UNIVERSITY OF TORONTO

Engineering

Deep Learning

Motivations behind Deep Learning

- Simple machine learning models struggle on tasks with complex data
- What are the obstacles faced by ML algorithms?

Motivations behind Deep Learning

- **Representation** of the given data.
  - Why is it challenging to build valuable features all the time?

Structured data
with features from
domain expert

v/s

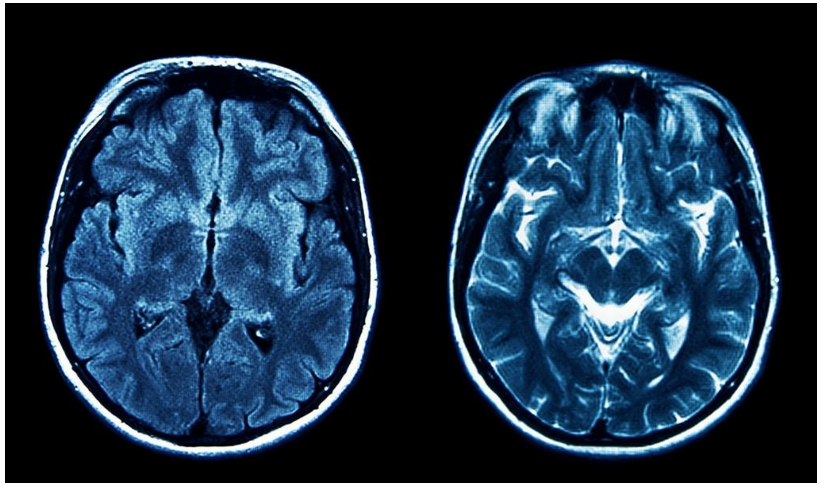
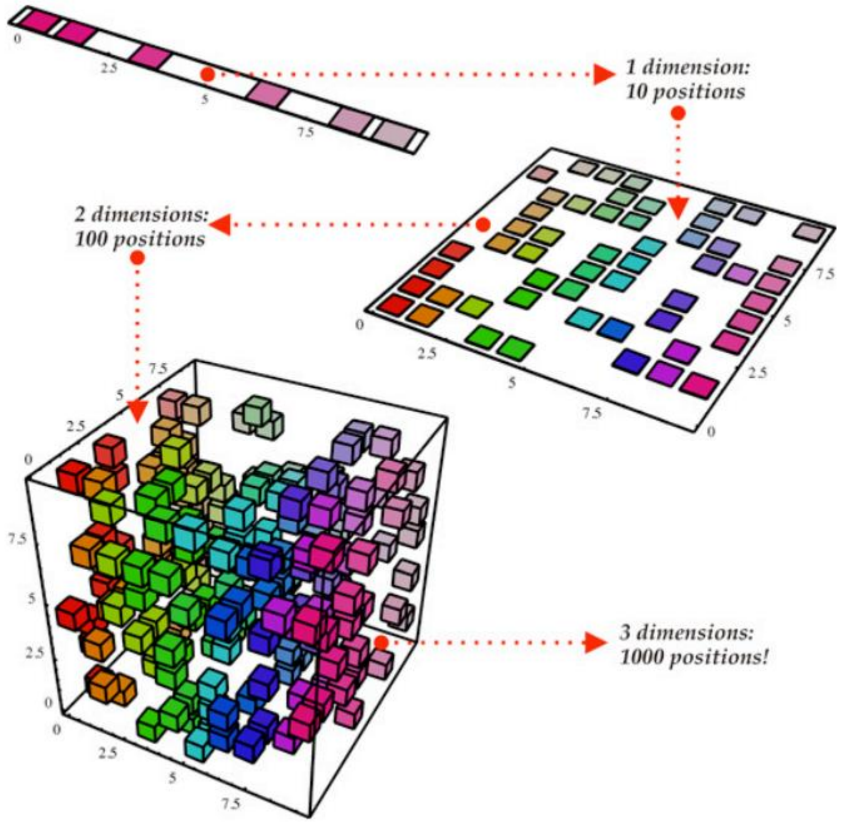


Image Source

Motivations behind Deep Learning

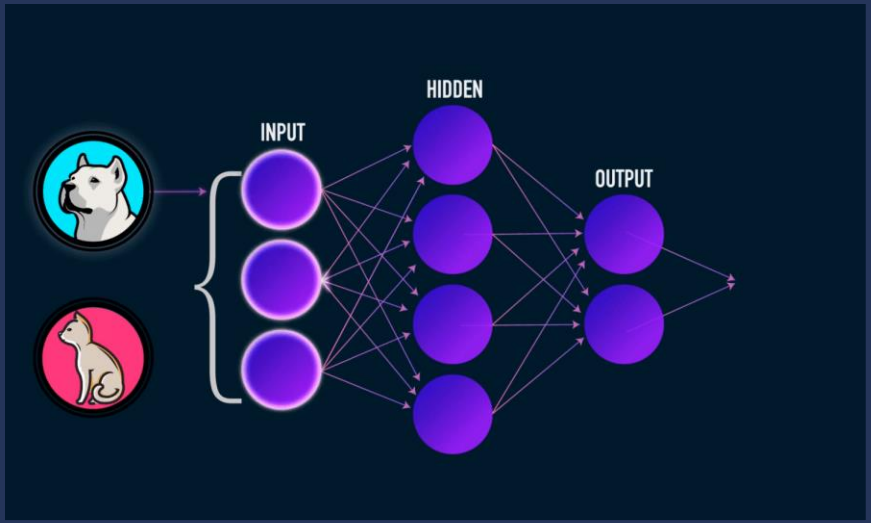
- **Number of Dimensions (Curse of Dimensionality)**
  - Why do you think the performance degrades?



[Source Image: Curse of Dimensionality]

