Probabilistic Graphical Models & Probabilistic AI

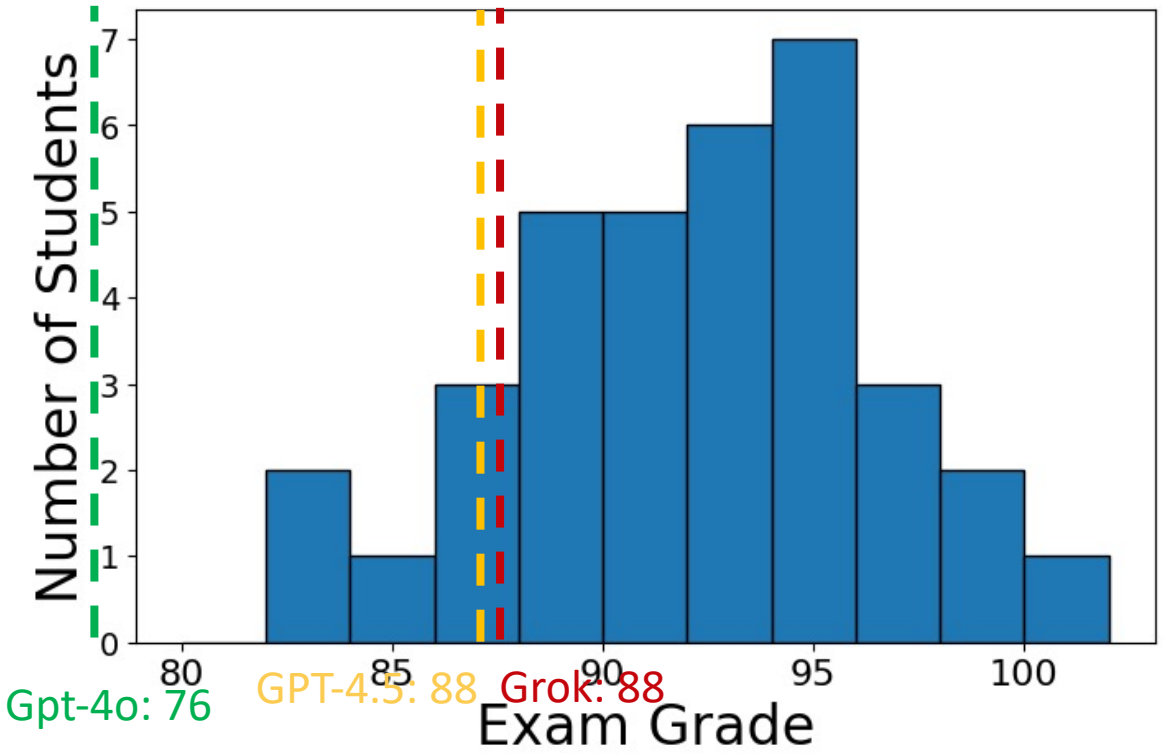
Ben Lengerich

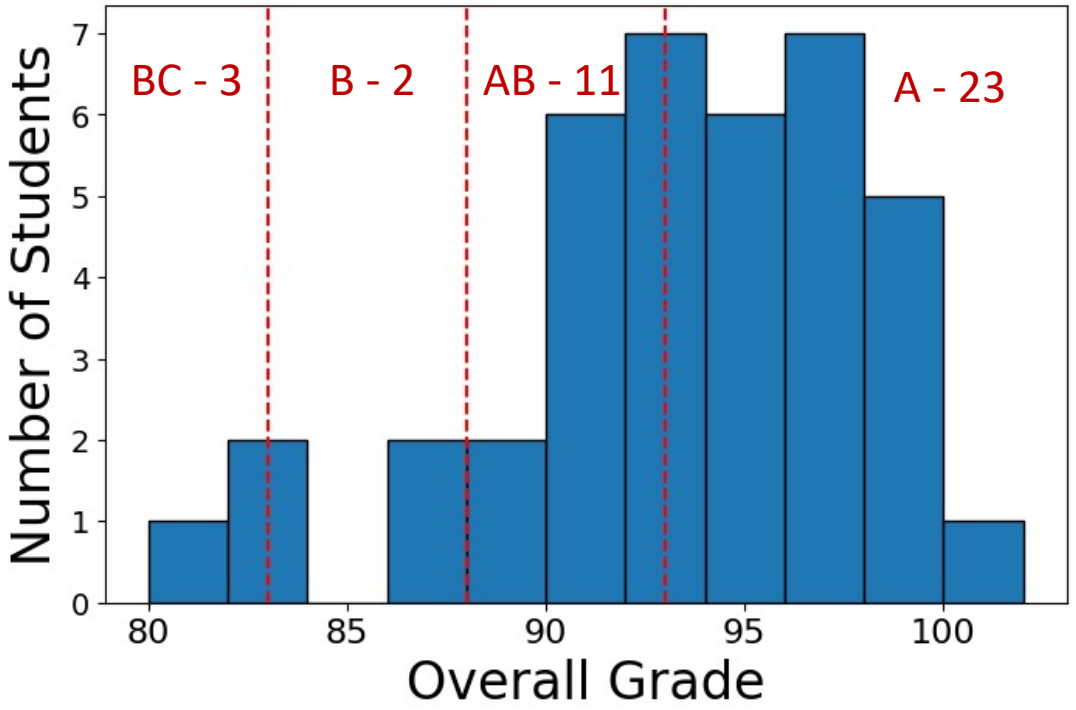
Lecture 17: CNNs, RNNs, Autoencoders

April 3, 2025

Reading: See course homepage

Exam Grades

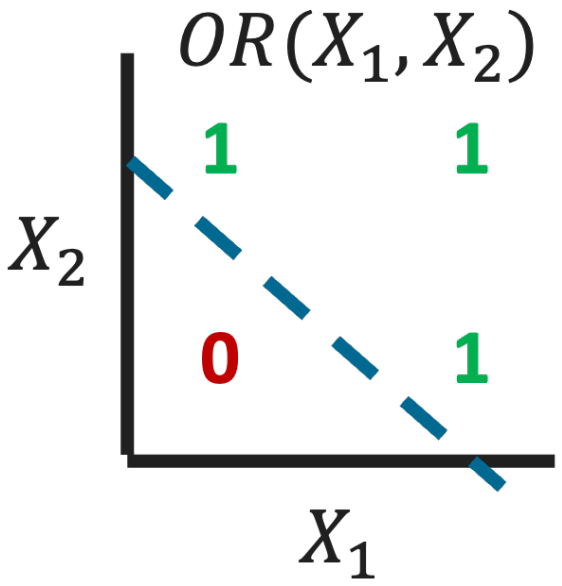


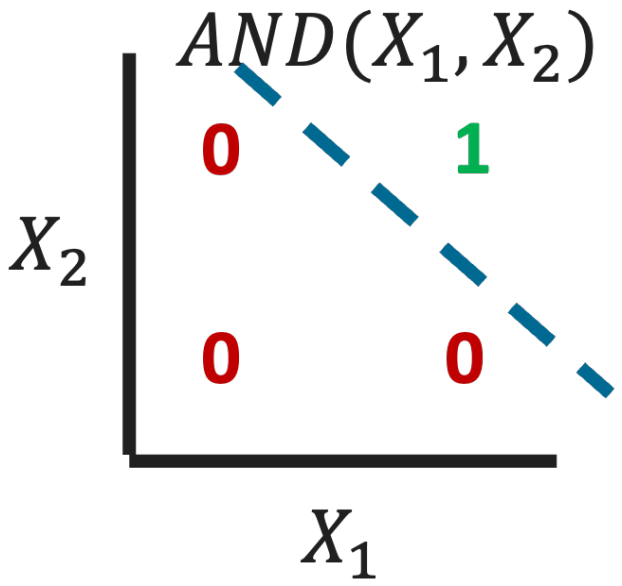


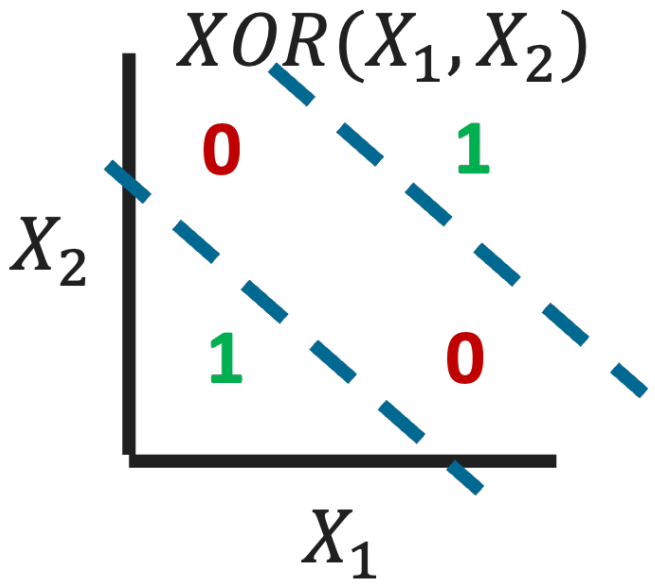
Outstanding graded material

- Project midterm report (5%, 4/11)
- Project presentation (5%, 4/29, 5/1)
  - Sign up here!
- Project final report (15%, 5/5)
- Extra credit (3%, sign-up)

