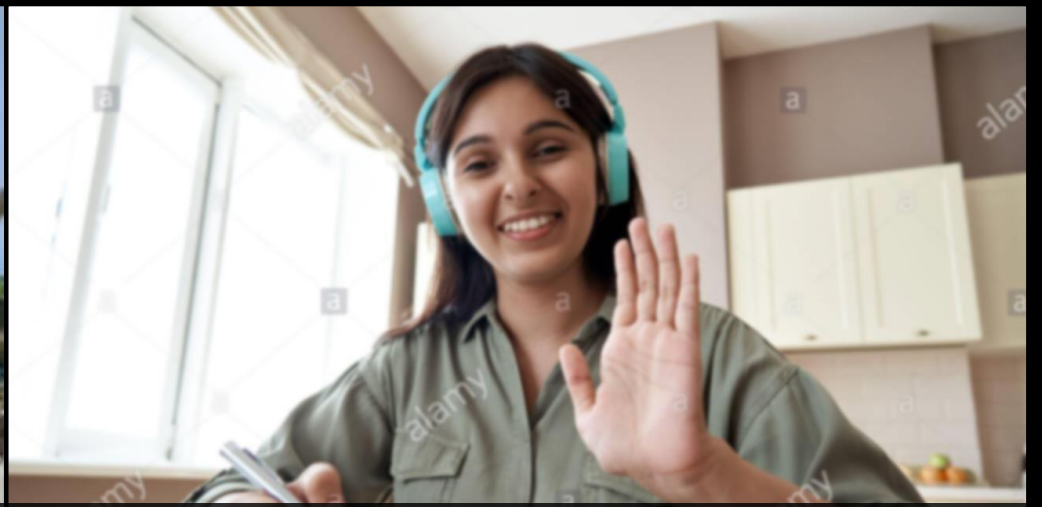
# Deep Learning

# 10 A

## CS 3244 Machine Learning

NUS | Computing
National University of Singapore



# Please **turn on** your webcam

Mystery Student

# SUSTAINABILITY IRL:

## Love Data Science? Concerned about sustainability?

## WE WANT YOU!

Partner **Treatsure**, **Charlotte Mei**, and **Upcircle** to unlock data-driven insights around topics like food waste and individual impact.

### The challenge:

Creatively visualise an answer to their problem statements, using our open dataset of >100k conversations. Or, create your own dataset.

### Prizes:

Up to $1,000 to be won

### JOIN US AT THE KICK-OFF EVENT

28th Oct (Thursday),
6 - 7.30pm Online

**Register here**

sustainabilityirl.
synthesis.partners

Hosted by synthesis

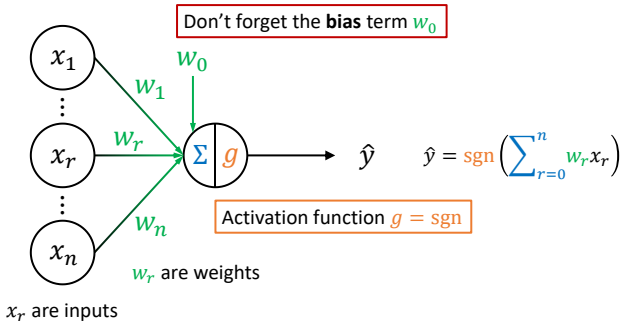
Our Partners XDS, treatsure, upcircle, TheCharlotteMei

Participate:
https://www.sustainabilityirl.synthesis.partners/

# Mid-Semester Anonymous Survey


- What worked
  - Slack is good for comms & awareness
  - Exercises (pre-lecture, in-class) are engaging
  - Easier midterm (less stressful)!
- What to improve
  - Want more coding teaching: Bonus programming lectures to be conducted by Prof Min and TAs.
  - Want more examples: Included in lecture.
  - Somewhat out-of-scope tutorial questions: We will align tutorial questions more closely with lectures.

## Perceptron






## Perceptron Learning Algorithm

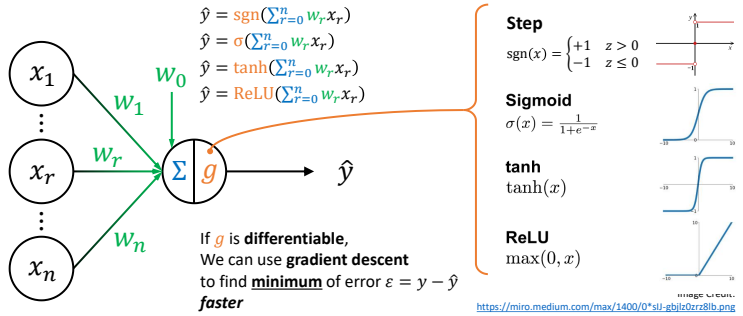
1. Initialize weights $\boldsymbol{w}$
   - Could be all zero, or random small values
2. For each instance $i$ with features $\boldsymbol{x}^{(i)}$
   - Classify $\hat{y}^{(i)} = \text{sgn}\left(\boldsymbol{w}^{\top}\boldsymbol{x}^{(i)}\right)$
3. Select one misclassified instance
   - Update weights: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta(y - \hat{y})\boldsymbol{x}$
4. Iterate steps 2 to 3 until
   - Convergence (classification error < threshold), or
   - Maximum number of iterations

$$\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_r \\ \vdots \\ w_n \end{pmatrix} \leftarrow \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_r \\ \vdots \\ w_n \end{pmatrix} + \eta(y - \hat{y}) \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_r \\ \vdots \\ x_n \end{pmatrix}$$

$$w_r \leftarrow w_r + \eta(y - \hat{y})x_r$$



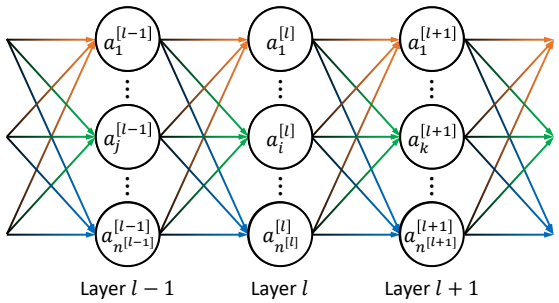
## Differentiable Activation Functions




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



## Multi-Layer Perceptron (Neural Network)






## Chain Rule

Consider composite function

$$g(x) = g\big(f(x)\big)$$

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

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

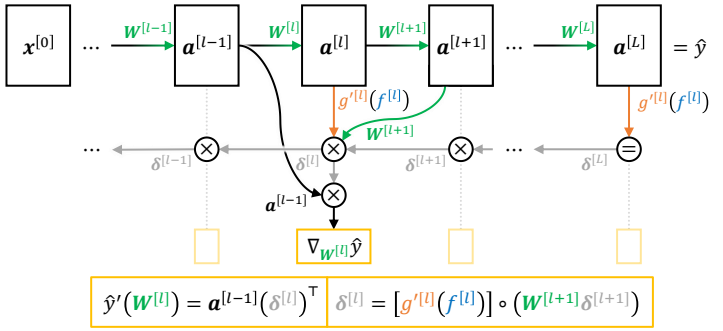
**Intuition**
Rate of change of $g$ relative to $x$ is the product of
- rates of change of $g$ relative to $f$ and
- rates of change of $f$ relative to $x$

*"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)



## Backward Propagation



# Notation

$n$ = Number of features in $\boldsymbol{x}$
$m$ = Number of instances in dataset

- **Scalar**: not bolded, lower case

$$x$$

- **Vector**: bolded, lower case

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

- **Matrix**: bolded, upper case

$$\boldsymbol{X} = \begin{pmatrix} x_{11} & \cdots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{pmatrix}$$

# Functions with Vectors and Matrices

- Scalar-by-scalar:
  - $y(x) = wx$ for scaling input
- Scalar-by-vector:
  - $y(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} = \boldsymbol{w}^\top \boldsymbol{x} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = w_1 x_1 ~~+ w_1 x_2 + w_2 x_1~~ + w_2 x_2$ for **weighted sum**
- Vector-by-vector:
  - $\boldsymbol{y}(\boldsymbol{x}) = w\boldsymbol{x} = w \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} wx_1 \\ wx_2 \end{pmatrix}$ for **scaled** outputs (same weight)

Even more Chain Rule:

# Gradient of Neural Network

$$\hat{y}(\boldsymbol{x}) = g^{[L]}(f^{[L]}(g^{[L-1]}(\cdots(g^{[l]}(f^{[l]}(g^{[l-1]}(\cdots(g^{[1]}(f^{[1]}(\boldsymbol{x}^{[0]}))))))))))$$

**Gradient** relative to $\boldsymbol{W}$

$$\hat{y}'(\boldsymbol{W}^{[L-1]}) = \frac{\partial g^{[L]}}{\partial \boldsymbol{W}^{[L-1]}} = \frac{\partial f^{[L]}}{\partial \boldsymbol{W}^{[L]}} \frac{\partial g^{[L]}}{\partial f^{[L]}} \quad \boldsymbol{\delta}^{[L-1]}$$

Reference

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

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

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

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

**Recursive**

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

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

# 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

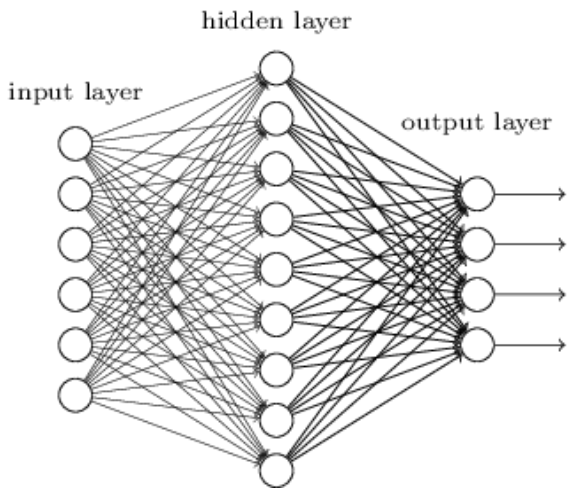
# Week 10A: Lecture Outline

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

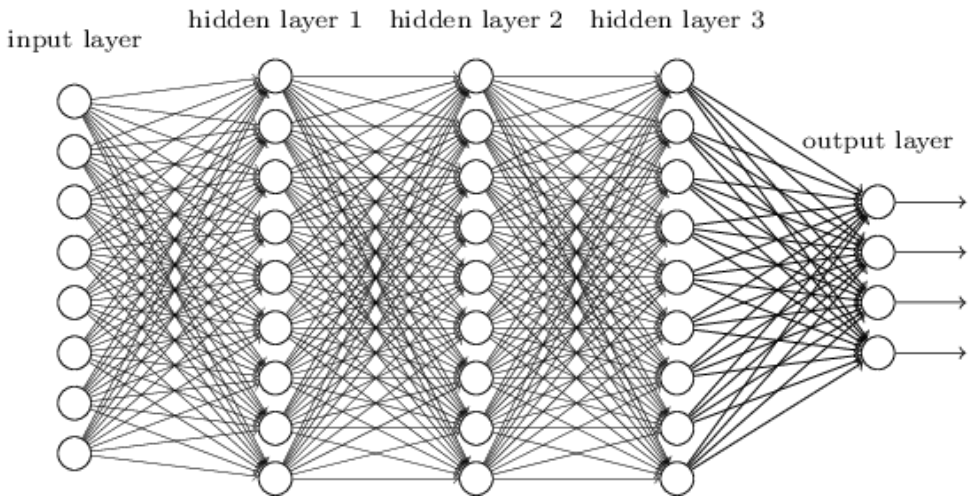
# Deep Neural Network

# Deep Neural Network = many hidden layers (≥3)

Shallow Network




Deep Network



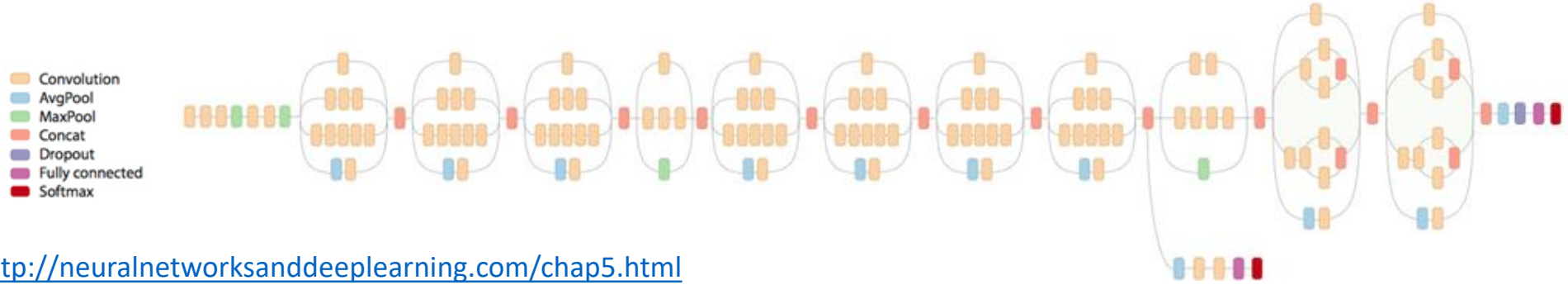
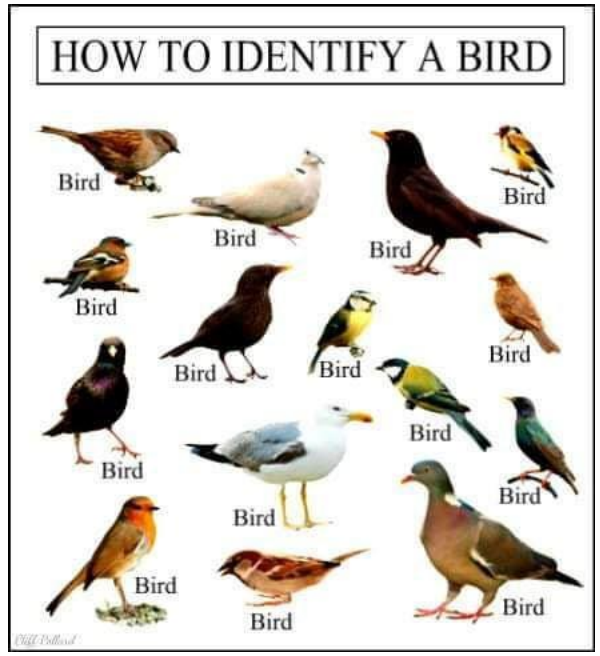



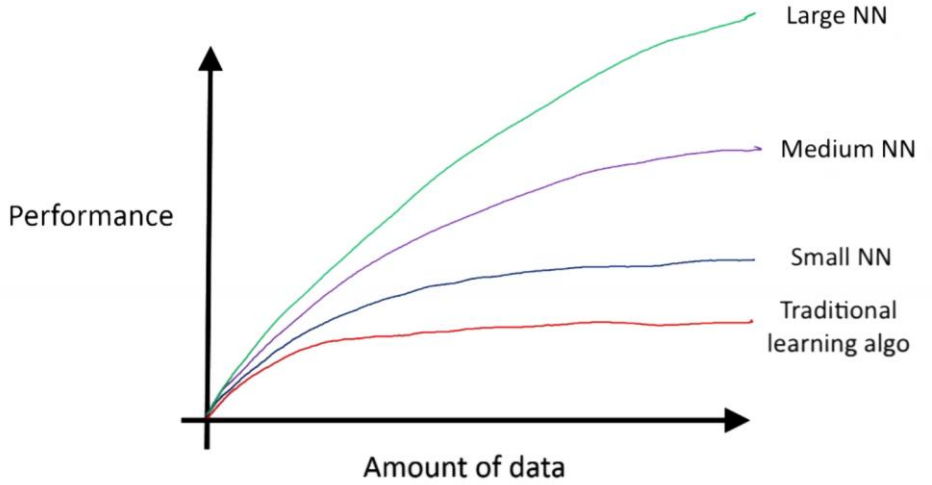

Image credit: http://neuralnetworksanddeeplearning.com/chap5.html
https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html

# Why Deep?

- Why need so many layers?
  - Need **many parameters**
  - Target functions of real-world tasks are **complex**
  - E.g., what is the function for recognizing birds or language?
- Why need so much training data?
  - Many parameters → Need more data
  - More data → Better performance