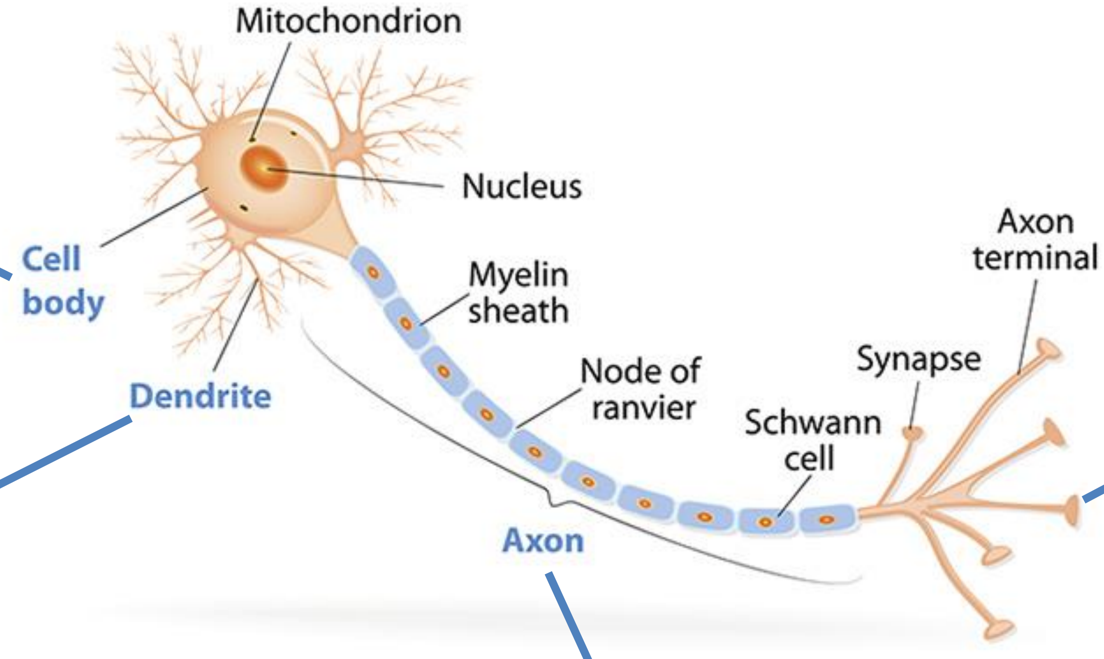
UNIVERSITY OF TORONTO | Engineering

Deep Feedforward Networks



Neural Network Inspiration

Cell body
collects and processes information

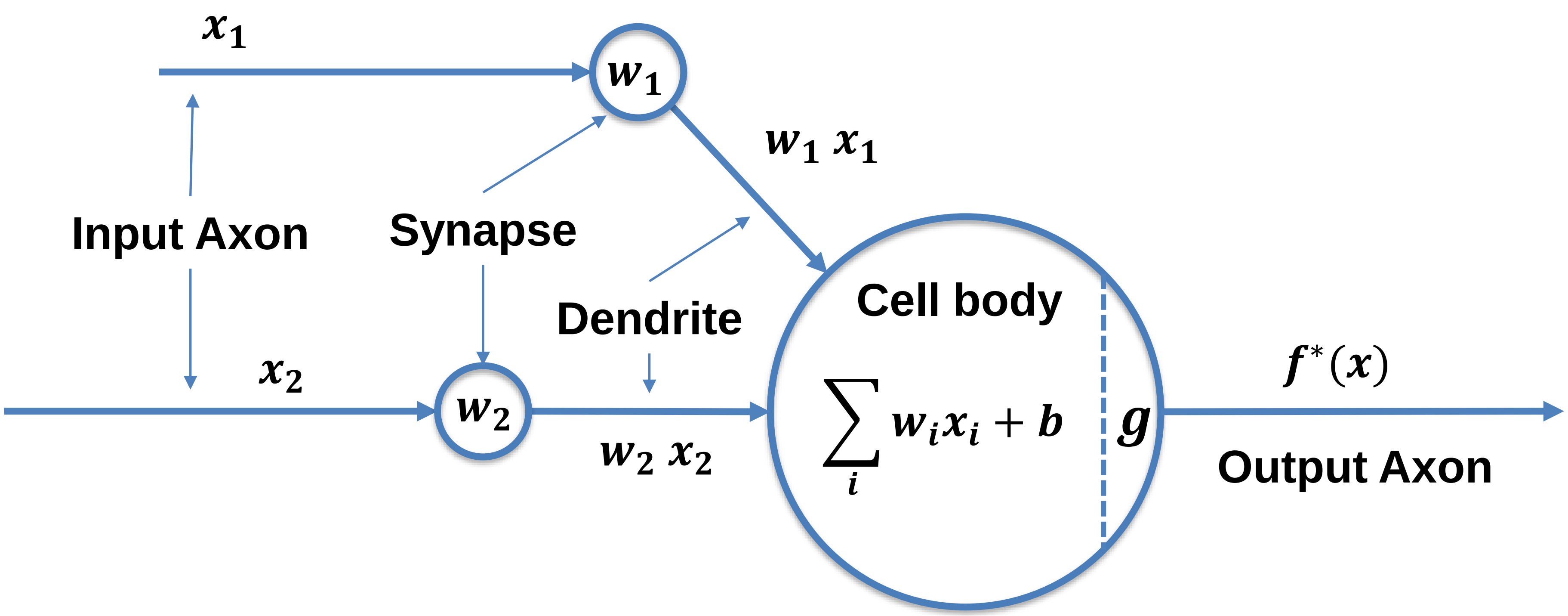


Synapse
connection between axon and dendrite

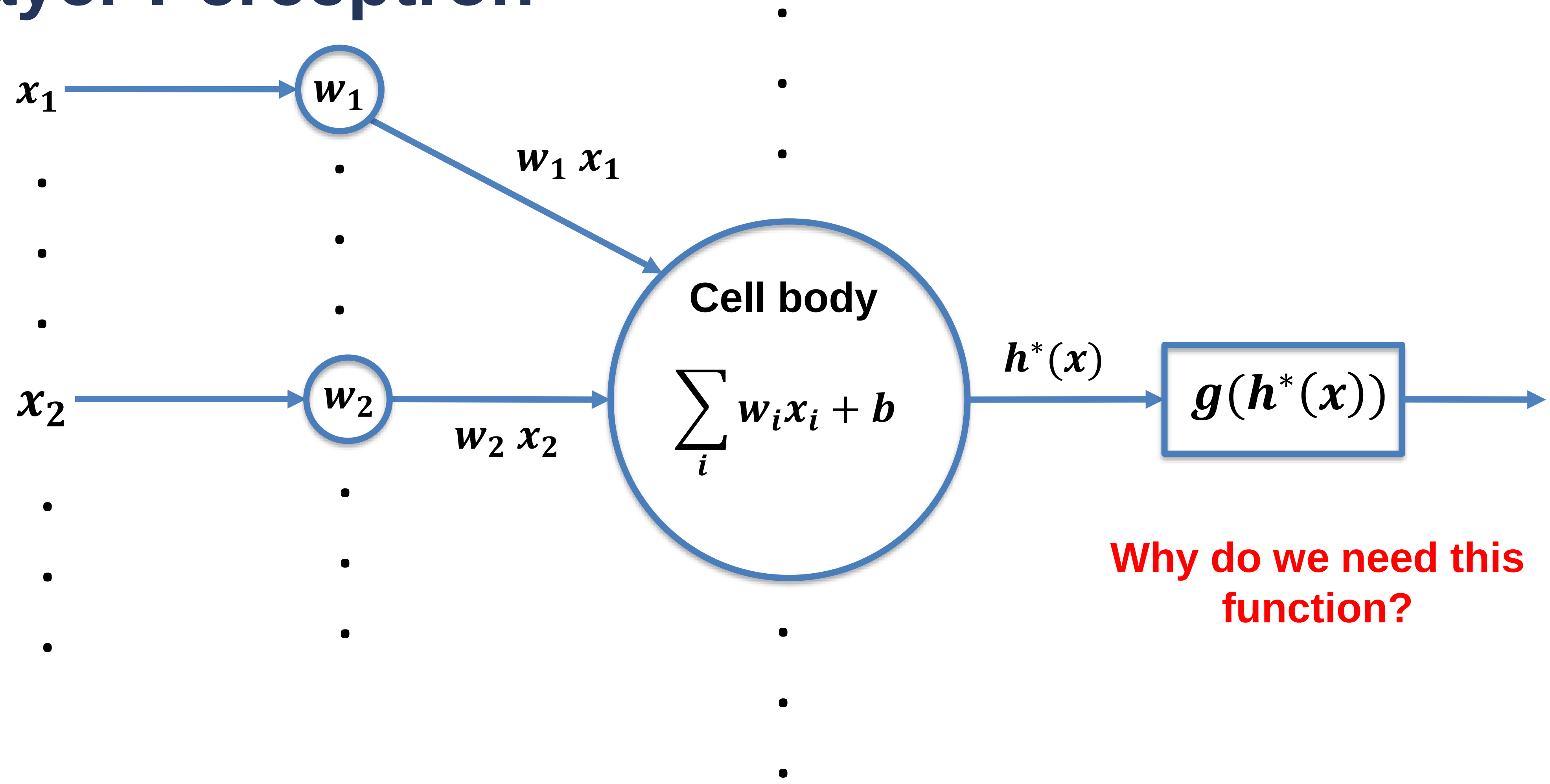
Dendrites
connected to other cells

Axon
passes information to other neurons

Neural Network Inspiration



Multilayer Perceptron



Why do we need this function?

