CS 760: Machine Learning
Recurrent Neural Networks

Fred Sala

University of Wisconsin-Madison

October 21, 2021

Announcements

- **Logistics**:
  - HW 4 due (Friday!),
  - Midterm review, midterm
- **Class roadmap**:

Thursday, Oct. 21	Neural Networks V
Tuesday, Oct. 26	Practical Aspects of Training + Review
Wed, Oct. 27	**Midterm**
Thursday, Oct. 28	Generative Models
Tuesday, Nov. 2	Kernels + SVMs

Mostly Supervised Learning

Outline

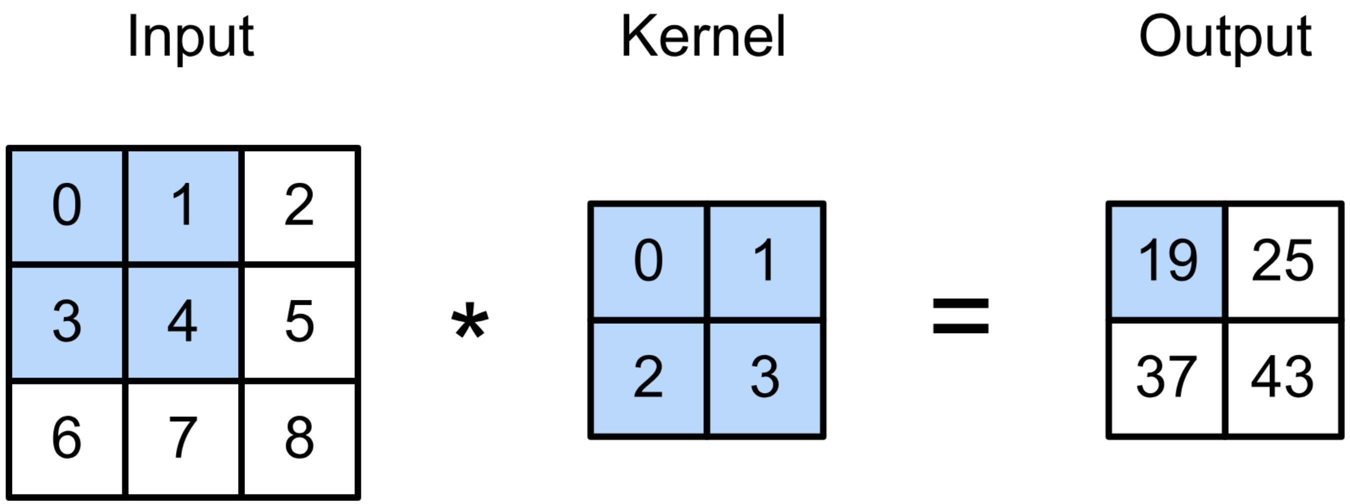
- **CNN Tasks & Architectures**
  - MNIST, ImageNet, LeNet, AlexNet, ResNets
- **RNN Basics**
  - Sequential tasks, hidden state, vanilla RNN
- **RNN Variants + LSTMs**
  - RNN training, variants, LSTM cells

Outline

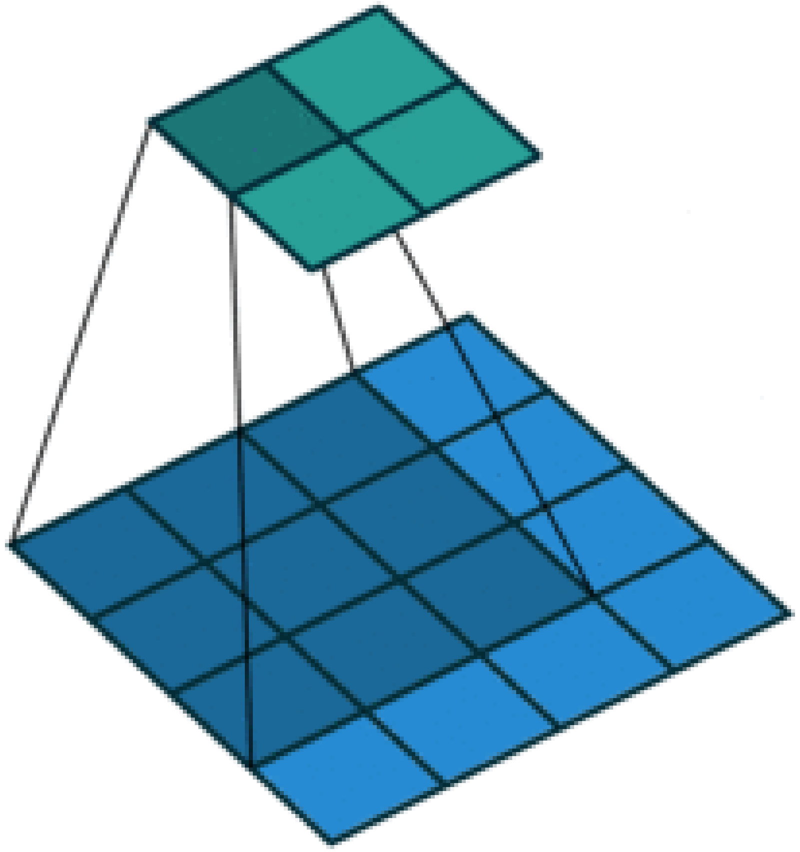
- **CNN Tasks & Architectures**
  - MNIST, ImageNet, LeNet, AlexNet, ResNets
- **RNN Basics**
  - Sequential tasks, hidden state, vanilla RNN
- **RNN Variants + LSTMs**
  - RNN training, variants, LSTM cells

Review: 2-D Convolutions

- Example:



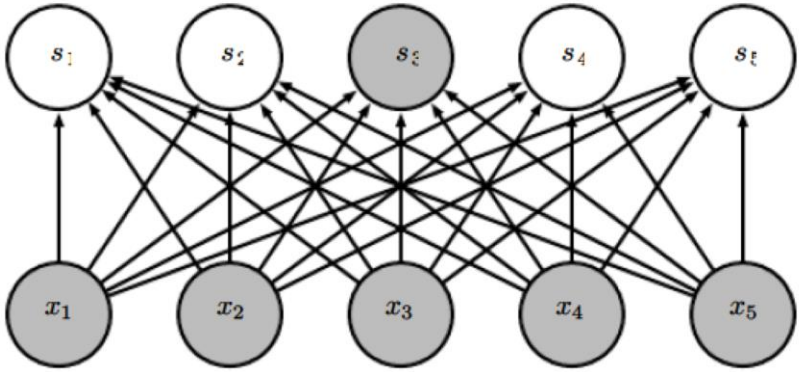
$$
\begin{aligned}
0\times 0+1\times 1+3\times 2+4\times 3&=19,\\
1\times 0+2\times 1+4\times 2+5\times 3&=25,\\
3\times 0+4\times 1+6\times 2+7\times 3&=37,\\
4\times 0+5\times 1+7\times 2+8\times 3&=43.
\end{aligned}
$$



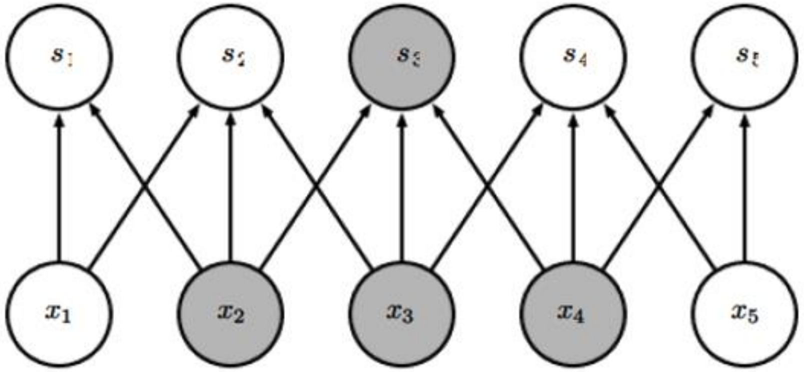
(vdumoulin@ Github)

Review: CNN Advantages

- Fully connected layer: m x n edges

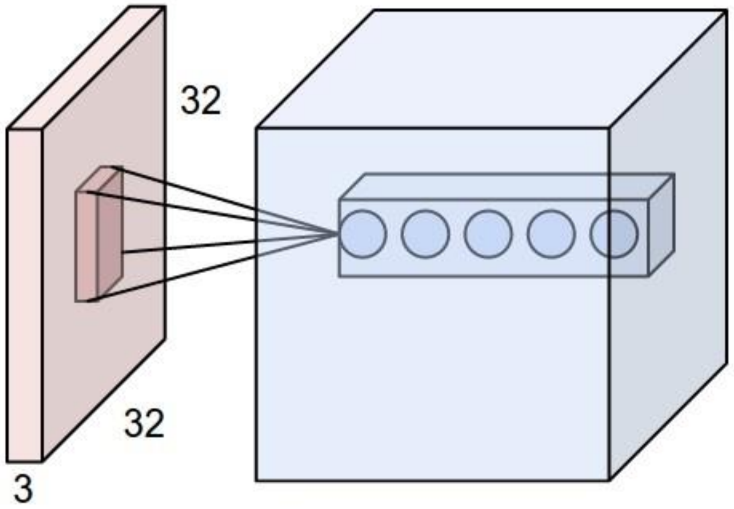


- Convolutional layer: ≤ m x k edges



Review: Convolutional Layers

- Properties
  - Input: volume c_i x n_h x n_w (channels x height x width)
  - Hyperparameters: # of kernels/filters c_o, size k_h x k_w, stride s_h x s_w, zero padding p_h x p_w
  - Output: volume c_o x m_h x m_w (channels x height x width)
  - Parameters: k_h x k_w x c_i per filter, total $(k_h$ x k_w x $c_i)$ x c_o