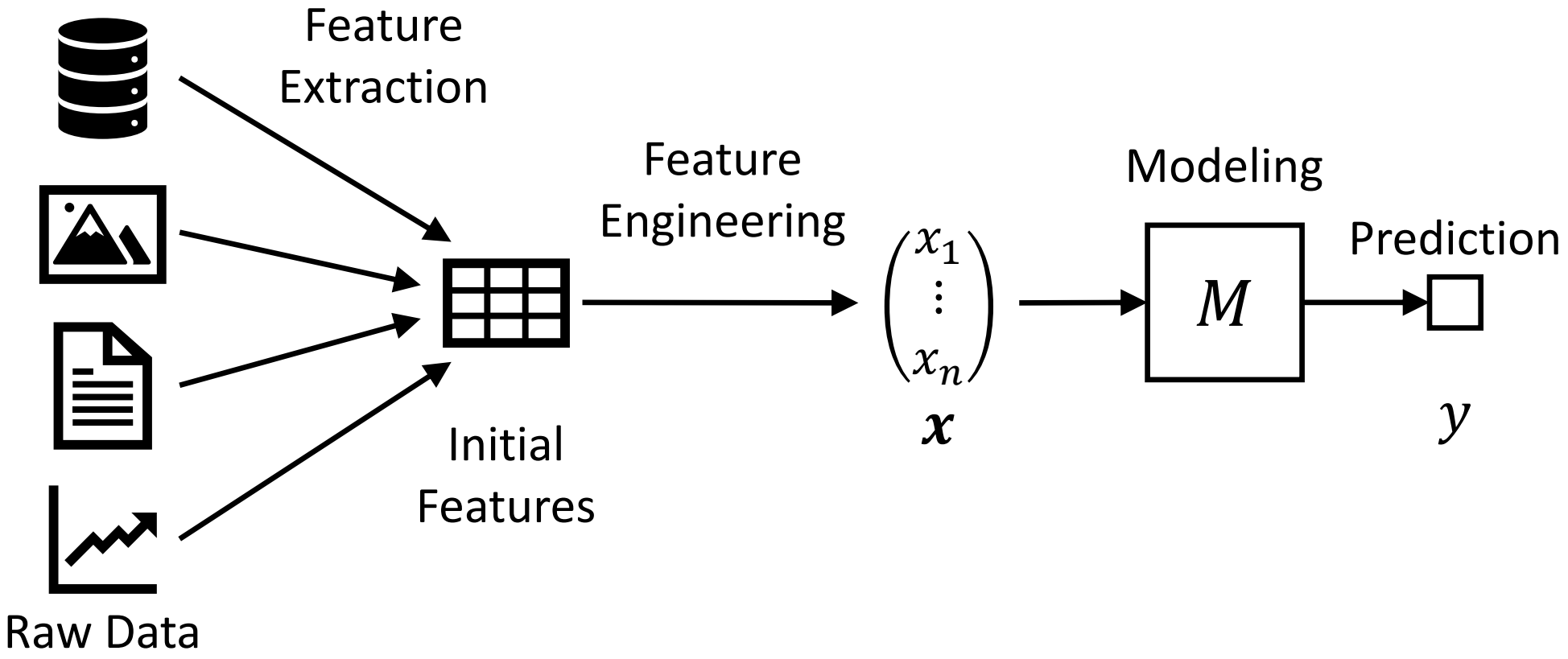



https://9gag.com/gag/ax9Roon

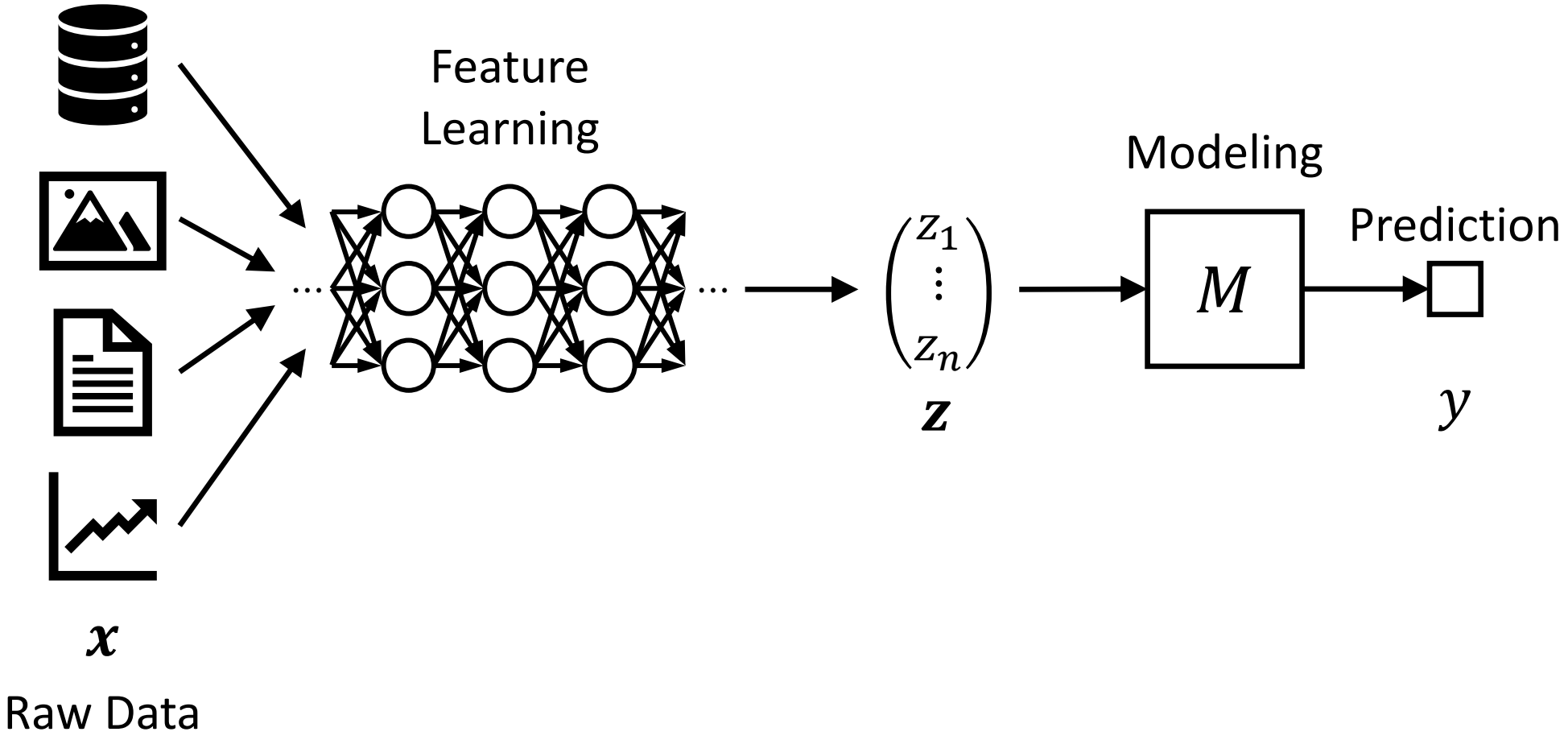



Andrew Ng https://youtu.be/LcfLo7YP8O4

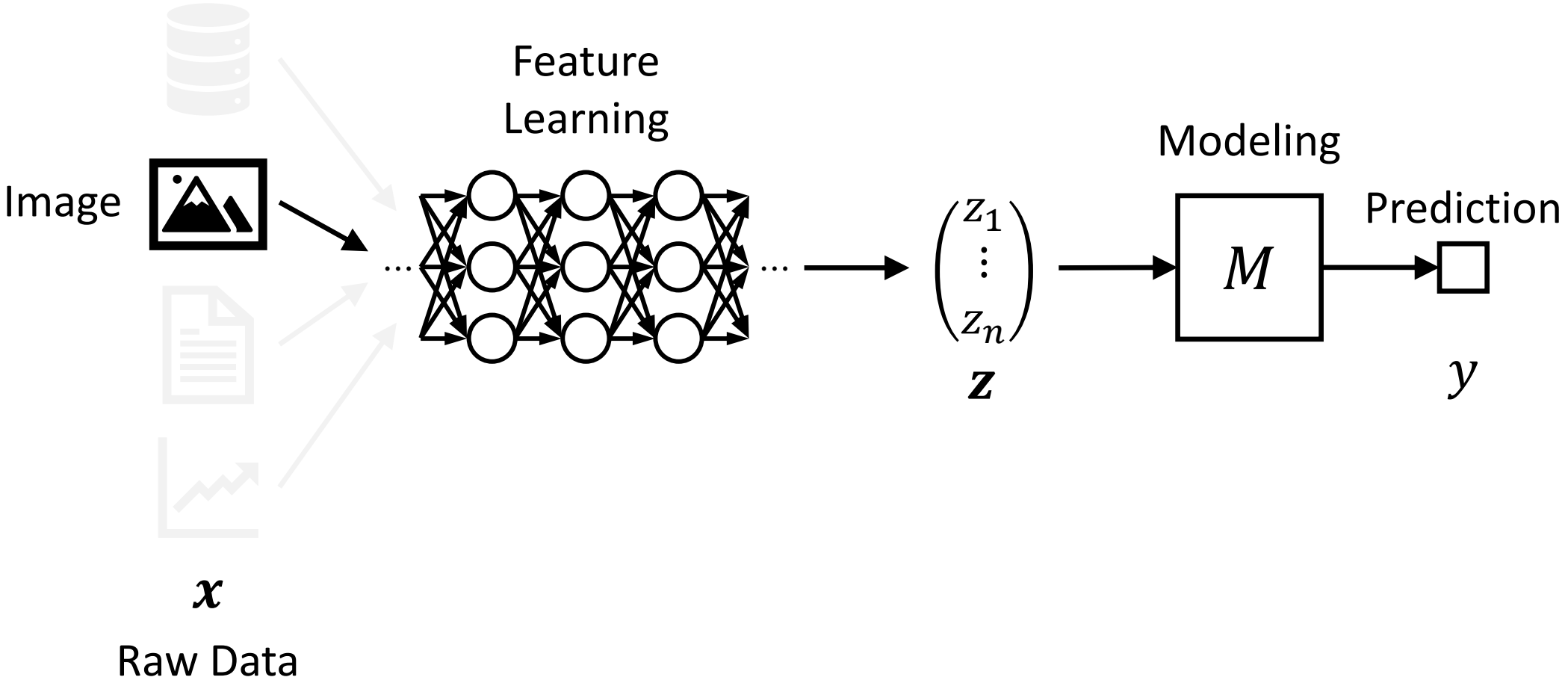
# Feature Extraction/Engineering → Modeling



# From Manual Feature Engineering To Automatic Feature Learning



# From Manual Feature Engineering To Automatic Feature Learning



# Convolutional Neural Networks (CNN)

# Applications of CNN

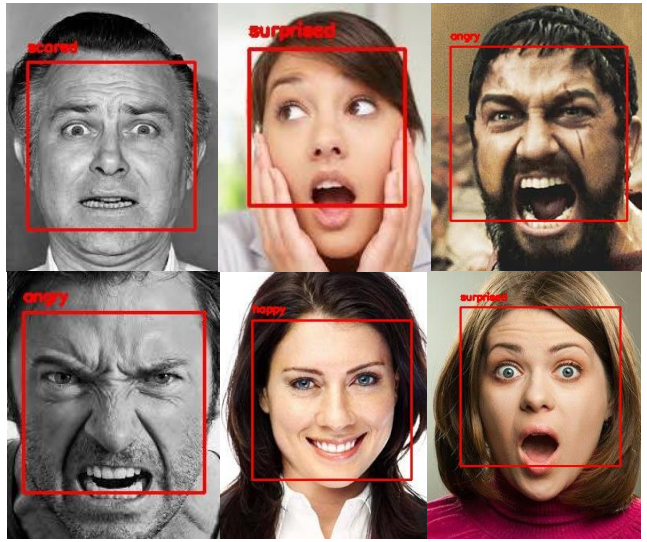
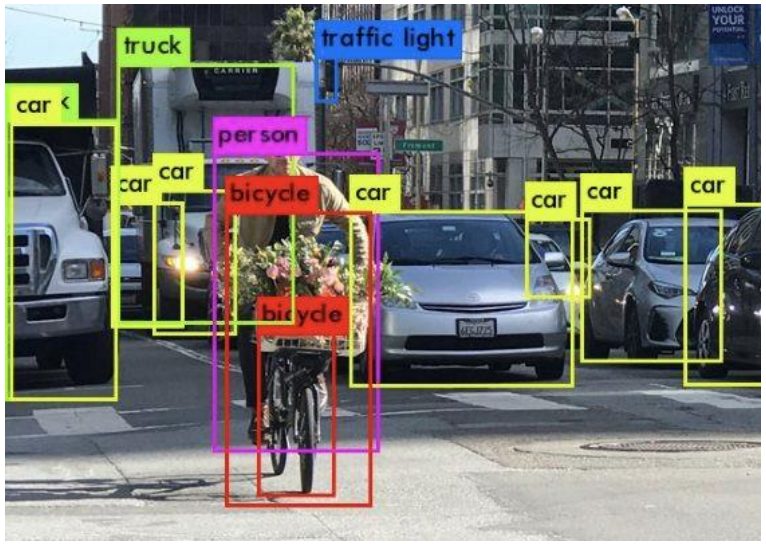



Image Classification
e.g., face emotions




Object Detection
e.g., self-driving cars

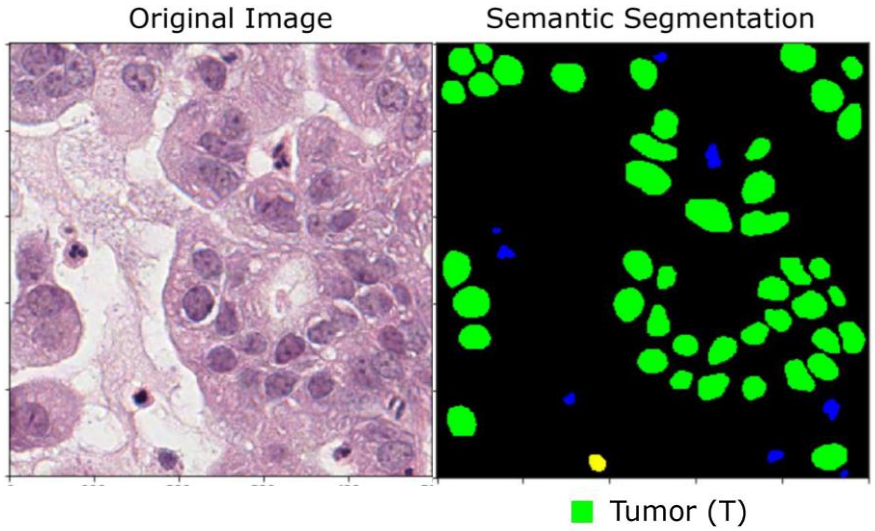



Image Segmentation
e.g., cancer cell detection

Image credit:
https://monica-dommaraju.medium.com/analysis-of-deep-learning-based-object-detection-f14d5138148
https://ajp.amjpathol.org/article/S0002-9440(18)31121-0/fulltext
https://appliedmachinelearning.blog/2018/11/28/demonstration-of-facial-emotion-recognition-on-real-time-video-using-cnn-python-keras/

# FOODAi Demo

https://foodai.org

Try out our demo below or visit our developer portal for our API services.

To try our demo, you can click the upload icon to choose the image, or copy and paste the image or drag and drop the image from desktop or internet to the upload area.

Chicken rice

Boiled kampung chicken

Chicken porridge

Fish porridge

Fried fish porridge

Register for FoodAI API Free Trial

# tastehealthy

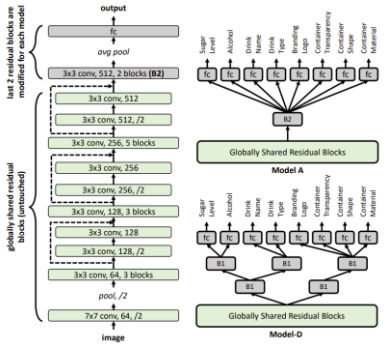
## Backend Models

### Food & Drink Recognition

It is a **dairy** type beverage called **strawberry milkshake** served in a **transparent glass cup without logo**, which is **high in sugar** and **does not have alcohol** in it



It is a **beer** type beverage called **ale** served in a **semi-transparent glass bottle with logo,** which has **no sugar** but **has alcohol** in it




### Food Recommendation

#### RecGAN

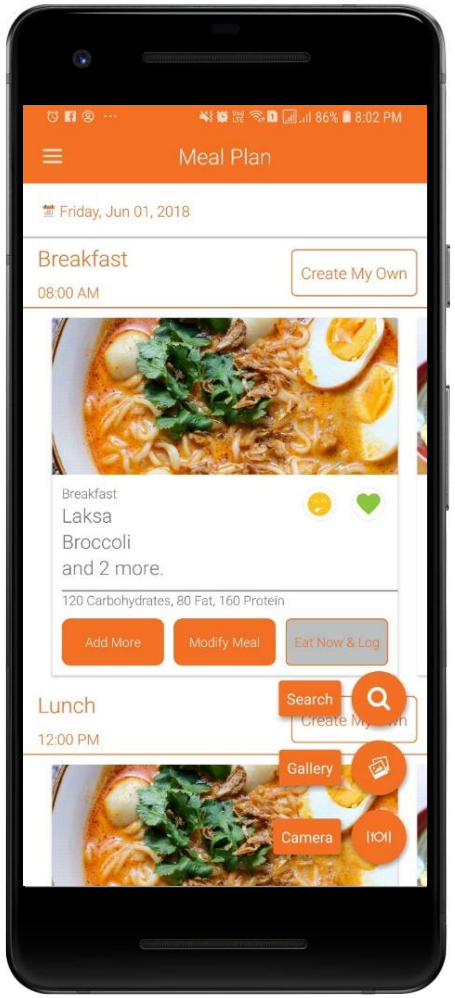
Generator

$$\mathbf{x}_t^u = RELU(\mathbf{W}_{xh}^u \mathbf{h}_{t-1}^u + \mathbf{W}_{xk}^u \mathbf{y}_t^u)$$
$$\mathbf{r}_t^u = \sigma(\mathbf{W}_{rh}^u \mathbf{h}_{t-1}^u + \mathbf{W}_{rk}^u \mathbf{y}_t^u)$$
$$\mathbf{m}_t^u = tanh(\mathbf{W}_h(\mathbf{r}_t^u \cdot \mathbf{h}_{t-1}^u) + \mathbf{W}_{input} \mathbf{y}_t^u)$$
$$\mathbf{h}_t^u = (1 - \mathbf{x}_t^u) \cdot \mathbf{h}_{t-1}^u + \mathbf{x}_t^u \cdot \mathbf{m}_t^u$$

Discriminator

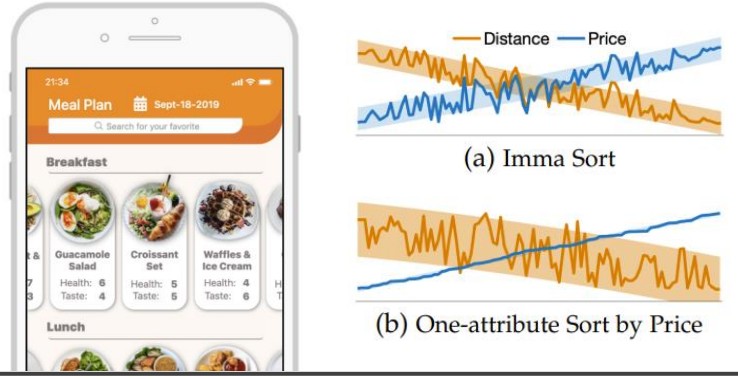
$$\hat{\mathbf{x}}_t^u = RELU(\mathbf{V}_{xh}^u \hat{\mathbf{h}}_{t-1}^u + \mathbf{V}_{xk}^u \mathbf{y}_t^u)$$
$$\hat{\mathbf{r}}_t^u = \sigma(\mathbf{V}_{rh}^u \hat{\mathbf{h}}_{t-1}^u + \mathbf{V}_{rk}^u \mathbf{y}_t^u)$$
$$\hat{\mathbf{m}}_t^u = tanh(\mathbf{V}_h(\hat{\mathbf{r}}_t^u \cdot \hat{\mathbf{h}}_{t-1}^u) + \mathbf{V}_{input} \mathbf{y}_t^u)$$
$$\hat{\mathbf{h}}_t^u = (1 - \hat{\mathbf{x}}_t^u) \cdot \hat{\mathbf{h}}_{t-1}^u + \hat{\mathbf{x}}_t^u \cdot \hat{\mathbf{m}}_t^u$$

## Frontend UI/UX




### Family Food Logging


### Multi-Attribute Sorting




Lyu, Y., Gao, F., Wu, I. S., & Lim, B. Y. 2020. Imma Sort by Two or More Attributes With Interpretable Monotonic Multi-Attribute Sorting. TVCG.
Park, H., Bharadhwaj, H., and Lim, B. Y. 2019. Hierarchical Multi-Task Learning for Healthy Drink Classification. *IJCNN.*
Bharadhwaj, H., Park, H., Lim, B. Y.. 2018. RecGAN: Recurrent Generative Adversarial Networks for Recommendation Systems. *RecSys '18.*
Lukoff, K., Li, T., Zhuang, Y., & Lim, B. Y. 2018. TableChat: Mobile Food Journaling to Facilitate Family Support for Healthy Eating. *CSCW '18.*