Terminology

- Depth
- Input Layer
- Output Layer
- Hidden Layer
- Activations

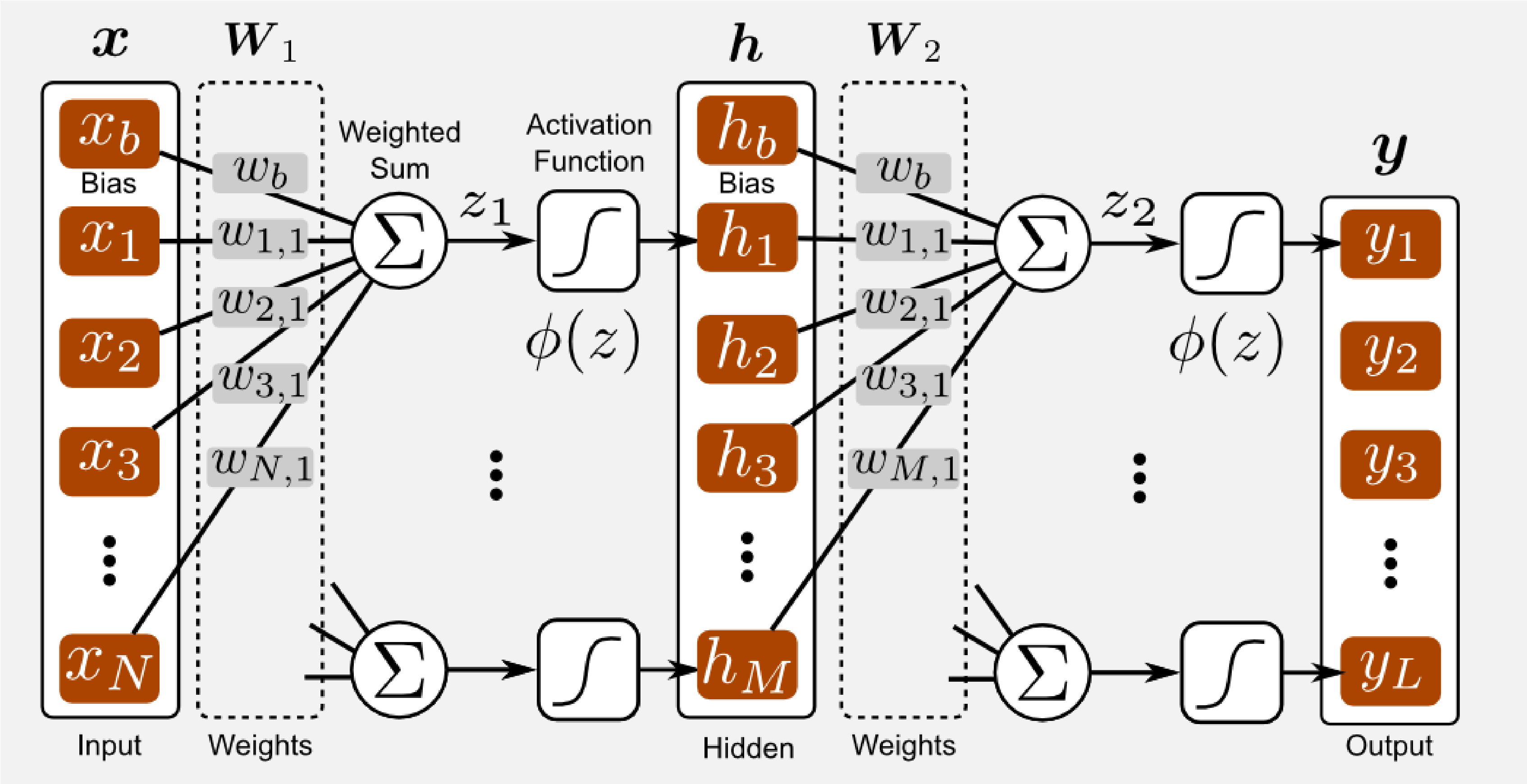


Image Source

Design Considerations

- Cost Functions
- Output Activation Units
- Hidden Activation Units
- Architecture Design

Cost Functions

- Mean Squared Error (**regression**): $\frac{1}{2N}\sum_{n=1}^{N}||\widehat{y_n} - y_n||^2 + const$

- Cross-Entropy Loss (**classification**): $-\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} t_{n,k}\log\left(y_{n,k}\right)$

Note: Mean Squared Error leads to poor results when used with saturating output units. Choice of activation function is critical here.

Activation Functions

- We use activation functions for non-linearity.

- Non-linear activation functions:
  - Sigmoid function
  - Tanh function (Hyperbolic Tangent)
  - ReLU function (Rectified Linear Unit)
  - Softmax function