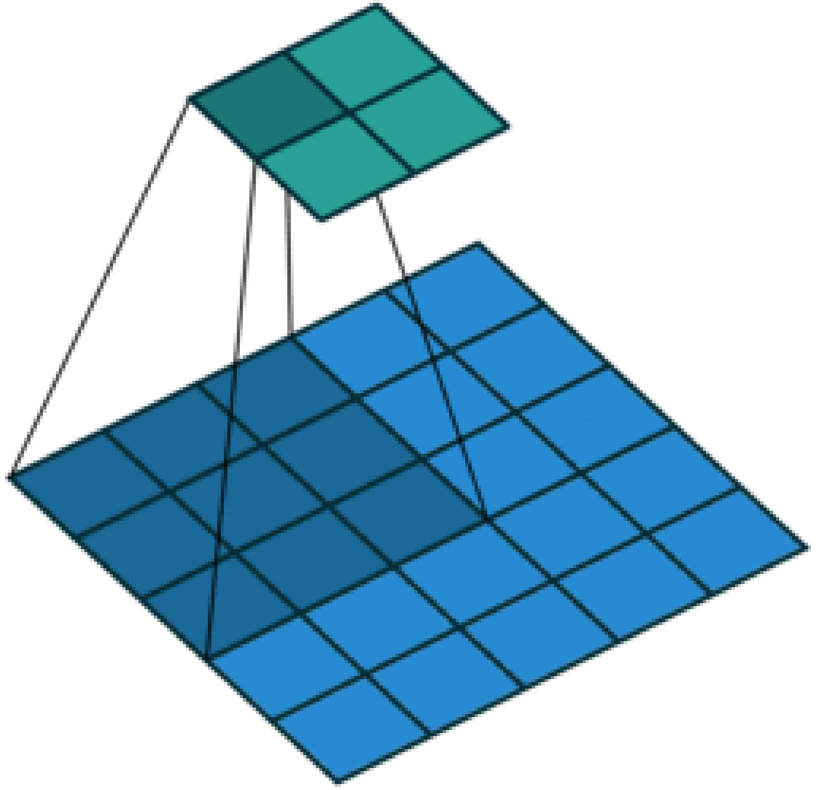


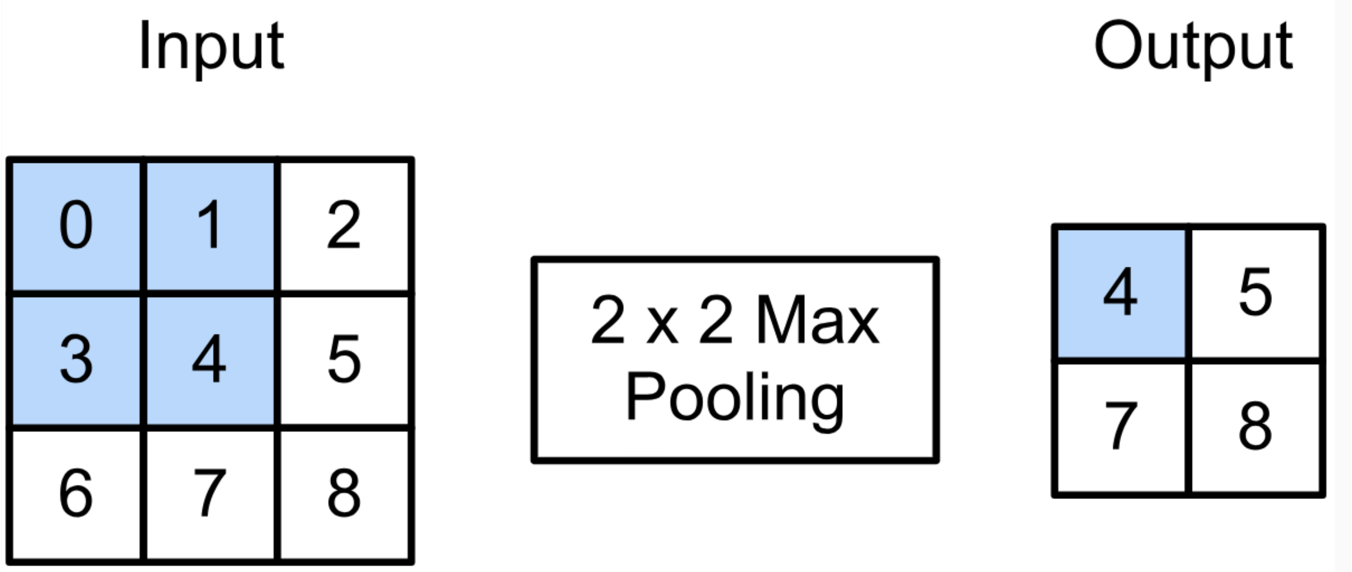
Stanford CS 231n

Review: Max Pooling

- Returns the maximal value in the sliding window

- Example:
  - max(0,1,3,4) = 4



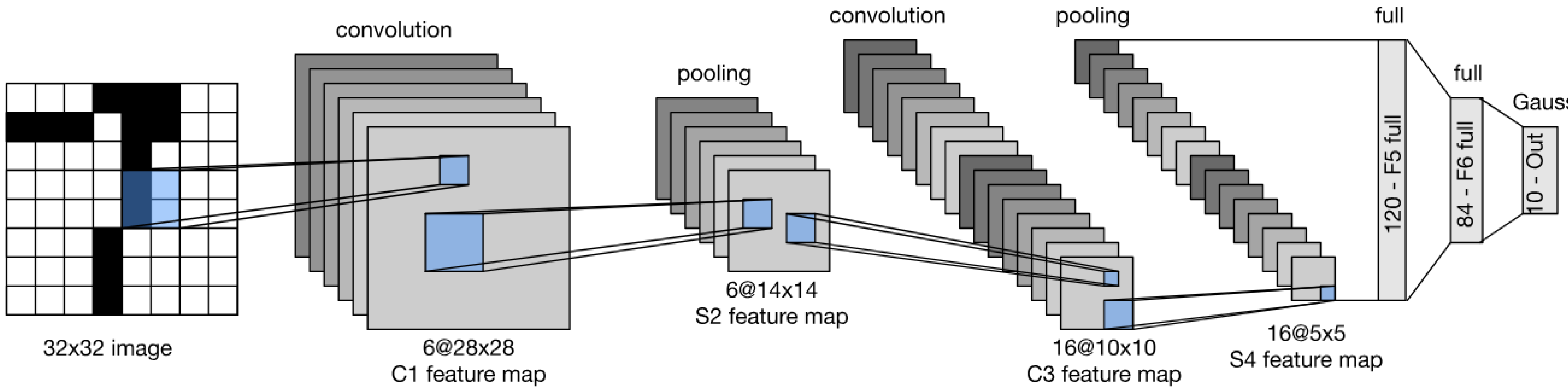


Review: CNN Architectures: LeNet

- Traditional tasks: handwritten digit recognition
- Classic dataset: MNIST
- 1989-1999: LeNet model

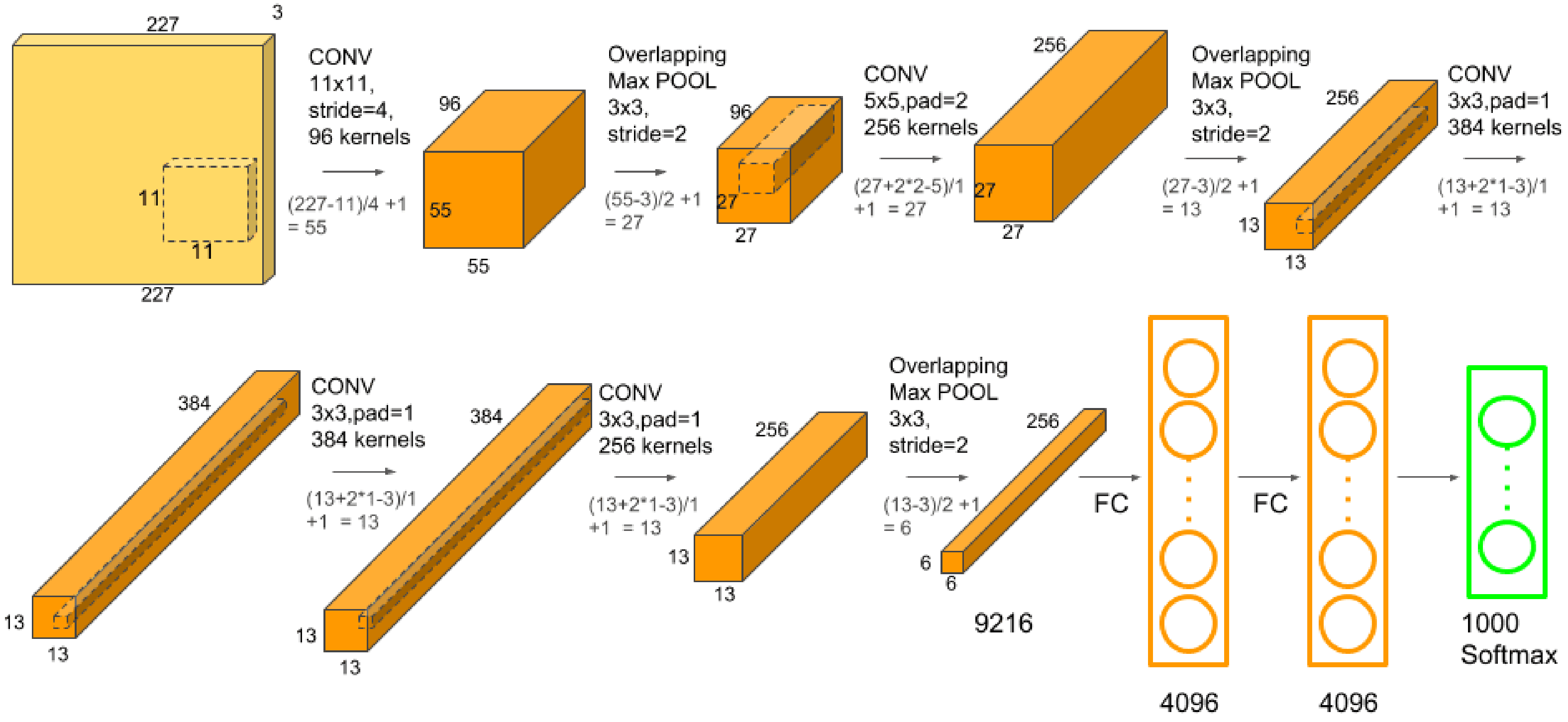
LeCun, Y et al. (1989). Backpropagation applied to handwritten zip code recognition. Neural Computation

LeCun, Y.; Bottou, L.; Bengio, Y. & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proc. IEEE



CNN Architectures: AlexNet

- First of the major advancements: AlexNet
- Wins 2012 ImageNet competition
- Major trends: deeper, bigger LeNet