NUS Ubicomp Lab
Apps and Analytics for Smart Cities and Health

# Images as 2D matrices

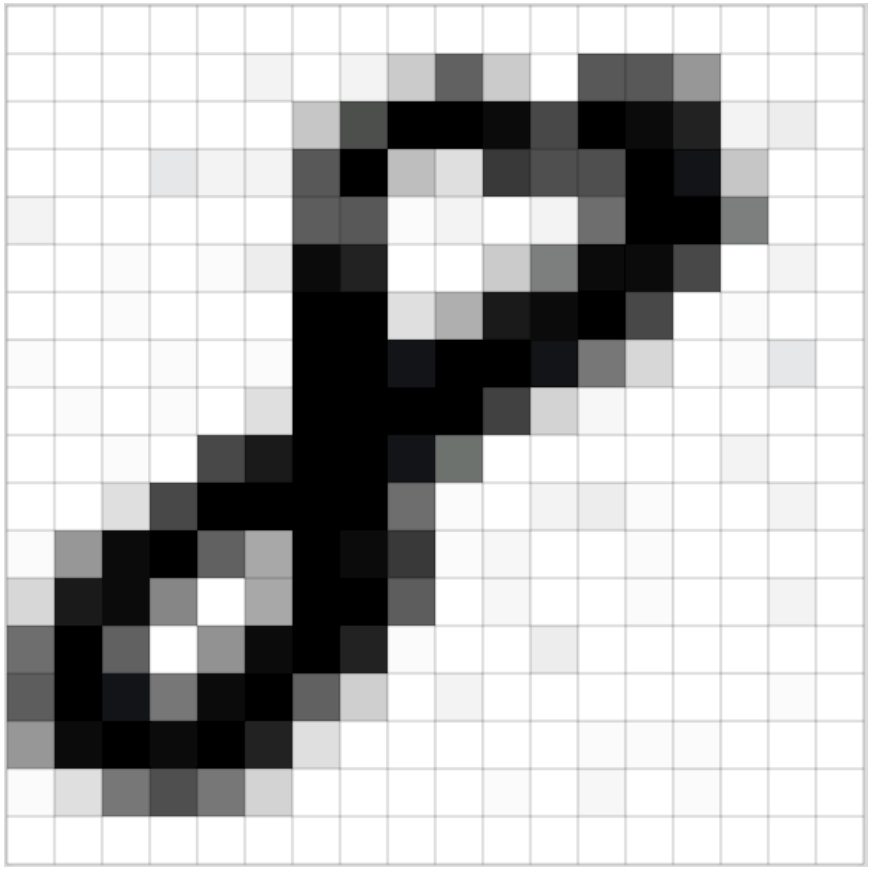
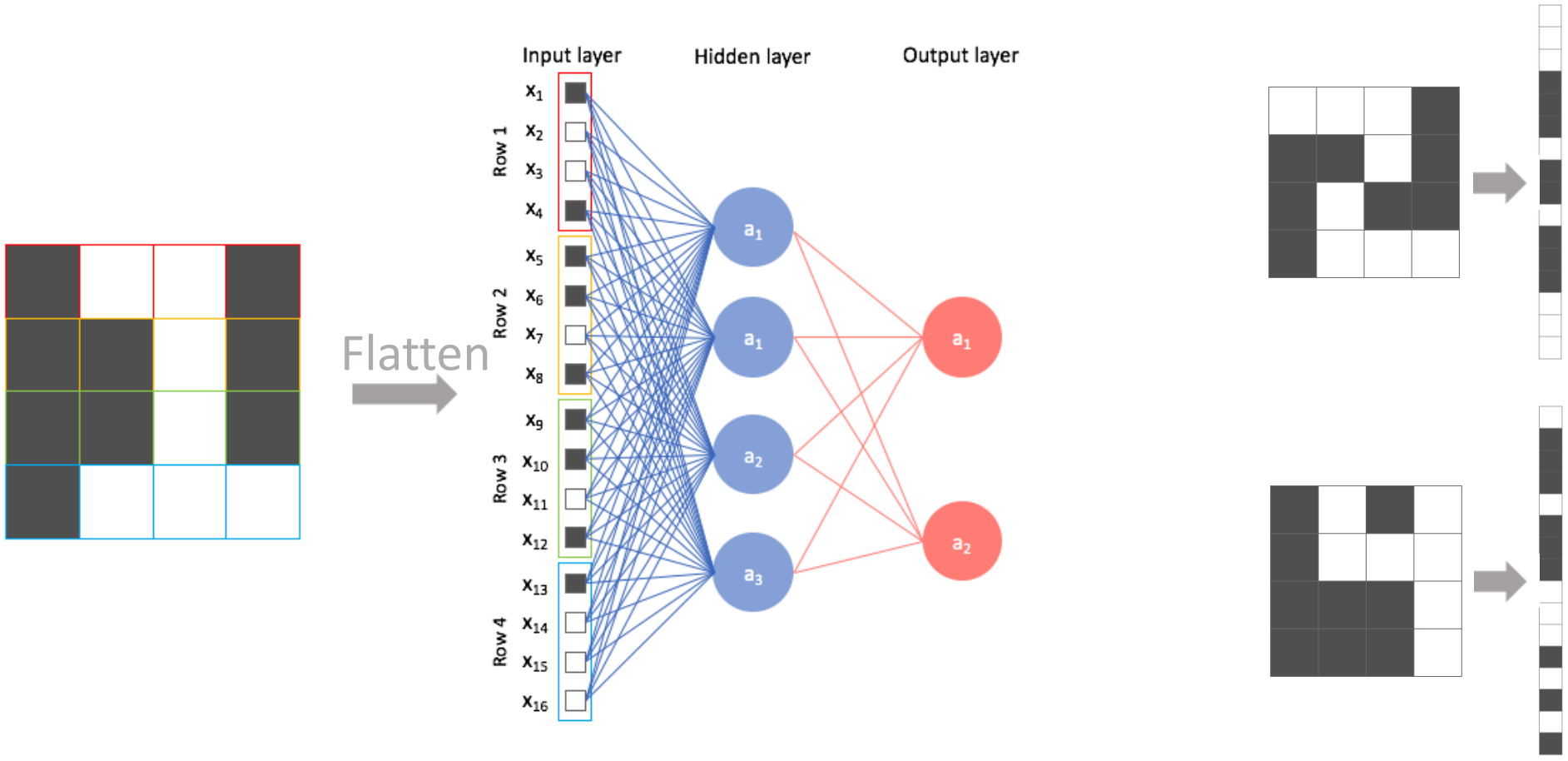
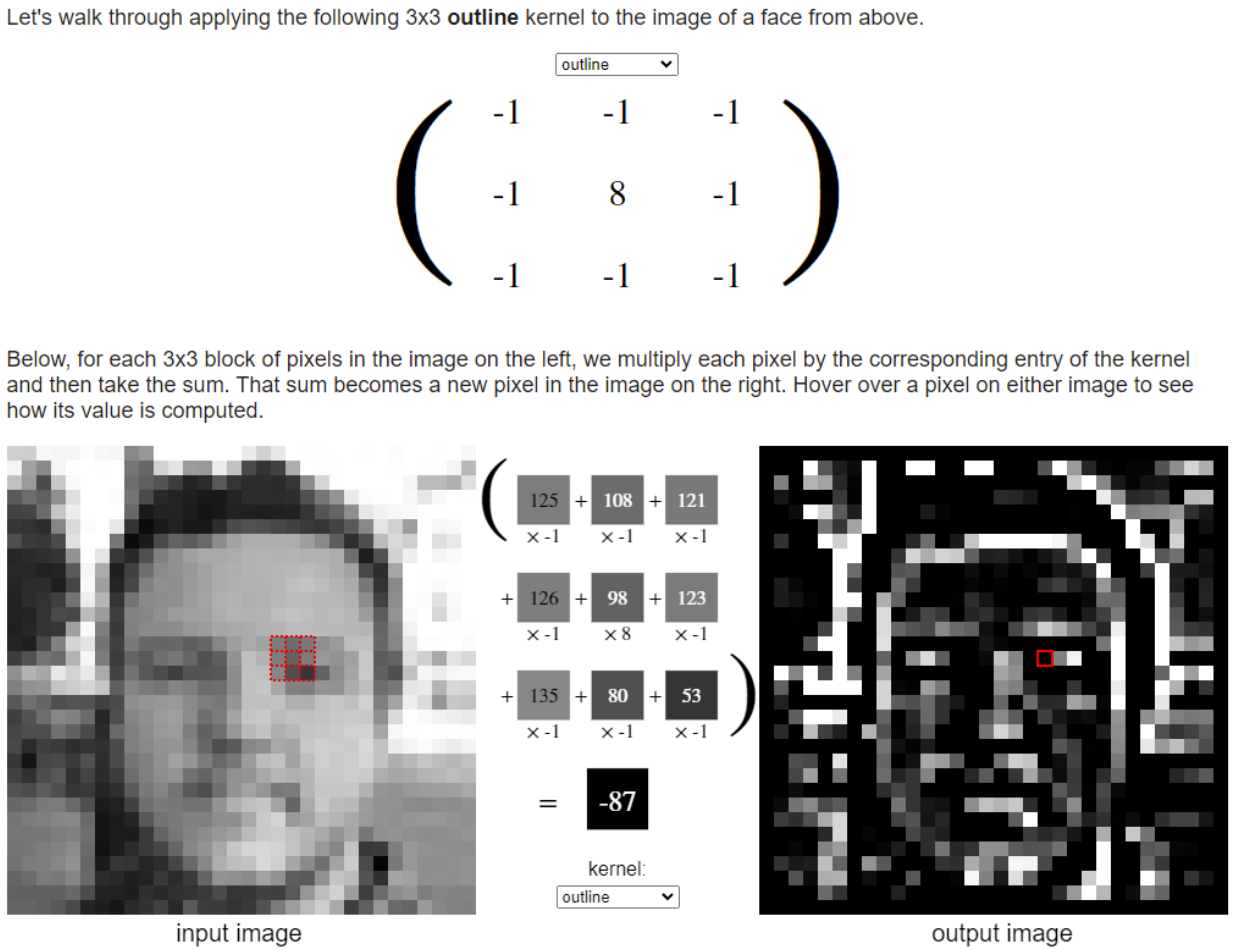


Image credit

# Image Feature Extraction with
## Fully Connected Neural Networks (Multi-Layer Perceptron)



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




**Manually** finding good filters is tedious

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

# High-level Feature Detection

Eyes, Nose, Mouth
Facial Hair

Wheels, Headlights,
Bonnet/Hood

Fish, Rice,
Vegetables

How to **automatically** learn these features?

# Feature Detectors: Intuition of Neuron Kernels in Layers

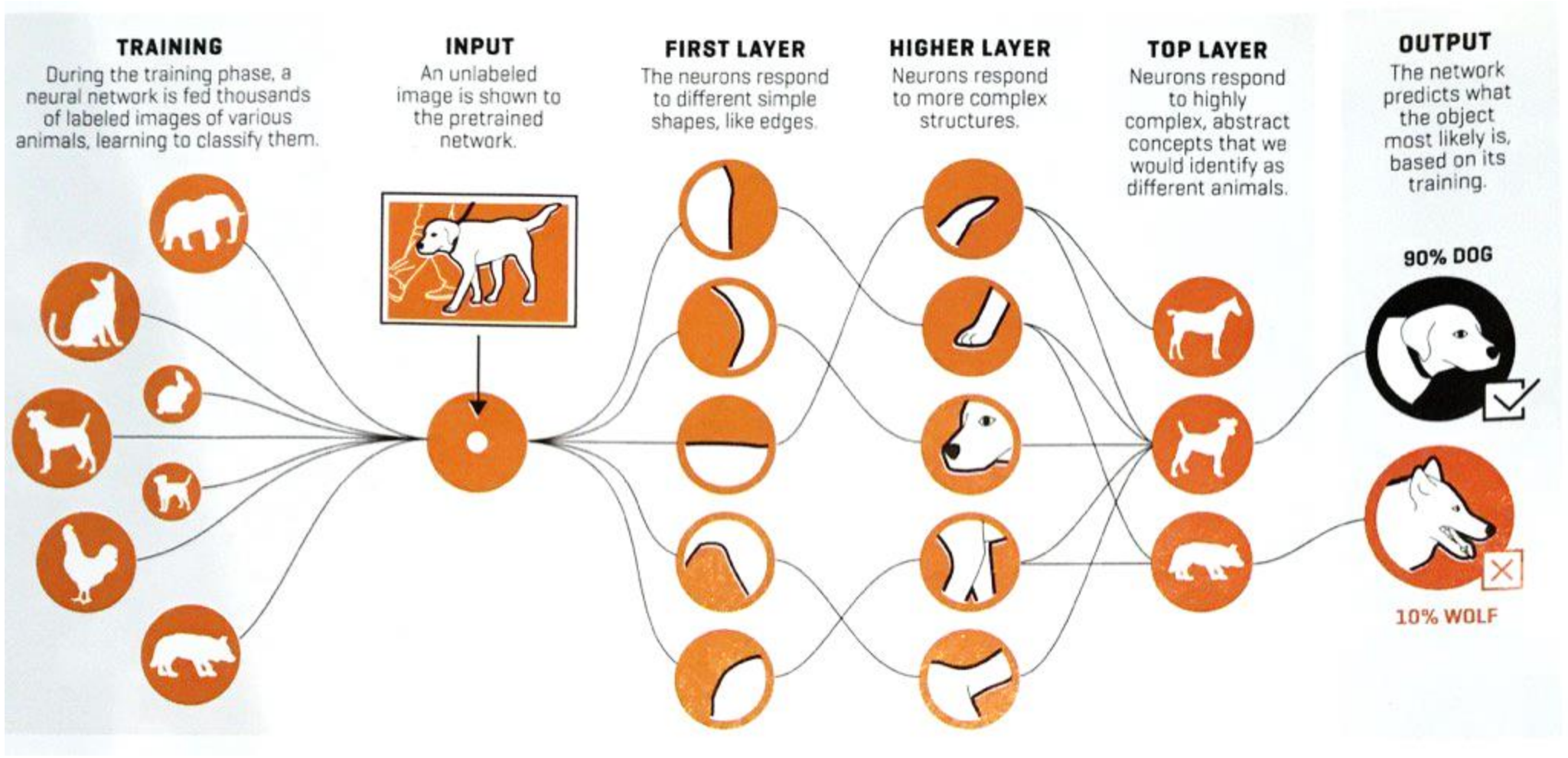



Image credit: https://fortune.com/longform/ai-artificial-intelligence-deep-machine-learning/

# **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/

# Convolutions: Kernel Size, Stride, Padding

$$\boldsymbol{W} = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \quad \boldsymbol{x} = \begin{pmatrix} 9 & 9 & 3 & 3 & 4 \\ 9 & 3 & 3 & 4 & 5 \\ 9 & 3 & 3 & 5 & 5 \\ 9 & 3 & 3 & 4 & 5 \\ 9 & 9 & 3 & 3 & 4 \end{pmatrix} \quad \boldsymbol{W} * \boldsymbol{x} = \begin{pmatrix} -6-6-6 & -6+1+2 & 1+2+2 \\ -6-6-6 & 1+2+1 & 2+2+2 \\ -6-6-6 & 2+1-6 & 2+2+1 \end{pmatrix}$$

Stride $s \neq 1$

$$\boldsymbol{W} = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \quad \boldsymbol{x} = \begin{pmatrix} 9 & 9 & 3 & 3 & 4 \\ 9 & 3 & 3 & 4 & 5 \\ 9 & 3 & 3 & 5 & 5 \\ 9 & 3 & 3 & 4 & 5 \\ 9 & 9 & 3 & 3 & 4 \end{pmatrix} \quad \boldsymbol{W} * \boldsymbol{x} = \begin{pmatrix} -6-6-6 & 1+2+2 \\ -6-6-6 & 2+2+1 \end{pmatrix}$$

Padding $p \neq 0$

$$\boldsymbol{W} = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \quad \boldsymbol{x} = \begin{pmatrix} & & & & \\ & 3 & 3 & 4 & \\ & 3 & 3 & 5 & \\ & 3 & 3 & 4 & \\ & & & & \end{pmatrix} \quad \boldsymbol{W} * \boldsymbol{x} = \begin{pmatrix} -6-6-6 & 1+2+2 \\ -6-6-6 & 2+2+1 \end{pmatrix}$$

# Exercise E10a.1

# Exercise E10a.1 Solution

# Convolutional Layer

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

Convolution of each **Feature Kernel** $\boldsymbol{W}^{[l]}$ produces a **Feature Map** $= \boldsymbol{W}^{[l]} * \boldsymbol{A}^{[l-1]}$

Weights = **Kernels** (3D matrix)

**2D Convolution** (height × width)

$$\boldsymbol{a}^{[l]} = g^{[l]}\left(\left(\boldsymbol{W}^{[l]}\right)^{\top}\boldsymbol{a}^{[l-1]}\right) \quad \text{vs.} \quad \boldsymbol{A}^{[l]} = g^{[l]}\left(\boldsymbol{W}^{[l]} * \boldsymbol{A}^{[l-1]}\right)$$

**Activation function** can be the same

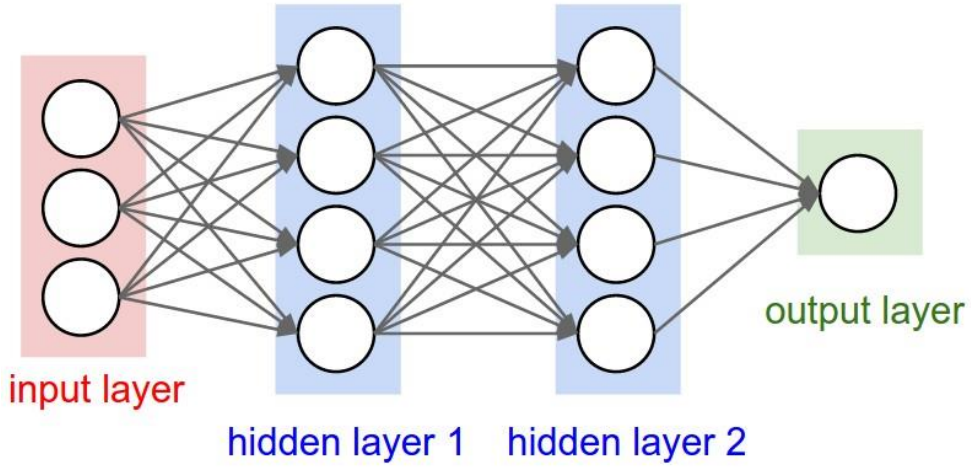
**Activation** is a 3D Matrix!
Concatenation of Feature Maps

# Convolutional Layers

**Remember:** each kernel is like a different lens filter


## Fully Connected Layers

- Each layer has multiple **neurons** ◯
- Neuron output: **0D scalar** activation
- Neuron input: **1D vector** of activations
  - Each *element* is a different neuron
- Each layer is a **1D vector**




## Convolutional Layers

- Each layer has multiple **kernels** ▦
- Kernel output: **2D matrix** feature map
- Kernel input: **3D matrix** of feature maps
  - Each *depth position* is a different kernel
  - Analogy: filters are "stacked" together
- Each layer is a **3D matrix**

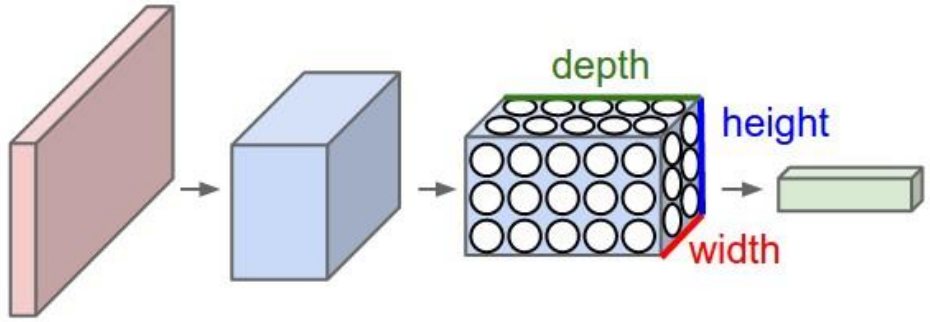
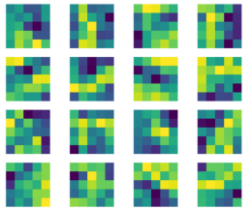



Image credit: https://cs231n.github.io/convolutional-networks/

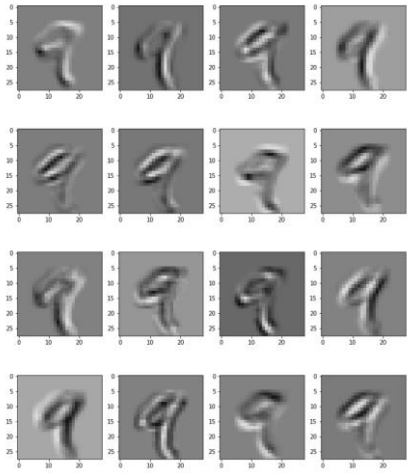
# Convolutional Layer:
# Feature Kernels & Feature Maps

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

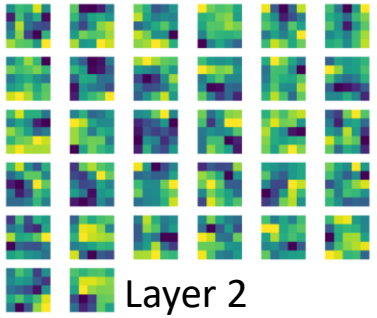
Input



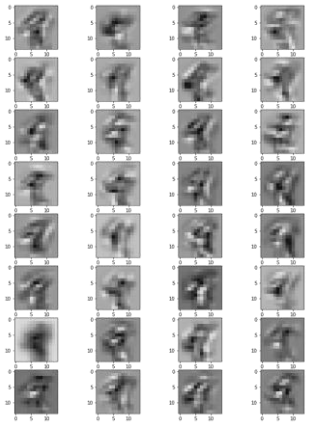
Layer 1 Feature Kernels



Layer 1
Feature Maps



Layer 2
Feature Kernels



Layer 2
Feature 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**.

