CS2109S: Introduction to AI and Machine Learning

Lecture 10: Introduction to Deep Learning

3 November 2023

Announcement

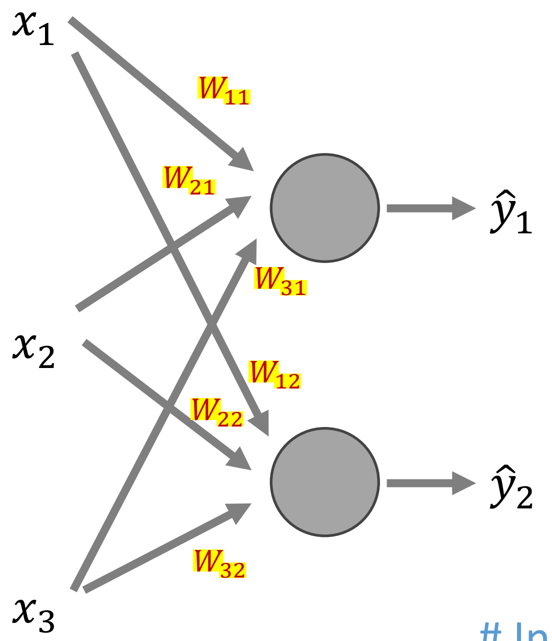
- Midterm grading is still on progress, sorry :(
  - Q1 is quite painful to grade
  - Probably will be done by **next week**…
- We’ll release the mockup final assessment next week (hopefully)

Recap

- Backpropagation
  - Backpropagation on different scenarios:
    - Path, branches, many features, and many samples (sum the gradients)
  - Biological plausibility of backpropagation – believed to be **<u>not</u> feasible**
- Automatic Differentiation
  - Reverse mode automatic differentiation – **backprop is a special case**: $\mathbb{R}^N \to \mathbb{R}$
  - ~~Comparison with other methods: **symbolic** and **numerical** differentiation~~
- Introduction to PyTorch
  - Tensors
    - **n-dimensional array representation** with GPU support
    - Maintain **computational graph**
  - Modules & Functions: Linear (linear), ReLU (relu), etc – they are equivalent
  - Loss function & Optimizers

Indexing start from 1,1 row then column

Neural Networks and Matrix Multiplication (1)



Input (number of weights per neuron / input variables)

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad W = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \end{bmatrix} \quad \hat{\boldsymbol{y}} = g(W^T\boldsymbol{x}) = g\left(\begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) = g\left(\begin{bmatrix} W_{11} & W_{21} & W_{31} \\ W_{12} & W_{22} & W_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix}$$

Output (number of layer's neurons / output variables)

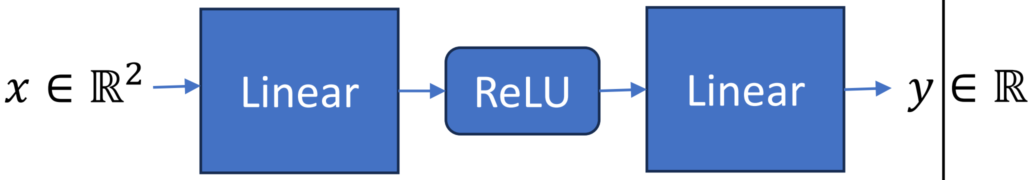
ERRATA

Modules and Functions API: Example

```python
class NeuralNetRegressor(torch.nn.Module):
    def __init__(self, input_size, hidden_size):
        super().__init__()
        self.linear1 = torch.nn.Linear(input_size, hidden_size)
        self.linear2 = torch.nn.Linear(hidden_size, 1)
        self.relu = torch.nn.ReLU()

    def forward(self, x):
        f1 = self.linear1(x)
        a1 = self.relu(f1)
        f2 = self.linear2(a1)
        return f2

model1 = NeuralNetClassifier(2,8) # 2 features, 8 hidden neurons
model2 = torch.nn.Sequential(torch.nn.Linear(2,8), torch.nn.ReLU(), torch.nn.Linear(8,1)) # same
```




```python
w1 = torch.tensor(8, 2, requires_grad=True)
w2 = torch.tensor(1, 8, requires_grad=True)

def neural_net_regressor(x): # also the same
    f1 = torch.nn.functional.linear(x,w1)
    a1 = torch.nn.functional.relu(f1)
    return torch.nn.functional.linear(a1,w2)
```

RECAP

Gradient Descent

- Start at some w
- Pick a nearby w that reduces $J(w)$

$$w_j \leftarrow w_j - \gamma \frac{\partial J(w_0, w_1, \ldots)}{\partial w_j}$$

- Repeat until minimum is reached

Single-layer Neural Networks with Sigmoid

$$\gamma(\hat{y} - y)\hat{y}(1 - \hat{y})x_i$$

Derive manually!

Multi-layer Neural Networks

Backpropagation!

RECAP

Example 2: Training NN using modular API

```
model = NeuralNetRegressor()


optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
loss_function = torch.nn.MSELoss()


for epoch in range(num_epochs):
    optimizer.zero_grad()
    y_pred = model(x)
    loss = loss_function(y_pred, y)
    loss.backward()
    optimizer.step()
```

Gradient Descent

- Start at some w
- Pick a nearby w that reduces $J(w)$

$$w_j \leftarrow w_j - \gamma \frac{\partial J(w_0, w_1, \dots)}{\partial w_j}$$

- Repeat until minimum is reached

Outline

- Deep Neural Networks
- Convolution Neural Networks
  - Motivation: handling spatial structure
  - Convolution, Pooling Layer, and Common Architectures
  - Applications
- Recurrent Neural Networks
  - Motivation: handling sequential data
  - Recurrent Neural Networks and Variants
  - Applications
- Attention, Transformers, GPT, and ChatGPT (if time permits)
- Issues with Deep Learning

Outline

- **Deep Neural Networks**
- Convolution Neural Networks
  - Motivation: handling spatial structure
  - Convolution, Pooling Layer, and Common Architectures
  - Applications
- Recurrent Neural Networks
  - Motivation: handling sequential data
  - Recurrent Neural Networks and Variants
  - Applications
- Attention, Transformers, GPT, and ChatGPT (if time permits)
- Issues with Deep Learning

Looking Back…

What is AI?

Deep Learning System

ChatGPT

Credit: IEEE Spectrum

Credit: Guardian

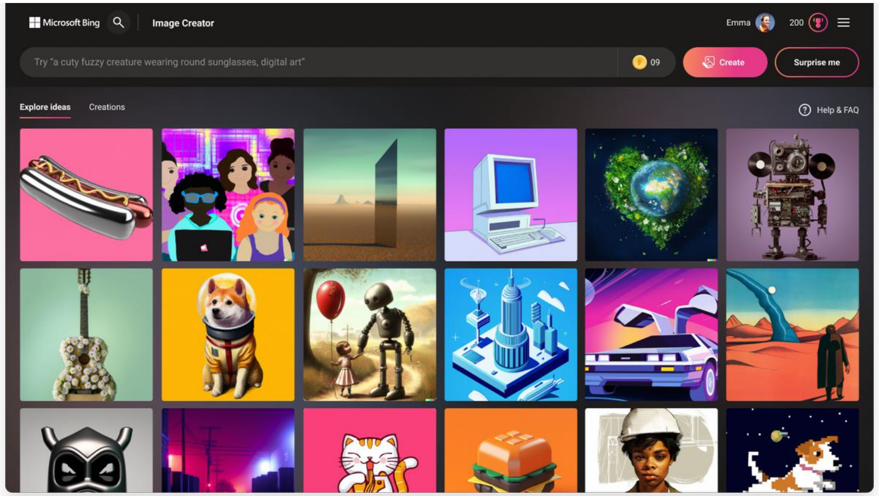
Credit: NYTimes

Looking Back…

What is AI?

Deep Learning System

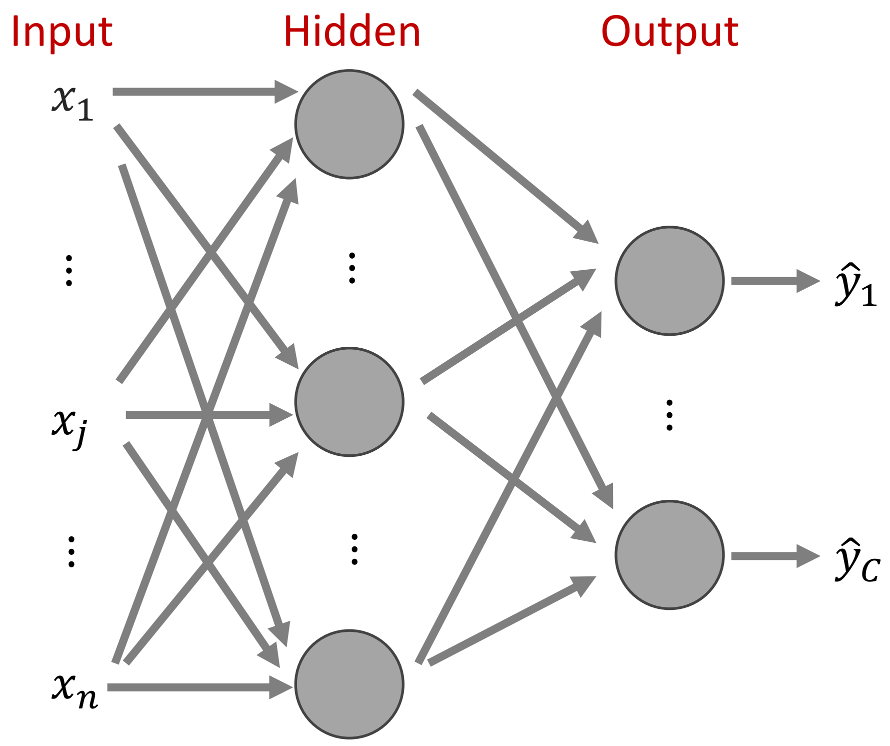
Credit: Tesla



Credit: Microsoft

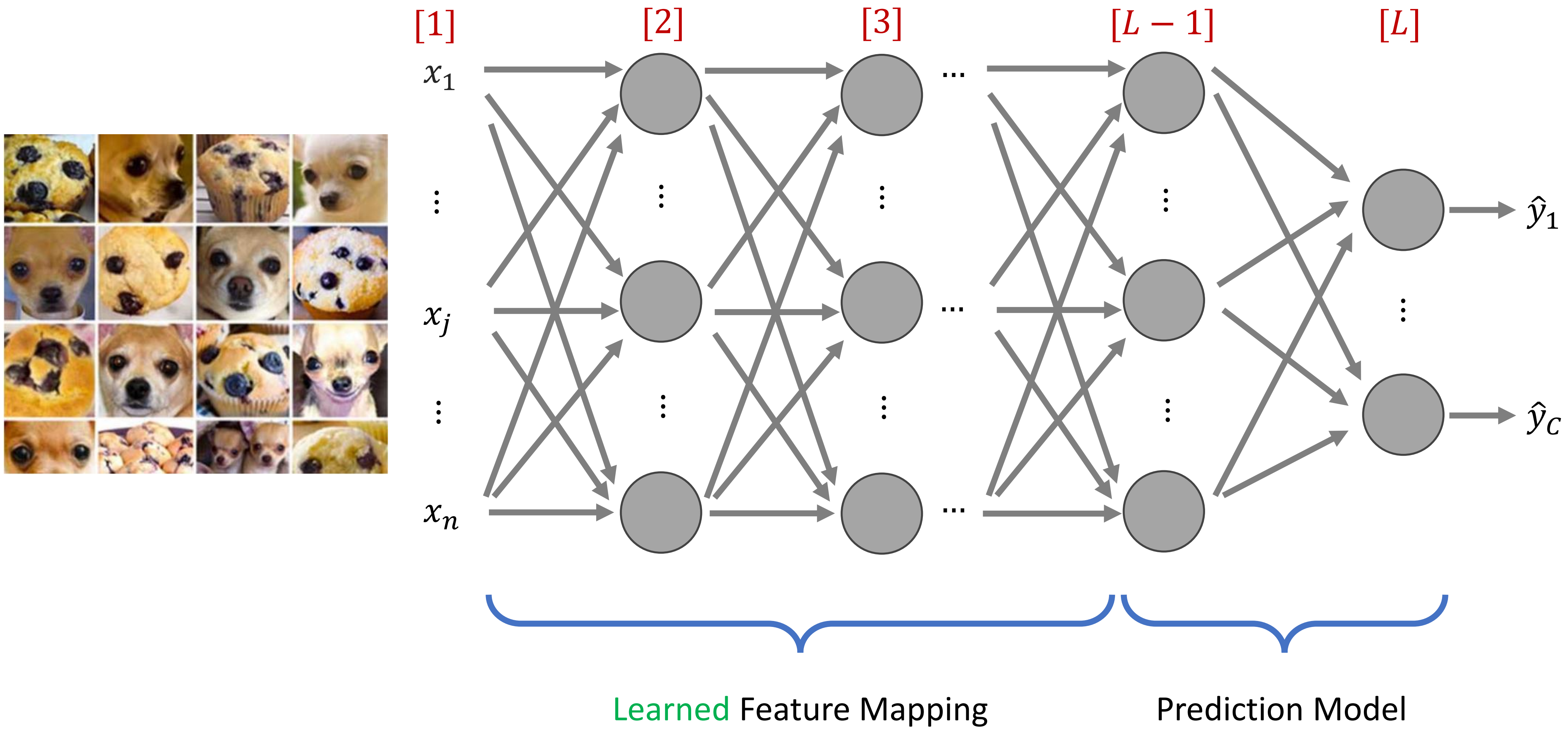
Credit: Eden AI

Shallow Neural Networks



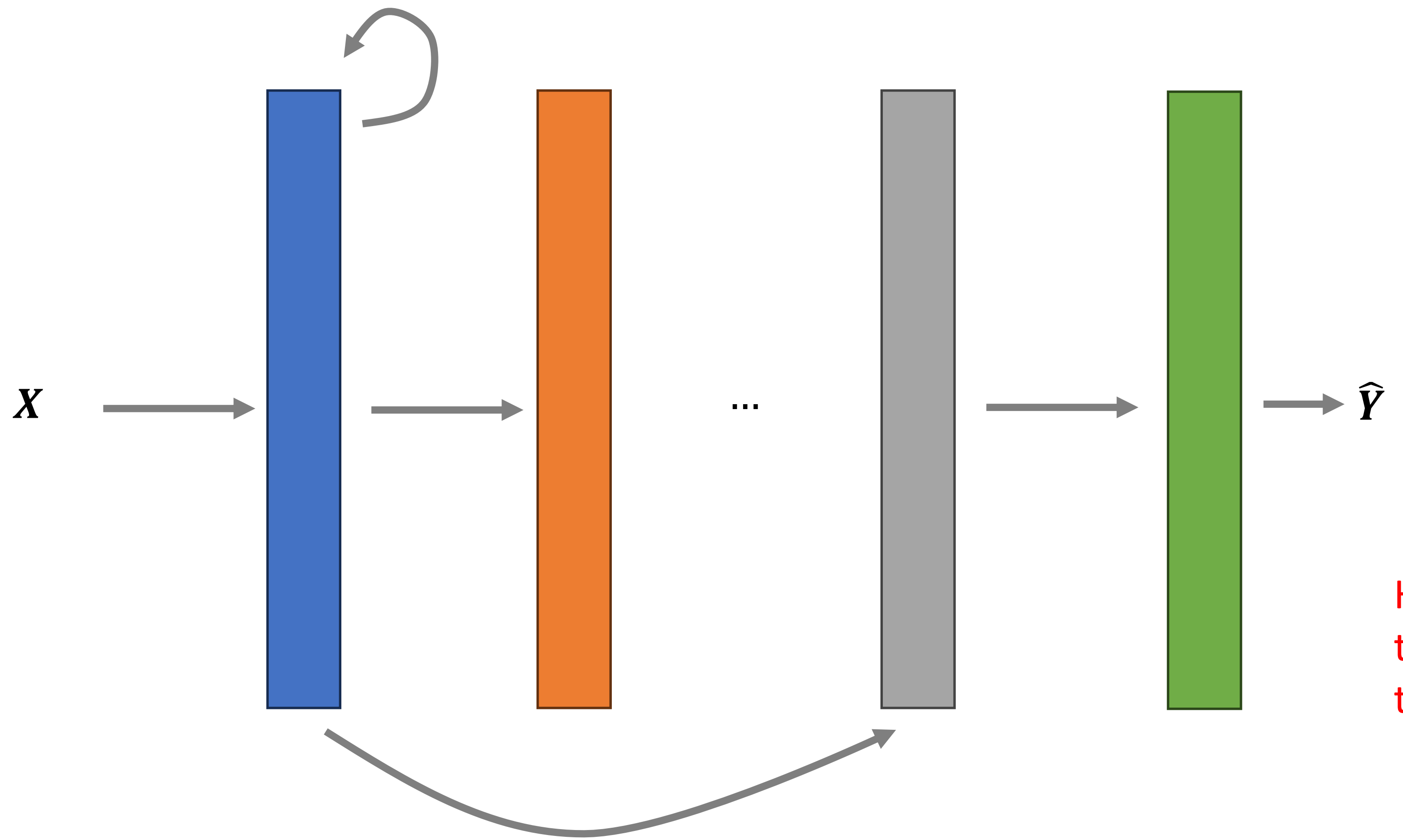
Deep Neural Networks

Neural Network with the number of layers $L > 3$



Deep Neural Networks

Neural Network with the number of layers $L > 3$



How do we decide the functions and the compositions?

Arbitrary function compositions: can have any* functions and any* compositions!

