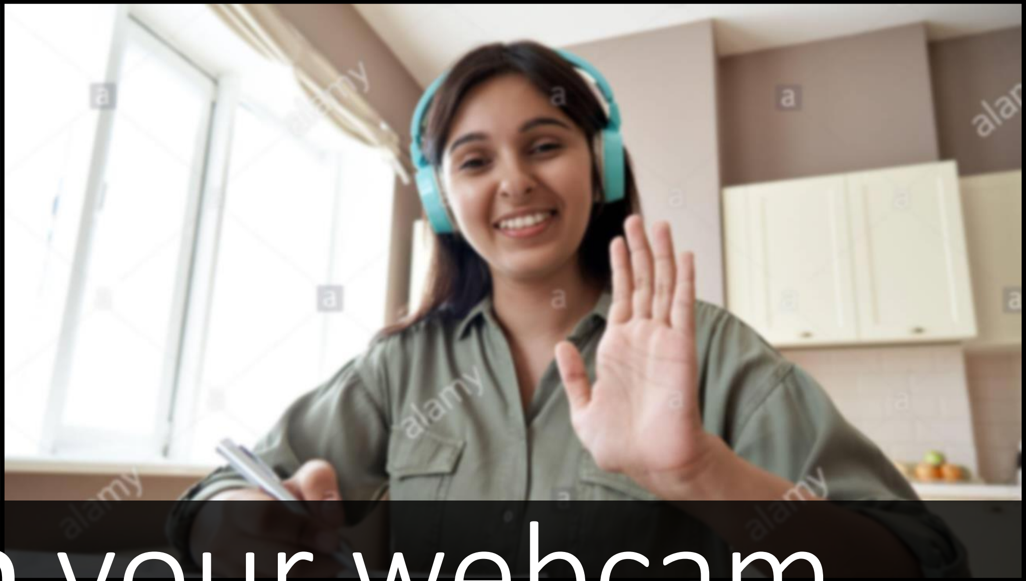
Deep Learning (CNN+RNN)

10 B

CS 3244 Machine Learning

NUS | Computing
National University of Singapore

Please **turn on** your webcam



Mystery Student

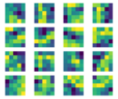
Convolutional Layer: Feature Kernels & Feature Maps

$k^{[l-1]} = c^{[l]}$
filters from previous layer $k^{[l-1]}$ is equal to #channels into current layer $c^{[l]}$

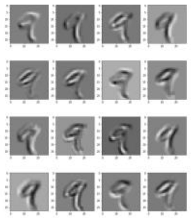
$$X^{[0]} \rightarrow g^{[1]}\left(W^{[1]} * X^{[0]}\right) = A^{[1]} \xrightarrow{\text{Pooling}} g^{[2]}\left(W^{[2]} * X^{[1]}\right) = A^{[2]}$$

1 channel · 16 filters · 16 channels · 32 filters · 32 channels

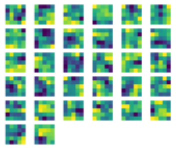
Input



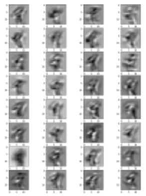
Layer 1 Feature Kernels



Layer 1 Activation Maps



Layer 2 Feature Kernels



Layer 2 Activation Maps

Hyperparameters
1. Number of kernels k
2. Kernel size $\boldsymbol{\kappa}$
3. Padding $\boldsymbol{p}$
4. Stride $\boldsymbol{s}$

Chosen manually, or automatically with hyperparameter tuning

Kernels are learned *automatically* through **weight updates**.

Interpretability: do you know what these kernel mean?

Image credit: https://medium.com/dataseries/visualizing-the-feature-maps-and-filters-by-convolutional-neural-networks-e1462340518e

Pooling Layer

- **Downsamples** Feature Maps
- Helps to train later kernels to detect **higher-level** features
- Reduces **dimensionality**
- Aggregation methods
  - Max-Pool (most used)
  - Average-Pool
  - Sum-Pool

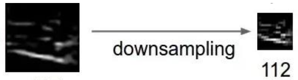
Calculation

12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

2×2 Max-Pool →

20	30
112	37

Example



224 × 224 → downsampling → 112 × 112

Image credit: https://computersciencewiki.org/index.php/Max-pooling_/_Pooling

Convolutional Layer

What's the differences between the left and right expressions?

$$a^{[l]} = g^{[l]}\left(\sum_{r \in R^{[l-1]}}^{n^{[l-1]}} w_r^{[l]} a_r^{[l-1]}\right) \quad \text{vs.} \quad A^{[l]} = g^{[l]}\left(\sum_{r \in R^{[l-1]}}^{c^{[l-1]}} W^{[l]} * A_r^{[l-1]}\right)$$

Sum across channels

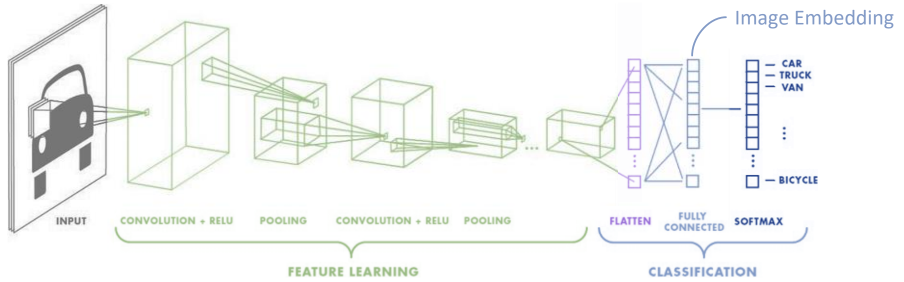
Weights → **Kernels** (4D matrix)

2D Convolution

Activation function can be the same

Activation is a 3D Matrix!
Activation Maps of multiple channels

Convolutional Neural Network



Key concepts

❶ Learn Spatial Feature
- Series of multiple convolution + pooling layers
- Progressively learn more diverse and higher-level features

❷ Flattening
- Convert to fixed-length 1D vector

❸ Learn Nonlinear Features
- With fully connected layers (regular neurons)
- Learns nonlinear relations with multiple layers

❹ Classification
- Softmax ≔ Multiclass Logistic Regression
- Feature input = image embedding vector (typically large vector)

Image credit: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Chain Rule

Consider composite function

Lagrange notation
Prime $'$ indicates first derivative relative to the function argument.
This can make writing derivatives more concise.
e.g., $y'(w) = dy/dw$

$$g(x) = g(f(x))$$

$$g = g(f), f = f(x)$$

$$\boxed{g'(x)} = \frac{dg}{dx} = \frac{dg}{df}\frac{df}{dx}$$

Intuition

Rate of change of g relative to x is the product of

- rates of change of g relative to f and
- rates of change of f relative to x

"If

- *a car travels 2x fast as a bicycle and*
- *the bicycle is 4x as fast as a walking man,*

then the car travels 2 × 4 = 8 times as fast as the man."

– George F. Simmons, Calculus with Analytic Geometry (1985)

Gradient Descent Weight Update

$$\underbrace{\boldsymbol{W}}_{\text{New weight}} \leftarrow \overbrace{\boldsymbol{W}}^{\text{Old weight}} - \underbrace{\eta}_{\text{Learning Rate}} \overbrace{\nabla\varepsilon}^{\text{Direction of steepest error increase}}$$

MSE error

$$\varepsilon = \frac{1}{2}(\hat{y} - y)^2$$

Gradient of error

$$\nabla\varepsilon = \frac{\partial\varepsilon}{\partial\boldsymbol{W}} = \frac{\partial\varepsilon}{\partial\hat{y}}\underbrace{\frac{\partial\hat{y}}{\partial\boldsymbol{W}}}_{\frac{\partial f}{\partial\boldsymbol{W}}\frac{\partial g}{\partial f}}$$

Reference

$$\boldsymbol{a}^{[l]} = g^{[l]}\left(f^{[l]}\right)$$

$$f^{[l]} = \left(W^{[l]}\right)^{\top}\boldsymbol{a}^{[l-1]}$$

Backprop → Weight Update

Suppose we want to calculate weight update for l^{th} layer

$$\boldsymbol{W}^{[l]} \leftarrow \boldsymbol{W}^{[l]} + \Delta\boldsymbol{W}^{[l]} = \boldsymbol{W}^{[l]} - \eta \frac{\partial \varepsilon}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \boldsymbol{W}^{[l]}}$$

1. Calculate gradient $\hat{y}'\left(\boldsymbol{W}^{[l]}\right) = \frac{\partial \hat{y}}{\partial \boldsymbol{W}^{[l]}}$
   1. With backprop, we have calculated $\delta^{[l+1]}$ from later layers
   2. Then $\delta^{[l]} = \left[g'^{[l]}\left(f^{[l]}\right)\right] \circ \left(\boldsymbol{W}^{[l+1]}\delta^{[l+1]}\right)$
   3. Then $\hat{y}'\left(\boldsymbol{W}^{[l]}\right) = \boldsymbol{a}^{[l-1]}\left(\delta^{[l]}\right)^{\top}$
2. Calculate weight update
   1. $\Delta\boldsymbol{W}^{[l]} = \eta \frac{\partial \varepsilon}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \boldsymbol{W}^{[l]}} = \boldsymbol{a}^{[l-1]}\left(\delta^{[l]}\right)^{\top}$
   2. Now our weights $\boldsymbol{W}^{[l]}$ should be better than before (higher prediction performance)

- $\delta^{[l+1]}$ from previous backprop gradient calculation
- $g^{[l]}$ from model specification
- $g'^{[l]}\left(f^{[l]}\right)$ calculated here
- $\boldsymbol{W}^{[l+1]}$ from stored weights (previous weight update training)
- $\boldsymbol{a}^{[l-1]}$ stored from forward prop

Week 10A: Learning Outcomes

1. Understand how deep learning enables better model performance than shallow machine learning
2. Explain how CNNs and RNNs are different from feedforward neural networks
3. Appropriately choose and justify when to use each architecture
4. Explain how to mitigate training issues in deep learning

Week 10A: Lecture Outline

1. Deep learning motivation
2. Popular Architectures
    1. Convolutional Neural Networks
    2. Recurrent Neural Networks
3. Deep learning training issues

Convolutional Neural Networks (CNN)

Department of Computer Science
School of Computing

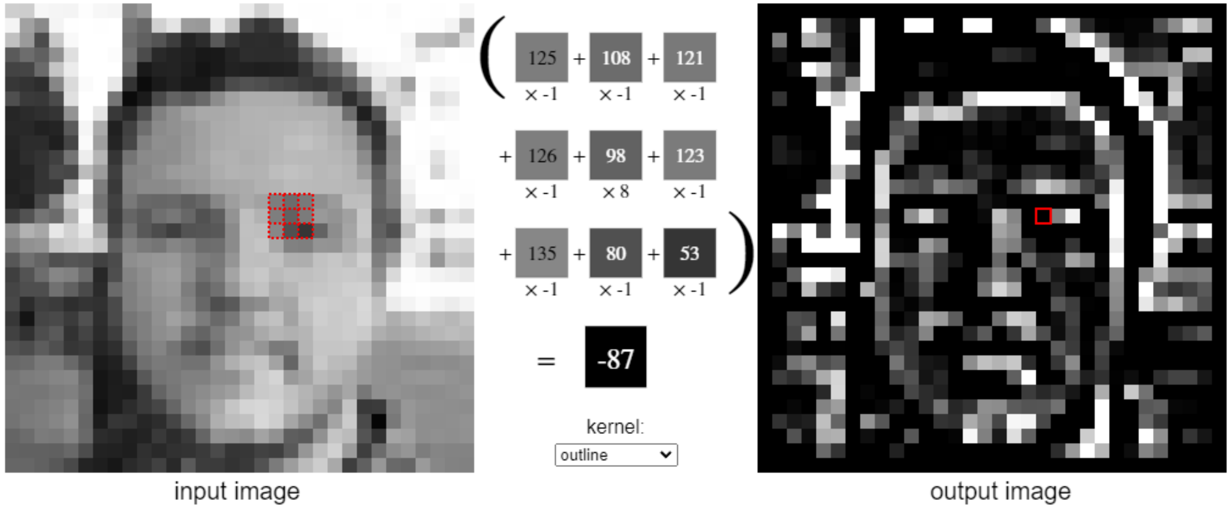
Reduce parameters for images:
Exploit **Spatial** Relations with **Convolutions**

Let's walk through applying the following 3x3 **outline** kernel to the image of a face from above.

outline

$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

Below, for each 3x3 block of pixels in the image on the left, we multiply each pixel by the corresponding entry of the kernel and then take the sum. That sum becomes a new pixel in the image on the right. Hover over a pixel on either image to see how its value is computed.



