Deep Learning (RNN+DNN)

10 B

CS 3244 Machine Learning

NUS National University of Singapore | Computing

Recap from W10a

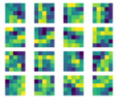
Convolutional Layer: Feature Kernels & Feature Maps

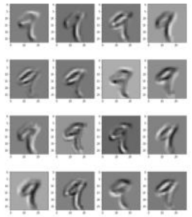
$k^{[l-1]} = c^{[l]}$
filters from previous layer $k^{[l-1]}$ is equal to #channels into current layer $c^{[l]}$

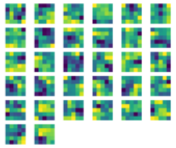
$$\boldsymbol{X}^{[0]} \rightarrow g^{[1]}\left(\boldsymbol{W}^{[1]} * \boldsymbol{X}^{[0]}\right) = \boldsymbol{A}^{[1]} \xrightarrow{\text{Pooling}} g^{[2]}\left(\boldsymbol{W}^{[2]} * \boldsymbol{X}^{[1]}\right) = \boldsymbol{A}^{[2]}$$

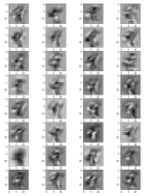
1 channel | 16 filters | 16 channels | 32 filters | 32 channels


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

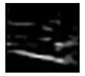

224 × 224 → downsampling → 112 × 112


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

Fully-Connected Neuron vs. Convolutional Activation Map

$$a_\rho^{[l]} = g^{[l]}\left(\sum_r^{n^{[l-1]}} w_{r\rho}^{[l]} a_r^{[l-1]}\right) \quad \text{vs.} \quad \boldsymbol{A}_k^{[l]} = g^{[l]}\left(\sum_c^{c^{[l-1]}} \boldsymbol{W}_{ck}^{[l]} * \boldsymbol{A}_c^{[l-1]}\right)$$

Weights → **Kernels**

Sum across channels

2D Convolution

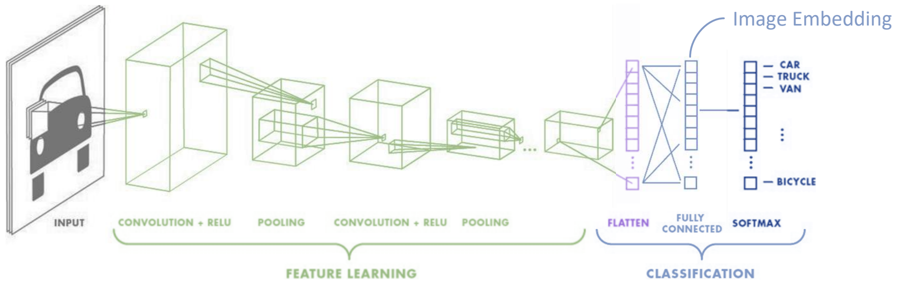
Activation function can be the same

Activation is a 2D Matrix for each channel

Layer stores activation of **each neuron** ρ separately

Layer stores activation map of **each kernel** k separately

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

Week 10A&B: 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

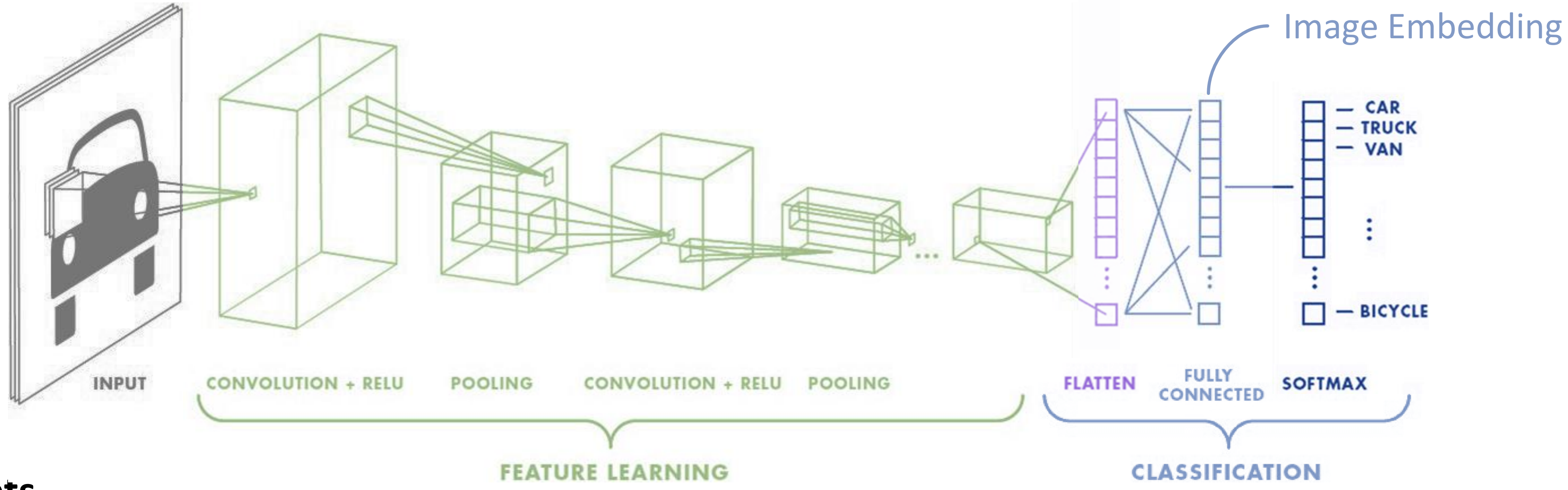
Week 10B: Lecture Outline

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

Convolutional Neural Network



Key concepts

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

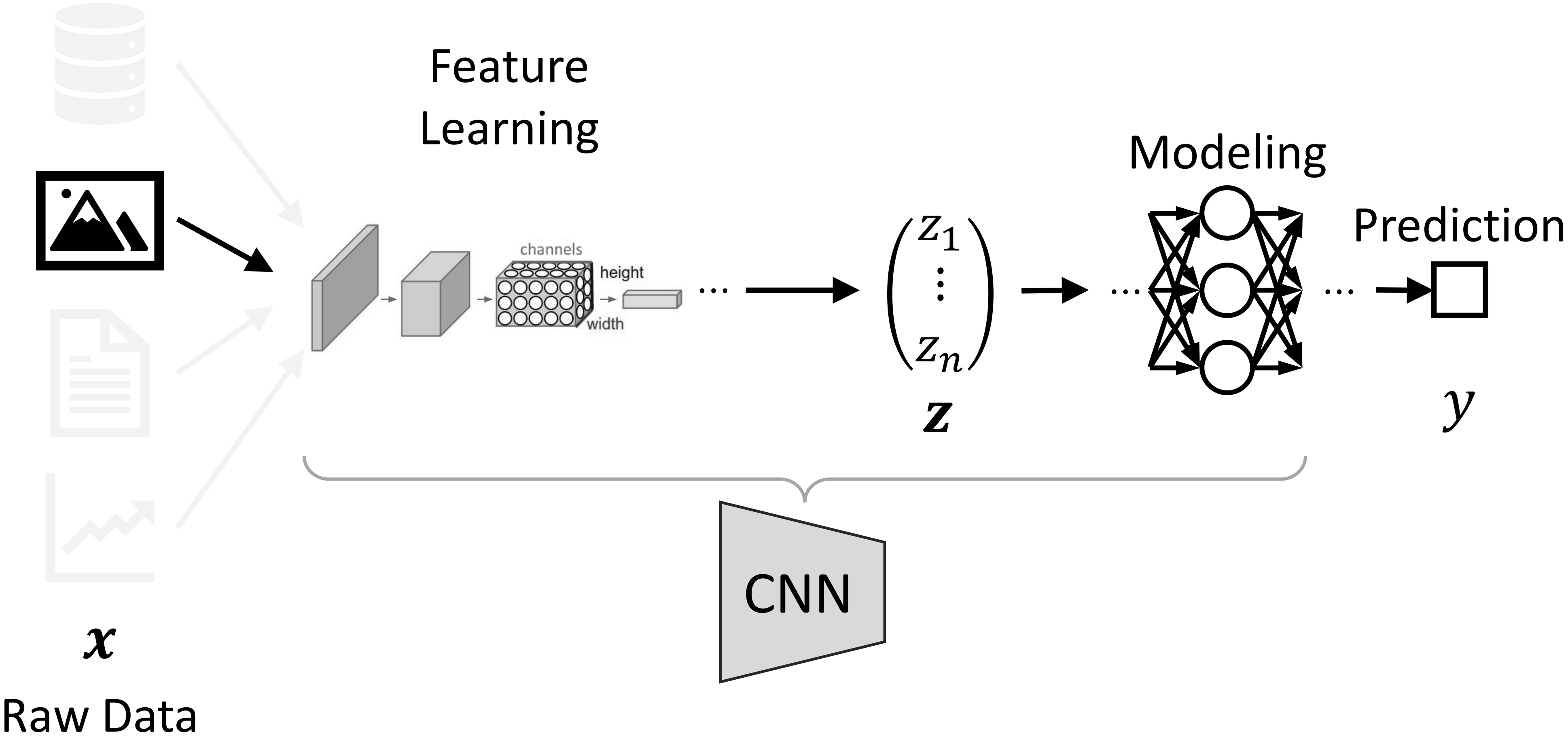
❷ Flattening
- Convert to fixed-length 1D vector