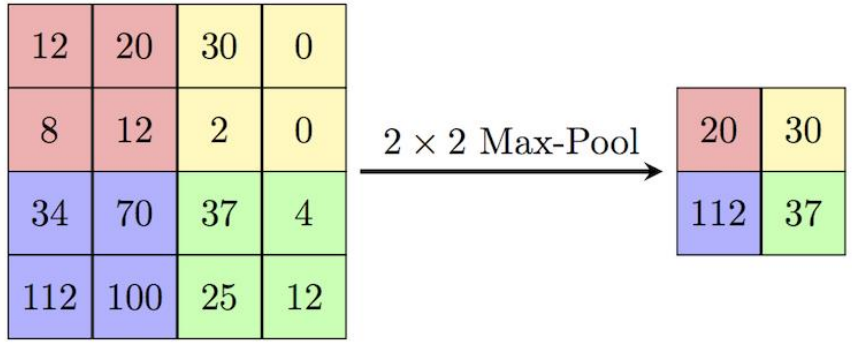
Interpretability: do you know what these kernel mean?

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

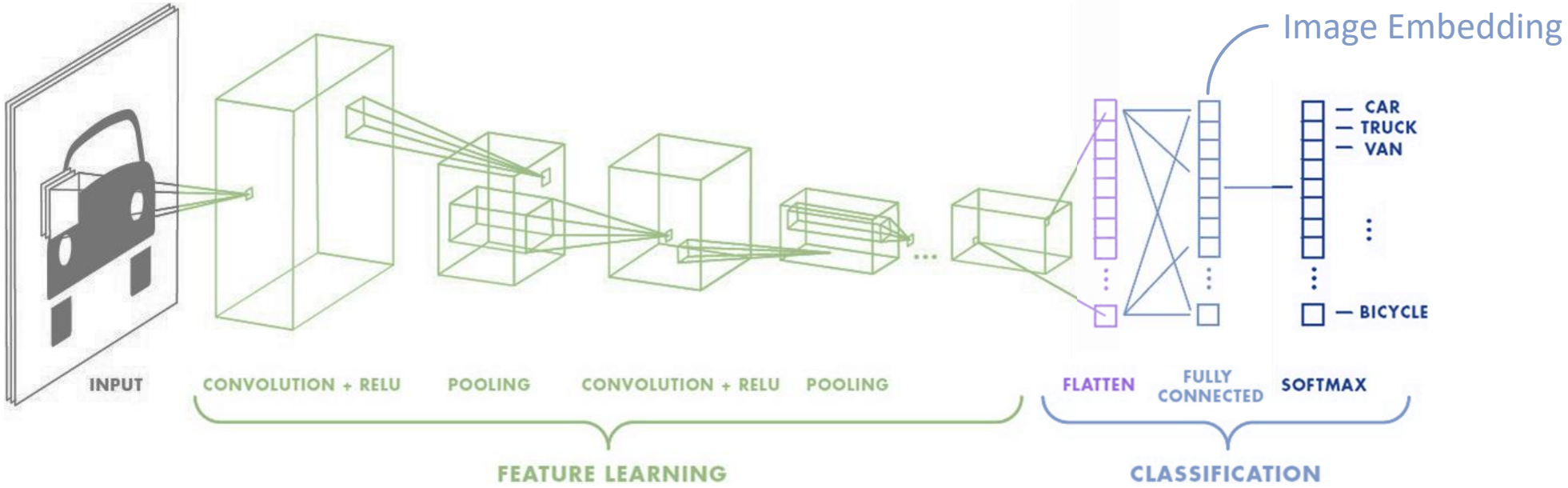
Calculation




Example


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

# Convolutional Neural Network




**Key concepts**

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

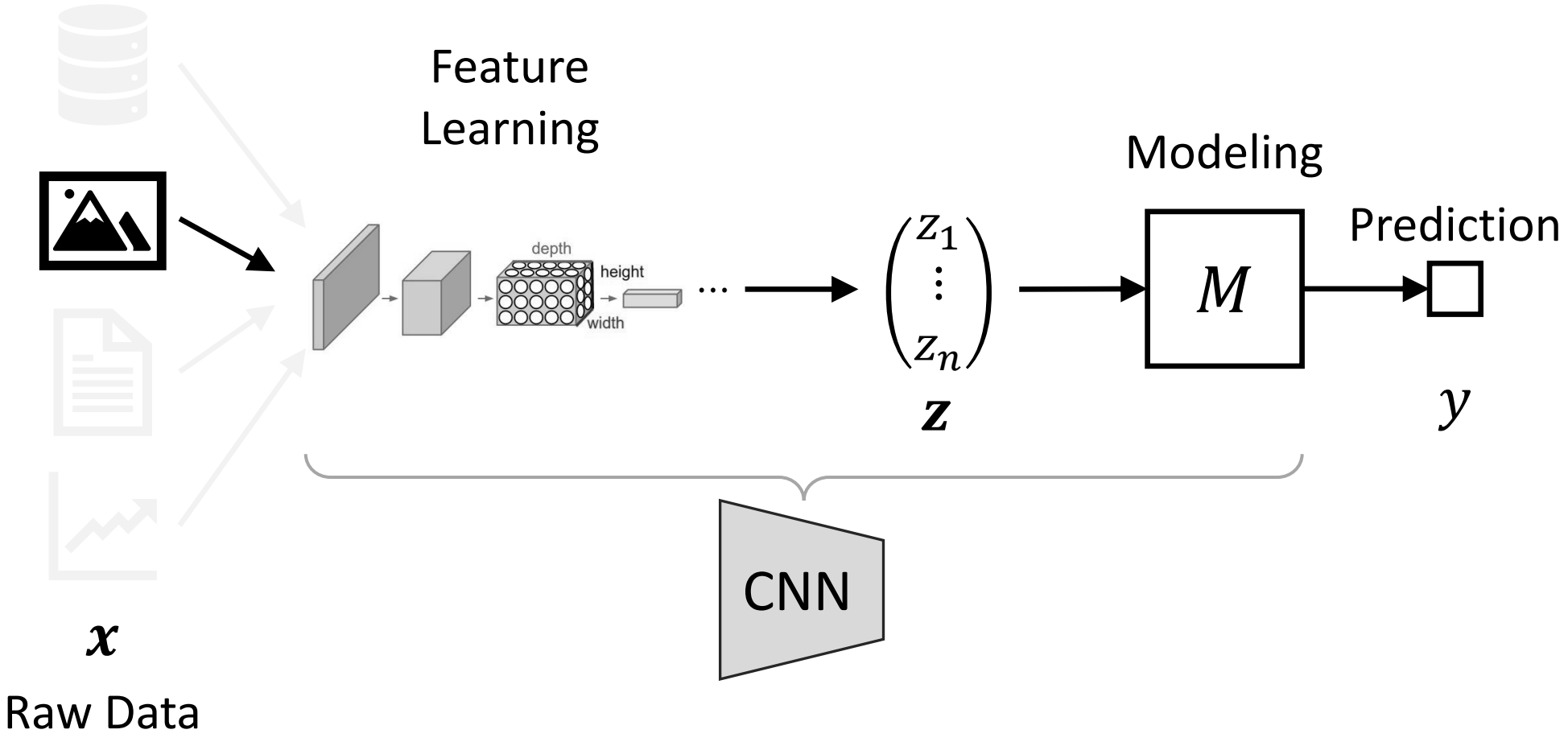
**❷ Flattening**
- Convert to fixed-length 1D vector

**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)

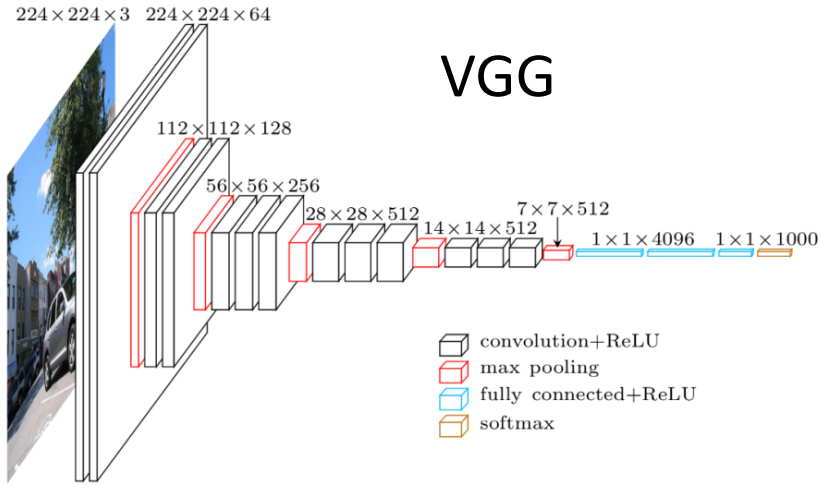
Image credit: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

# From Manual Feature Engineering To Automatic Feature Learning

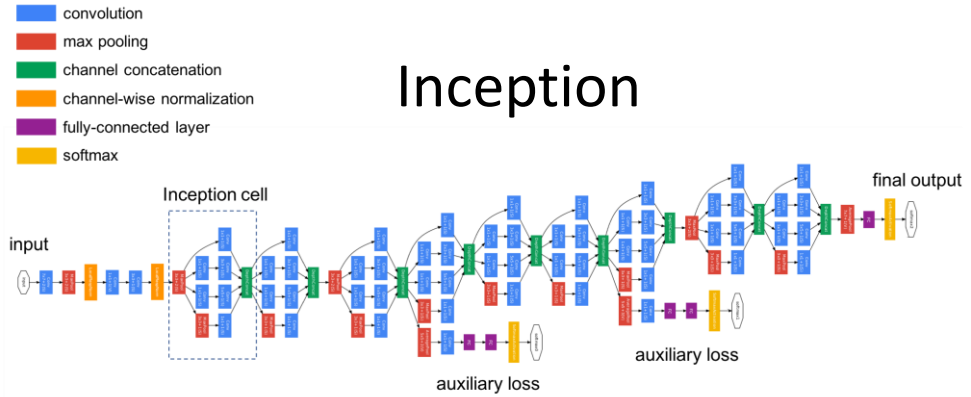


# Other popular CNN architectures

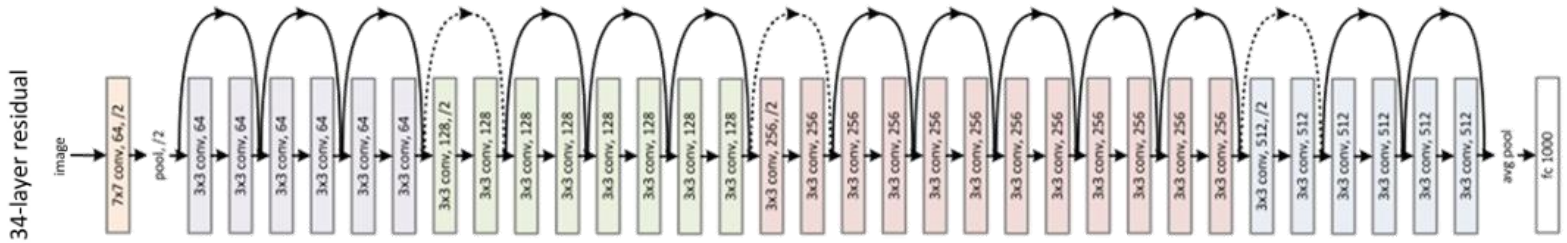
VGG



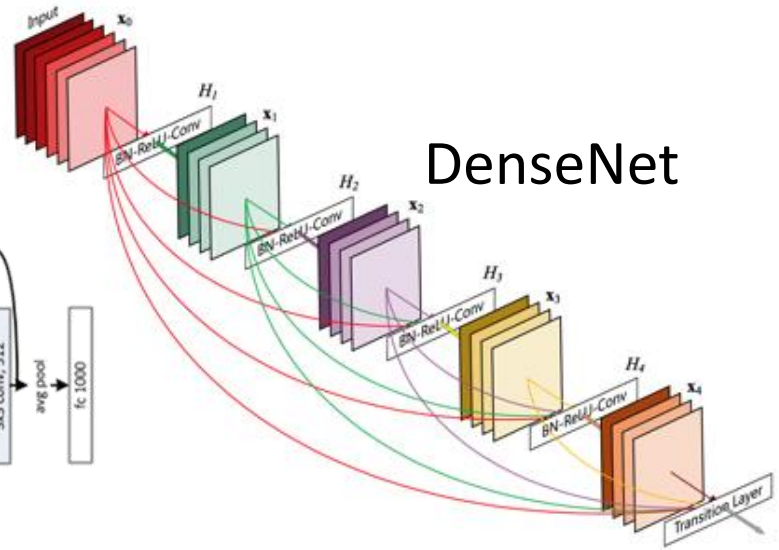

Inception



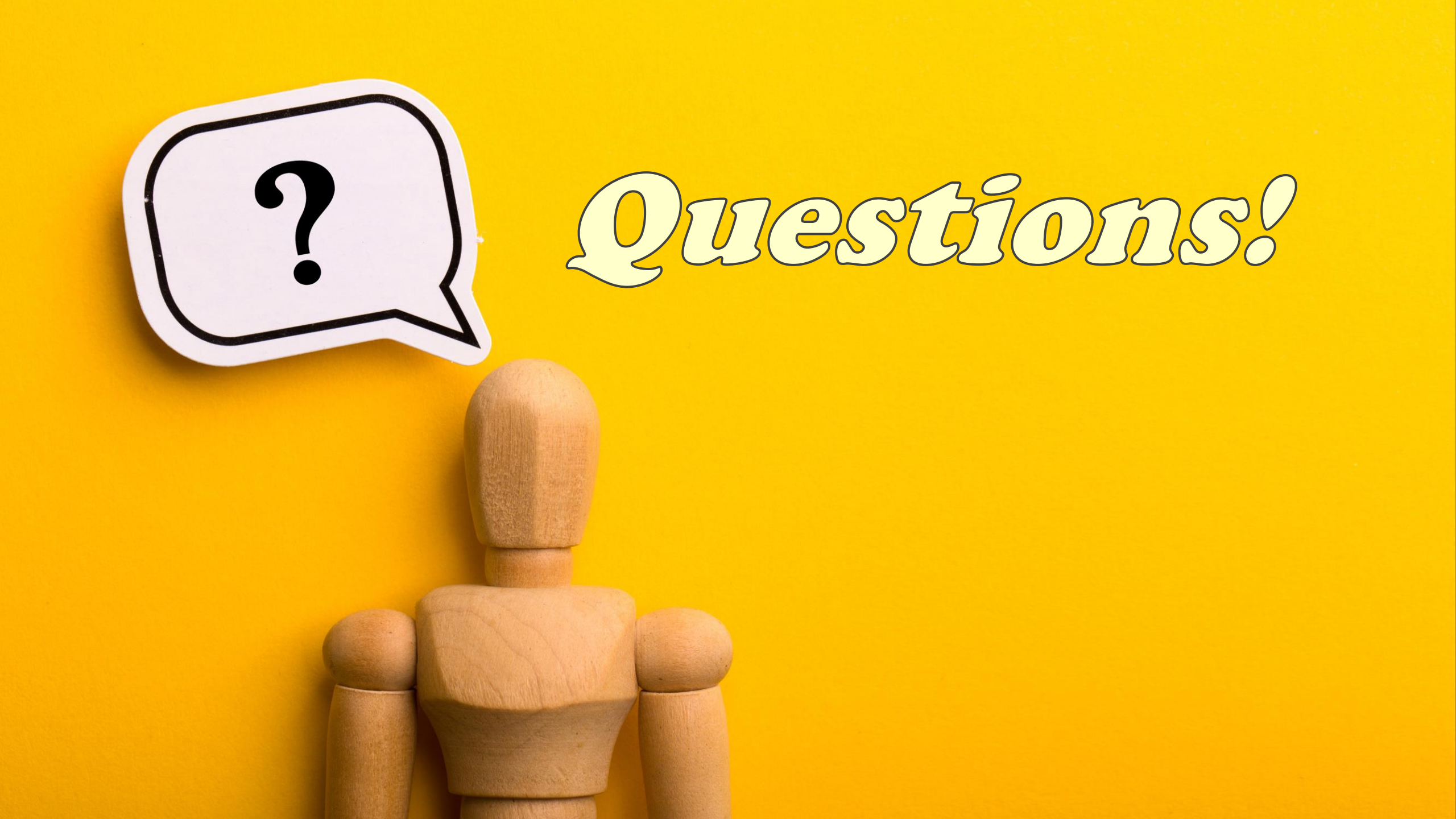

ResNet



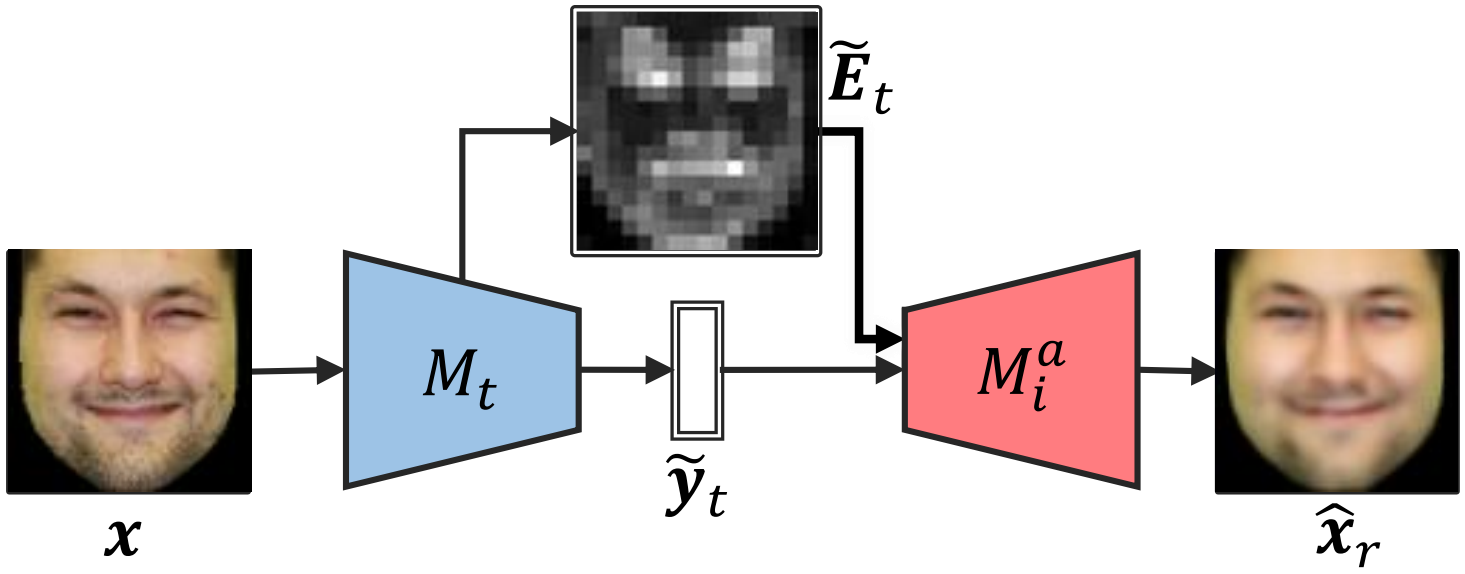

DenseNet




Further reading: https://www.jeremyjordan.me/convnet-architectures/


?
Questions!