More CNN Architectures

- AlexNet vs LeNet
  - Architecture comparison

1000 Classes At Output

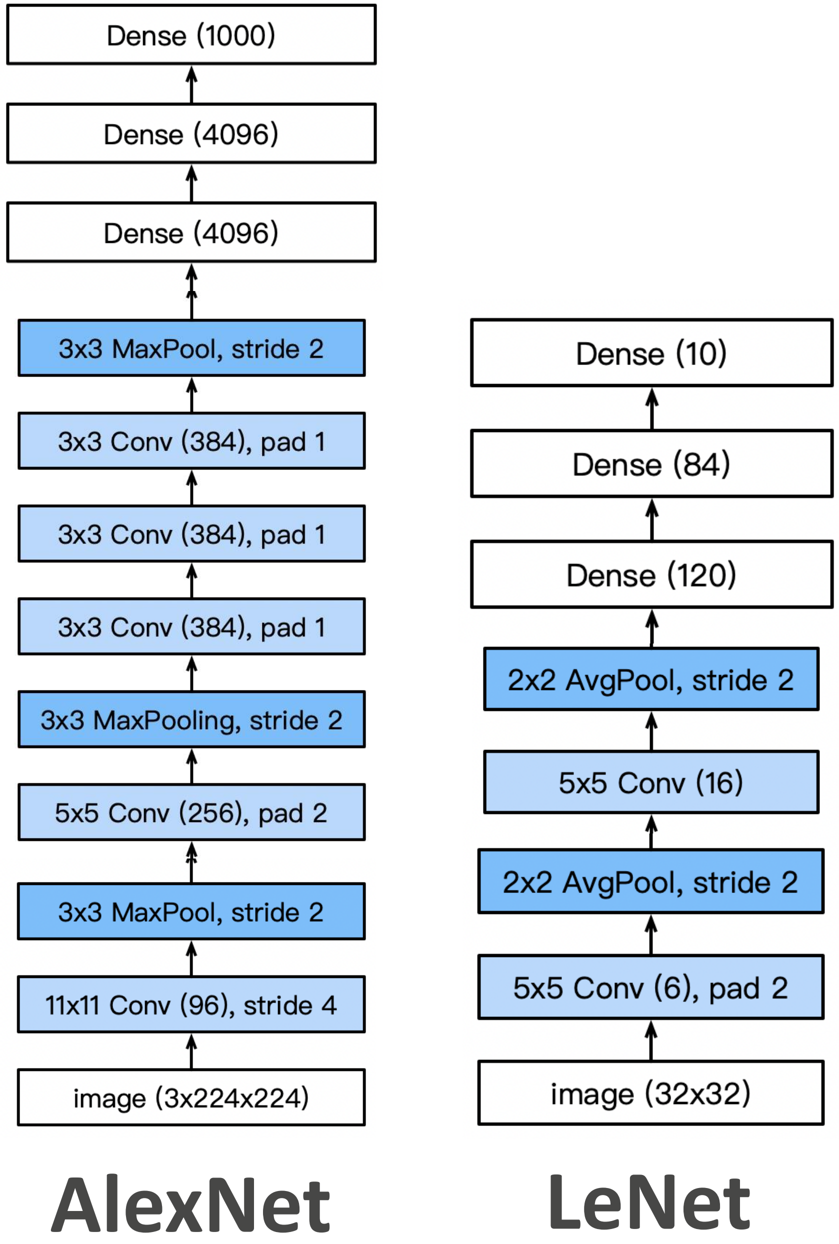
FC Layers Increased Size

More Convolutional Layers

More Output Channels

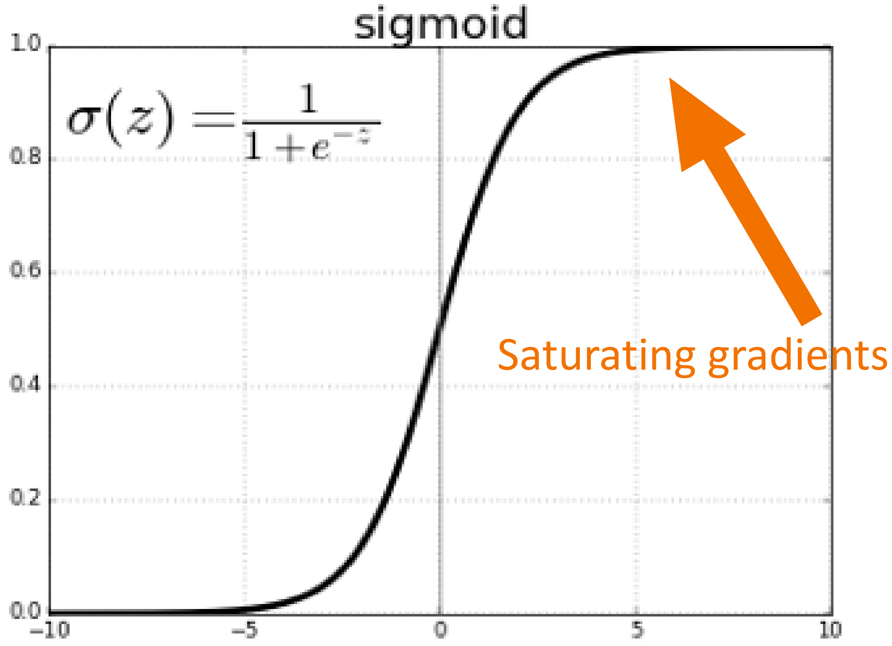
Larger Pool Size

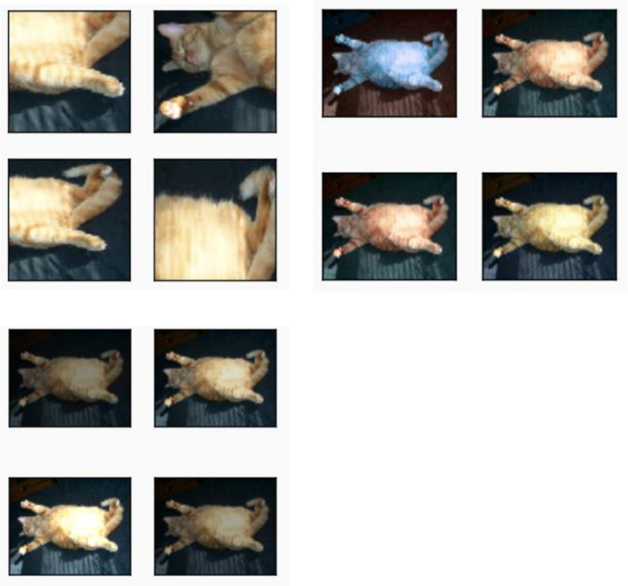
Larger kernel size, stride for increased image size, and more output channels.



More Differences

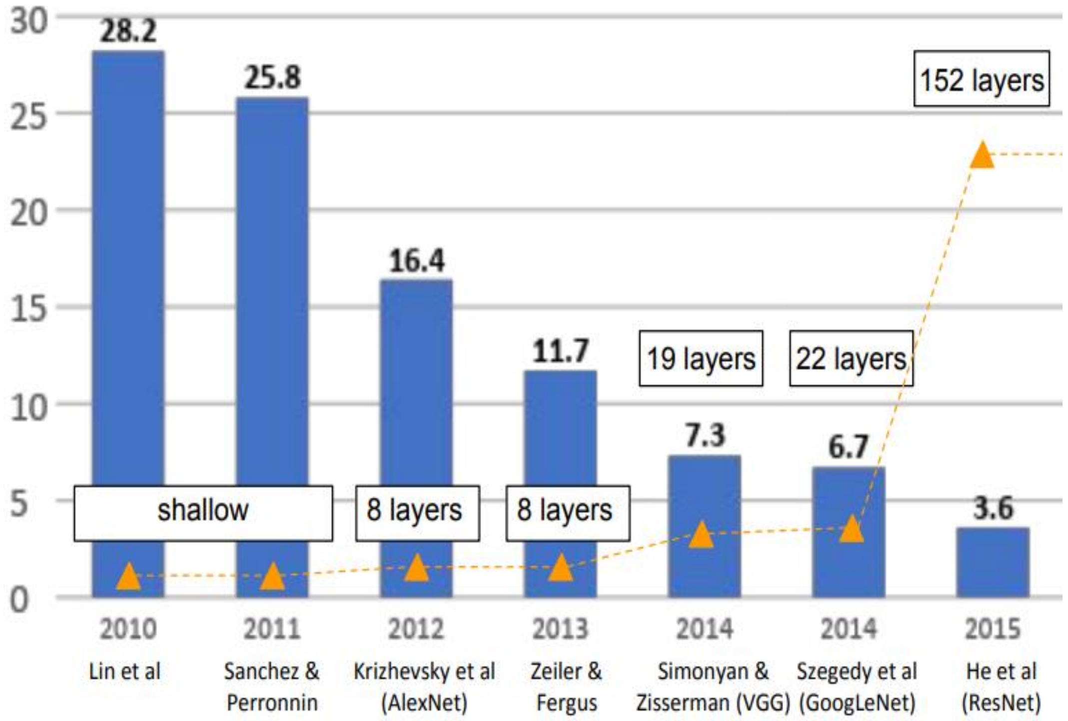
- Activations: from sigmoid to ReLU
  - Deal with vanishing gradient issue
- Data Augmentation





Going Further

- ImageNet error rate
  - Competition winners; note layer count on right.



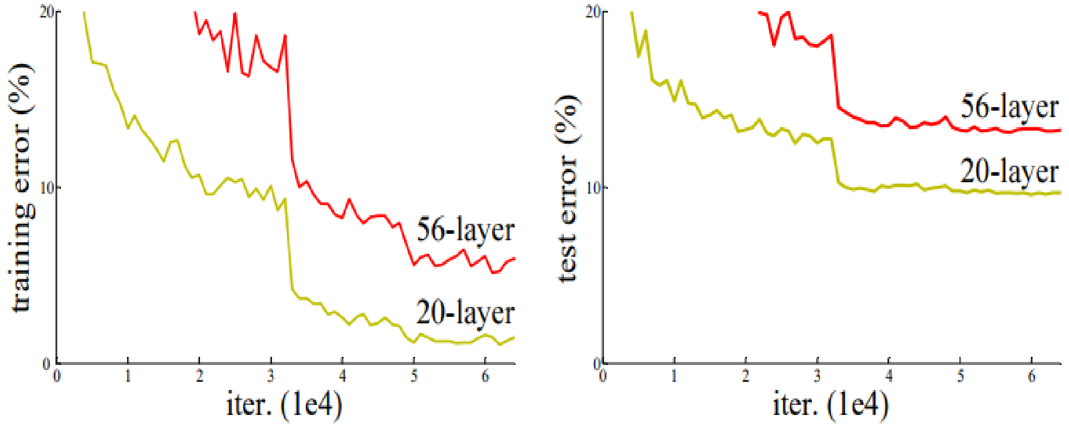
Credit: Stanford CS 231n

Add More Layers: Enough?

VGG: 19 layers. ResNet: 152 layers. **Add more layers**… sufficient?