Manually finding good filters is tedious

Further study:
https://setosa.io/ev/image-kernels/

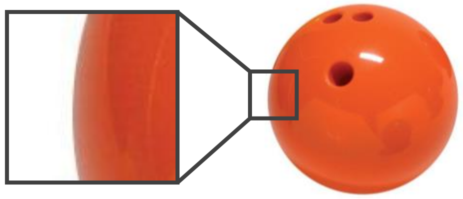
Analogy: activations of different filters learned by CNNs is like seeing the image through different lens filters

Image credit: https://www.amazon.com/Godefa-Samsung-Andriod-Smartphone-Universal/dp/B07RQRLQYH
https://www.yankodesign.com/2020/02/17/this-retro-inspired-camera-records-dreamy-looking-gifs-that-replicate-vintage-8mm-film/

Analogy: kernel update ≡ glass lens grinding

Image Credit: https://www.schneider-om.com/precision-optics/processes/polishing.html

Multi-Channel Convolutions



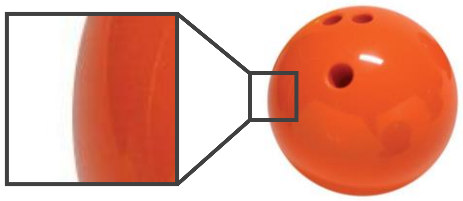
$$W_{11} = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \quad X_1 = \begin{pmatrix} 9 & 9 & 3 & 3 & 4 \\ 9 & 3 & 3 & 4 & 5 \\ 9 & 3 & 3 & 5 & 5 \\ 9 & 3 & 3 & 4 & 5 \\ 9 & 9 & 3 & 3 & 4 \end{pmatrix} \quad W_{11} * X_1 = \begin{pmatrix} -6-6-6 & -6+1+2 & 1+2+2 \\ -6-6-6 & 1+2+1 & 2+2+2 \\ -6-6-6 & 2+1-6 & 2+2+1 \end{pmatrix} = \begin{pmatrix} -18 & -3 & 5 \\ -18 & 4 & 6 \\ -18 & -3 & 5 \end{pmatrix}$$

$$W_{12} = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \quad X_2 = \begin{pmatrix} 9 & 9 & 1 & 1 & 2 \\ 9 & 1 & 1 & 2 & 3 \\ 9 & 1 & 1 & 3 & 3 \\ 9 & 1 & 1 & 2 & 3 \\ 9 & 9 & 1 & 1 & 2 \end{pmatrix} \quad W_{12} * X_2 = \begin{pmatrix} -8-8-8 & -8+1+2 & 1+2+2 \\ -8-8-8 & 1+2+1 & 2+2+2 \\ -8-8-8 & 2+1-8 & 2+2+1 \end{pmatrix} = \begin{pmatrix} -24 & -5 & 5 \\ -24 & 4 & 6 \\ -24 & -5 & 5 \end{pmatrix} \rightarrow + \rightarrow \sum_{r=1}^{c=3} W_{1r} * X_r = \begin{pmatrix} -69 & -14 & 12 \\ -69 & 12 & 15 \\ -69 & -14 & 12 \end{pmatrix}$$

$$W_{13} = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \quad X_3 = \begin{pmatrix} 9 & 9 & & & \\ 9 & & & & 1 \\ 9 & & & & 1 \\ 9 & & & & 1 \\ 9 & 9 & & & \end{pmatrix} \quad W_{13} * X_3 = \begin{pmatrix} -9-9-9 & -9+1+2 & 0+1+1 \\ -9-9-9 & 1+2+1 & 1+1+1 \\ -9-9-9 & 2+1-9 & 1+1+0 \end{pmatrix} = \begin{pmatrix} -27 & -6 & 2 \\ -27 & 4 & 3 \\ -27 & -6 & 2 \end{pmatrix}$$

Kernels (Filters) | Input (Image)

Multi-Channel Convolutions



$$W_{21} = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \quad X_1 = \begin{pmatrix} 9 & 9 & 3 & 3 & 4 \\ 9 & 3 & 3 & 4 & 5 \\ 9 & 3 & 3 & 5 & 5 \\ 9 & 3 & 3 & 4 & 5 \\ 9 & 9 & 3 & 3 & 4 \end{pmatrix} \quad W_{21} * X_1 = \begin{pmatrix} -6-6-6 & -6+1+2 & 1+2+2 \\ -6-6-6 & 1+2+1 & 2+2+2 \\ -6-6-6 & 2+1-6 & 2+2+1 \end{pmatrix} = \begin{pmatrix} -18 & -3 & 5 \\ -18 & 4 & 6 \\ -18 & -3 & 5 \end{pmatrix}$$

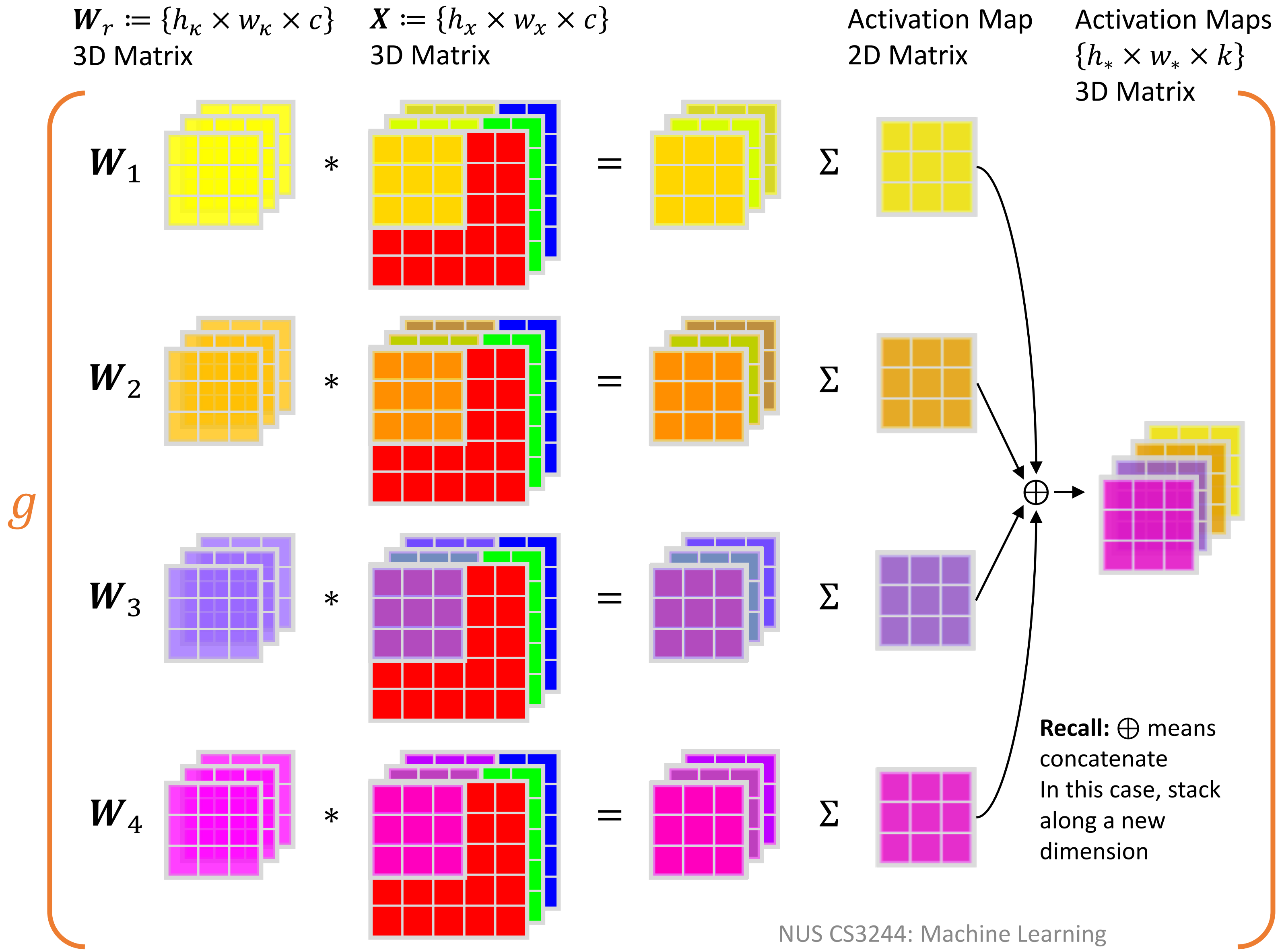
$$W_{22} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad X_2 = \begin{pmatrix} 9 & 9 & 1 & 1 & 2 \\ 9 & 1 & 1 & 2 & 3 \\ 9 & 1 & 1 & 3 & 3 \\ 9 & 1 & 1 & 2 & 3 \\ 9 & 9 & 1 & 1 & 2 \end{pmatrix} \quad W_{22} * X_2 = \begin{pmatrix} 0+0+0 & 0+0+0 & 0+0+0 \\ 0+0+0 & 0+0+0 & 0+0+0 \\ 0+0+0 & 0+0+0 & 0+0+0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$W_{23} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad X_3 = \begin{pmatrix} 9 & 9 & & & \\ 9 & & & & 1 \\ 9 & & & & 1 \\ 9 & & & & 1 \\ 9 & 9 & & & \end{pmatrix} \quad W_{23} * X_3 = \begin{pmatrix} 0+0+0 & 0+0+0 & 0+0+0 \\ 0+0+0 & 0+0+0 & 0+0+0 \\ 0+0+0 & 0+0+0 & 0+0+0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$+ \rightarrow \sum_{r=1}^{c=3} W_{2r} * X_r = \begin{pmatrix} -18 & -3 & 5 \\ -18 & 4 & 6 \\ -18 & -3 & 5 \end{pmatrix}$$

Kernels can be **different** for **different channels**.
In this case, vertical edge detector for red channel only

Multi-Channel Convolutions ($c = 3$ channels, $k = 4$ filters)



Key Takeaways

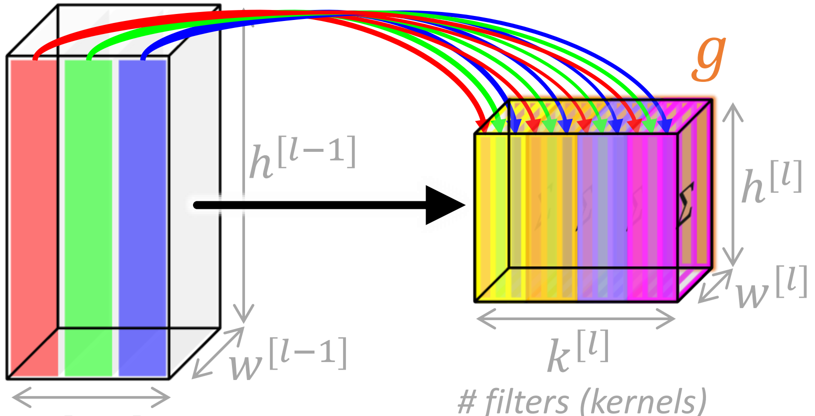
- Each <u>Kernel</u> convolves on **all channels** of image
- Each <u>Activation Map</u> is summed across channels
- 1 Activation Map per Kernel
- Kernel indexed in 3rd dimension of Activation Maps

Multi-Channel Convolutions (layers diagram)

$$\boldsymbol{W}^{[l]} = \begin{pmatrix} \boldsymbol{W}_{11} & \boldsymbol{W}_{21} & \boldsymbol{W}_{31} & \boldsymbol{W}_{41} \\ \boldsymbol{W}_{12} & \boldsymbol{W}_{22} & \boldsymbol{W}_{32} & \boldsymbol{W}_{42} \\ \boldsymbol{W}_{13} & \boldsymbol{W}_{23} & \boldsymbol{W}_{33} & \boldsymbol{W}_{43} \end{pmatrix}$$

$\boldsymbol{W}^{[l]}$ 4D Matrix
$\{h_\kappa^{[l]} \times w_\kappa^{[l]} \times \underline{c^{[l-1]}} \times \underline{k^{[l]}}\}$

$\boldsymbol{A}^{[l-1]}$



$$\boldsymbol{A}^{[l]} = g^{[l]}\left(\sum_{r \in R^{[l-1]}}^{c^{[l-1]}} \boldsymbol{W}^{[l]} * \boldsymbol{A}_r^{[l-1]}\right)$$

$k^{[l-1]} = c^{[l-1]}$ *# channels*
 # filters of previous layer

 # filters (kernels)

$\boldsymbol{A}^{[l-1]}$ 3D Matrix
$\{h^{[l-1]} \times w^{[l-1]} \times \underline{c^{[l-1]}}\}$

$\boldsymbol{A}^{[l]}$ 3D Matrix
$\{h^{[l]} \times w^{[l]} \times \underline{k^{[l]}}\}$

$h^{[l]}$ and $w^{[l]}$ calculated based on conv hyperparameters [E10a.1]

Convolutional Layers

Remember: each kernel is like a different lens filter

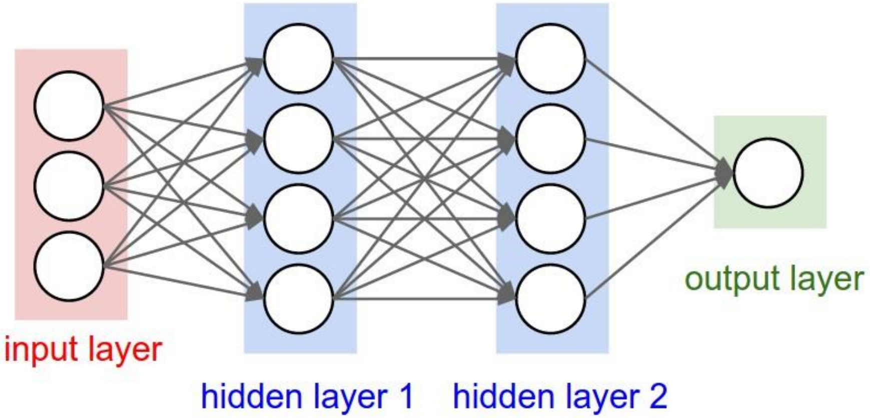
Fully Connected Layers

Each layer has multiple activations

- Each activation is a **0D scalar**
- Layer is a **1D vector**

Weights →

- All weights map activations of previous layer (1D) to current layer (1D)
- All weights represented as a **2D vector**



Convolutional Layers

Each layer has multiple activation maps

- Each activation map is a **2D matrix**
- Each map is on a different *channel* (1D)
- Layer is a **3D matrix**

Kernels

- Convolves on activation map (2D) of *all channels* (1D) in previous layer, then summed
- Each kernel represented as a **3D matrix**
- Each kernel stored as separate *filters* (1D)
- All kernels represented as **4D matrix**

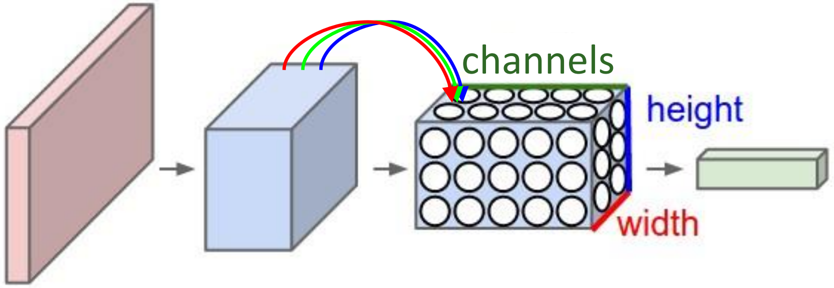


Image credit: https://cs231n.github.io/convolutional-networks/

Convolutional Layer

What's the differences between the left and right expressions?