XNOR, XOR

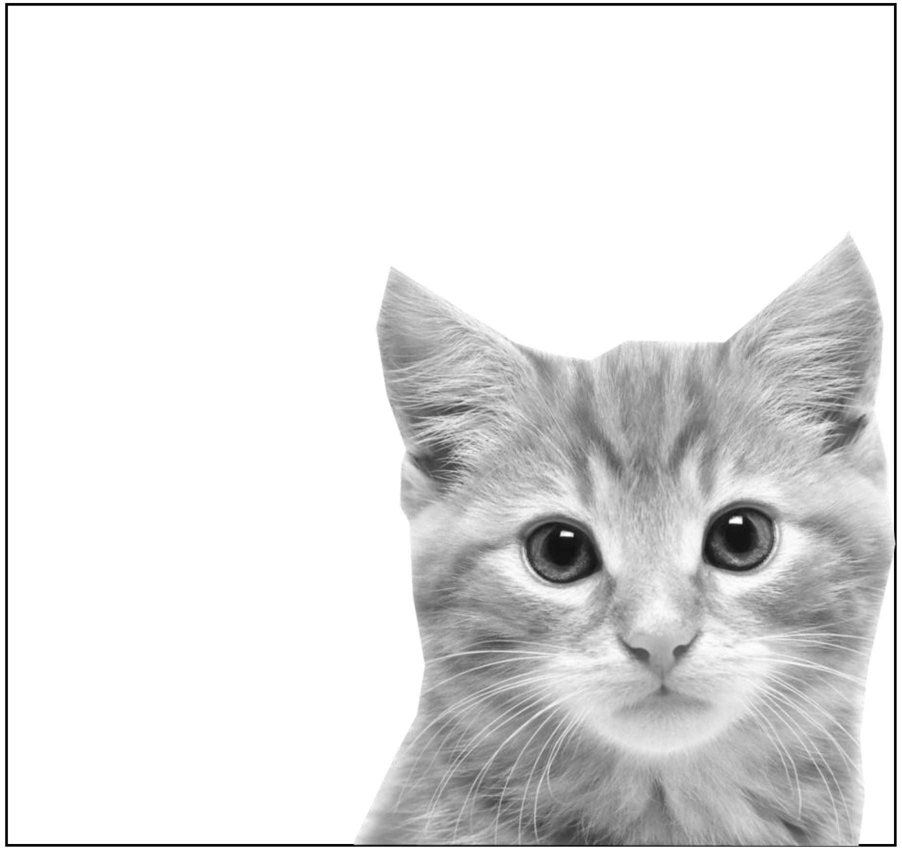
Differentiable

T&C applies

Outline

- Deep Neural Networks
- **Convolution Neural Networks**
  - Motivation: handling spatial structure
  - Convolution, Pooling Layer, and Common Architectures
  - Applications
- Recurrent Neural Networks
  - Motivation: handling sequential data
  - Recurrent Neural Networks and Variants
  - Applications
- Attention, Transformers, GPT, and ChatGPT (if time permits)
- Issues with Deep Learning

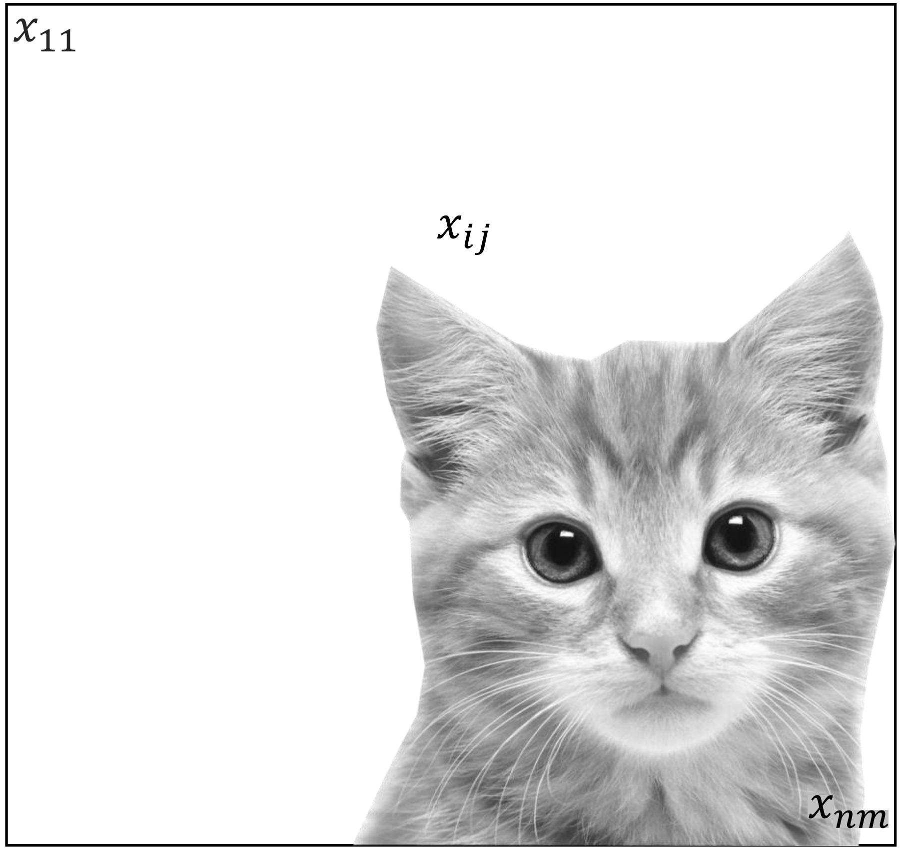
Computer Vision Problem



Cat or **Dog**?

Image credit: https://www.deviantart.com/klara1008/art/Cat-png-277228793

Computer Vision Problem: A Naïve Attempt



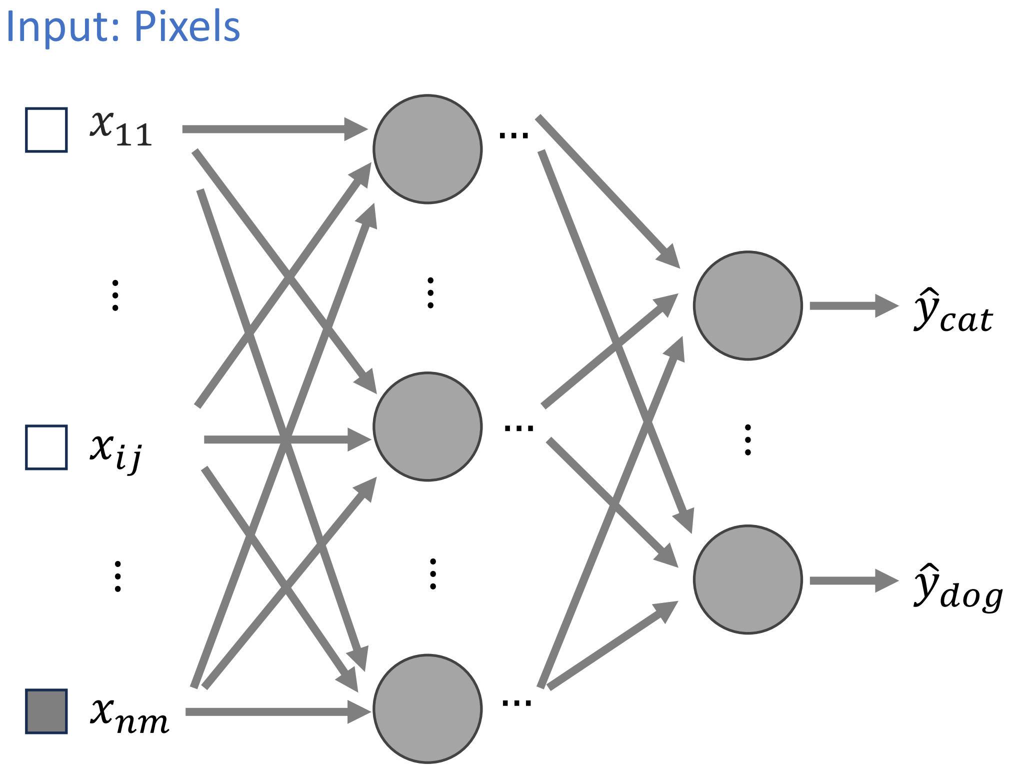


Image credit: https://www.deviantart.com/klara1008/art/Cat-png-277228793

Computer Vision Problem: A Naïve Attempt

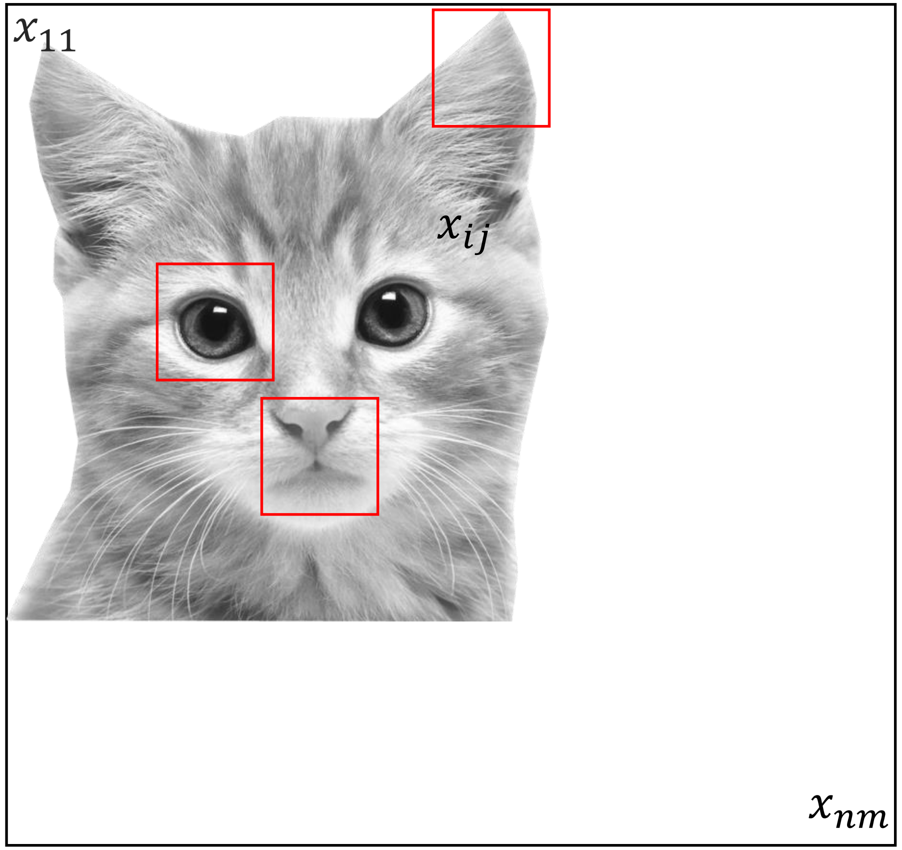
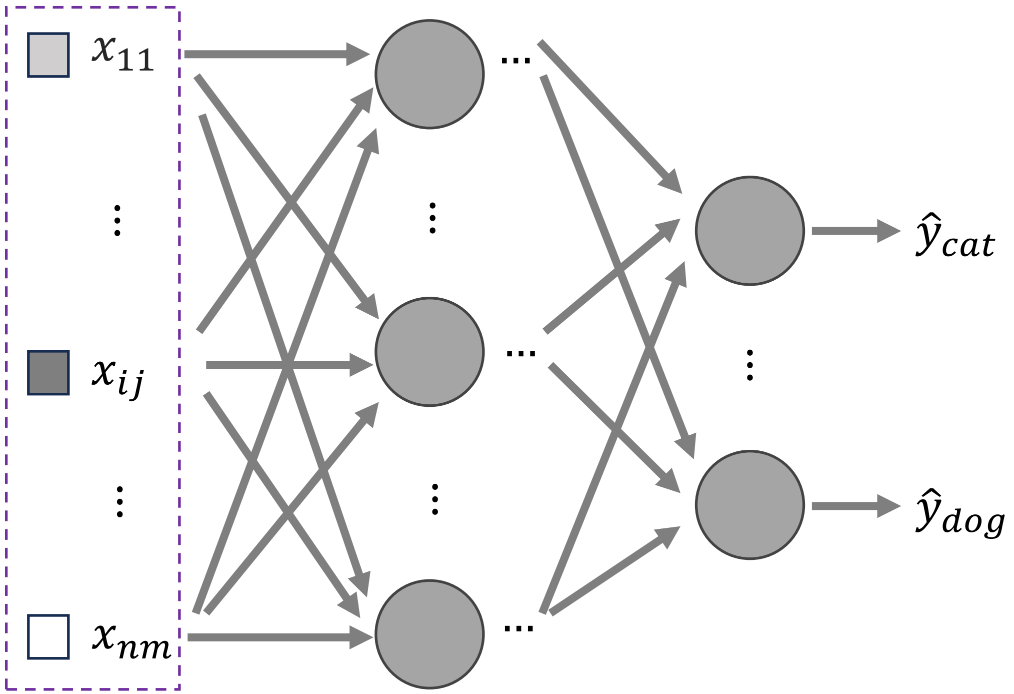


Image has *spatial structures*, which are **ignored**

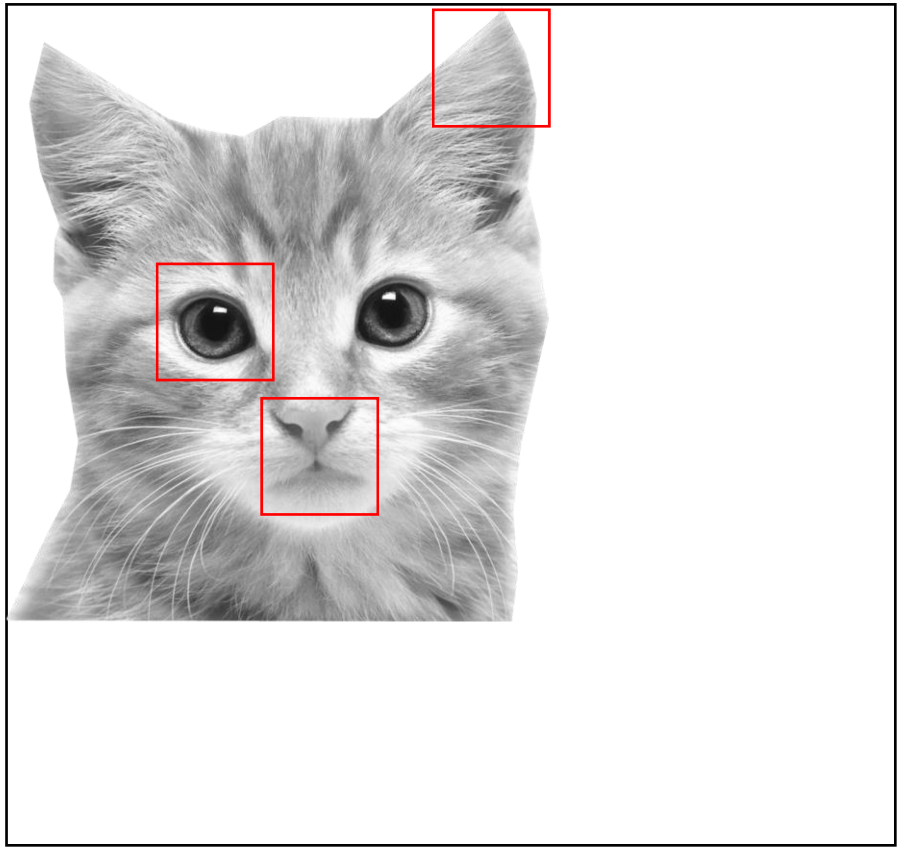
Input: Pixels



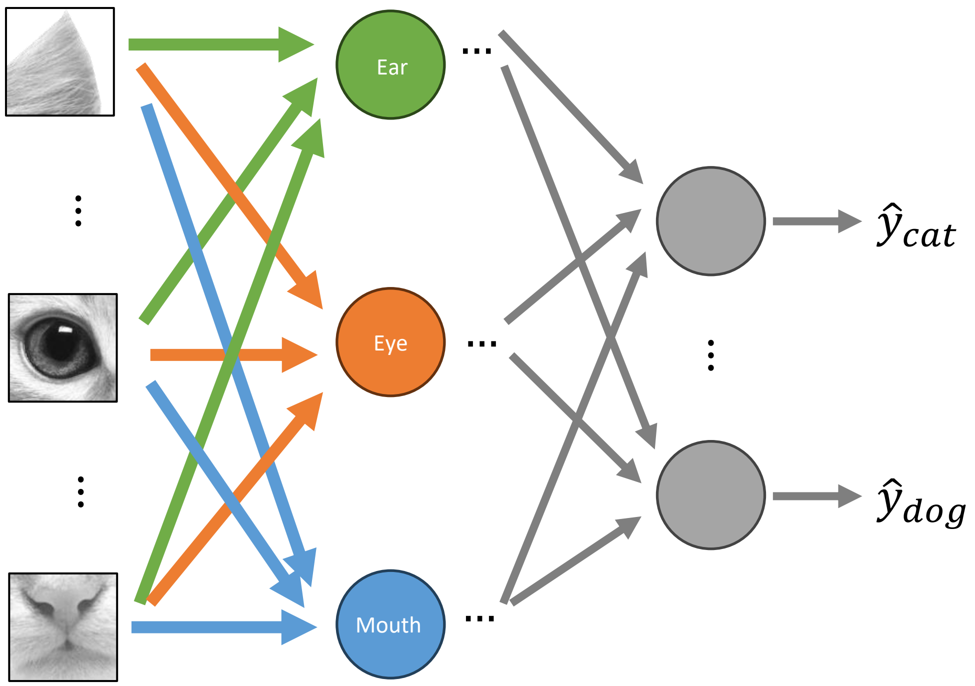
Same image, shifted → **dramatic change** in input of NN

Image credit: https://www.deviantart.com/klara1008/art/Cat-png-277228793

Computer Vision Problem: A Better Idea



Input: A **group** of Pixels



Note: same colour → same function

Image credit: https://www.deviantart.com/klara1008/art/Cat-png-277228793

Computer Vision Problem: A Better Idea

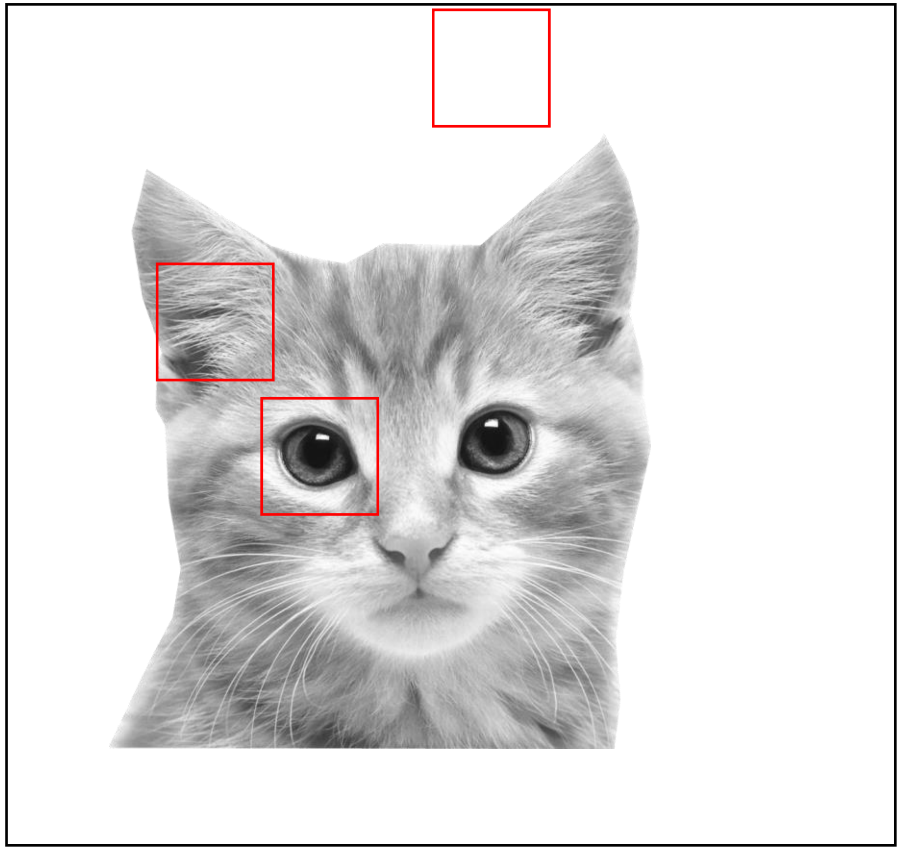
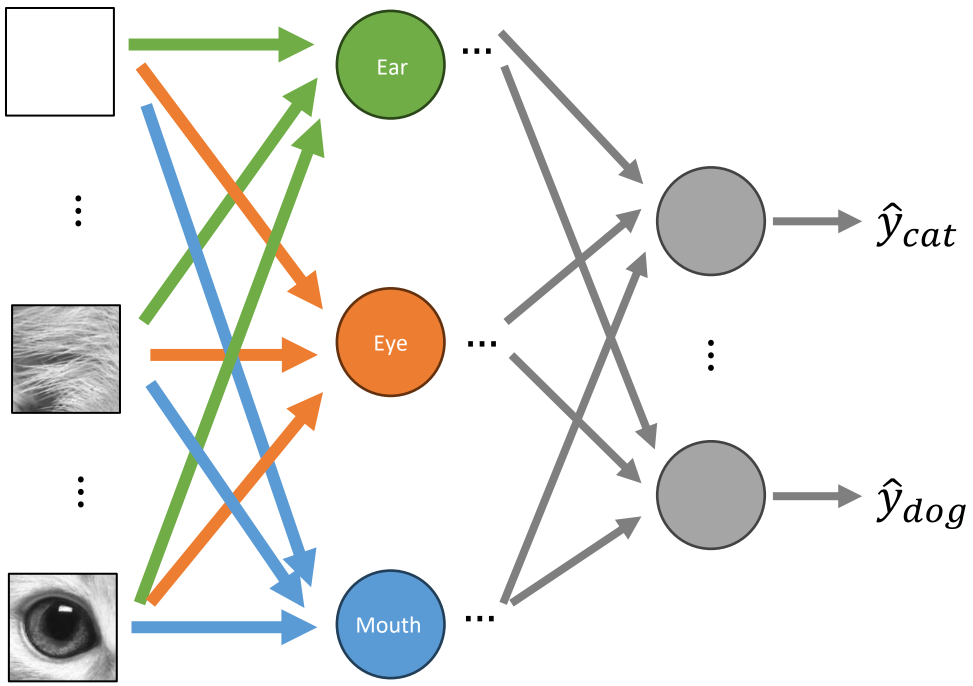


Image has *spatial structures*, which are **preserved**

Input: A **group** of Pixels



Note: same colour → same function

Same image, shifted → **same** set of inputs

How to do this?