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

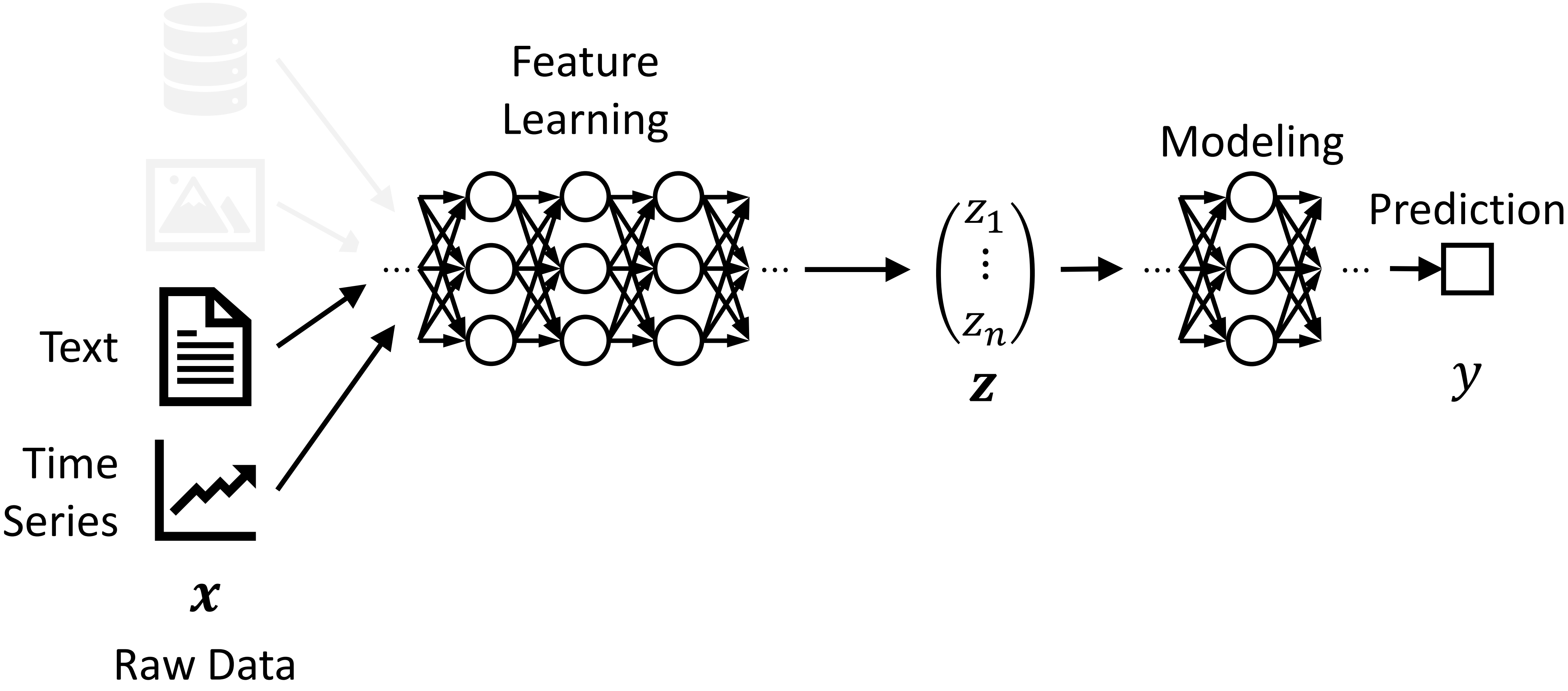
❹ Classification
- Softmax ≔ Multiclass Logistic Regression
- Feature input = image embedding vector (typically large vector)
- Analogy: decision making

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

From Manual Feature Engineering To Automatic Feature Learning



From Manual Feature Engineering To Automatic Feature Learning



Recurrent Neural Networks (RNN)

Department of Computer Science
School of Computing

Applications of RNN

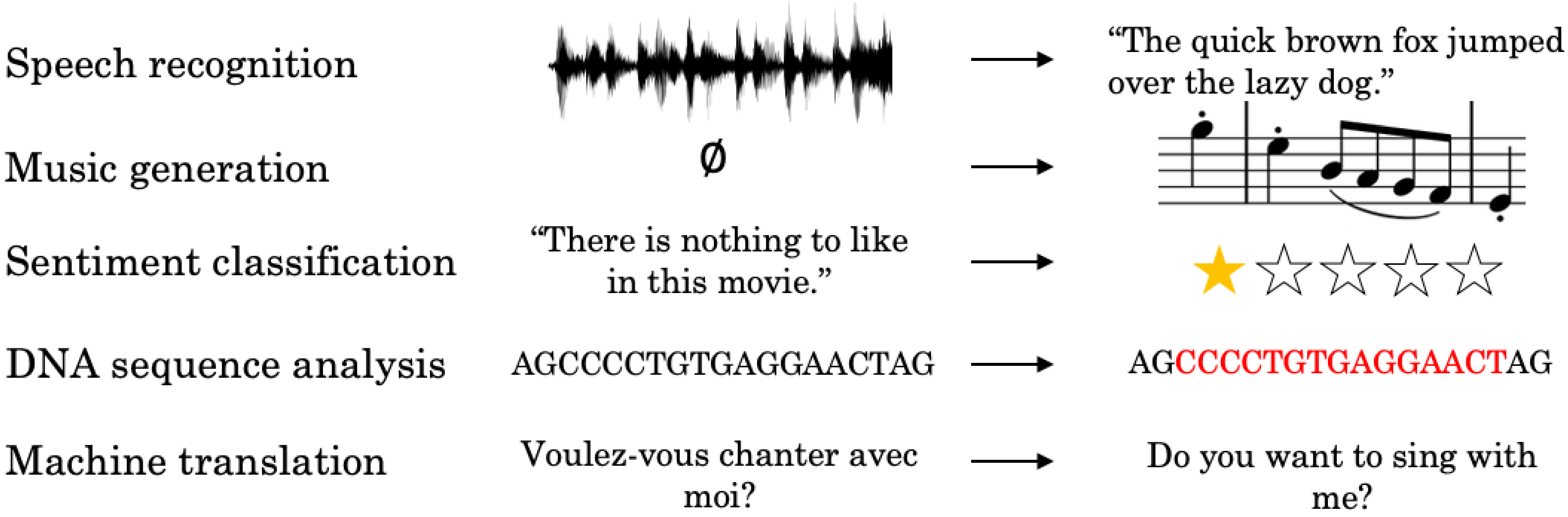


Image credit: https://laptrinhx.com/understanding-of-recurrent-neural-networks-lstm-gru-3720007533/

info random clear

Model: butterfly

Exercise E10B.1

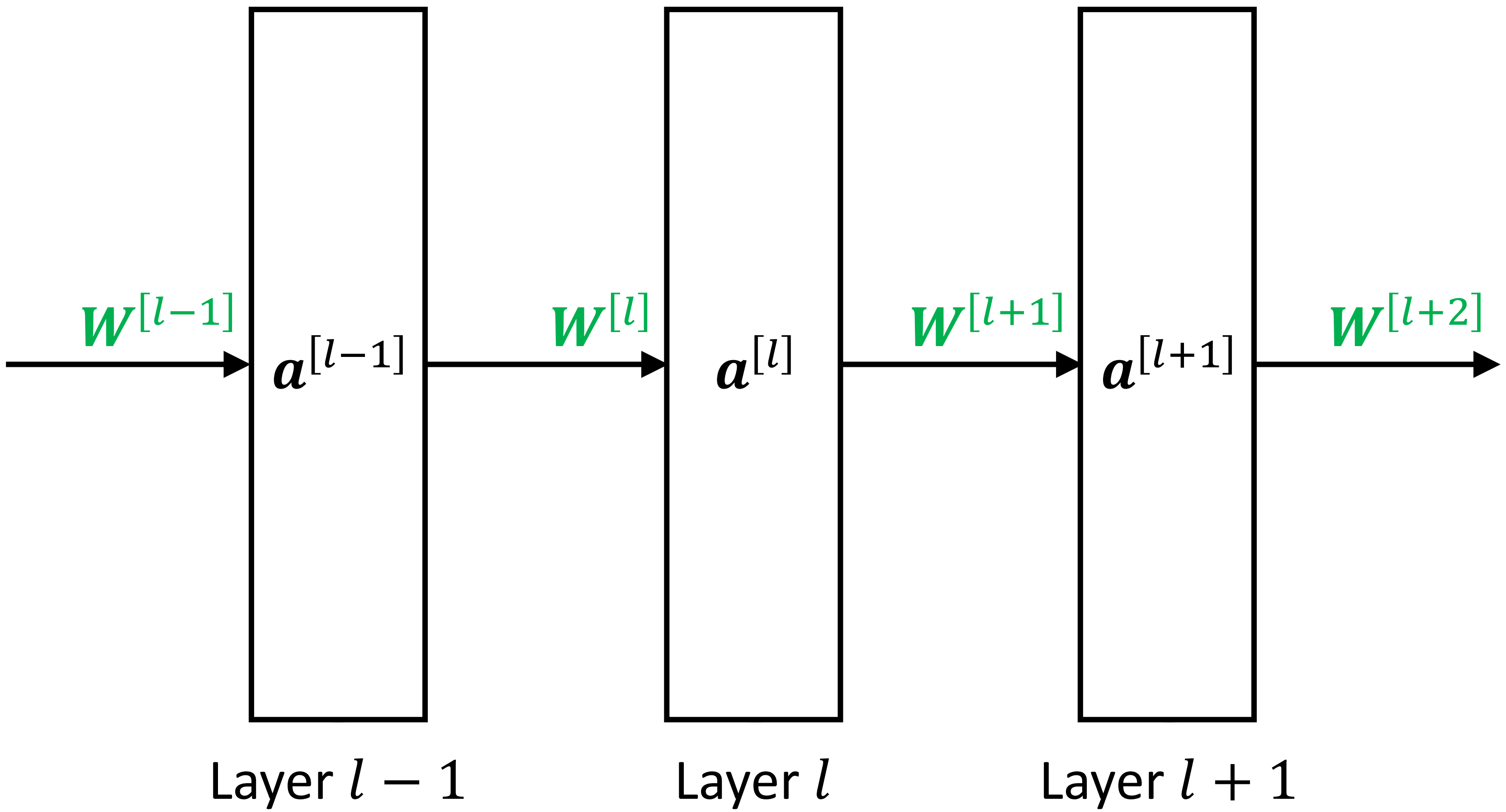
draw a shape, any shape.