Weights → **Kernels** (4D matrix)

Sum across channels

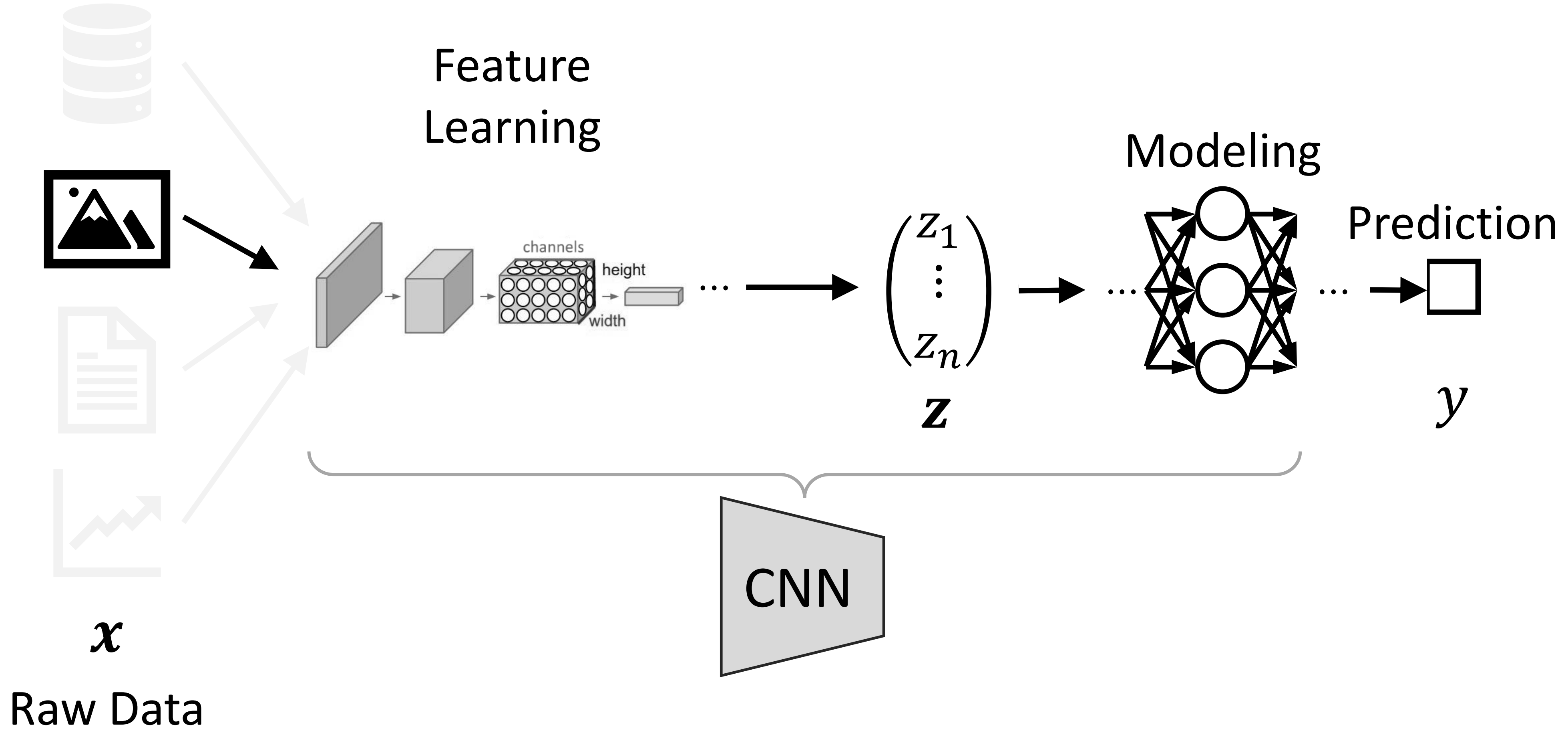
2D Convolution

$$\boldsymbol{a}^{[l]} = g^{[l]}\left(\sum_{r \in R^{[l-1]}}^{n^{[l-1]}} w_r^{[l]} a_r^{[l-1]}\right) \quad \text{vs.} \quad \boldsymbol{A}^{[l]} = g^{[l]}\left(\sum_{r \in R^{[l-1]}}^{c^{[l-1]}} \boldsymbol{W}^{[l]} * \boldsymbol{A}_r^{[l-1]}\right)$$

Activation function can be the same

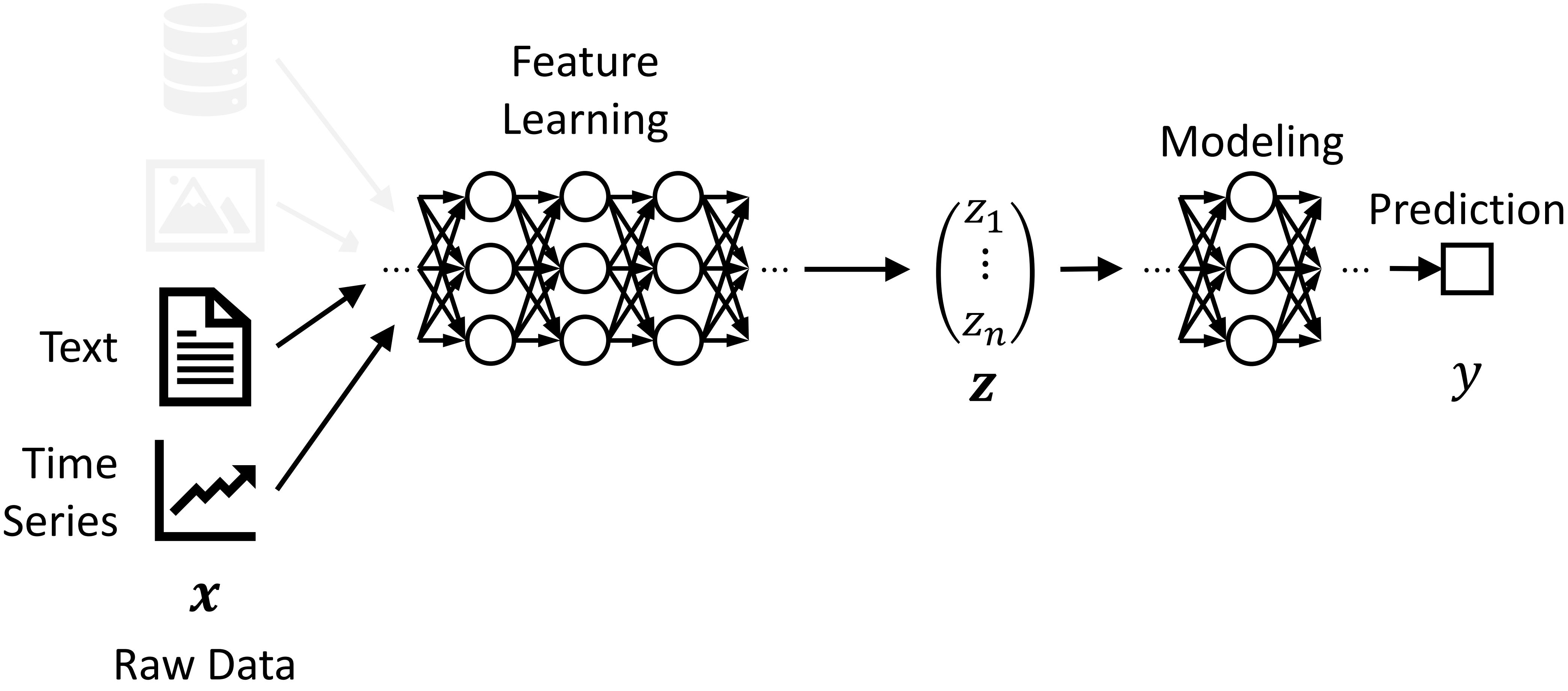
Activation is a 3D Matrix!
Activation Maps of multiple channels

From Manual Feature Engineering To Automatic Feature Learning




?
Questions!

From Manual Feature Engineering To Automatic Feature Learning



Recurrent Neural Networks (RNN)

NUS National University of Singapore | Department of Computer Science School of Computing

Applications of RNN

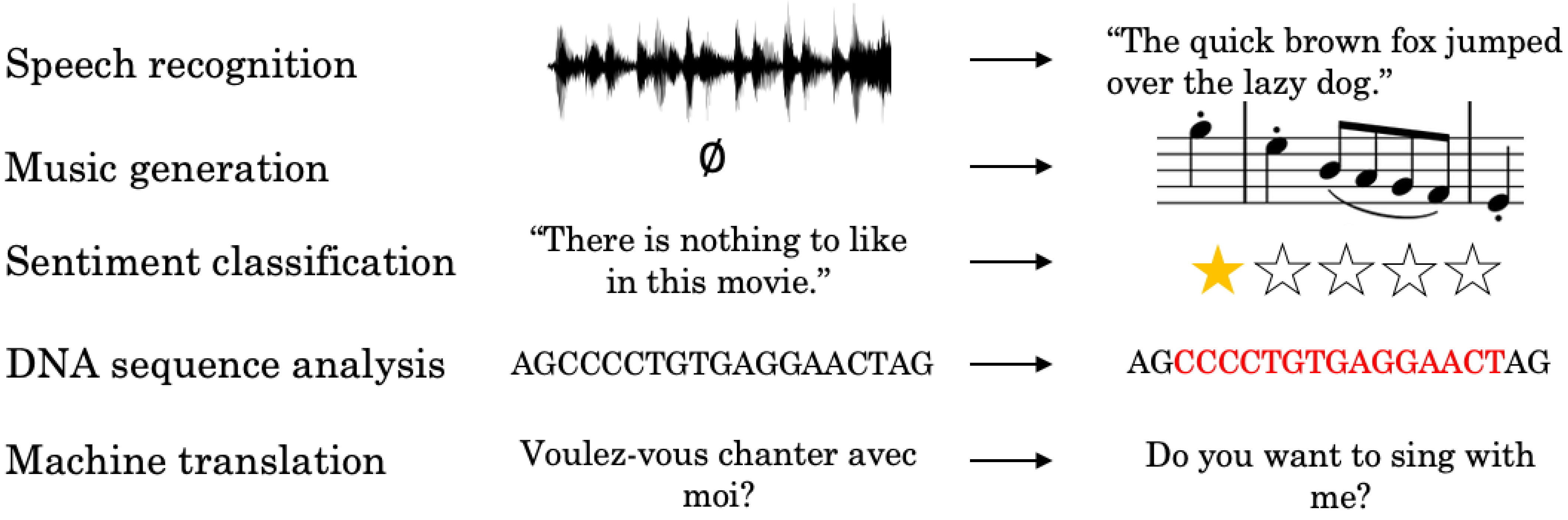


Image credit: https://laptrinhx.com/understanding-of-recurrent-neural-networks-lstm-gru-3720007533/