Image credit: https://www.deviantart.com/klara1008/art/Cat-png-277228793

Convolution: 2D

$f_{conv}(\boldsymbol{X}) = \boldsymbol{W} * \boldsymbol{X}$

0	1	1	0	0
0	1	1	0	0
1	0	0	1	1
0	1	1	0	1
1	1	1	1	1

Image Input
$\boldsymbol{X}$

$*$

0	-1	0
-1	5	-1
0	-1	0

Kernel / Filter
$\boldsymbol{W}$

$=$

3		

Feature Map

Multiply the sliding input window with kernel then **sum**

Convolution: 2D

$f_{conv}(\boldsymbol{X}) = \boldsymbol{W} * \boldsymbol{X}$

0	1	1	0	0
0	1	1	0	0
1	0	0	1	1
0	1	1	0	1
1	1	1	1	1

Image Input
$\boldsymbol{X}$

$*$

0	-1	0
-1	5	-1
0	-1	0

Kernel / Filter
$\boldsymbol{W}$

$=$

3	3	

Feature Map

Multiply the sliding input window with kernel then **sum**

Convolution: 2D

$f_{conv}(\boldsymbol{X}) = \boldsymbol{W} * \boldsymbol{X}$

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 3 & 3 & -2 \\ & & \\ & & \end{bmatrix}$$

Image Input
$\boldsymbol{X}$

Kernel / Filter
$\boldsymbol{W}$

Feature Map

Multiply the sliding input window with kernel then **sum**

Convolution: 2D

$$f_{conv}(\boldsymbol{X}) = \boldsymbol{W} * \boldsymbol{X}$$

$$\begin{array}{|c|c|c|c|c|}\hline 0 & 1 & 1 & 0 & 0 \\ \hline 0 & 1 & 1 & 0 & 0 \\ \hline 1 & 0 & 0 & 1 & 1 \\ \hline 0 & 1 & 1 & 0 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 \\ \hline \end{array} \; * \; \begin{array}{|c|c|c|}\hline 0 & -1 & 0 \\ \hline -1 & 5 & -1 \\ \hline 0 & -1 & 0 \\ \hline \end{array} \; = \; \begin{array}{|c|c|c|}\hline 3 & 3 & -2 \\ \hline -3 & & \\ \hline & & \\ \hline \end{array}$$

Image Input
$\boldsymbol{X}$

Kernel / Filter
$\boldsymbol{W}$

Feature Map

Multiply the sliding input window with kernel then **sum**

Convolution: 2D

$$f_{conv}(\boldsymbol{X}) = \boldsymbol{W} * \boldsymbol{X}$$

0	1	1	0	0
0	1	1	0	0
1	0	0	1	1
0	1	1	0	1
1	1	1	1	1

Image Input
$\boldsymbol{X}$

$*$

0	-1	0
-1	5	-1
0	-1	0

Kernel / Filter
$\boldsymbol{W}$

$=$

3	3	-2
-3	-3	

Feature Map

Multiply the sliding input window with kernel then **sum**

Convolution: 2D

$f_{conv}(\boldsymbol{X}) = \boldsymbol{W} * \boldsymbol{X}$

0	1	1	0	0
0	1	1	0	0
1	0	0	1	1
0	1	1	0	1
1	1	1	1	1

Image Input
$\boldsymbol{X}$

$*$

0	-1	0
-1	5	-1
0	-1	0

Kernel / Filter
$\boldsymbol{W}$

$=$

3	3	-2
-3	-3	4

Feature Map

Multiply the sliding input window with kernel then sum

Convolution: 2D

$$f_{conv}(\boldsymbol{X}) = \boldsymbol{W} * \boldsymbol{X}$$

0	1	1	0	0
0	1	1	0	0
1	0	0	1	1
0	1	1	0	1
1	1	1	1	1

Image Input
$\boldsymbol{X}$

$*$

0	-1	0
-1	5	-1
0	-1	0

Kernel / Filter
$\boldsymbol{W}$

$=$

3	3	-2
-3	-3	4
3		

Feature Map

Multiply the sliding input window with kernel then **sum**

Convolution: 2D

$$f_{conv}(\boldsymbol{X}) = \boldsymbol{W} * \boldsymbol{X}$$

0	1	1	0	0
0	1	1	0	0
1	0	0	1	1
0	1	1	0	1
1	1	1	1	1

Image Input
$\boldsymbol{X}$

$*$

0	-1	0
-1	5	-1
0	-1	0

Kernel / Filter
$\boldsymbol{W}$

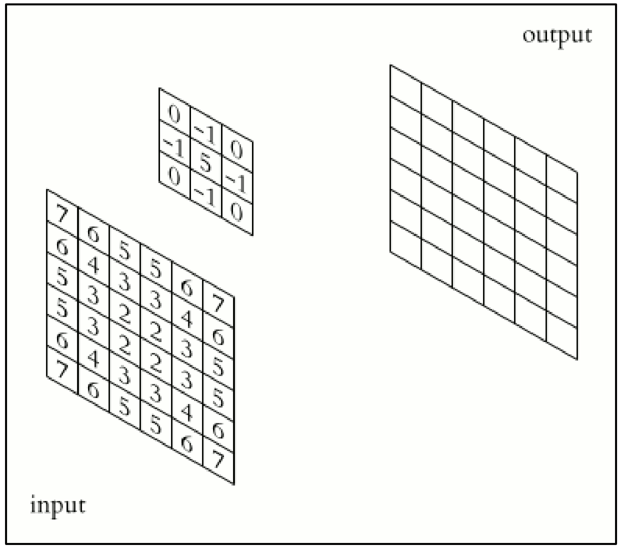
$=$

3	3	-2
-3	-3	4
3	3	

Feature Map

Multiply the sliding input window with kernel then **sum**

Image credit: https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37



Convolution: 2D

$$f_{conv}(\boldsymbol{X}) = \boldsymbol{W} * \boldsymbol{X}$$

0	1	1	0	0
0	1	1	0	0
1	0	0	1	1
0	1	1	0	1
1	1	1	1	1

Image Input
$\boldsymbol{X}$

$*$

0	-1	0
-1	5	-1
0	-1	0

Kernel / Filter
$\boldsymbol{W}$

$=$

3	3	-2
-3	-3	4
3	3	4

Feature Map

What if we want to detect other features (e.g., mouth)?

Multiply the sliding input window with kernel then **sum**

Convolution: 2D

$f_{conv}(\boldsymbol{X}) = \boldsymbol{W} * \boldsymbol{X}$

0	1	1	0	0
0	1	1	0	0
1	0	0	1	1
0	1	1	0	1
1	1	1	1	1

Image Input
$\boldsymbol{X}$

$*$

0	-1	0
-1	5	-1
0	-1	0

Kernel / Filter
$\boldsymbol{W}^{[1]}$

$=$

3	3	-2
-3	-3	4
3	3	4

Feature Map 1

$*$

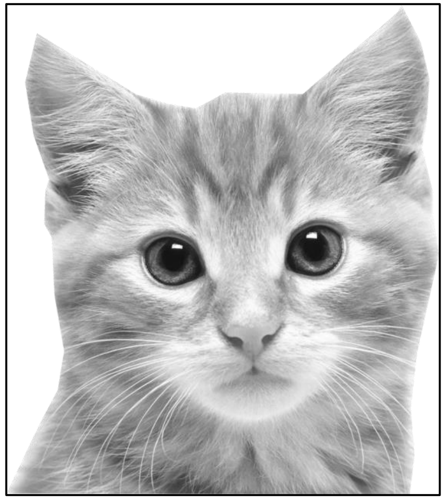
0	1	0
1	0	1
0	1	0

Kernel / Filter
$\boldsymbol{W}^{[2]}$

$=$

Feature Map 2

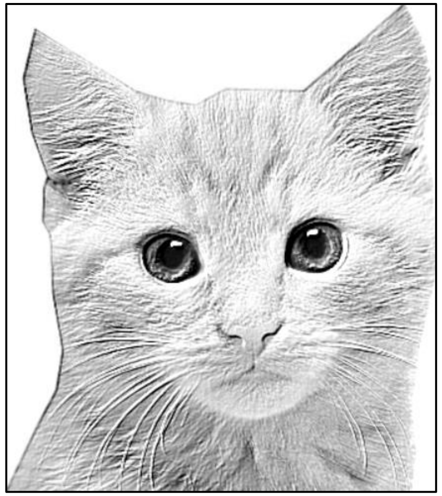
Convolution: 2D Example



Original

Outline

-1	-1	-1
-1	8	-1
-1	-1	-1



Sobel

-2	-1	0
-1	1	1
0	1	2

Blur

.06	.01	.06
.12	.25	.12
.06	.12	.06

Try it yourself: https://setosa.io/ev/image-kernels/

Convolution or Cross-correlation?

$f_{conv}(\boldsymbol{X}) = \boldsymbol{W} * \boldsymbol{X}$

0	1	1	0	0
0	1	1	0	0
1	0	0	1	1
0	1	1	0	1
1	1	1	1	1

Image Input
$\boldsymbol{X}$

$*$

1	2	3
4	5	6
7	8	9

Kernel / Filter
$\boldsymbol{W}$

$=$

Feature Map

This is **cross-correlation**!
Need to flip the kernel

Convolution or Cross-correlation?

$f_{conv}(\boldsymbol{X}) = \boldsymbol{W} * \boldsymbol{X}$

0	1	1	0	0
0	1	1	0	0
1	0	0	1	1
0	1	1	0	1
1	1	1	1	1

Image Input
$\boldsymbol{X}$

$*$

9	8	7
6	5	4
3	2	1

Kernel / Filter
$flip(\boldsymbol{W})$

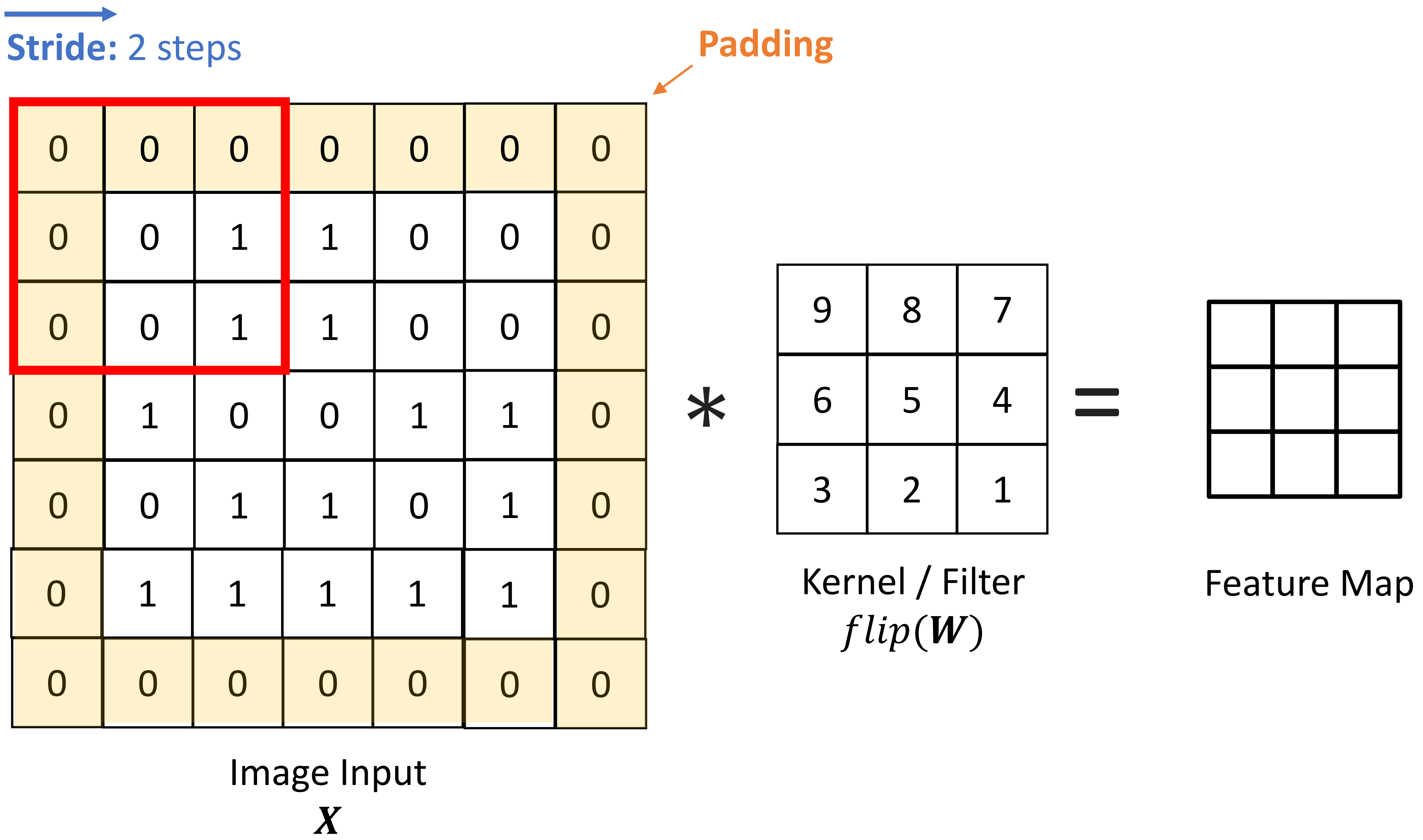
$=$

Feature Map

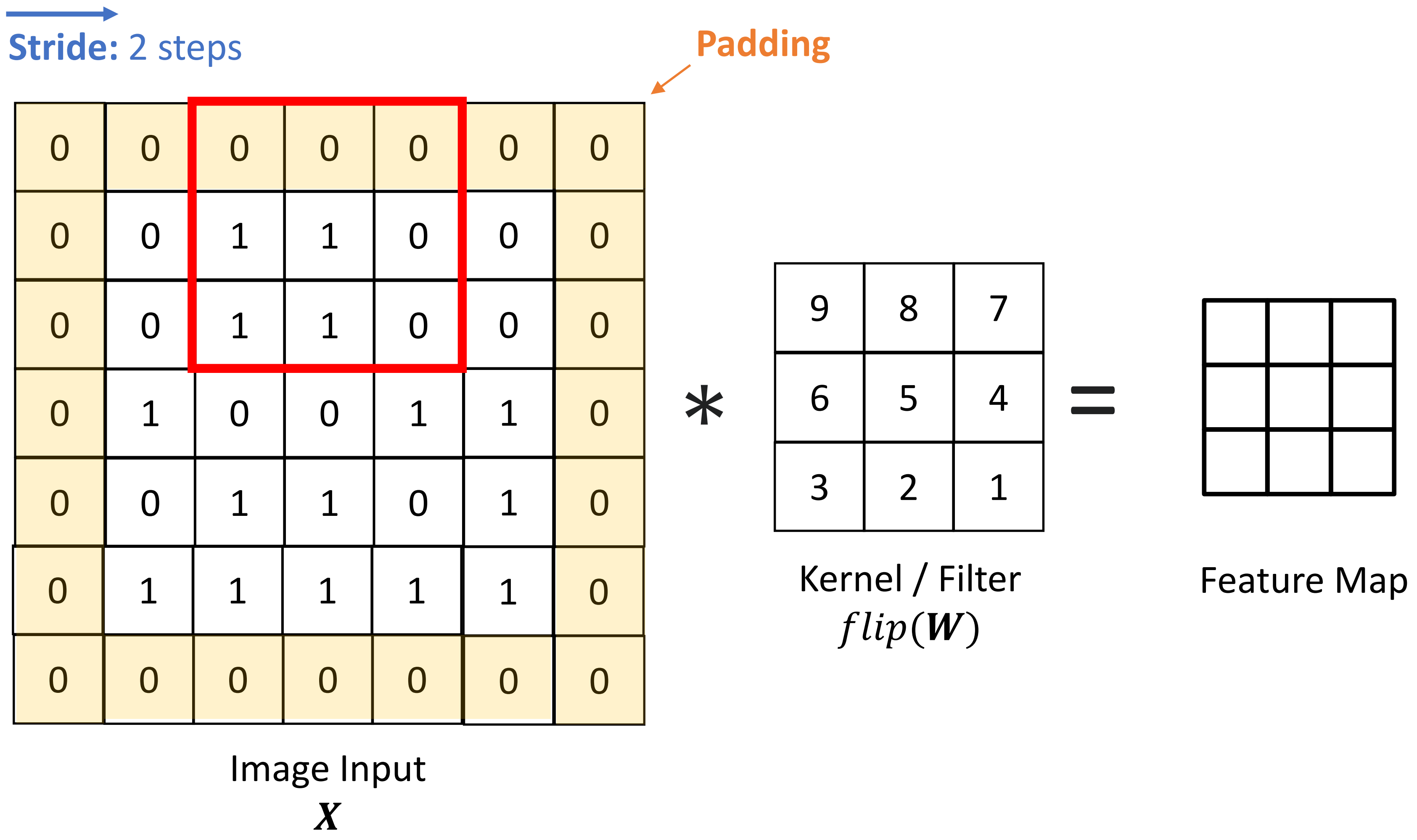
This is now **convolution**!

