CS 760: Machine Learning
Recurrent Neural Networks

Misha Khodak

University of Wisconsin-Madison

15 October 2025

Logistics

- Midterm: 75 min in-class October 22nd
  - covers material through October 15th, focusing on everything **before** neural networks
  - mix of short answer and derivations
  - one double-sided 8.5x11 cheat sheet
  - no calculators
  - review in-class on October 20th (slides posted)

- Homework 3 posted after class, due in **three** weeks

- Monday lecture will end 5-10 min early and Monday office hours cancelled that day (will still hold them on Tuesday)

Outline

- **RNN basics**
  - sequential tasks, hidden state, vanilla RNN
- **RNN variants + LSTMs**
  - RNN training, variants, LSTM cells
- **Practical training**
  - data pipelines, initialization, hyperparameter tuning

Outline

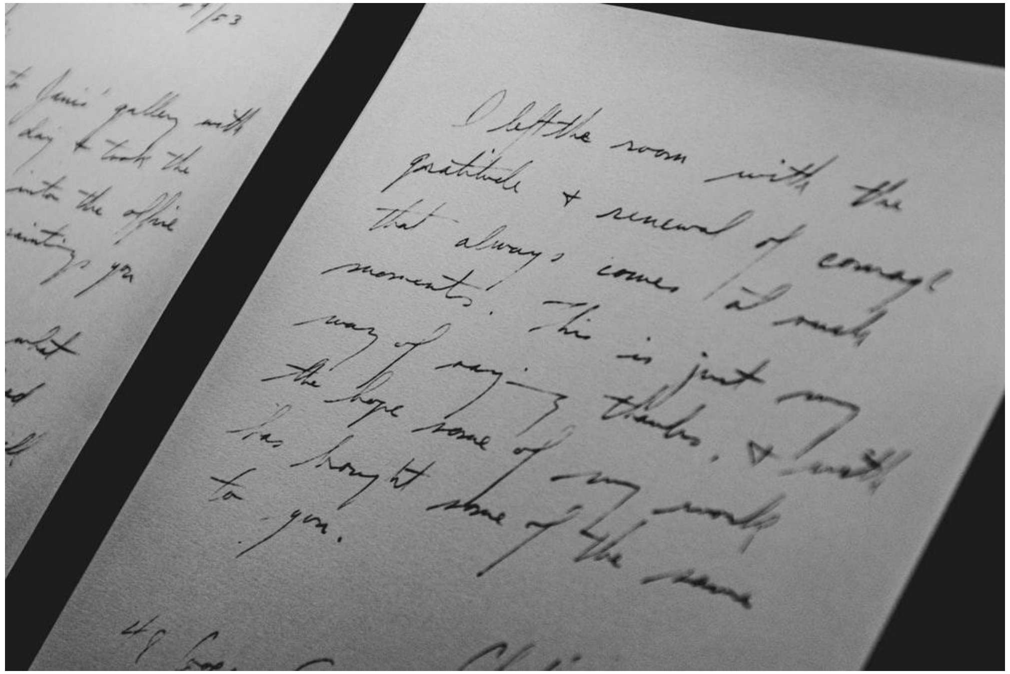
- **RNN basics**
  - sequential tasks, hidden state, vanilla RNN
- **RNN variants + LSTMs**
  - RNN training, variants, LSTM cells
- **Practical training**
  - data pipelines, initialization, hyperparameter tuning

So Far…

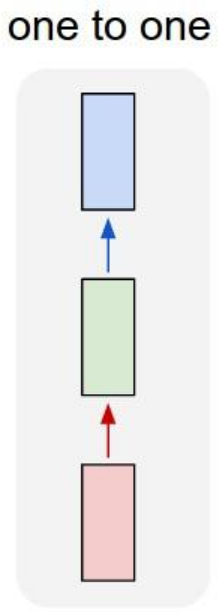
- Our models take **one input** object to **one output** object
  - Fixed-dimensional input vector

- What about sequential data?
  - i.e. language!
  - also, video, many other data

- What should our models do?

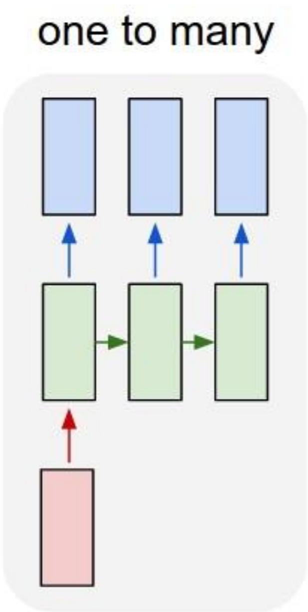


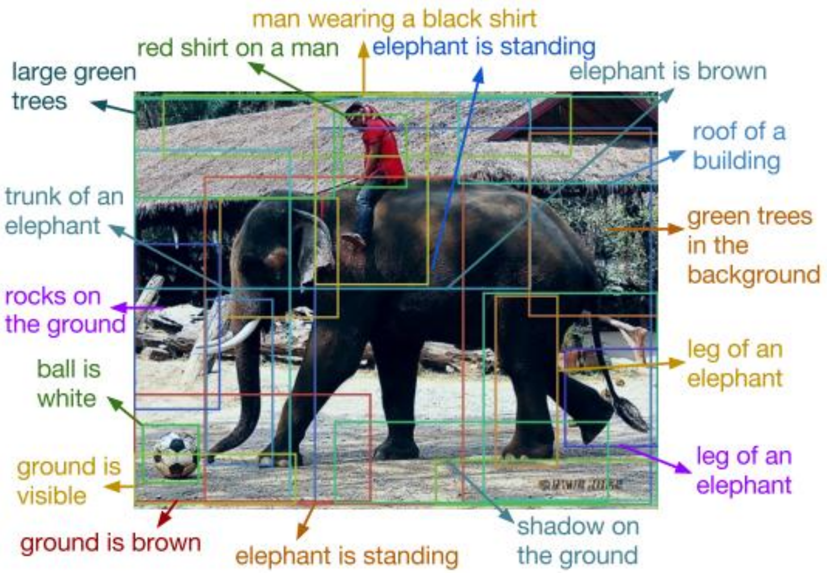
Tasks We Can Handle?



- Our standard model so far. One fixed input type, one output
  - Image classification
  - Doc classification (when represented as a fixed-dimensional vector)

Tasks We Can Handle?

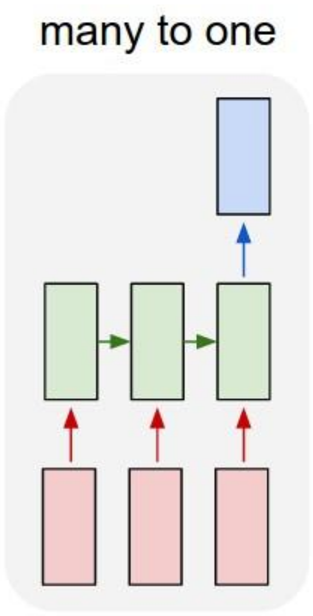


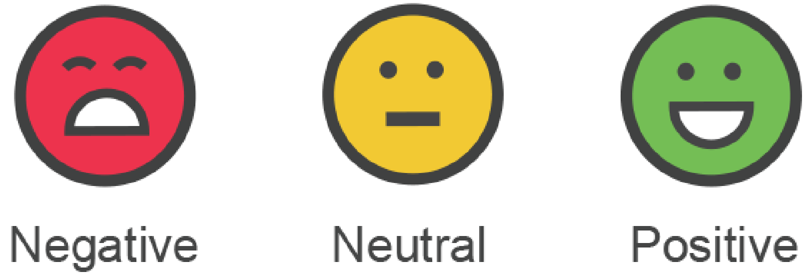


"DenseCap: Fully Convolutional Localization Networks for Dense Captioning", Johnson, Karpathy, Li

- One input, but sequence at the output
  - **Ex**: image captioning.
    - Input: one image
    - Output: sequence of words

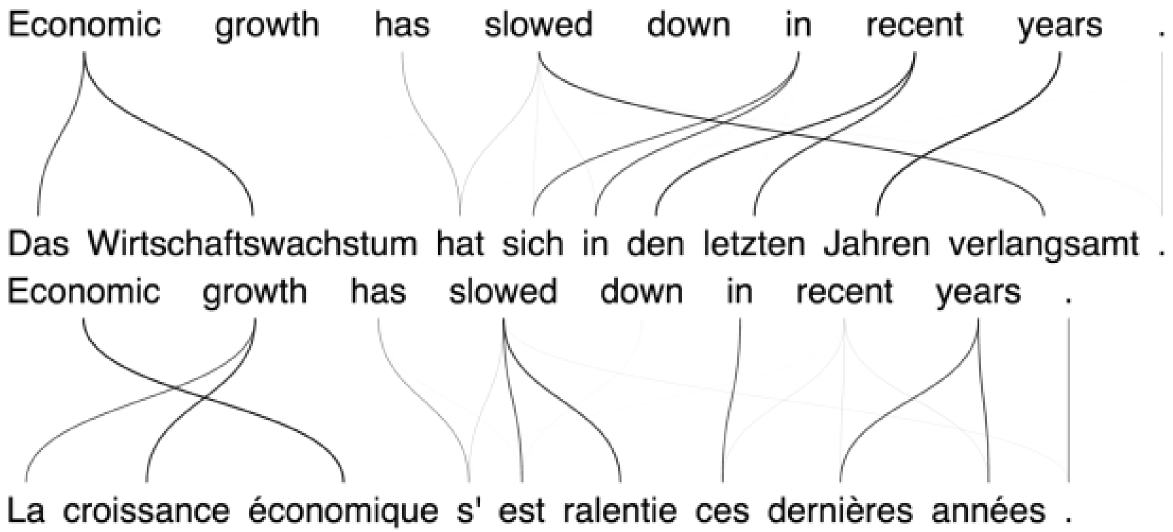
Tasks We Can Handle?