Recap: Perceptron Decision Boundaries





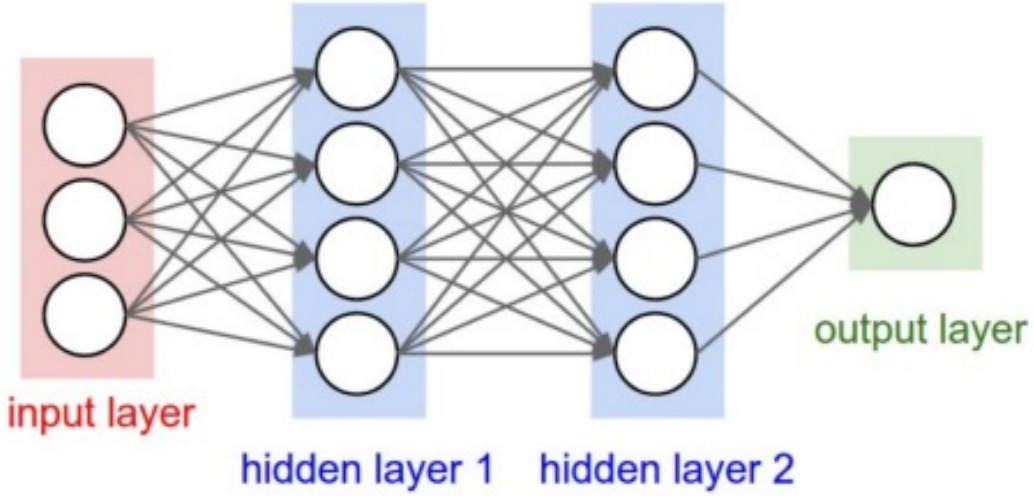


Today

- CNNs
- RNNs
- Autoencoders

Convolutional Neural Networks (CNNs)

Full connectivity is a problem for large inputs

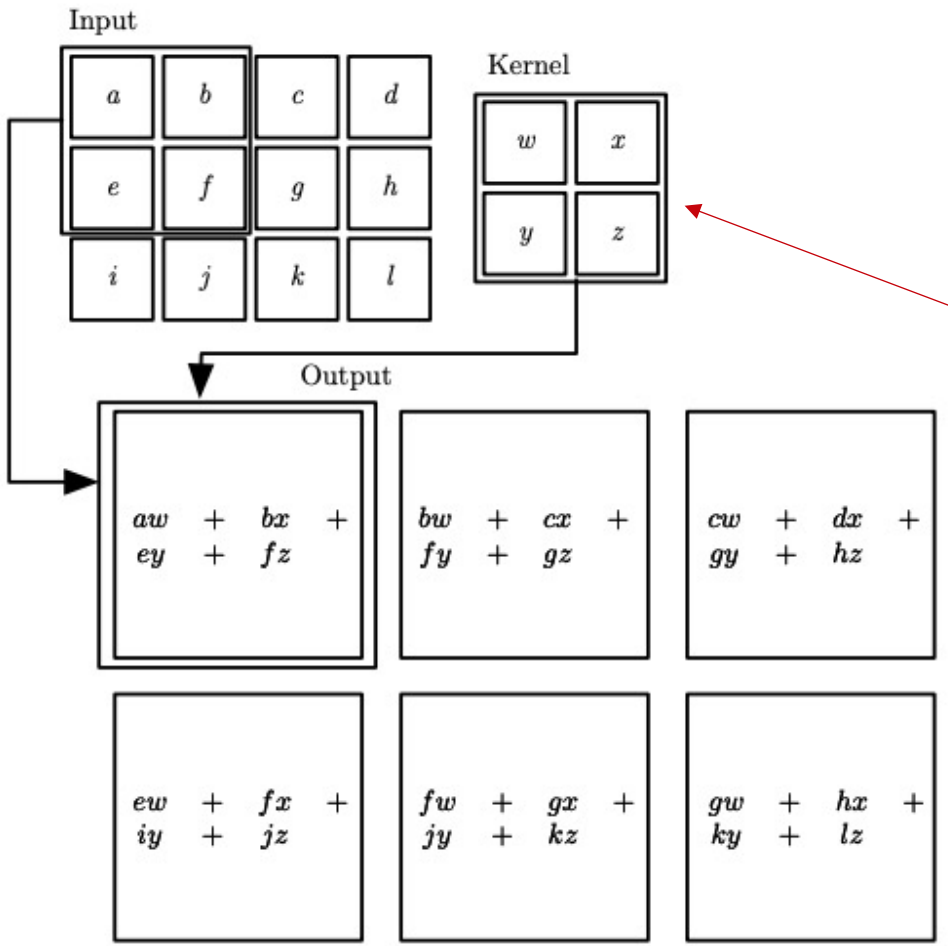


- 200x200x3 images imply **120,000** weights per neuron in first hidden layer

Convolutional Neural Networks [LeCun 1989]

- If inputs are images (and more generally, data with a grid-like topology) then we can share parameters.
- Instead of learning position-specific weights, learn weights defined for **relative positions**
  - Learn "filters" that are reused across the image
  - Generalize across spatial translation of input
- Key idea:
  - Replace matrix multiplication in neural networks with a convolution

Convolutional Neural Networks [LeCun 1989]



Sliding filters (kernels)

Reused weights (small)!