- No! Some problems:
  - i) Vanishing gradients: more layers ➔ more likely
  - ii) Instability: can't guarantee we learn **identity** maps

Reflected in training error:

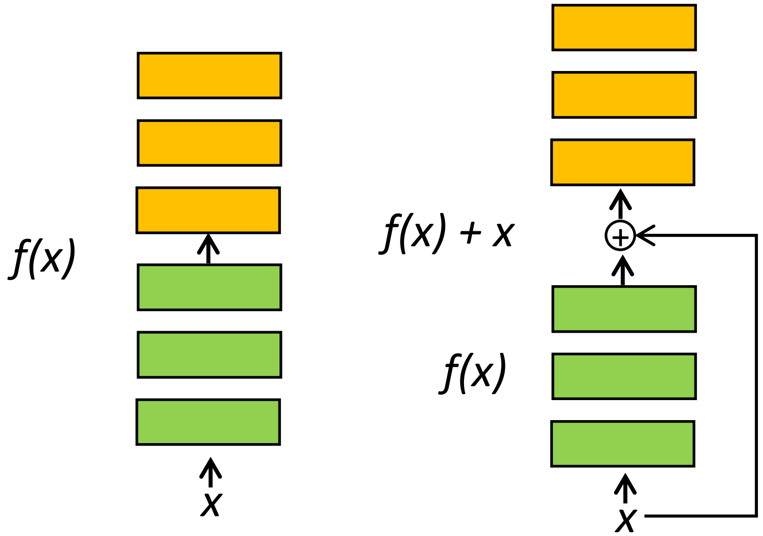


He et al: "Deep Residual Learning for Image Recognition"

Residual Connections

Idea: adding layers can't make worse if we can learn identity

- But, might be hard to learn identity
- Zero map is easy…
  - Make all the weights tiny, produces zero for output



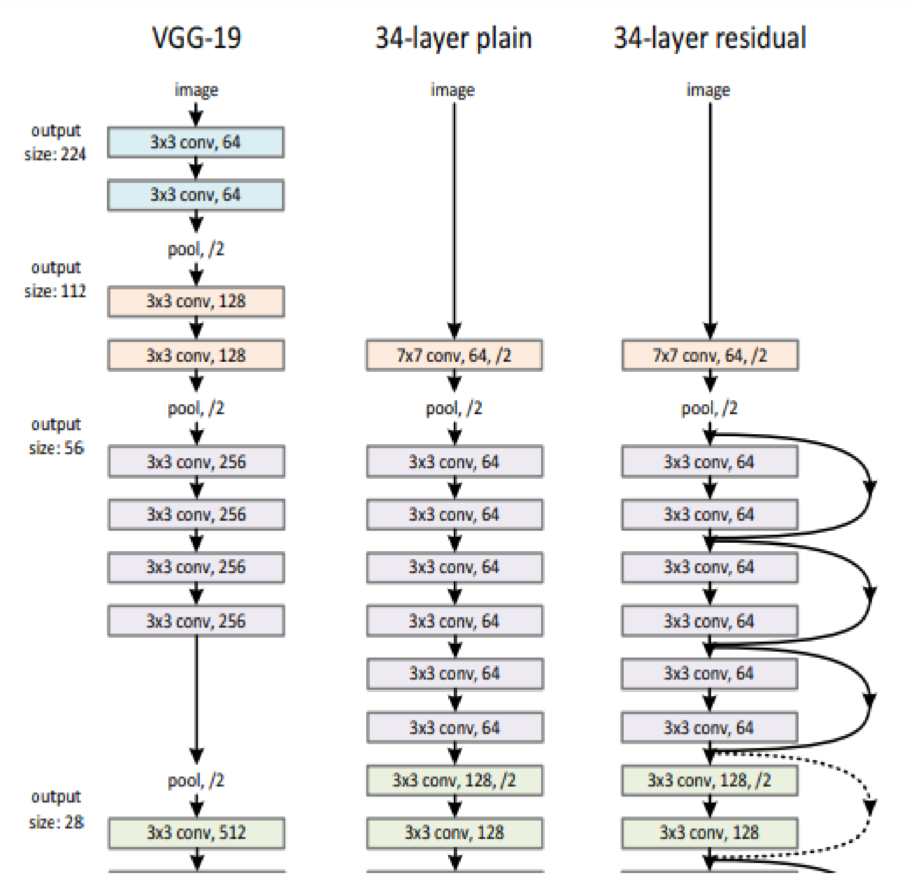
Left: Conventional layers block

Right: **Residual** layer block

To learn identity *f(x) = x*, layers now need to learn *f(x) = 0* ➔ easier

ResNet Architecture

- **Idea**: Residual (skip) connections help make learning easier
- Example architecture:
- Note: residual connections
  - Every two layers for ResNet34
- Vastly better performance
  - No additional parameters!
  - Records on many benchmarks



He et al: “Deep Residual Learning for Image Recognition”

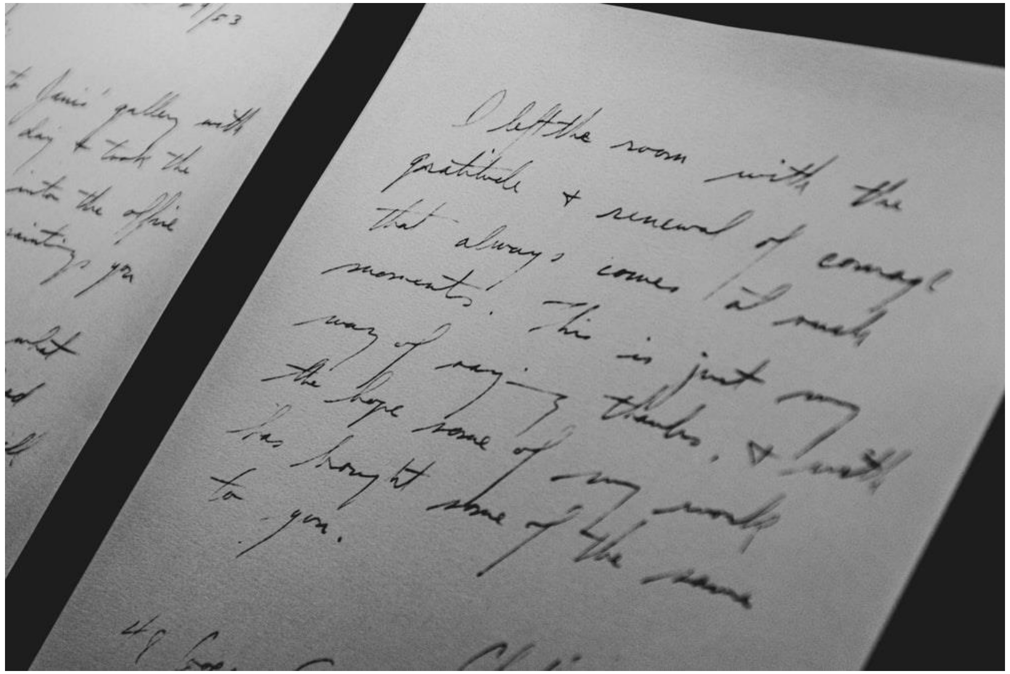
Break & Quiz

Outline

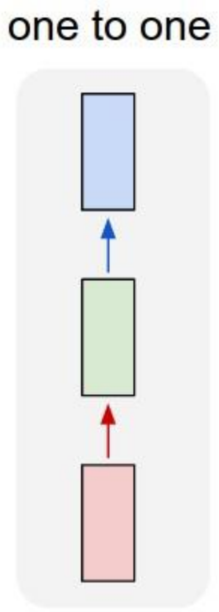
- **CNN Tasks & Architectures**
  - MNIST, ImageNet, LeNet, AlexNet, ResNets
- **RNN Basics**
  - Sequential tasks, hidden state, vanilla RNN
- **RNN Variants + LSTMs**
  - RNN training, variants, LSTM cells

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?

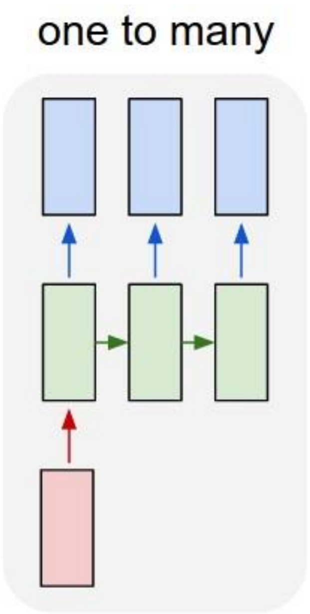


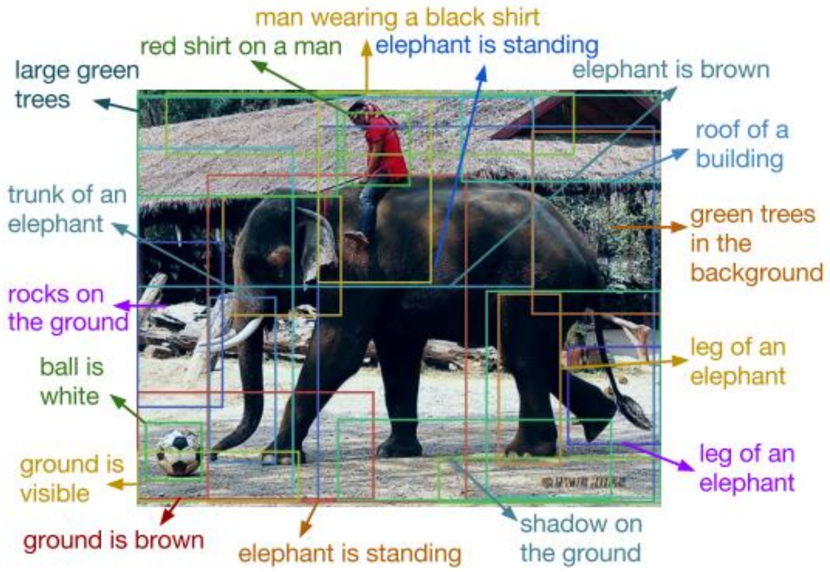
Tasks We Can Handle?



- Our standard model so far. One fixed input type, one output
  - Image classification

Tasks We Can Handle?

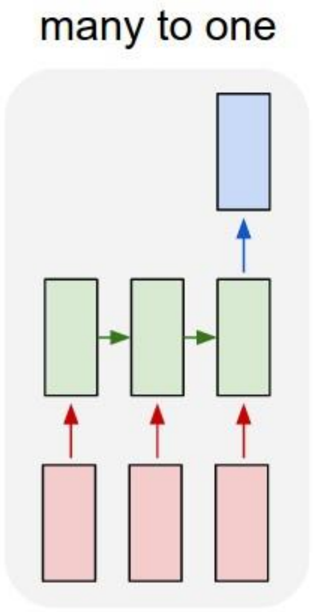


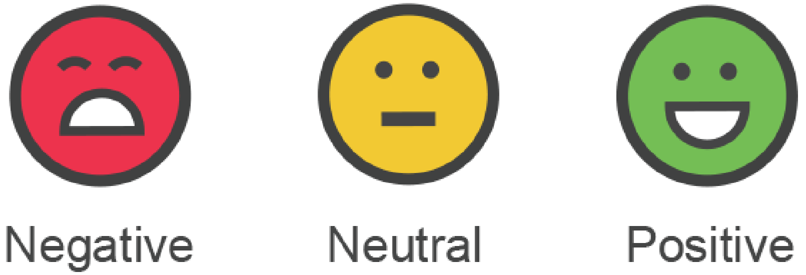


"DenseCap: Fully Convolutional Localization Networks for Dense Captioning", Johnson, Karpathy, Li