Does this matter?

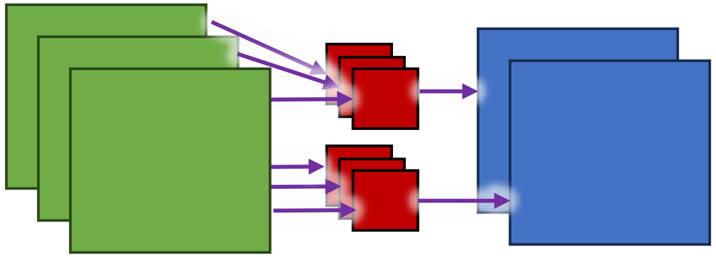
Convolution: Common Practice



Convolution: Common Practice



Convolution Layer vs Feedforward Layer



Input:
3 Channels

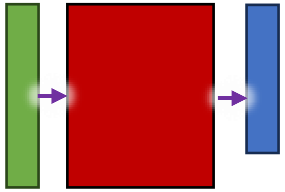
Output:
2 Channels

Convolution Layer

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

Weights = **Kernels**
(4D tensor)

Input and Output (3D Matrix)
Concatenation of **feature maps**



Feedforward (Linear) Layer

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

Weights
(2D matrix)

Pooling Layer

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

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

2x2 Max-Pool
Stride = 2

20	30
112	37

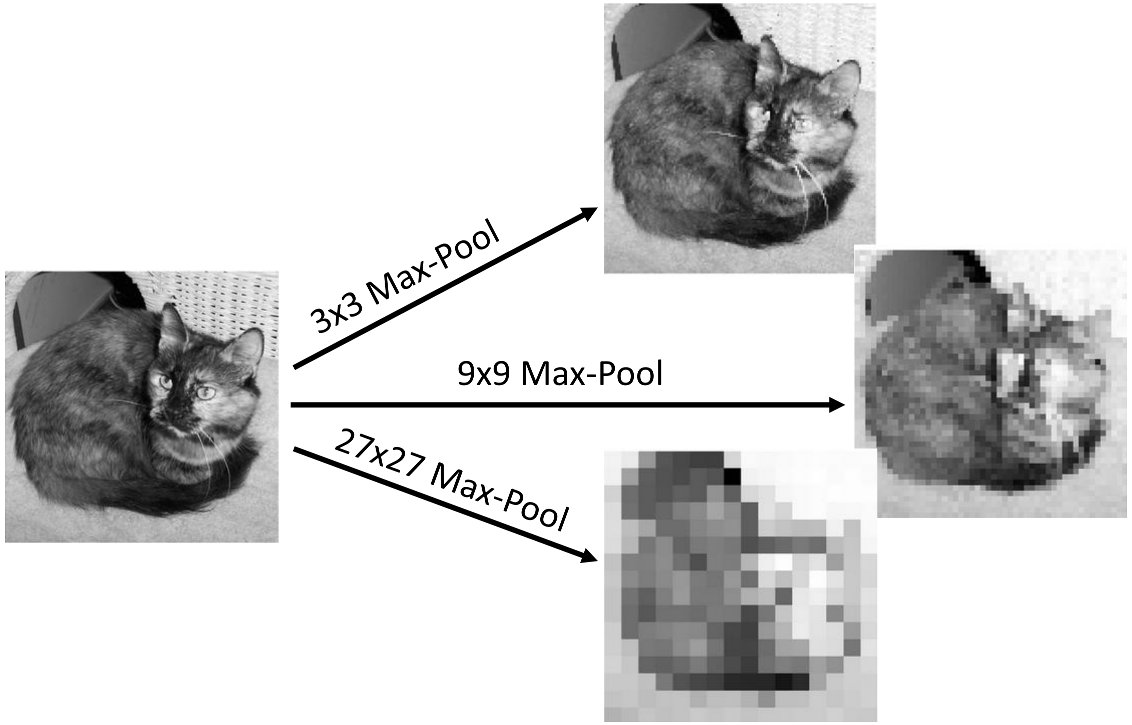


Image credit: https://blog.paperspace.com/pooling-in-convolutional-neural-networks/

Typical

Convolutional Neural Networks (CNN)

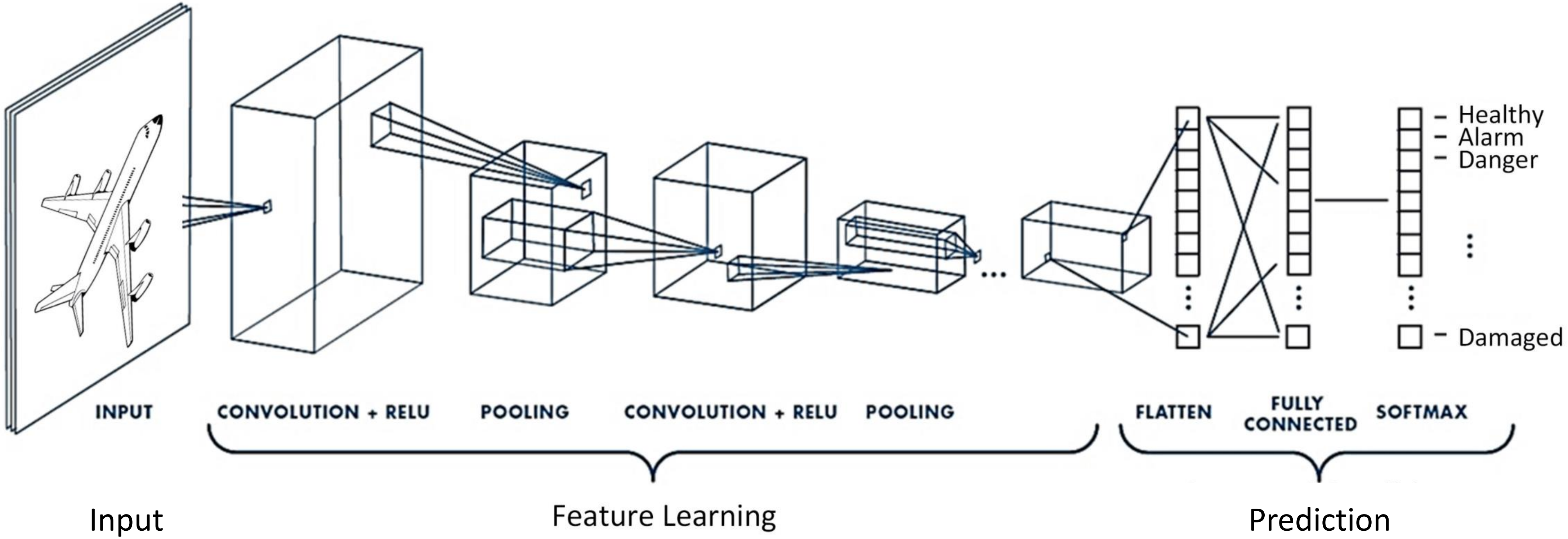
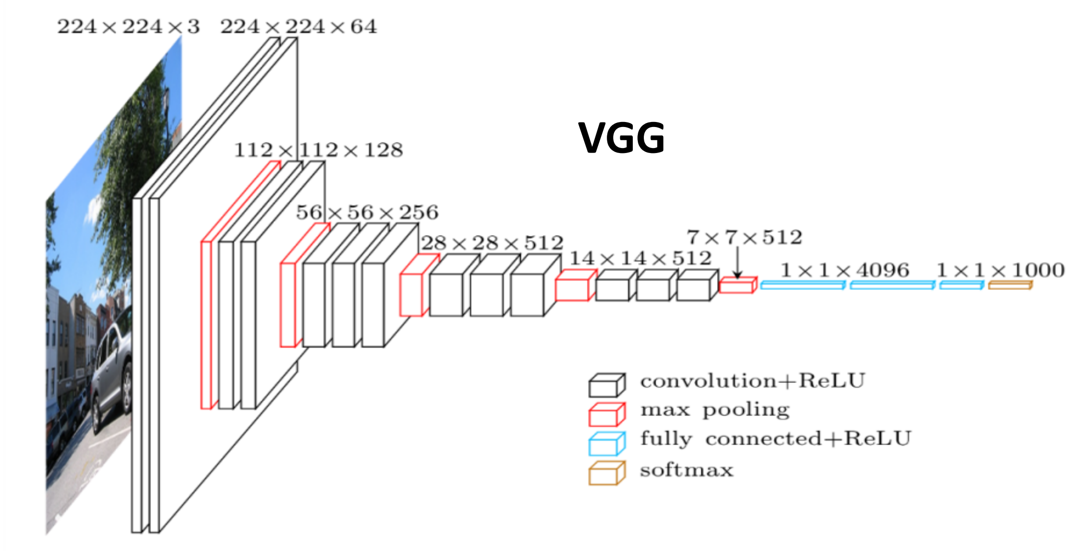


Image credit: https://www.mdpi.com/1424-8220/19/22/4933

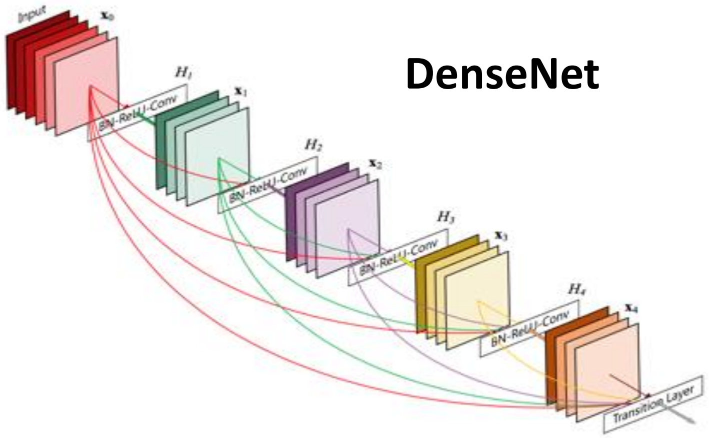
Popular CNN Architectures

VGG

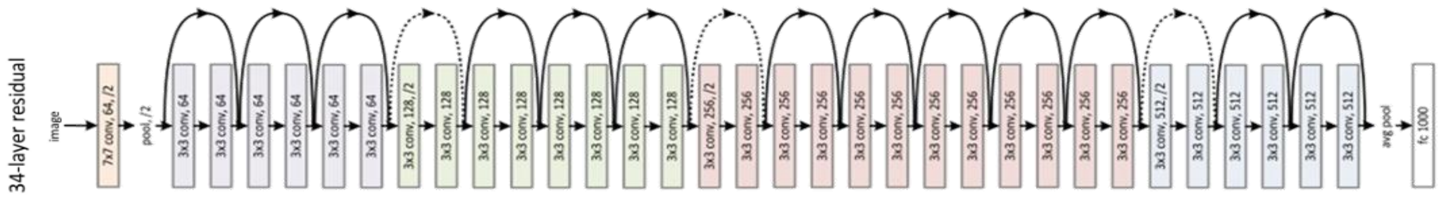


Inception

DenseNet



ResNet



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

Applications of CNN

Image Classification
e.g., face emotions

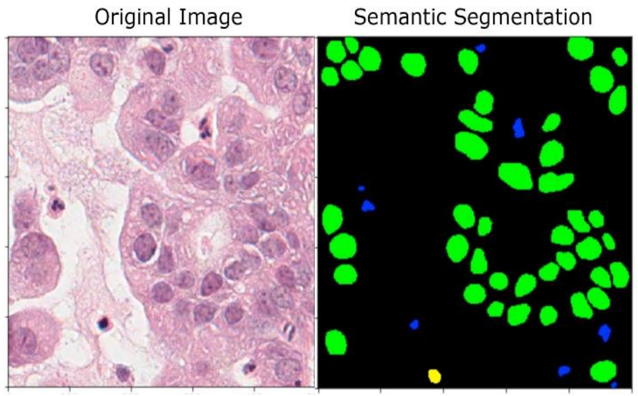
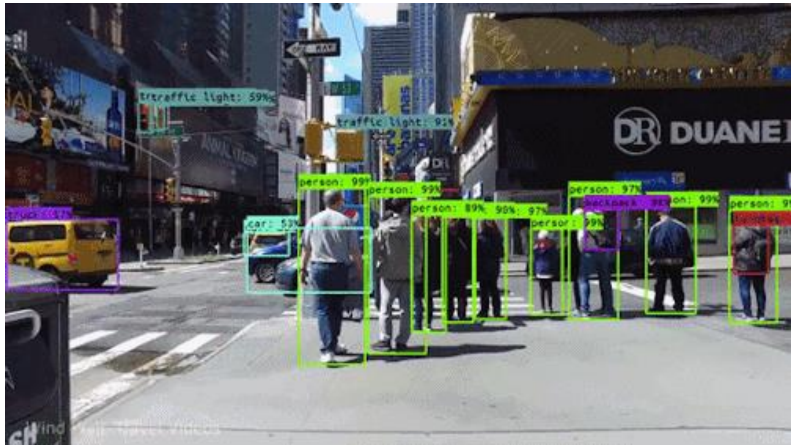


Image Segmentation
e.g., cancer cell detection



Object Detection
e.g., self-driving cars

Image Credit:
https://analyticsindiamag.com/face-emotion-recognizer-in-6-lines-of-code/
https://medium.com/@albertlai631/how-do-self-driving-cars-see-13054aee2503
https://ajp.amjpathol.org/article/S0002-9440(18)31121-0/fulltext

Outline

- Deep Neural Networks
- Convolution Neural Networks
  - Motivation: handling spatial structure
  - Convolution, Pooling Layer, and Common Architectures
  - Applications
- **Recurrent Neural Networks**
  - Motivation: handling sequential data
  - Recurrent Neural Networks and Variants
  - Applications
- Attention, Transformers, GPT, and ChatGPT (if time permits)
- Issues with Deep Learning

Sequential Data

Sentiment Analysis

“Indomie is the best noodles ever”

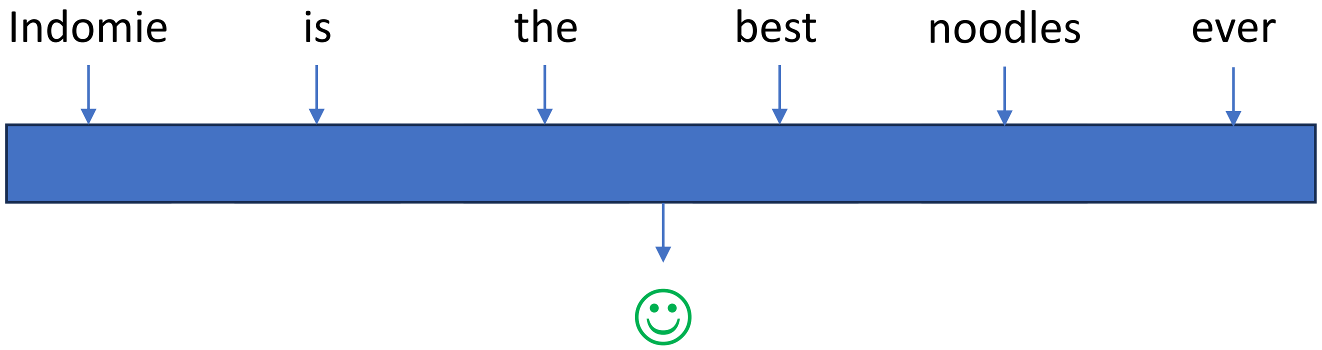
Credit: Indomie

Sequential Data: 1st Attempt

Credit: Indomie

Sentiment Analysis

“Indomie is the best noodles ever”

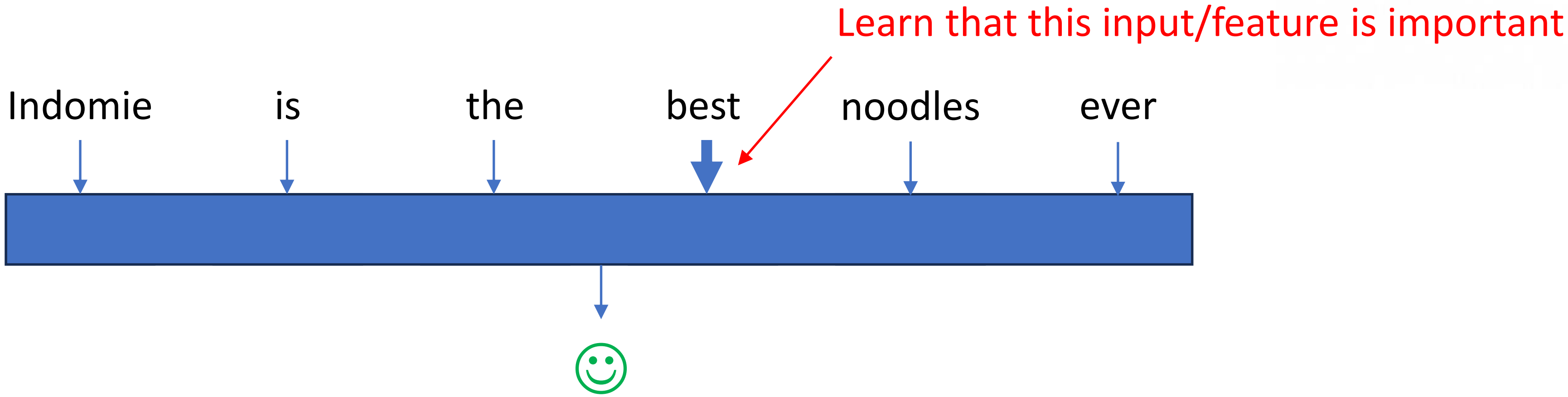


Sequential Data: 1st Attempt

Credit: Indomie

Sentiment Analysis

“Indomie is the best noodles ever”

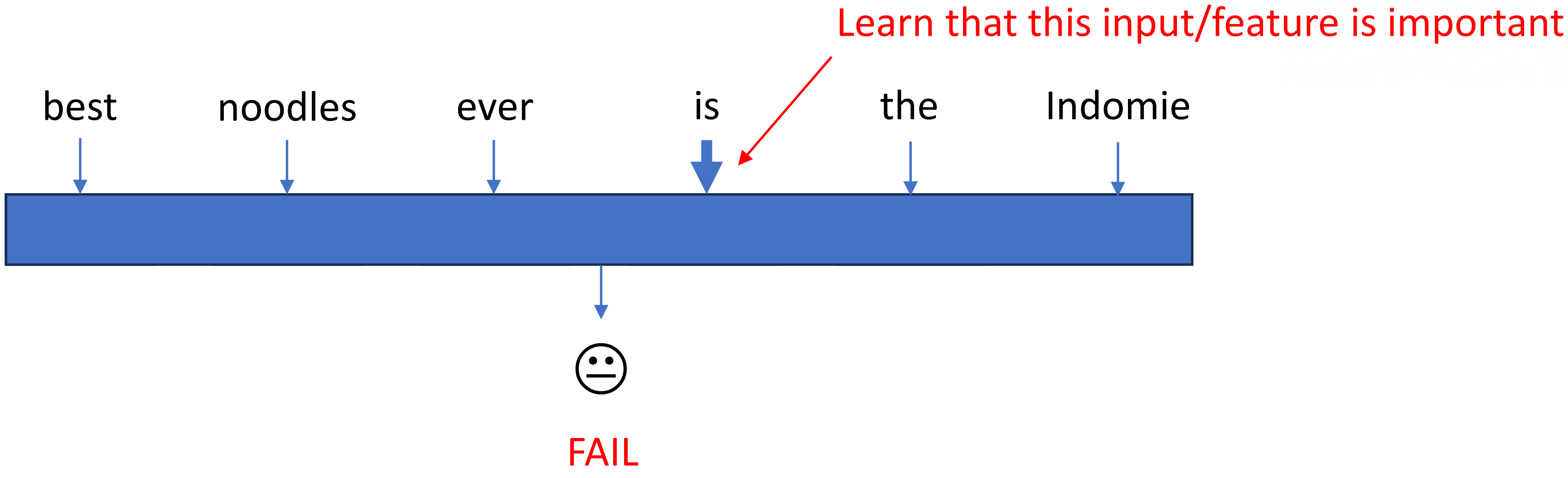


Sequential Data: 1st Attempt

Credit: Indomie

Sentiment Analysis

“Indomie is the best noodles ever”