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

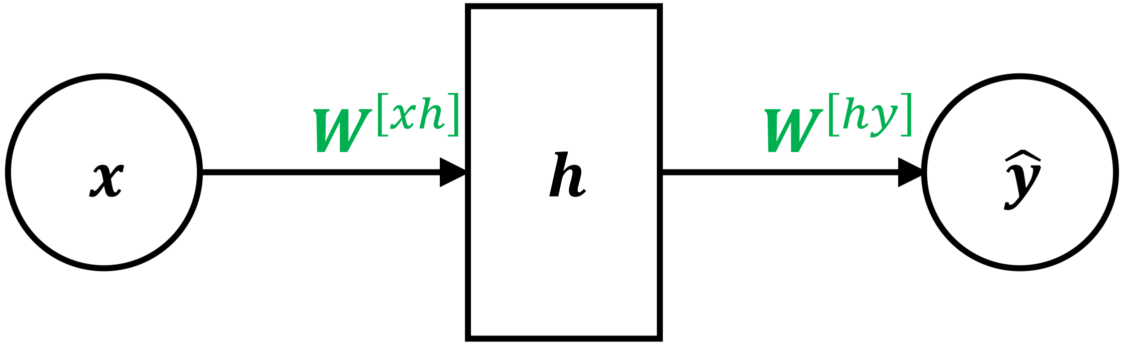
Try yourself: https://magenta.tensorflow.org/assets/sketch_rnn_demo/index.html

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

Feedforward Neural Network

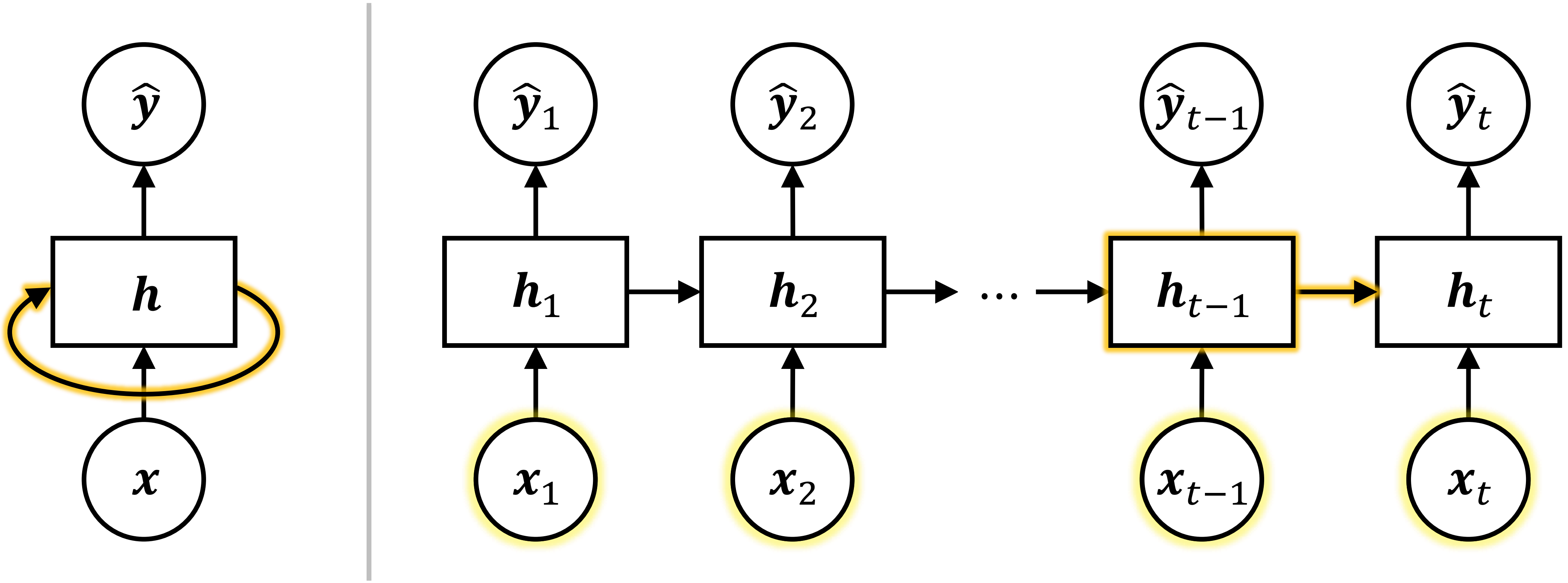


Neural Network (simplified 1-hidden layer)



Input Layer Hidden Layer Output Layer

Neurons with Recurrence



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

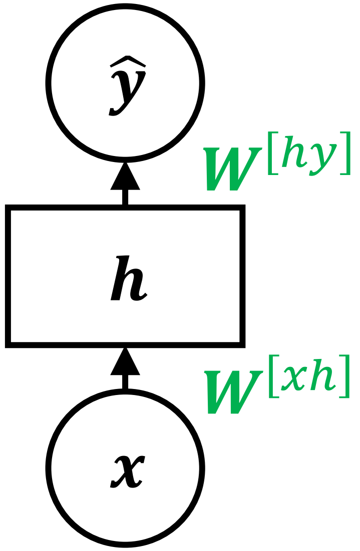
Recurrent Neural Network (RNN)

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

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

RNN Weights

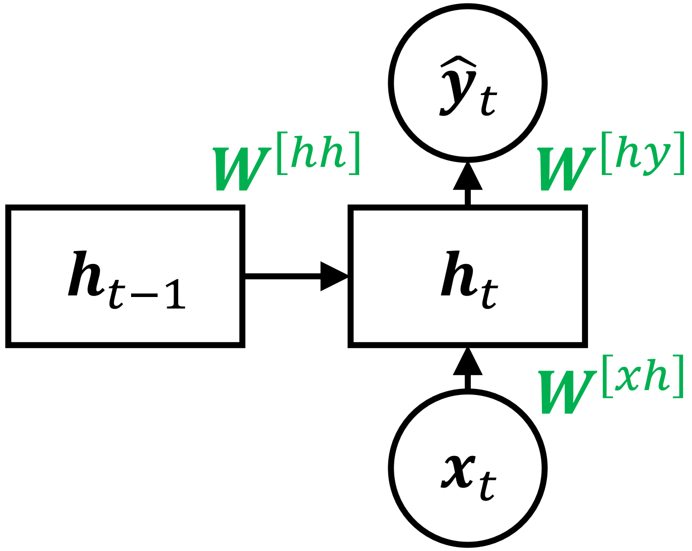
Question: Do these weights change for different time t?