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

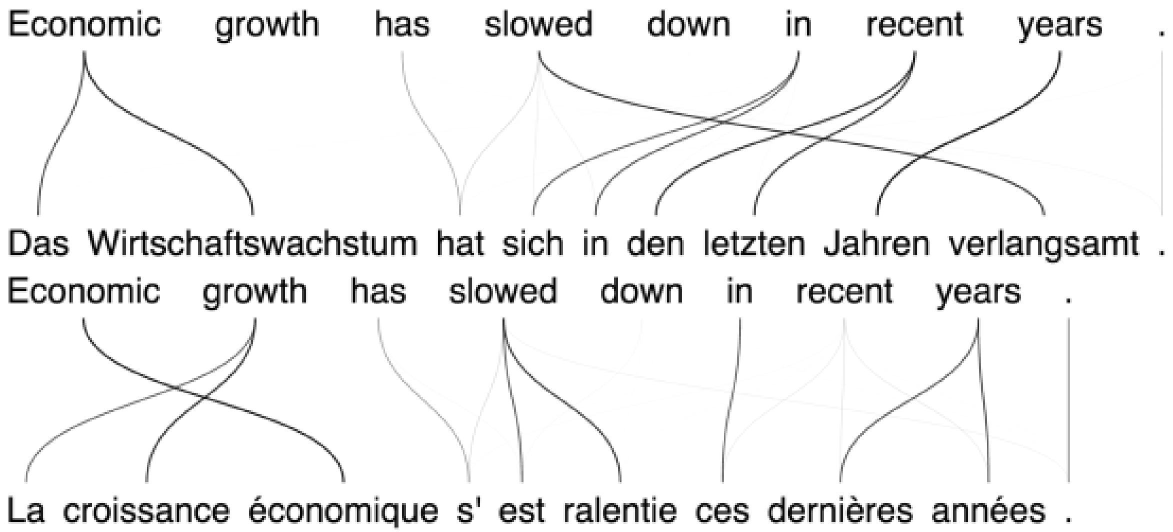
Tasks We Can Handle?



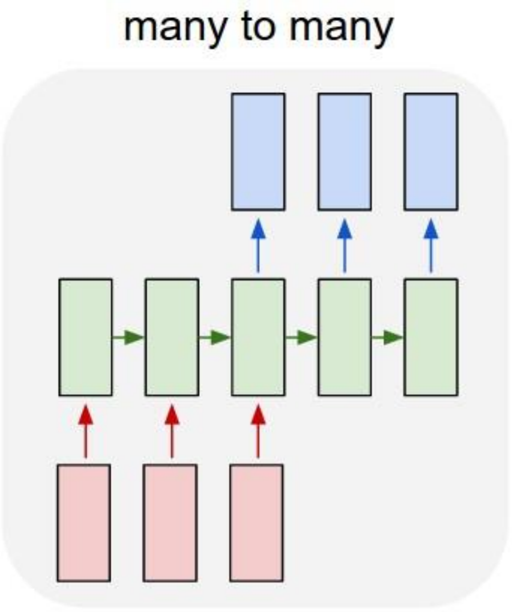


- Sequence input, one output
  - **Ex**: sentiment analysis. Input is a sentence, output is one of {positive, neutral, negative}

Tasks We Can Handle?

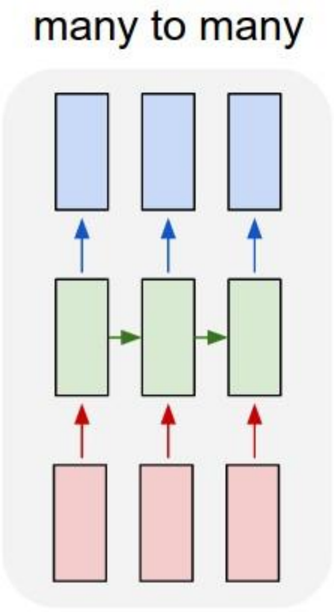


devblogs.nvidia.com



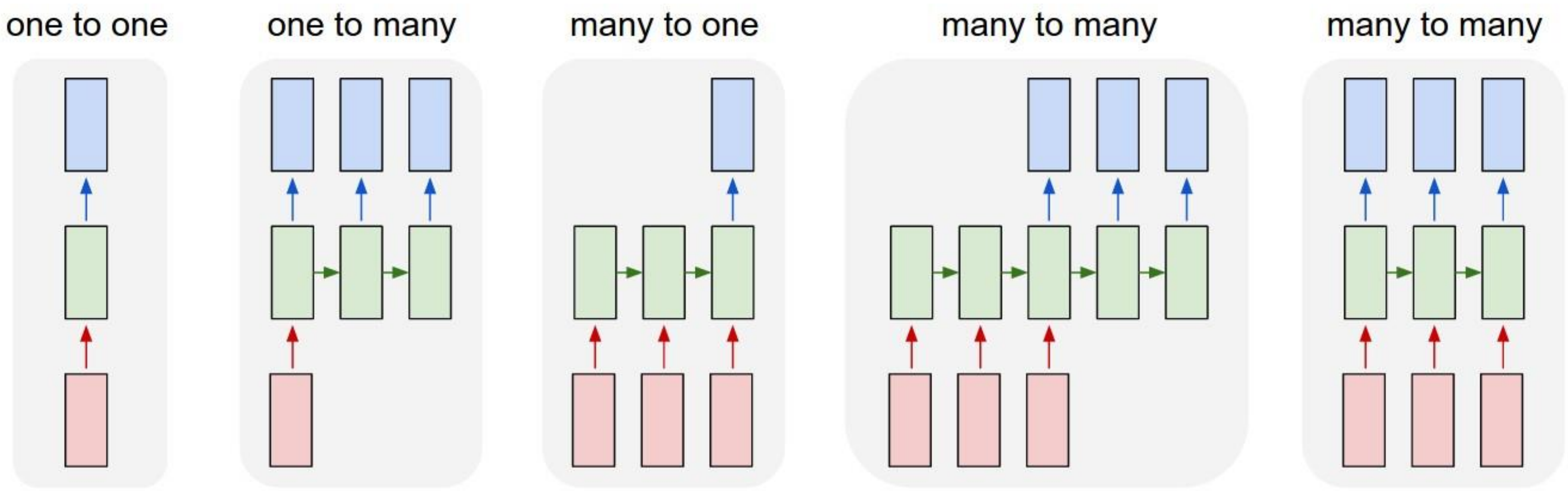
- Sequence input, sequence output
  - **Ex:** machine translation. Translate from language A to language B

Tasks We Can Handle?



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

Tasks We Can Handle?

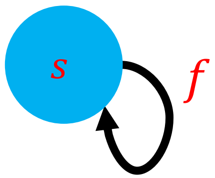


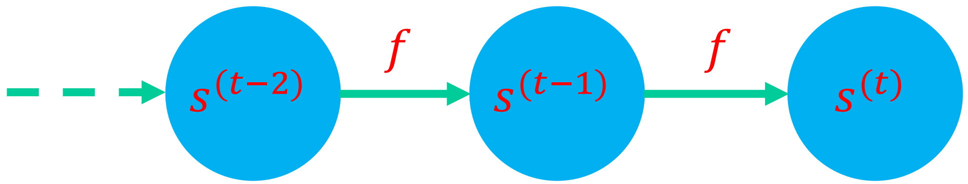
- Don’t have the ability to do anything except (1) so far…
  - Need a new kind of model

Modeling Sequential Data

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

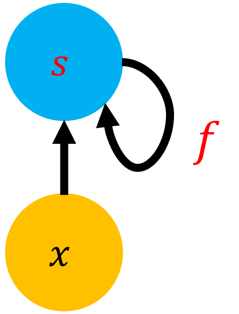
$$s^{(t+1)} = f(s^{(t)}; \theta)$$



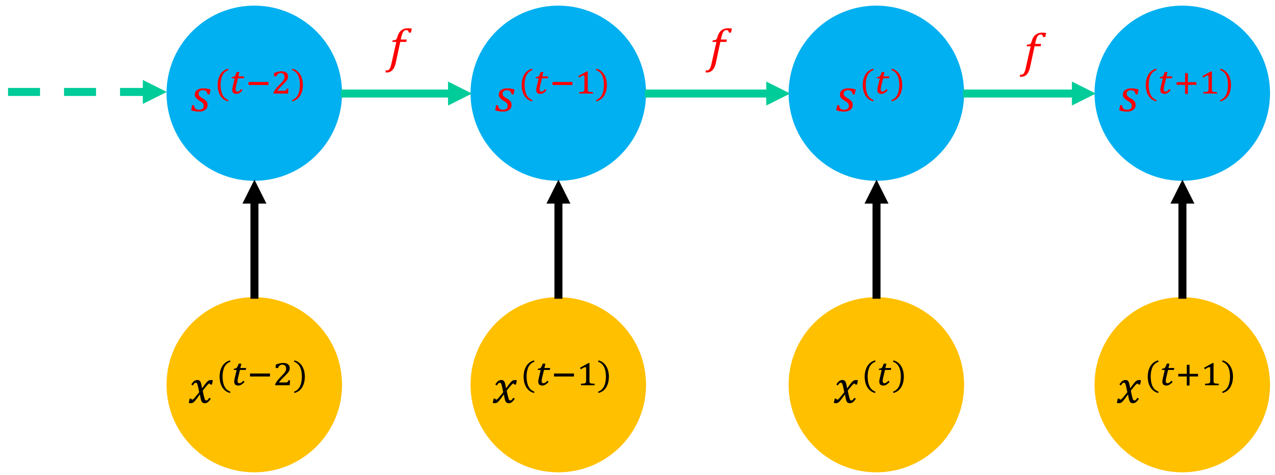


Modeling Sequential Data: External Input

- 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 f and θ for all time steps