Fig. Goodfellow et al. 2016

CNNs give sparse connectivity

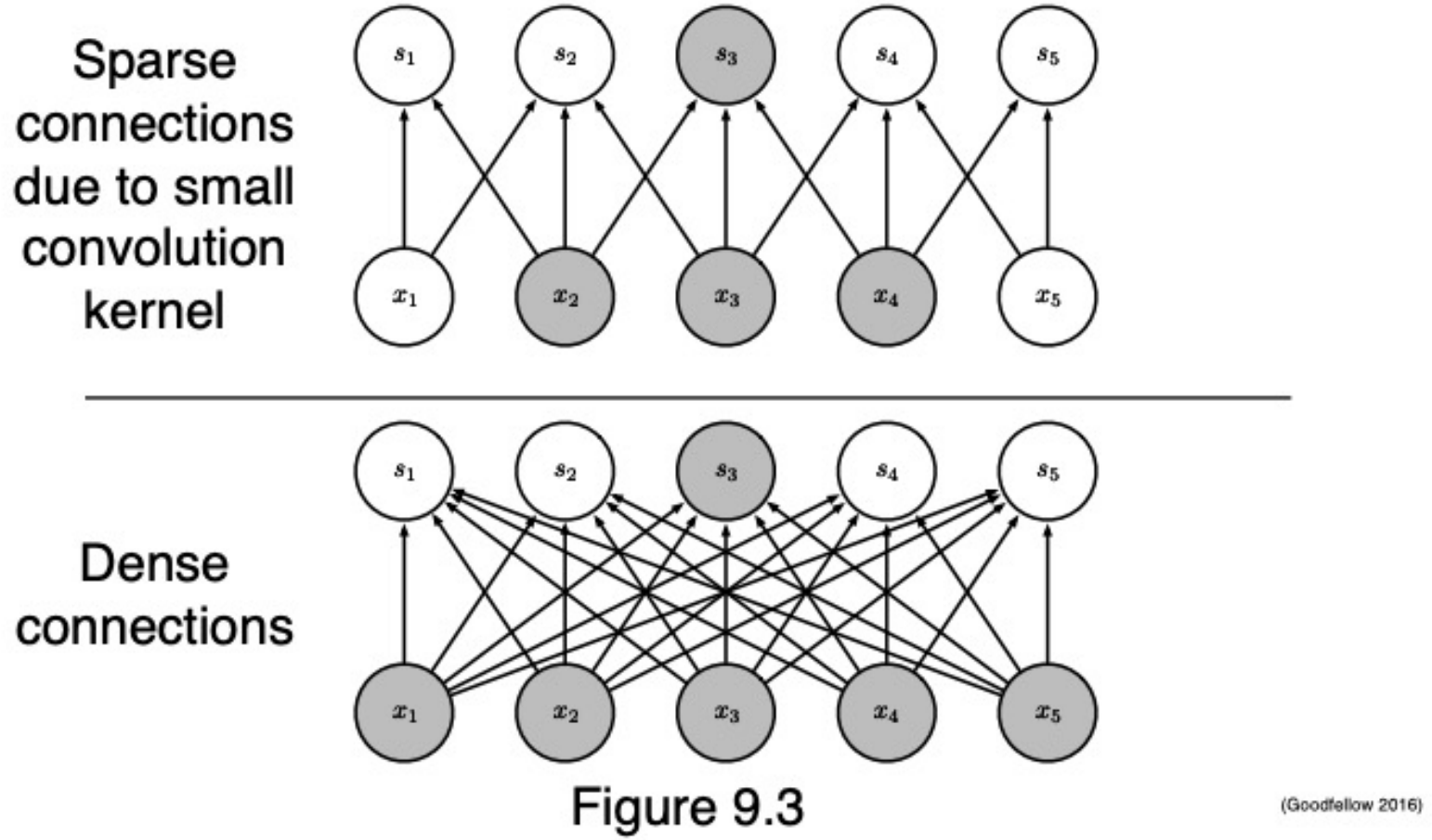


Figure 9.3

(Goodfellow 2016)

Receptive fields grow over depth

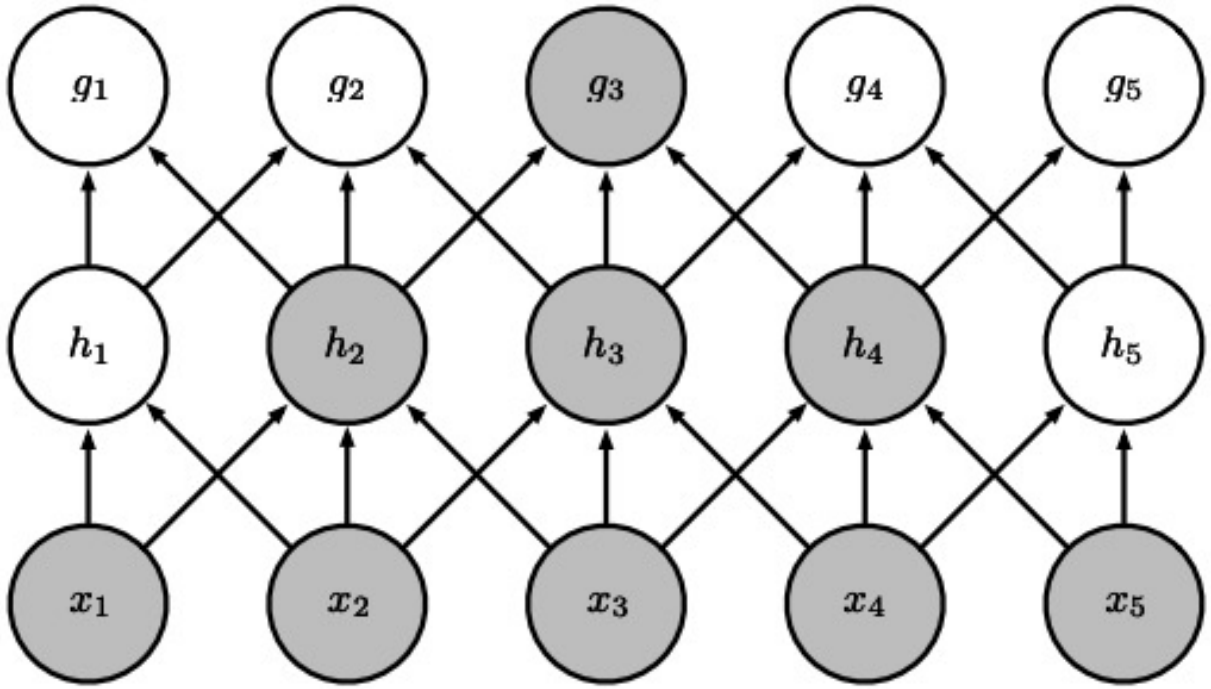


Figure 9.4

(Goodfellow 2016)

Parameter sharing

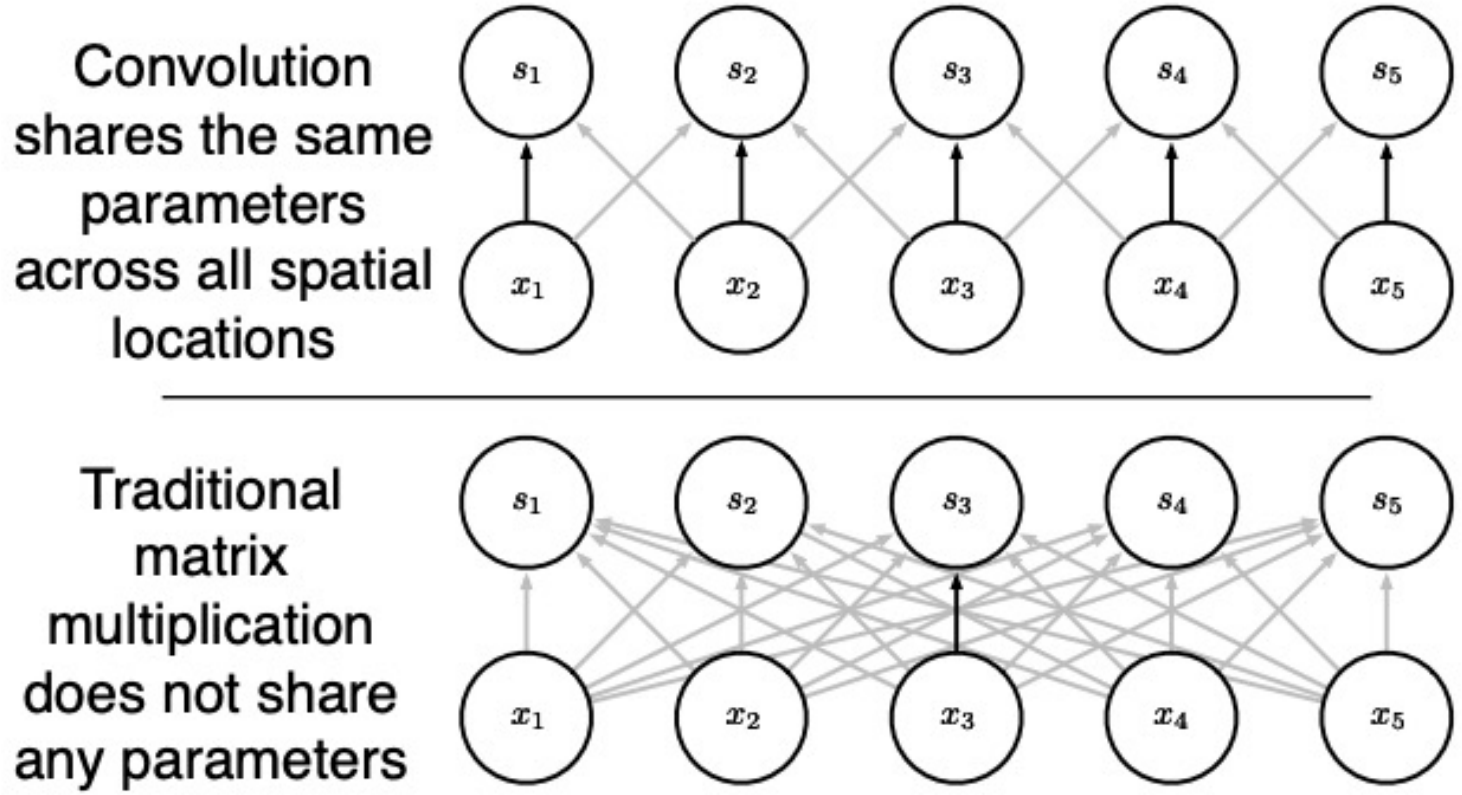


Figure 9.5

(Goodfellow 2016)