Vanishing and Exploding Gradients

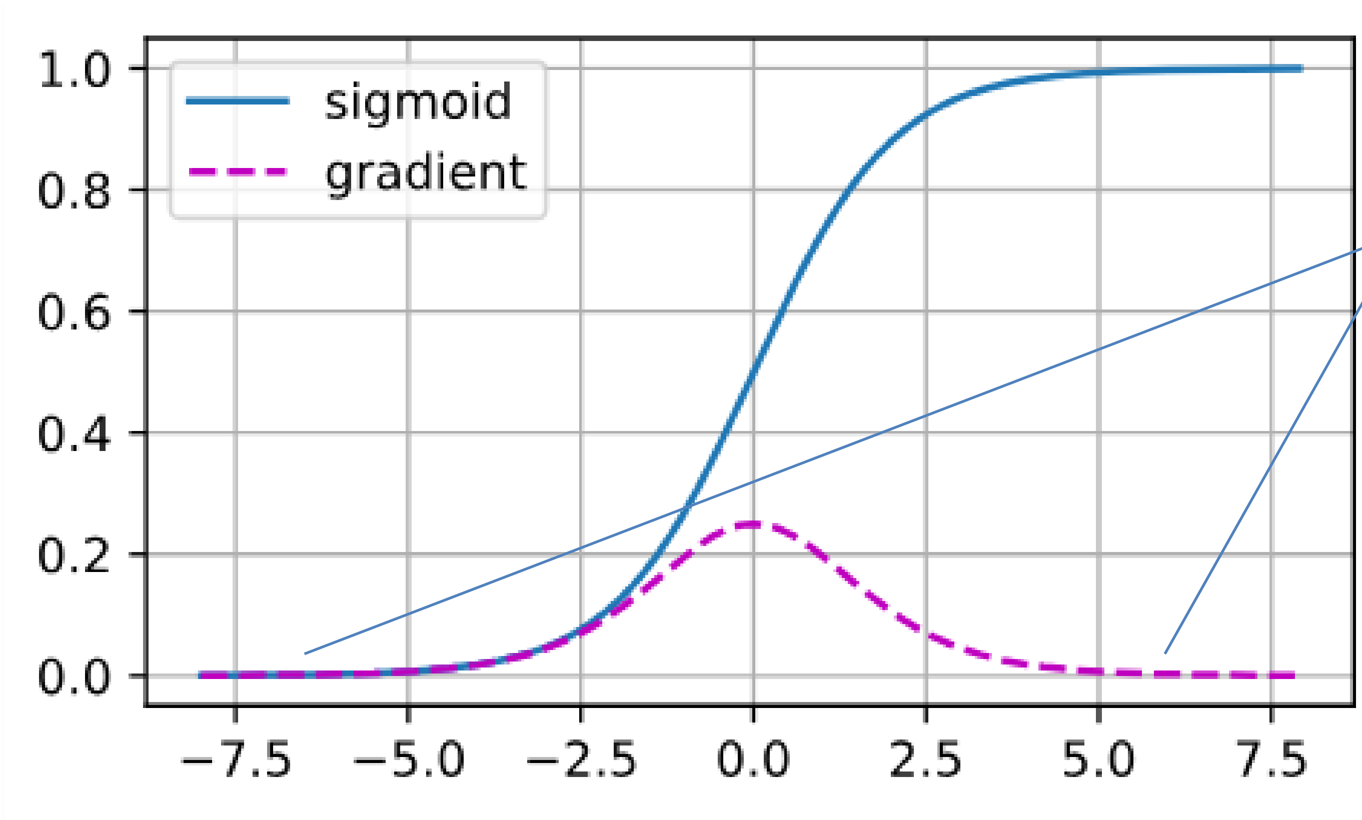


Image Source

Vanishing Gradients
The gradients on both ends of the sigmoid function are zero. This leads to a vanishing gradient problem.

Exploding Gradients
This is hard to pinpoint. Due to the chain multiplication of 100s of matrices, at times, the gradients become extremely large.

Design choices for Output Units

- **Linear Units for Gaussian Output Distributions:**
  - This is a simple output unit with no non-linearity and doesn't create any issue for gradient-based optimization.
  - Given h, this output unit produces $\hat{y} = W^T h + b$

Design choices for Output Units

- **Sigmoid Units for Bernoulli Output Distributions:**
  - Used when we are predicting binary variable.
  - The maximum likelihood approach defines a Bernoulli distribution with just one class, i.e., $P(y = 1|x)$. The output is constrained from [0, 1]
  - This leads us to a sigmoid output unit.

Design choices for Output Units

- **Softmax Units for Multinomial Output Distributions:**
  - Used when we have multiple classes. To obtain the desired output, we exponentiate and normalize z, which is the output of the linear layer.

$$\mathrm{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Design choices for Hidden Units

- The choice of activation functions in hidden units is difficult, and trial and error is normally used.

Note: Neural networks do not focus on differentiability of activation functions. They usually return one of the one-sided derivatives.

Design choices for Hidden Units

- **Rectified Linear Units:**
  - Easy to optimize as its behavior is close to linear.
  - Drawback: They cannot learn when the activation outputs zero.

- Three generalizations: $\max(0, x) - \alpha\min(0,\ x)$
  - Absolute value rectifications: $\alpha = -1$; $g(z) = |z|$
  - Leaky ReLU: $\alpha = 0.01$
  - Parametric ReLU: α *is learnable*

Architecture Design

- **Depth of the network:** Number of Layers

- **Width of each layer:** Number of units in the layers

Architecture Design

A ***large*** *single–layered* MLP will be able to represent any function, but

learning the function can fail:

- Optimization algorithm is unable to find the required parameters.
- Algorithm can choose a wrong function that overfits.

Deeper networks with fewer parameters usually generalize well.

Some Consideration in Architecture Design

- **Skip Connections:** helps with gradient flow
- **Connection between layers:**
  - **MLP** uses a linear transformation with a weight matrix. Every input unit is connected to every output unit.
  - **CNN** uses specialized kernels. We will discuss these in later sections.

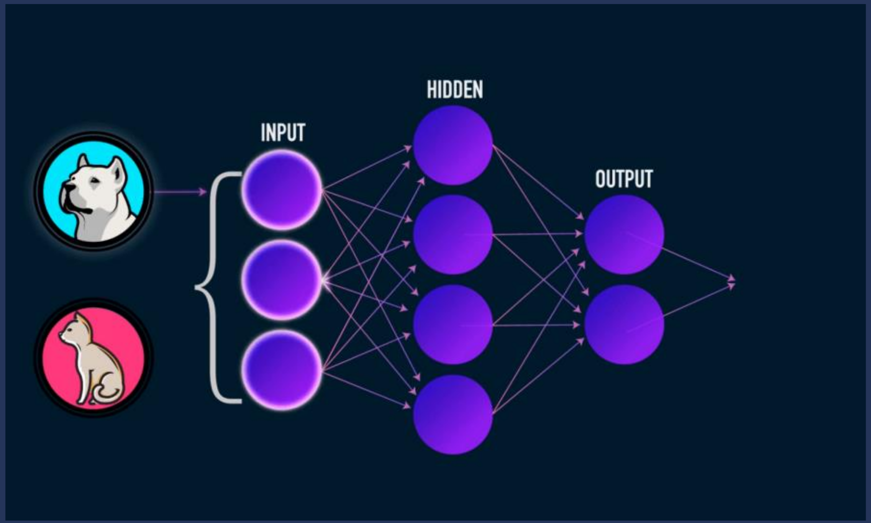
How do we train these deep networks?

- Once the architecture is built, we pass the input x and get an output $\hat{y}$ using **forward-propagation**.
- Calculate the cost function, $J(\theta)$
- To optimize, calculate gradients using **back-propagation.**
- To perform learning, **gradient descent** is used.

UNIVERSITY OF TORONTO

Engineering

Regularization for Deep Learning



Regularization Strategies

❖ Parameter Norm Penalties

- **Objective Function:** $\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha p(\theta)$; where $\alpha \in [0, \infty]$ and p is the penalty term.
- Only weights are penalized, biases are not.
- Sometimes, it is desirable to use a separate penalty term for each layer.

Regularization Strategies: Norm Penalties

- **L2 Parameter Regularization:** $\boldsymbol{p(w) = \frac{\alpha}{2} w^T w}$
  - $\nabla_w \tilde{J}(\theta; X, y) = \alpha w + \nabla_w J(w; X, y)$
  - Update: $w \leftarrow w - \epsilon(\alpha w + \nabla_w J(w; X, y)) = (1 - \epsilon\alpha)w - \epsilon \nabla_w J(w; X, y)$
  - We can see that the weight vector is shrunk by a constant factor on each step, leading to less complex models.

Regularization Strategies: Norm Penalties

- **L1 Parameter Regularization:** $\boldsymbol{p(w)} = \sum |\boldsymbol{w}|$
  - $\nabla_w \tilde{J}(\theta; X, y) = \alpha sign(w) + \nabla_w J(w; X, y)$
  - Update: $w \leftarrow w - \epsilon(\alpha sign(w) + \nabla_w J(w; X, y))$
  - Like L2 regularization, L1 also encourages zero coefficients.

Regularization Strategies: Data Augmentation

- More data → a Better generalized model

- Data is limited. Another way to increase the dataset size is augmentation.

Computer Vision Tasks: Translation, Flips, Random cropping, Rotating, Color

Speech Recognition: Speeding up, Slowing down, Jitter, Pitch shift

Regularization Strategies: Noise Robustness

- An interesting way of increasing model robustness is adding noise to network weights.
- It encourages the parameters to go to regions of parameter space where small perturbations of weights have a small influence on the output → **points of minima surrounded by flat regions.**

Regularization Strategies: Early Stopping

- Checkpoints of the model are stored at every training epoch.
- The checkpoint that has the lowest validation error is selected.