Exercise E10a.2

info random clear Model: bicycle

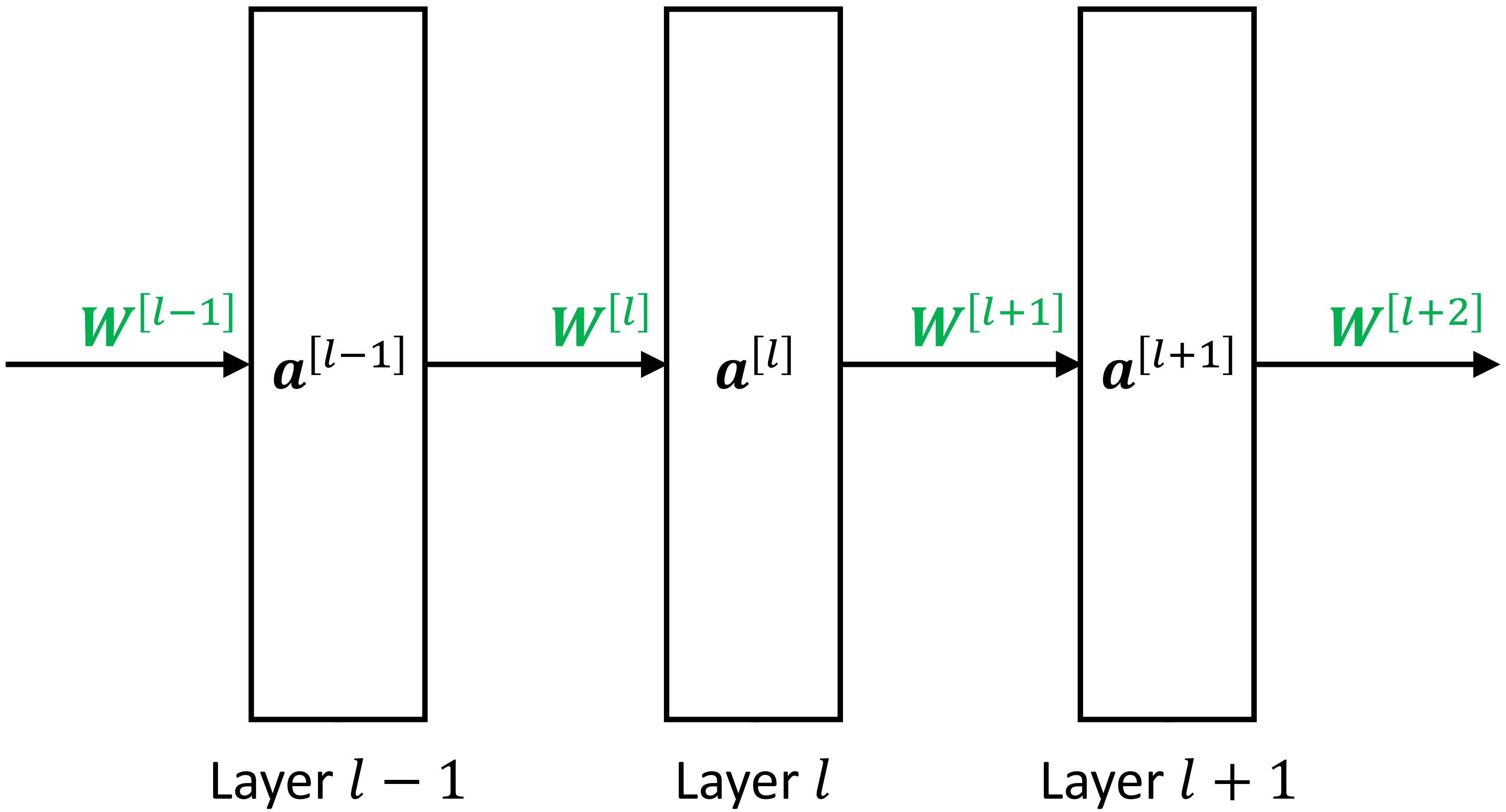
start drawing bicycle.

- **Draw something** and see what the AI will continue drawing
- **Take screenshot** and post your results
  - No obscenities please (we will have to take disciplinary action)
- **Emote**
  - Up vote those you like
  - Down vote those with mistakes
- **Discuss** how you think the model predicted what to draw next

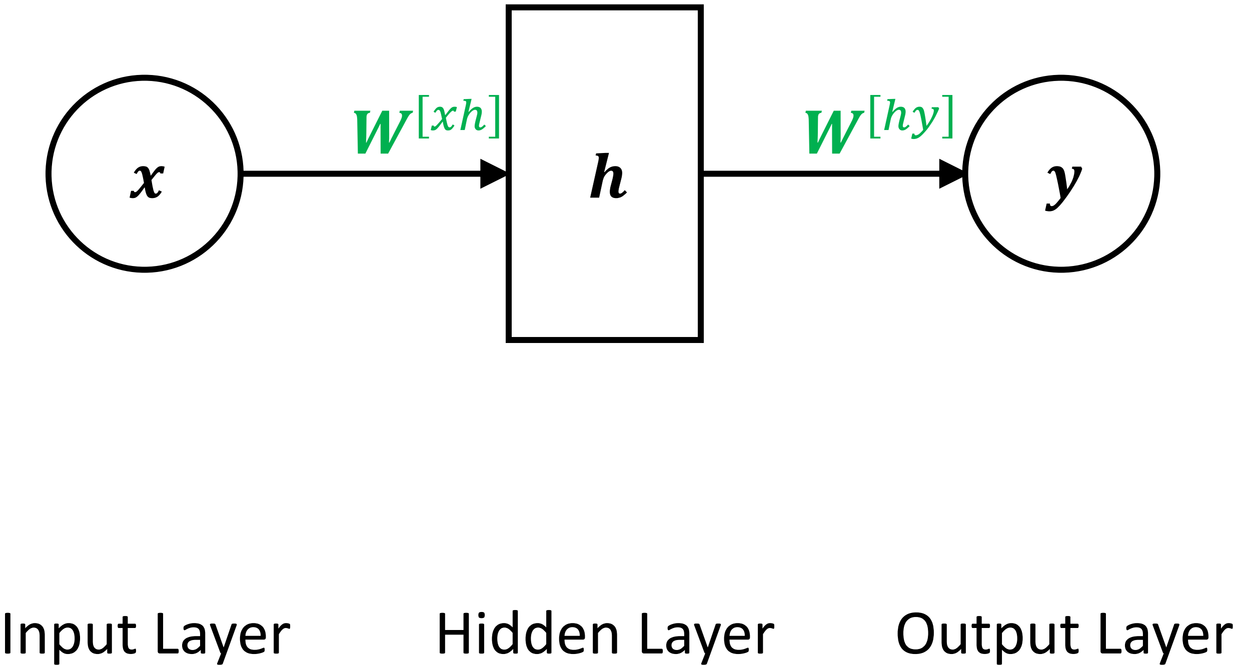
Try yourself:
https://magenta.tensorflow.org/assets/sketch_rnn_demo/index.html

Source: https://magenta.tensorflow.org/assets/sketch_rnn_demo/index.html

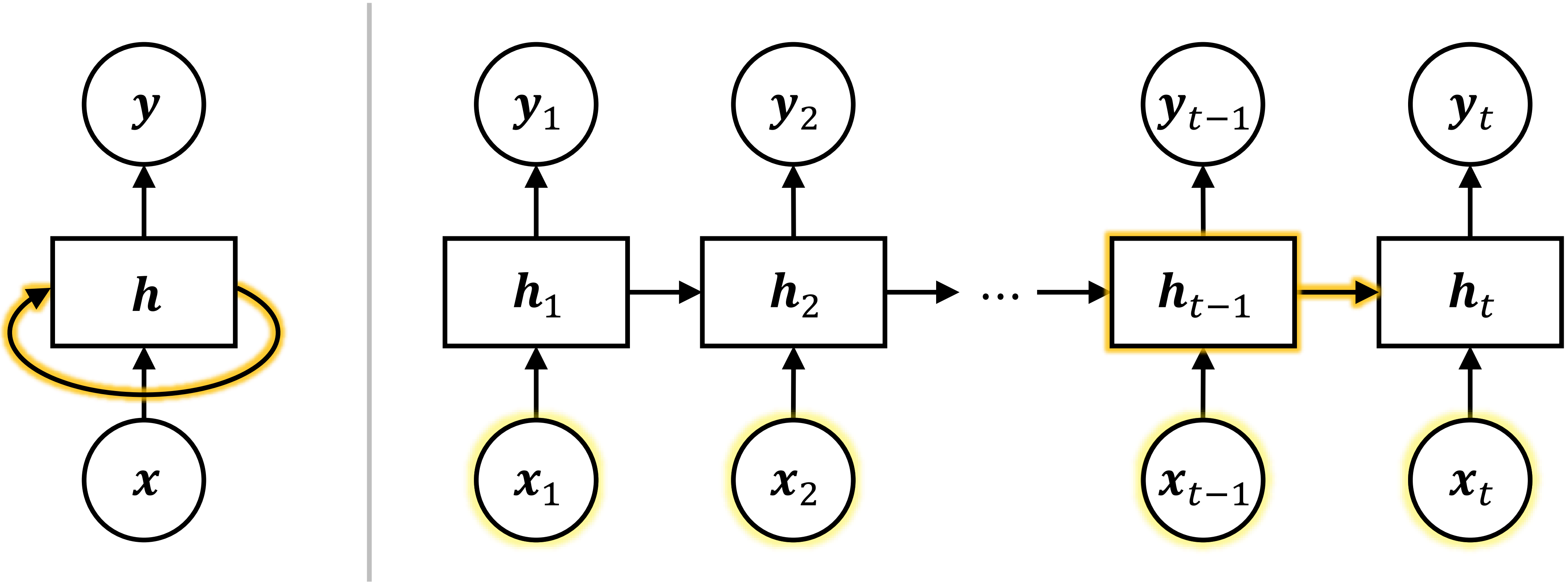
Feedforward Neural Network



Neural Network (simplified 1-hidden layer)



Neurons with Recurrence



$$\hat{\boldsymbol{y}} = g^{[y]}\left(g^{[h]}(\boldsymbol{x}_t, \boldsymbol{h}_{t-1})\right)$$

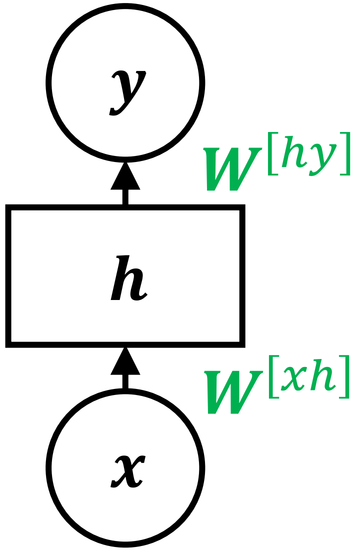
Recurrent Neural Network (RNN)

$$\hat{\boldsymbol{y}} = g^{[y]}(\boldsymbol{h}_t)$$

$$\boldsymbol{h}_t = g^{[h]}(\boldsymbol{x}_t, \boldsymbol{h}_{t-1})$$

RNN Weights

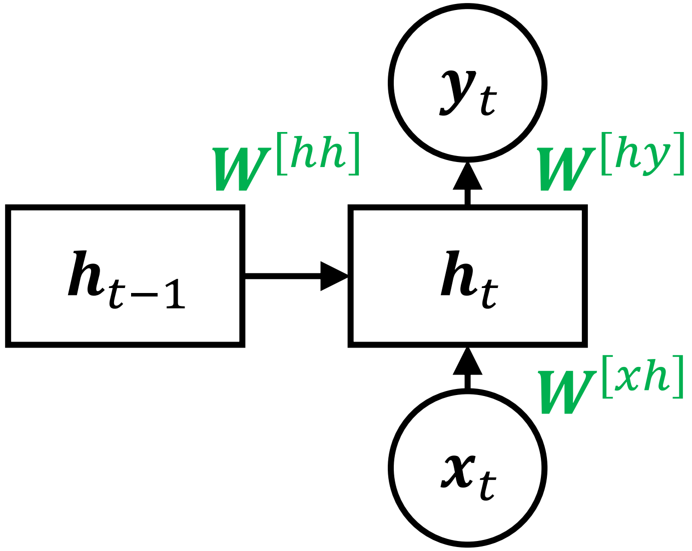
Question: Do these weights change for different time t?



Feedforward Neural Network

$$\boldsymbol{y} = g^{[y]}\left(\left(\boldsymbol{W}^{[hy]}\right)^{\top}\boldsymbol{h}\right)$$

$$\boldsymbol{h} = g^{[h]}\left(\left(\boldsymbol{W}^{[xh]}\right)^{\top}\boldsymbol{x}\right)$$



Recurrent Neural Network

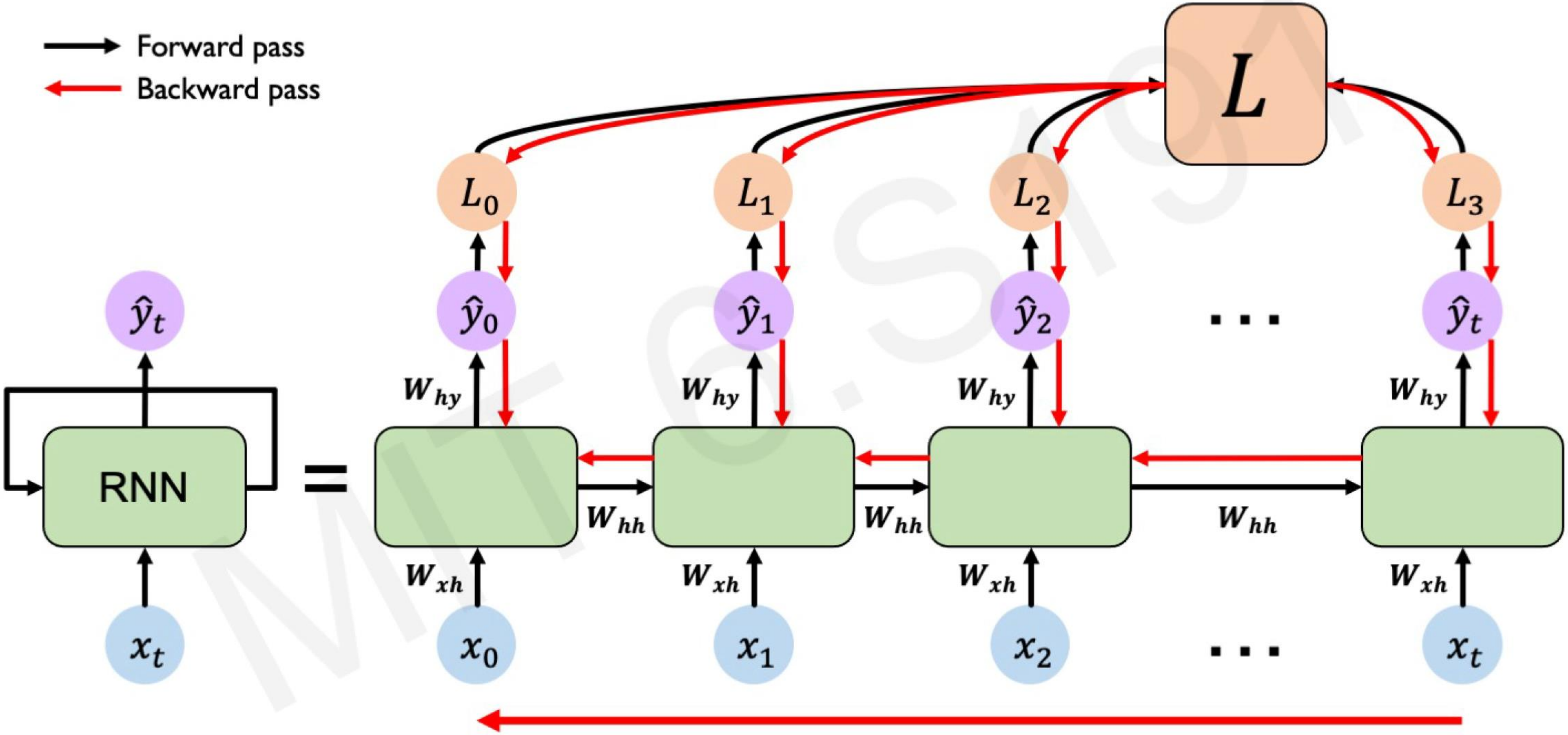
$$\boldsymbol{y}_t = g^{[y]}\left(\left(\boldsymbol{W}^{[hy]}\right)^{\top}\boldsymbol{h}_t\right)$$

$$\boldsymbol{h}_t = g^{[h]}\left(\left(\boldsymbol{W}^{[xh]}\right)^{\top}\boldsymbol{x}_t + \left(\boldsymbol{W}^{[hh]}\right)^{\top}\boldsymbol{h}_{t-1}\right)$$

$$\boldsymbol{h}_t = g^{[h]}\left(\left(\boldsymbol{W}^{[xh]} \oplus \boldsymbol{W}^{[hh]}\right)^{\top}(\boldsymbol{x}_t \oplus \boldsymbol{h}_{t-1})\right)$$


