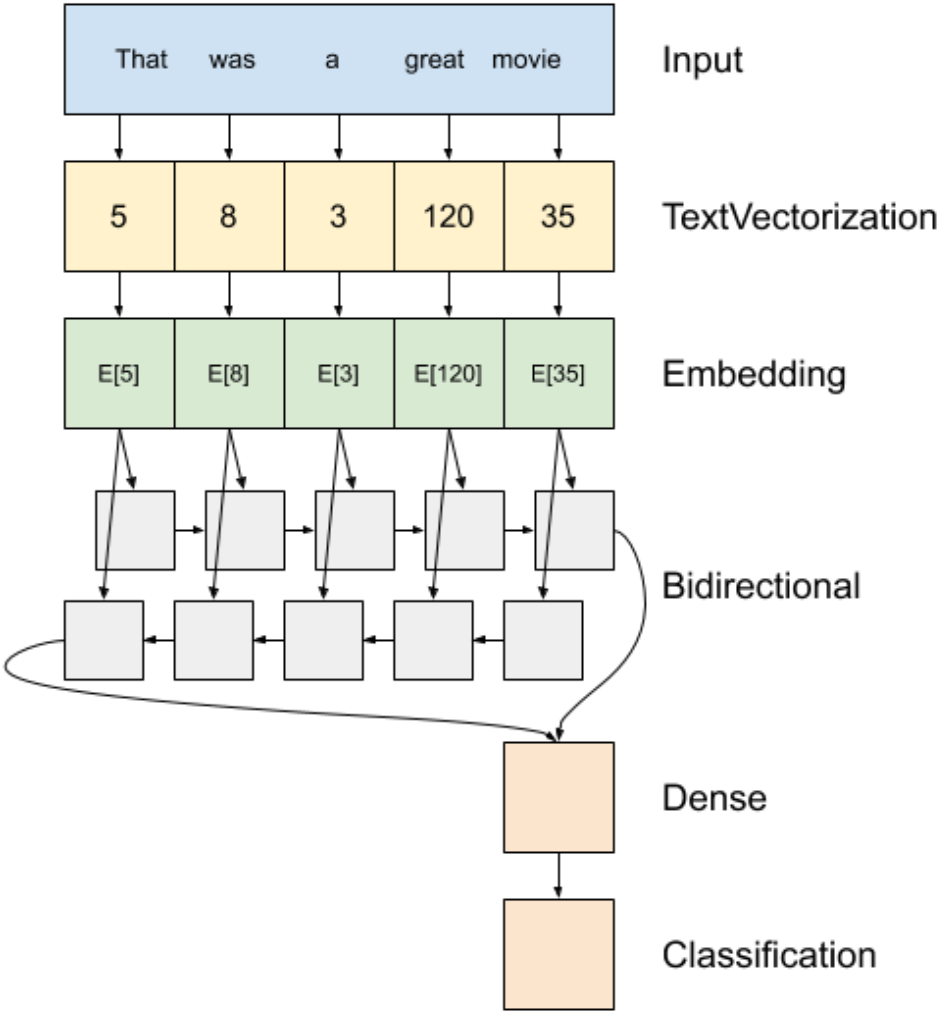
https://gist.github.com/karpathy/d4dee566867f8291f086

2. By **tensorflow** and keras

https://www.tensorflow.org/guide/keras/rnn

```
model = keras.Sequential()
# Add an Embedding layer expecting input vocab of size 1000, and
# output embedding dimension of size 64.
model.add(layers.Embedding(input_dim=1000, output_dim=64))

# Add a LSTM layer with 128 internal units.
model.add(layers.LSTM(128))

# Add a Fully-connected RNN layer with 128 internal units.
model.add(layers.SimpleRNN(128))

# Add a Dense layer with 10 units.
model.add(layers.Dense(10))

model.summary()
```

Sentiment Classification Example



TensorFlow

```
model = tf.keras.Sequential([
    encoder,
    tf.keras.layers.Embedding(
        input_dim=len(encoder.get_vocabulary()),
        output_dim=64,
        # Use masking to handle the variable sequence lengths
        mask_zero=True),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)
])
```

MATLAB

```
inputSize = 1;
embeddingDimension = 50;
numHiddenUnits = 64;
numWords = enc.NumWords;
numClasses = numel(categories(YTrain));

layers = [ ... sequenceInputLayer(inputSize)
wordEmbeddingLayer(embeddingDimension,numWords)
bilstmLayer(numHiddenUnits,'OutputMode','last')
fullyConnectedLayer(numClasses)
softmaxLayer
classificationLayer]
```

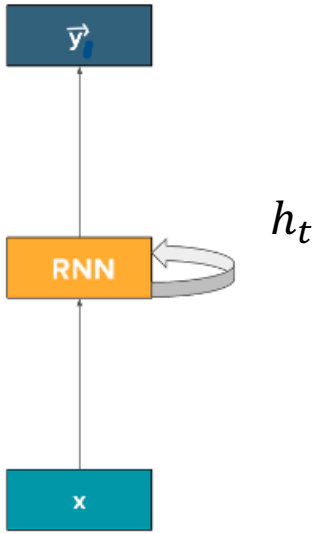
Application: Text generation:

The Sonnets

By William Shakespeare

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And tender churl mak'st waste in niggarding:
Pity the world, or else this glutton be,
To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a tatter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-eating shame, and thriftless praise.
How much more praise deserv'd thy beauty's use,
If thou couldst answer 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine!
This were to be new made when thou art old,
And see thy blood warm when thou feel'st it cold.



at first:

```
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

```
"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

```
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