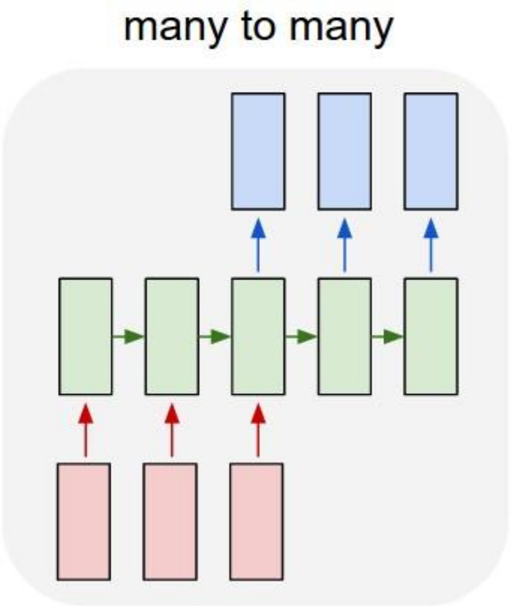


- Sequence input, one output
  - **Ex**: sentiment analysis.
    - Input is a sentence (represented as a fixed-dimensional vector)
    - Output is one of {positive, neutral, negative}

Tasks We Can Handle?

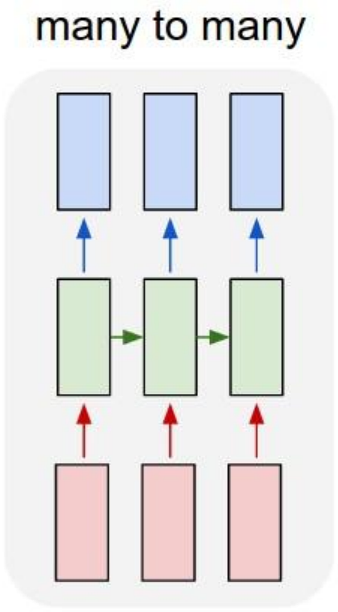


devblogs.nvidia.com



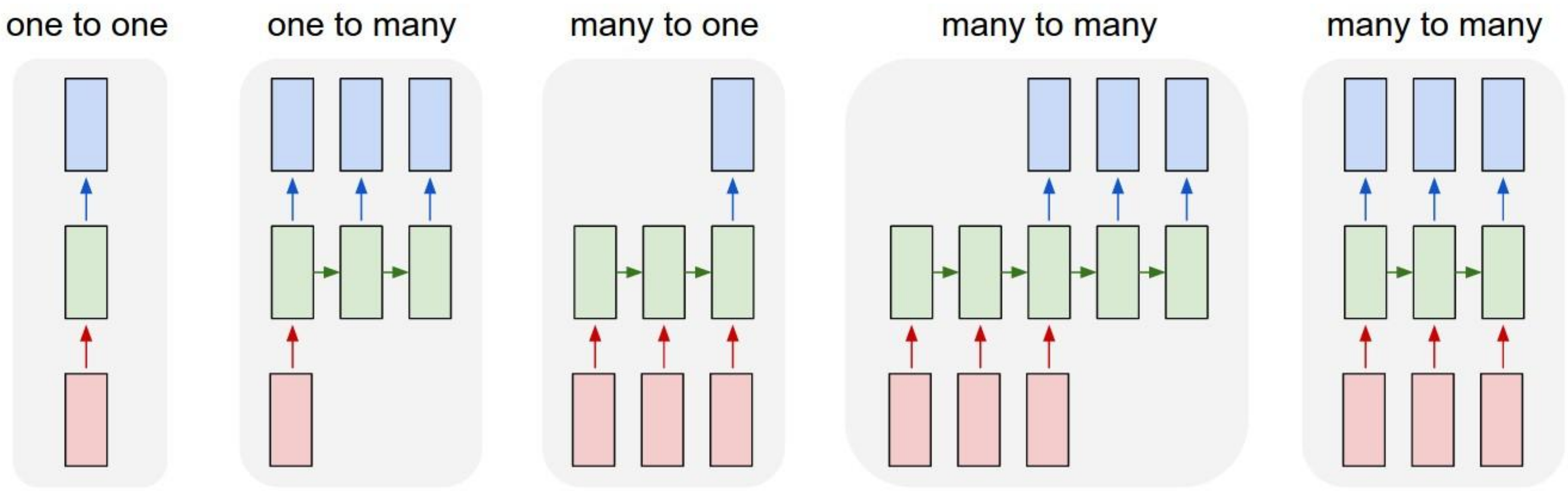
- Sequence input, sequence output
  - **Ex:** machine translation. Translate from language A to language B

Tasks We Can Handle?



- Synchronized input and output
  - **Ex:** Video classification: label each frame of a video

Tasks We Can Handle?

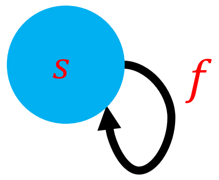


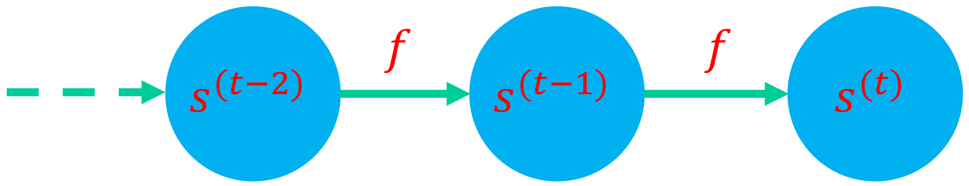
- Don’t have the ability to do anything except the first so far…
  - Need a new kind of model

Modeling Sequential Data

- Simplistic model:
  - $s^{(t)}$ state at time t. Transition function f

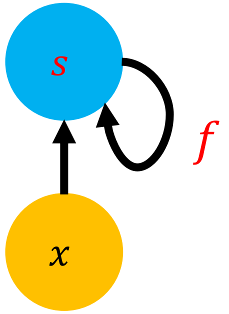
$$s^{(t+1)} = f(s^{(t)}; \theta)$$



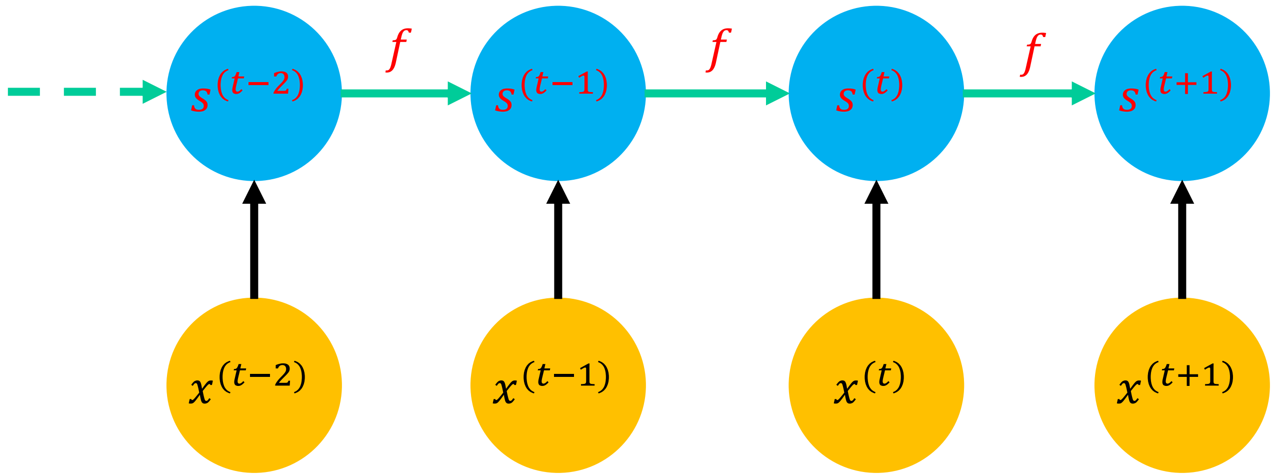


Modeling Sequential Data: External Input

- External inputs can also influence transitions
  - $s^{(t)}$ state at time t. Transition function f
  - $x^{(t)}$: input at time t

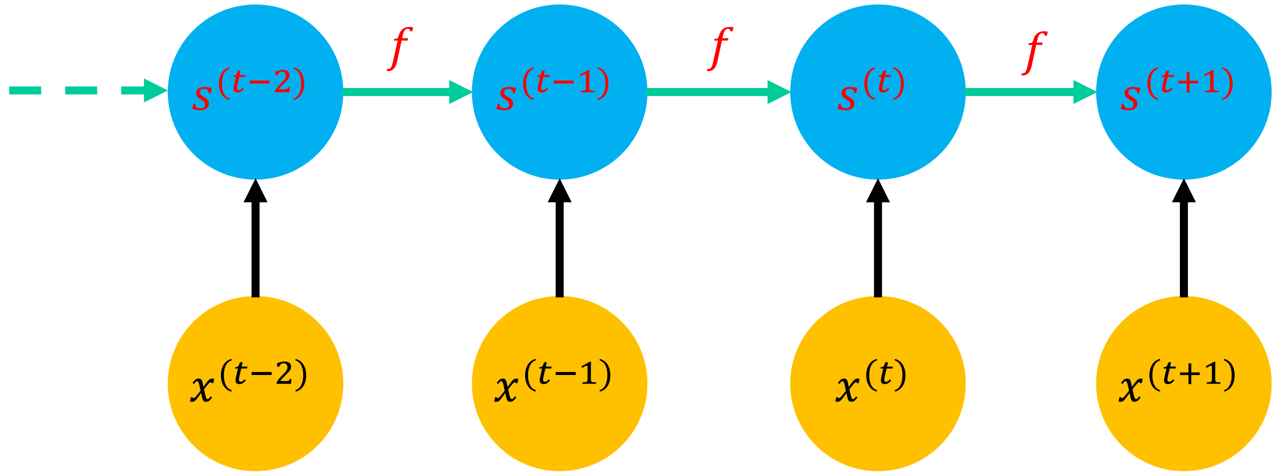


$$s^{(t+1)} = f(s^{(t)}, x^{(t+1)}; \theta)$$



Important: the same $\boldsymbol{f}$ and $\boldsymbol{\theta}$ for all time steps