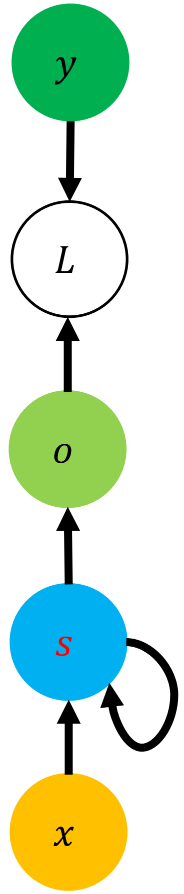
Recurrent Neural Networks

- 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

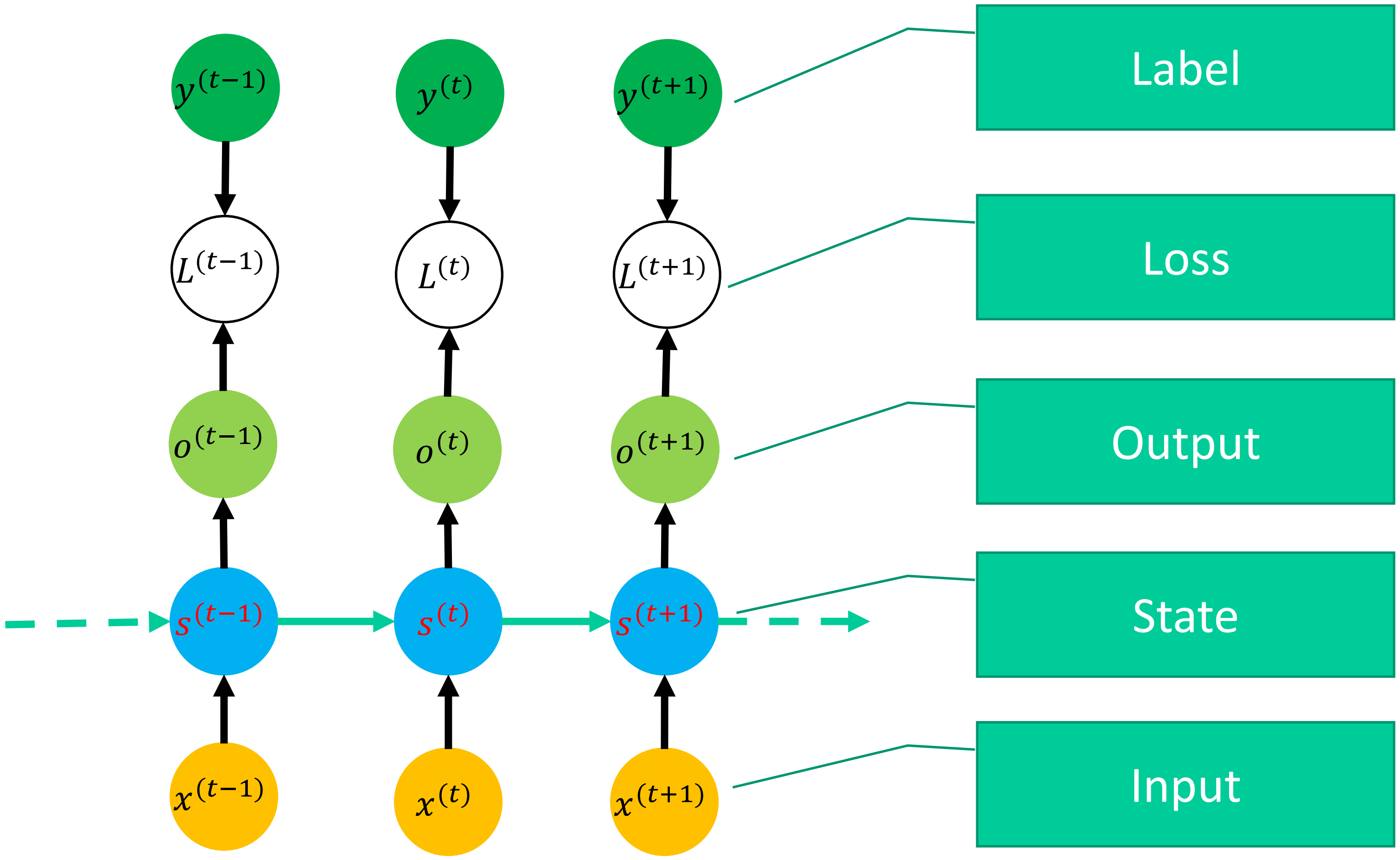
RNNs: Basic Components

- 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

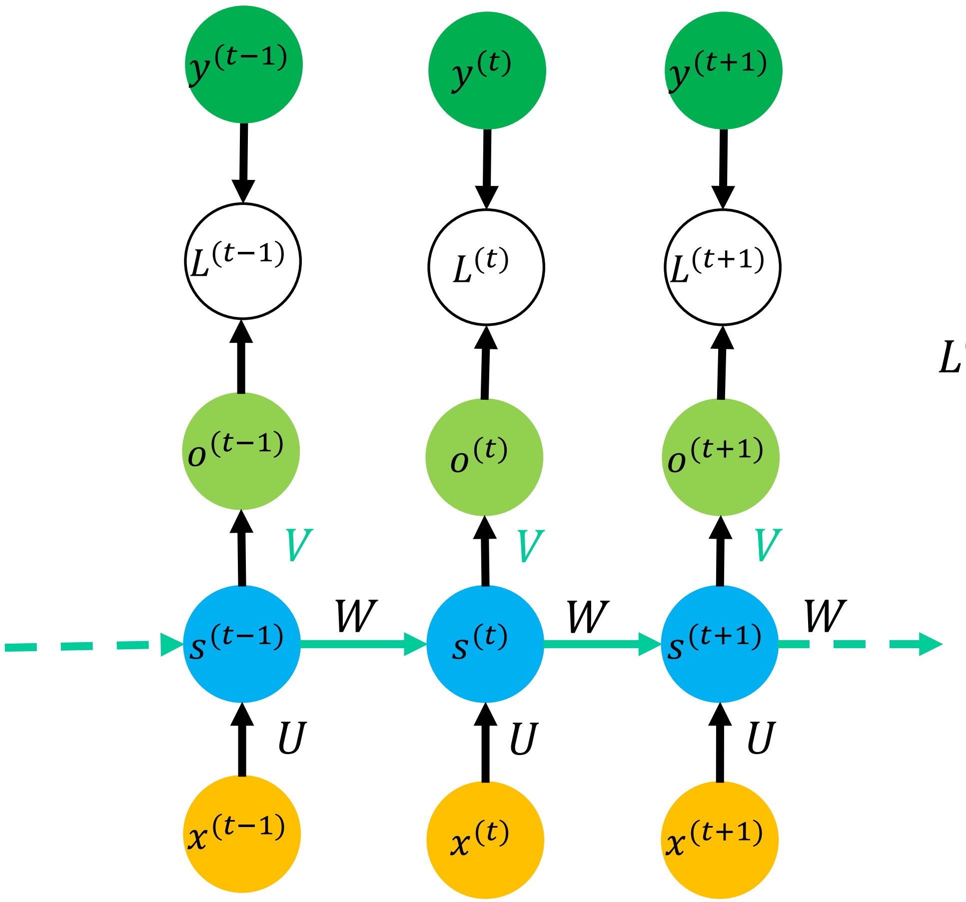


RNNs: Unrolled Graph



Simple RNNs

- Classical RNN variant:



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

Properties

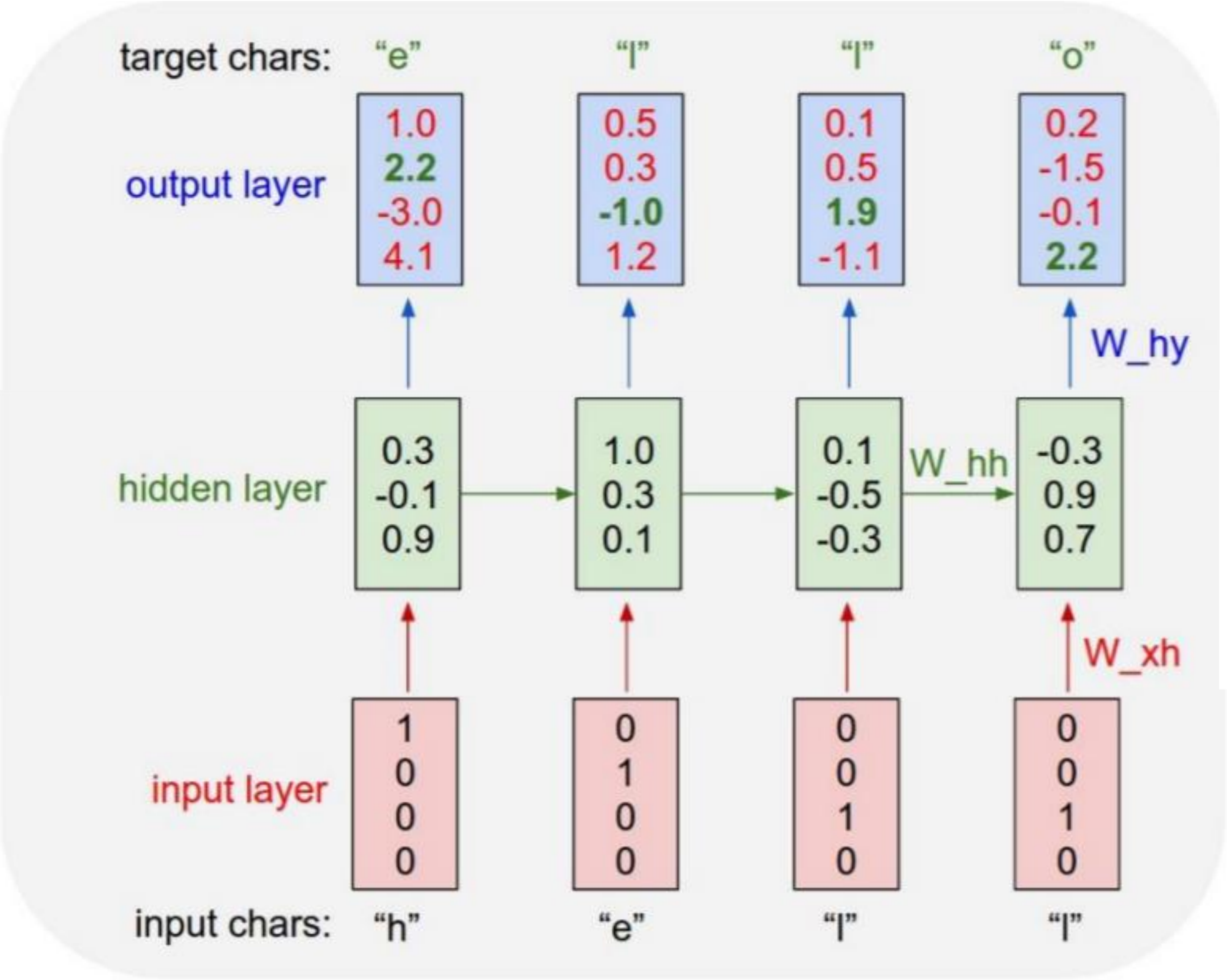
- **Hidden state**: a lossy summary of the past
- Shared functions / parameters
  - Reduce the capacity and good for **generalization**
- Uses the knowledge that sequential data can be processed in the same way at different time step
- Powerful (**universal**): any function computable by a Turing machine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