A Probabilistic Perspective on CNNs

Convolution: Adding two random variables

- Let X $\sim P_X, Y \sim P_Y$ be independent RVs. What's $E[X] + E[Y]$?
- What's $P(X + Y = z)$?

$$P(X + Y = z) = \int P(X = x, Y = z - x)dx$$

$$= \int P_X(X = x)P_Y(Y = z - x)dx$$

$$= \int P_X(x)P_Y(z - x)dx$$

- This is known as a **convolution** of P_X and P_Y:

$$(P_X * P_Y)(z) = \int P_X(x)P_Y(z - x)dx$$

“Convolutions” in CNNs

- Let:
  - $P_I(i,j)$ be the probability that a pattern is centered at i,j in the image.
  - $P_K(m,n)$ be the probability that a pattern offset by (m,n) should be recognized.
  - $P_Z(i,j)$ be the probability of recognizing a pattern at i,j considering all relevant offsets.
- We can write:

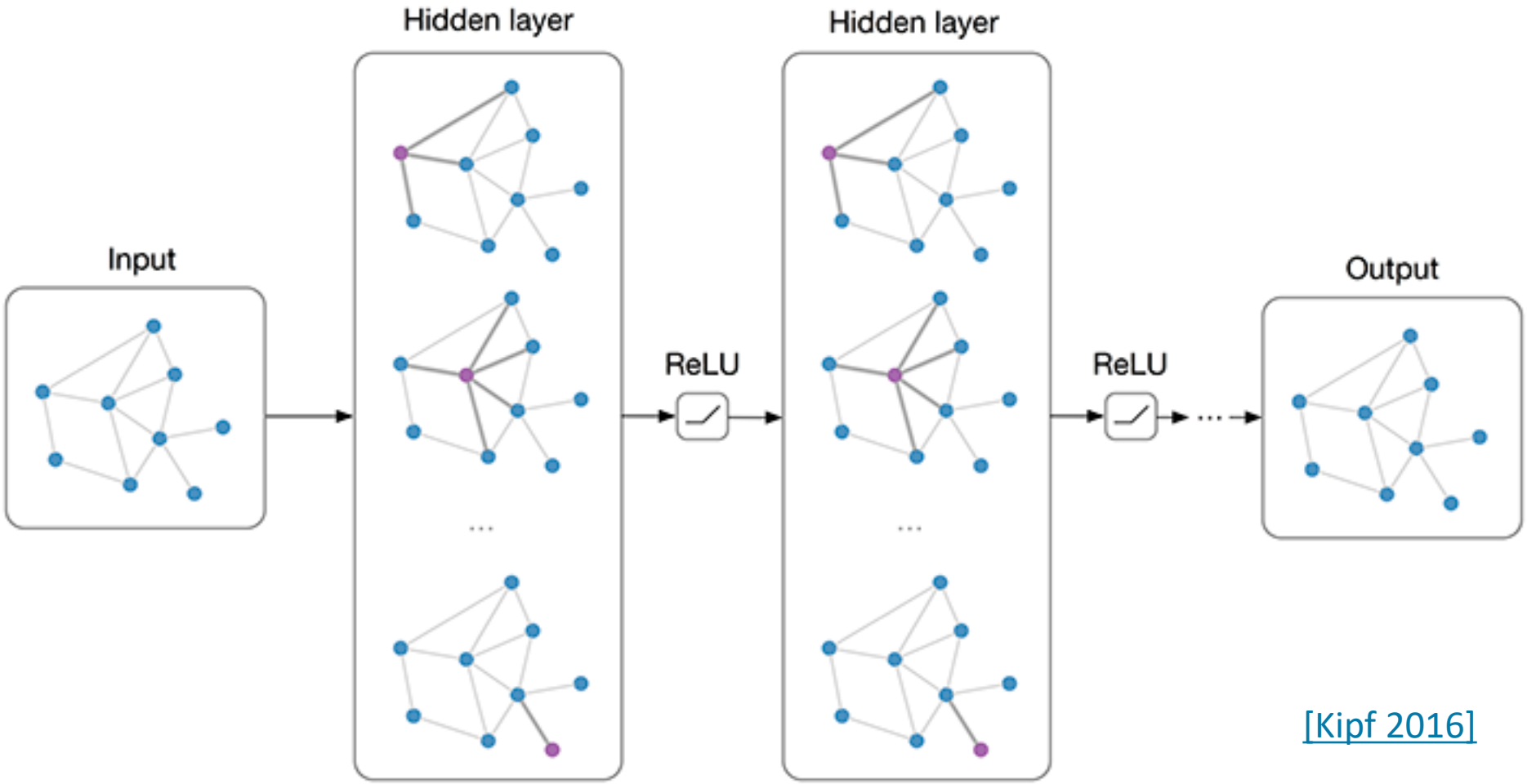
$$P_Z(i,j) = \sum_m \sum_n P_I\big(I = (i+m, j+n)\big) P_L\big(L = (m,n)\big)$$

- Alternatively, define:
  - $X = \log P_I\,, Y = \log P_K$
- Then $\mathrm{Z}(i,j) = \log \sum_m \sum_n \exp\big(X(i+m, j+n) + Y(m,n)\big)$

$$\approx \sum_m \sum_n X(i+m, j+n) Y(m,n) = X * Y$$

Convolutions on non-image data?

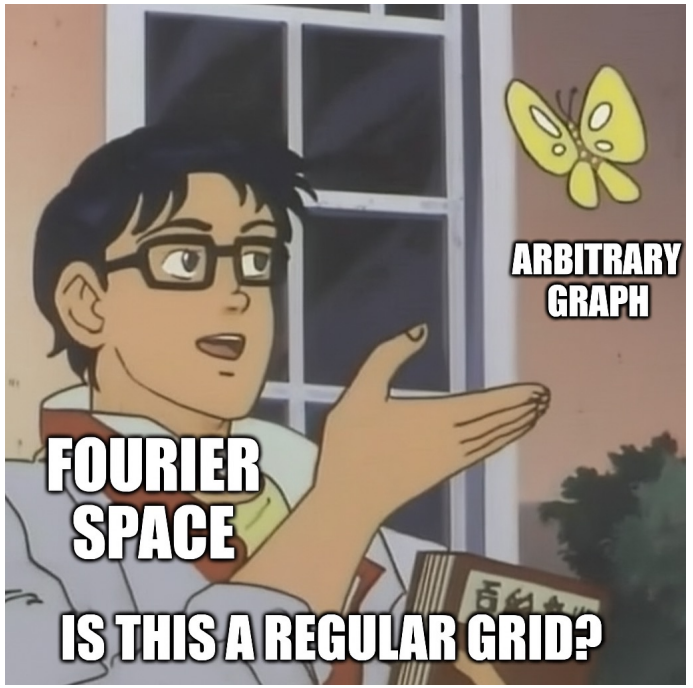
Graph Convolutional Networks



[Kipf 2016]

Spectral convolutions

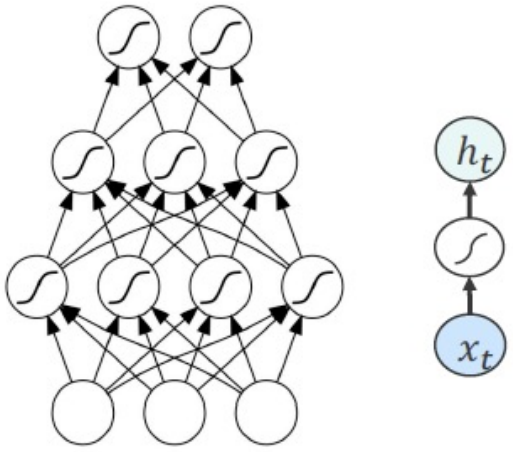
- Spatial domain – graphs may be irregular
- Spectral domain – represent graph by the eigenvectors of the graph Laplacian
  - Orthogonal basis
  - Convolution filter in the spectral domain is equivalent to a localized operation in the spatial domain
  - By designing filters to operate on certain eigenvalues, you can control which features (smooth vs high-frequency) you are retaining.