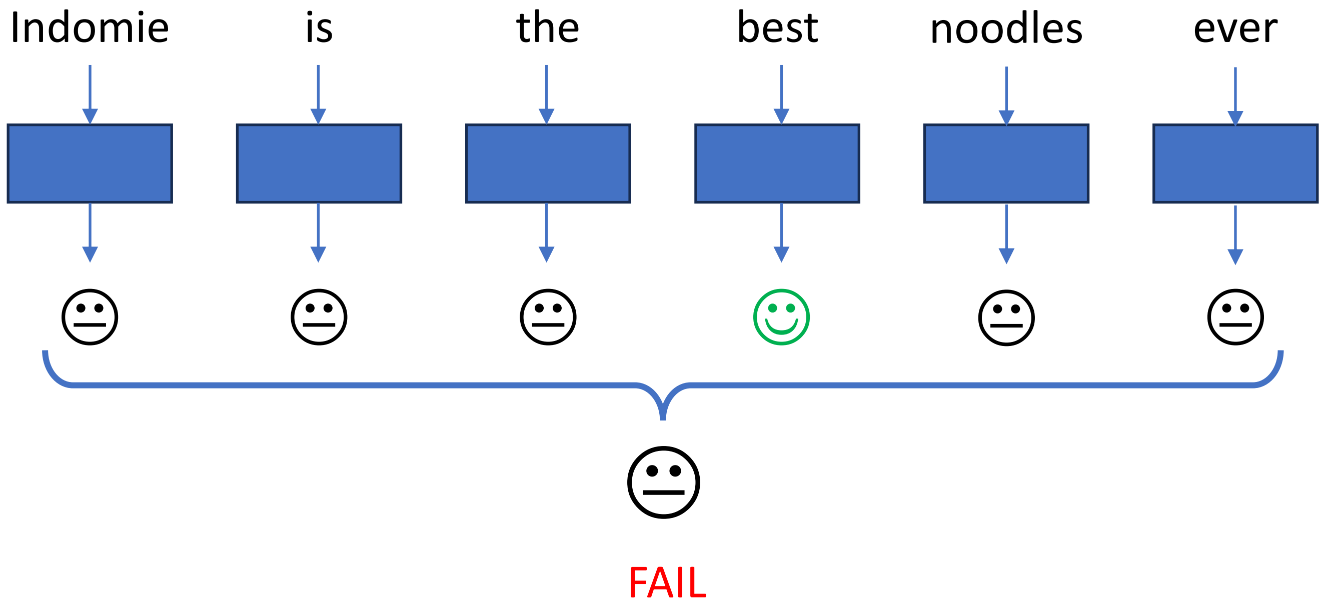


Credit: Indomie

Sequential Data: 2nd Attempt

Sentiment Analysis

“Indomie is the best noodles ever”

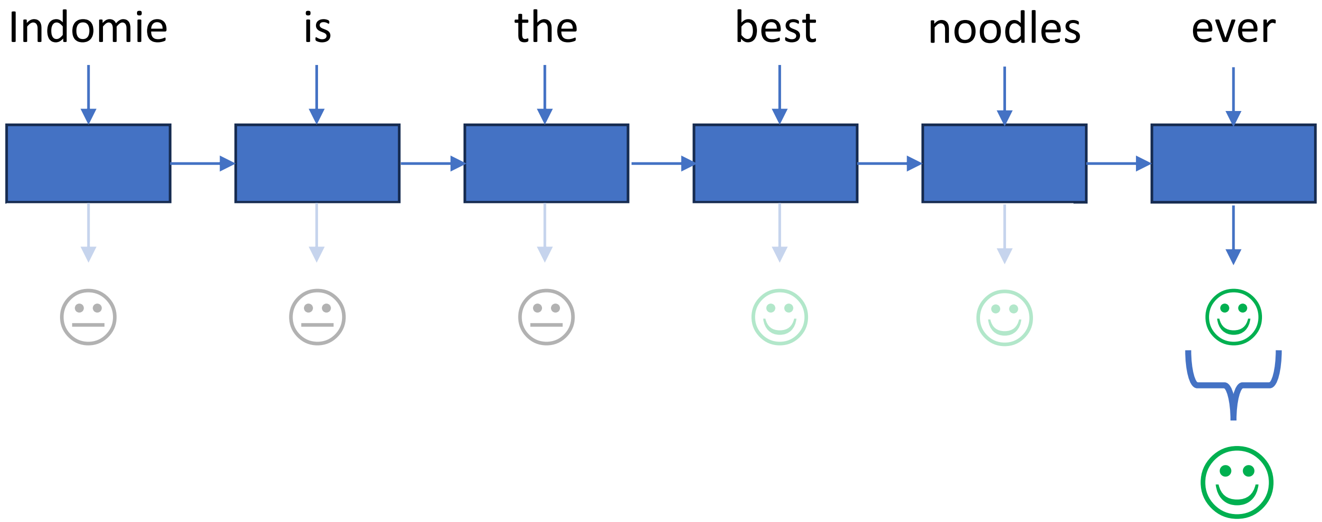


Credit: Indomie

Sequential Data: 3rd Attempt

Sentiment Analysis

"Indomie is the best noodles ever"

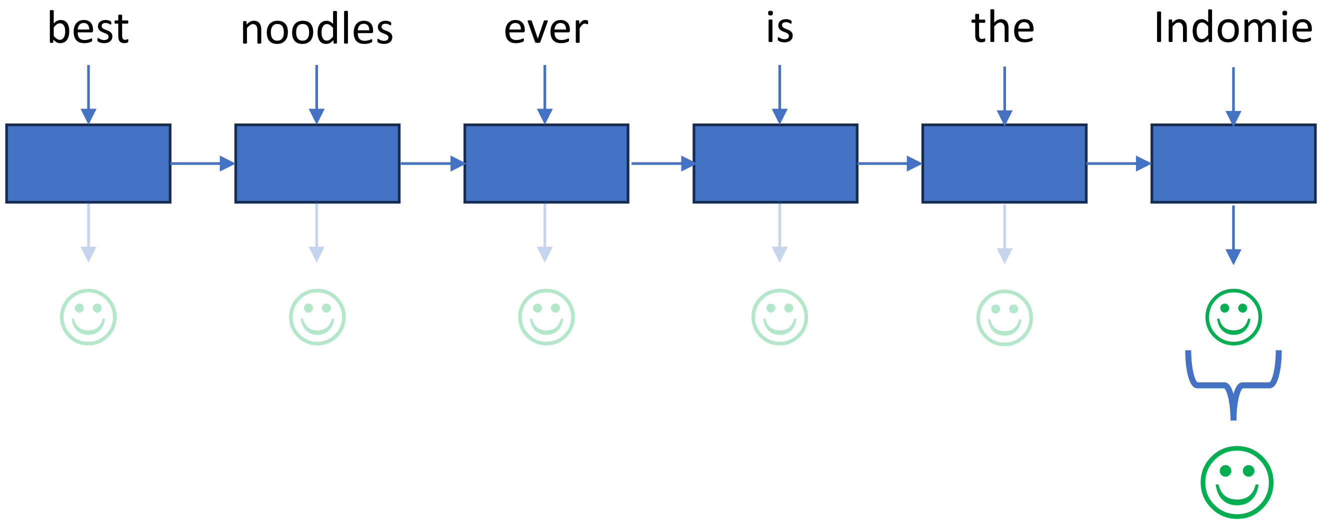


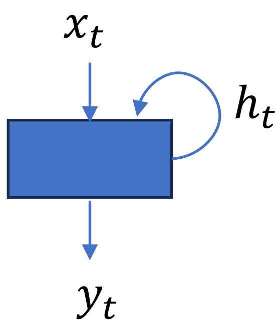
Credit: Indomie

Sequential Data: 3rd Attempt

Sentiment Analysis

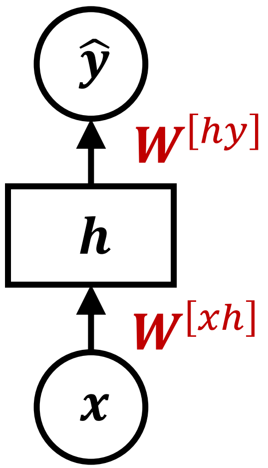
“Indomie is the best noodles ever”





Recurrent Neural Networks (RNN)

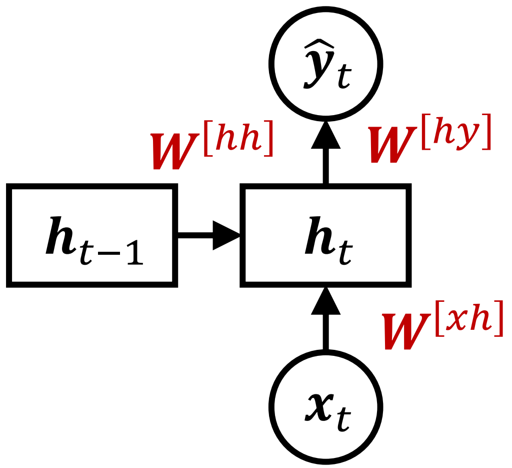
Recurrent Neural Networks (RNN)



Feed-forward networks

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

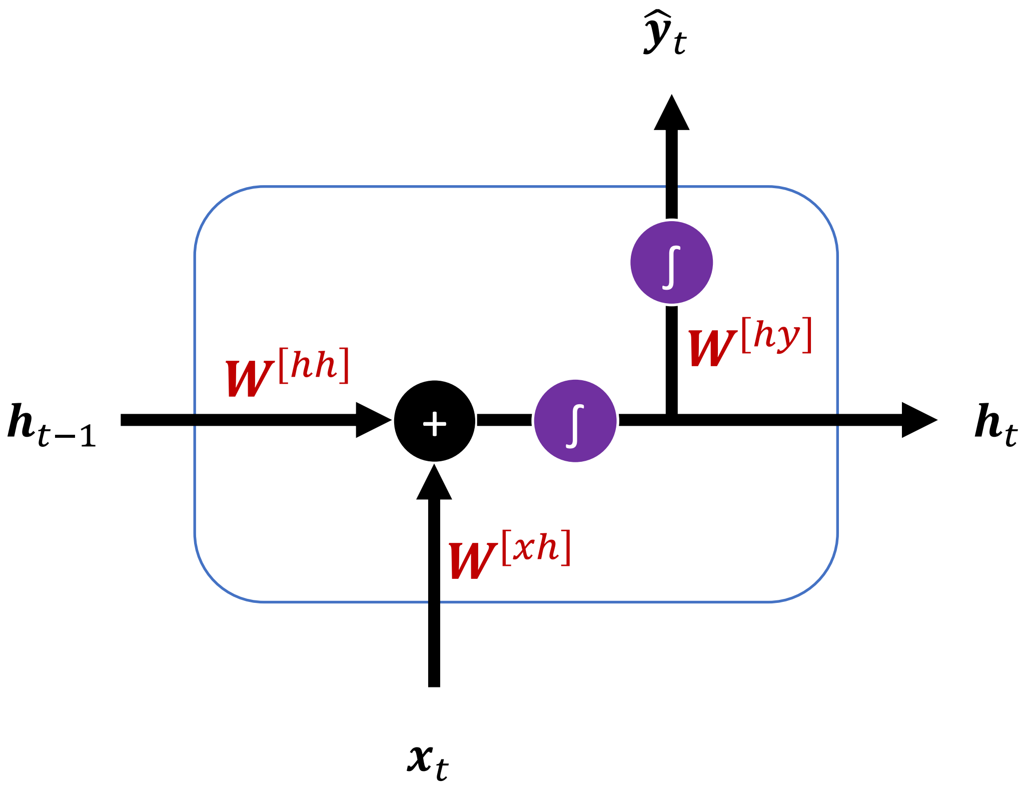
$$h = g^{[h]}\left(\left(W^{[xh]}\right)^{\top} x\right)$$



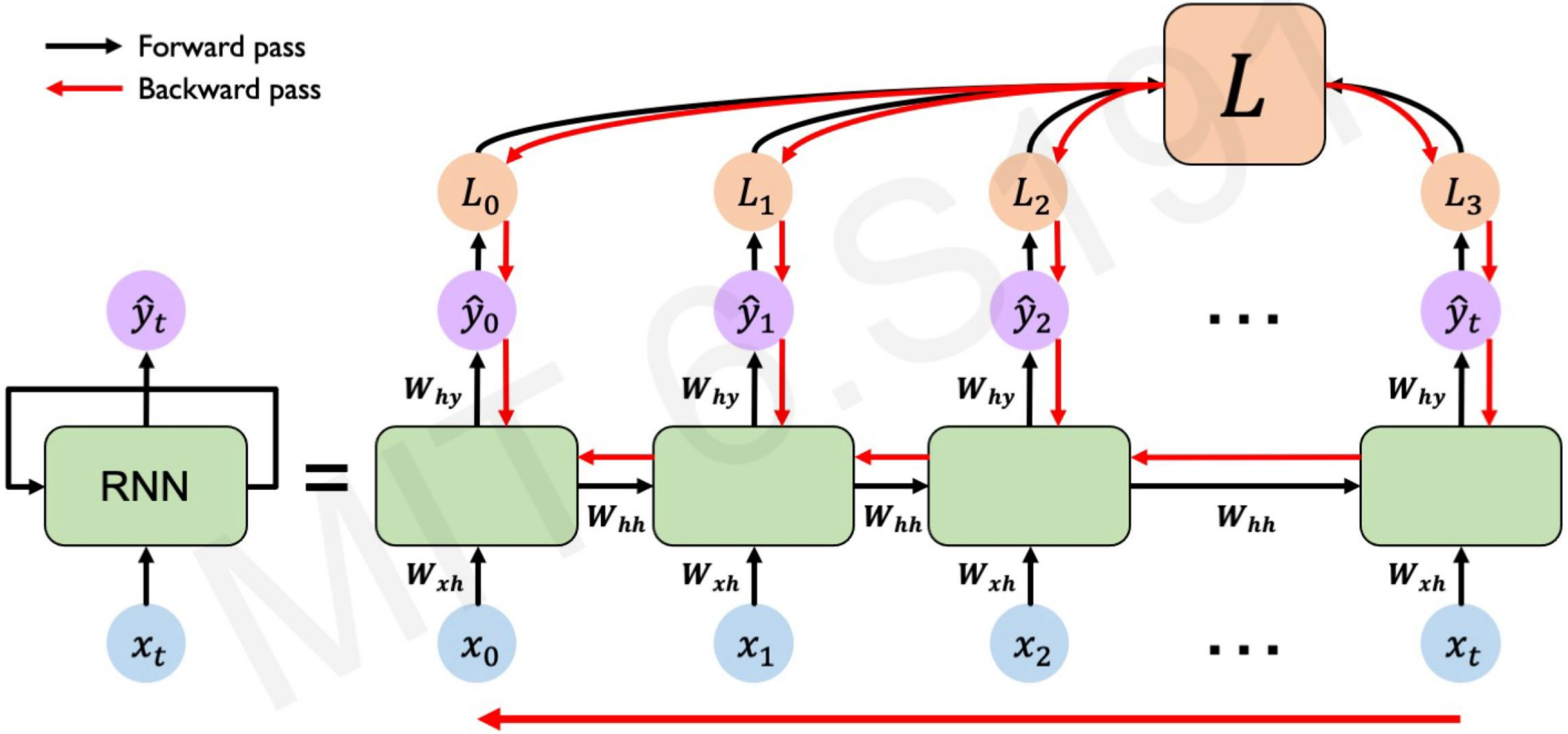
Recurrent Neural Networks

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

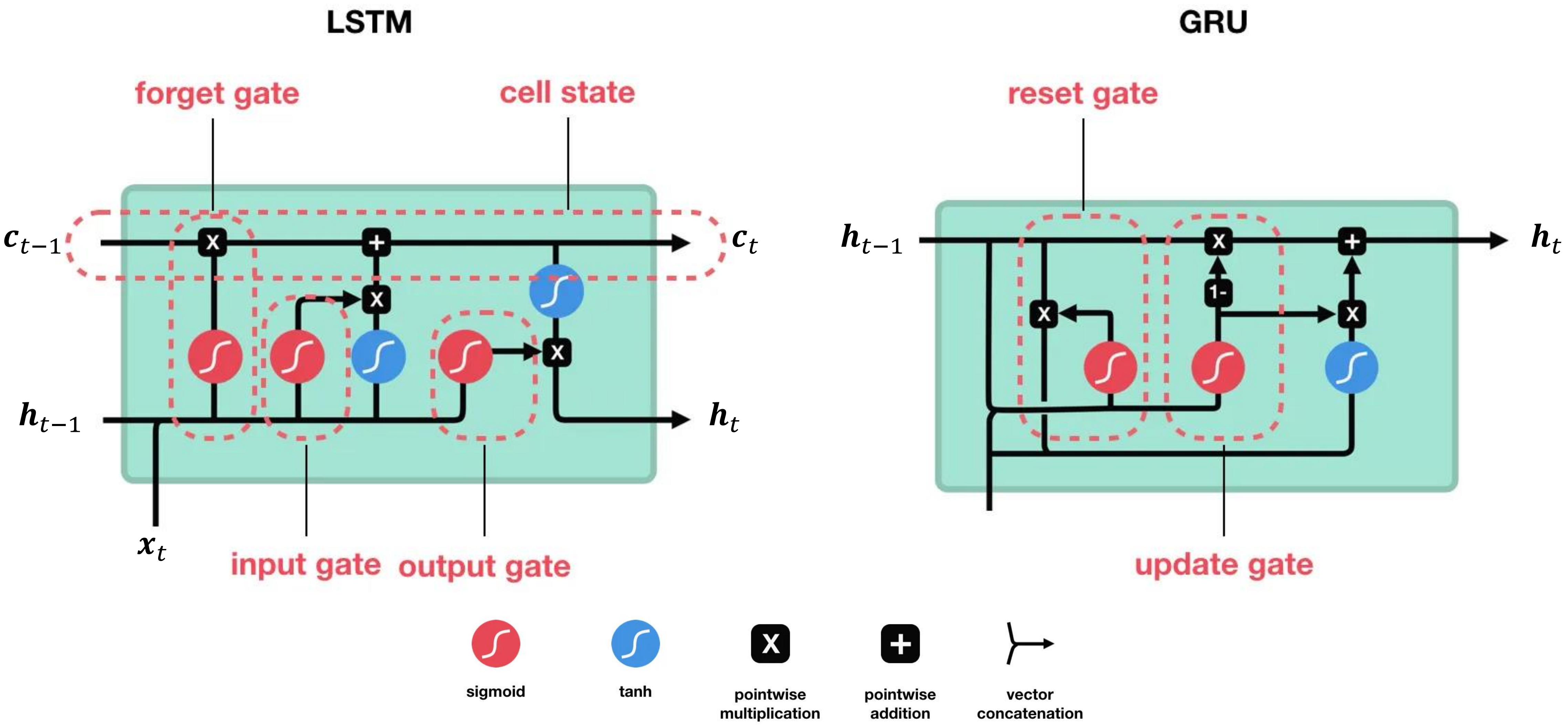
$$h_t = g^{[h]}\left(\left(W^{[xh]}\right)^{\top} x_t + \left(W^{[hh]}\right)^{\top} h_{t-1}\right)$$