# Model Inversion Attack:
# Predicting Images from Classification Vector

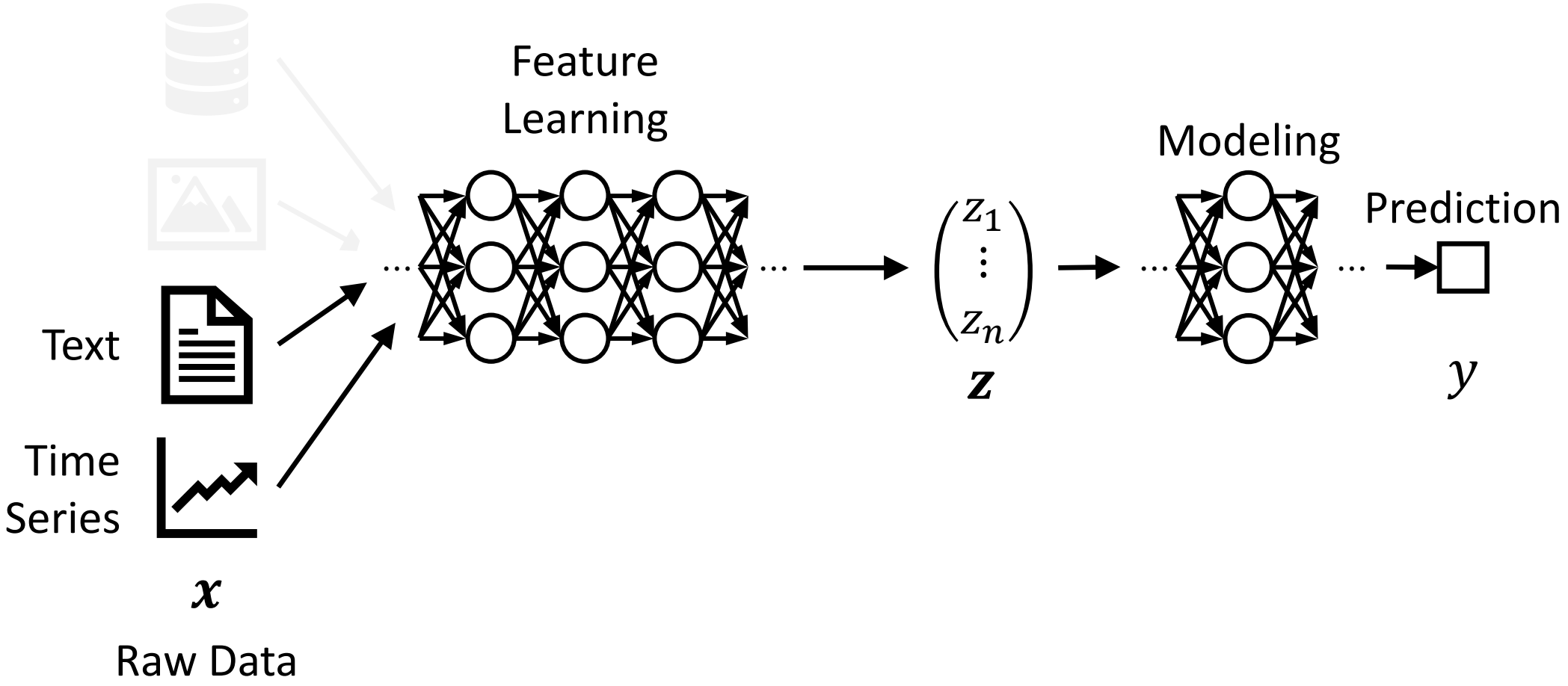



Model inversion attacks can reconstruct private face photos from prediction vectors only.

Model explanations can **worsen** model inversion attacks.

Zhao, X., Zhang, W., Xiao, X., & Lim, B. Y. (2021). Exploiting Explanations for Model Inversion Attacks. *ICCV 2021*.

# From Manual Feature Engineering To Automatic Feature Learning



# Recurrent Neural Networks (RNN)

# Applications of RNN

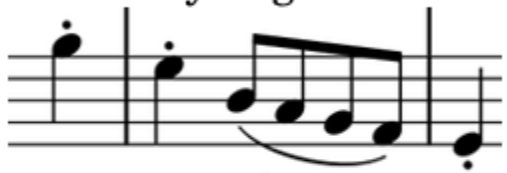
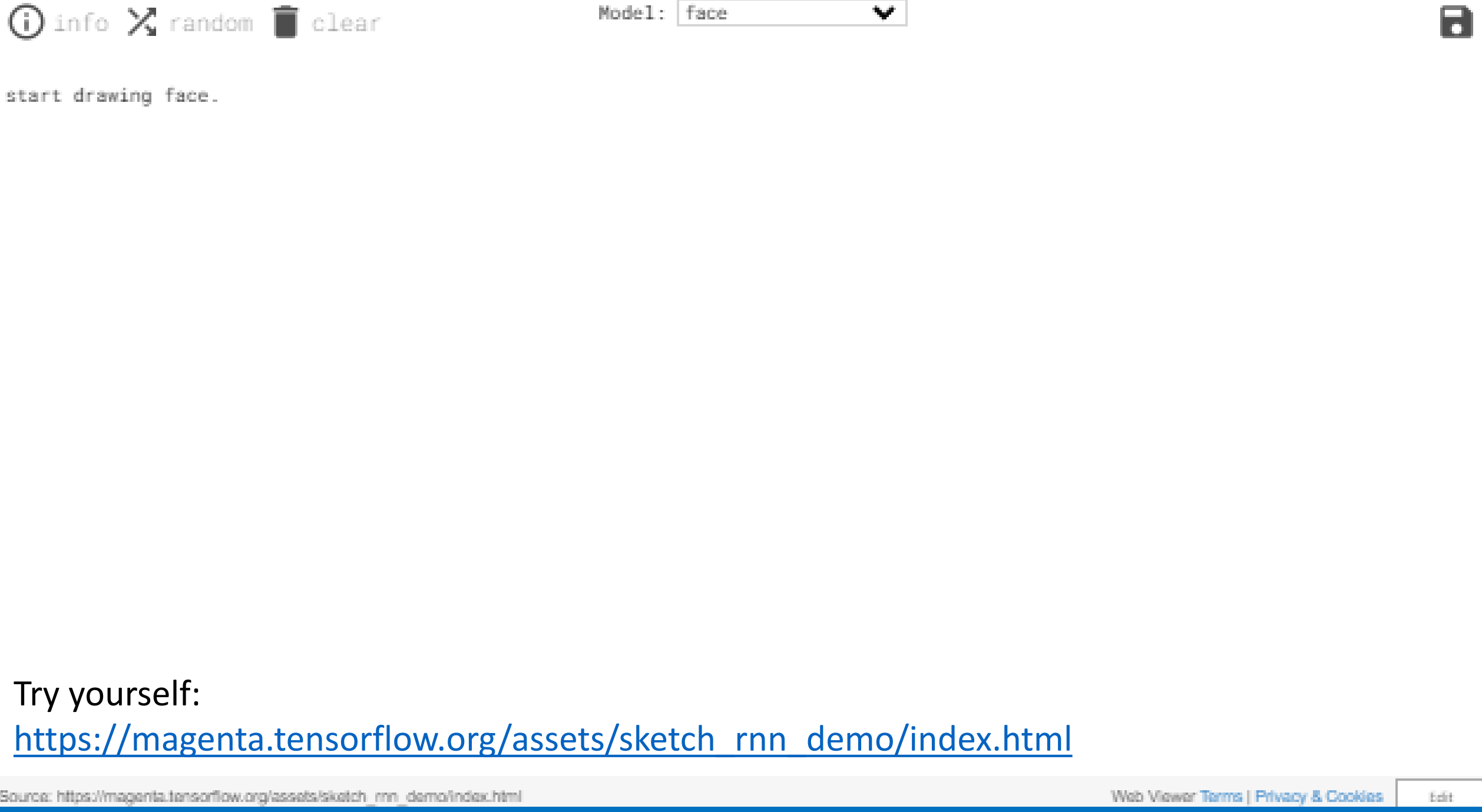
| | | | |
|---|---|---|---|
| Speech recognition |  | → | "The quick brown fox jumped over the lazy dog." |
| Music generation | Ø | → |  |
| Sentiment classification | "There is nothing to like in this movie." | → | ★☆☆☆☆ |
| DNA sequence analysis | AGCCCCTGTGAGGAACTAG | → | AGCCCCTGTGAGGAACTAG |
| Machine translation | Voulez-vous chanter avec moi? | → | Do you want to sing with me? |

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

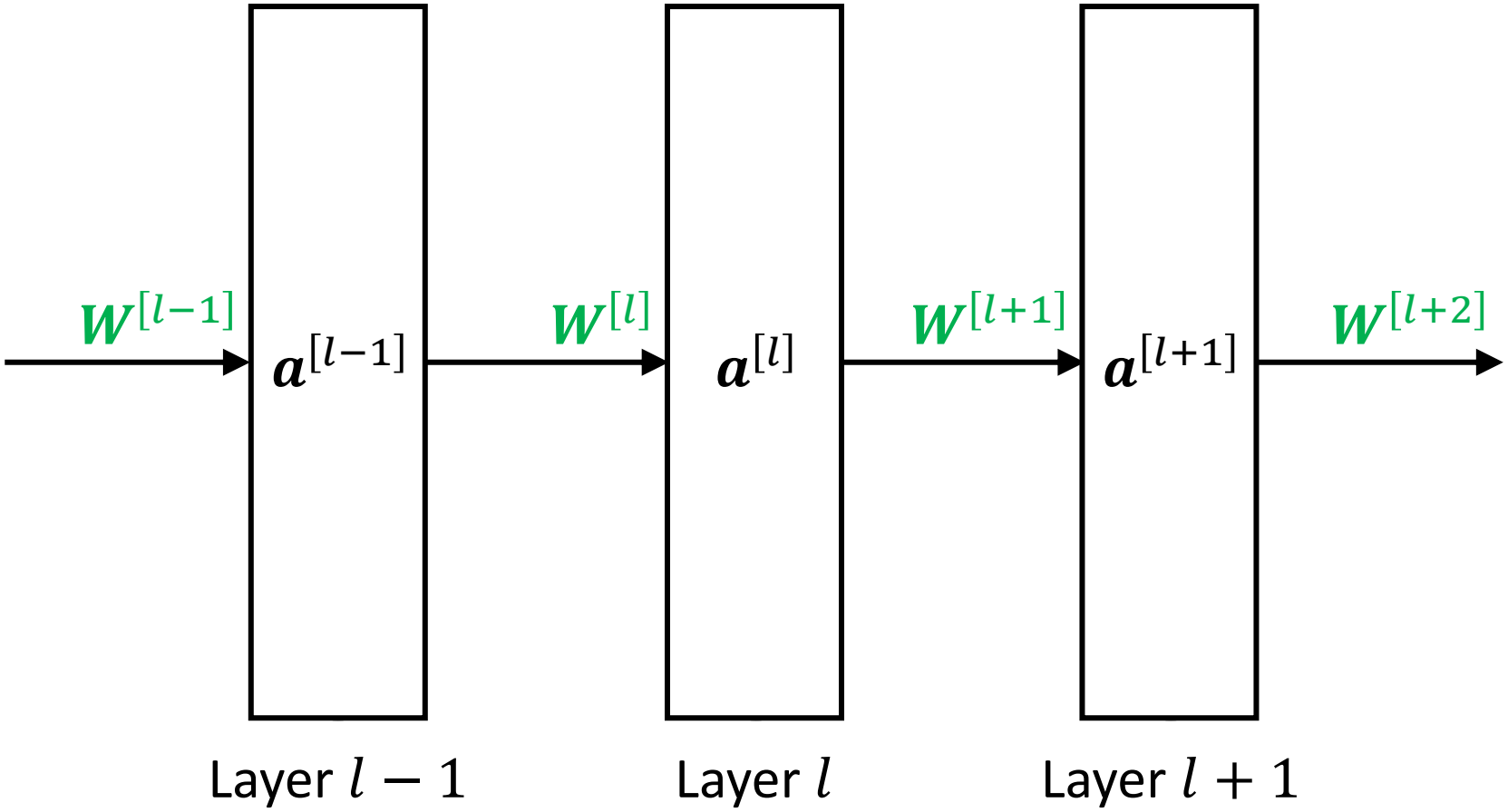



Try yourself:
https://magenta.tensorflow.org/assets/sketch_rnn_demo/index.html

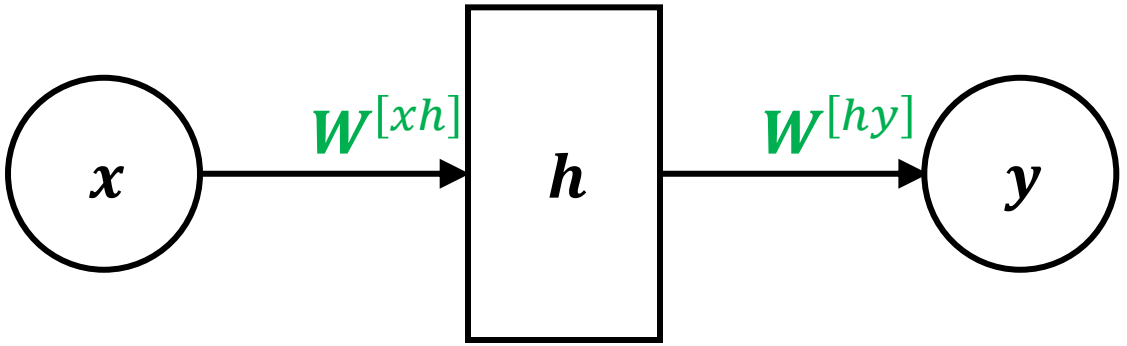
# Play

- Visit: https://magenta.tensorflow.org/assets/sketch_rnn_demo/index.html
- 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

# Feedforward Neural Network

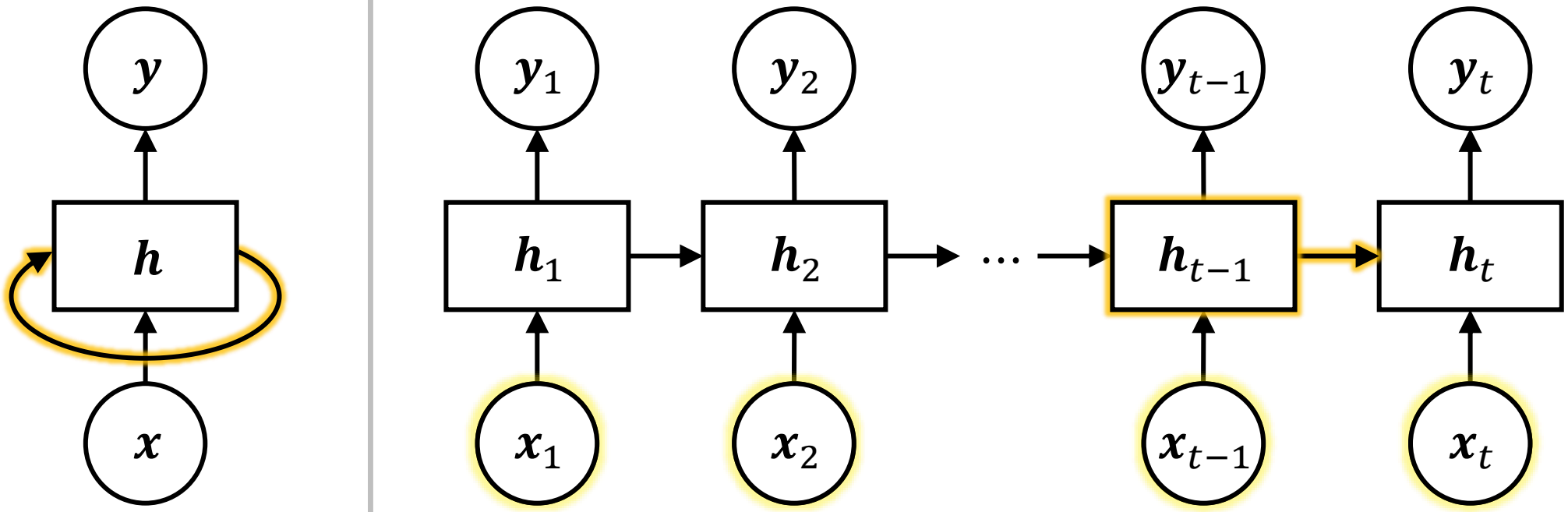


# Neural Network (simplified 1-hidden layer)




Input Layer Hidden Layer Output Layer

# Neurons with Recurrence




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

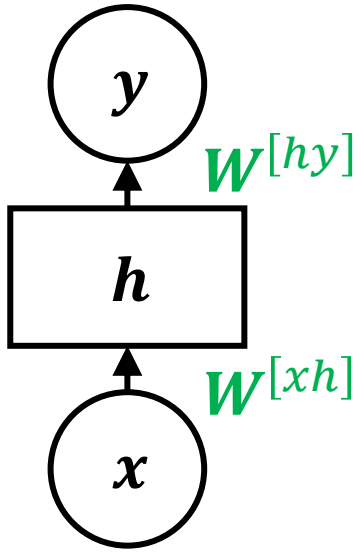
**Recurrent** Neural Network (RNN)

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

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

# RNN Weights

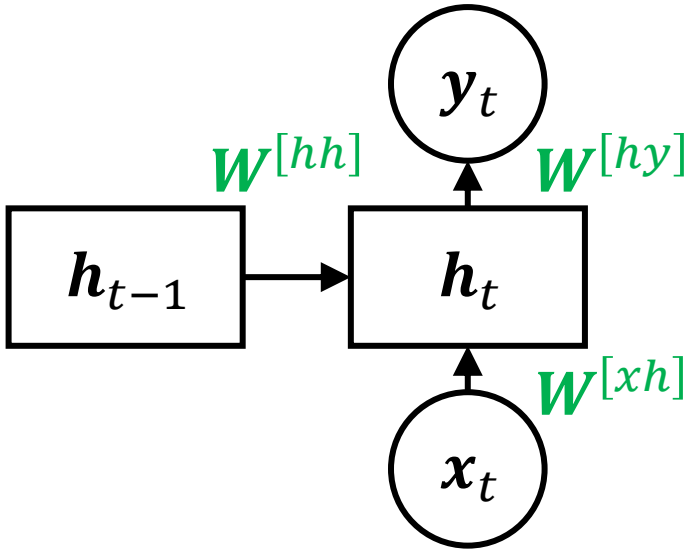
**Question:** Do these weights change for different time $t$?