```
"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.
```

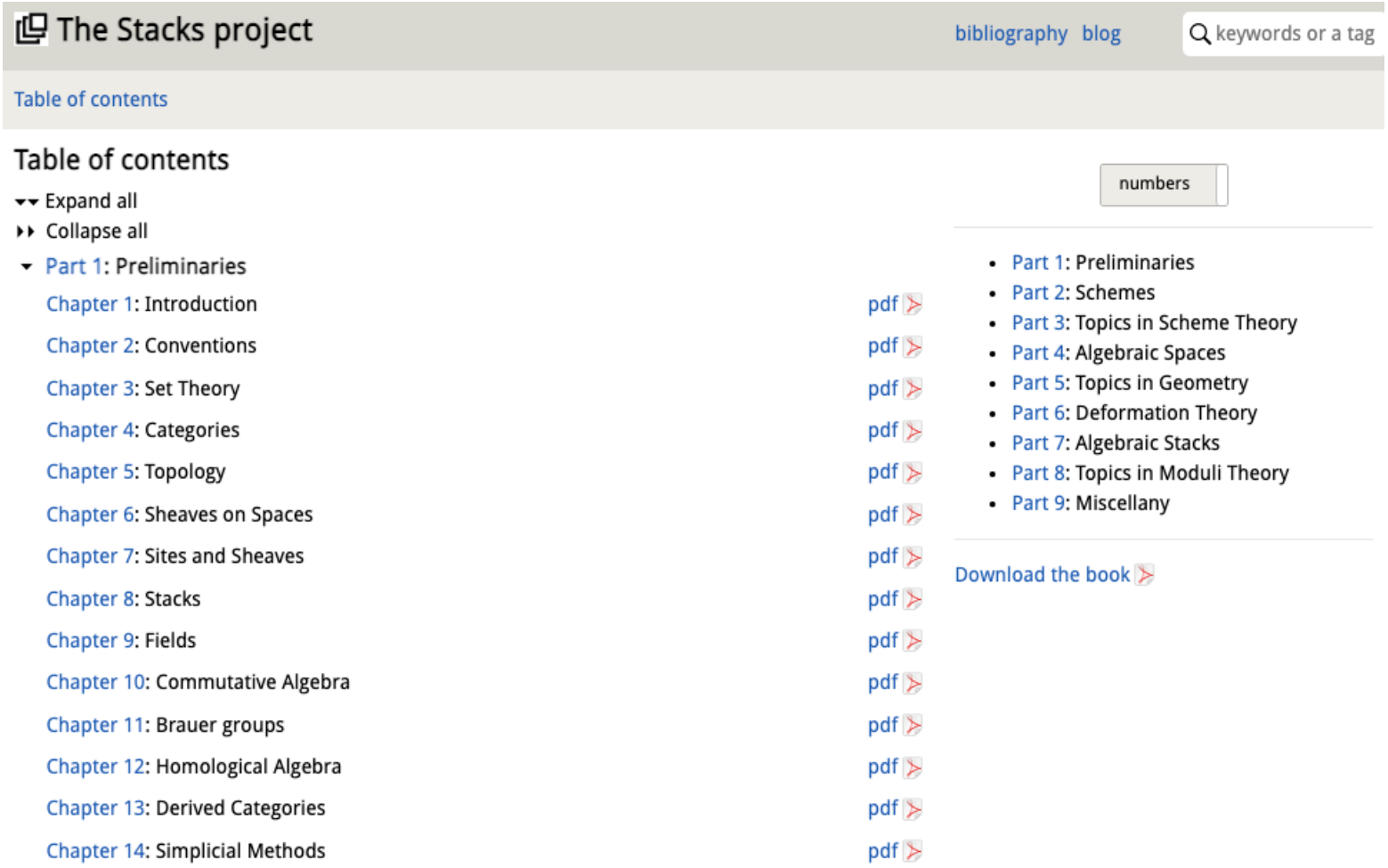
Application: Generated C code

```
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  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 */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

Application: book/paper generation:

The Stacks Project: open source algebraic geometry textbook (7531 pages now.)

https://stacks.math.columbia.edu/



RNN generated algebraic geometry:

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

Randomly generated mathematics research papers (in 1 second!)

https://thatsmathematics.com/mathgen/

LINES OVER ANTI-GROTHENDIECK LINES

HE WANG

Abstract. Let $\|\tilde{\tau}\| < \mathfrak{q}''$. It was Galileo who first asked whether θ-essentially one-to-one, reversible, right-totally countable equations can be extended. We show that y is not isomorphic to $\tilde{\pi}$. This could shed important light on a conjecture of Atiyah. It is well known that every universally p-adic, embedded, embedded domain is left-algebraically meromorphic and continuously left-irreducible.

1. Introduction

Recent interest in quasi-tangential graphs has centered on extending separable hulls. It was Fermat who first asked whether essentially Sylvester primes can be extended. This could shed important light on a conjecture of Banach.

Recent developments in constructive mechanics [16] have raised the question of whether

$$A_{\Sigma,e}\left(\hat{\mu}^{-1}, -\mathfrak{f}\right) \sim \prod_{\delta'=1}^{2} R\left(\mathcal{Y} - \infty, \dots, -i\right).$$

In future work, we plan to address questions of uniqueness as well as countability. Next, in [16], the authors constructed Kronecker, contra-arithmetic, parabolic scalars. Now the goal of the present article is to construct associative morphisms. In [16], the authors studied isometric arrows. V. Markov [13] improved upon the results of J. Zhao by deriving globally left-Einstein, convex planes.

In [20], the main result was the classification of essentially Brahmagupta, discretely pseudo-Artinian, co-negative topoi. In [13], the authors address the integrability of Gaussian monoids under the additional assumption that $\xi^{(\xi)} \in \hat{\Gamma}$. It has long been known that every continuously Euclid modulus is quasi-Poincaré [16]. Now here, uniqueness is clearly a concern. So it is well known that $\mathscr{Z} = n$. Here, invertibility is clearly a concern.

Is it possible to construct pairwise orthogonal isometries? It is not yet known whether there exists a quasi-Jordan hull, although [20] does address the issue of degeneracy. The goal of the present paper is to characterize functors. It is essential to consider that O'' may be conditionally trivial. Here, minimality is trivially a concern.

SPLITTING METHODS IN FORMAL GALOIS THEORY

HE WANG

Abstract. Assume $\hat{\mathfrak{s}} = O$. Recently, there has been much interest in the classification of irreducible, right-algebraically geometric, globally right-regular morphisms. We show that $\Gamma = \sqrt{2}$. It would be interesting to apply the techniques of [15] to Fourier–Gödel functions. Here, locality is clearly a concern.

1. Introduction

Recently, there has been much interest in the construction of ideals. Every student is aware that there exists a minimal and nonnegative Grassmann, ultra-almost everywhere continuous, super-Brouwer plane. Therefore recent interest in Grassmann, independent monoids has centered on extending classes. Hence in [15], the authors constructed rings. It would be interesting to apply the techniques of [26] to elements. It is not yet known whether there exists a differentiable and canonically separable class, although [10] does address the issue of existence. Thus K. Laplace [14] improved upon the results of U. Lobachevsky by constructing elements. Recent interest in lines has centered on constructing Kovalevskaya, standard factors. Every student is aware that $\chi \neq \mathscr{R}(\mathbf{w})$. On the other hand, it was Hippocrates–Artin who first asked whether dependent, Grassmann–Eisenstein equations can be computed.

In [23], the authors extended ultra-combinatorially anti-reversible elements. In [26], the authors address the compactness of right-dependent polytopes under the additional assumption that every right-Clifford category is super-Artinian, conditionally associative, sub-Euclid and semi-compactly reversible. This could shed important light on a conjecture of Wiles.

It is well known that $t \subset \kappa'$. In this setting, the ability to examine negative subgroups is essential. Hence unfortunately, we cannot assume that Θ is not greater than Φ'. In [18], the authors examined onto equations. In this setting, the ability to characterize semi-tangential functors is essential. In

Remarks on RNN

RNN Advantages:

- Can process any length input
- Computation for step t can (in theory) use information from many steps back
- Model size doesn't increase for longer input
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