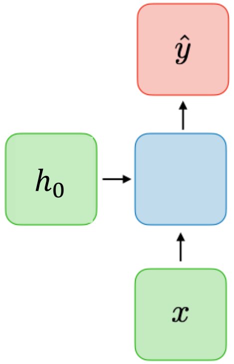
Backpropagation Through Time (BPTT)



Long Short-Term Memory (LSTM) & Gated Recurrent Unit (GRU)



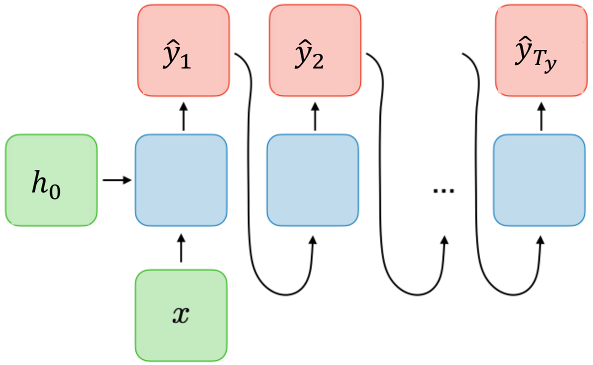
Sequence Modelling



One-to-one

$T_x = T_y = 1$

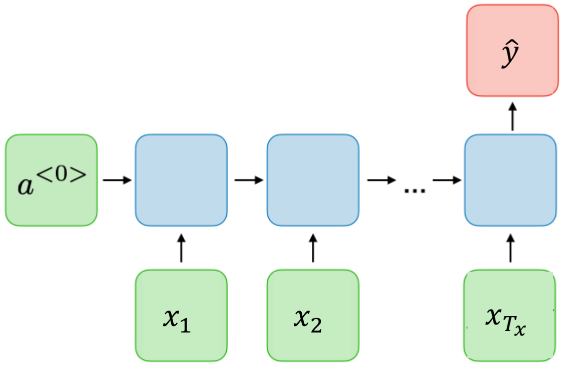
Feedforward Network



One-to-many

$T_x = 1, T_y > 1$

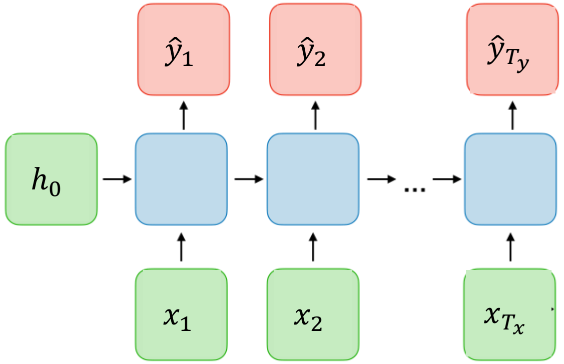
Image captioning, text generation



Many-to-one

$T_x > 1, T_y = 1$

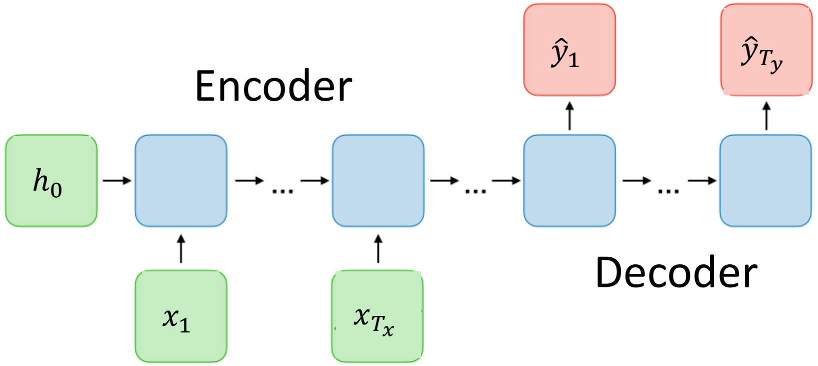
Sentiment classification



Many-to-many

$T_x > 1, T_y > 1$

Name entity recognition

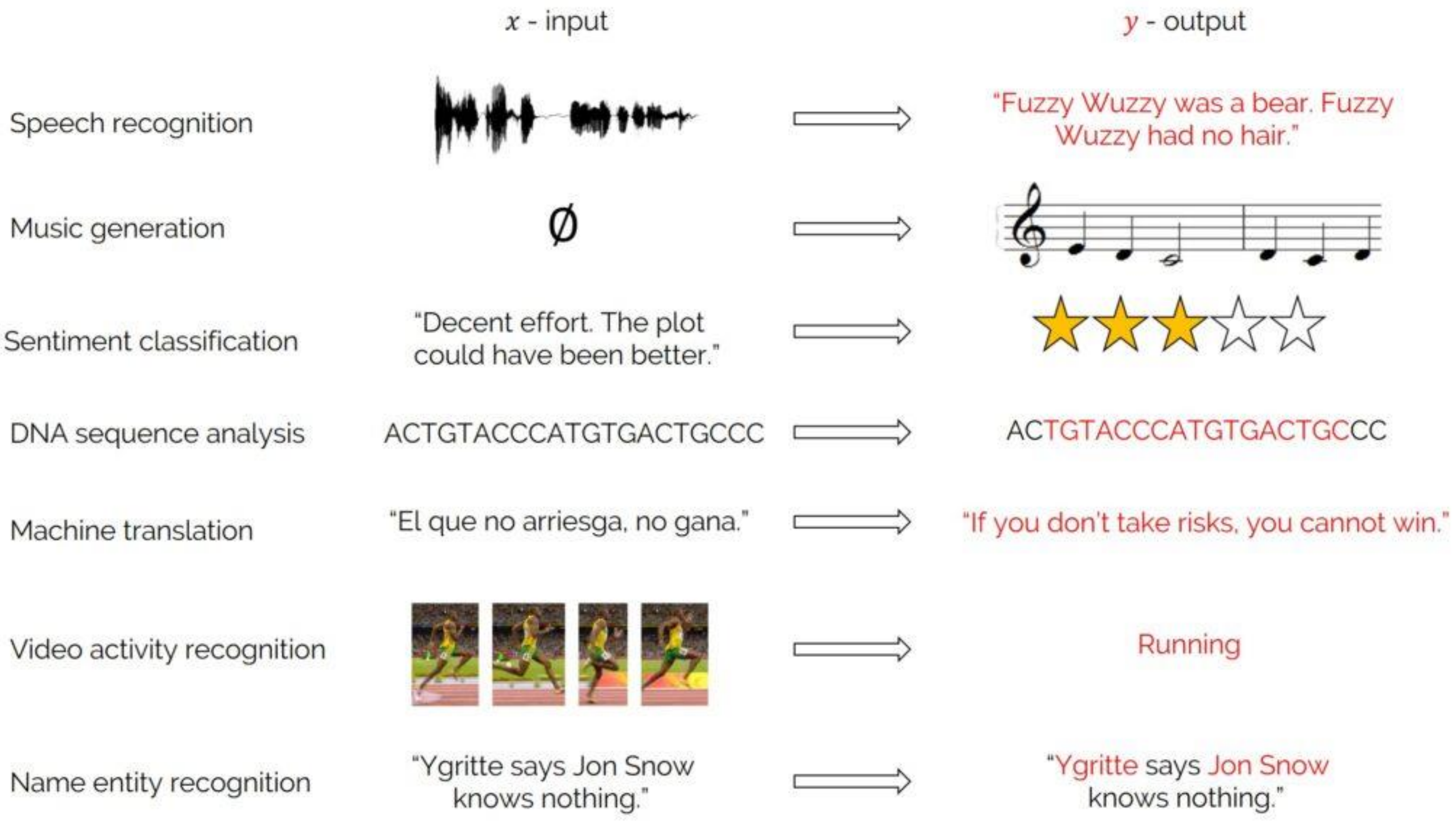


Many-to-many

$T_x \neq T_y$

Machine translation

Applications of RNN



Source: https://datahacker.rs/003-rnn-architectural-types-of-different-recurrent-neural-networks/

Outline

- Deep Neural Networks
- Convolution Neural Networks
  - Motivation: handling spatial structure
  - Convolution, Pooling Layer, and Common Architectures
  - Applications
- Recurrent Neural Networks
  - Motivation: handling sequential data
  - Recurrent Neural Networks and Variants
  - Applications
- **Attention, Transformers, GPT, and ChatGPT (if time permits)**
- Issues with Deep Learning

Credit: Indomie

Long term dependencies problem

Machine Translation (English → Japanese)

“Indomie is the best noodles ever created by humankind. … <100 words later> … This concludes my argument why I like the noodles.”

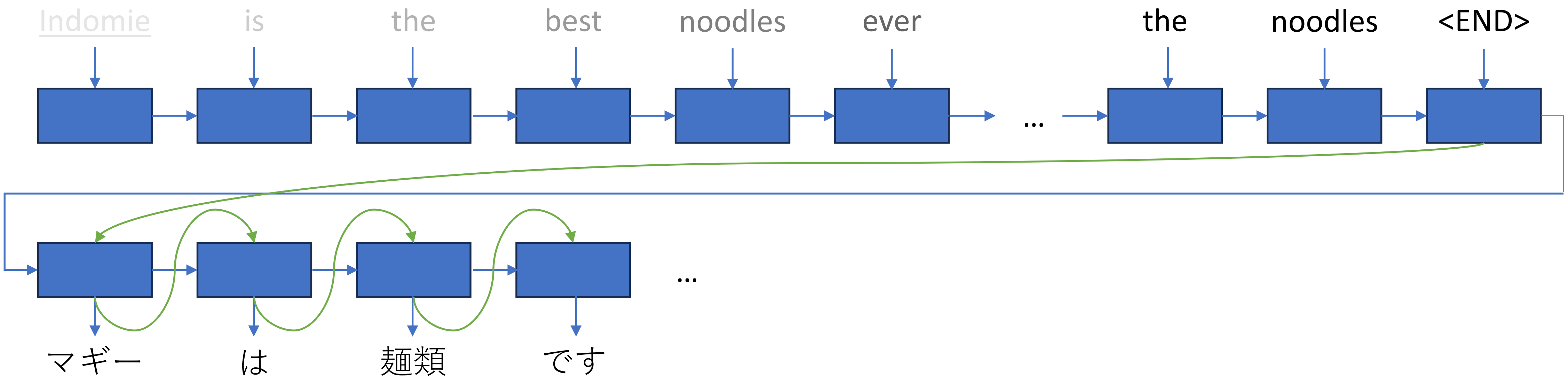
Long term dependencies problem

Credit: Indomie

Machine Translation (English → Japanese)

“Indomie is the best noodles ever created by humankind. … <100 words later> … This concludes my argument why I like the noodles.”

Forgot about what it has read before!



“Maggi is noodles…”

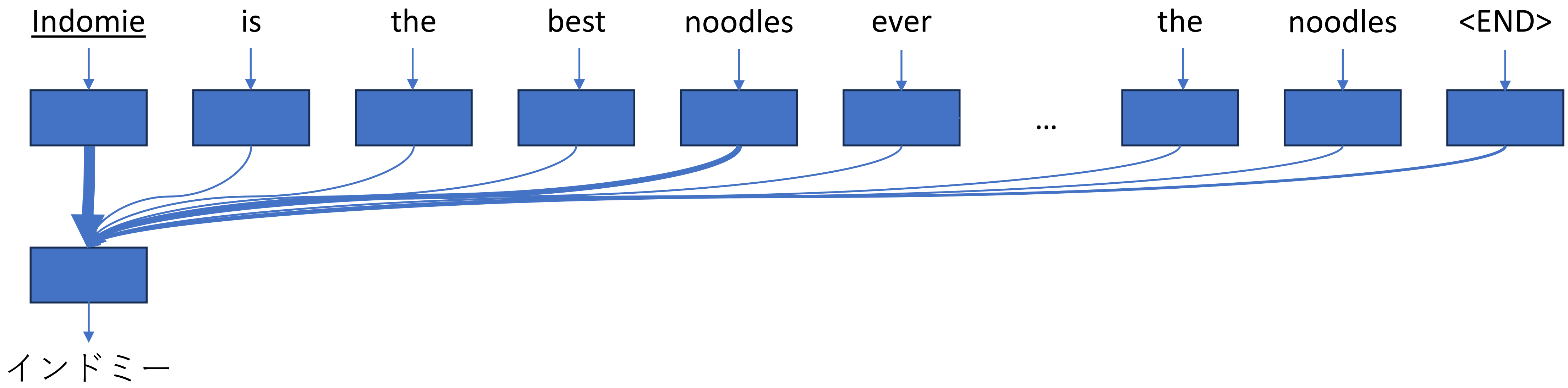
Credit: Indomie

Long term dependencies problem

Machine Translation (English → Japanese)

“Indomie is the best noodles ever created by humankind. … <100 words later> … This concludes my argument why I like the noodles.”

Attends to (focus on) what’s necessary!



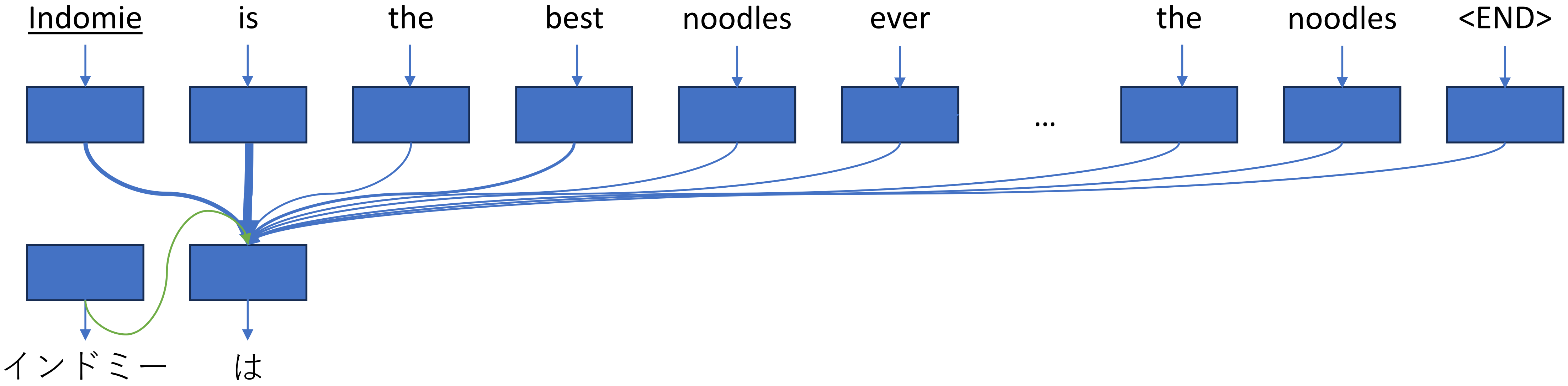
Credit: Indomie

Long term dependencies problem

Machine Translation (English → Japanese)

“Indomie is the best noodles ever created by humankind. … <100 words later> … This conclude my argument why I like the noodles.”

Attends to (focus on) what’s necessary!

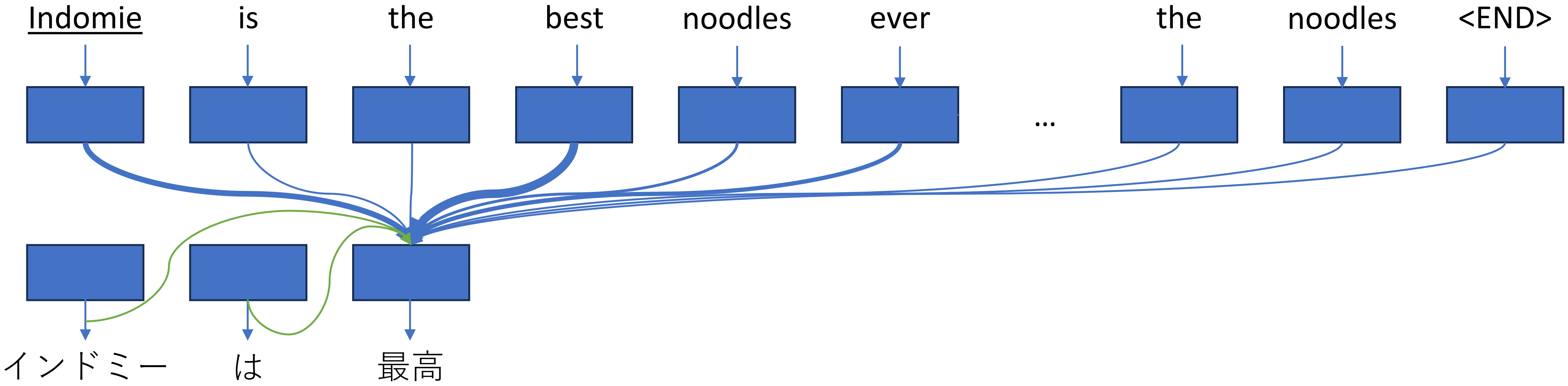


Long term dependencies problem

Machine Translation (English → Japanese)

"Indomie is the best noodles ever created by humankind. … <100 words later> … This conclude my argument why I like the noodles."

Attends to (focus on) what's necessary!