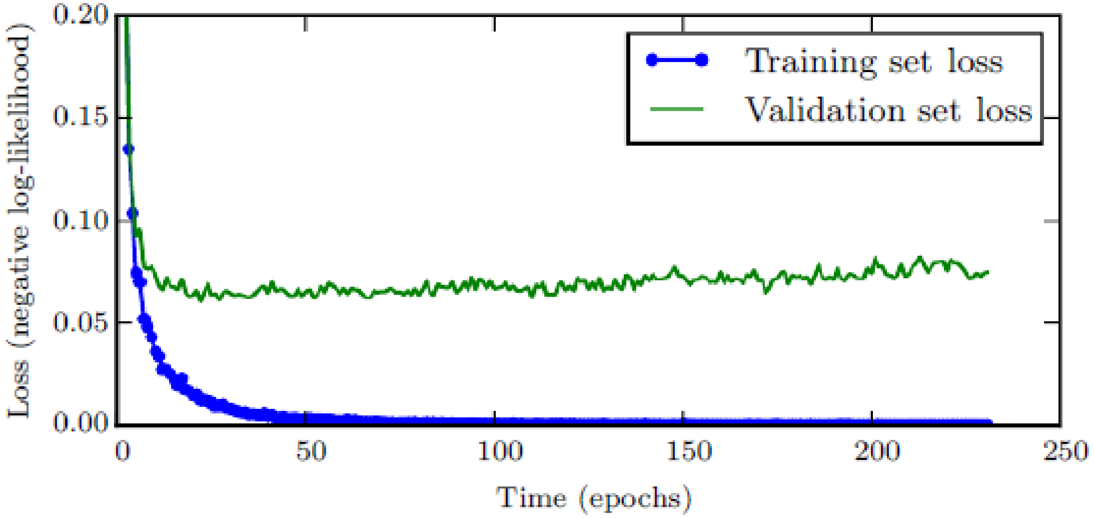
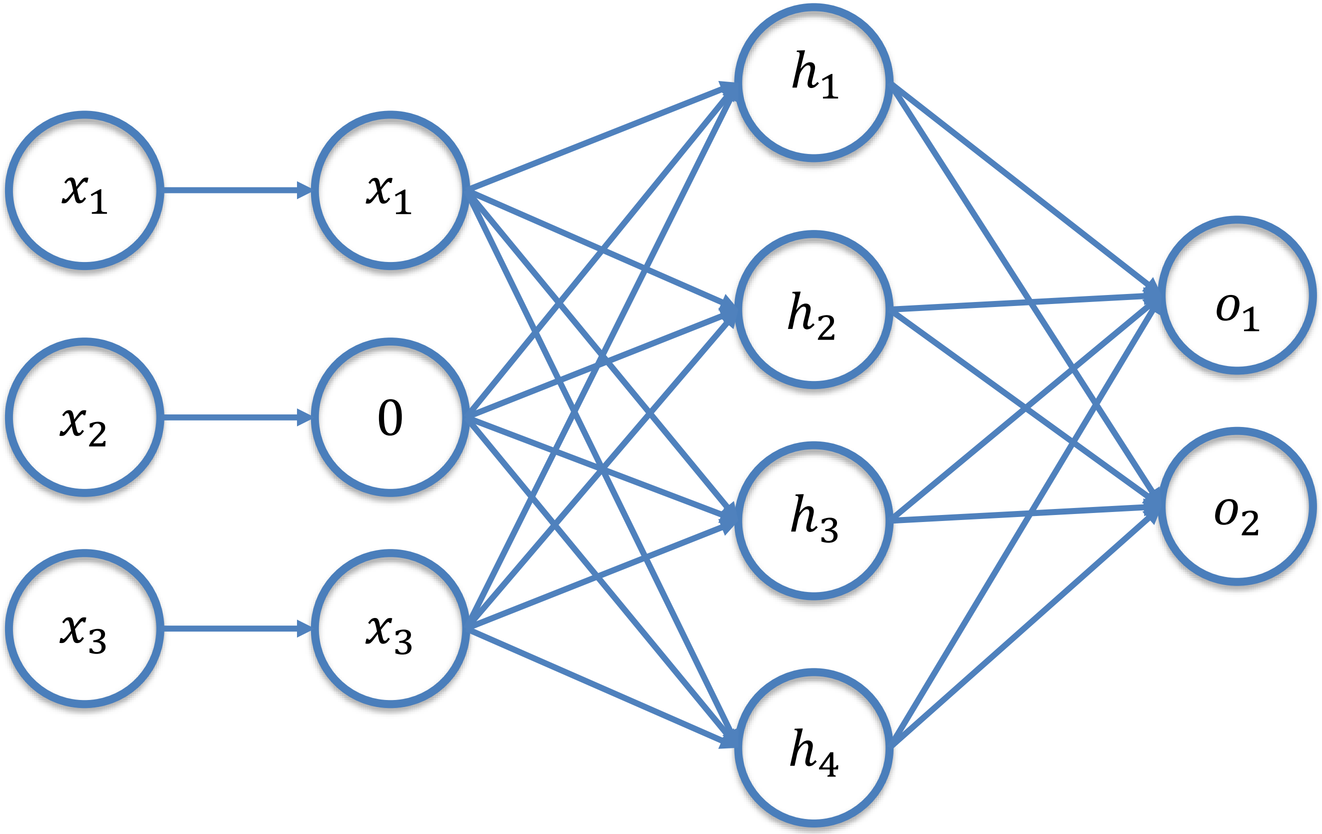


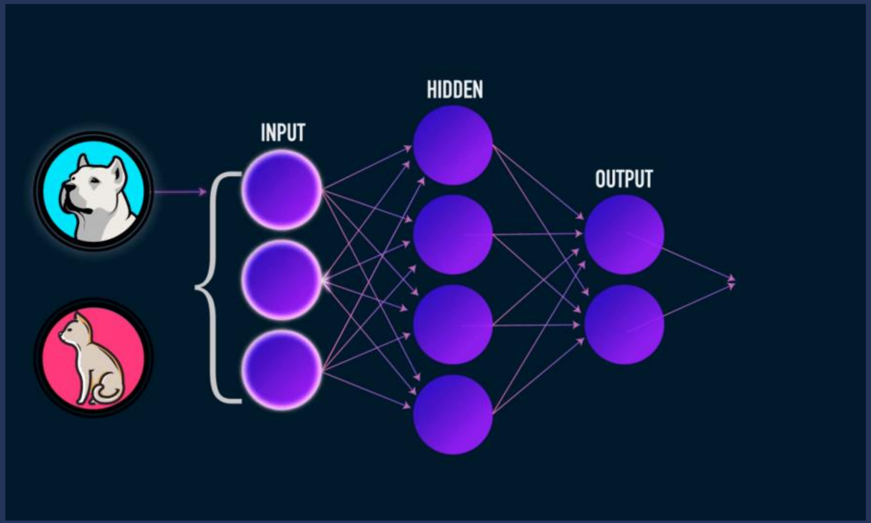
Image Source

Regularization Strategies: Dropout

- Zero out randomly selected inputs.



Optimization for Deep Learning



Empirical Risk Minimization

- Goal of a machine learning algorithm is to reduce the expected generalization error also known as risk: $\boldsymbol{J}^*(\boldsymbol{\theta}) = \mathbf{E}_{(\boldsymbol{x},\boldsymbol{y})\sim \boldsymbol{p}_{\boldsymbol{data}}} \boldsymbol{L}(\boldsymbol{f}(\boldsymbol{x};\ \boldsymbol{\theta}), \boldsymbol{y})$
- $\boldsymbol{p}_{\boldsymbol{data}}$ is the true distribution which is not known, and we only have a training set during model development.
- The $\boldsymbol{p}_{\boldsymbol{data}}$ is replaced with empirical distribution and we minimize empirical risk.