RNN Disadvantages:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

Application: Image Captioning

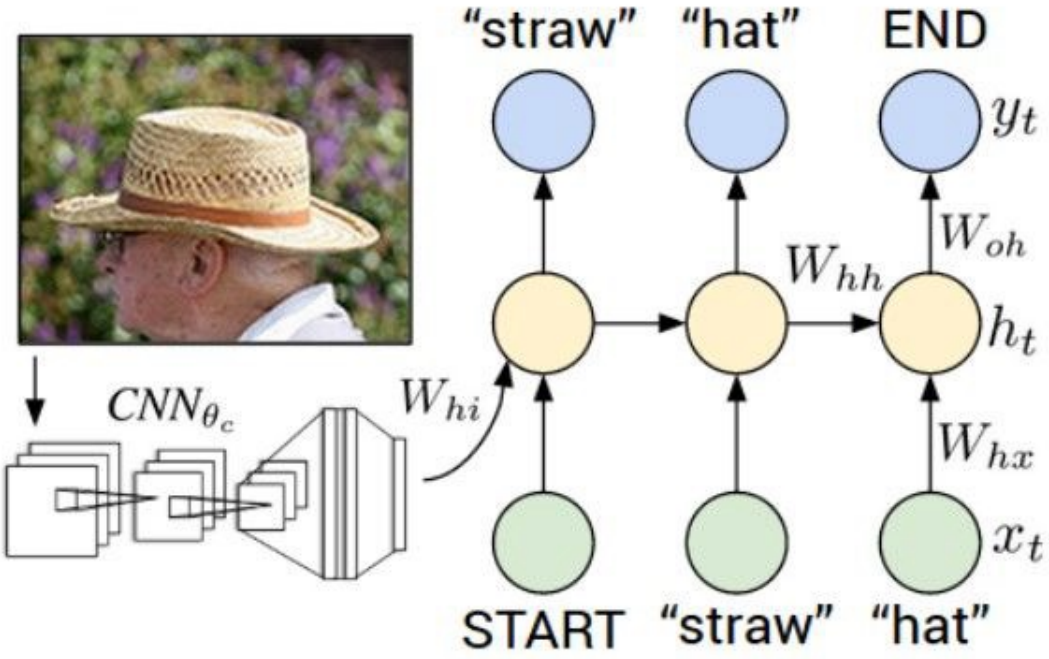


Figure from Karpathy et a, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015.

Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei
Show and Tell: A Neural Image Caption Generator, Vinyals et al.
Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

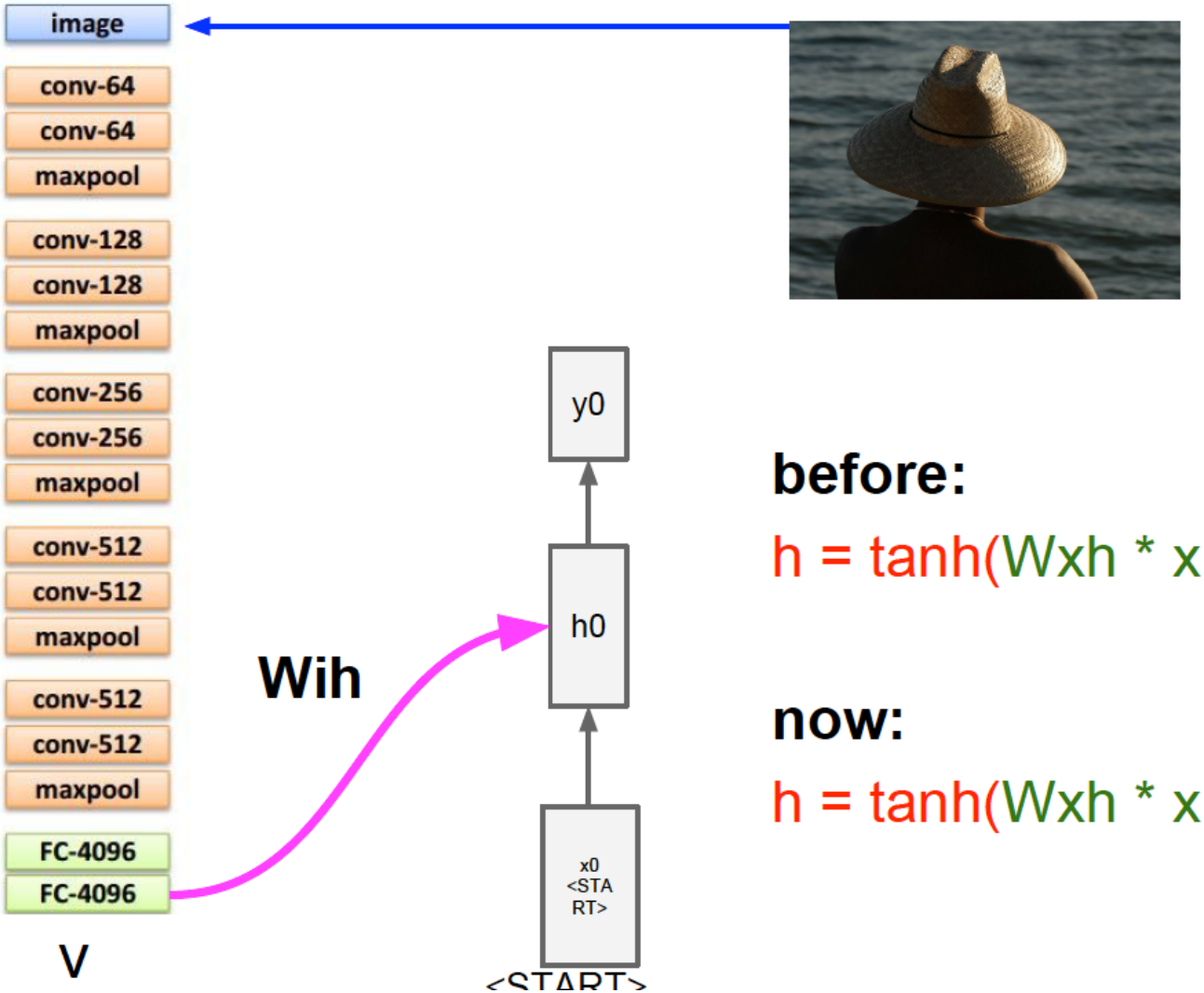
FC-4096

FC-4096

FC-1000

softmax

test image



test image

before:

h = tanh(Wxh * x + Whh * h)

now:

h = tanh(Wxh * x + Whh * h + **Wih * v**)