Feedforward Neural Network

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

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



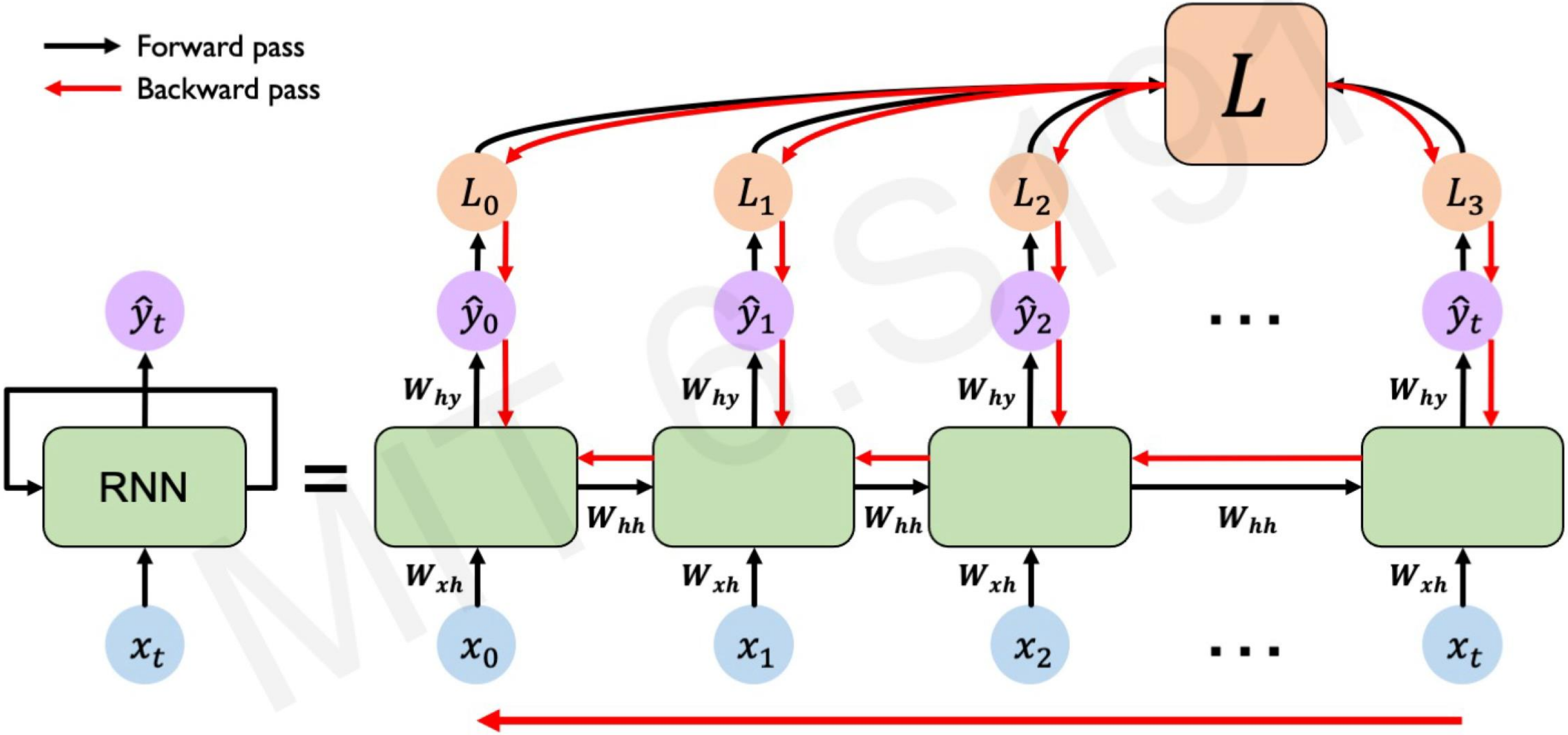
Recurrent Neural Network

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

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


2021

RNNs: Backpropagation Through Time



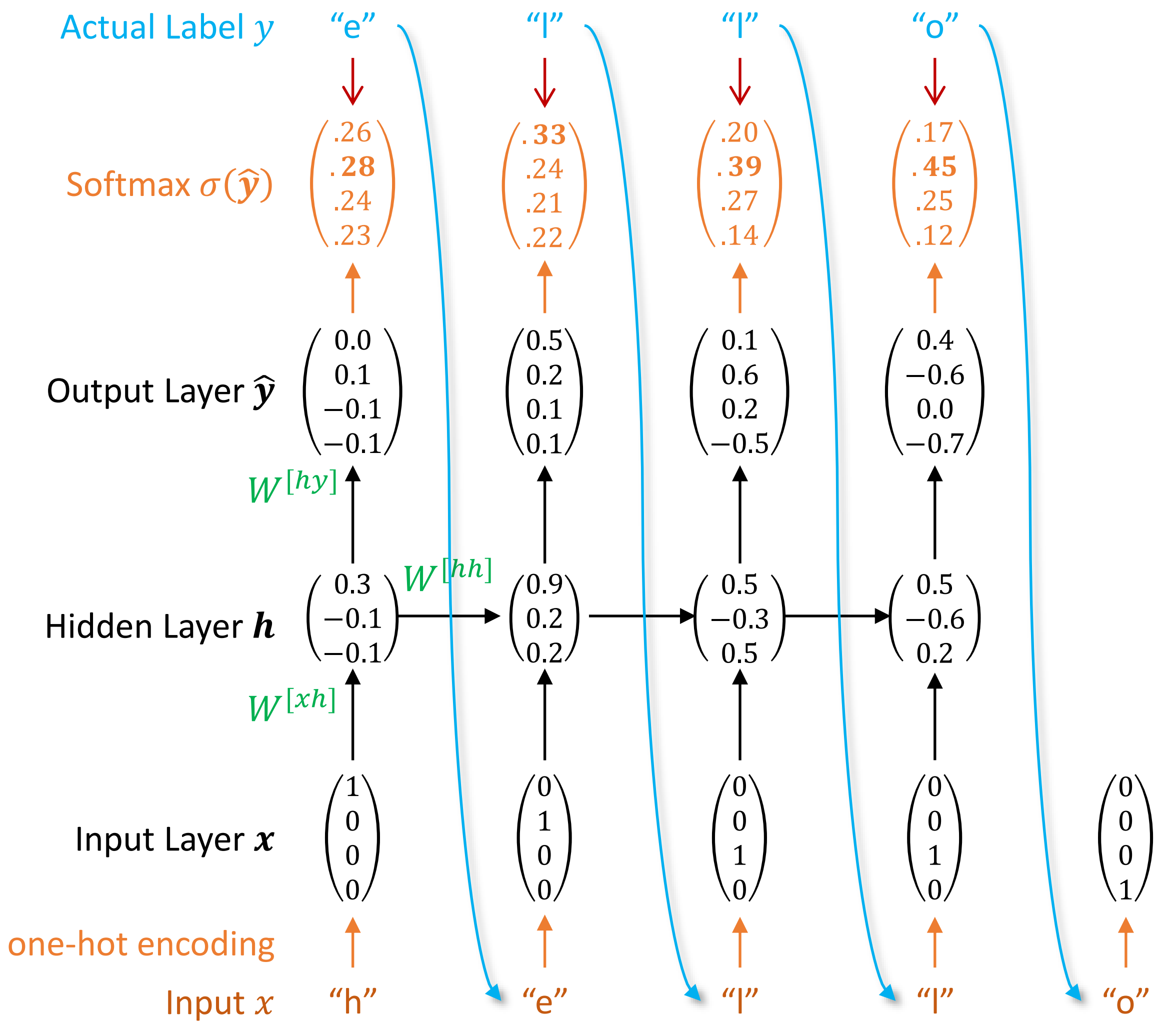
Mozer *Complex Systems* 1989. 1/19/21

$$W^{[xh]} = \begin{pmatrix} 0.3 & 1.0 & 0.1 & 0.5 \\ -0.1 & 0.3 & -0.3 & 0.1 \\ -0.1 & 0.1 & -0.5 & -0.5 \end{pmatrix} \quad W^{[hy]} = \begin{pmatrix} 0.3 & 0.9 & 0.1 \\ 0.2 & -0.6 & 0.5 \\ -0.1 & 0.1 & 0.5 \\ -0.1 & 1.0 & -0.2 \end{pmatrix} \quad W^{[hh]} = \begin{pmatrix} 0.1 & 0.4 & 0.8 \\ -0.1 & 0.5 & -0.2 \\ 0.9 & 0.2 & 0.6 \end{pmatrix} \quad \sigma(\hat{\boldsymbol{y}}) = \frac{e^{\hat{\boldsymbol{y}}}}{\mathbf{1} \cdot (\mathbf{1} + e^{\hat{\boldsymbol{y}}})}$$

Example RNN

Training text prediction

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

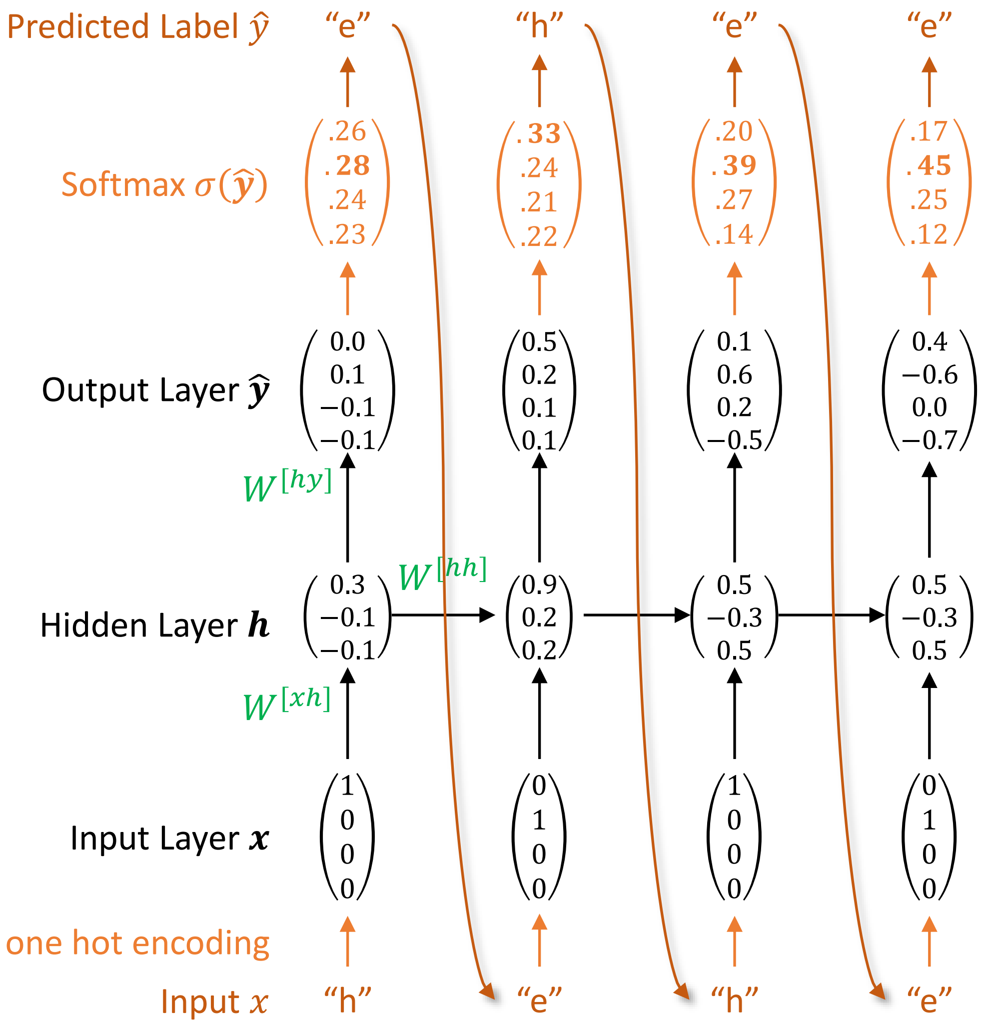


$$W^{[xh]} = \begin{pmatrix} 0.3 & 1.0 & 0.1 & 0.5 \\ -0.1 & 0.3 & -0.3 & 0.1 \\ -0.1 & 0.1 & -0.5 & -0.5 \end{pmatrix} \quad W^{[hy]} = \begin{pmatrix} 0.3 & 0.9 & 0.1 \\ 0.2 & -0.6 & 0.5 \\ -0.1 & 0.1 & 0.5 \\ -0.1 & 1.0 & -0.2 \end{pmatrix} \quad W^{[hh]} = \begin{pmatrix} 0.1 & 0.4 & 0.8 \\ -0.1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