Feedforward Neural Network

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

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




Recurrent Neural Network

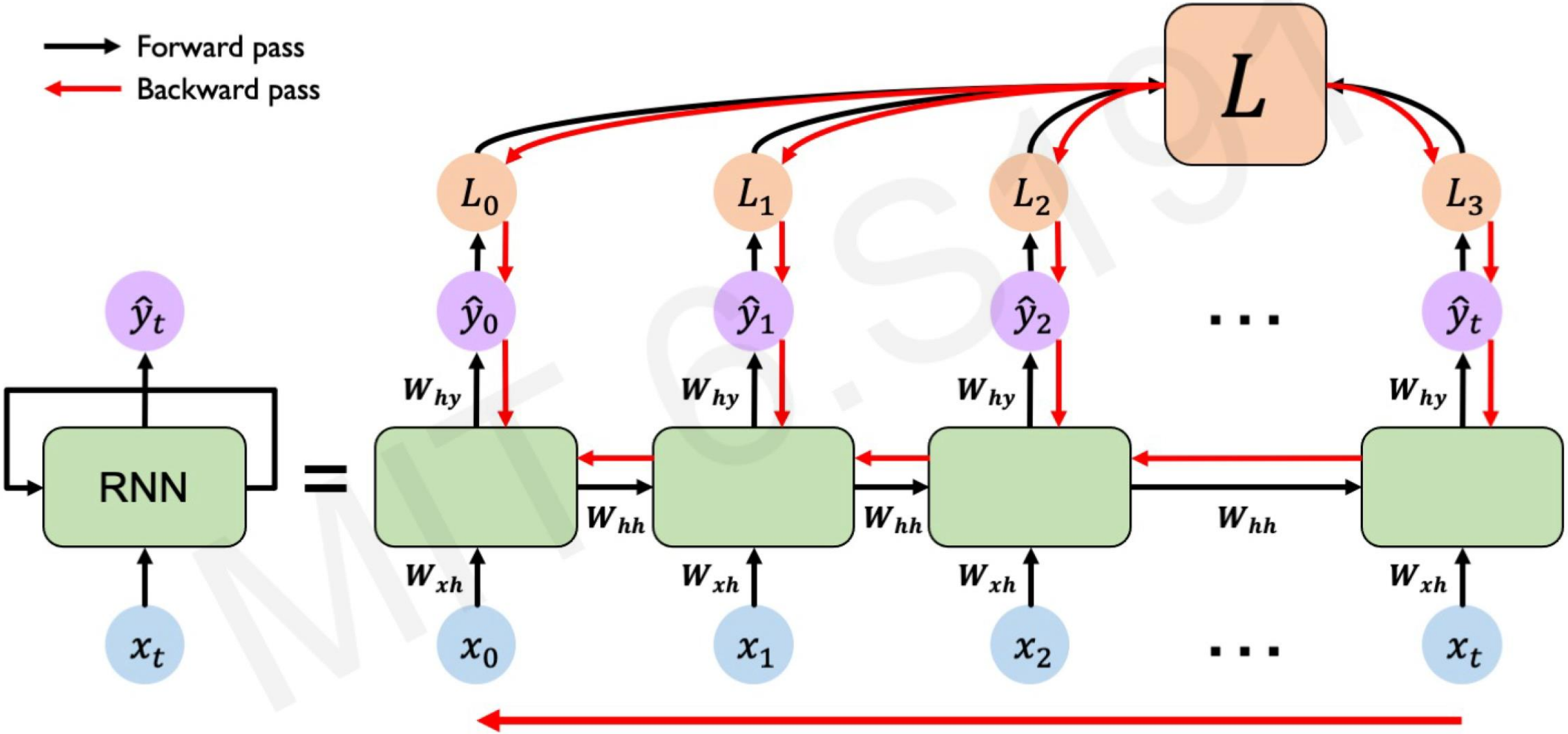
$$y_t = g^{[hy]}\left(\left(W^{[hy]}\right)^{\top} h_t\right)$$

$$h_t = g^{[hh]}\left(\left(W^{[xh]}\right)^{\top} x_t + \left(W^{[hh]}\right)^{\top} h_{t-1}\right)$$

$$h_t = g^{[hh]}\left(\left(W^{[xh]} \oplus W^{[hh]}\right)^{\top} (x_t \oplus h_{t-1})\right)$$


2021

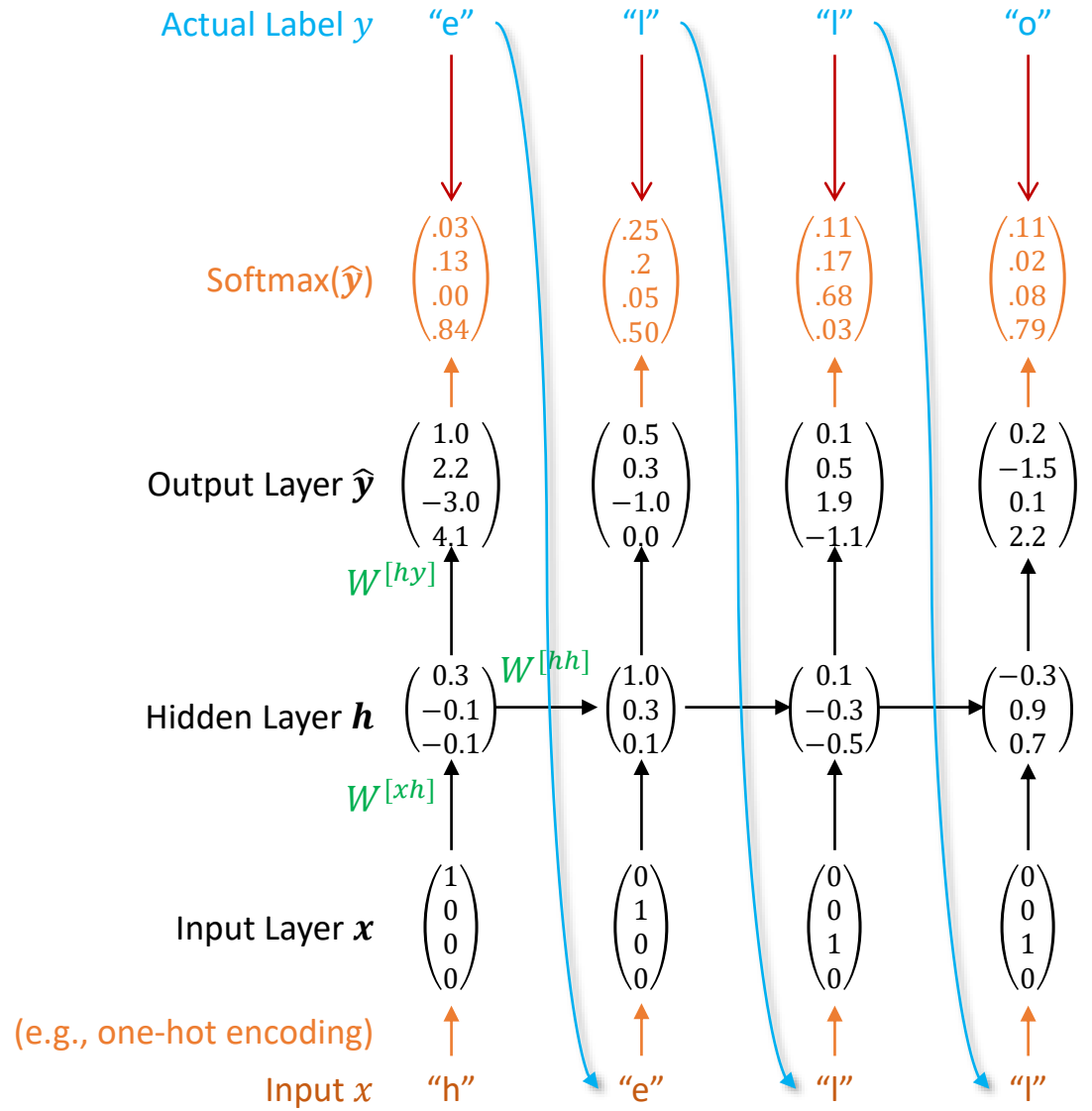
# RNNs: Backpropagation Through Time



Example RNN

# Text character prediction

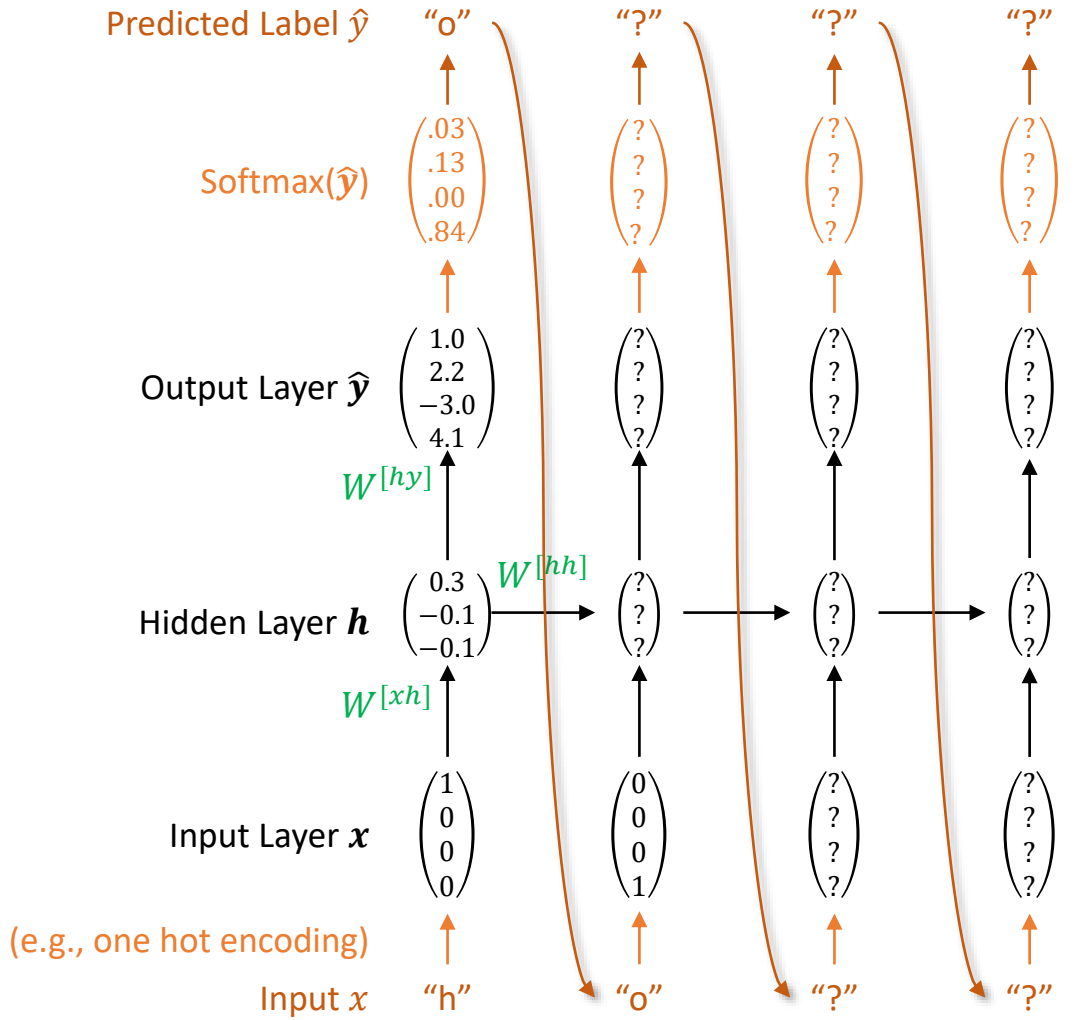
- Dictionary
  - [h, e, l, o]
- Training: sequence “hello”
- Encoding and Decoding chars
  - One-hot encoding (e.g., BOW)
  - Softmax classification
- At training time,
  - $x_t = y_{t-1}$
  - Loss (Error) is calculated as cross-entropy loss between $\hat{y}_t$ and $y_t$



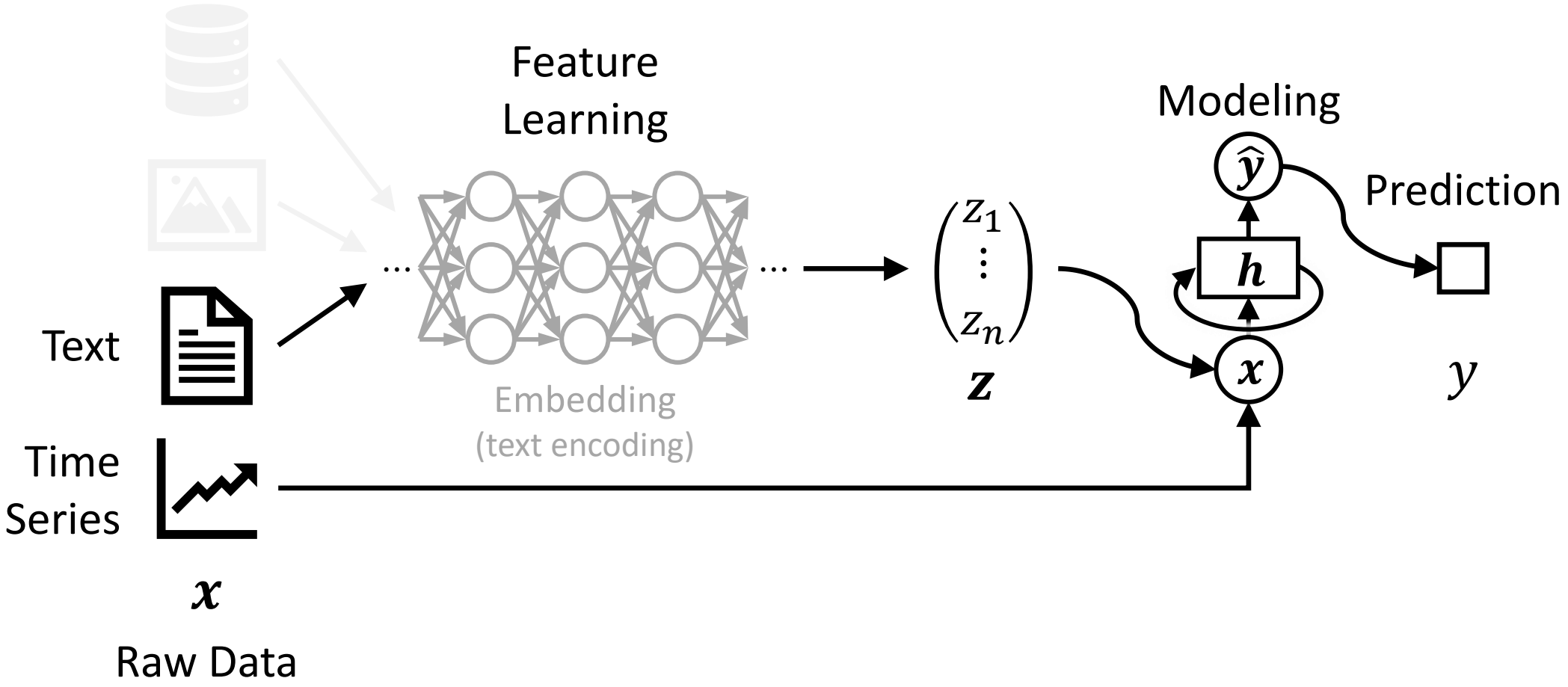
Example RNN

# Text character prediction

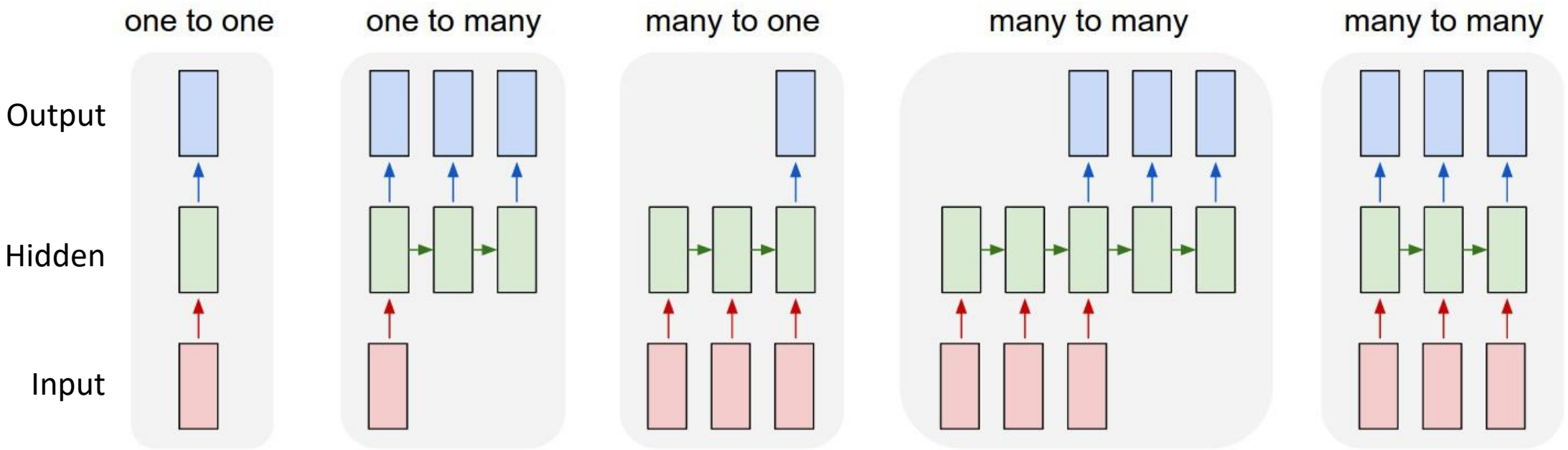
- Bag-of-Words dictionary
  - [h, e, l, o]
- At prediction time,
  - $x_t = \hat{y}_{t-1}$
  - Then propagate calculating activations to generate sequence of characters