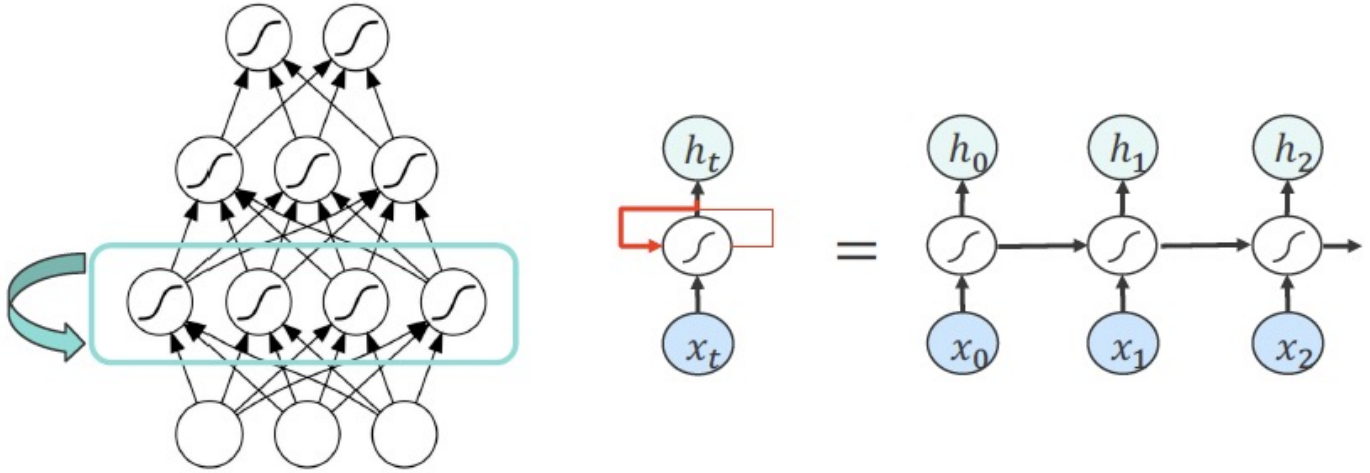
Recurrent Neural Networks (RNNs)

CNNs → Recurrent Neural Networks (RNNs)

- Spatial Modeling *vs.* Sequential Modeling
- Fixed *vs.* variable number of computation steps.

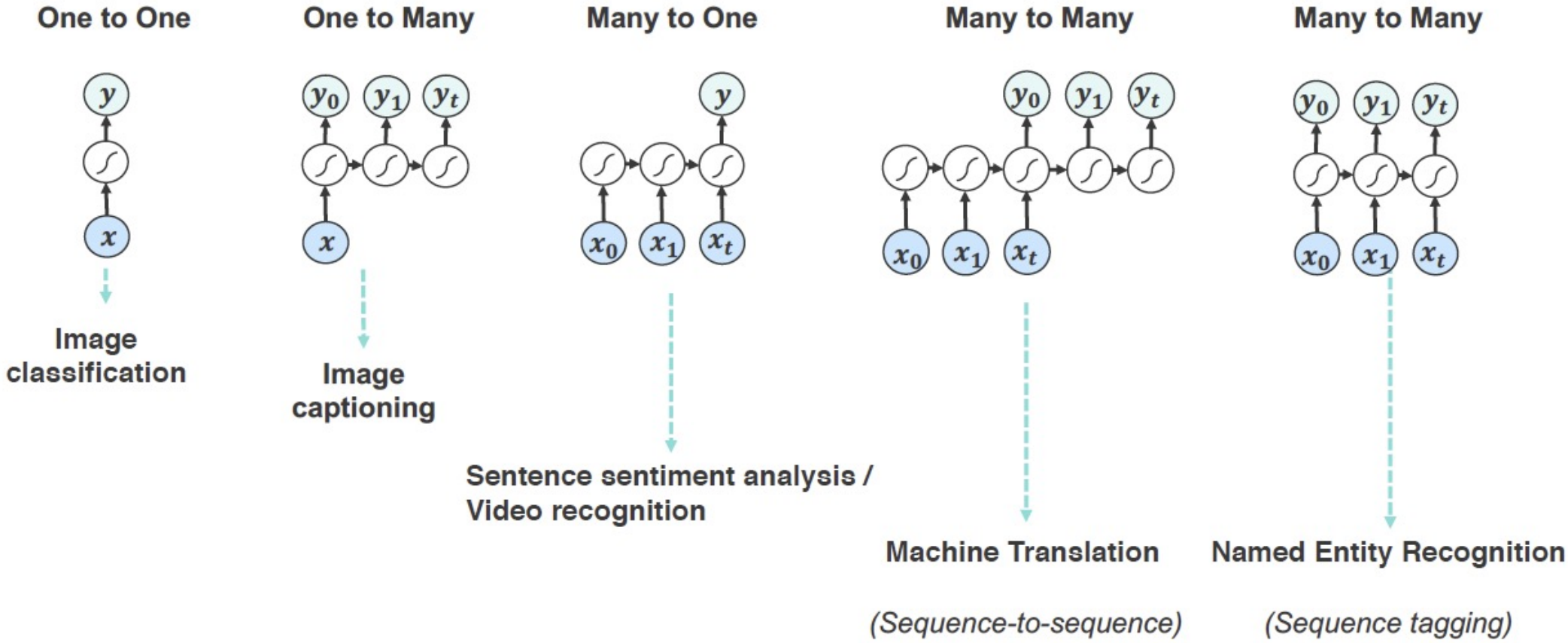


The output depends ONLY on the current input



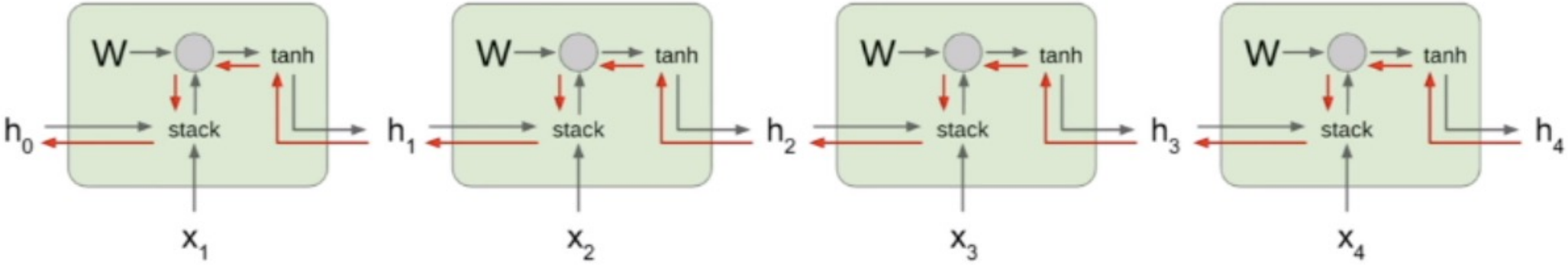
The hidden layers and the output additionally depend on previous states of the hidden layers

RNNs in many forms



A challenge: Vanishing / exploding gradients

$$\boldsymbol{h}_t = tanh(W^{hh}\boldsymbol{h}_{t-1} + W^{hx}\boldsymbol{x}_t)$$



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1: **Exploding gradients** → **Gradient clipping**: Scale gradient if its norm is too big