Recurrent Neural Networks

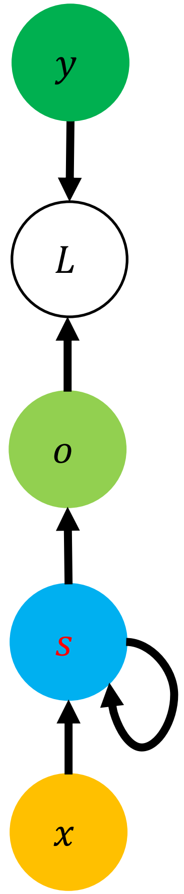


- Use the principle from the system above:
  - Same computational function and parameters across different time steps of the sequence
- Each time step: takes the input entry and the previous hidden state to compute the current hidden state and the **output** entry
- Training: loss typically computed at every time step

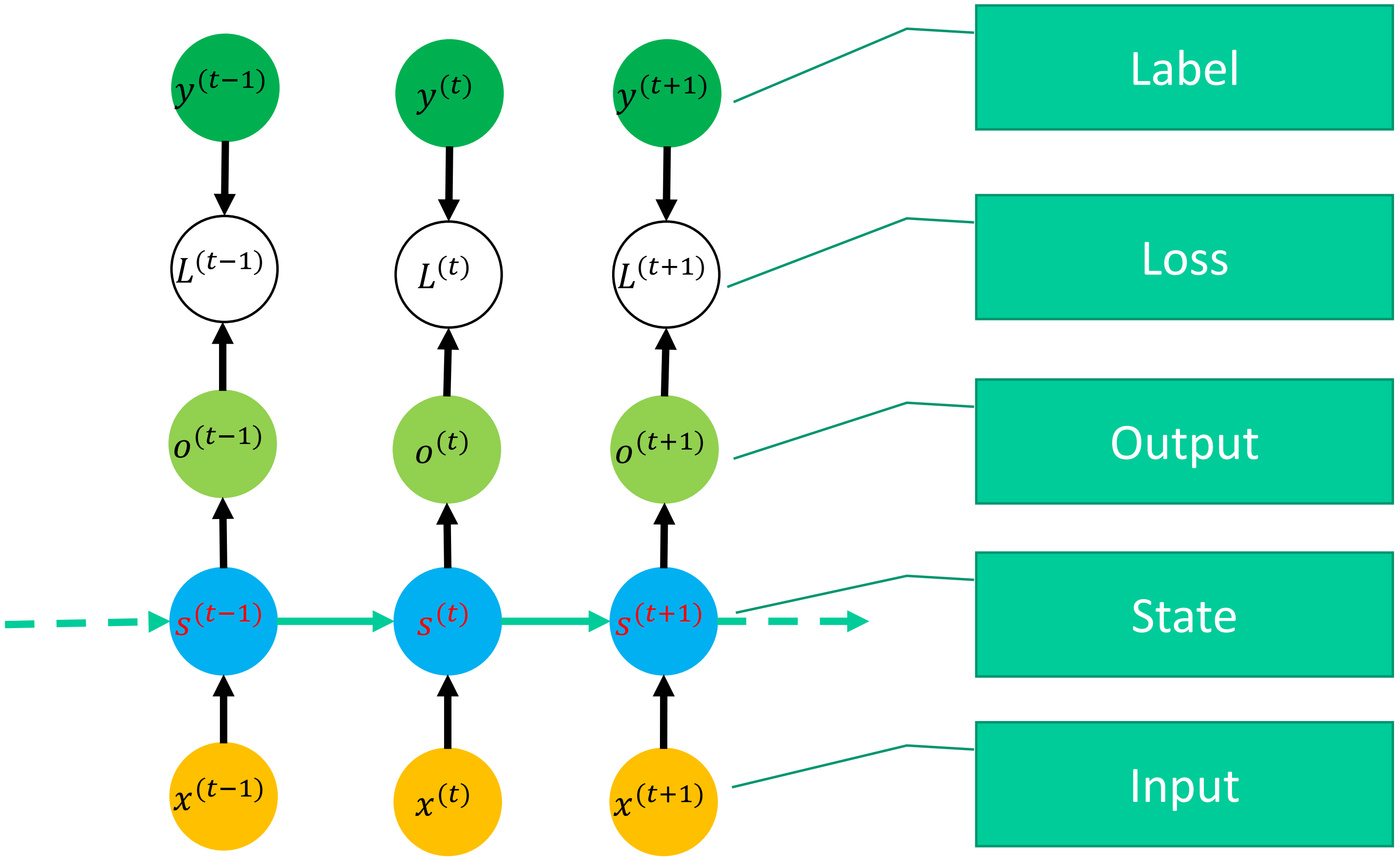
RNNs: Basic Components

- What do we need for our new network?
  - Input x
  - State s
  - Output o
  - Labels y & Loss function L
    - Still need to train!

Recurrent: state is plugged back into itself

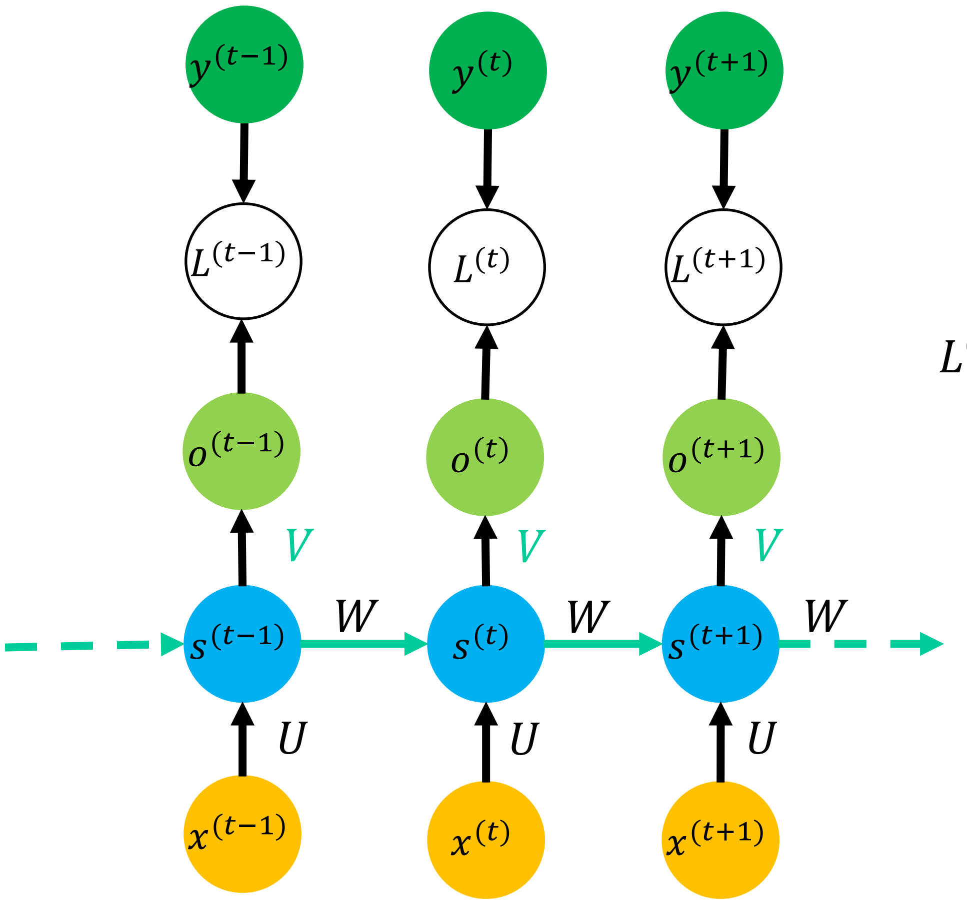


RNNs: Unrolled Graph



Simple RNNs

- Classical RNN variant:



$$a^{(t)} = b + Ws^{(t-1)} + Ux^{(t)}$$
$$s^{(t)} = \tanh\left(a^{(t)}\right)$$
$$o^{(t)} = c + Vs^{(t)}$$
$$\hat{y}^{(t)} = \text{softmax}\left(o^{(t)}\right)$$
$$L^{(t)} = \text{CrossEntropy}(y^{(t)}, \hat{y}^{(t)})$$