# 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

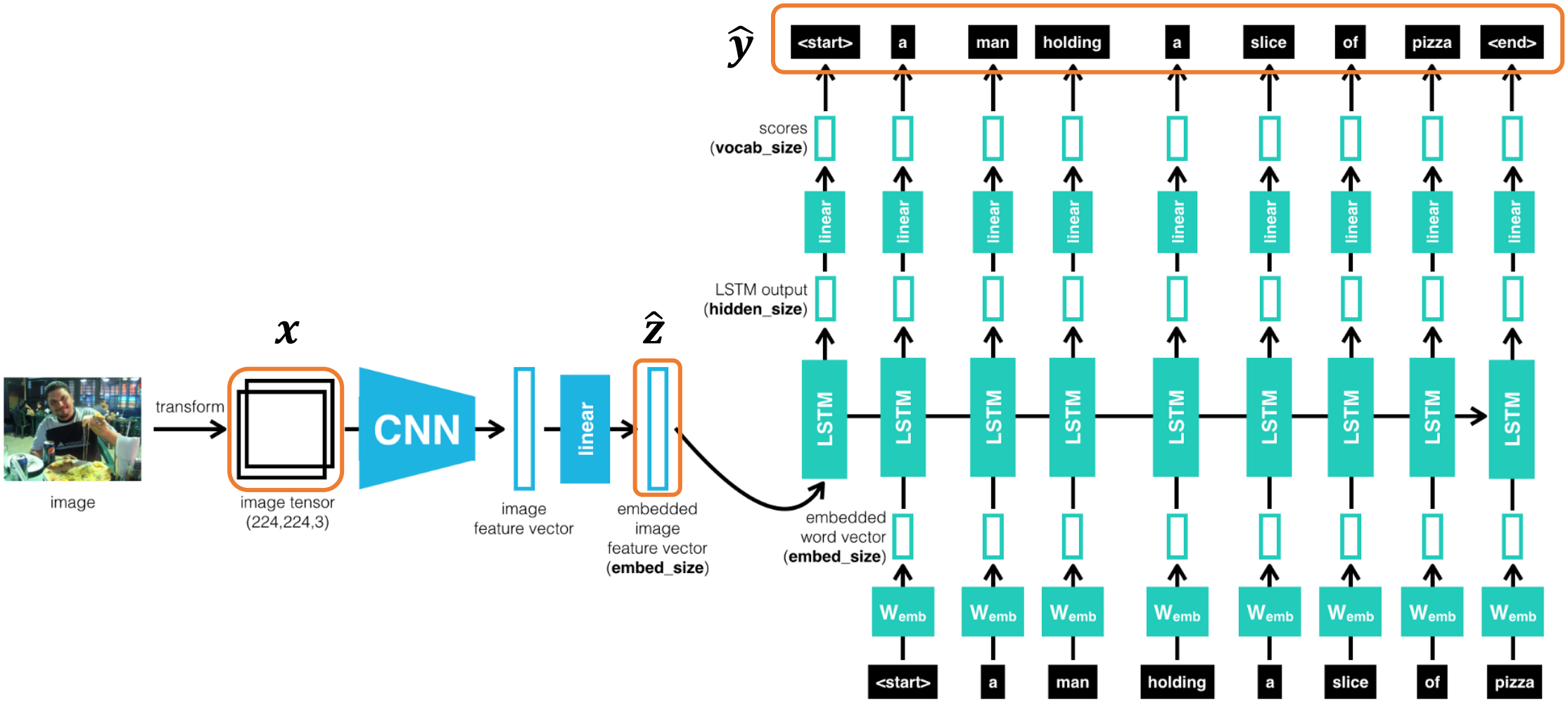
Encoder-Decoder
Processes *all inputs first*, then predicts output

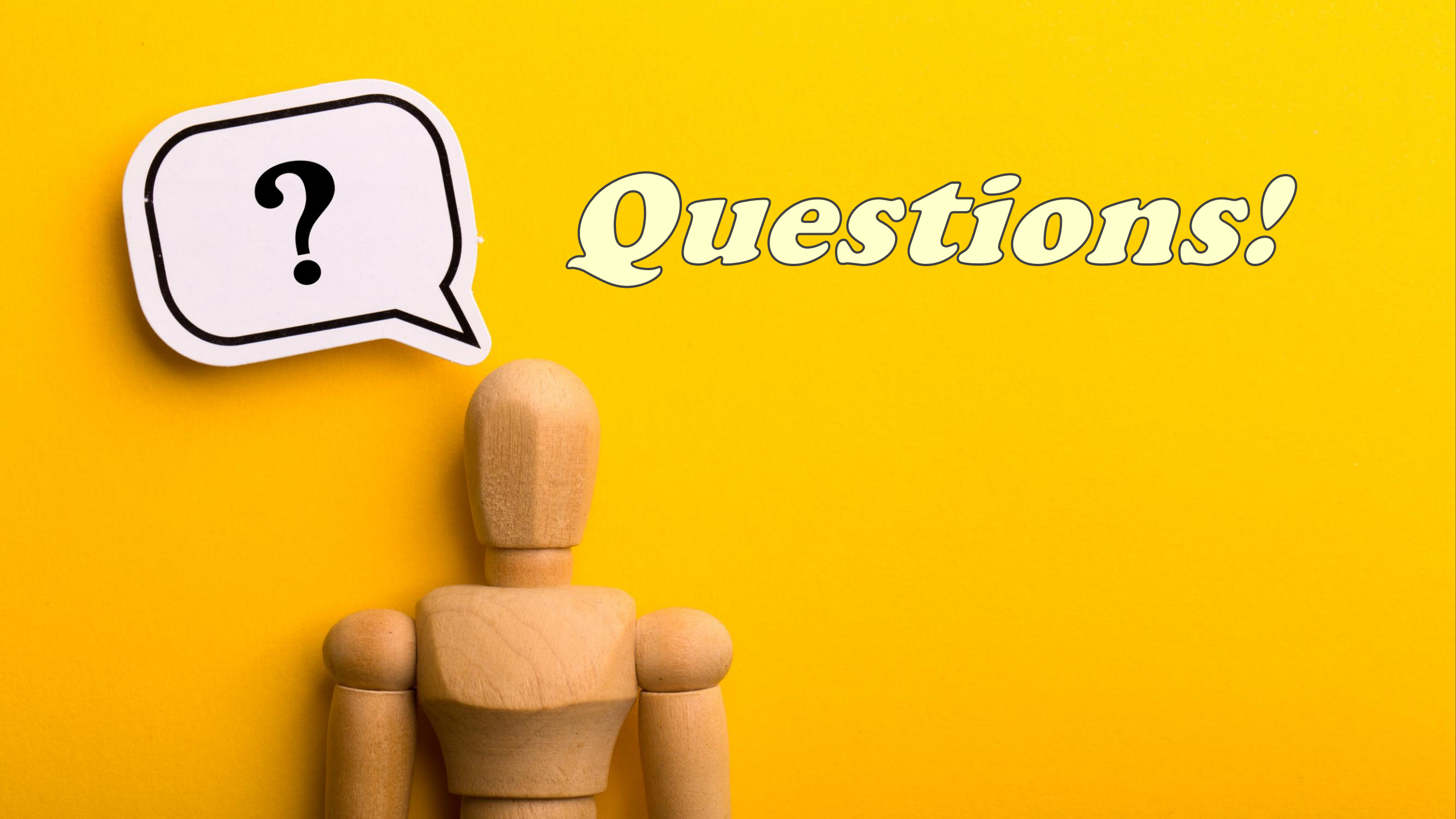
Plain RNN with *time dependency*

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!

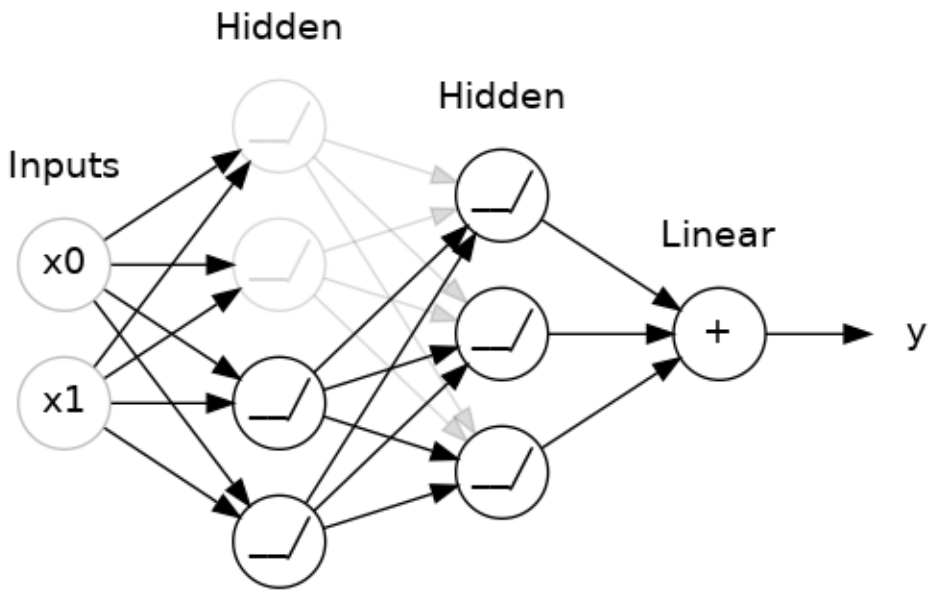
# Deep Learning Training Issues

# 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

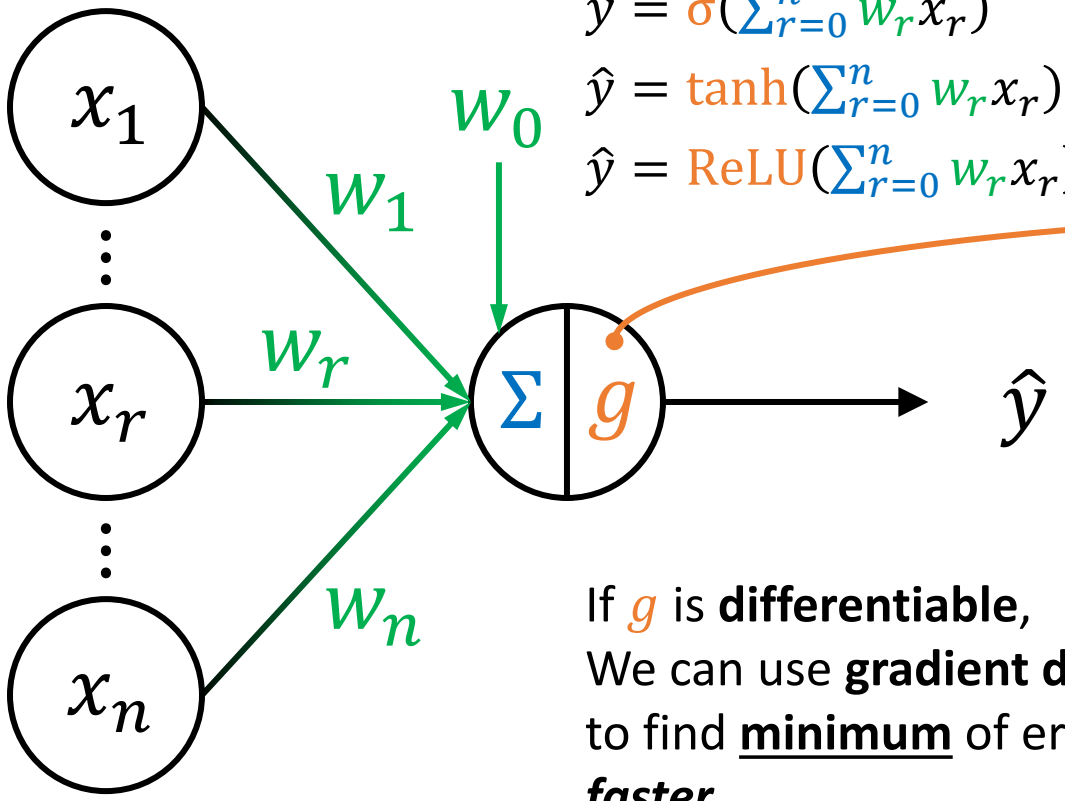



Further reading: https://towardsdatascience.com/12-main-dropout-methods-mathematical-and-visual-explanation-58cdc2112293

# Deep Learning Training Issues

- Overfitting
- Saturating Gradient Problem
- Vanishing Gradient Problem

# Differentiable Activation Functions




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

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

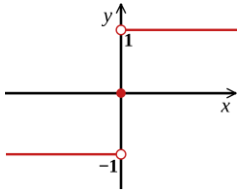
$\hat{y} = \tanh(\sum_{r=0}^{n} w_r x_r)$

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

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

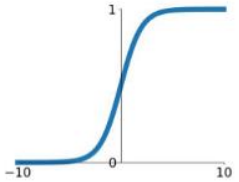
**Step**

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



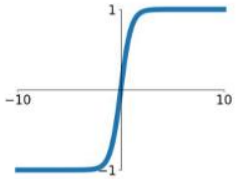

**Sigmoid**

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




**tanh**

$$\tanh(x)$$




**ReLU**

$$\max(0, x)$$

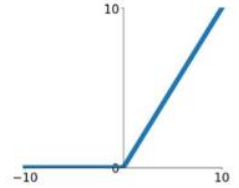



Image credit:
https://miro.medium.com/max/1400/0*sIJ-gbjlz0zrz8lb.png

# Gradient Descent Weight Update

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

- $\boldsymbol{W}$: New weight
- $\boldsymbol{W}$: Old weight
- $\eta$: Learning Rate
- $\varepsilon$: Error

MSE error

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

Gradient of error

$$\nabla \varepsilon = \frac{\partial \varepsilon}{\partial \boldsymbol{W}} = \frac{\partial \varepsilon}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \boldsymbol{W}}$$

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

Reference

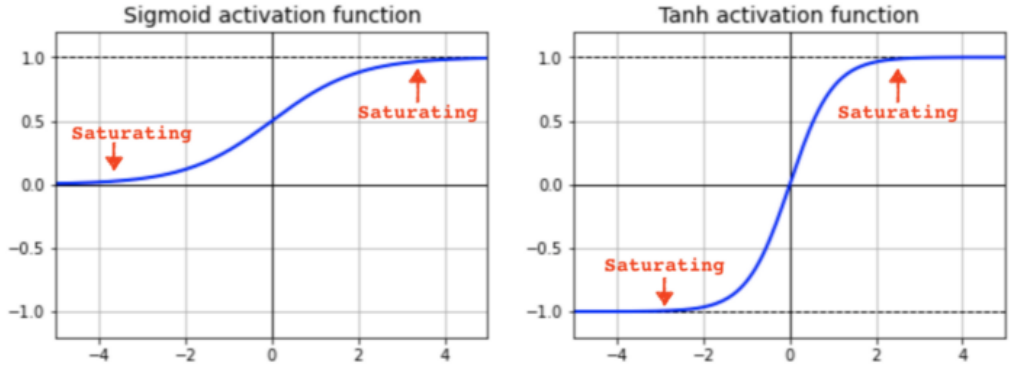
$$\boldsymbol{a}^{[l]} = g^{[l]}(f^{[l]})$$

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

# **Saturating Gradient** Problem due to activation functions

# Mitigate with ReLU activation function

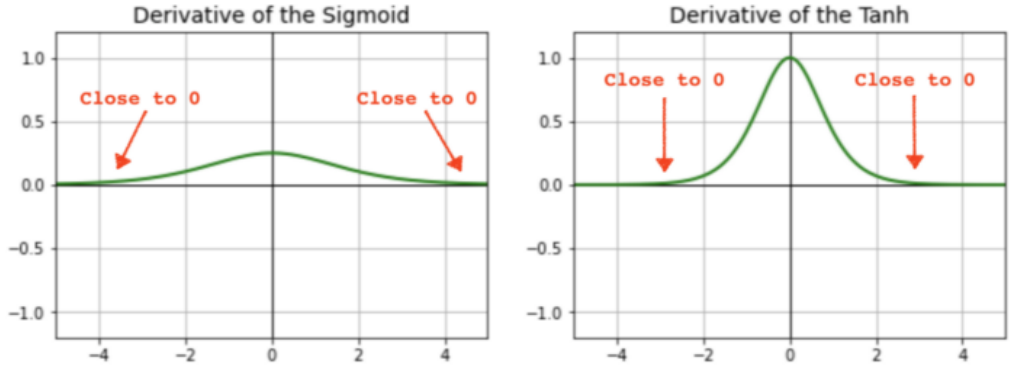
$g(f)$







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

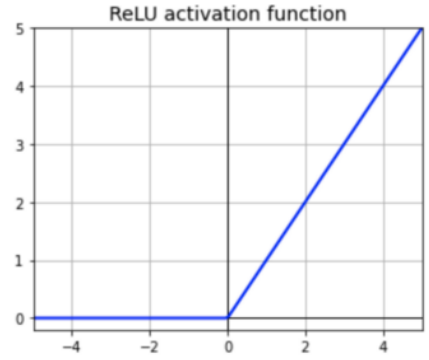


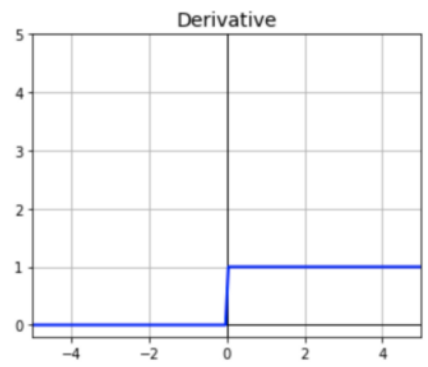




When $x$ value far from $0$, gradient $\rightarrow 0$ (saturating)
When gradient $\approx 0$, then weights don't update much