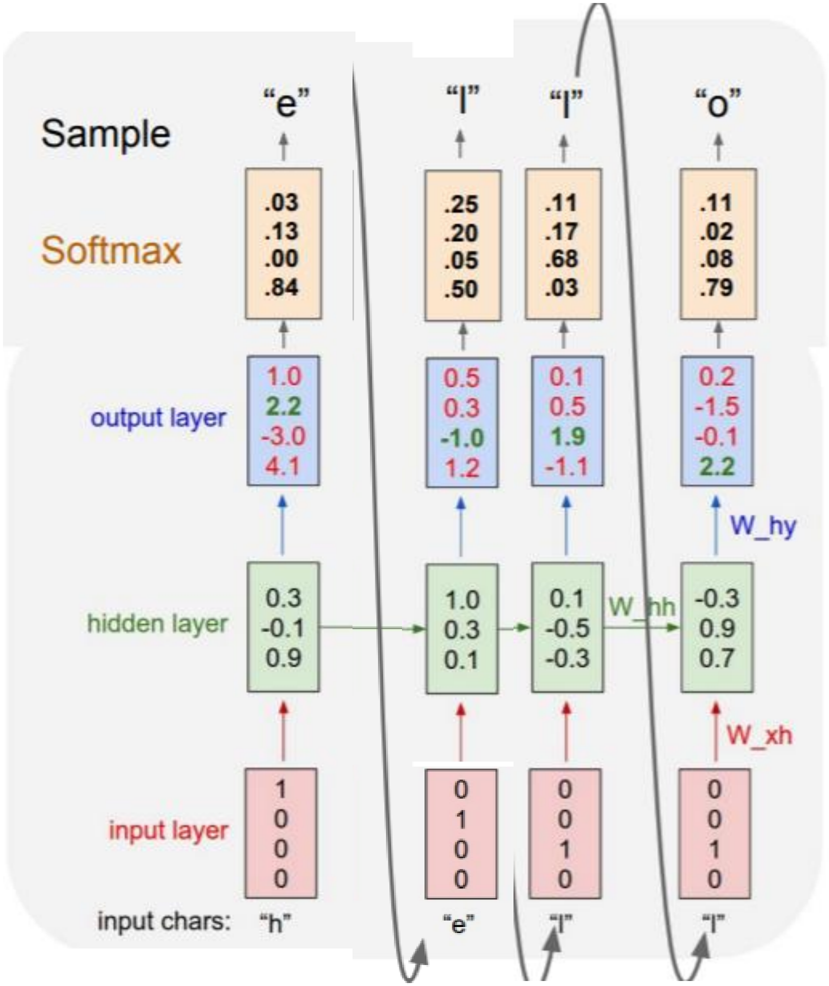
Stanford CS231N

Example: Char. Level Language Model

- LM goal: predict next character:

- Vocabulary
  {h,e,l,o}

- **Test** time:
  - Sample chars, feed into model



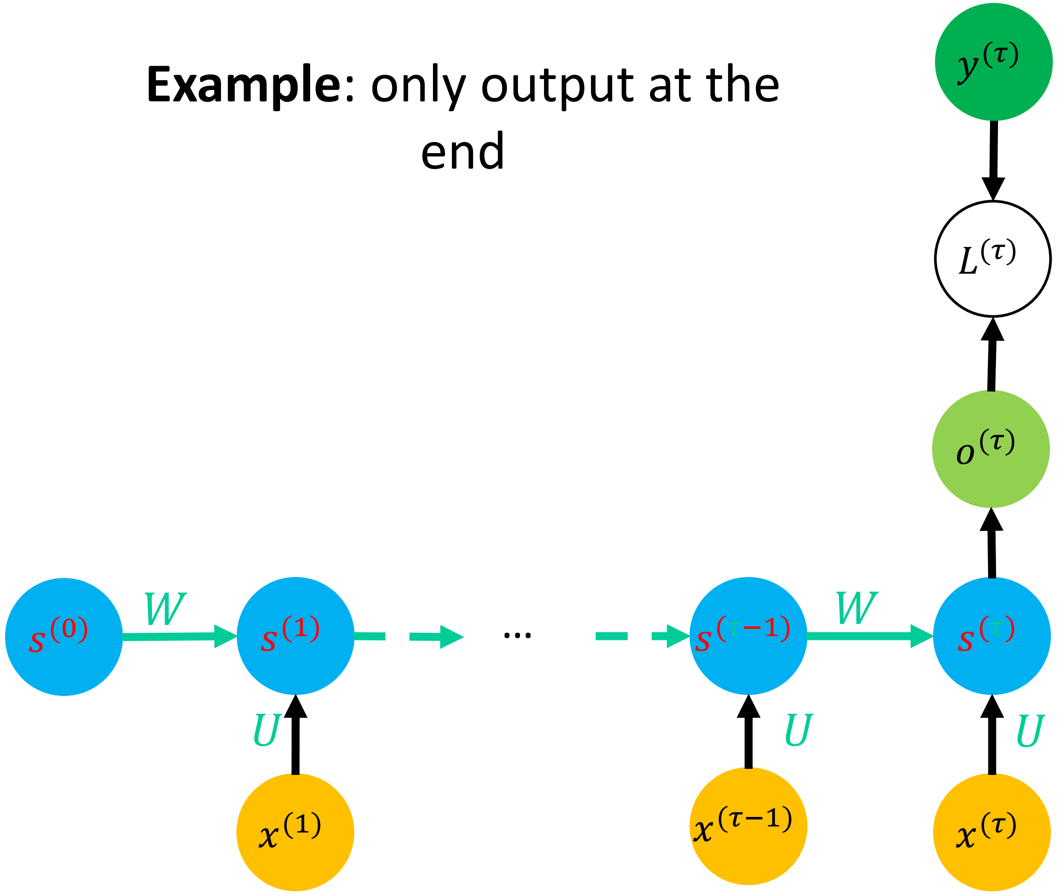
Break & Quiz

Outline

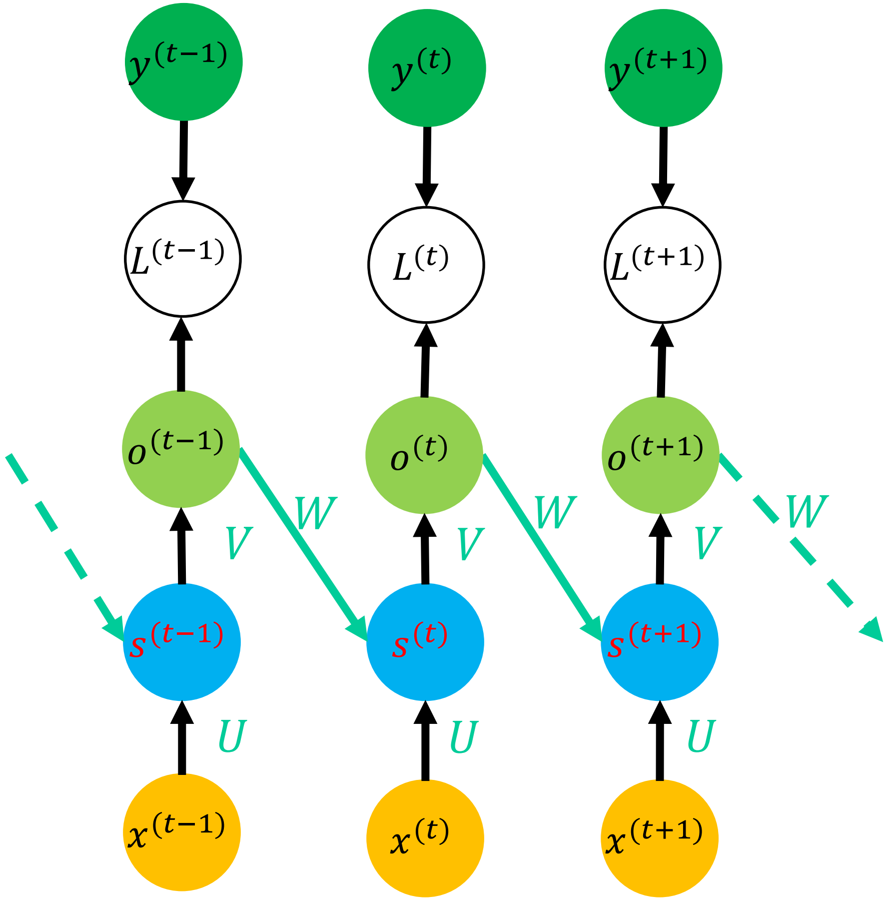
- **CNN Tasks & Architectures**
  - MNIST, ImageNet, LeNet, AlexNet, ResNets
- **RNN Basics**
  - Sequential tasks, hidden state, vanilla RNN
- **RNN Variants + LSTMs**
  - RNN training, variants, LSTM cells

RNN Variants

Example: only output at the end



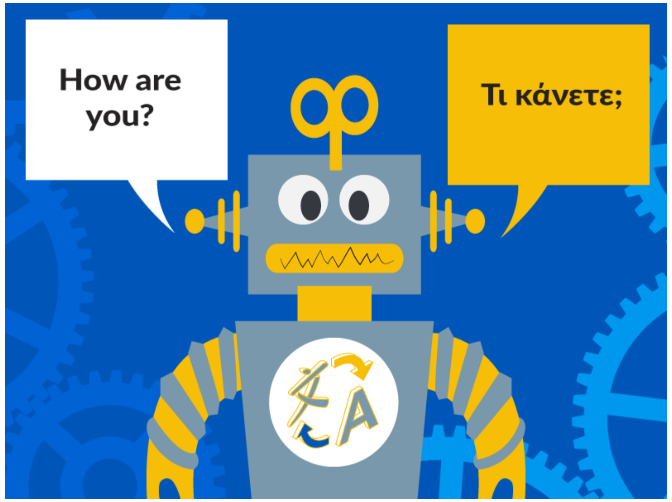
Example: use the output at the previous step