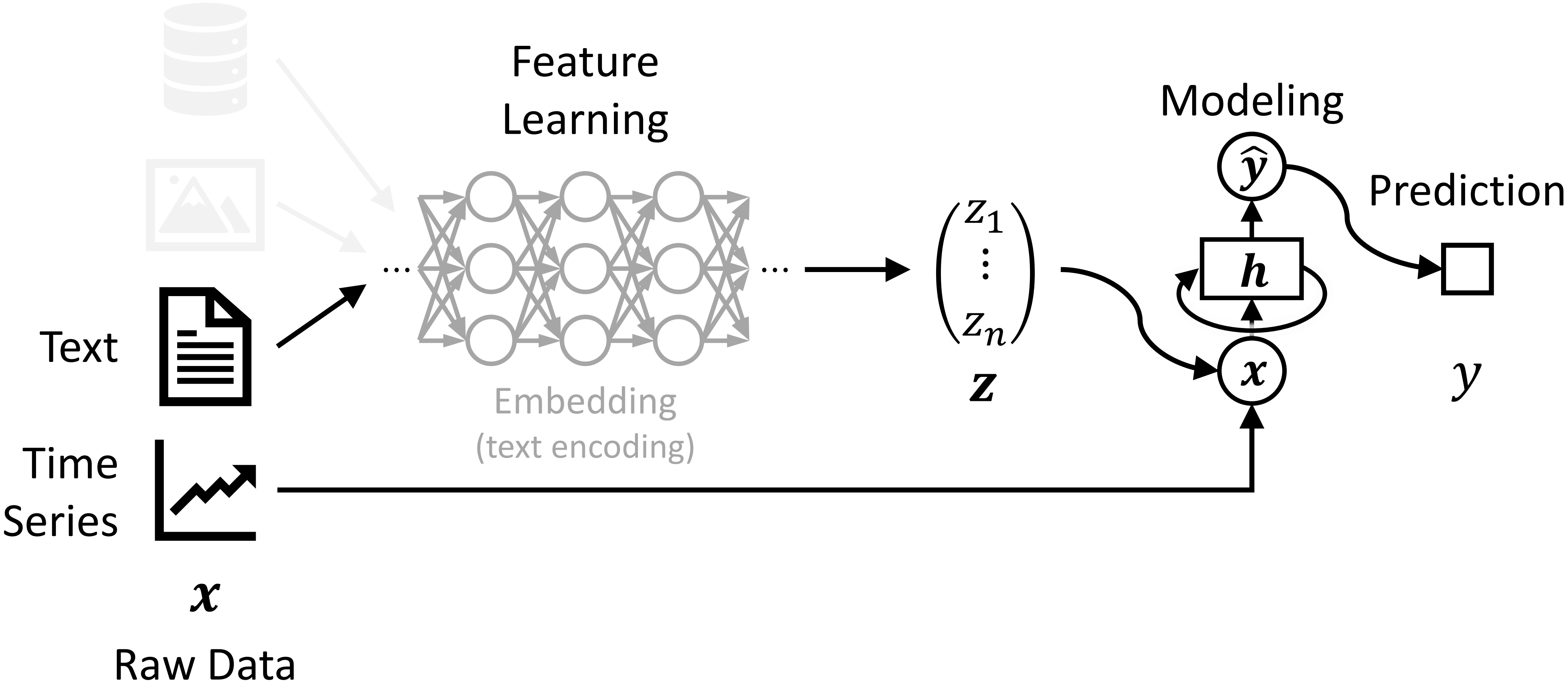
Example RNN

Predicting Text

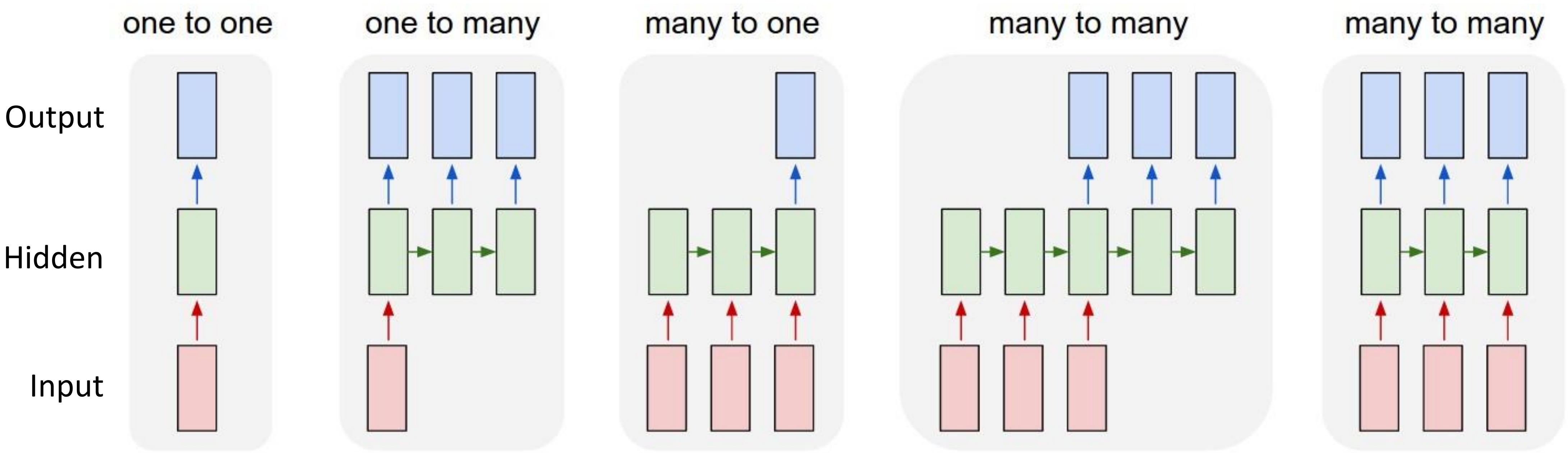
- At prediction time,
  - Forward propagate calculating activations to generate sequence of characters
  - $x_t = \hat{y}_{t-1}$



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

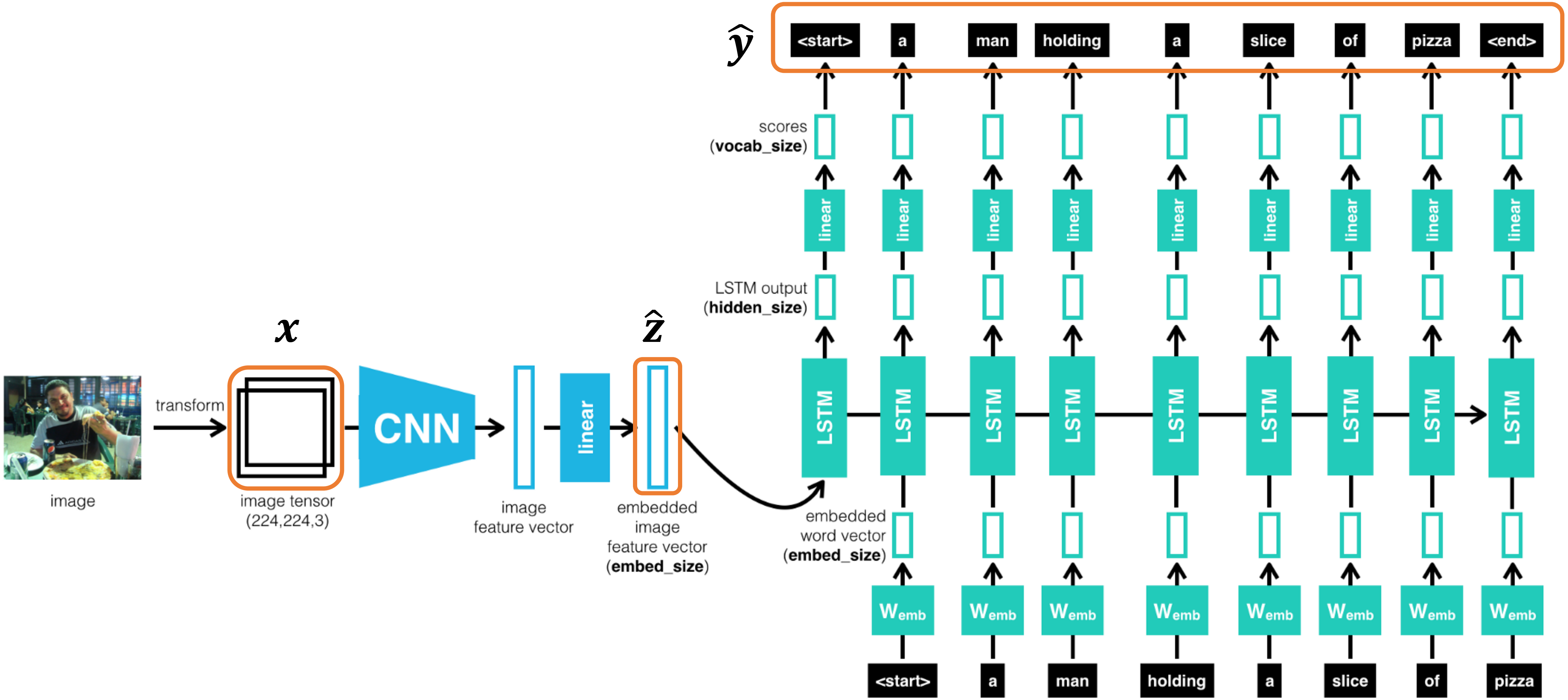
Encoder-Decoder
Processes *all inputs first*, then predicts output

Plain RNN with *time dependency*

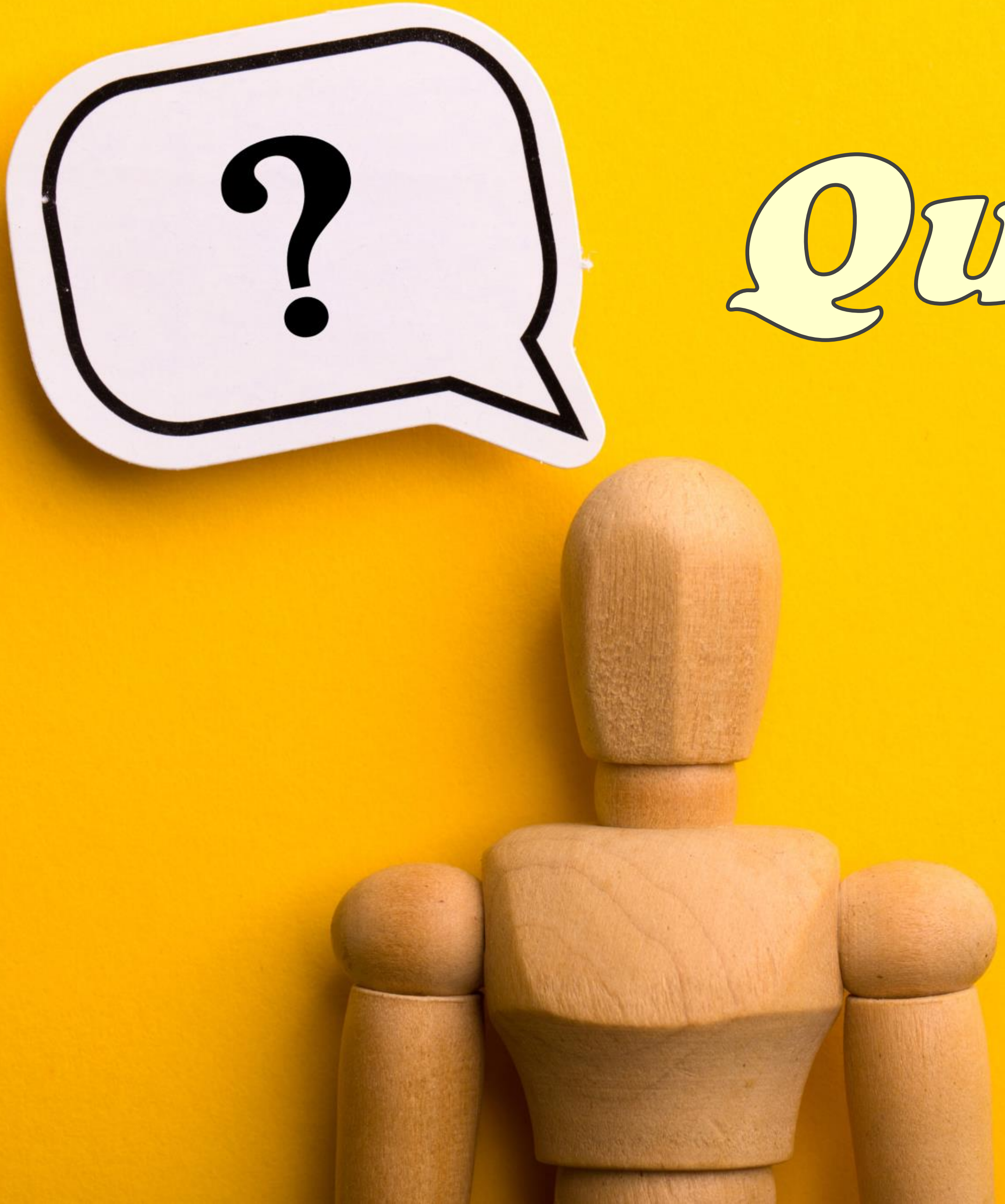
Processes 1 input *at a time*

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

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



Questions!