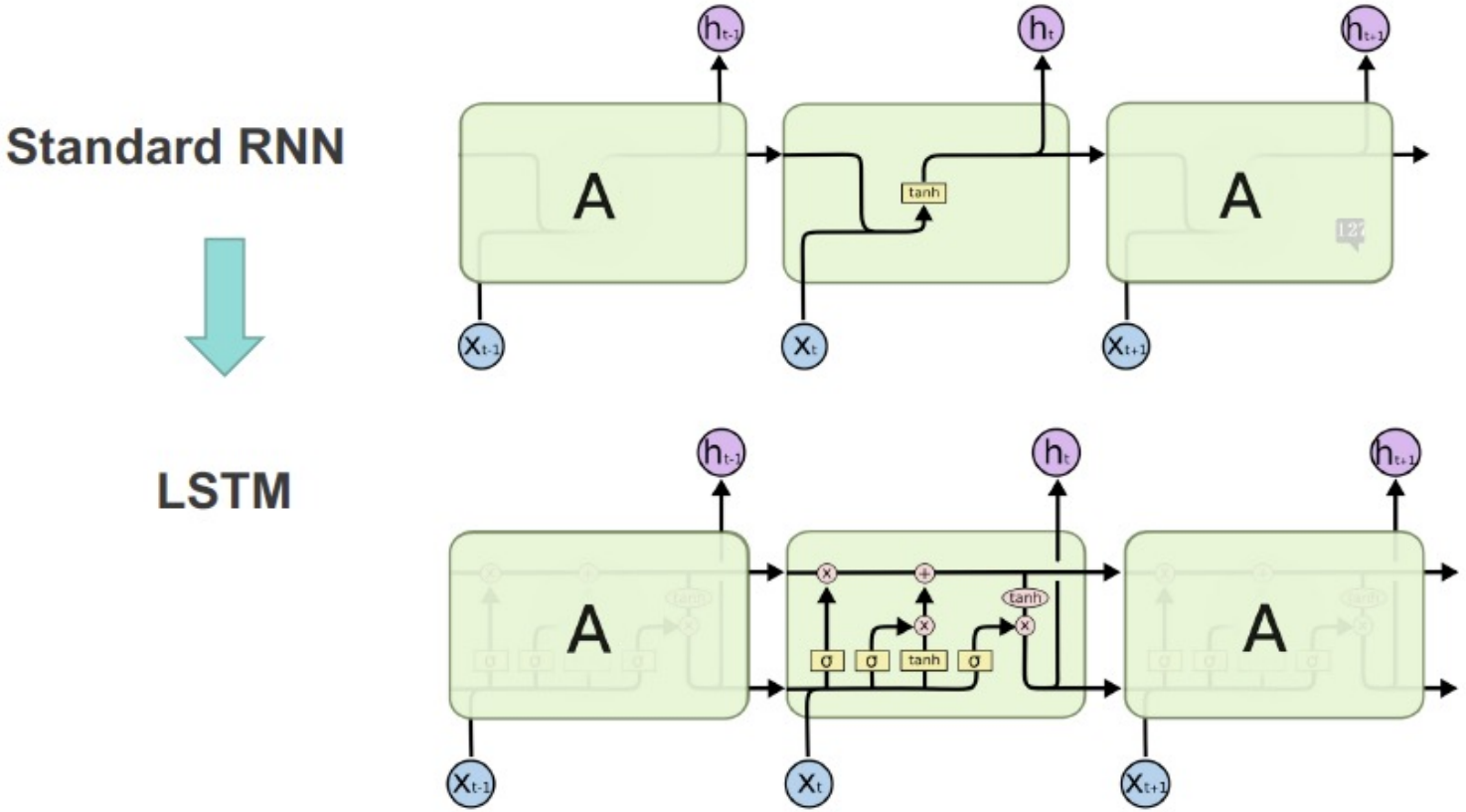
```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

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

Bengio et al., 1994 "Learning long-term dependencies with gradient descent is difficult"
Pascanu et al., 2013 "On the difficulty of training recurrent neural neworks"

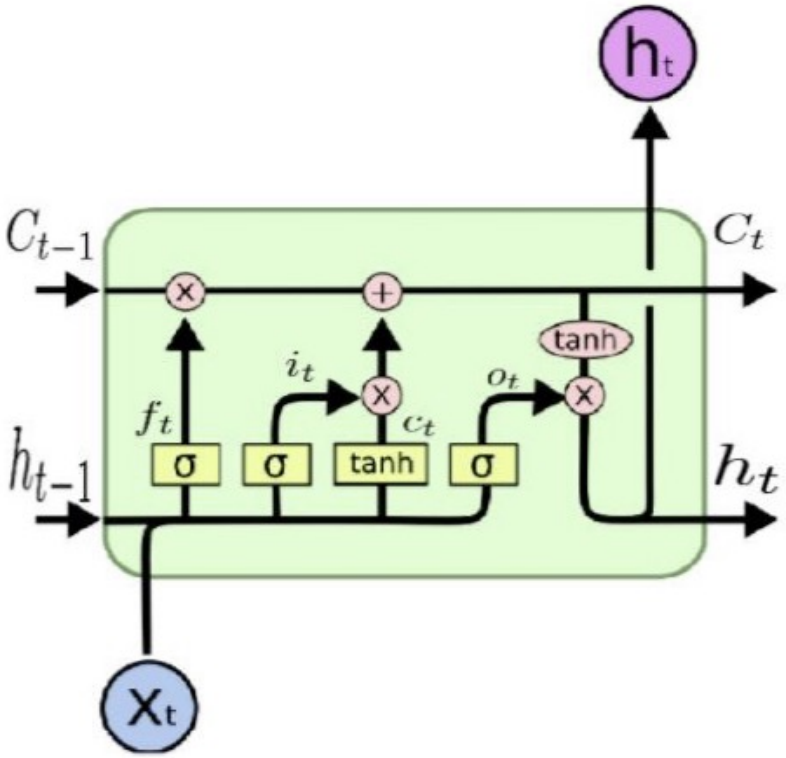
Long Short Term Memory (LSTM)

- LSTMs are designed to explicitly alleviate the long-term dependency problem [Horchreiter & Schmidhuber (1997)]



Long Short Term Memory (LSTM)

- Linear memory cells + multiplicative gate units to store read, write, and reset information

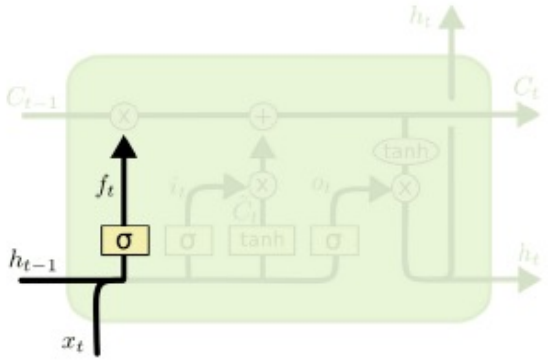


Gating functions to select information for passing and remembering

- Forget gate: whether to erase cell (reset)
- Input gate: whether to write to cell (write)
- Output gate: how much to reveal cell (read)

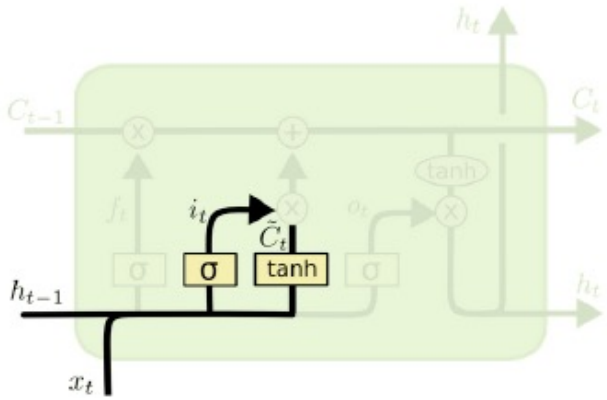
Long Short Term Memory (LSTM)

- Forget gate: decides what must be removed from $\boldsymbol{h_{t-1}}$



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

- Input gate: decides what new information to store in the cell

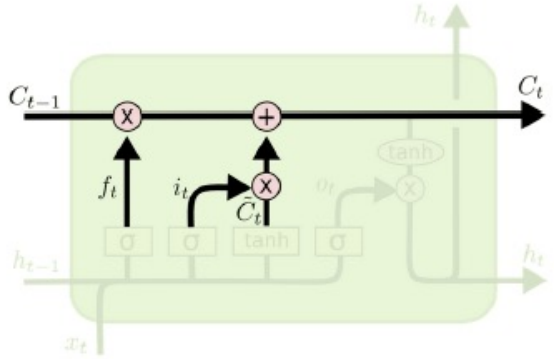


$$\boldsymbol{i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)}$$

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

Long Short Term Memory (LSTM)

- Update cell state:

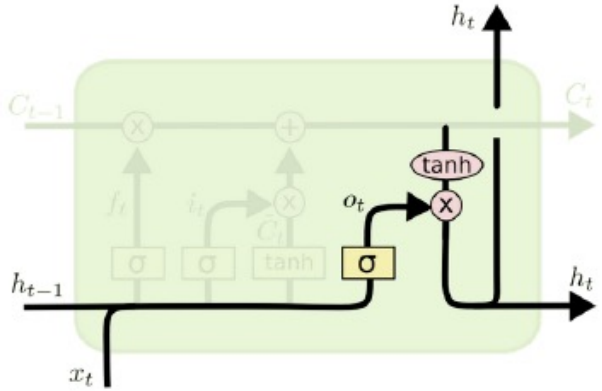


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