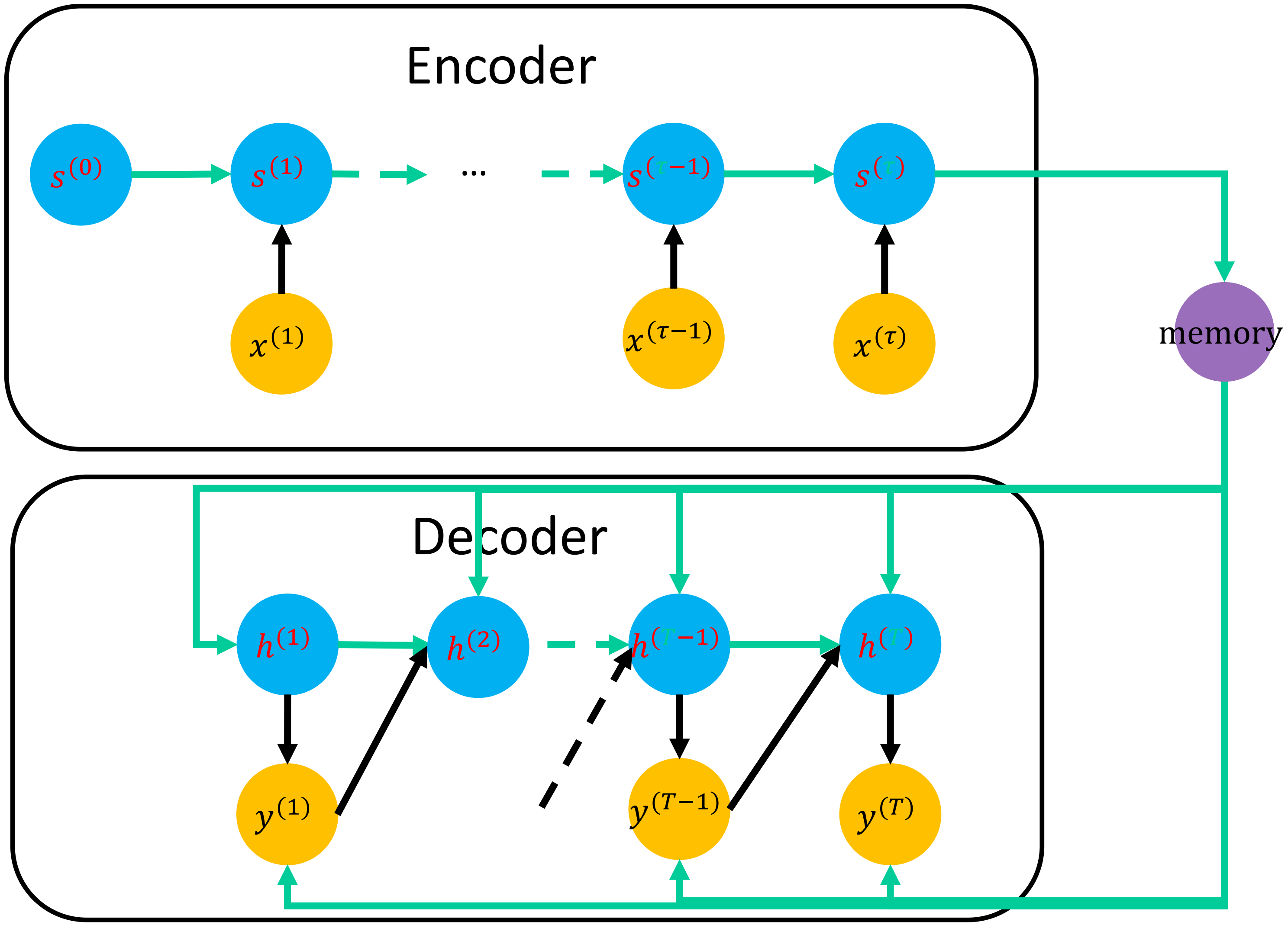
RNN Variants: Encoder/Decoder

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

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



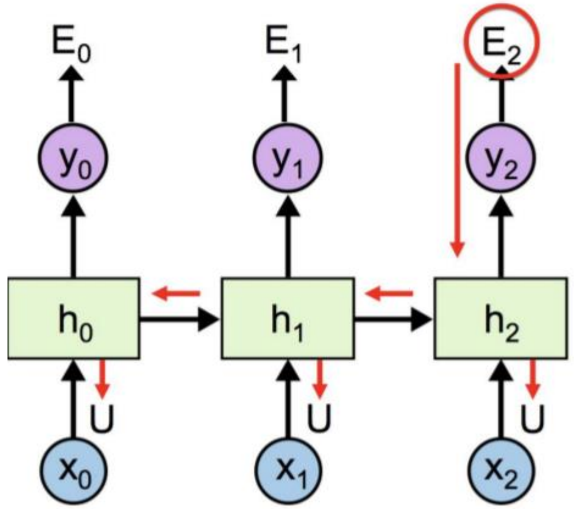
RNN Variants: Encoder/Decoder



Training RNNs

- 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



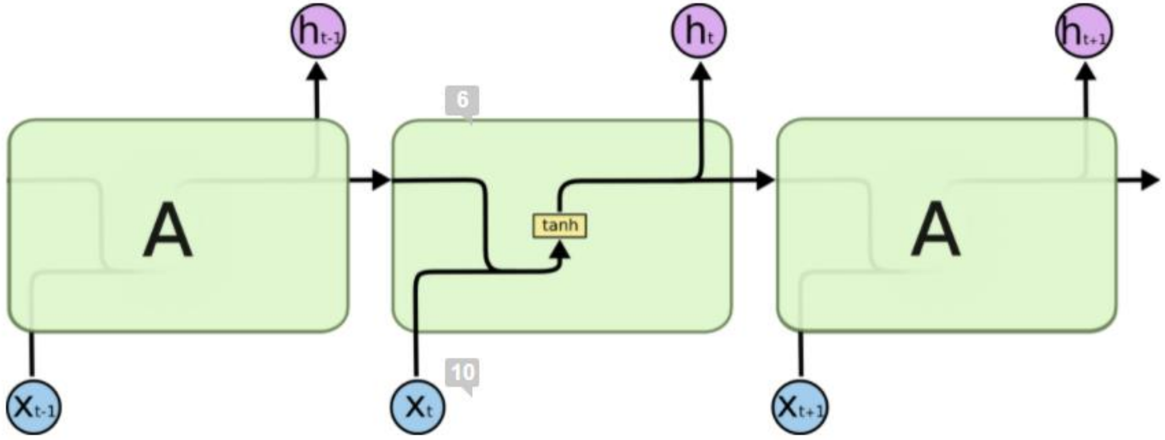
$$\frac{\partial E_2}{\partial U} = \frac{\partial E_2}{\partial h_2}\left(x_2^T + \frac{\partial h_2}{\partial h_1}\left(x_1^T + \frac{\partial h_1}{\partial h_0}x_0^T\right)\right)$$

RNN Problems

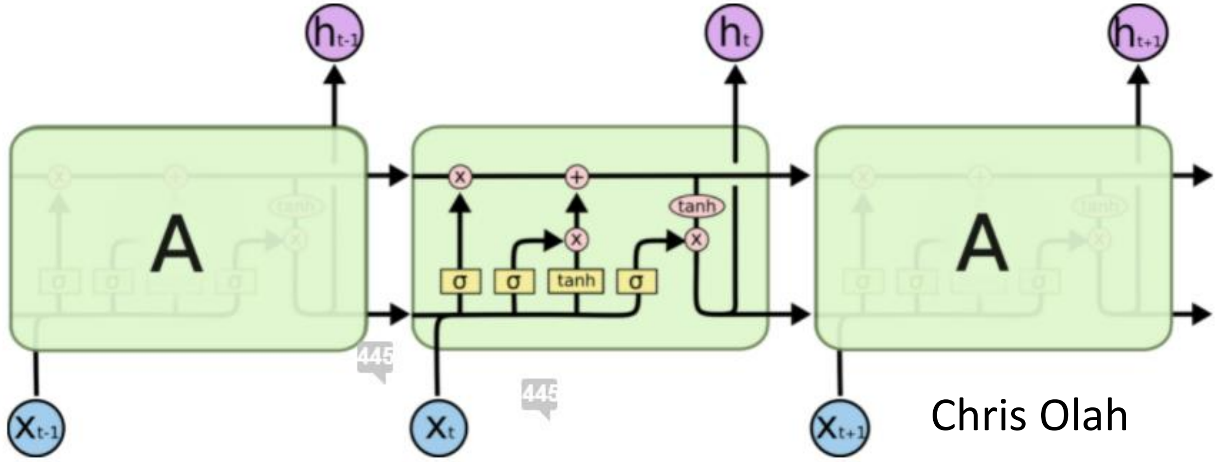
- What happens to gradients in backprop w. many layers?
  - In an RNN trained on long sequences (*e.g.* 100 time steps) the gradients can easily explode or vanish.
  - We can avoid this by initializing the weights very carefully.
- Even with good initial weights, very hard to detect that current target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

LSTM Architecture

- RNN: can write structure as:



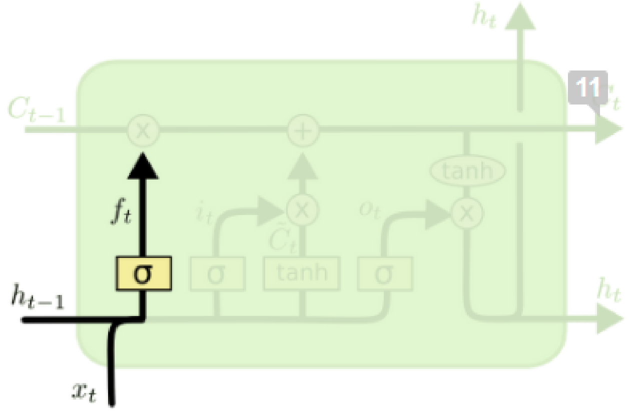
- Long Short-Term Memory: deals with problem. Cell:



Chris Olah

Understanding the LSTM Cell

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

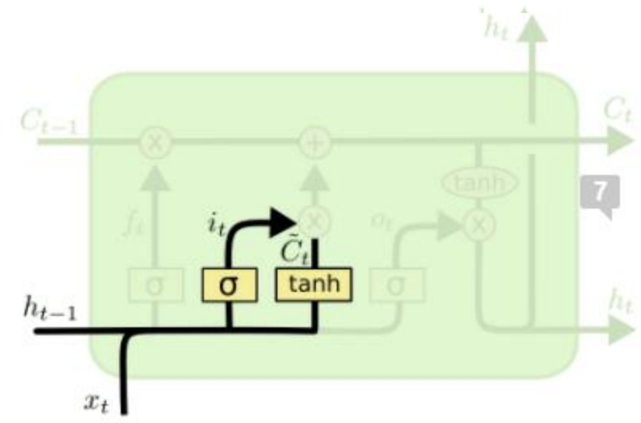


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

- “**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

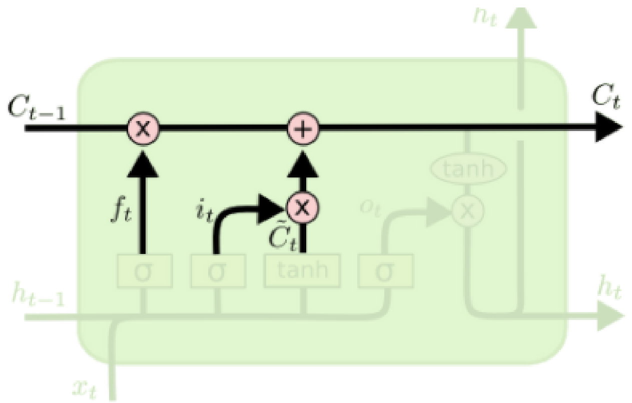


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \;+\; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \;+\; b_C)$$

- **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

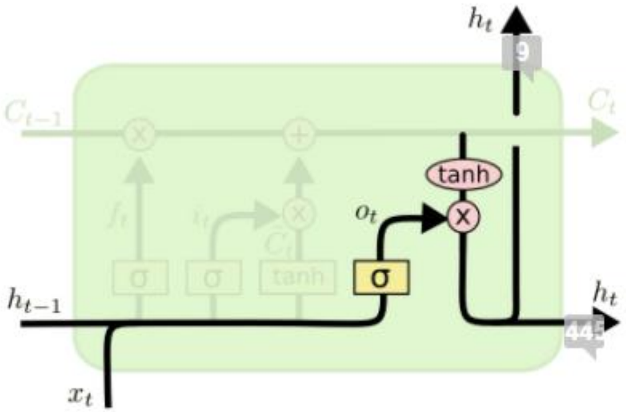


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

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

Understanding the LSTM Cell

- Step-by-step



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

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

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