Properties

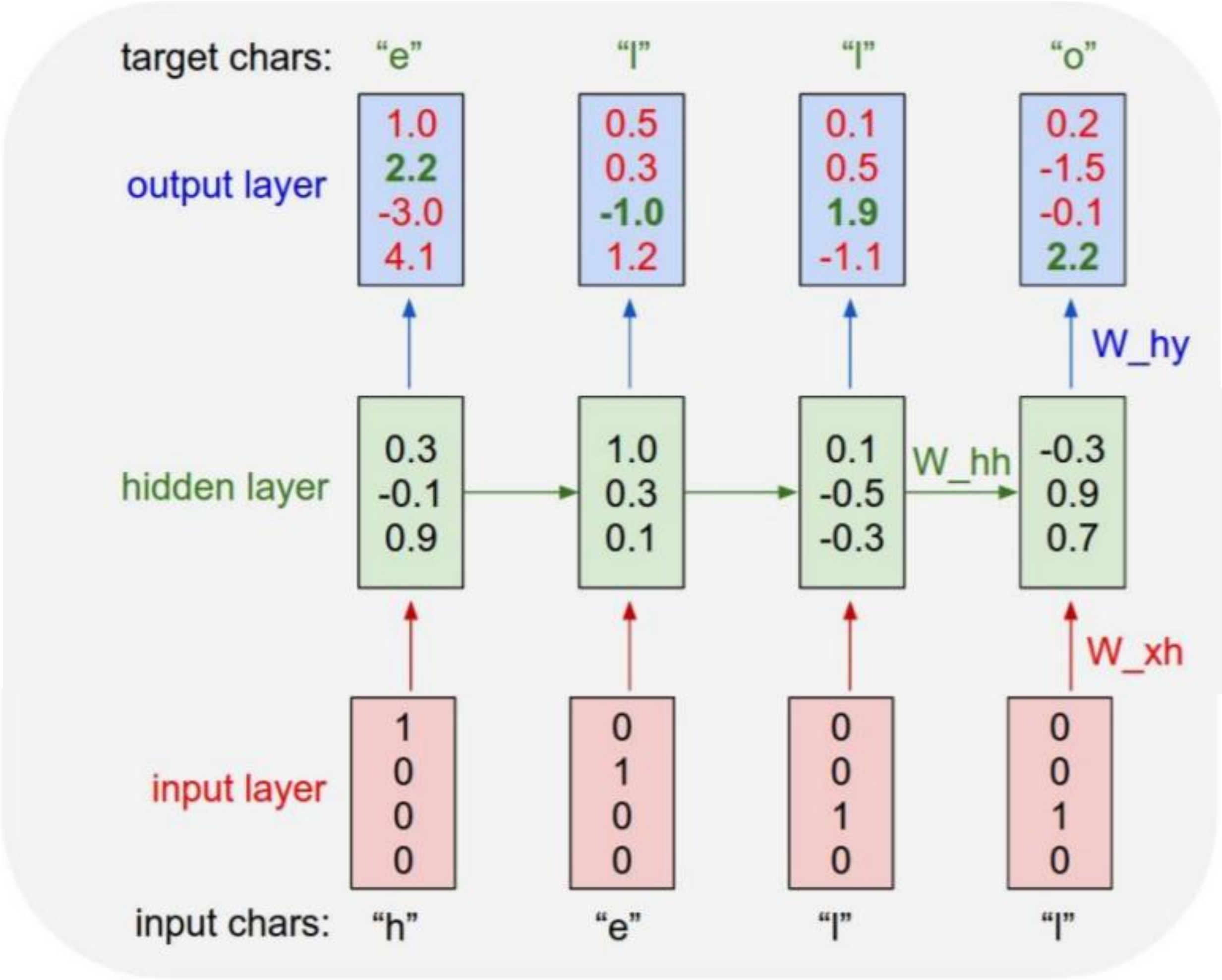
- **Hidden state**: a lossy summary of the past
- Shared functions / parameters
  - Reduce the capacity and good for **generalization**
- Uses the knowledge that sequential data can be processed in the same way at different time step
- Powerful (**universal**): any function computable by a Turing machine computed by such a RNN of a finite size
  - Siegelmann and Sontag (1995)

Example: Char. Level Language Model

- LM goal: predict next character:

- Vocabulary
  {h,e,l,o}

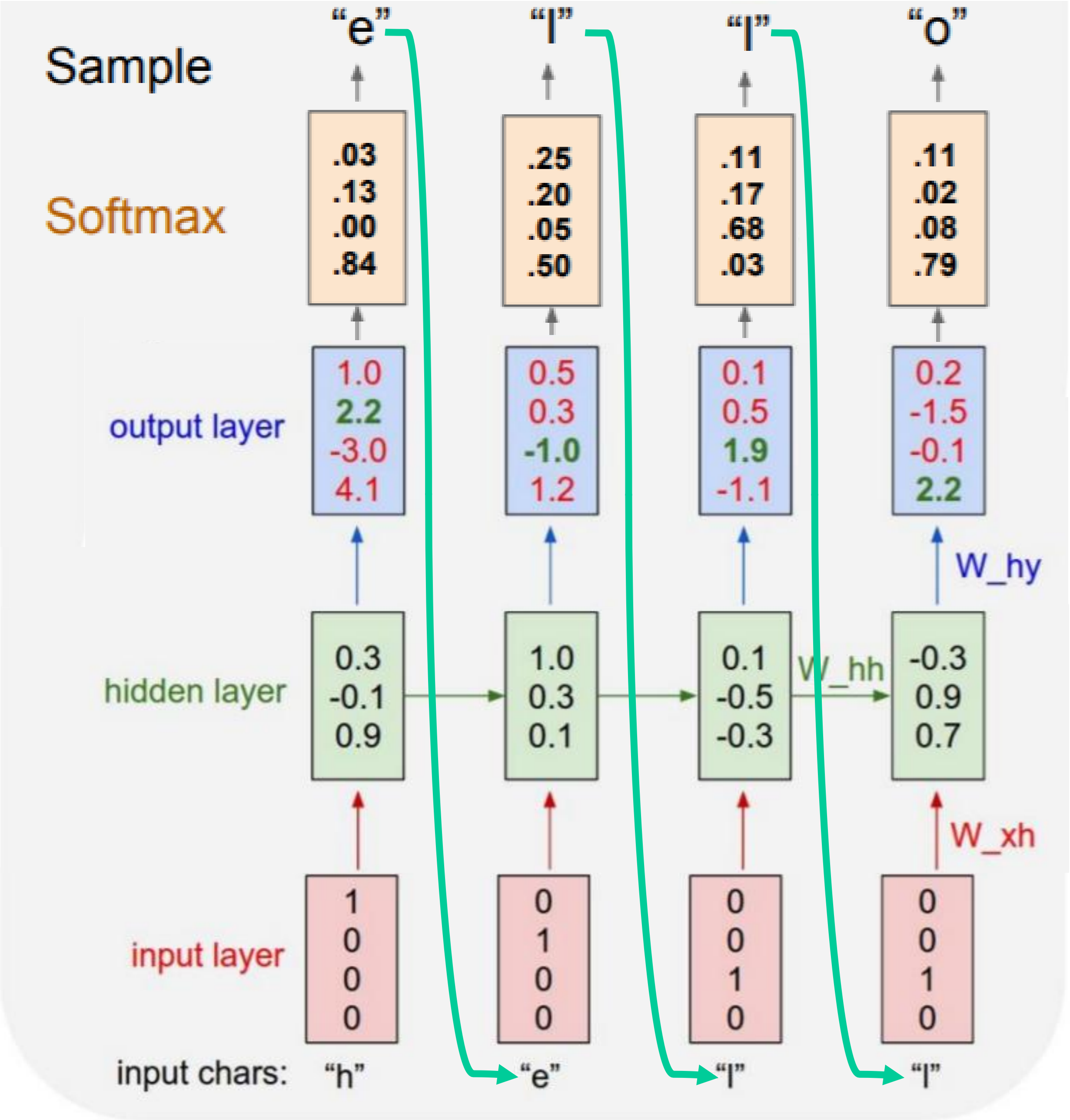
- **Training** sequence:
"hello"



Stanford CS231N

Example: Char. Level Language Model

- LM goal: predict next char
- Vocabulary
  {h,e,l,o}
- **Training** sequence:
"hello"
- Test time:
  - Sample chars and feed back into the model



Break & Quiz

Q: Are these statements true or false?
(A) Order matters in sequential data.
(B) A batch of sequential data always contains sequences of a same length.

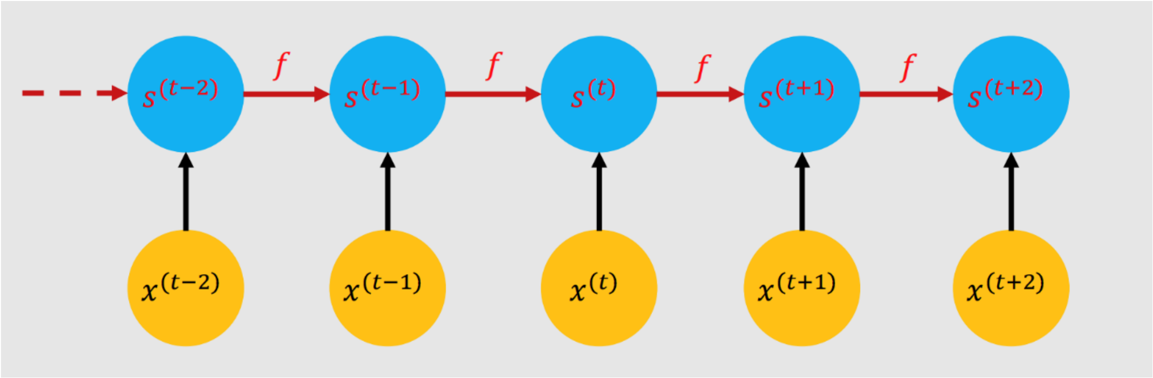
1. True, True
2. True, False
3. False, True
4. False, False

Q: Are these statements true or false?
(A) Order matters in sequential data.
(B) A batch of sequential data always contains sequences of a same length.

1. True, True
2. **True, False**
3. False, True
4. False, False

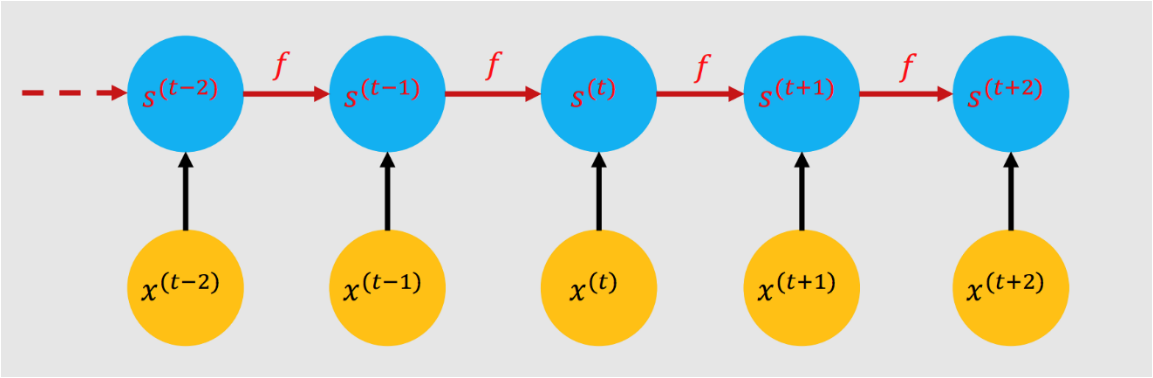
(A) As is shown by its name “sequential”, order matters in sequential data.
(B) A batch of sequential data can have different length, such as different sentences.