Deep Learning Training Issues

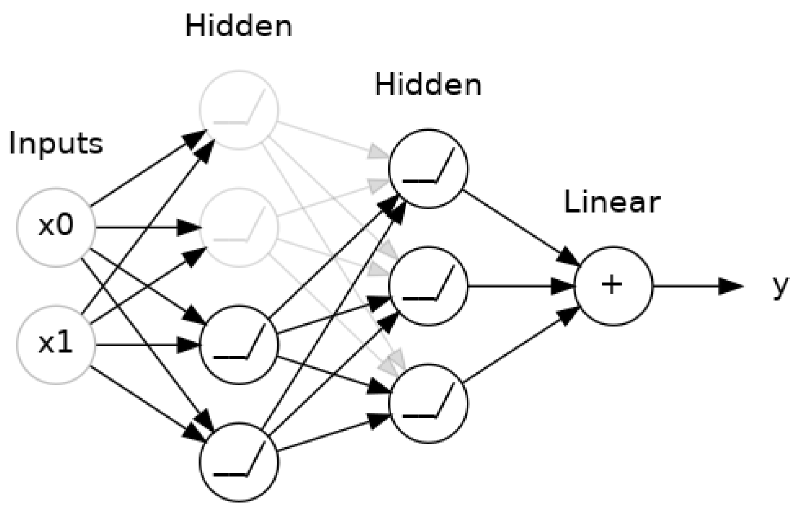
NUS National University of Singapore | Department of Computer Science School of Computing

Deep Learning Training Issues

- Overfitting
- Saturating Gradient Problem
- Vanishing Gradient Problem

Overfitting in deep neural networks

- Recall: what is overfitting?
  - Performance: Validation < Training
- Why can deep learning overfit?
  - Too many parameters!
  - Model is more expressive than neeeded
- 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

Gradient Descent Weight Update (Neural Network)

Direction of fastest error increase

Old weight

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

New weight

Learning Rate

Gradient of error

$$\nabla\varepsilon = \frac{d\varepsilon}{d\boldsymbol{W}} = \frac{d\varepsilon}{d\hat{y}} \underbrace{\frac{d\hat{y}}{d\boldsymbol{W}}}_{\frac{df}{d\boldsymbol{W}}\frac{dg}{df}}$$

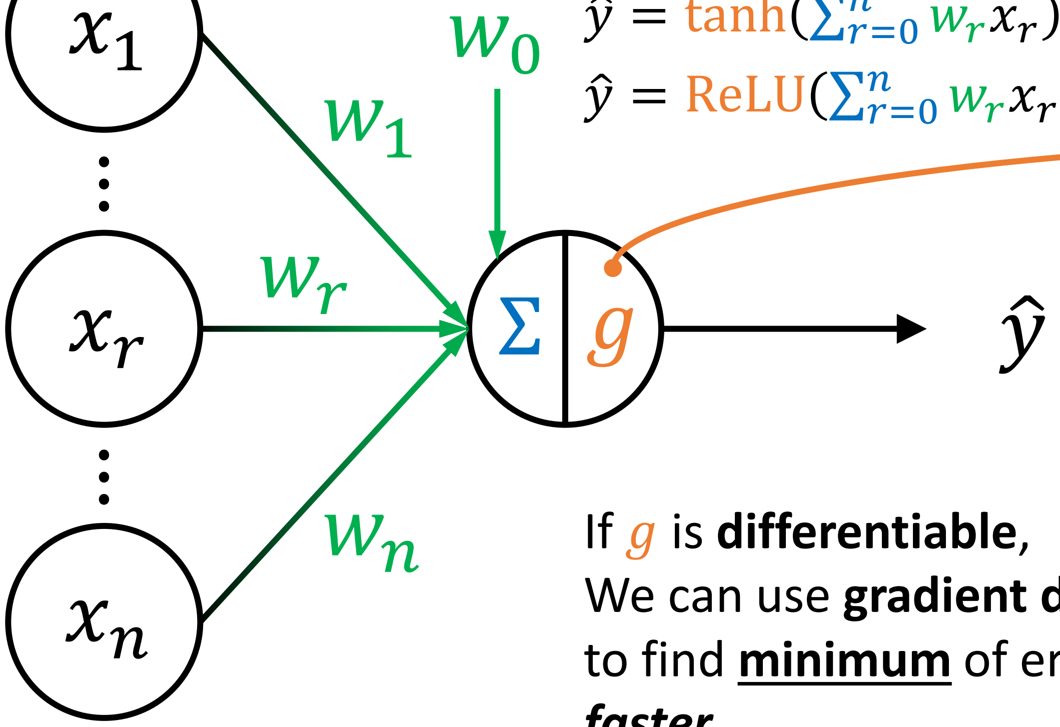
Differentiable Activation Functions

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

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

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

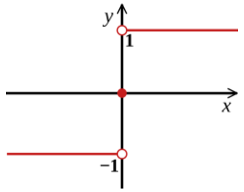
$\hat{y} = \mathrm{ReLU}(\sum_{r=0}^{n} w_r x_r)$



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

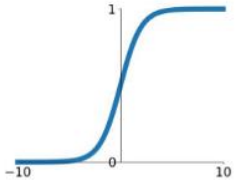
Step

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



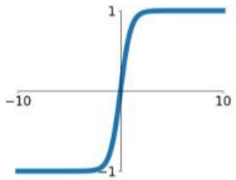
Sigmoid

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



tanh

$\tanh(x)$



ReLU

$\max(0, x)$

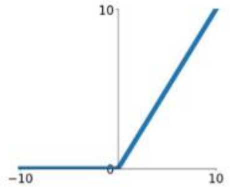
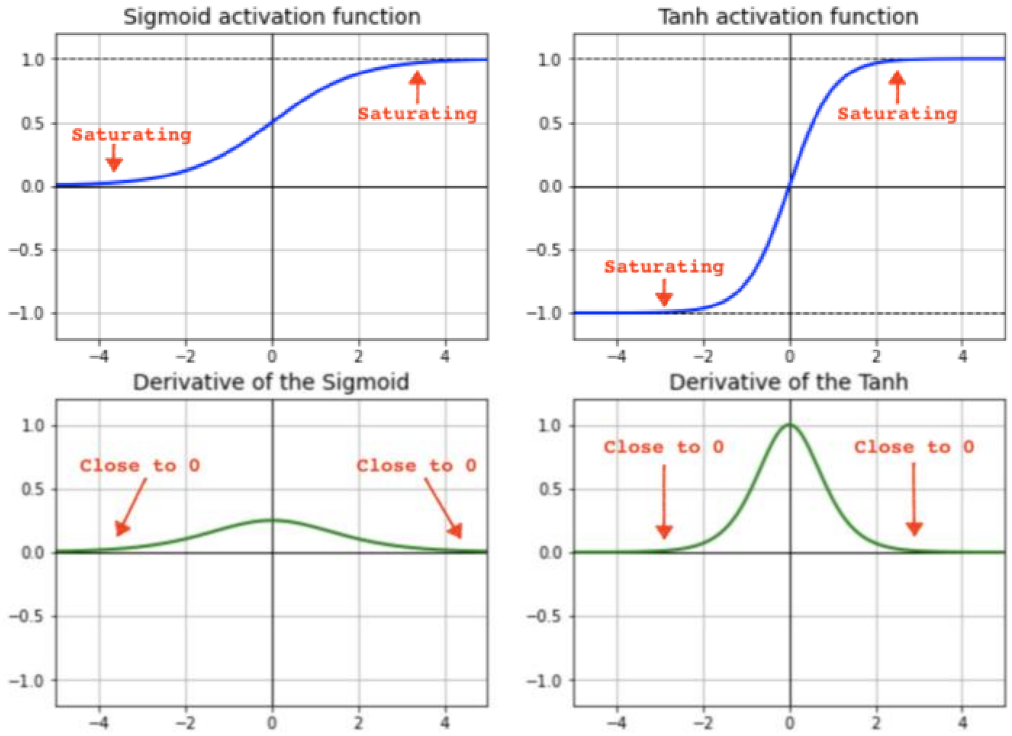


Image Credit:
https://miro.medium.com/max/1400/0*slJ-gbjlz0zrz8lb.png

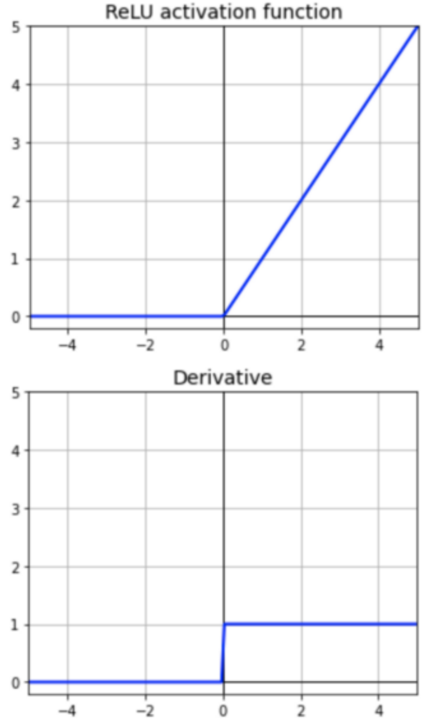
Saturating Gradient Problem due to activation functions
Mitigate with ReLU activation function