2021

RNNs: Backpropagation Through Time

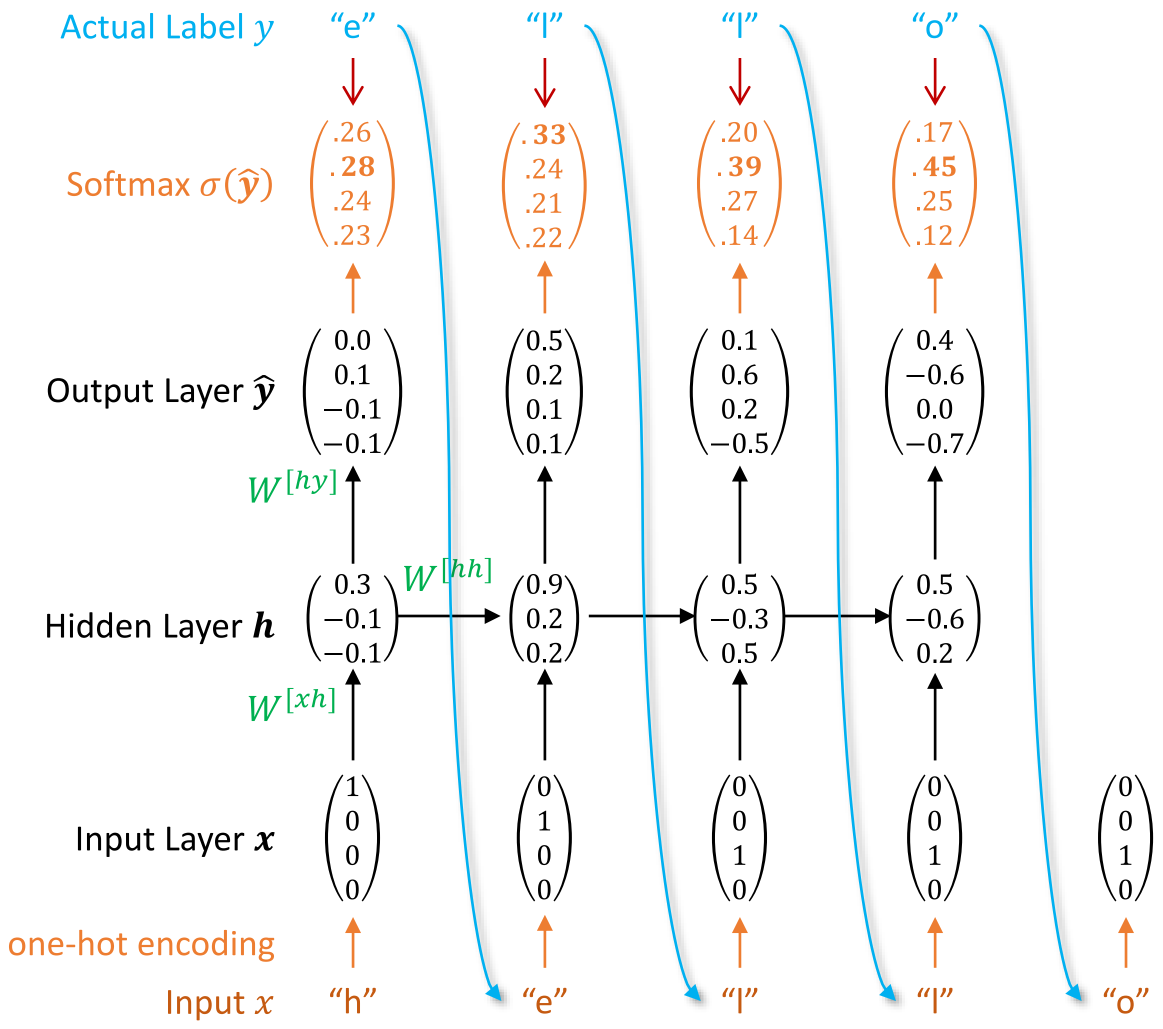


$$W^{[xh]} = \begin{pmatrix} 0.3 & 1.0 & 0.1 & 0.5 \\ -0.1 & 0.3 & -0.3 & 0.1 \\ -0.1 & 0.1 & -0.5 & -0.5 \end{pmatrix} \quad W^{[hy]} = \begin{pmatrix} 0.3 & 0.9 & 0.1 \\ 0.2 & -0.6 & 0.5 \\ -0.1 & 0.1 & 0.5 \\ -0.1 & 1.0 & -0.2 \end{pmatrix} \quad W^{[hh]} = \begin{pmatrix} 0.1 & 0.4 & 0.8 \\ -0.1 & 0.5 & -0.2 \\ 0.9 & 0.2 & 0.6 \end{pmatrix} \quad \sigma(\hat{\mathbf{y}}) = \frac{\exp(\hat{\mathbf{y}})}{\sum_{c=1}^{C} \exp(\hat{y}_c)}$$

Example RNN

Text character prediction

- Dictionary
  - [h, e, l, o]
- Training: sequence “hello”
- Encoding and Decoding chars
  - One-hot encoding (e.g., BOW)
  - Softmax classification
- At training time,
  - $x_t = y_{t-1}$
  - Loss (Error) is calculated as cross-entropy loss between $\hat{y}_t$ and y_t

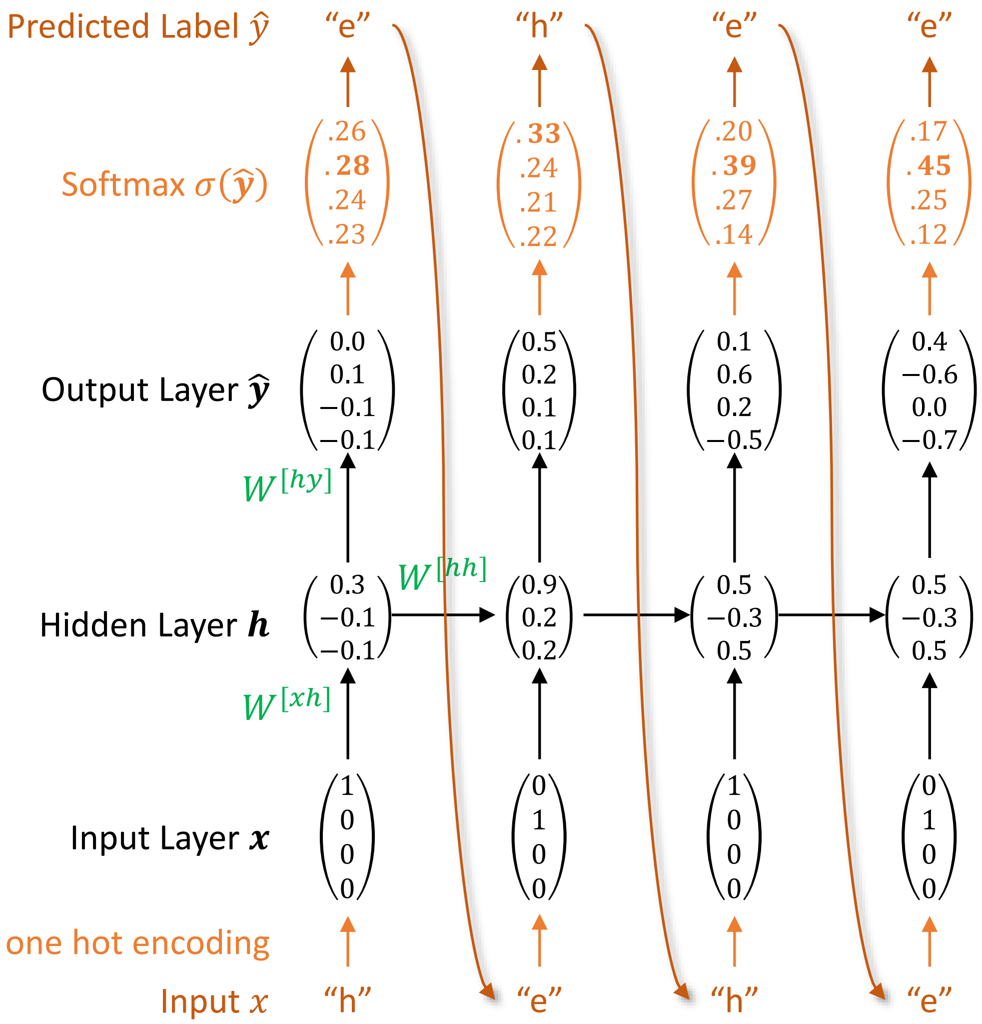


$$W^{[xh]} = \begin{pmatrix} 0.3 & 1.0 & 0.1 & 0.5 \\ -0.1 & 0.3 & -0.3 & 0.1 \\ -0.1 & 0.1 & -0.5 & -0.5 \end{pmatrix} \quad W^{[hy]} = \begin{pmatrix} 0.3 & 0.9 & 0.1 \\ 0.2 & -0.6 & 0.5 \\ -0.1 & 0.1 & 0.5 \\ -0.1 & 1.0 & -0.2 \end{pmatrix} \quad W^{[hh]} = \begin{pmatrix} 0.1 & 0.4 & 0.8 \\ -0.1 & 0.5 & -0.2 \\ 0.9 & 0.2 & 0.6 \end{pmatrix} \quad \sigma(\hat{\boldsymbol{y}}) = \frac{\exp(\hat{\boldsymbol{y}})}{\sum_{c=1}^{C} \exp(\hat{y}_c)}$$