Q: Please choose the representation of $s^{(t+2)}$ in terms of $s^{(t)}, x^{(t)}, \ x^{(t+1)}, x^{(t+2)}$ in the following dynamic system $s^{(t+1)} = f_\theta(s^{(t)}, x^{(t+1)})$.



1. $f_\theta(s^{(t)}, x^{(t+1)})$
2. $f_\theta(s^{(t)}, x^{(t+2)})$
3. $f_\theta(f_\theta(s^{(t)}, x^{(t)}), x^{(t+1)})$
4. $f_\theta(f_\theta(s^{(t)}, x^{(t+1)}), x^{(t+2)})$

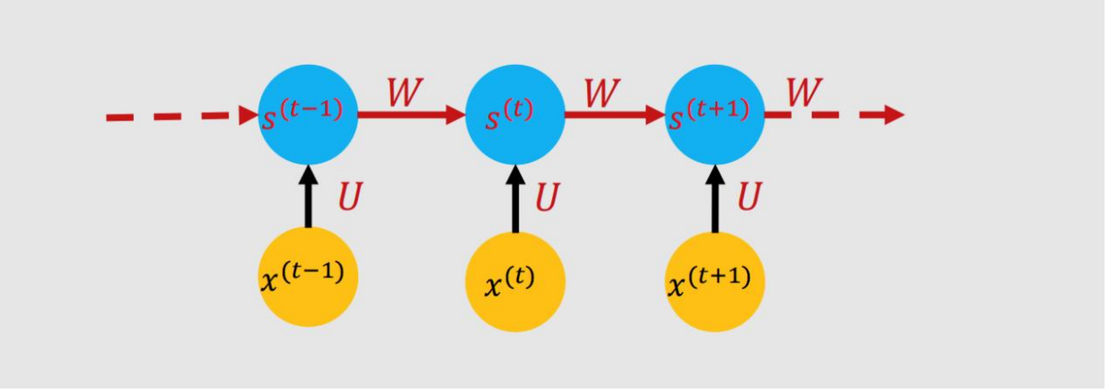
Q: Please choose the representation of $s^{(t+2)}$ in terms of $s^{(t)}, x^{(t)}, x^{(t+1)}, x^{(t+2)}$ in the following dynamic system $s^{(t+1)} = f_\theta(s^{(t)}, x^{(t+1)})$.



1. $f_\theta(s^{(t)}, x^{(t+1)})$
2. $f_\theta(s^{(t)}, x^{(t+2)})$
3. $f_\theta(f_\theta(s^{(t)}, x^{(t)}), x^{(t+1)})$
4. $\boldsymbol{f_\theta(f_\theta(s^{(t)}, x^{(t+1)}), x^{(t+2)})}$

As is shown in this dynamic system, we have $s^{(t+2)} = f_\theta(s^{(t+1)}, x^{(t+2)}) = f_\theta(f_\theta(s^{(t)}, x^{(t+1)}), x^{(t+2)})$, as $s^{(t+1)} = f_\theta(s^{(t)}, x^{(t+1)})$.

Q: Are these statements true or false?
(A) The hidden state $s^{(t)}$ is the linear combination of the previous hidden state $s^{(t-1)}$ and the external data $x^{(t)}$.
(B) Sharing functions and parameters in RNN leads to inherent limitation on the learning ability of the model.

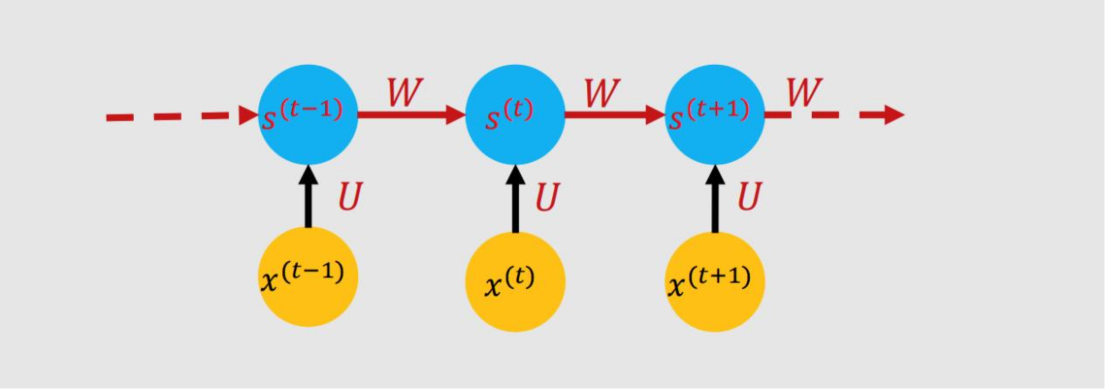


1. True, True
2. True, False
3. False, True
4. False, False

Q: Are these statements true or false?

(A) The hidden state $s^{(t)}$ is the linear combination of the previous hidden state $s^{(t-1)}$ and the external data $x^{(t)}$.

(B) Sharing functions and parameters in RNN leads to inherent limitation on the learning ability of the model.



1. True, True
2. True, False
3. False, True
4. **False, False**

(A) We need to use an activation function to compute the hidden states, so it’s not linear.

(B) As is shown in the lecture, such RNN of a finite size can be universal.