Batch and Minibatch Algorithms

- **Batch or deterministic** gradient methods: Use the entire training set.
- **Stochastic or online** gradient descent: Use one example at a time.
- **Minibatch *stochastic* methods:** More than one but fewer than all.

Basic Algorithms: Stochastic Gradient Descent

Algorithm 8.1 Stochastic gradient descent (SGD) update

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$
Require: Initial parameter $\boldsymbol{\theta}$
  $k \leftarrow 1$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
    $k \leftarrow k + 1$
  end while

Image Source

Basic Algorithms: Momentum

Without Momentum

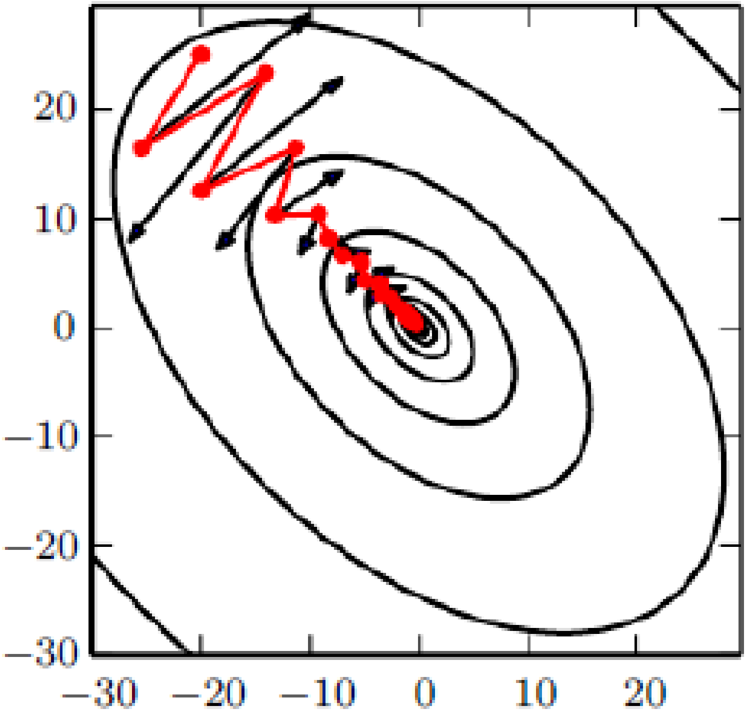


Image Source

- Direction of the update has some variance.
- **With Momentum,** the updates are smoother.
  - Stores the directions of previous gradients, $\boldsymbol{v}$.
  - **Moves** faster if the directions are similar.

Basic Algorithms: Momentum

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α

Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.

Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$.

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$.

end while

Image Source

Parameter Initialization

- **Speeds** up convergence of gradient descent.
- **Can** lead to a lower generalization error.
- Different Strategies:
  - Zero Initialization: Not effective
  - Random Initialization: breaks symmetry, large values are problematic.
  - He Initialization: random initialization * $\sqrt{\dfrac{2}{dimension\ of\ the\ previous\ layer}}$

Questions to consider

- **What happens if the batch size is too small? Too large?**

- **Too small:**
  - See new instances in every step
  - Average loss very noisy

- **Too large:**
  - Expensive

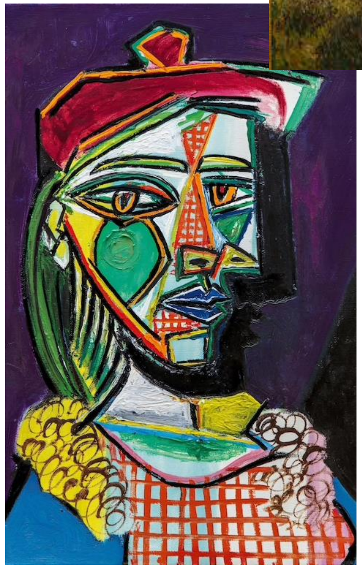
Questions to consider

- **What happens if the learning rate is too small? Too large?**
- **Too small:**
  - Minor updates to parameters in each iteration
  - Training time is very high
- **Too large:**
  - Noisy updates and unstable