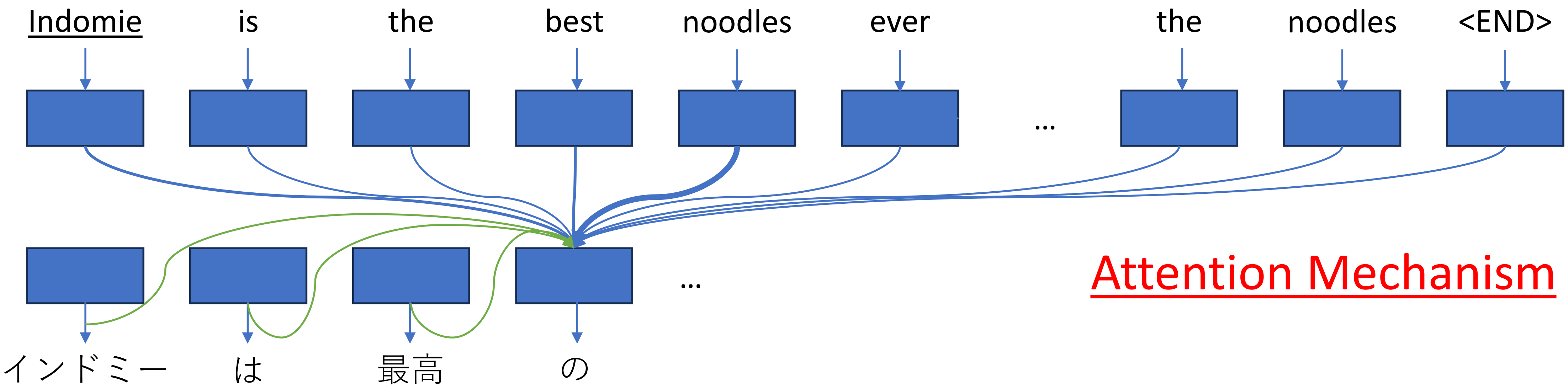
Long term dependencies problem

Credit: Indomie

Machine Translation (English → Japanese)

“Indomie is the best noodles ever created by humankind. … <100 words later> … This concludes my argument why I like the noodles.”

Attends to (focus on) what’s necessary!



Attention Mechanism

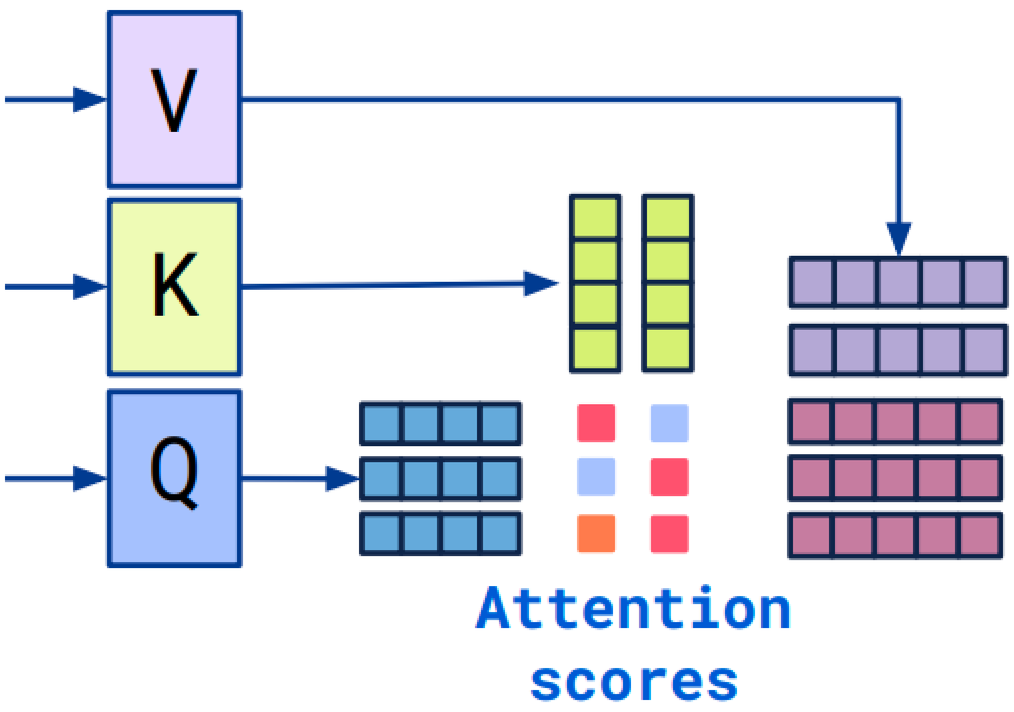
“Indomie is the best …”

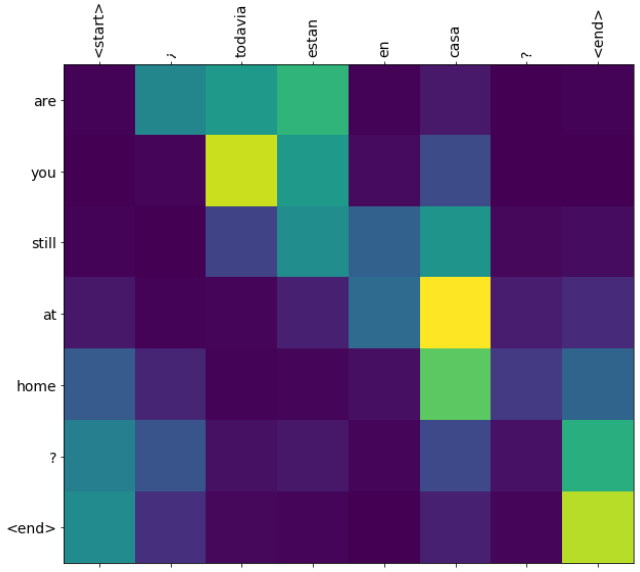
Image credit: https://vaclavkosar.com/ml/cross-attention-in-transformer-architecture

Attention

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

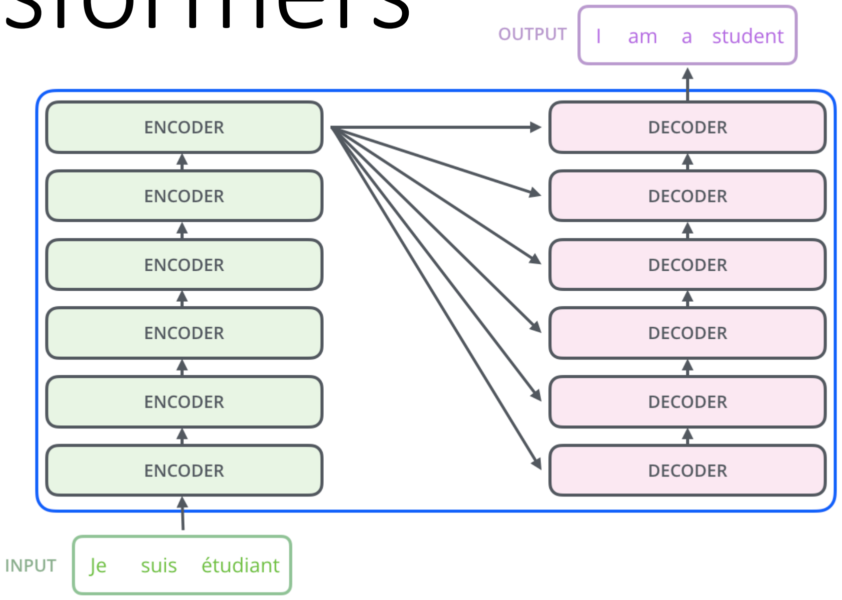
Query Key Value





Source: https://jalammar.github.io/illustrated-transformer/

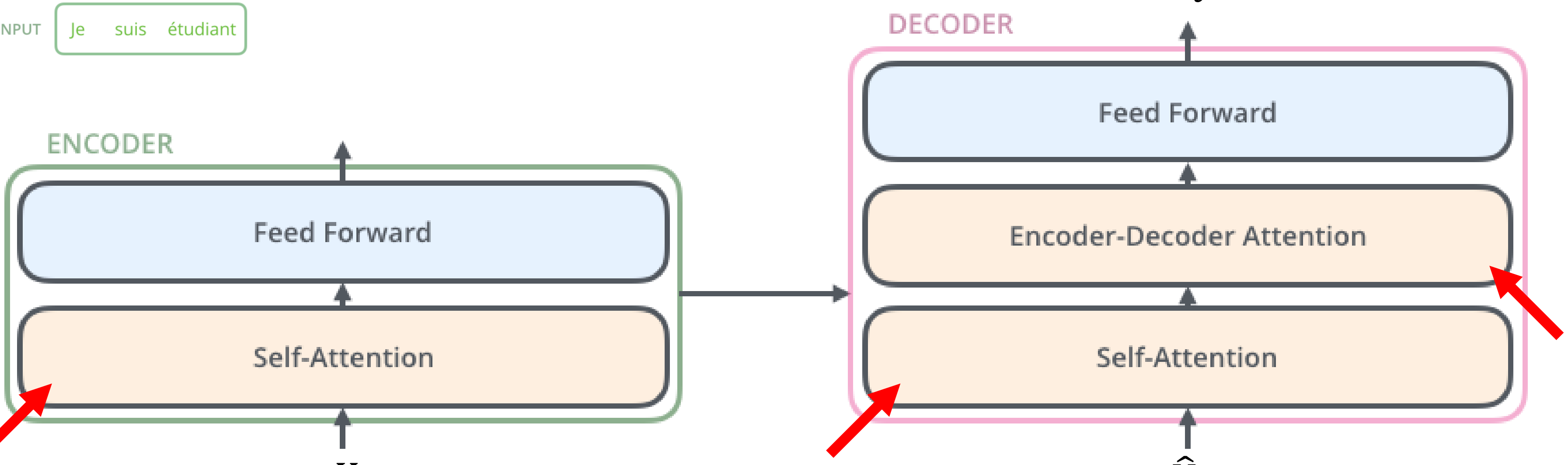
Transformers



$$\boxed{Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V}$$

インドミーは最高**の麺**

$\hat{Y}_t$



$Attention(\hat{Y}_{t-1}, X, X)$

$Attention(X,X,X)$

X

$Attention(\hat{Y}_{t-1},\hat{Y}_{t-1},\hat{Y}_{t-1})$

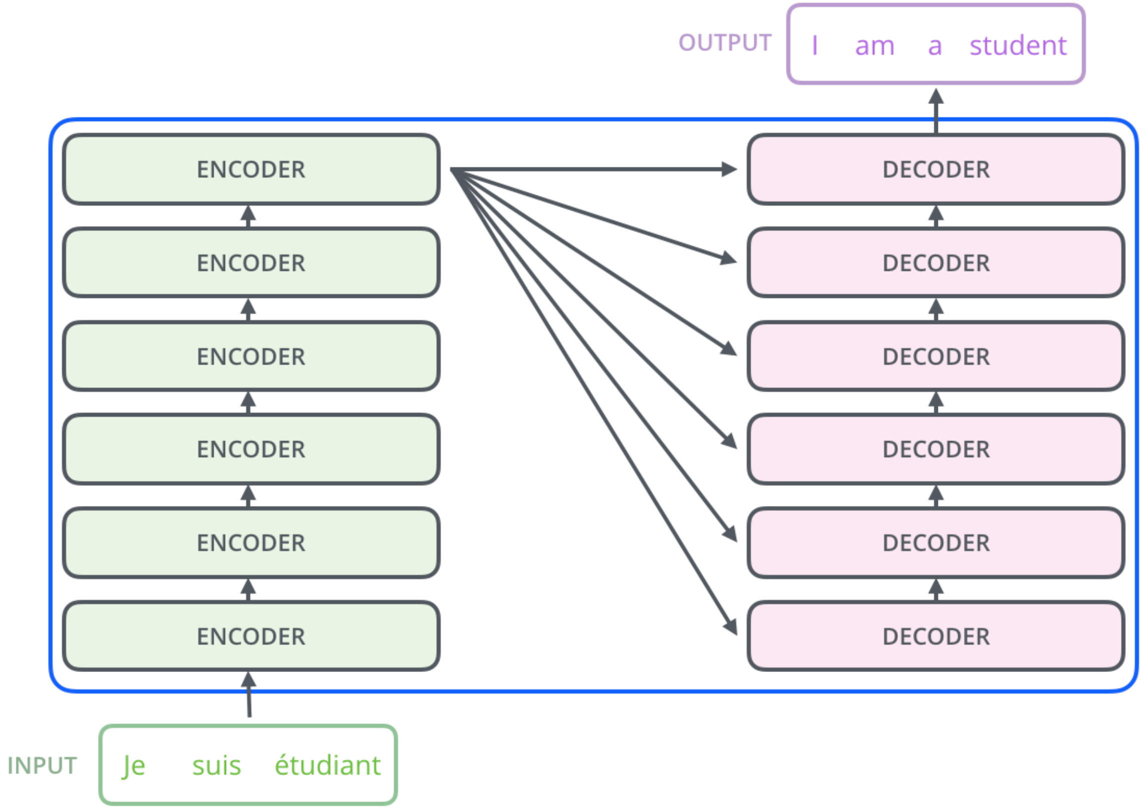
$\hat{Y}_{t-1}$

Indomie is the best noodles

インドミー は 最高

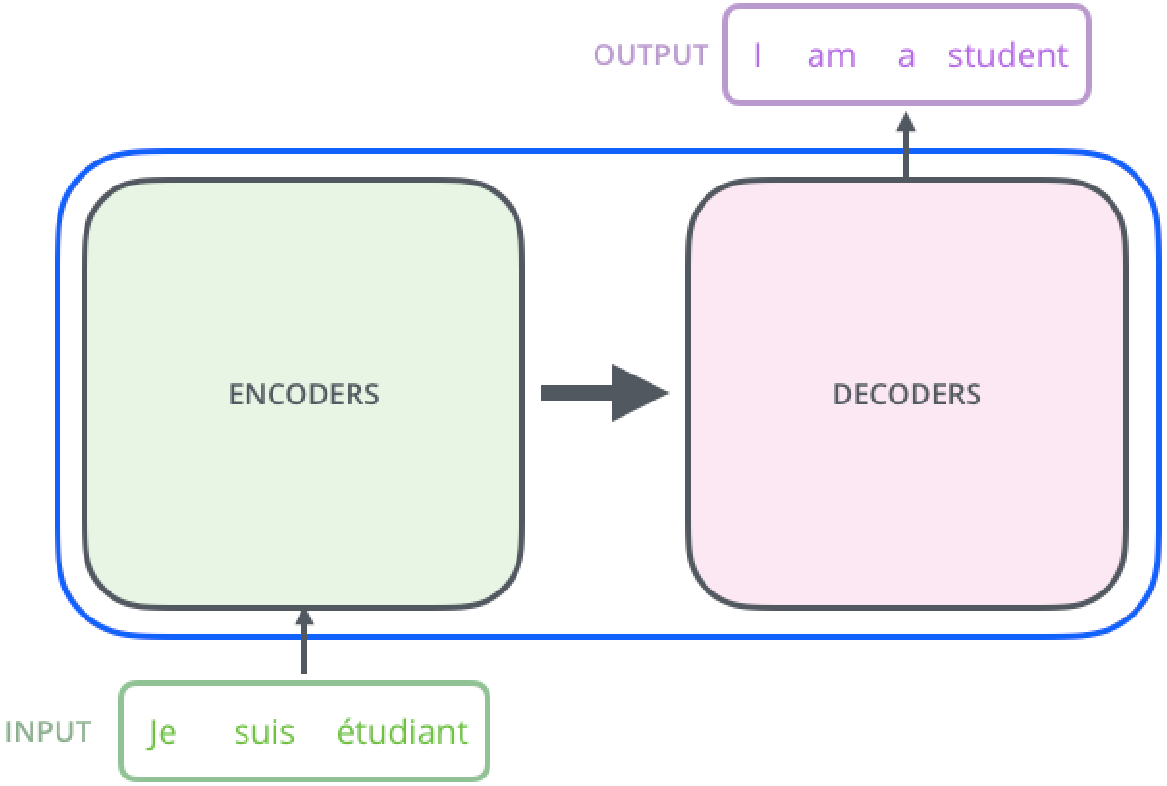
Source: https://jalammar.github.io/illustrated-transformer/

Transformers



Source: https://jalammar.github.io/illustrated-transformer/

Transformers



Generative Pretrained Transformers (GPT)

Indomie is the best noodles ever created by _

Transformer

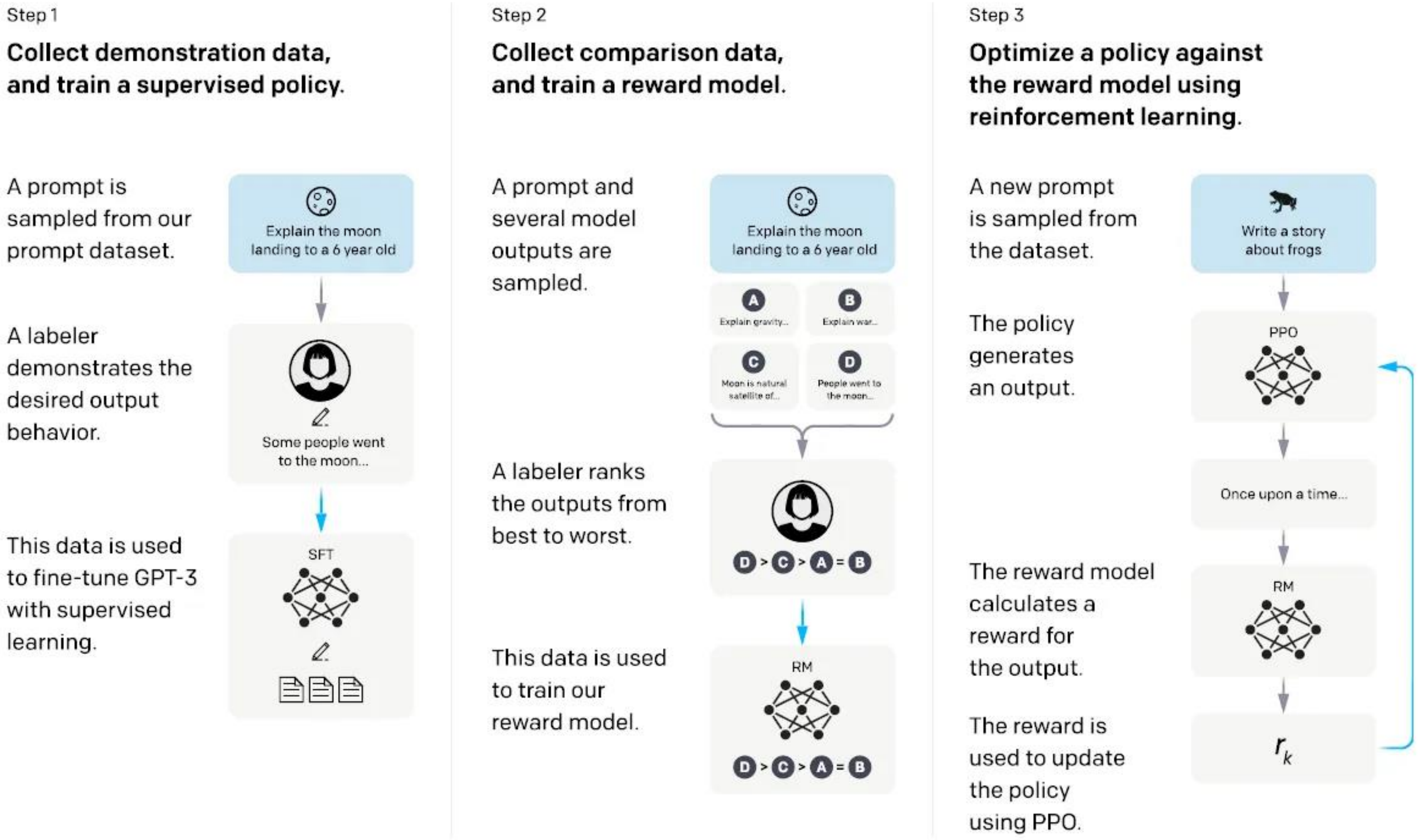
(GPT-3: 175 Billion Parameters)