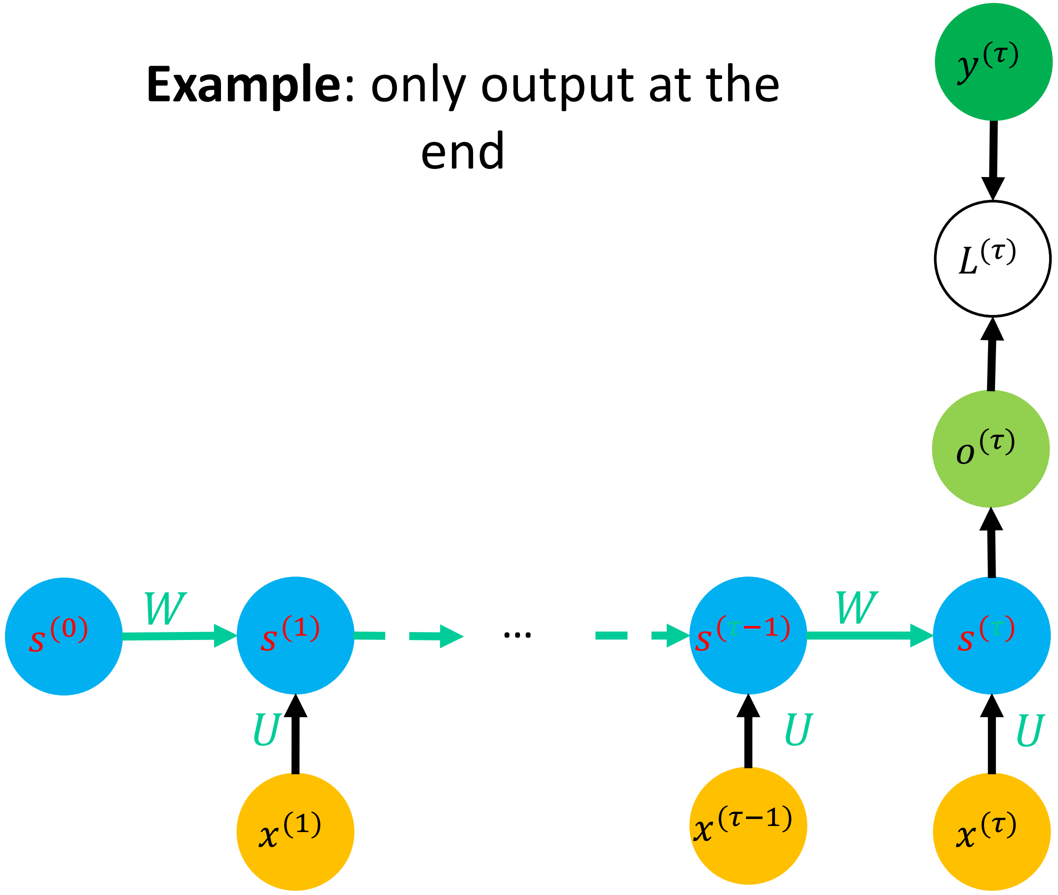
Outline

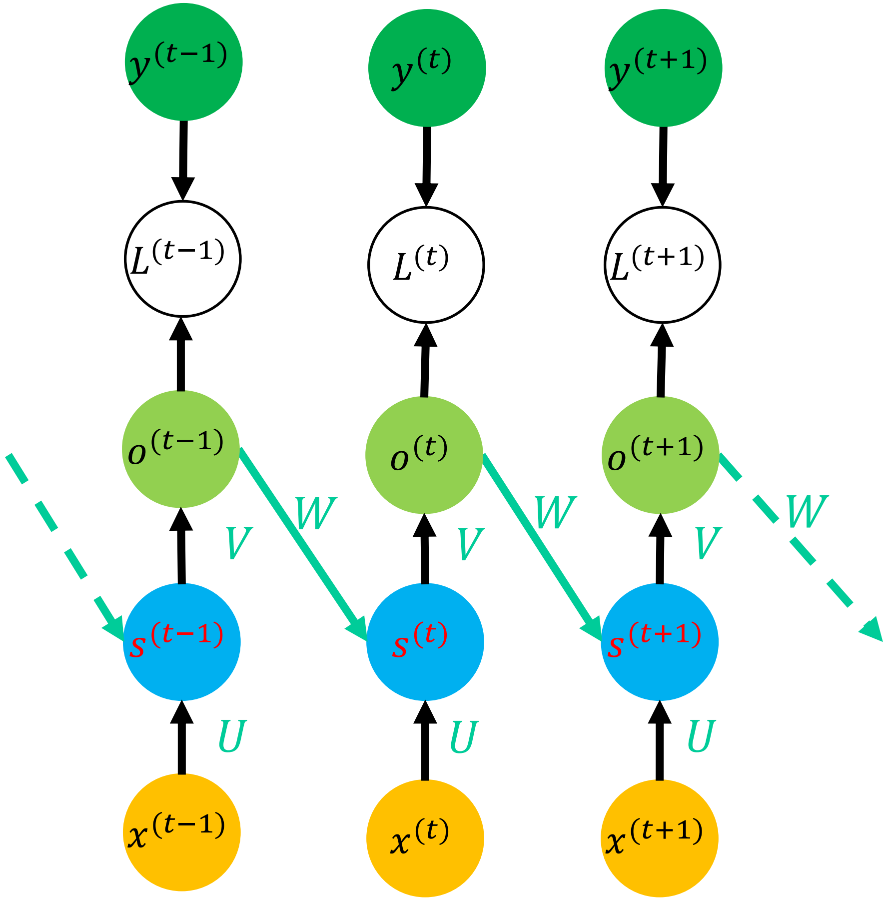
- **RNN basics**
  - sequential tasks, hidden state, vanilla RNN
- **RNN variants + LSTMs**
  - RNN training, variants, LSTM cells
- **Practical training**
  - data pipelines, initialization, hyperparameter tuning

RNN Variants

Example: use the output at the previous step

Example: only output at the end

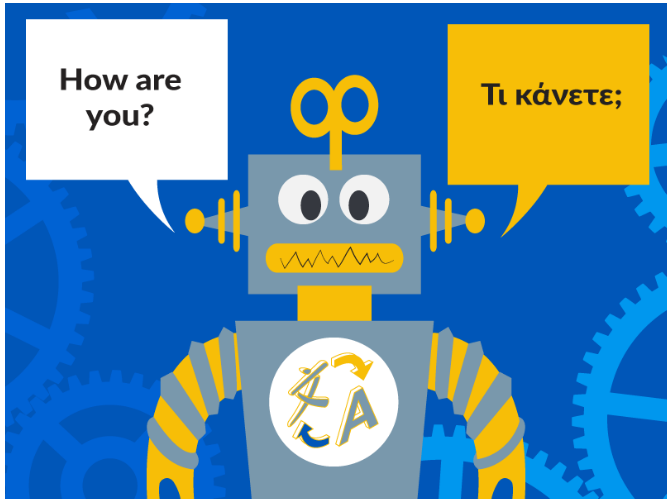




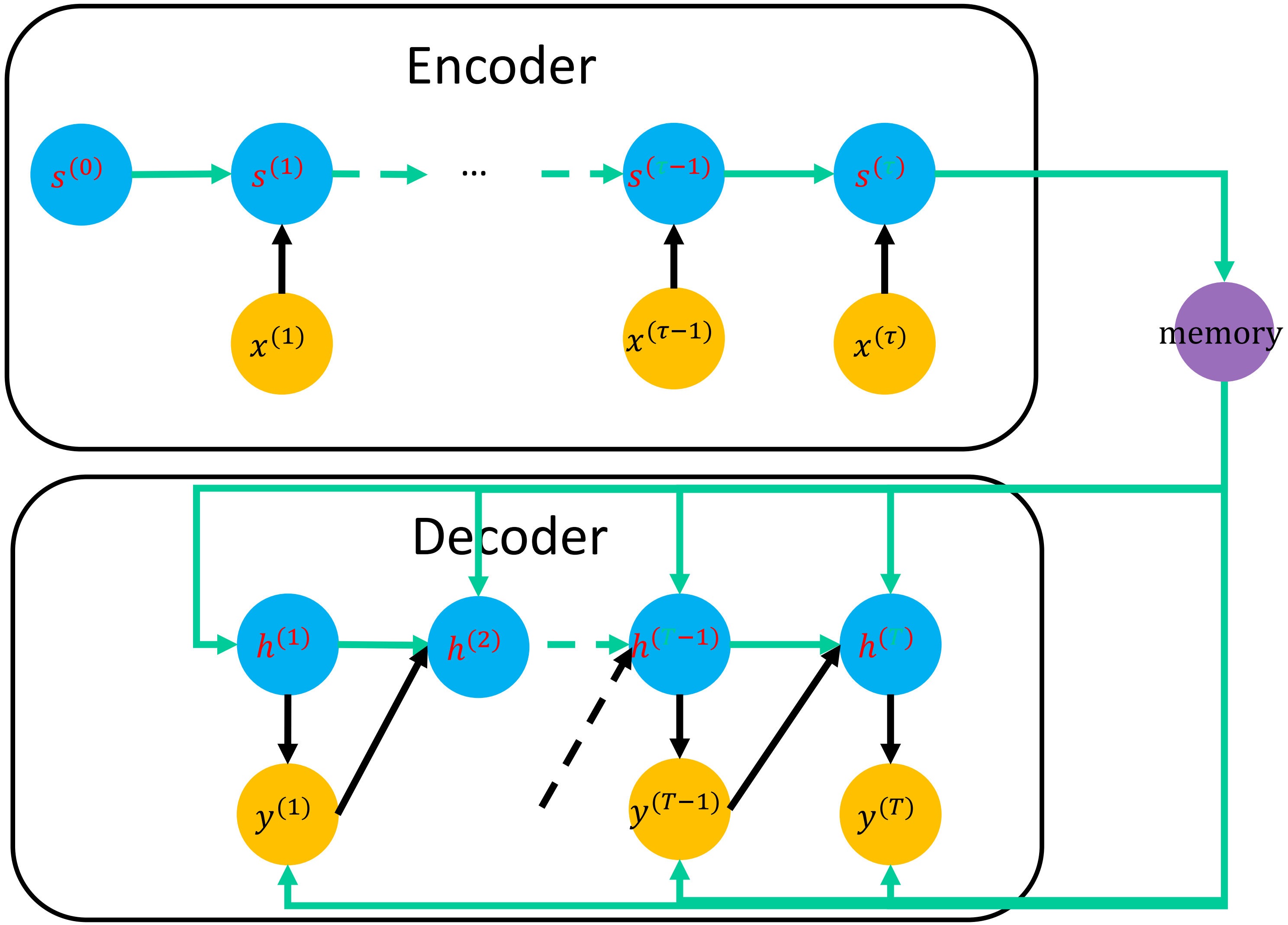
RNN Variants: Encoder/Decoder

- RNNs:
  - can map a sequence to one vector
  - or to sequences of same length

- What about mapping sequence to sequence of different length?
  - **Ex**: speech recognition, machine translation, question answering, etc.



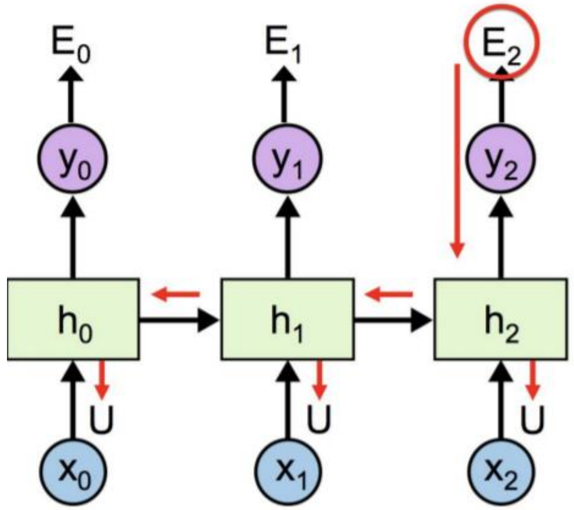
RNN Variants: Encoder/Decoder



Training RNNs

- How: Backpropagation Through Time
  - Idea: unfold the computational graph, and use backpropagation

- Conceptually: first compute the gradients of the internal nodes, then compute the gradients of the parameters



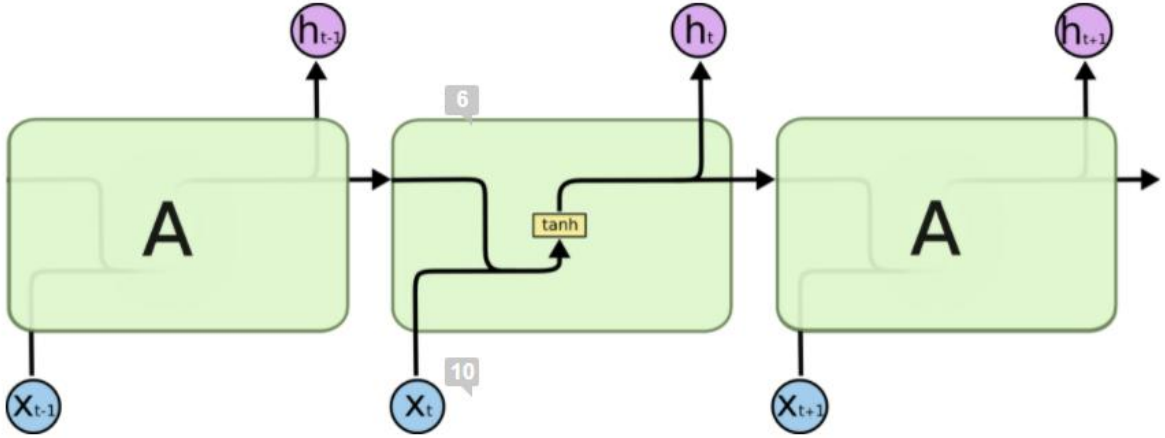
$$\frac{\partial E_2}{\partial U} = \frac{\partial E_2}{\partial h_2}\left(x_2^T + \frac{\partial h_2}{\partial h_1}\left(x_1^T + \frac{\partial h_1}{\partial h_0} x_0^T\right)\right)$$