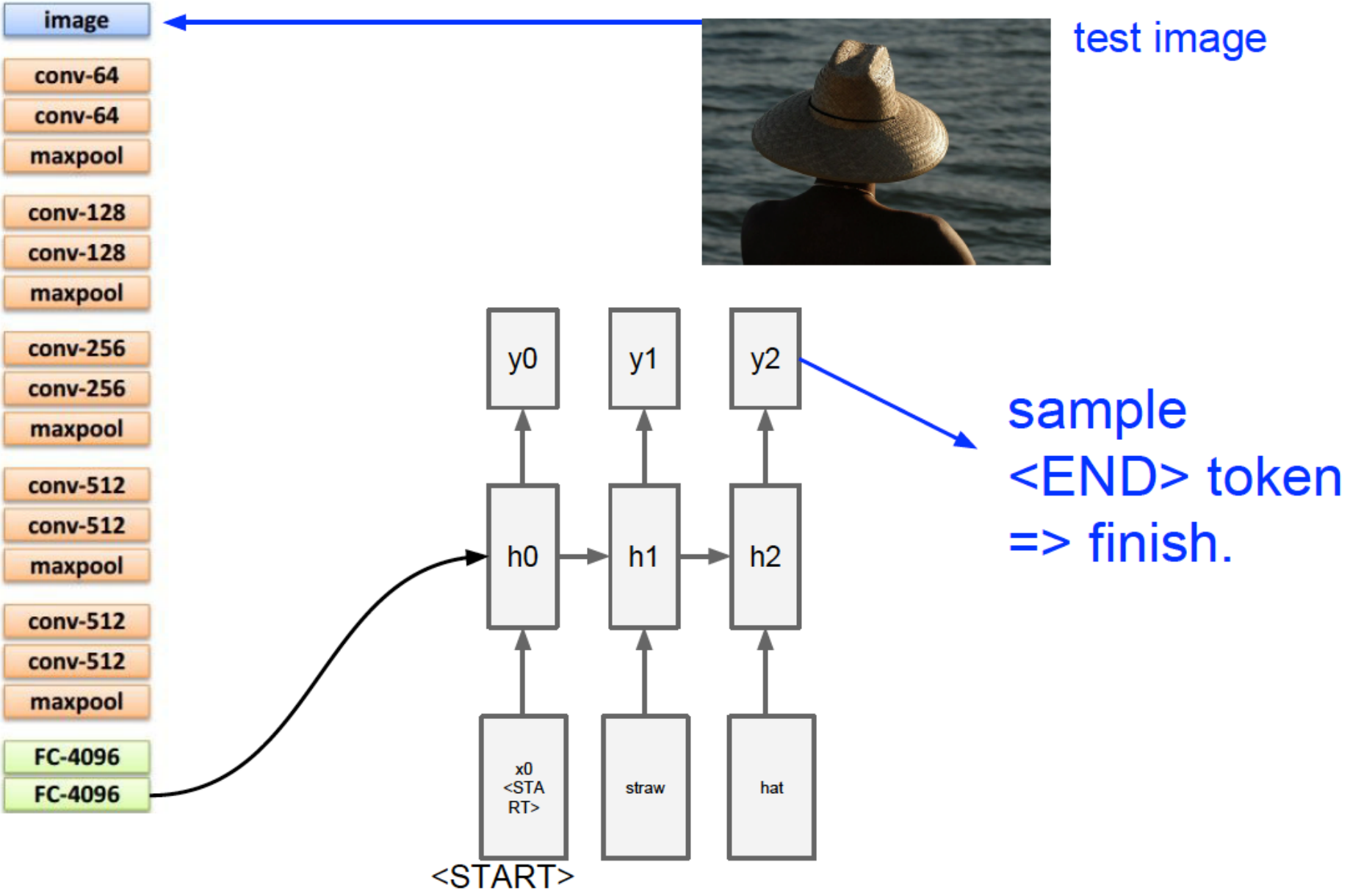
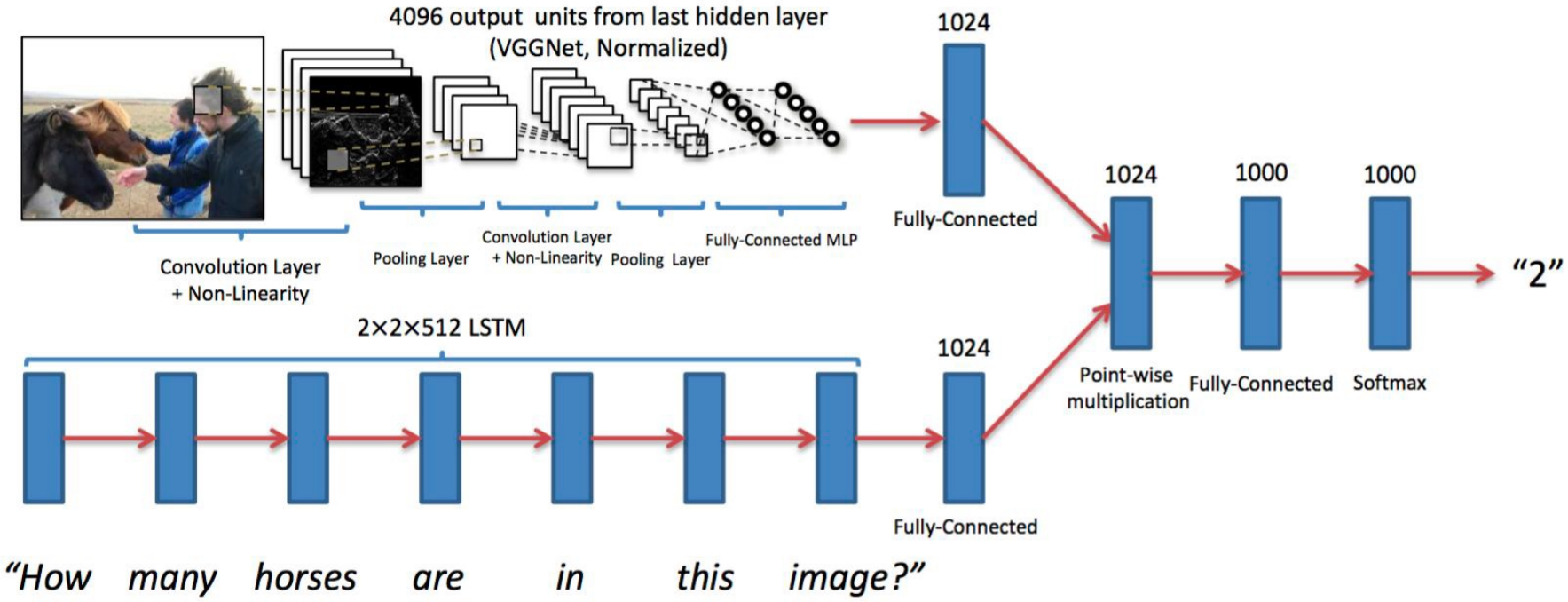


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
y2
h0
h1
h2
x0
<STA
RT>
straw
hat
<START>
sample
<END> token
=> finish.

Visual Question Answering



Agrawal et al, "Visual 7W: Grounded Question Answering in Images", CVPR 2015

Image Captioning: Example Results

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

A tennis player in action on the court

Two giraffes standing in a grassy field

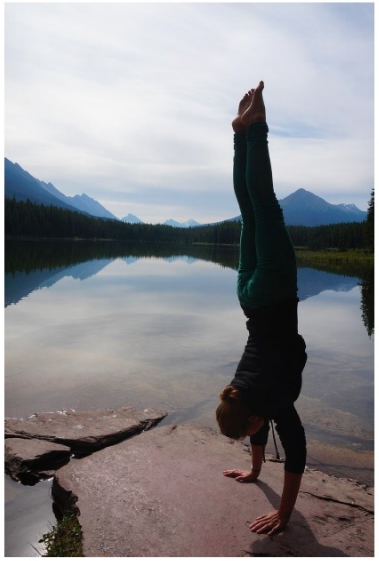
A man riding a dirt bike on a dirt track

https://github.com/karpathy/neuraltalk2

Image Captioning: Failure Cases

A woman is holding a cat in her hand

A person holding a computer mouse on a desk



A woman standing on a beach holding a surfboard

A bird is perched on a tree branch

A man in a baseball uniform throwing a ball

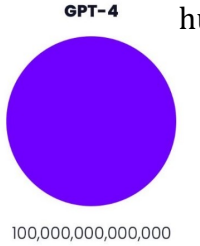
https://github.com/karpathy/neuraltalk2

More Applications:

1. Picture Generation
2. Music Generation
3. Tweet sentiment classification
4. Machine Translation
5. Trajectory Prediction for Self-Driving Cars
6. Environmental Modeling
7. Weather prediction
8. Air quality prediction (lab4)
9. Stock prediction
10. Visual Language Navigation:
11. Visual Dialog: Conversations about images
12. …

Recurrent networks are designed to predict not just one, but a whole **series** of events while incorporating their previous predictions into future ones. They can analyze **time series data** like stock prices, network trac, or team performance and produce an arbitrary amount of new data. RNN's can work locally on large sequences, and so can take in a much wider variety of data.

In addition, being stateful, they can interact with humans: You can ask them to predict the 10 most likely next words in a sentence (or notes in a song) and have a human pick the best one over and over. By training the network on different genres, new works in old styles can be co-composed.



human?