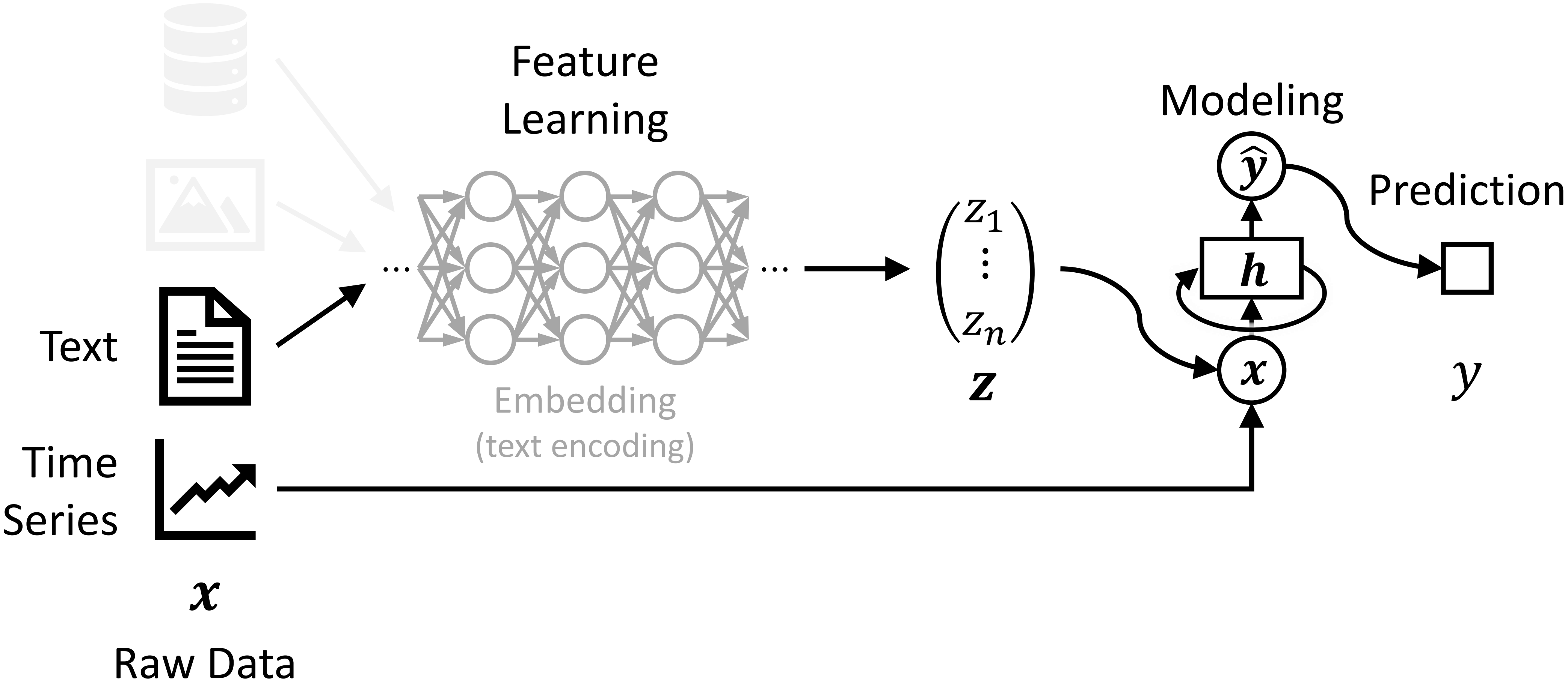
Example RNN

Text character prediction

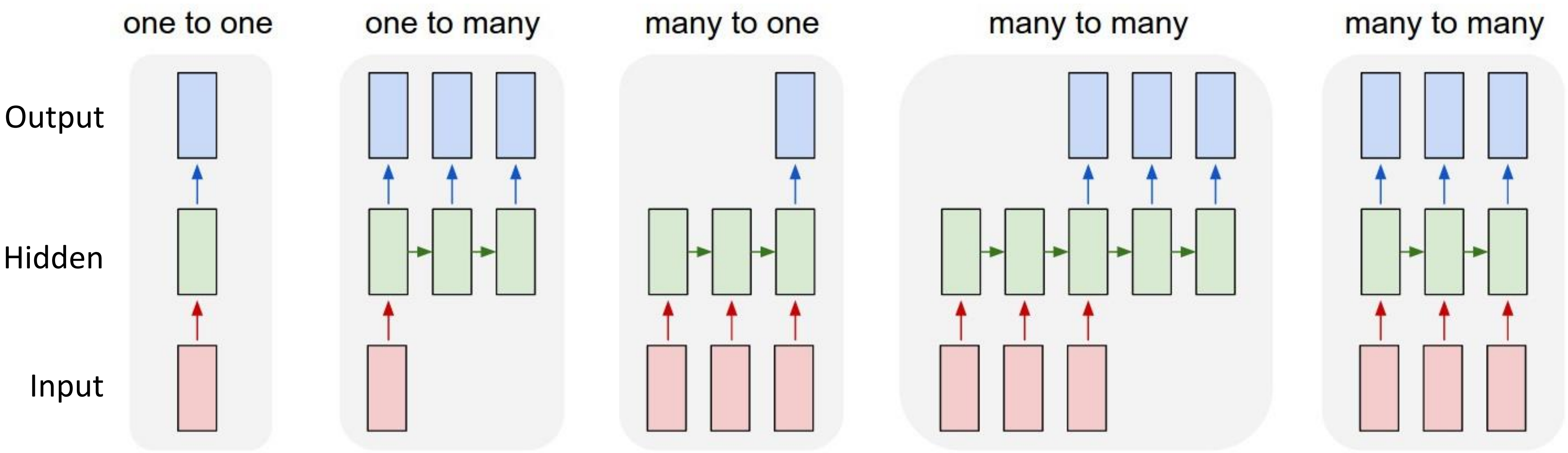
- Bag-of-Words dictionary
  - [h, e, l, o]
- At prediction time,
  - Forward propagate calculating activations to generate sequence of characters
  - $x_t = \hat{y}_{t-1}$



From Manual Feature Engineering To Automatic Feature Learning



Sequence Modeling Applications



Each time is *independent* (Not RNN)

E.g., image captioning, text generation

E.g., text classification, (sentiment recognition)

E.g., language translation, music generation

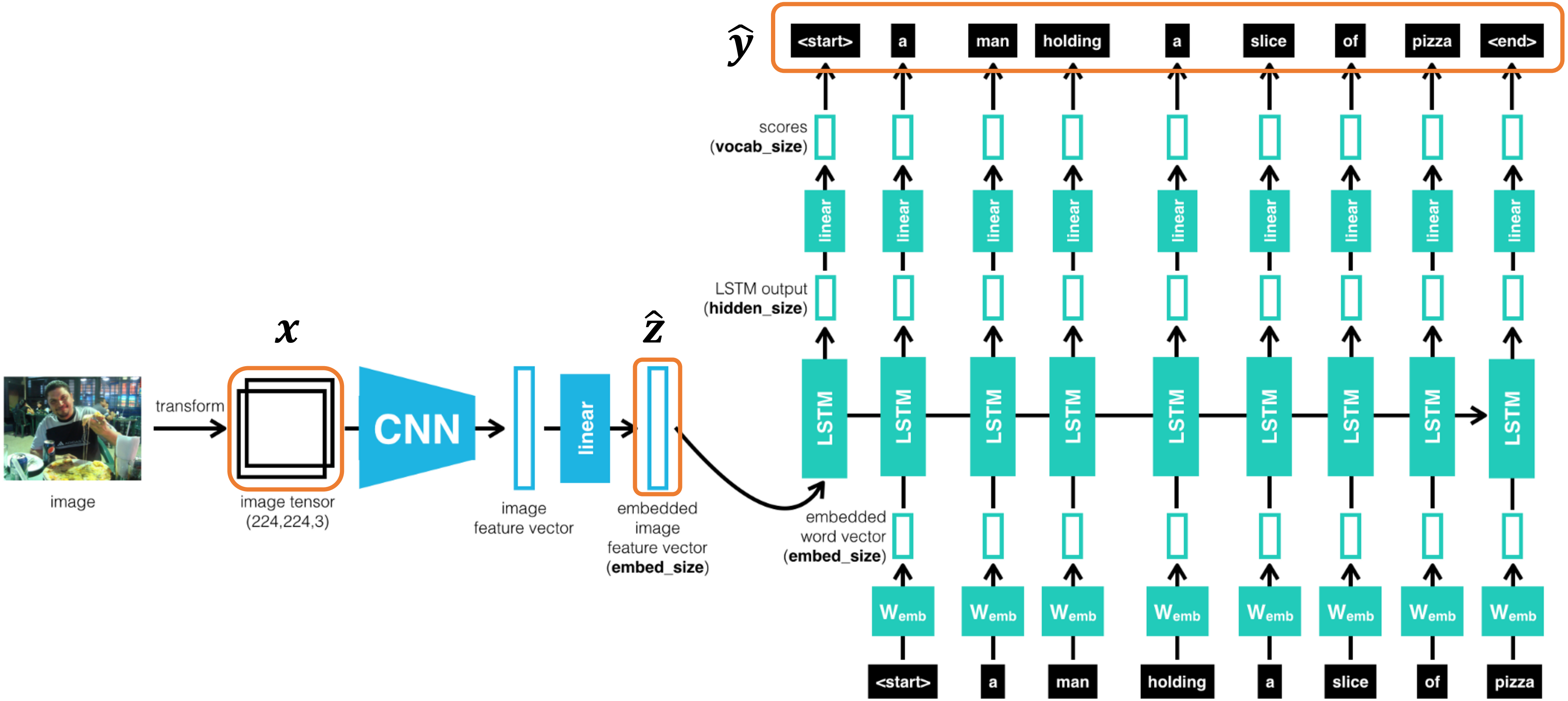
Encoder-Decoder Processes *all inputs first*, then predicts output

Plain RNN with *time dependency*

Processes 1 input *at a time*

Further reading: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Image Captioning: CNN + RNN (LSTM) – not in exam




?
Questions!

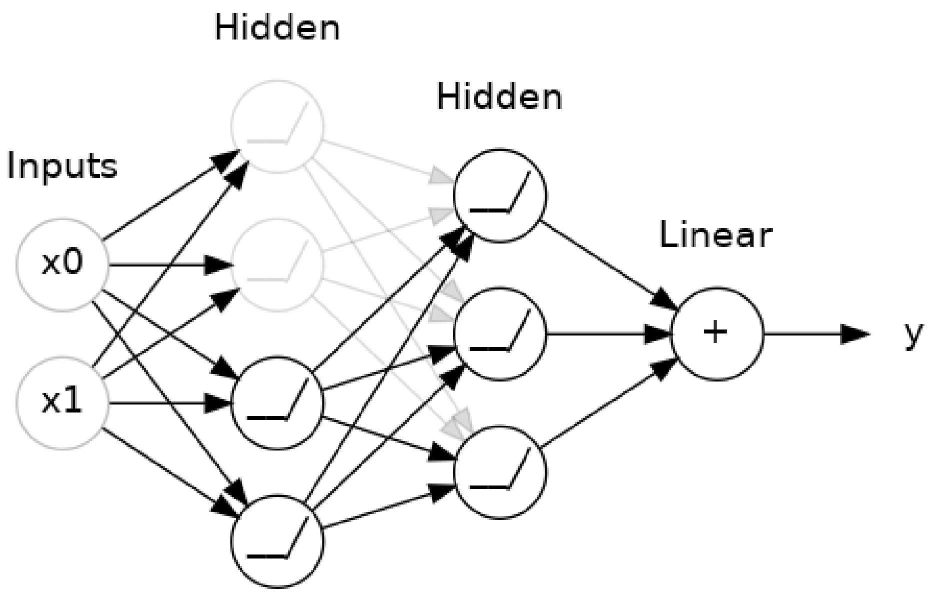
Deep Learning Training Issues

Deep Learning Training Issues

- Overfitting
- Saturating Gradient Problem
- Vanishing Gradient Problem

Overfitting in deep neural networks

- Recall: what is overfitting?
- Why can deep learning overfit?
  - Too many parameters!
- Mitigation?
  - Dropout
    - **Randomly "drop out"** some neurons during batch **training**
    - **Cannot propagate** through those neurons during training
    - Note: **all nodes** are still used for prediction



Further reading: https://towardsdatascience.com/12-main-dropout-methods-mathematical-and-visual-explanation-58cdc2112293

Deep Learning Training Issues

- Overfitting
- Saturating Gradient Problem
- Vanishing Gradient Problem

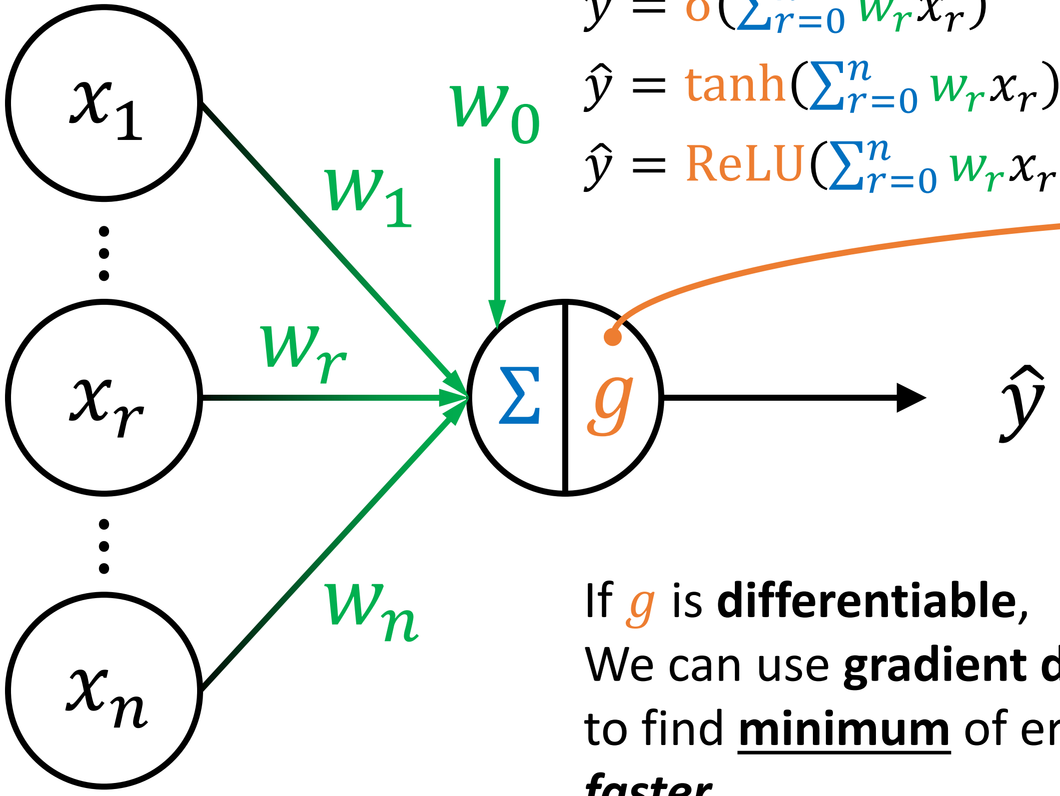
Differentiable Activation Functions

$\hat{y} = \mathrm{sgn}\left(\sum_{r=0}^{n} w_r x_r\right)$

$\hat{y} = \sigma\left(\sum_{r=0}^{n} w_r x_r\right)$

$\hat{y} = \tanh\left(\sum_{r=0}^{n} w_r x_r\right)$

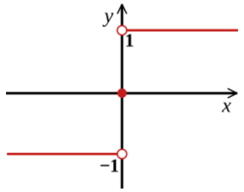
$\hat{y} = \mathrm{ReLU}\left(\sum_{r=0}^{n} w_r x_r\right)$



If g is **differentiable**,
We can use **gradient descent**
to find **<u>minimum</u>** of error $\varepsilon = y - \hat{y}$
faster