$g(f)$

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



Mitigation

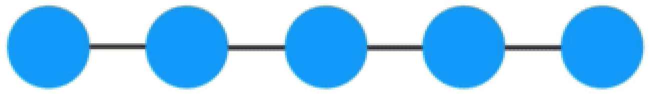


When x value far from 0, gradient $\rightarrow 0$ (saturating)
When gradient ≈ 0, then $\Delta \boldsymbol{W} = \eta \nabla \varepsilon$ weights don't update much

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



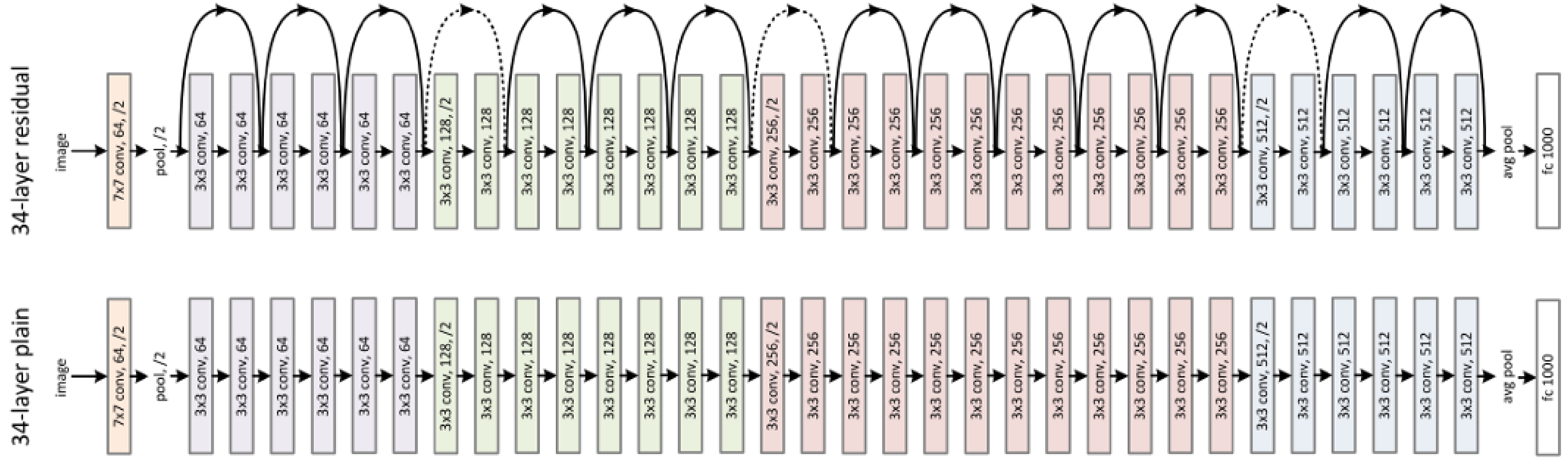
$$\frac{\partial \hat{y}}{\partial \boldsymbol{W}^{[1]}} = \frac{df^{[1]}}{d\boldsymbol{W}^{[1]}} \frac{dg^{[1]}}{df^{[1]}} \cdots \frac{dg^{[l]}}{df^{[l]}} \frac{df^{[l+1]}}{dg^{[l]}} \frac{dg^{[l+1]}}{df^{[l+1]}} \cdots \frac{df^{[L]}}{dg^{[L-1]}} \frac{dg^{[L]}}{df^{[L]}}$$

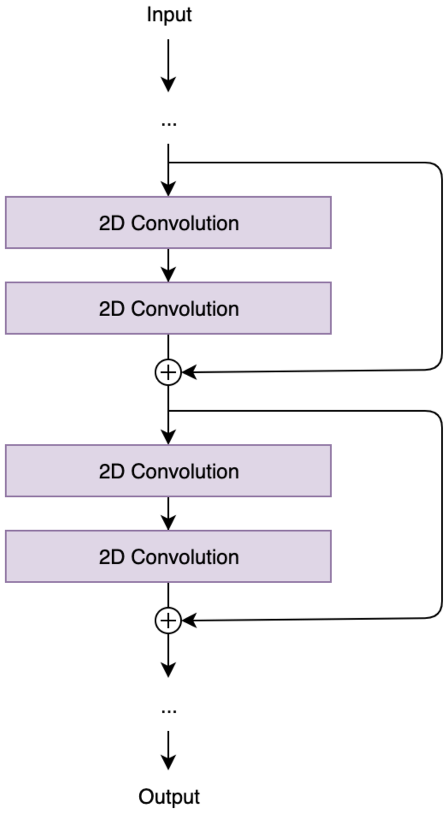
If some gradients are small (< 1),
multiplying many small numbers equals a very small number.
E.g., $0.5^{15} \approx 0.0003$

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

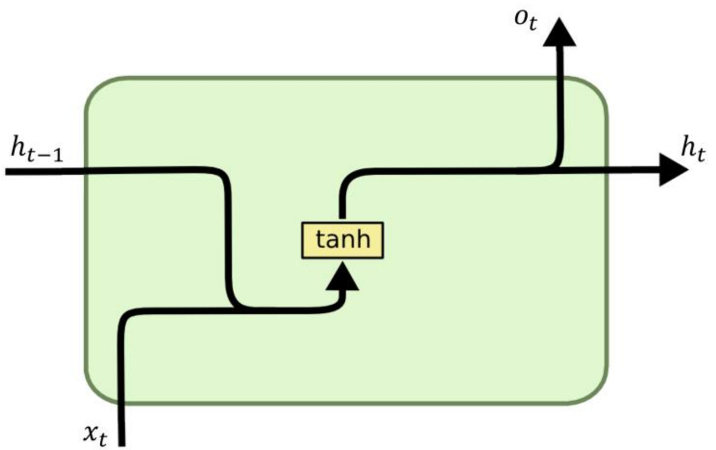




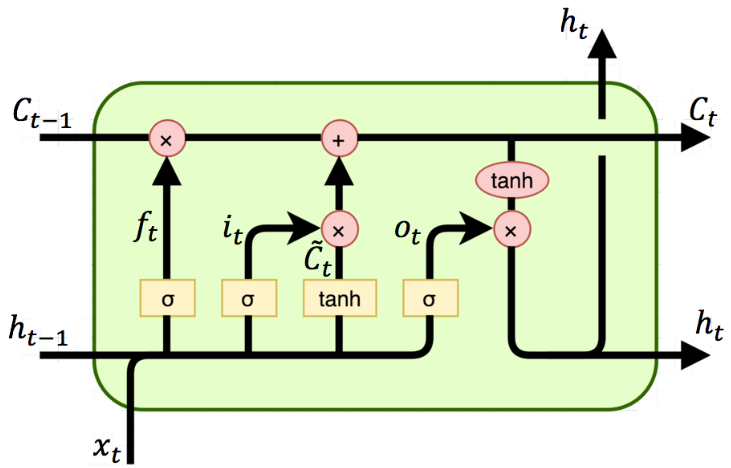
Further reading: https://towardsdatascience.com/vggnet-vs-resnet-924e9573ca5c
Image credit: https://www.kaggle.com/keras/resnet50

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

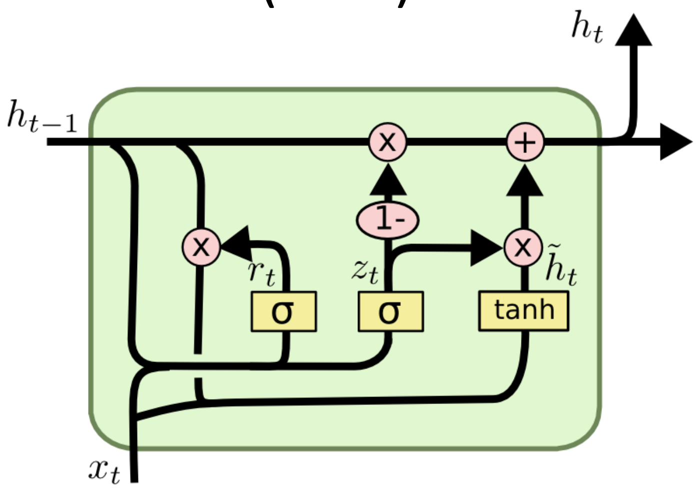
Plain RNN



Long-Short Term Memory (LSTM)



Gated Recurrent Unit (GRU)