forgetting unneeded things

scaling the new candidate values by how much we decided to update each state value.

- Output gate: decides what to output from our cell state

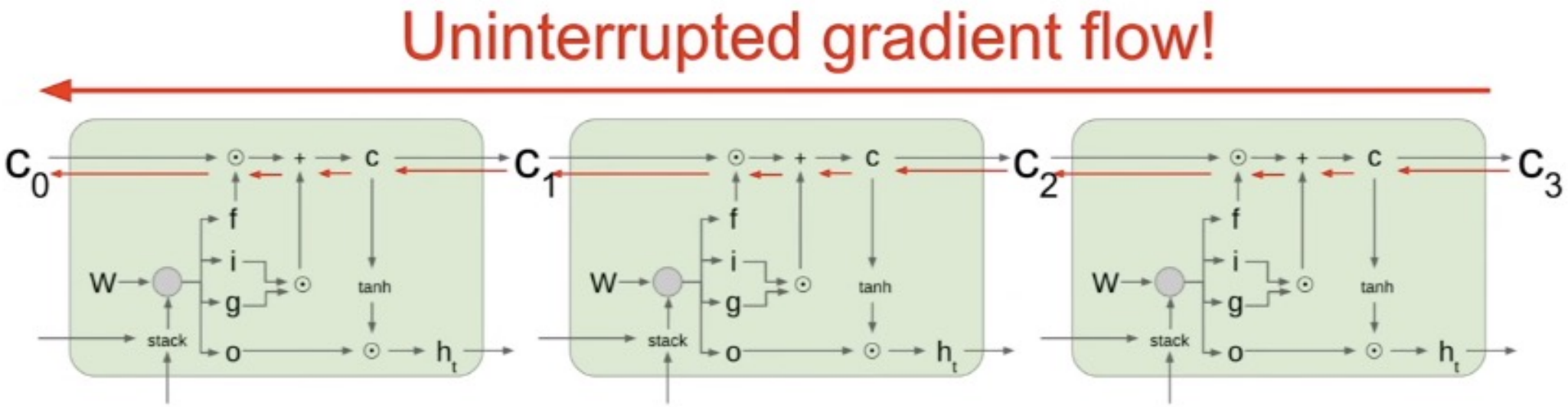


$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

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

sigmoid decides what parts of the cell state we're going to output

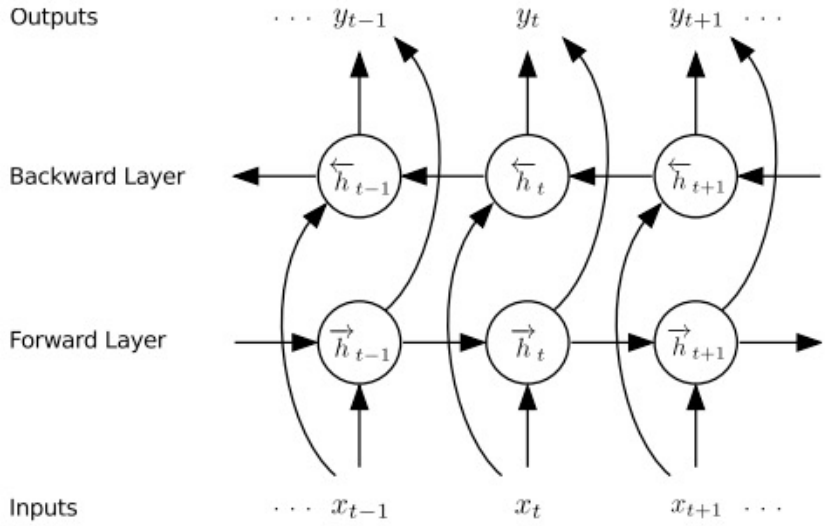
Backprop in LSTM



- No multiplication with matrix W during backprop
- Multiplied by different values of forget gate -> less prone to vanishing/exploding gradient

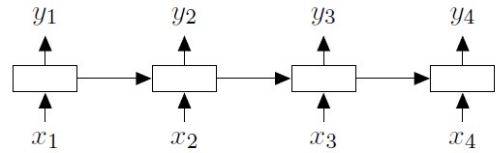
RNNs in Various Forms

- Bi-directional RNN
  - Hidden state is the concatenation of both forward and backward hidden states.
  - Allows the hidden state to capture both past and future information.
- Tree-structured RNN
  - Hidden states condition on both an input vector and the hidden states of arbitrarily many child units.
  - Standard LSTM = a special case of tree-LSTM where each internal node has exactly one child.

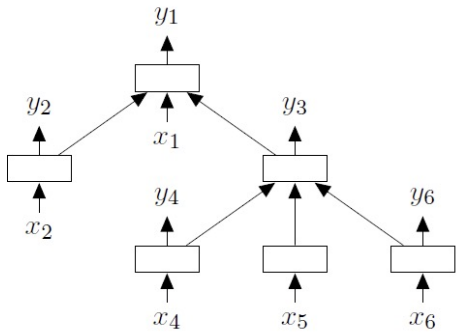


[Speech Recognition with Deep Recurrent Neural Networks, Alex Graves]

Chain LSTM



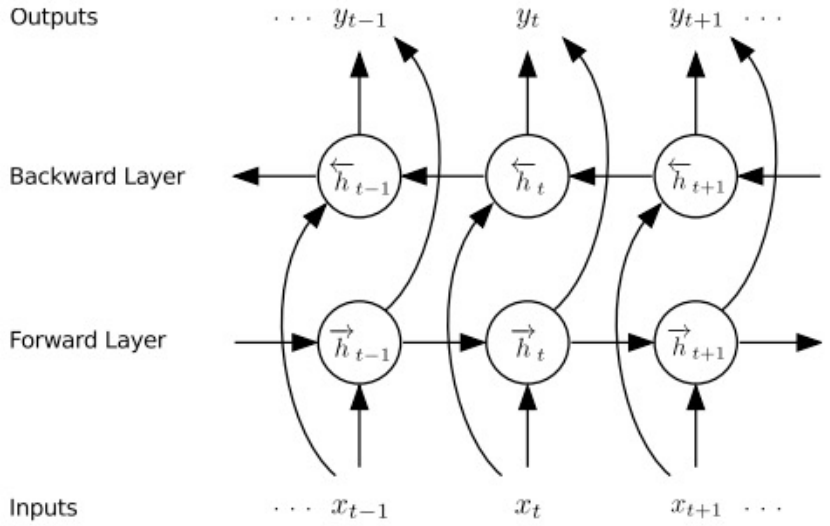
Tree LSTM



Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks, Tai. et al.

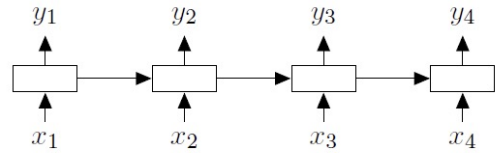
RNNs in Various Forms

- Bi-directional RNN