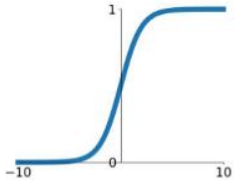
Step

$$\mathrm{sgn}(x) = \begin{cases} +1 & z > 0 \\ -1 & z \leq 0 \end{cases}$$



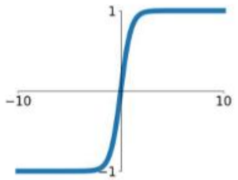
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



tanh

$$\tanh(x)$$



ReLU

$$\max(0, x)$$

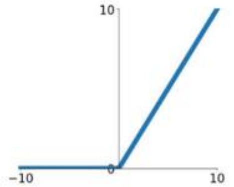


Image Credit:
https://miro.medium.com/max/1400/0*sIJ-gbjlz0zrz8lb.png

Gradient Descent Weight Update

Old weight

Direction of fastest error increase

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \nabla \varepsilon$$

New weight

Learning Rate

MSE error

$$\varepsilon = \frac{1}{2}(\hat{y} - y)^2$$

Gradient of error

$$\nabla \varepsilon = \frac{\partial \varepsilon}{\partial \boldsymbol{W}} = \frac{\partial \varepsilon}{\partial \hat{y}} \underbrace{\frac{\partial \hat{y}}{\partial \boldsymbol{W}}}_{\frac{\partial f}{\partial \boldsymbol{W}} \frac{\partial g}{\partial f}}$$

Reference

$$\boldsymbol{a}^{[l]} = g^{[l]}(f^{[l]})$$

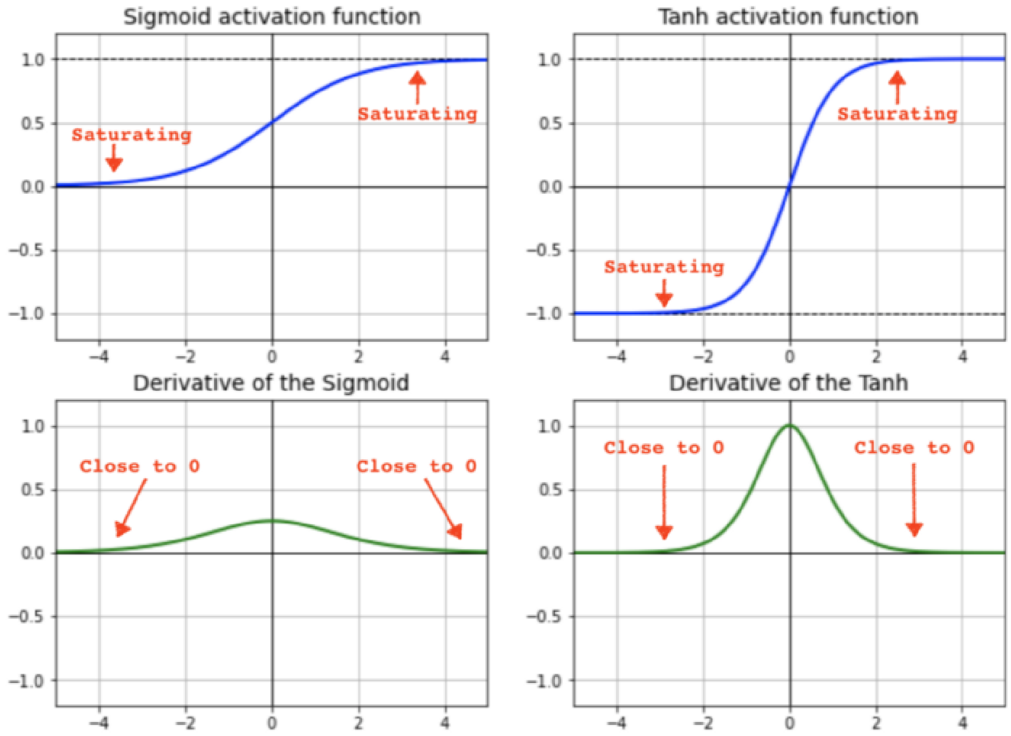
$$f^{[l]} = \left(W^{[l]}\right)^{\top} \boldsymbol{a}^{[l-1]}$$

Saturating Gradient Problem due to activation functions

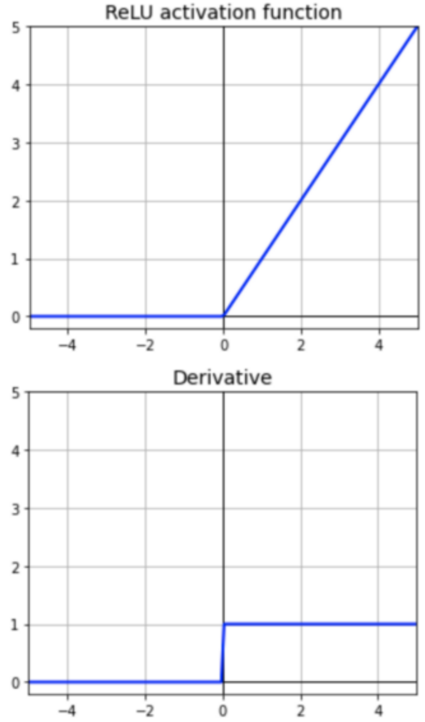
Mitigate with ReLU activation function

$g(f)$

$\frac{\partial g}{\partial f}$



Mitigation



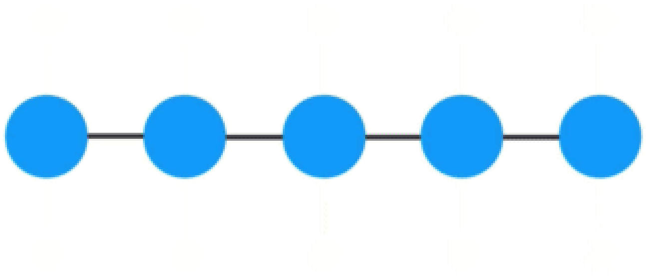
When x value far from 0, gradient $\rightarrow 0$ (saturating)
When gradient ≈ 0, then weights don't update much

$$\Delta \boldsymbol{W} = \eta \nabla \varepsilon$$

With ReLU, gradient is always 1 (for $x > 0$)
Can always update weights (for $x > 0$)

Image credit: https://towardsdatascience.com/why-rectified-linear-unit-relu-in-deep-learning-and-the-best-practice-to-use-it-with-tensorflow-e9880933b7ef

Vanishing Gradient Problem



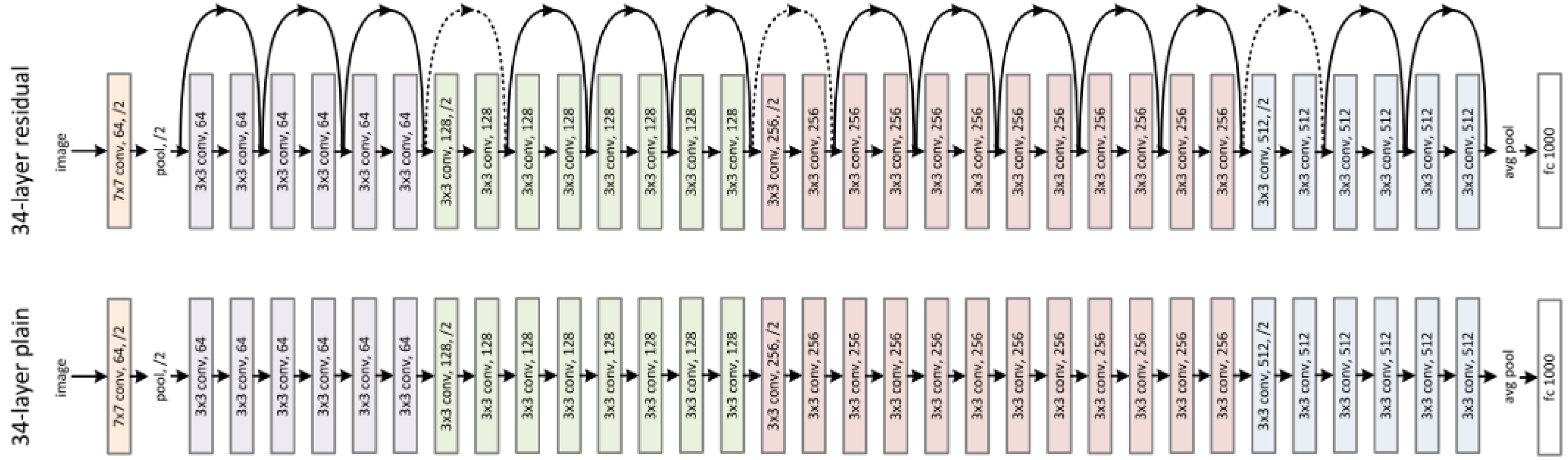
$$\hat{y}'\left(\boldsymbol{W}^{[1]}\right) = \frac{\partial g^{[L]}}{\partial \boldsymbol{W}^{[1]}} = \frac{\partial f^{[1]}}{\partial \boldsymbol{W}^{[1]}} \frac{\partial g^{[1]}}{\partial f^{[1]}} \cdots \frac{\partial g^{[l]}}{\partial f^{[l]}} \frac{\partial f^{[l+1]}}{\partial g^{[l]}} \frac{\partial g^{[l+1]}}{\partial f^{[l+1]}} \cdots \frac{\partial f^{[L]}}{\partial g^{[L-1]}} \frac{\partial g^{[L]}}{\partial f^{[L]}}$$

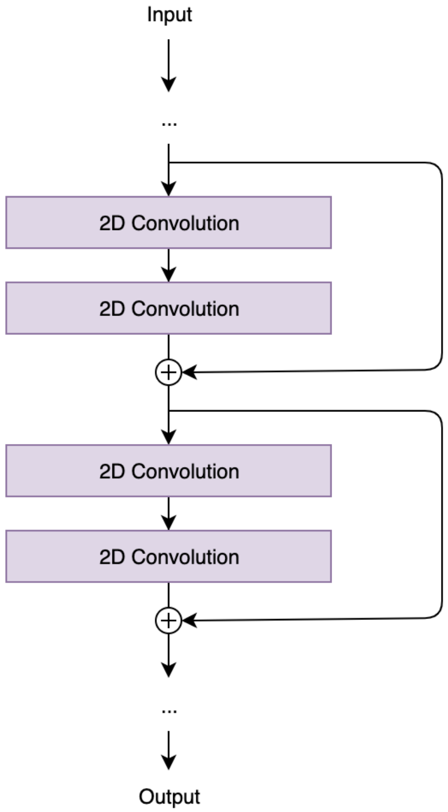
If some gradients are small (< 1),
multiplying many small numbers equals a very small number.
E.g., $0.5^{15} \approx 0.0003$

Image credit: https://towardsdatascience.com/understanding-rnns-lstms-and-grus-ed62eb584d90

Mitigating Vanishing Gradients in CNN:
Using architecture with “shortcut” connections

- ResNet (Residual Networks)
- Propagates residuals (forward) and gradients (backwards) through “**shortcut connections**”
- Gradients through shortcuts will not be as small

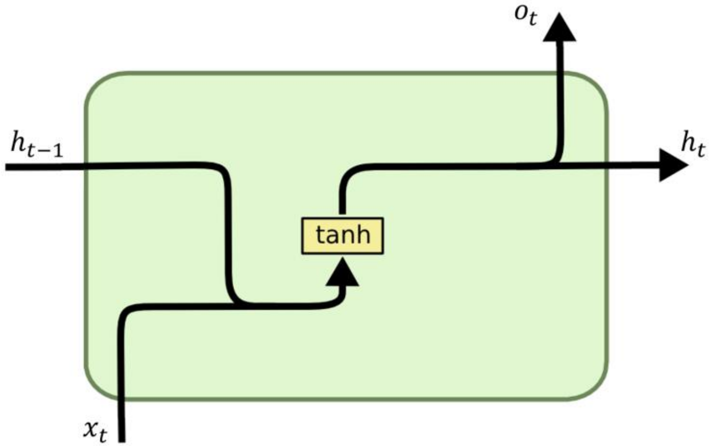




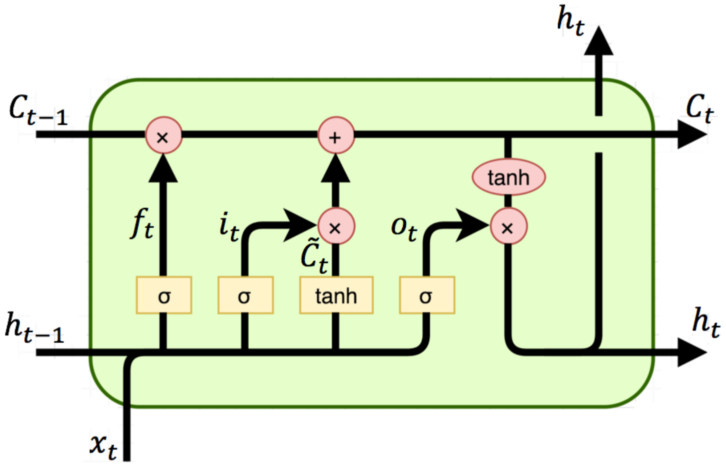
Further reading: https://towardsdatascience.com/vggnet-vs-resnet-924e9573ca5c
Image credit: https://www.kaggle.com/keras/resnet50

Mitigating Vanishing Gradients in RNN
Using architectures with "forget" gates

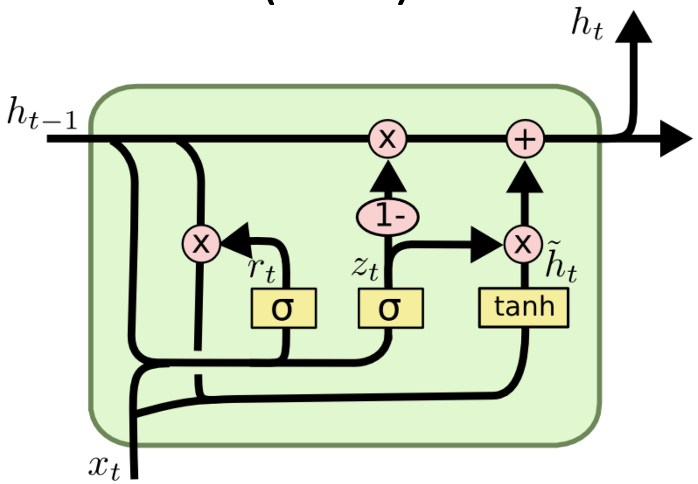
Plain RNN



Long-Short Term Memory (LSTM)