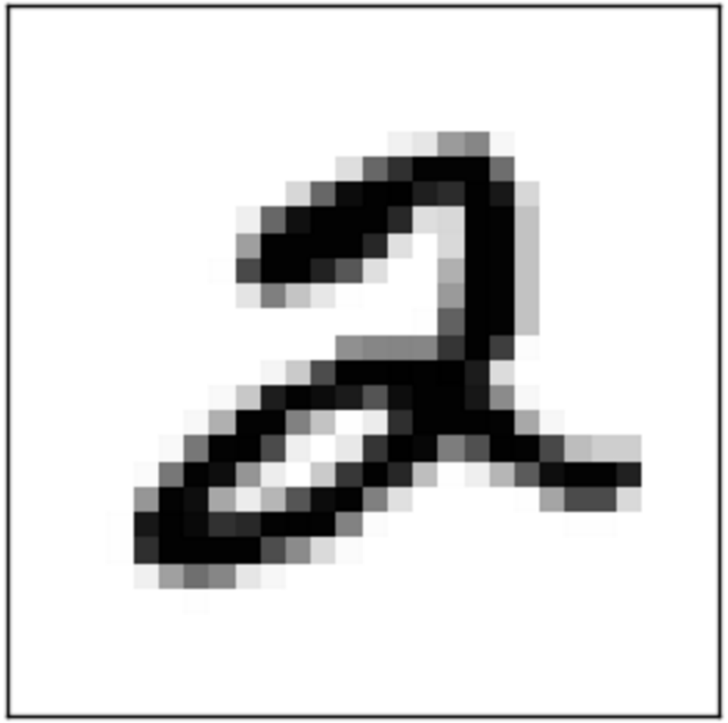
UNIVERSITY OF TORONTO | Engineering

Convolutional Networks

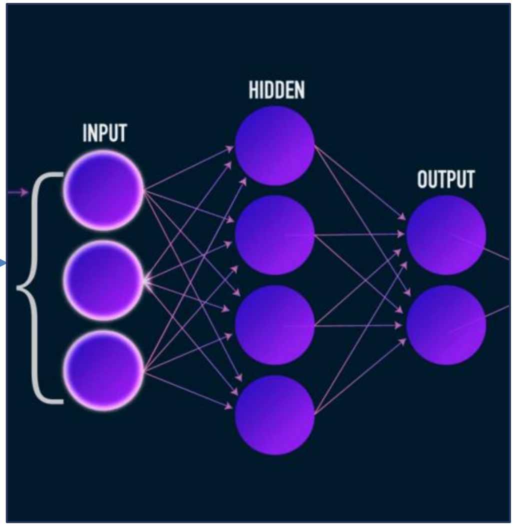
Deep Learning



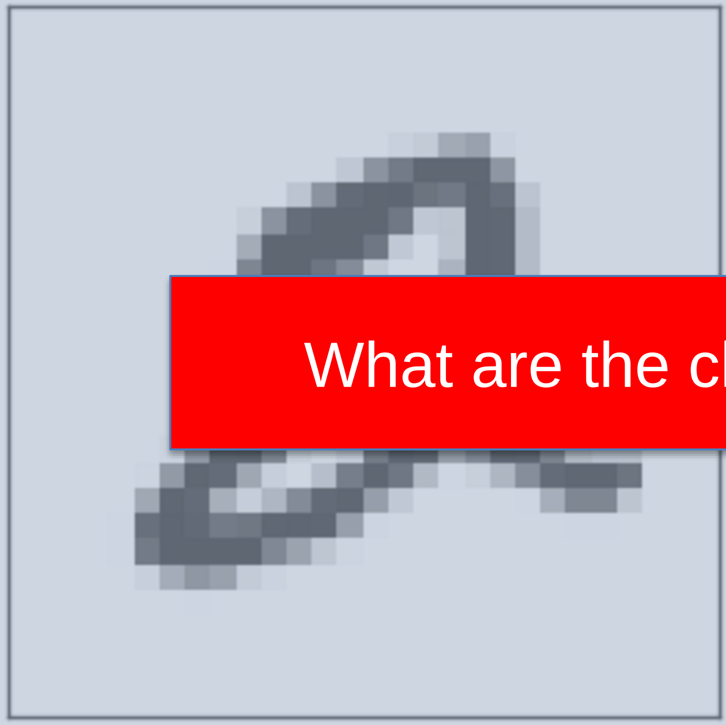
MLP with Image data



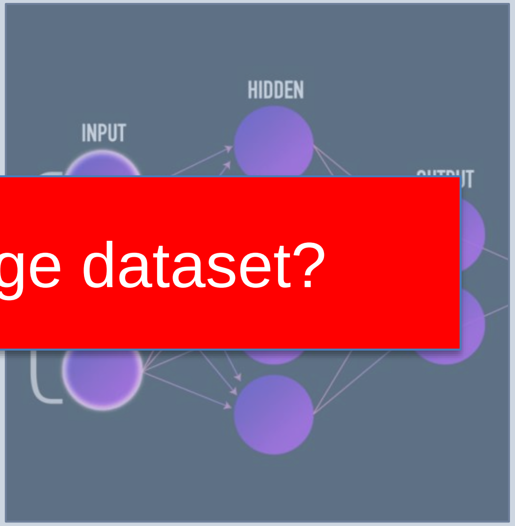
Pixel 1
Pixel 2
Pixel 3
.
.
.
Pixel 783
Pixel 784



MLP with Image data



Pixel 1
Pixel 2
Pixel 3
.
Pixel 783
Pixel 784



What are the challenges faced in MLP with image dataset?

Challenges

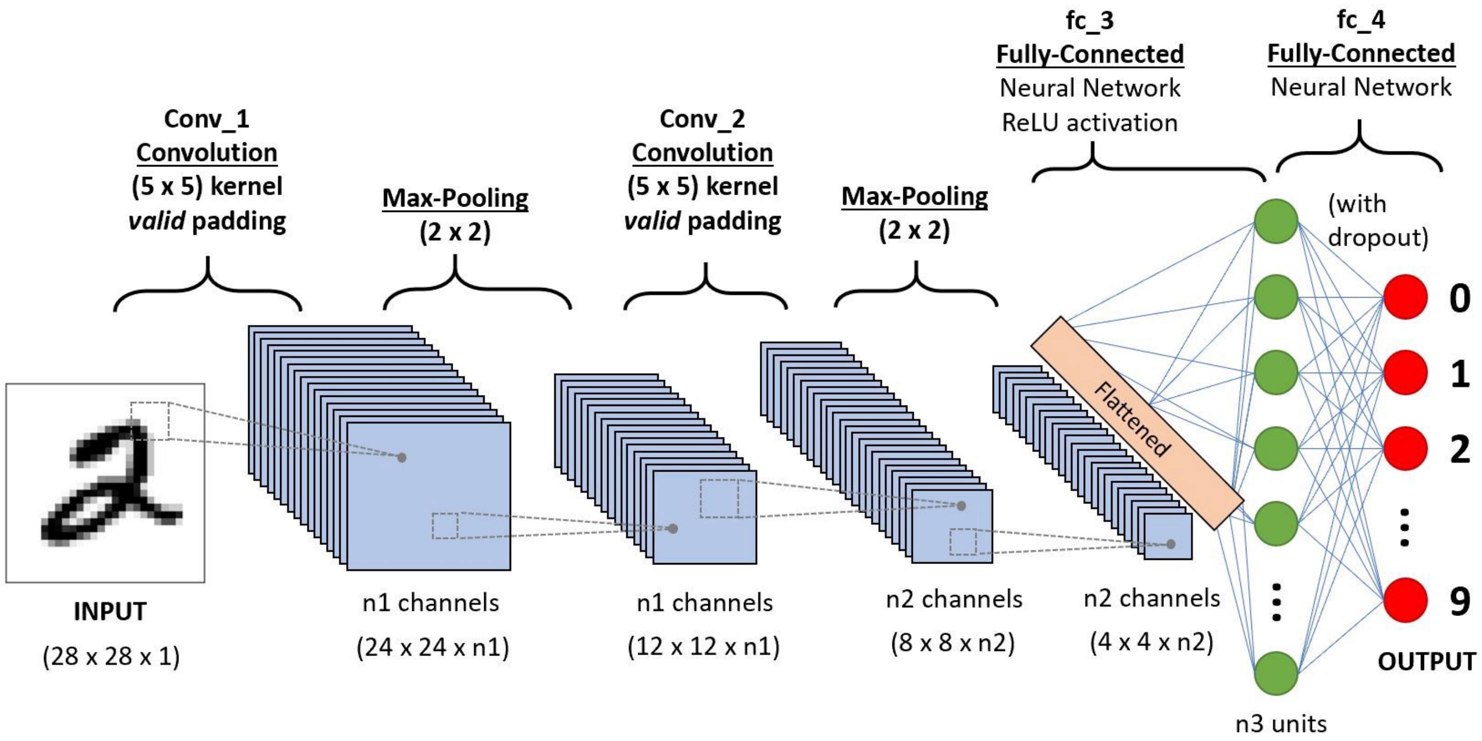
- Computation time will be higher
- Large number of weights
- Each pixel has its own associated weight. Small shifts in image can result in large change in prediction.
- Does not exploit geometric relationships in image.

General Architecture of CNN

Weight Sharing

Local connection

Feature Engineering



Steps in 2D Convolution

- Flip rows and columns of the kernel.
- Element-wise multiplication of each pixel in the range of the kernel.
- Slide kernel to cover the entire input image.