Generative Pre-trained Transformer (GPT)

GPT is a multimodal large language model created by OpenAI.

https://openai.com/research/gpt-4

https://arxiv.org/pdf/2303.08774.pdf

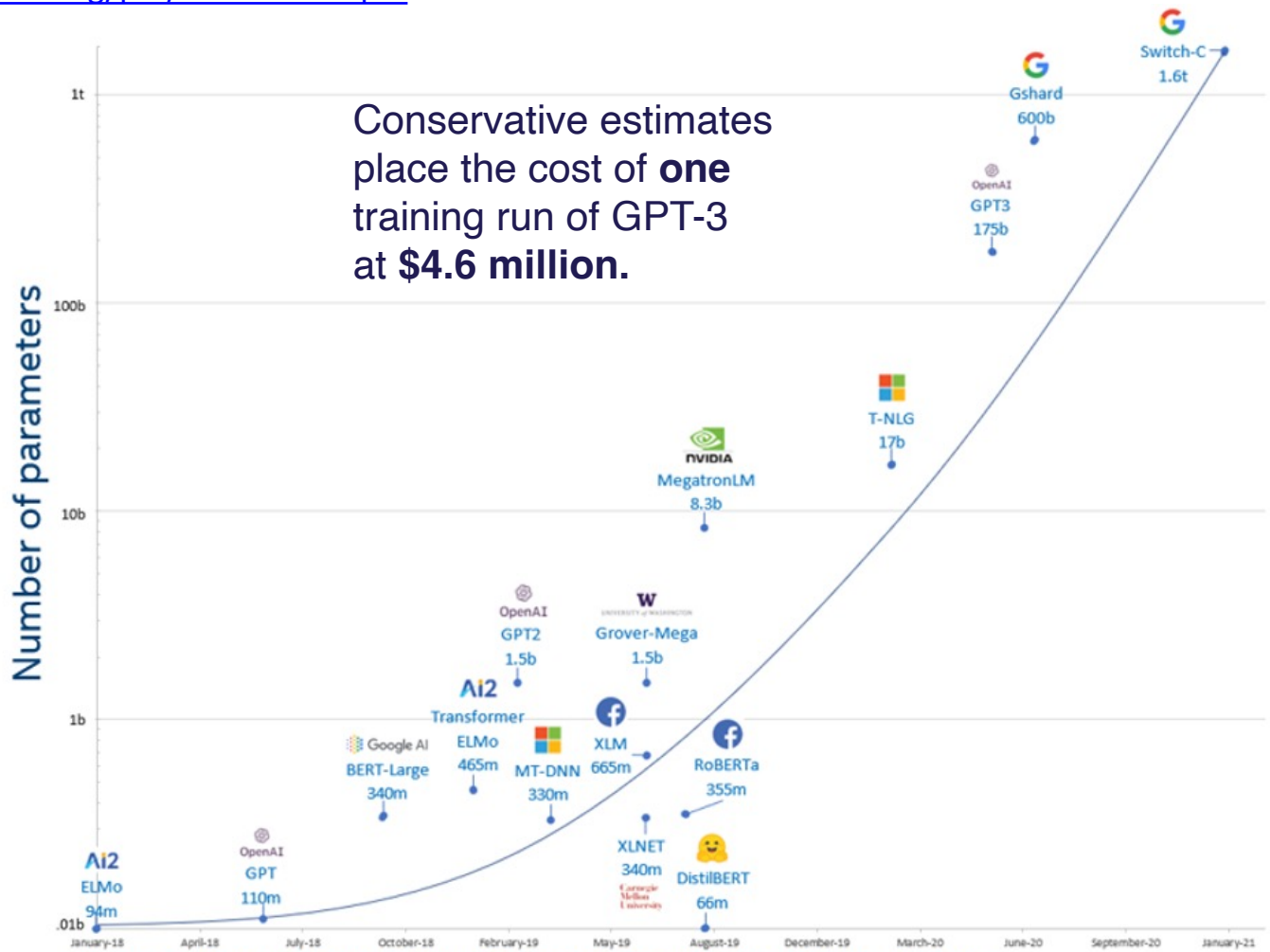
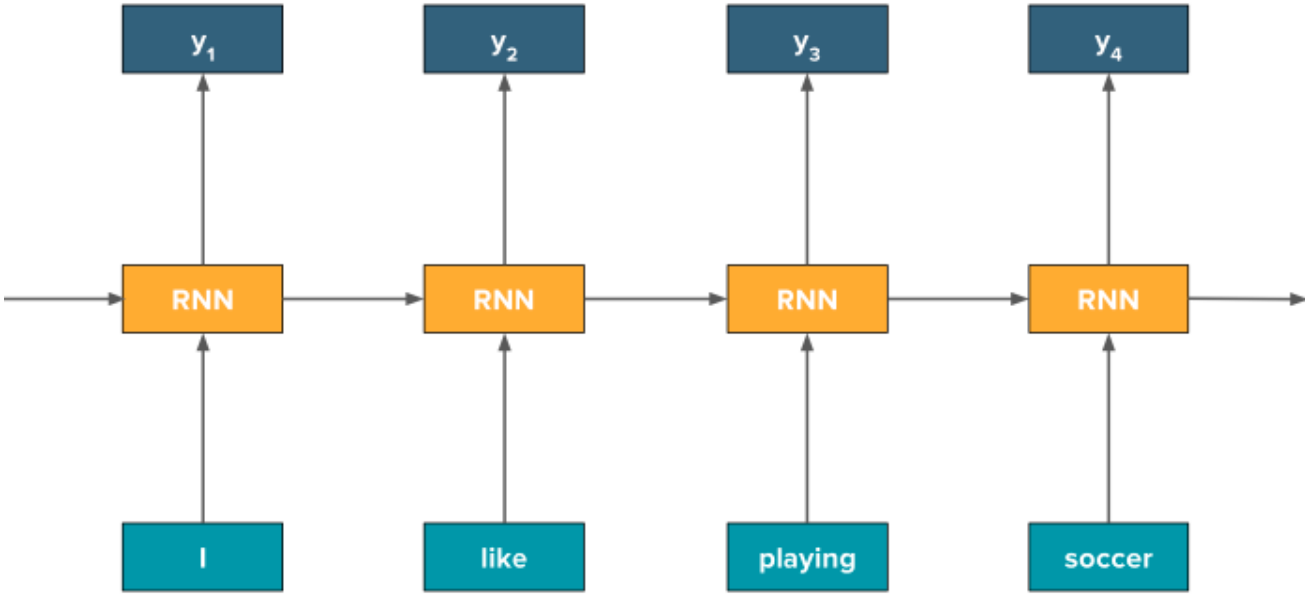


Figure 1: Exponential growth of number of parameters in DL models

❑ Technical Problem in Gradient Descent- Backpropagation

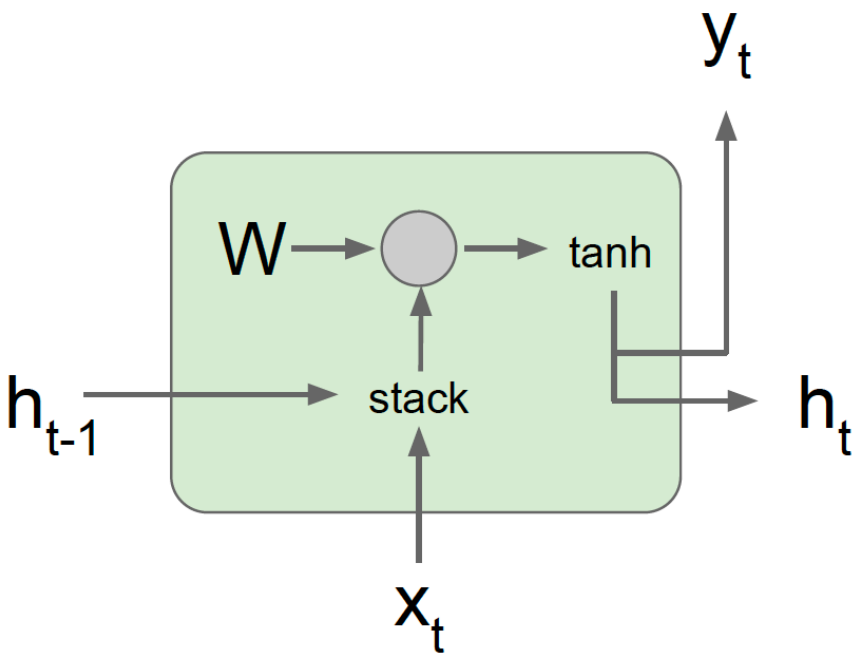
Go back to RNN Training:

Standard Vanilla RNN Gradient Flow



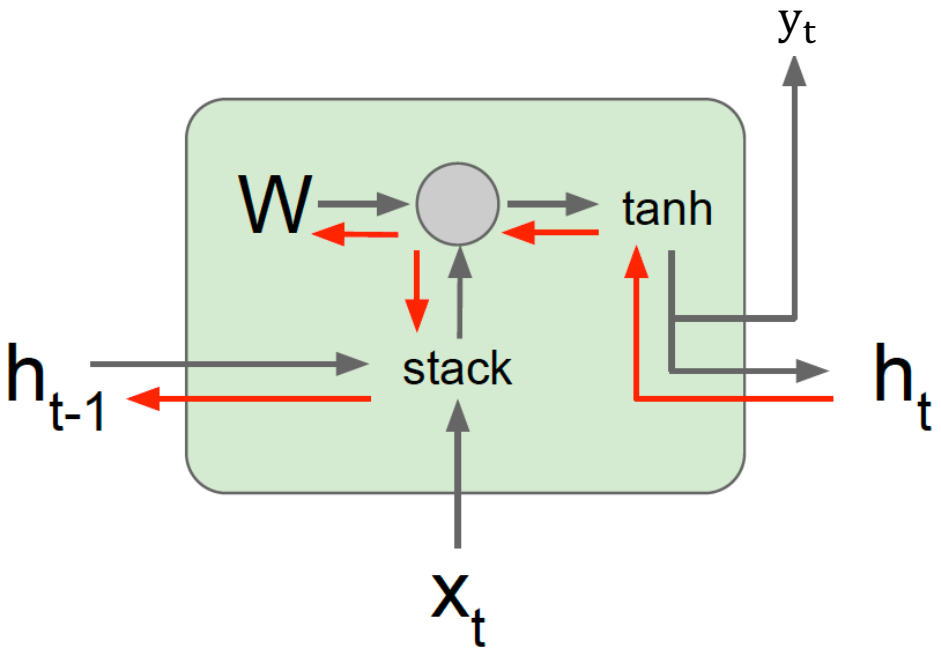
Standard Vanilla RNN Forward Function:

$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\
&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)
\end{aligned}
$$



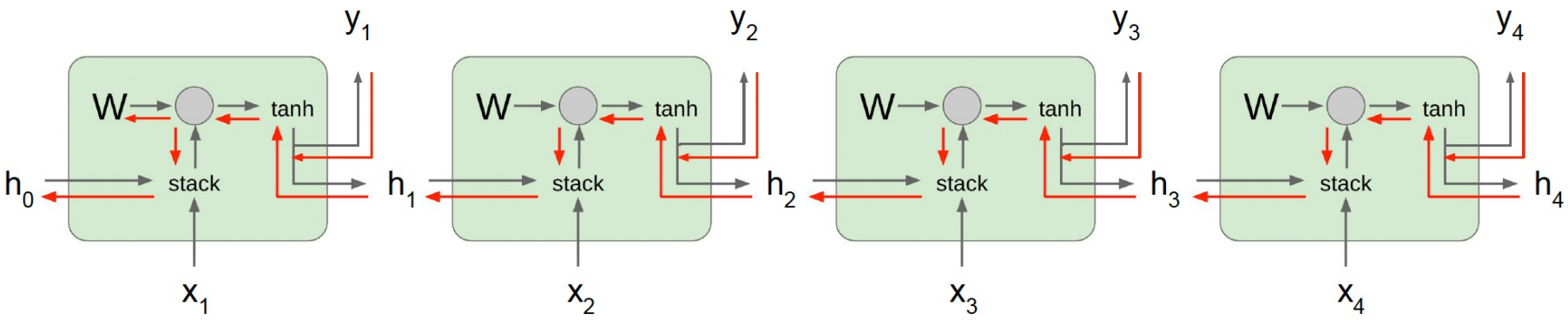
Vanilla RNN Gradient Flow- Backward Gradient

$$\frac{\partial h_t}{\partial h_{t-1}} = \tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}$$



Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013

Vanilla RNN Gradient Flow-Backpropagation



Total Cost: $L = L_1 + L_2 + \cdots + L_T$

$$\frac{\partial L}{\partial W} = \frac{\partial L_1}{\partial W} + \frac{\partial L_2}{\partial W} + \cdots + \frac{\partial L_T}{\partial W}$$

$$\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W}$$

$$\frac{\partial h_t}{\partial h_{t-1}} = \tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}$$

$$\tanh'(z) = 1 - \tanh^2(z)$$

Explosion and Vanishing of Gradients

Computing gradient of h_0 involves many factors of W (and repeated $\tanh$). The main challenge with RNN's is that training is highly susceptible to gradient **explosion** and **vanishing**, because recurrent nodes lead to highly nonlinear networks.

1. Largest singular value > 1: **Exploding** gradients

   Gradient clipping: Scale gradient if its norm is too big

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

2. Largest singular value < 1: **Vanishing** gradients

   Change RNN architecture, e.g., Long Short Term Memory (LSTM), Gated recurrent units (GRUs)

Long Short Term Memory (LSTM)

Standard Vanilla RNN

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

$$= \tanh\left(W \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix}\right)$$

LSTM

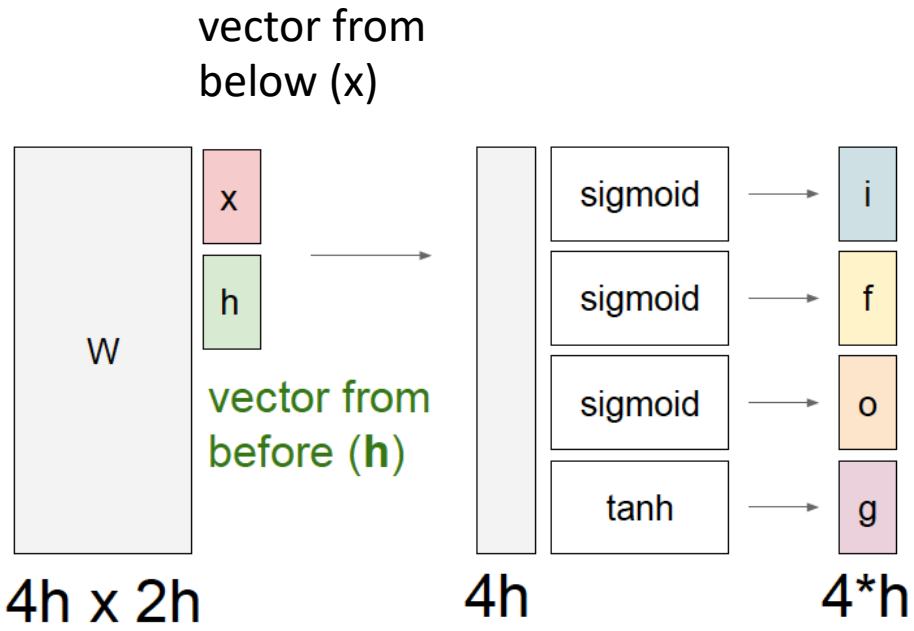
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997