  - Hidden state is the concatenation of both forward and backward hidden states.
  - Allows the hidden state to capture both past and future information.
- Tree-structured RNN
  - Hidden states condition on both an input vector and the hidden states of arbitrarily many child units.
  - Standard LSTM = a special case of tree-LSTM where each internal node has exactly one child.

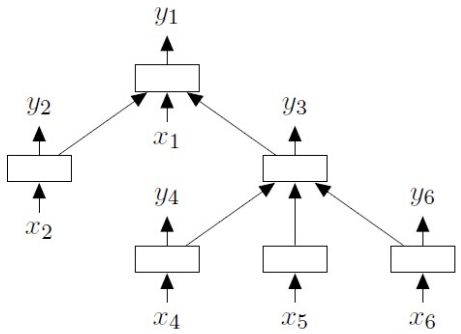


[Speech Recognition with Deep Recurrent Neural Networks, Alex Graves]

Chain LSTM



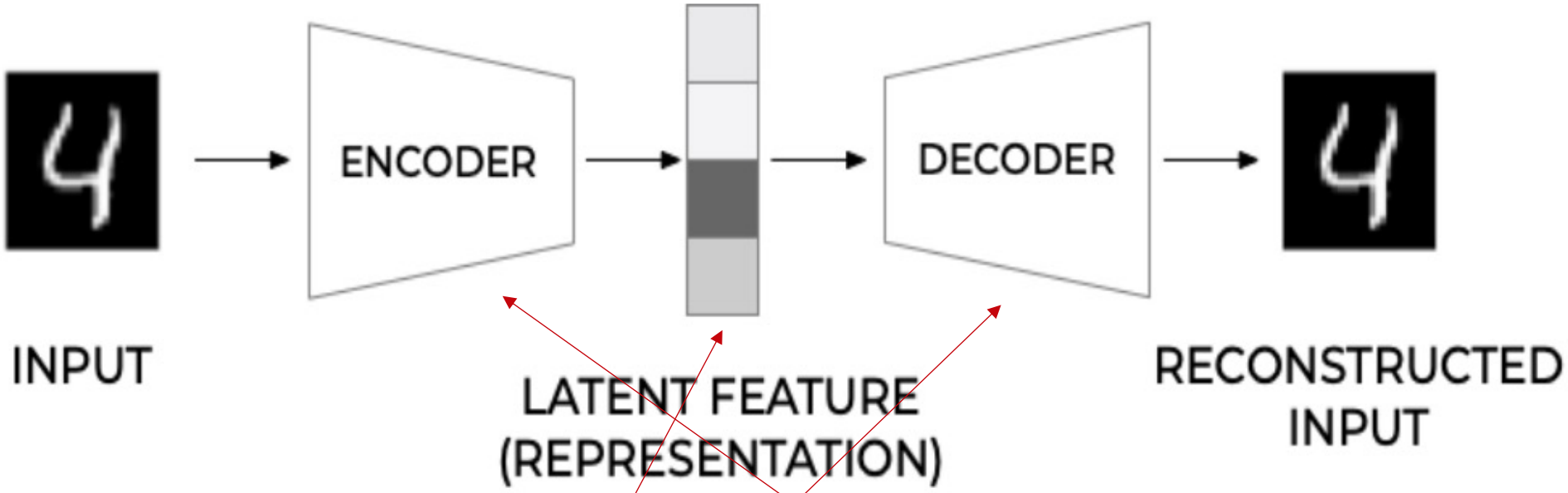
Tree LSTM



Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks, Tai. et al.

Autoencoders

Autoencoders



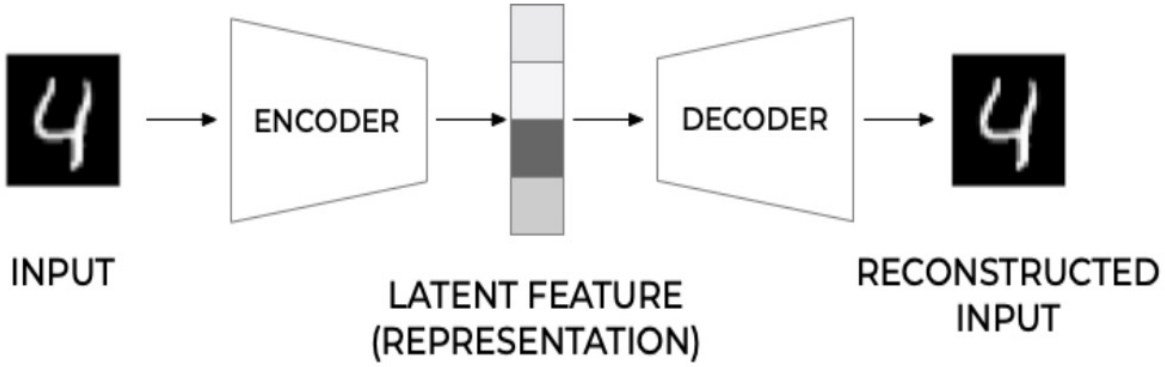
$$\tilde{x} = f(h) = f\big(g(x)\big)$$

[Michelucci 2022]

Why reduce dimensionality?

- Reduce computation cost of downstream tasks.
- Improve statistical stability of downstream tasks.
- Denoise observation.
- Learn to generate samples (variational autoencoders).

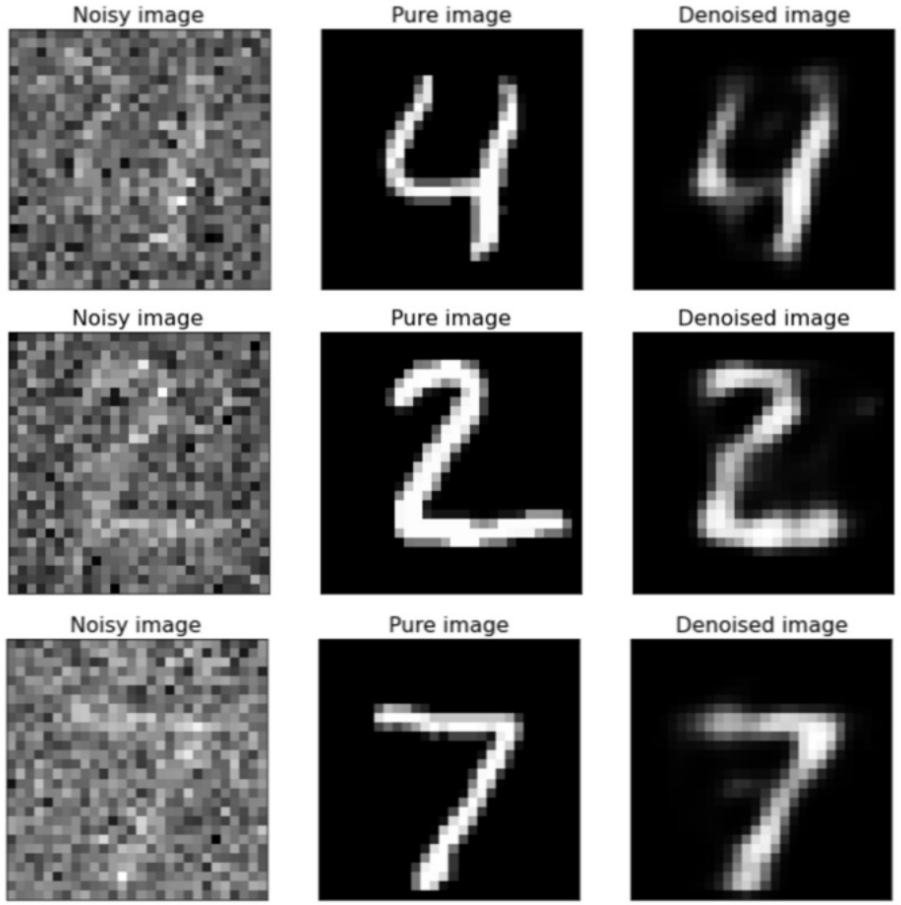
Autoencoders vs PCA for dimensionality reduction



- PCA: Dimensionality reduction by linear projection onto eigenvectors of covariance
  - Requires O(d^2) space for data in d dimensions (expensive!)
- Autoencoder:
  - O(d) space.
  - Train with batches.
- Equivalent! If:
  - Autoencoder is linear + loss function is MSE and inputs are normalized.

Denoising Autoencoders

- What if we train our autoencoder on data with intentionally-added noise?



[Michelucci 2022]

Questions?