Gated Recurrent Unit (GRU)



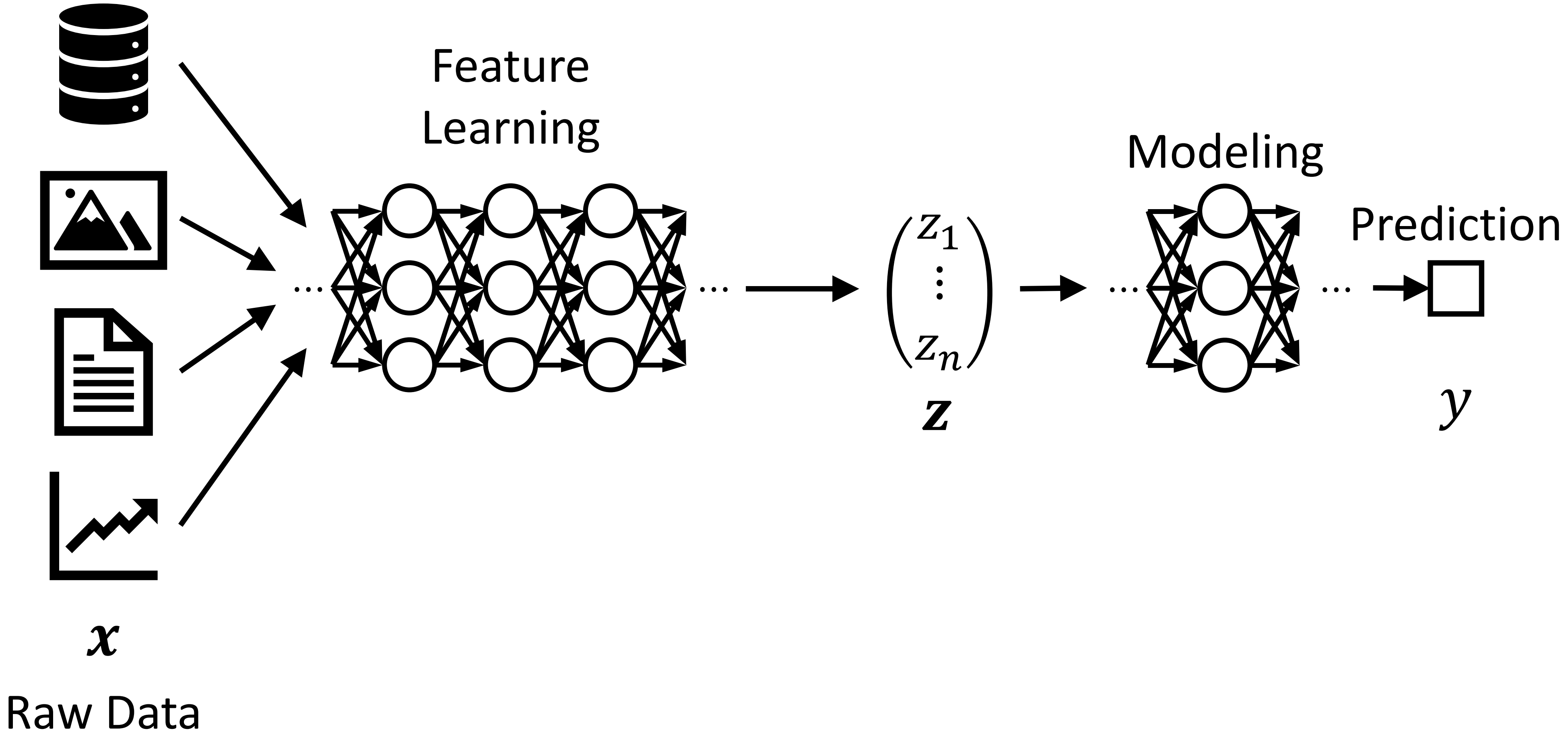
Includes **"forget" gates**

Image Credit: http://dprogrammer.org/rnn-lstm-gru
Further reading: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Wrapping Up

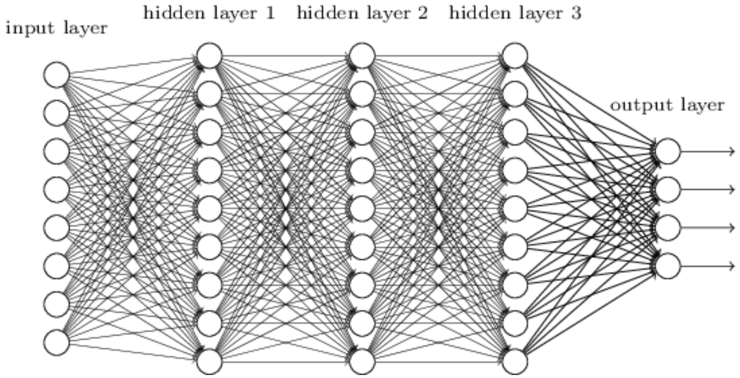
Department of Computer Science
School of Computing

From Manual Feature Engineering To Architecture Engineering

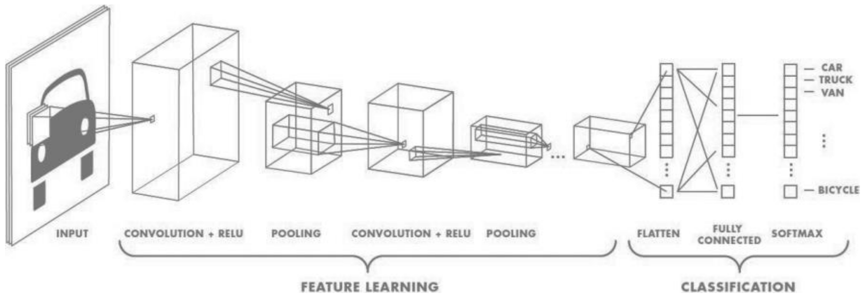


What did we learn?

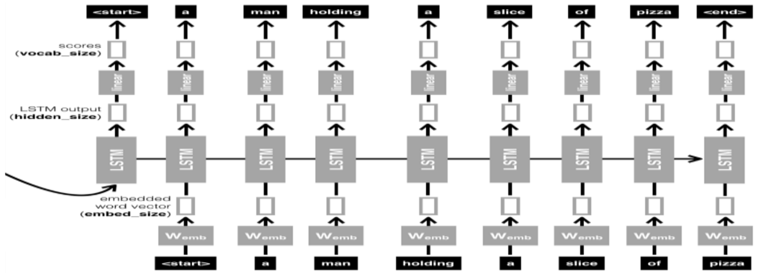
- Feature Engineering → Architecture Engineering



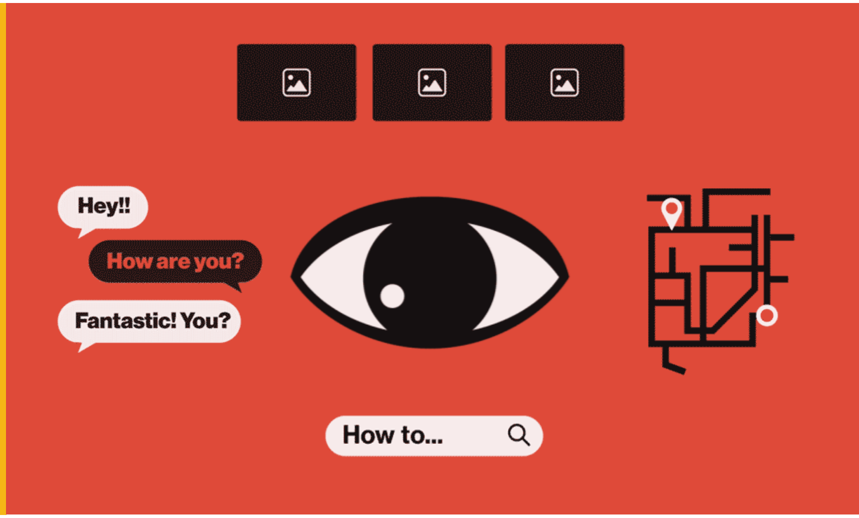
- **C**NN: exploits <u>spatial information</u> using **convolutions**



- **R**NN: exploits <u>history information</u> using **recurrence**



Grand issues with AI (Deep Learning)

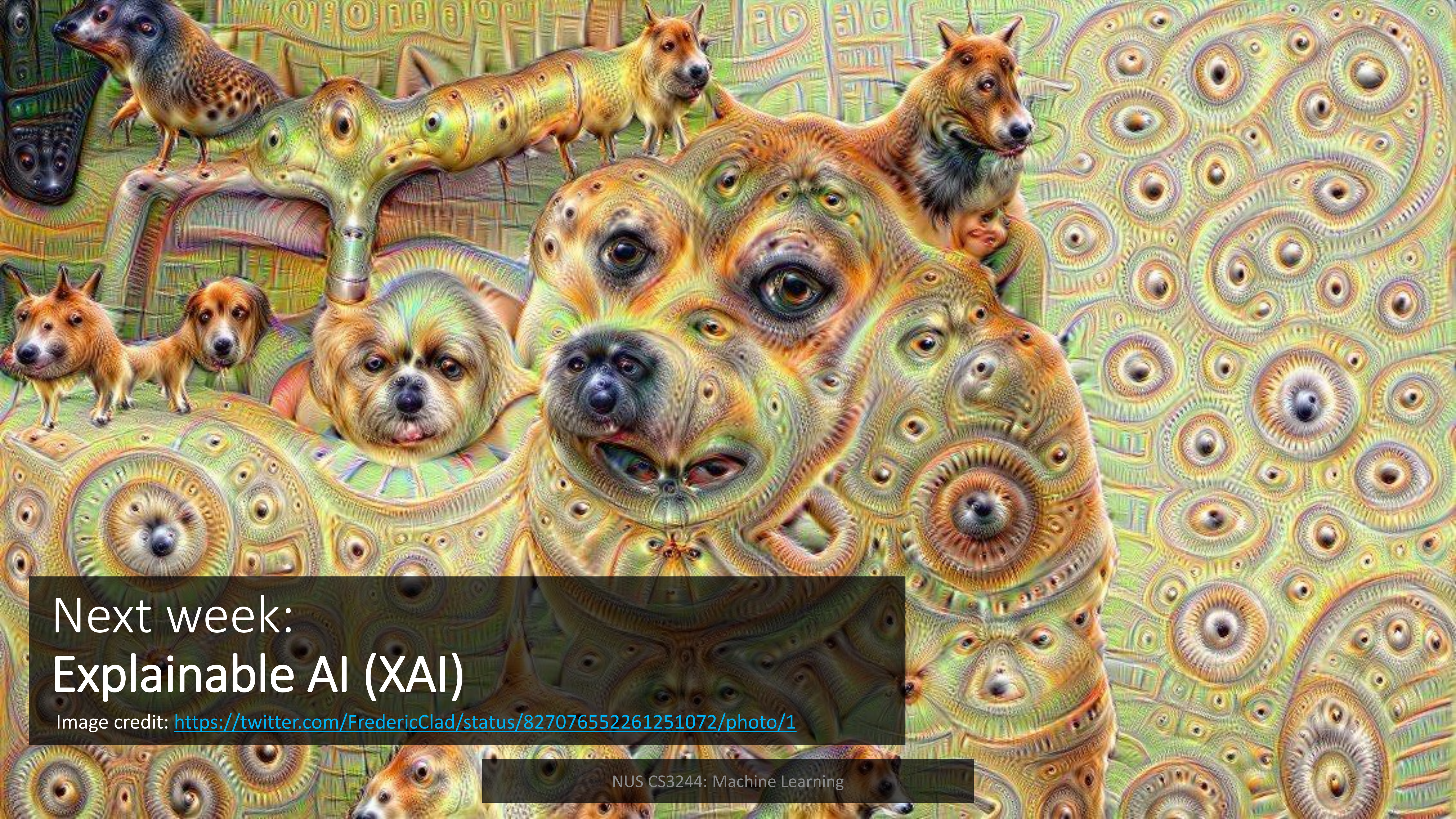


Lack of **Explainability**
[W11a]

Algorithmic Bias (Societal)
[W12a]

Data **Privacy**

Image credits:
https://miro.medium.com/max/2000/1*H4cW-_RCyHpu5FNtVaAPoQ.gif
https://www.insperity.com/wp-content/uploads/bias_1200x630.png
https://www.fightforprivacy.co/_nuxt/img/512f421.gif



Next week:
Explainable AI (XAI)

Image credit: https://twitter.com/FredericClad/status/827076552261251072/photo/1

Next week:
Unsupervised Learning

Image credit: https://hip2save.com/2019/11/27/lego-classic-creative-fun-900-piece-set-only-20-at-walmart-regularly-40/

W11 Pre-Lecture Task (due before next Thu)

Read

1. Clustering With More Than Two Features? Try This To Explain Your Findings by Mauricio Letelier

Task

1. Describe other use cases where you need to **apply domain knowledge** with data-driven **unsupervised learning** to better understand your business or engineering problem

   Tip: you can your own projects too; you don't have to be correct

2. Post a 1–2 sentence answer to the topic in your tutorial group: #tg-xx