**Mitigation**




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

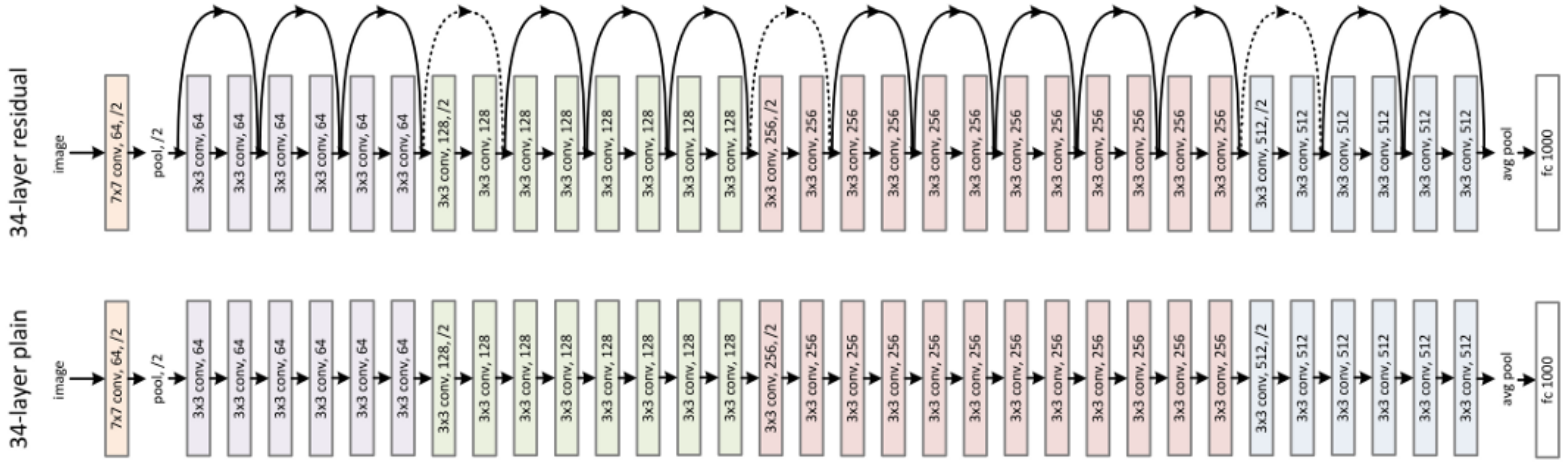
$$\hat{y}'(\boldsymbol{W}^{[1]}) = \frac{\partial g^{[L]}}{\partial \boldsymbol{W}^{[1]}} = \frac{\partial f^{[1]}}{\partial \boldsymbol{W}^{[1]}} \frac{\partial g^{[1]}}{\partial f^{[1]}} \cdots \frac{\partial g^{[l]}}{\partial f^{[l]}} \frac{\partial f^{[l+1]}}{\partial g^{[l]}} \frac{\partial g^{[l+1]}}{\partial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

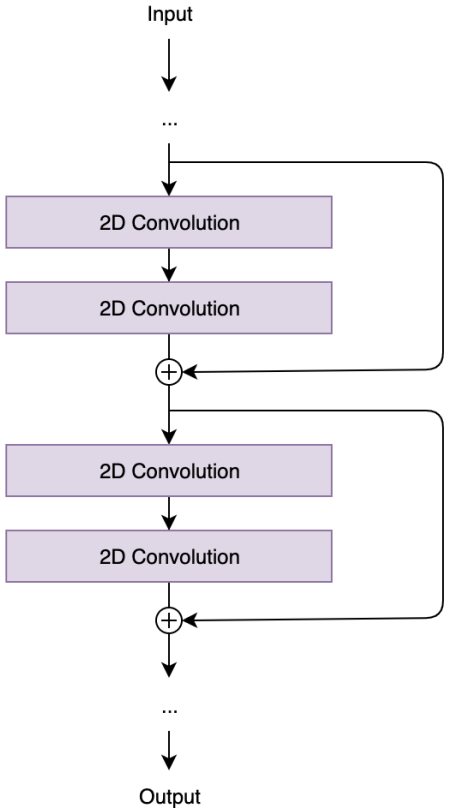
If some gradients are small ($< 1$),
multiplying many small numbers equals a very small number.
E.g., $0.5^{15} \approx 0.0003$

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



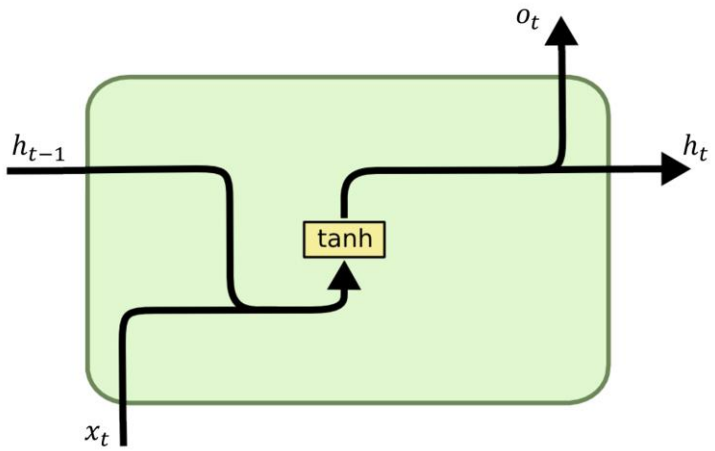




Further reading: https://towardsdatascience.com/vggnet-vs-resnet-924e9573ca5c
Image credit: https://www.kaggle.com/keras/resnet50

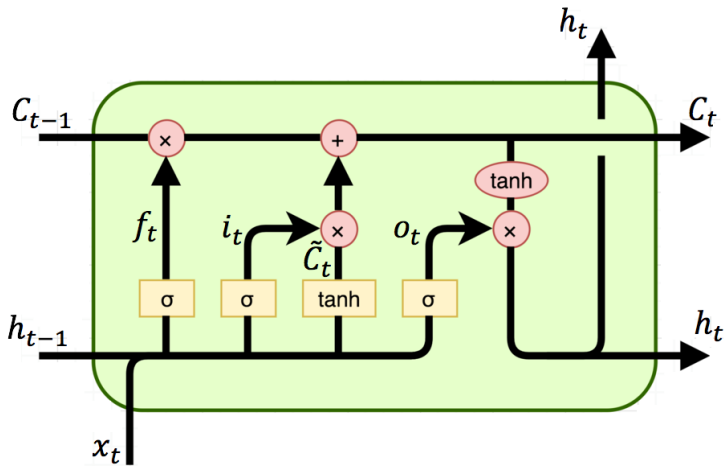
## Mitigating Vanishing Gradients in RNN

# Using architectures with "forget" gates

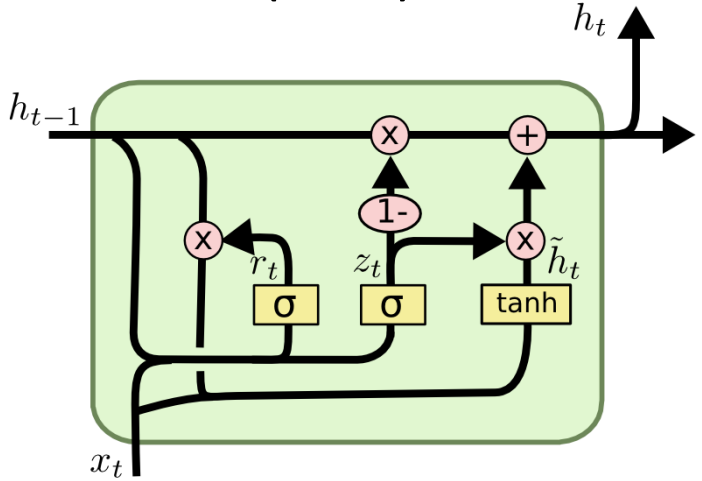
Plain RNN




Long-Short Term Memory (LSTM)




Gated Recurrent Unit (GRU)



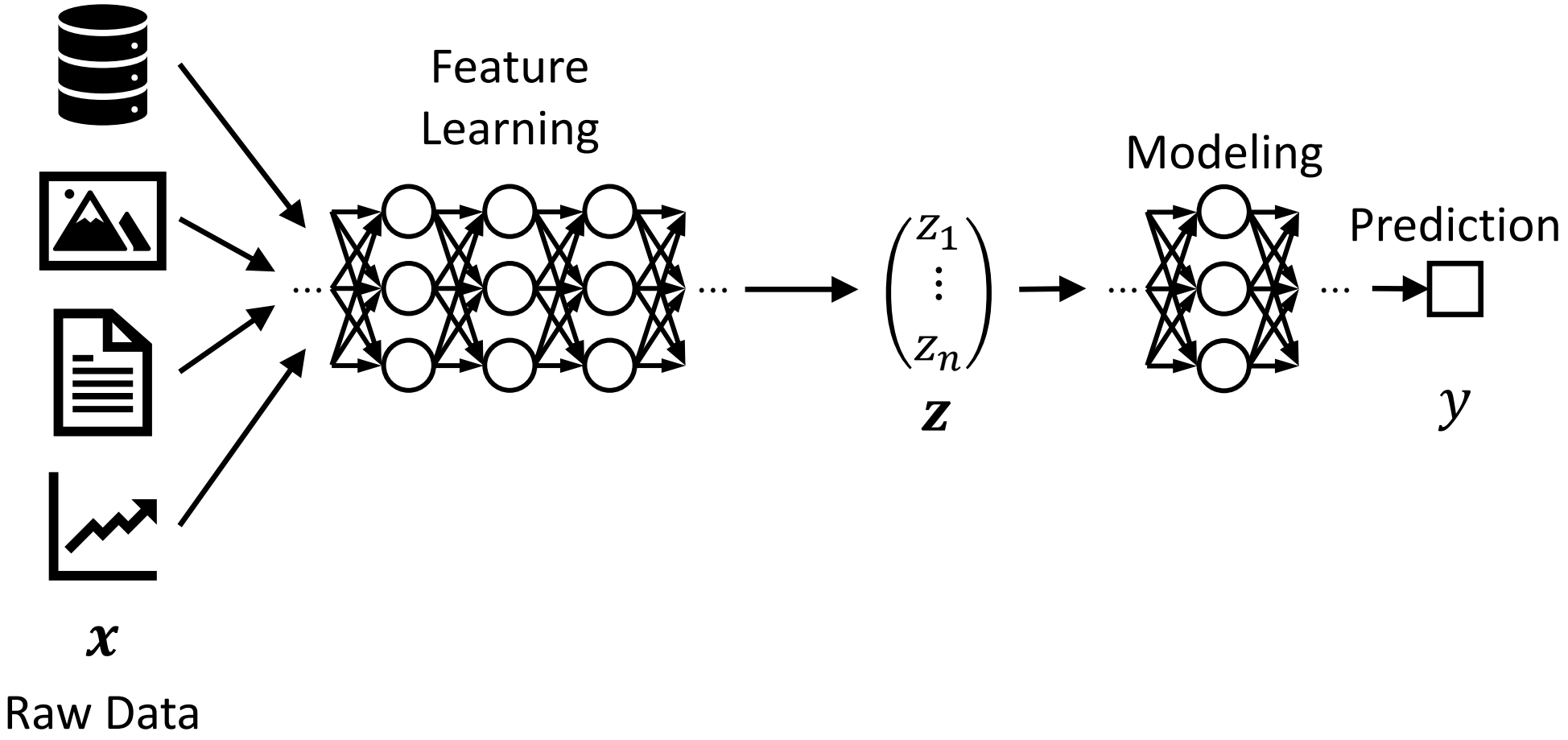

Includes **"forget" gates**

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

# Wrapping Up


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

# From Manual Feature Engineering To Architecture Engineering

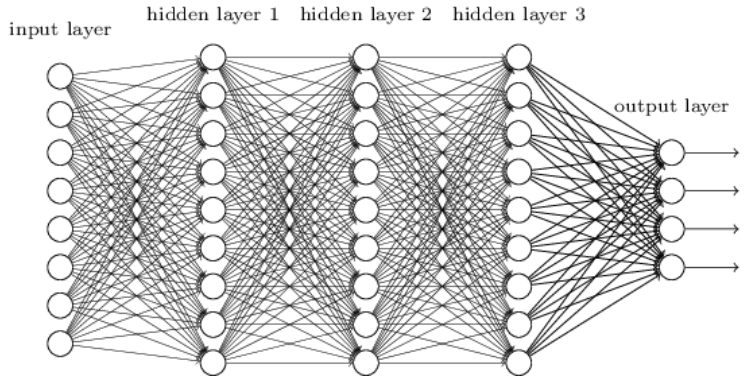


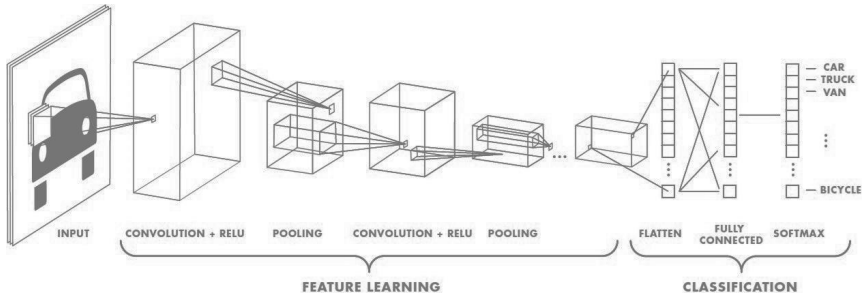
# What did we learn?

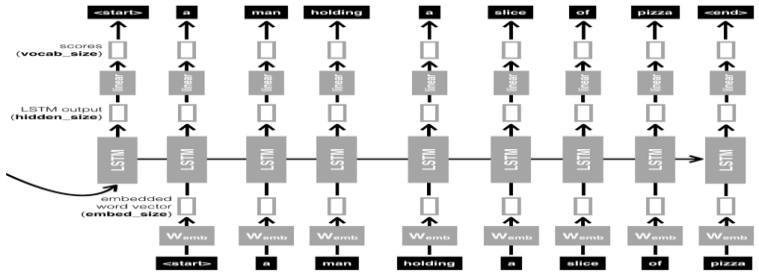
- Feature Engineering → Architecture Engineering

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

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



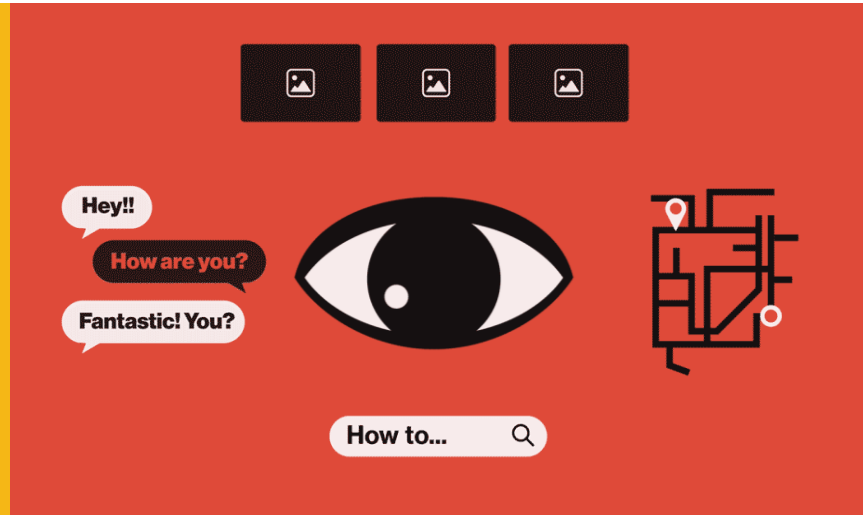



# Grand issues with AI (Deep Learning)

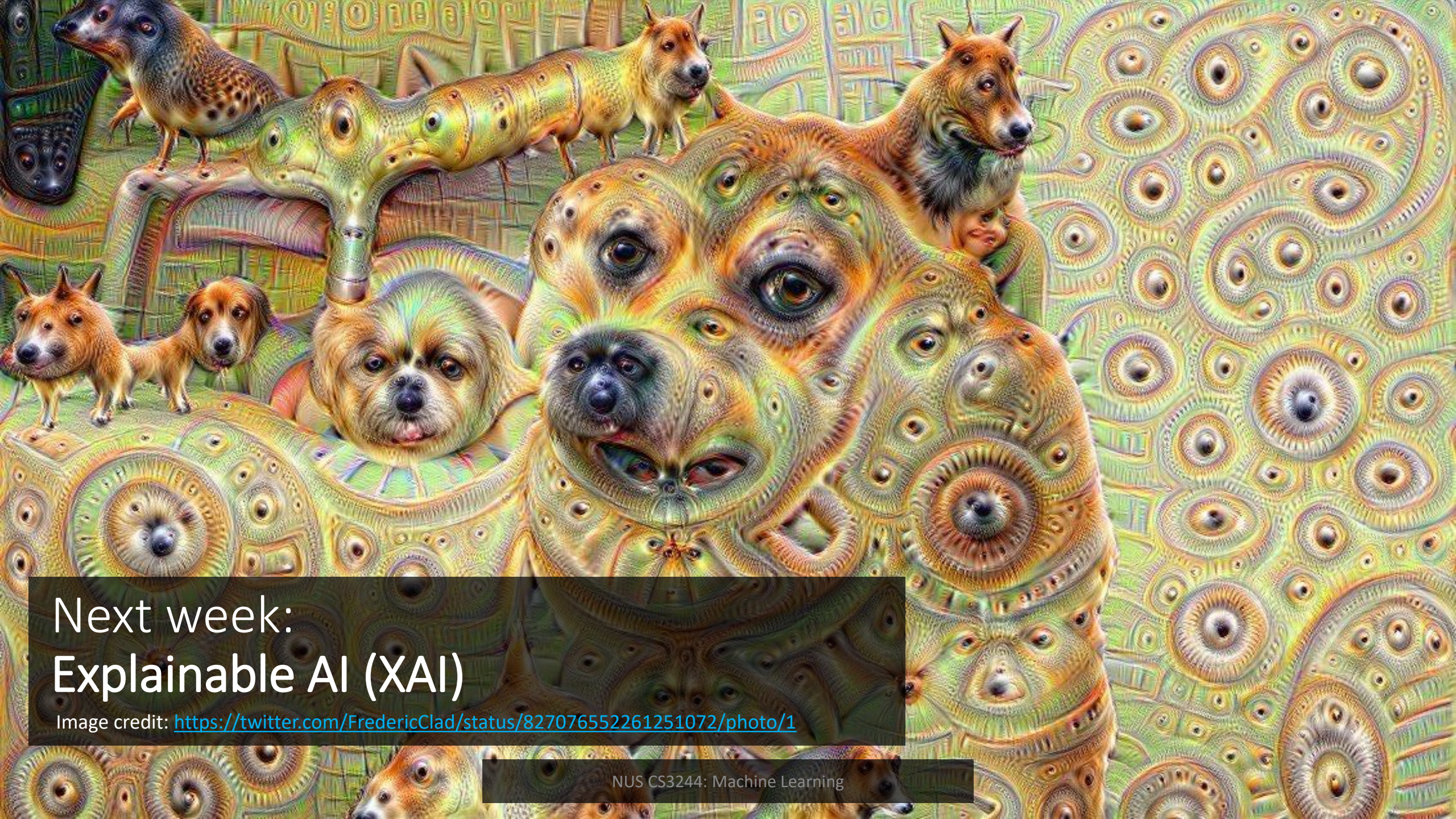
Lack of **Explainability** [W10b]

**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