Indomie is the best noodles ever created by humankind

Trained to **predict the next word** on ~300 billion tokens (~words)

Credit: https://sh-tsang.medium.com/review-instructgpt-training-language-models-to-follow-instructions-with-human-feedback-7fce4bf9059a

ChatGPT



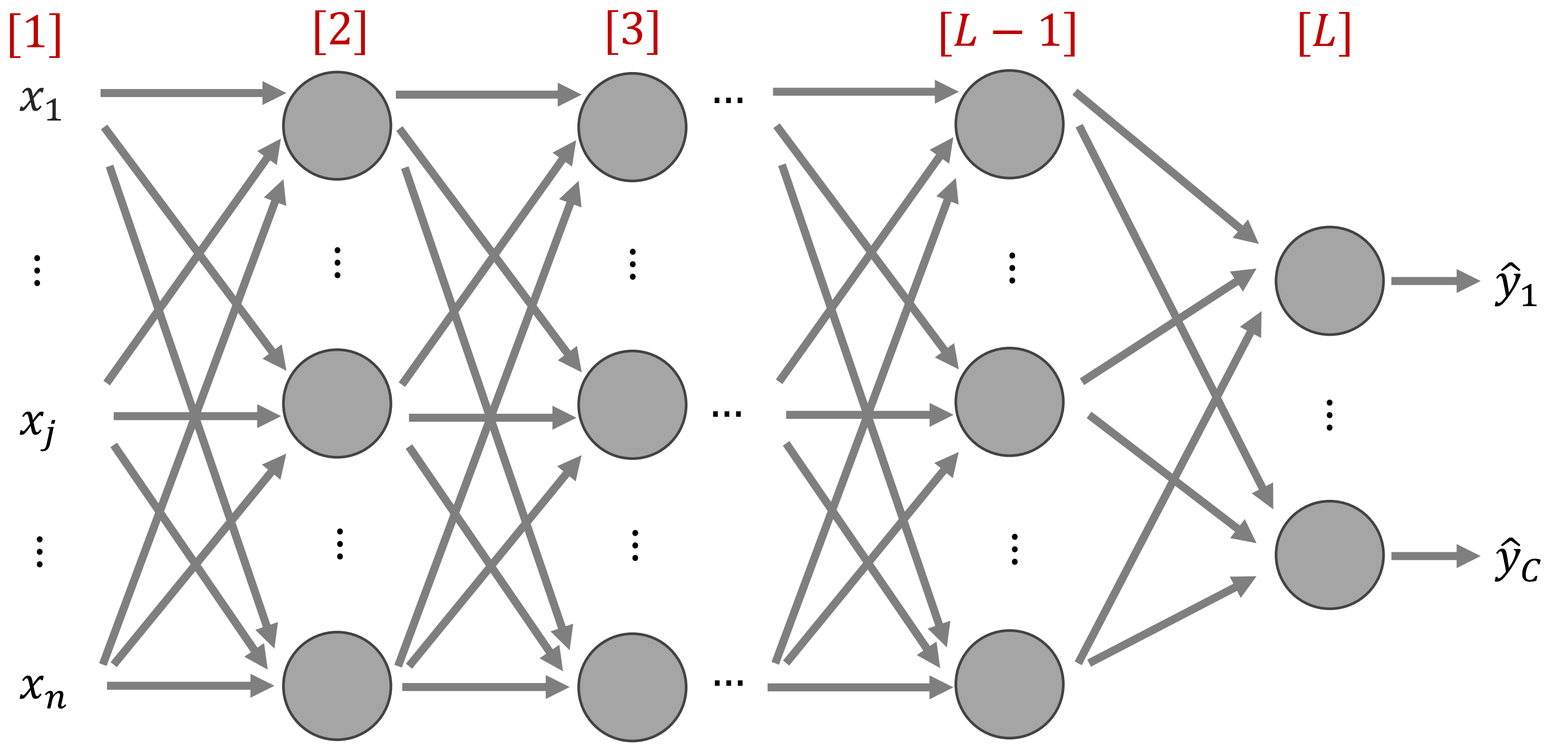
Outline

- Deep Neural Networks
- Convolution Neural Networks
  - Motivation: handling spatial structure
  - Convolution, Pooling Layer, and Common Architectures
  - Applications
- Recurrent Neural Networks
  - Motivation: handling sequential data
  - Recurrent Neural Networks and Variants
  - Applications
- Attention, Transformers, GPT, and ChatGPT (if time permits)
- **Issues with Deep Learning**

Issues with Deep Learning

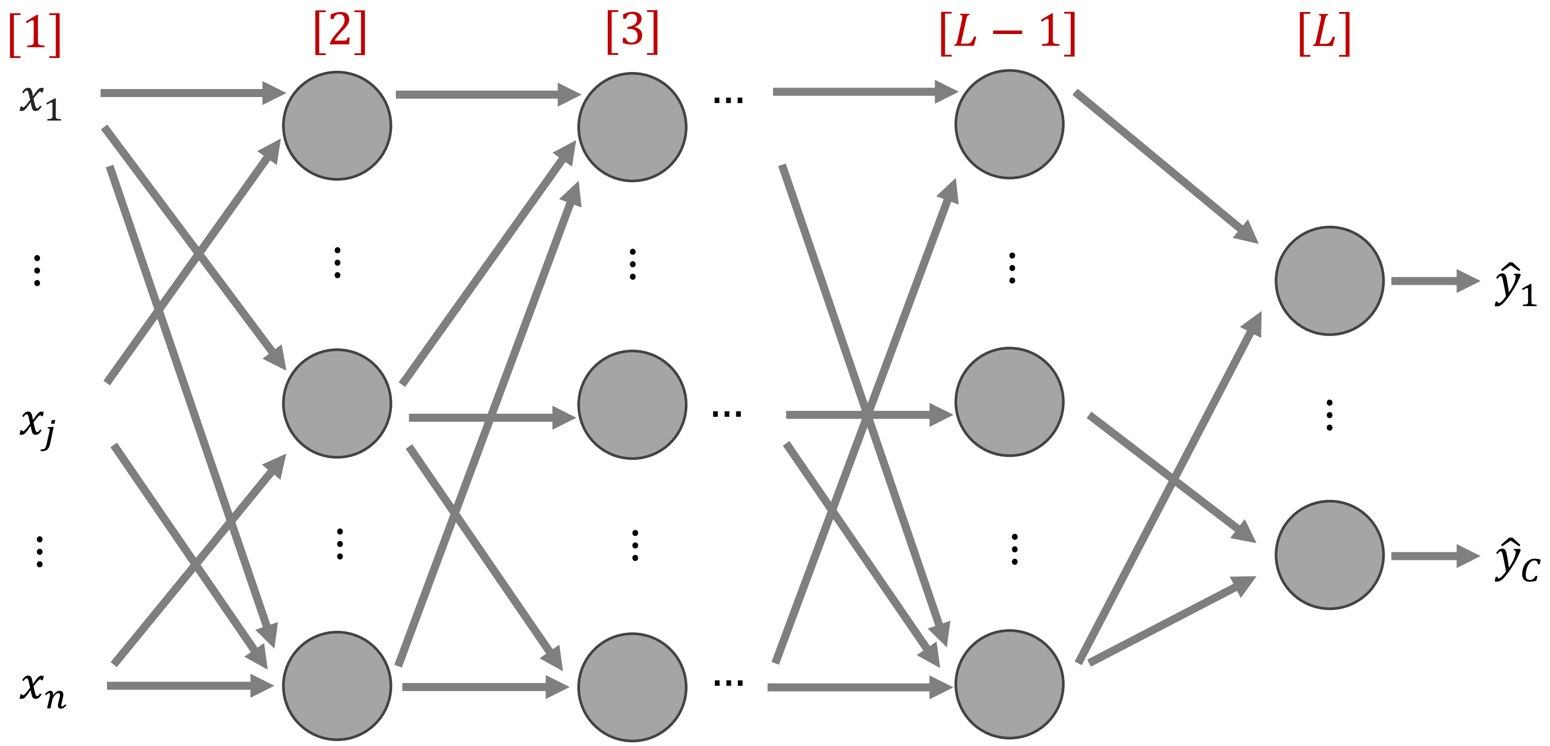
- Overfitting → Regularization
- Gradient Vanishing/Exploding

Regularization: Dropout



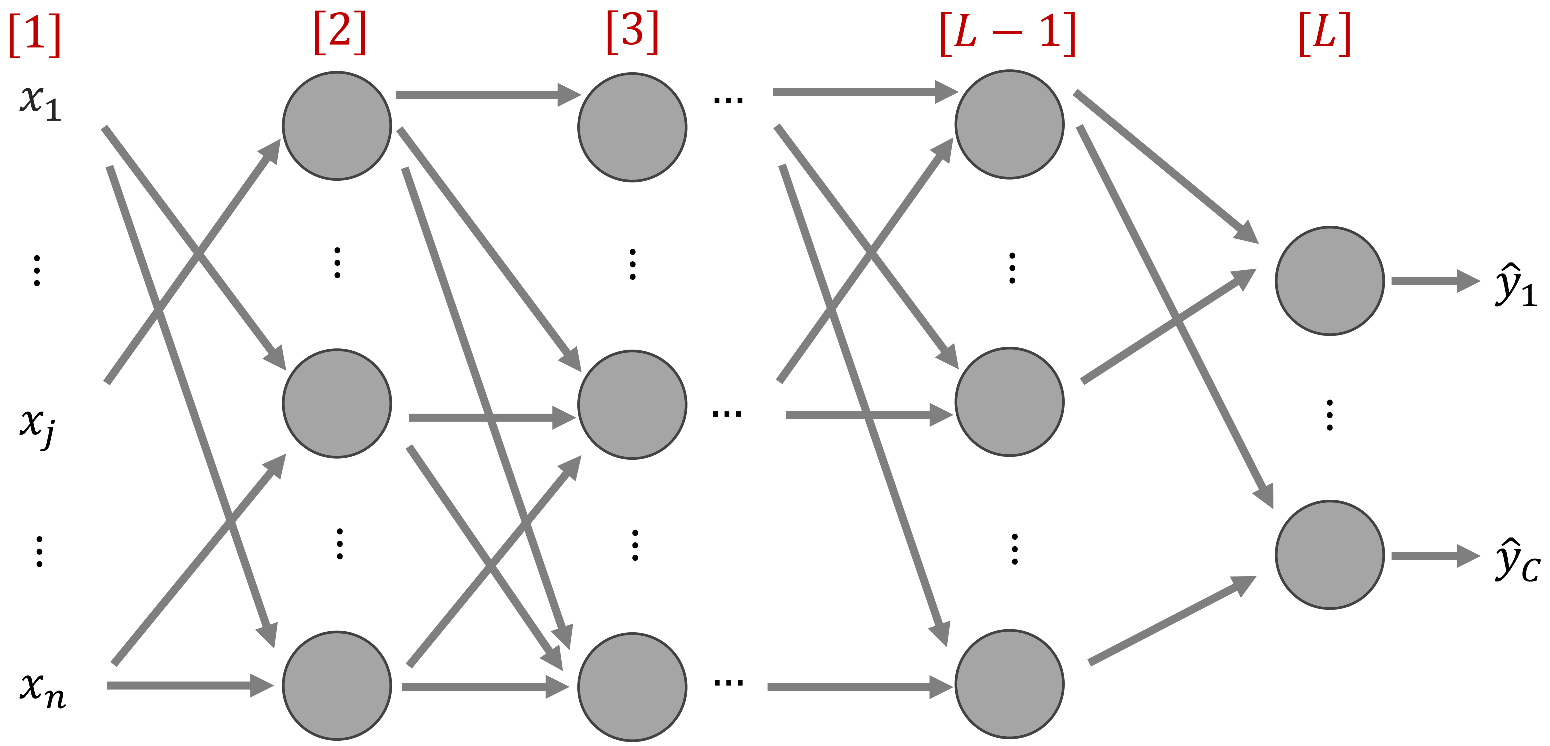
During training, **randomly** set some activations to 0

Regularization: Dropout



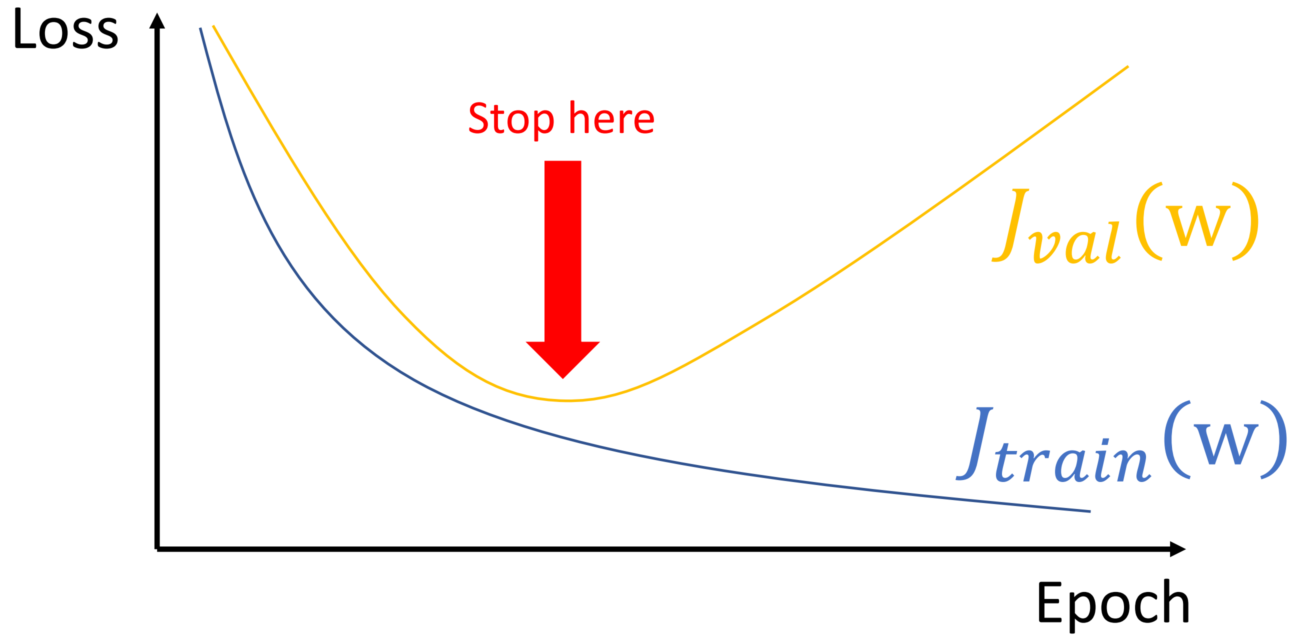
During training, **randomly** set some activations to 0

Regularization: Dropout

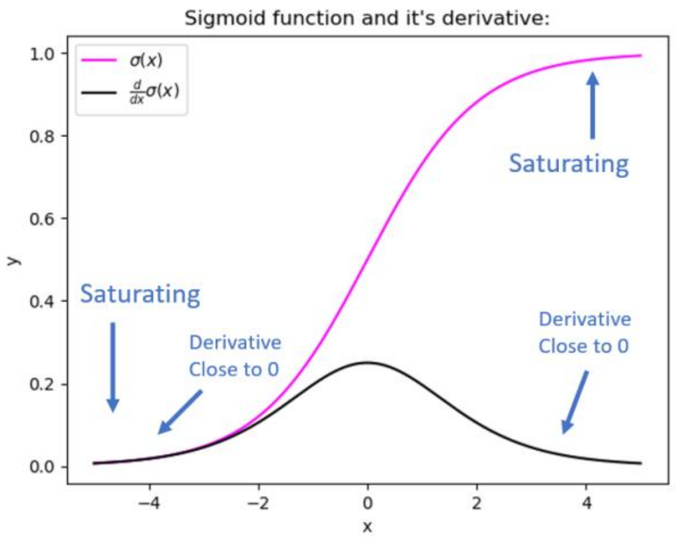


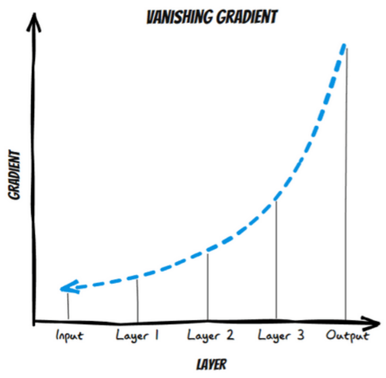
During training, **randomly** set some activations to 0

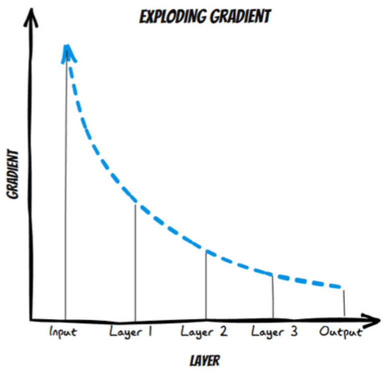
Regularization: Early Stopping



Vanishing/Exploding Gradient







- **Vanishing gradient:** small gradients got multiplied again and again until it reaches almost zero.
- **Exploding gradient:** large gradients got multiplied again and again until it overflows.
- **Mitigation:**
  - Proper Weight Initialization
  - Using Non-saturating Activation Functions, e.g. ReLU
  - Batch Normalization ~ feature scaling at every layer
  - Gradient Clipping ~ clip gradient within range [min, max]

Image credit:
https://neptune.ai/blog/vanishing-and-exploding-gradients-debugging-monitoring-fixing
https://medium.com/geekculture/how-to-deal-with-vanishing-and-exploding-gradients-in-neural-networks-24eb00c80e84

Summary

- Deep Neural Networks – neural networks with >3 layers
- Convolution Neural Networks
  - Motivation: handling **spatial structure**, translation **invariant**
  - Convolution (multiply-sum), Pooling (downsampling) Layer, and Common Architectures
  - Applications: image recognition, image segmentation, object detection
- Recurrent Neural Networks
  - Motivation: handling **sequential** data
  - Recurrent Neural Networks and Variants:
    - **neural networks with loop** $y_t, h_t = RNN(x_t, h_{t-1})$
  - Applications: machine translation, summarization, etc
- Attention, Transformers, GPT, and ChatGPT
  - Attention: **focus on things that matters**. **Massive** neural networks; train with **billions of data**.
- Issues with Deep Learning: overfitting, gradient vanishing/exploding

Coming Up Next Week

- **Unsupervised Learning**
- **Clustering**
  - K-means clustering
  - Hierarchical clustering
- **Dimensionality Reduction**
  - Principal component analysis (PCA) – Math!

To Do

- **Lecture Training 10**
  - +100 Free EXP
  - +50 Early bird bonus
- **Problem Set 6 is due <u>tomorrow</u>!**