RNN Problems

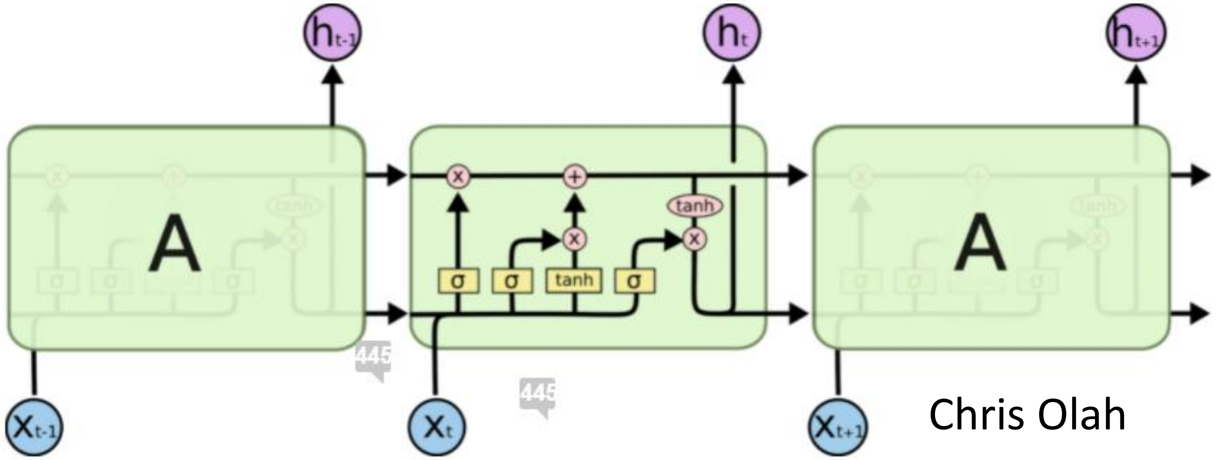
- What happens to gradients in backprop w. many layers?
  - In an RNN trained on long sequences (*e.g.* 100 time steps) the gradients can easily **explode or vanish**.
  - We can avoid this by initializing the weights very carefully.
- Even with good initial weights, very hard to detect that current target output **depends** on an input from long ago.
- RNNs have difficulty dealing with long-range dependencies.
- **Most popular solution: LSTMs**

LSTM Architecture

- RNN: can write structure as:



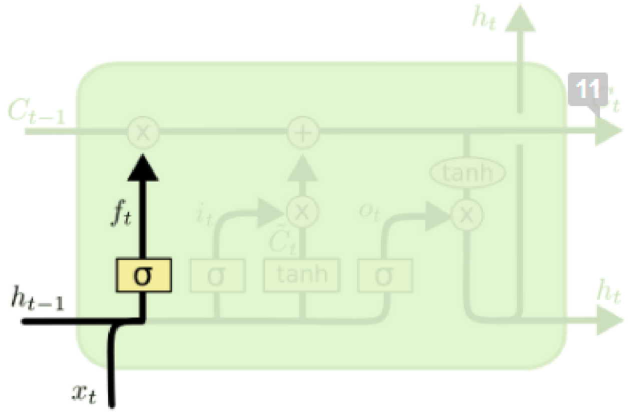
- Long Short-Term Memory:



Chris Olah

Understanding the LSTM Cell

- Step-by-step
  - Good reference: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

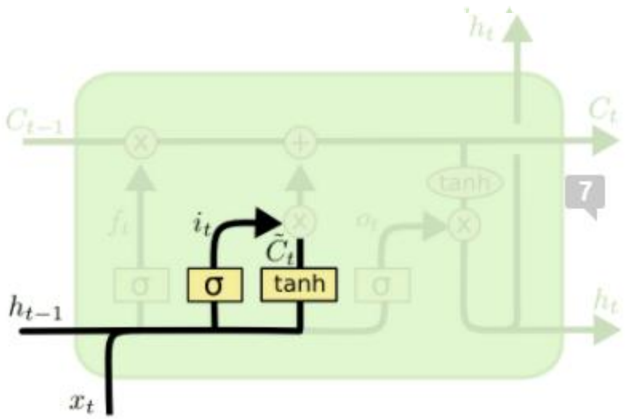


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

- "**Forget**" gate.
  - Can remove all or part of any entry in cell state C
  - Note the sigmoid activation

Understanding the LSTM Cell

- Step-by-step

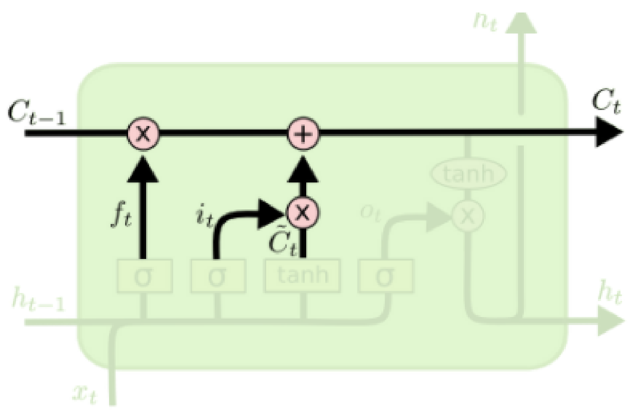


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

- **Input** gate. Combine:
  - What entries in C_{t-1} we'll update
  - Candidates for updating: $Ć_t$
  - Add information to cell state C_{t-1} (post-forgetting)

Understanding the LSTM Cell

- Step-by-step

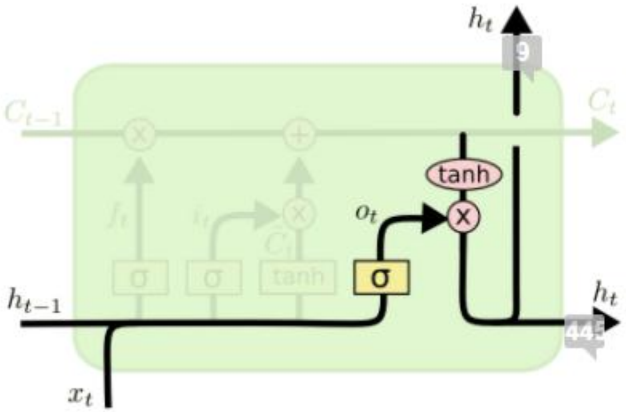


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- Updating C_{t-1} to C_t
  - Forget, then
  - Add new information

Understanding the LSTM Cell

- Step-by-step



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

- **Output** gate
  - Combine hidden state, input as before, but also
  - Modify according to cell state C_t

Outline

- **RNN basics**
  - sequential tasks, hidden state, vanilla RNN
- **RNN variants + LSTMs**
  - RNN training, variants, LSTM cells
- **Practical training**
  - data pipelines, initialization, hyperparameter tuning

Tips & Tricks: Initial Pipeline

First step: building a simple pipeline

- Set up data, model training, evaluation loop

- Use a fixed seed
  - Don’t want to get different values each time

- Overfit on one batch
  - Goal: see that we can get zero loss, catch any bugs

- Check training loss: goes down?

Tips & Tricks: Data

- Shuffle the training data
  - In training ,usually don’t select random examples, but rather go through the dataset for each epoch
  - Shuffle to avoid relationships between consecutive points

- Pay attention to your data
  - Properties?

Tips & Tricks: Initialization

Usually want to pick small random values

- Final layer, could use knowledge of problem.
  - **Ex:** if mean is u, initialize to u
- Don’t want the same value: symmetry means every weights has same gradient, hard to break out of

- Multiple methods: various rules of thumb
  - Sample from a normal distribution
  - Note that #inputs affects the variance… grows as d^2 for d inputs. Can correct by dividing by $1/\sqrt{n}$

Tips & Tricks: Learning Rate Schedule

- Simple ways:
  - Constant
  - Divide by a factor ever certain number of epochs (annealing)
  - Look at validation loss and reduce on plateau

- Also simple: use an optimizer like Adam that internally tracks learning rates
  - In fact, per parameter step-size

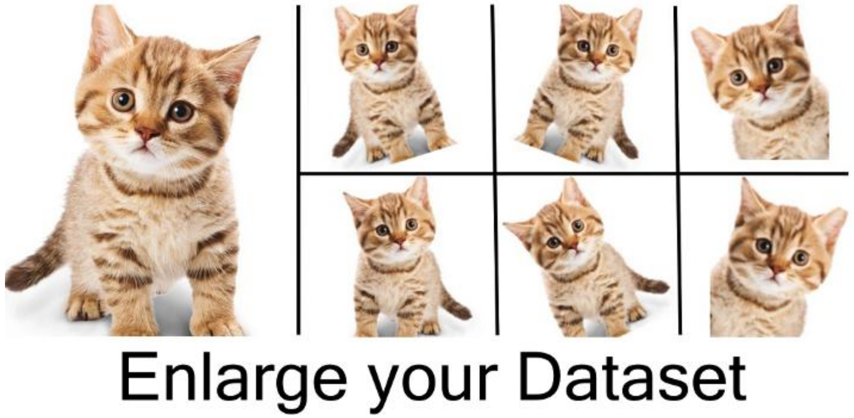
- Lots of variations available

Tips & Tricks: Regularizing

- Best thing to do: get more data!
  - Not always possible or cheap, but start here.