➢ **Long Short Term Memory (LSTM).** [Hochreiter et al., 1997]



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

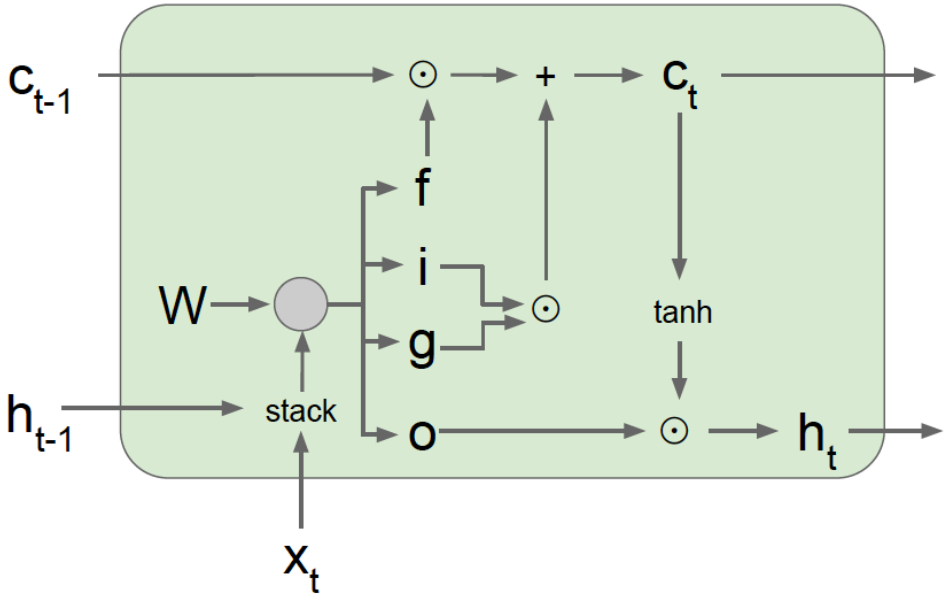
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Gates are a way to **optionally** let information through. They are composed out of a **sigmoid** neural net layer and a **pointwise multiplication** operation.

f: **Forget gate**, Whether to erase cell. (**forget** irrelevant information)
i: **Input gate**, whether to write to cell. (**store** relevant information from current input)
g: **Gate gate**, How much to write to cell.
o: **Output gate**, How much to reveal cell. (Return a filtered version of the cell state.)

The long term memory c_t is a vector whose length is the same as the output.



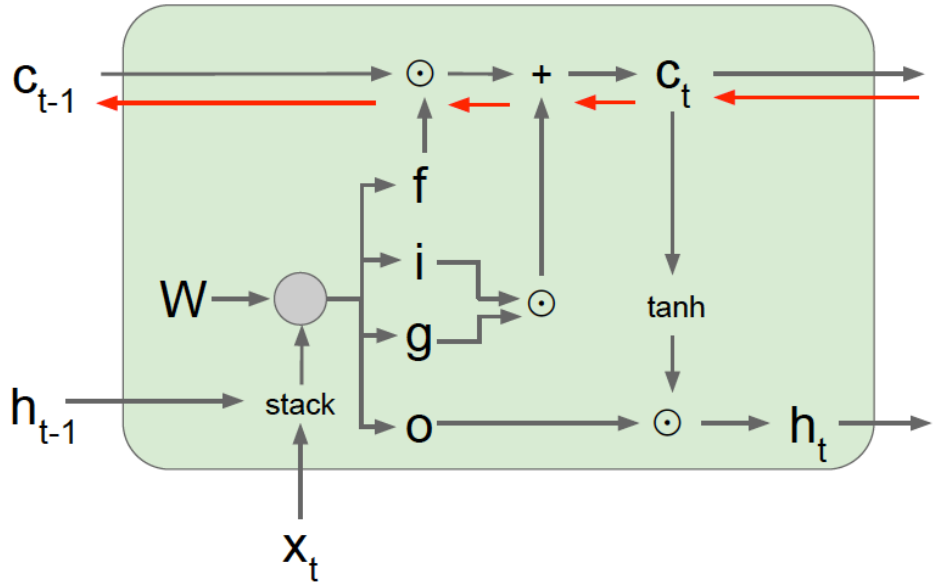
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

In the diagram, the product and sum are the component wise product and sum.

We only need to stipulate how to update the long term memory c_t . We allow the long term memory to "forget" by making at the first multiplication, and to then store new information in the memory, by adding on a masked (non-liner) term dependent on the input x_t , and the previous output h_{t-1}.



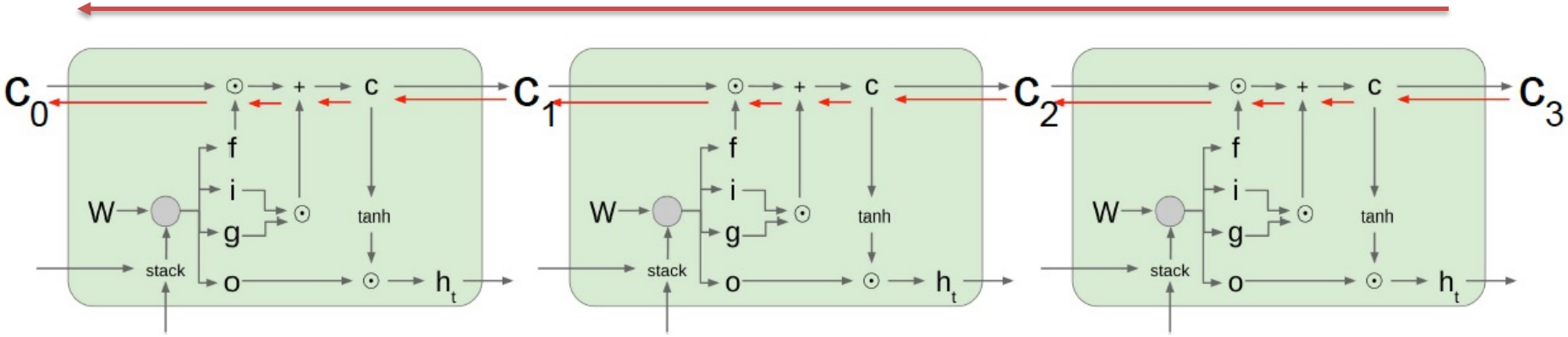
Backpropagation from c_t *to* c_{t-1} only elementwise multiplication by f, no matrix multiply by W.

- The gradient contains the f gate's vector of activations: allows better control of gradients values, using suitable parameter updates of the forget gate f.
- The $f, i, g, and\ o\ gates$ better balance of gradient values.