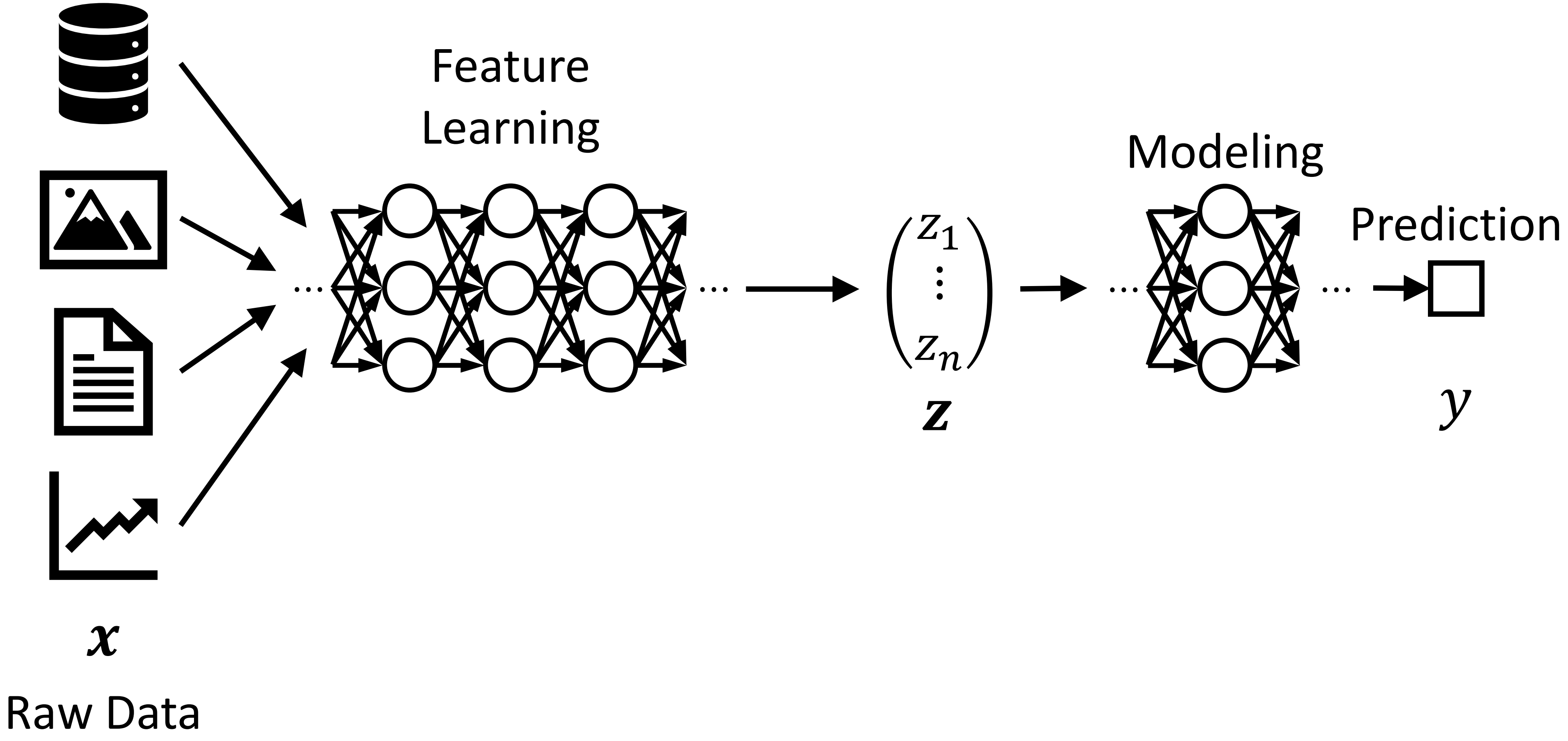


Includes **"forget" gates**

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

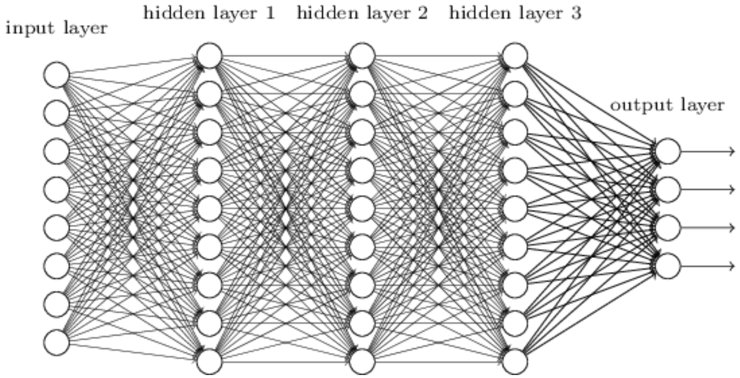
Wrapping Up

From Manual Feature Engineering To Architecture Engineering

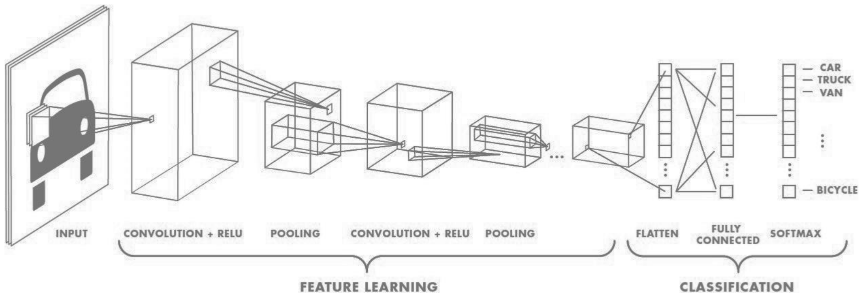


What did we learn?

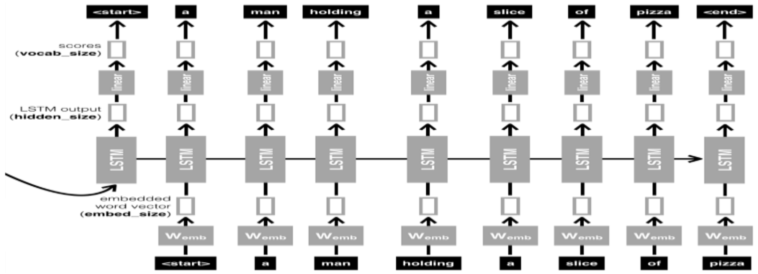
- Feature Engineering → Architecture Engineering



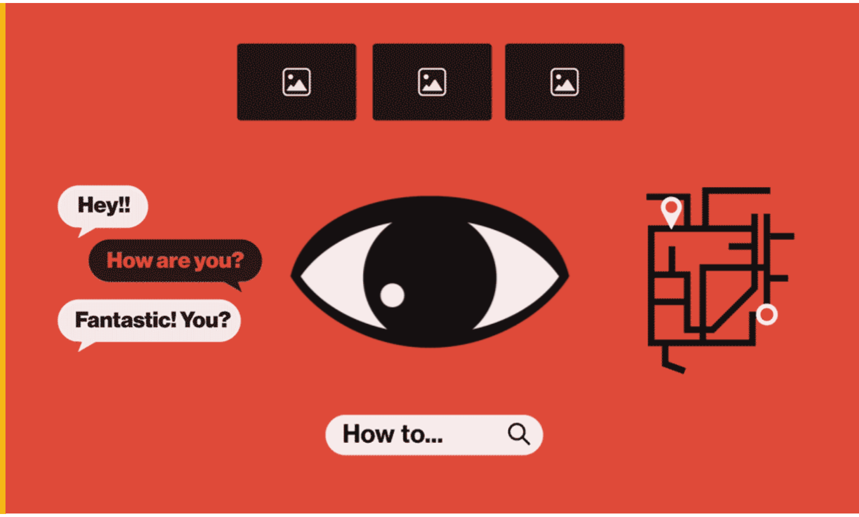
- **C**NN: exploits <u>spatial information</u> using **convolutions**



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



Grand issues with AI (Deep Learning)

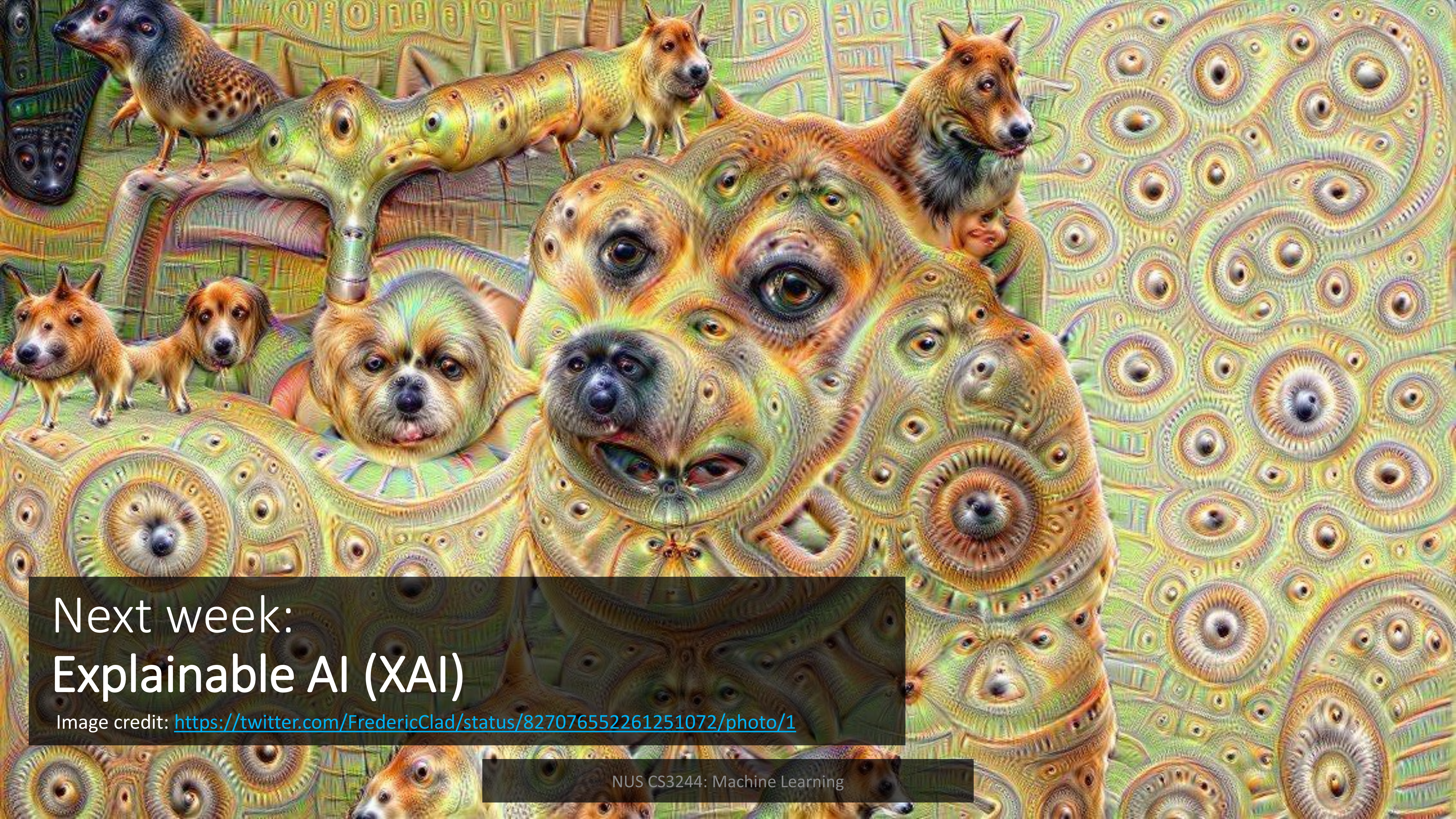


Lack of **Explainability**
[W11b]

Algorithmic Bias (Societal)
[W13a]

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

Following week:
Unsupervised Learning

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