- Augmentation
  - But make sure you understand the transformations

- Use other strategies: dropout, weight decay, early stopping
  - Check each strategy one-at-a-time



Nanonets

Tips & Tricks: Hyperparameter Tuning

Many solutions:

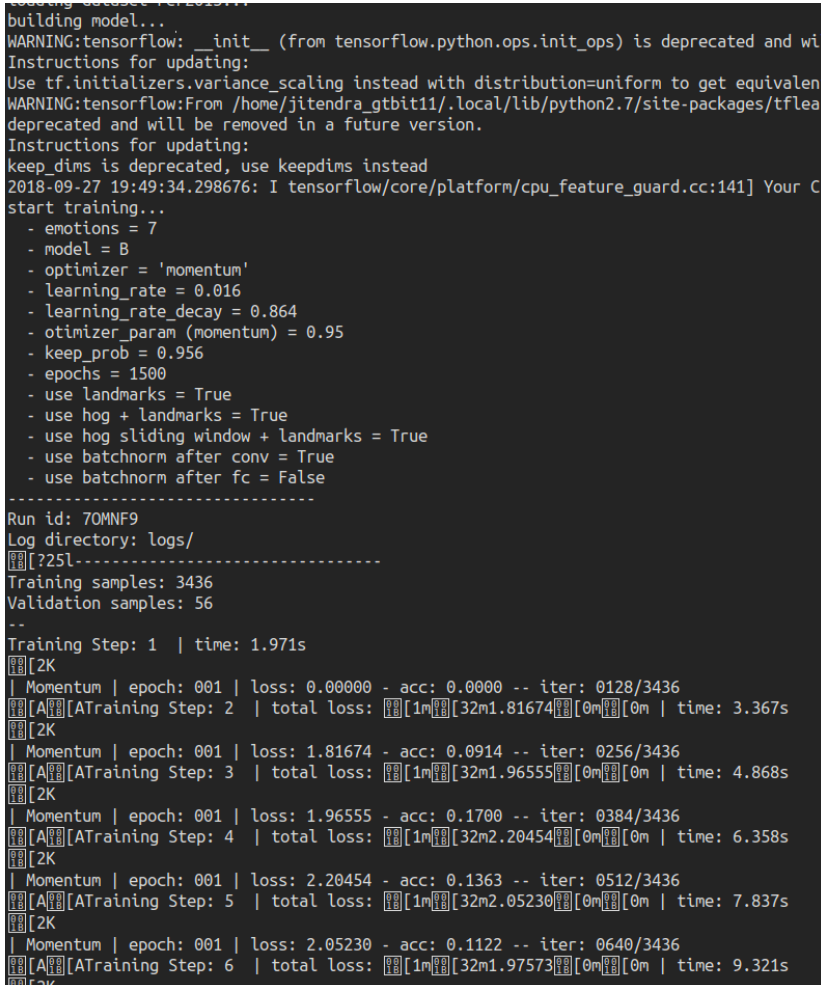
- **Grid search**: pick candidate sets $S_1,\dots,S_k$ for each hparam, search over every combination in $S_1 \times S_2 \times \dots \times S_k$
- **Random search**
- **Bayesian approaches**
- **Successive halving approaches**

Tips & Tricks: Monitoring & Logging

- Checkpoint your models
  - Save weights regularly

- Log information from training process

  - At least keep track of train / test losses, time elapsed, current training settings. Log regularly

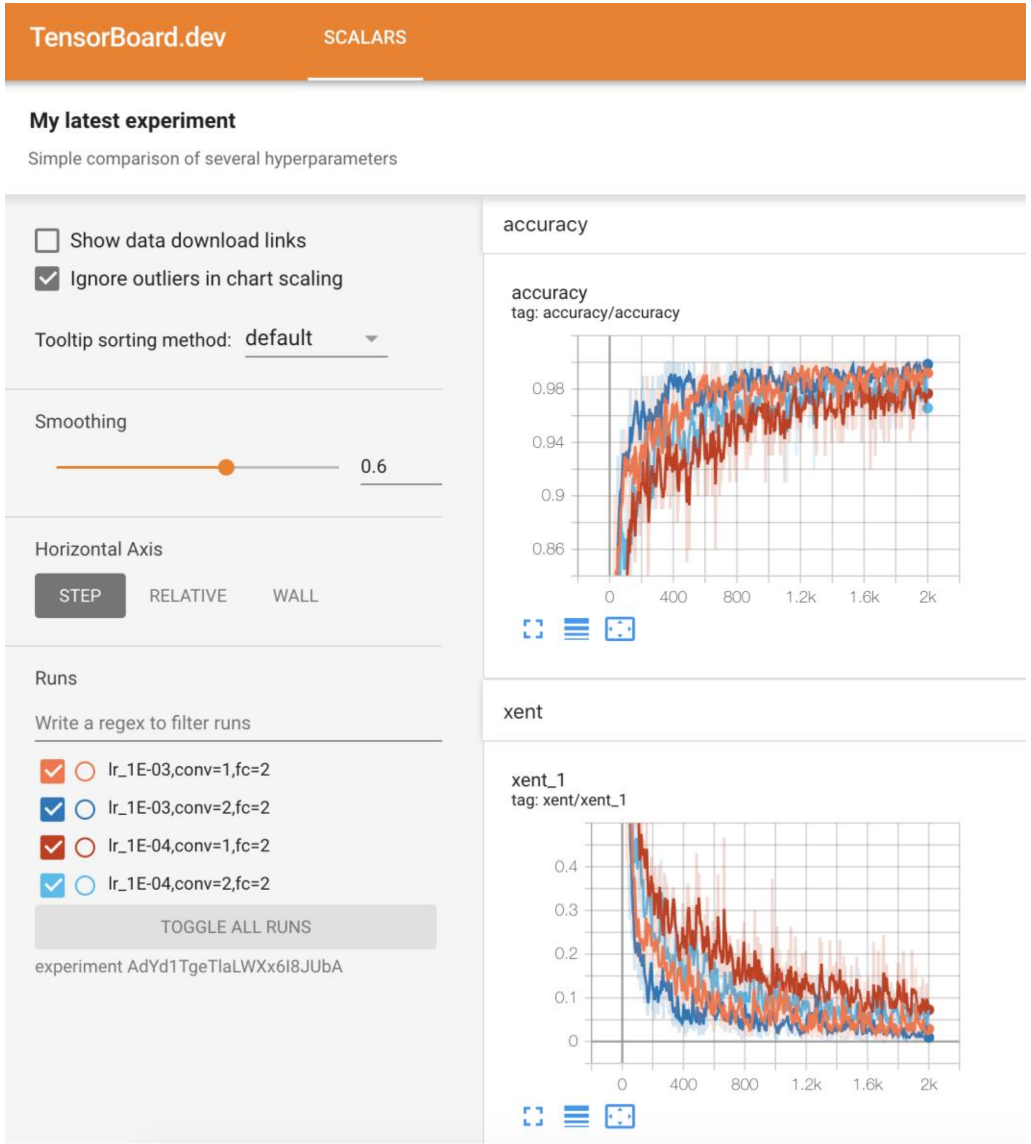


```
building model...
WARNING:tensorflow: __init__ (from tensorflow.python.ops.init_ops) is deprecated and wi
Instructions for updating:
Use tf.initializers.variance_scaling instead with distribution=uniform to get equivalen
WARNING:tensorflow:From /home/jitendra_gtbit11/.local/lib/python2.7/site-packages/tflea
deprecated and will be removed in a future version.
Instructions for updating:
keep_dims is deprecated, use keepdims instead
2018-09-27 19:49:34.298676: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your C
start training...
  - emotions = 7
  - model = B
  - optimizer = 'momentum'
  - learning_rate = 0.016
  - learning_rate_decay = 0.864
  - otimizer_param (momentum) = 0.95
  - keep_prob = 0.956
  - epochs = 1500
  - use landmarks = True
  - use hog + landmarks = True
  - use hog sliding window + landmarks = True
  - use batchnorm after conv = True
  - use batchnorm after fc = False
--------------------------------
Run id: 7OMNF9
Log directory: logs/
[?25l--------------------------------
Training samples: 3436
Validation samples: 56
--
Training Step: 1  | time: 1.971s
[2K
| Momentum | epoch: 001 | loss: 0.00000 - acc: 0.0000 -- iter: 0128/3436
[A[ATraining Step: 2  | total loss: [1m[32m1.81674[0m[0m | time: 3.367s
[2K
| Momentum | epoch: 001 | loss: 1.81674 - acc: 0.0914 -- iter: 0256/3436
[A[ATraining Step: 3  | total loss: [1m[32m1.96555[0m[0m | time: 4.868s
[2K
| Momentum | epoch: 001 | loss: 1.96555 - acc: 0.1700 -- iter: 0384/3436
[A[ATraining Step: 4  | total loss: [1m[32m2.20454[0m[0m | time: 6.358s
[2K
| Momentum | epoch: 001 | loss: 2.20454 - acc: 0.1363 -- iter: 0512/3436
[A[ATraining Step: 5  | total loss: [1m[32m2.05230[0m[0m | time: 7.837s
[2K
| Momentum | epoch: 001 | loss: 2.05230 - acc: 0.1122 -- iter: 0640/3436
[A[ATraining Step: 6  | total loss: [1m[32m1.97573[0m[0m | time: 9.321s
```

NeptuneAI

Tips & Tricks: Monitoring & Logging

- Log information from training process
  - Use software packages
  - Also have built-in visualization
  - Example: TensorBoard, WandB



pytorch.org

Thanks Everyone!

Some of the slides in these lectures have been adapted/borrowed from materials developed by Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Elad Hazan, Tom Dietterich, Pedro Domingos, Jerry Zhu, Yingyu Liang, Volodymyr Kuleshov , Sharon Li, Chris Olah, Fred Sala