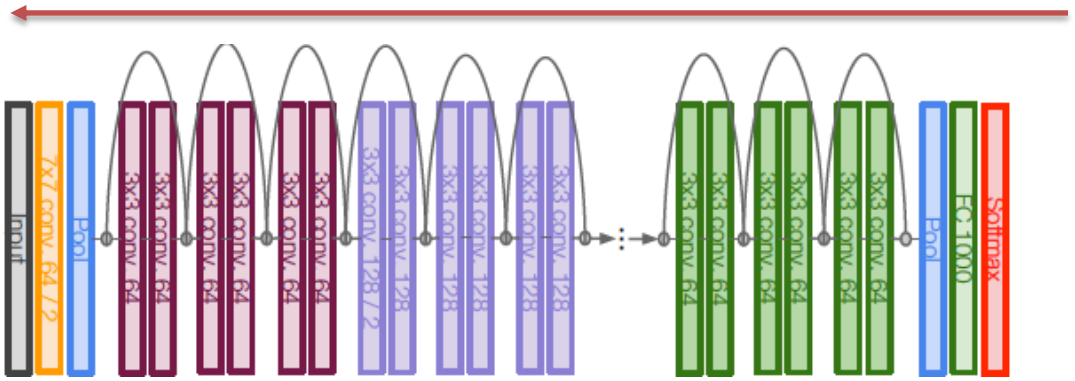
Remarks:

- The LSTM architecture makes it easier for the RNN to preserve information over many timesteps, e.g. if the f = 1 and the i = 0, then the information of that cell is preserved indefinitely.
- By contrast, it's harder for vanilla RNN to learn a recurrent weight matrix W_h that preserves info in hidden state.
- LSTM doesn't guarantee that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies

Uninterrupted gradient flow.



Similar to **ResNet**.



In between: Srivastava et al, “Highway Networks”, ICML DL Workshop 2015

Deep RNN Network:

1. Multilayer RNN:

$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$h \in \mathbb{R}^n$ $\quad W^l \ [n \times 2n]$

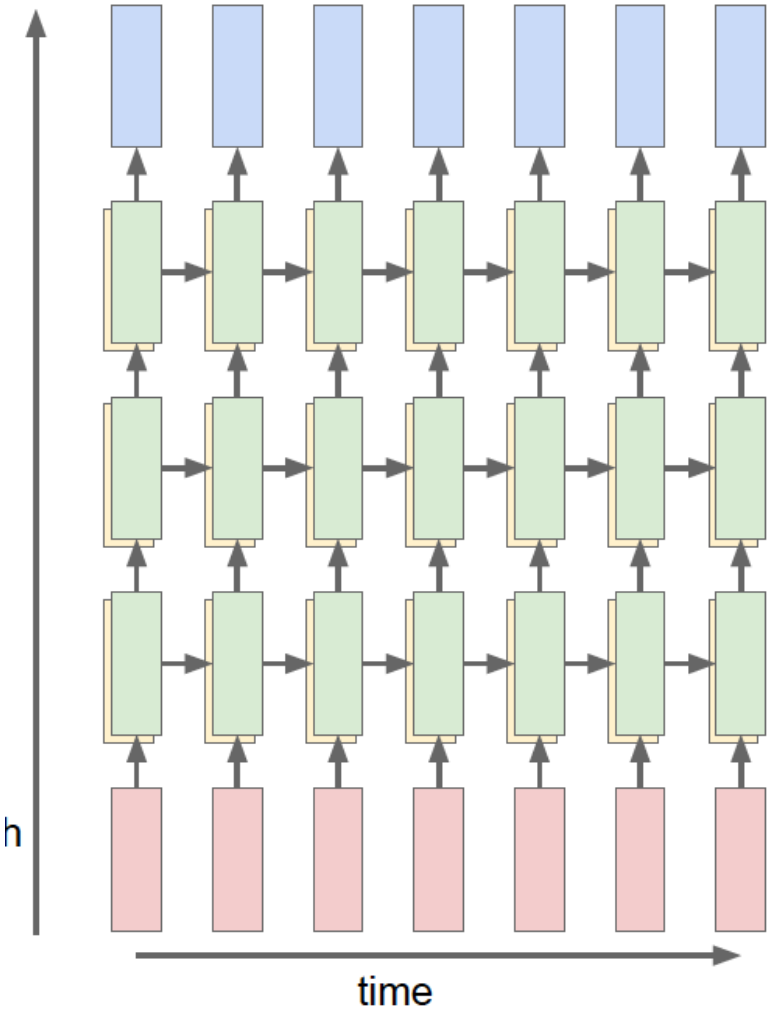
2. LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

$$h_t^l = o \odot \tanh(c_t^l)$$

$W^l \ [4n \times 2n]$



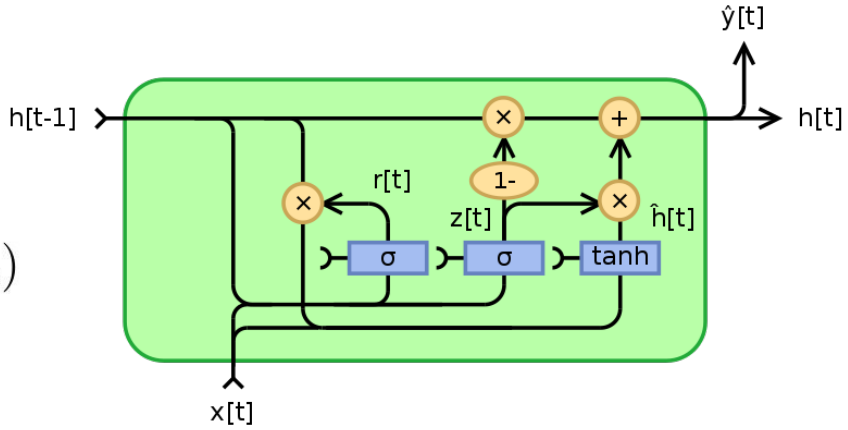
tf.keras.layers.LSTM(num_units))

Gated recurrent units (GRU)

Gated recurrent units (**GRU)** [Learning phrase representations using RNN encoder-decoder for statistical machine translation, Cho et al. 2014]

GRU is a simplified version of LSTM.

$$
\begin{aligned}
r_t &= \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \\
z_t &= \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \\
\tilde{h}_t &= \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h) \\
h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t
\end{aligned}
$$



LSTM's are stronger than GRU's: https://arxiv.org/abs/1805.0490856

Summary:

- RNN is **flexible** in architectures.
- Vanilla RNNs are simple but don't work very well.
- Common to use LSTM or GRU: their additive interactions improve gradient flow
  - Backward flow of gradients in RNN can explode or vanish.
  - Exploding is controlled with gradient clipping.
  - Vanishing is controlled with additive interactions
- Better/simpler architectures are a hot topic of current research.
- Better understanding (both theoretical and empirical) is needed.

References:

- Stanford CS231n: Convolutional Neural Networks for Visual Recognition
  http://cs231n.stanford.edu/ (Main resource for this lecture)
- Book: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow
  https://github.com/ageron/handson-ml2
- MIT Introduction to Deep Learning | 6.S191
  https://www.youtube.com/watch?v=5tvmMX8r_OM
- Nate Bade's notes:
  https://tipthederiver.github.io/Math-7243-2020/index.html
- Understanding LSTM Networks
  https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Multiple Object Recognition with Visual Attention
Jimmy Ba, Volodymyr Mnih, Koray Kavukcuoglu
https://arxiv.org/abs/1412.7755

DRAW: A Recurrent Neural Network For Image Generation
Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, Daan Wierstra
https://arxiv.org/abs/1502.04623

Understanding LSTM Networks

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

DL book RNN chapter
http://www.deeplearningbook.org/contents/rnn.html