Image

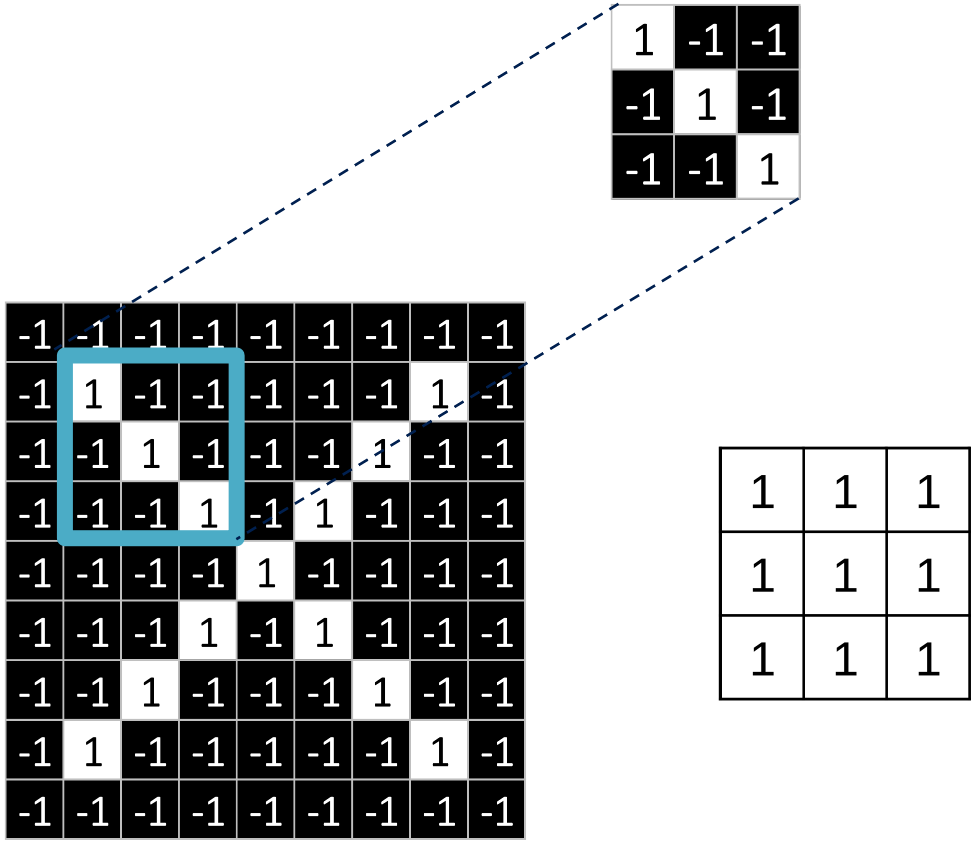
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

Kernel

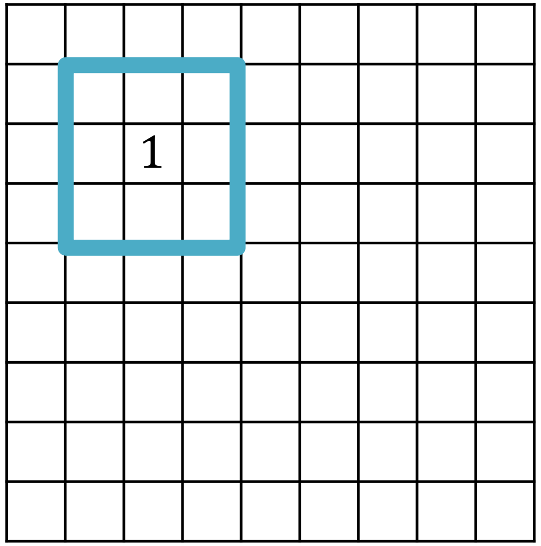
1	-1	1
-1	1	-1
1	-1	1

Steps in CNN

- Filtering



$$\frac{1+1+\cdots+1}{9} = 1$$



Steps in CNN

- Filtering

1	-1	-1
-1	1	-1
-1	-1	1

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

1	1	1
1	1	1
1	1	1

$$\frac{1+1+\cdots+1}{9} = 1$$

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

Convolution

$$
\begin{bmatrix}
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
-1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\
-1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\
-1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\
-1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 \\
-1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\
-1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\
-1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\
-1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1
\end{bmatrix}
\otimes
\begin{bmatrix}
1 & -1 & -1 \\
-1 & 1 & -1 \\
-1 & -1 & 1
\end{bmatrix}
=
\begin{bmatrix}
0.77 & -0.11 & 0.11 & 0.33 & 0.55 & -0.11 & 0.33 \\
-0.11 & 1.00 & -0.11 & 0.33 & -0.11 & 0.11 & -0.11 \\
0.11 & -0.11 & 1.00 & -0.33 & 0.11 & -0.11 & 0.55 \\
0.33 & 0.33 & -0.33 & 0.55 & -0.33 & 0.33 & 0.33 \\
0.55 & -0.11 & 0.11 & -0.33 & 1.00 & -0.11 & 0.11 \\
-0.11 & 0.11 & -0.11 & 0.33 & -0.11 & 1.00 & -0.11 \\
0.33 & -0.11 & 0.55 & 0.33 & 0.11 & -0.11 & 0.77
\end{bmatrix}
$$

Convolution on Multiple Filters

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

$\otimes$

1	-1	1
-1	1	-1
1	-1	1

$=$

0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.11	0.33	-0.77	1.00	-0.77	0.33	-0.11
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

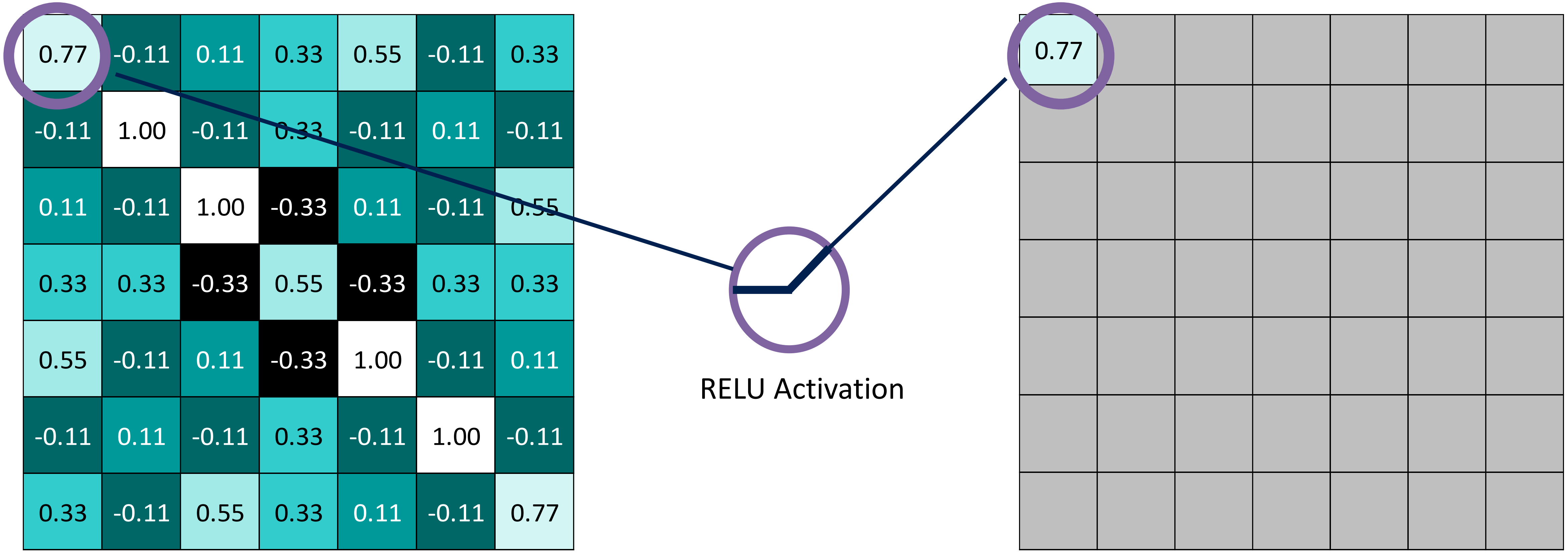
$\otimes$

-1	-1	1
-1	1	-1
1	-1	-1

$=$

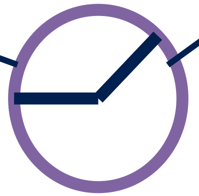
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33

Activation


0.77 -0.11 0.11 0.33 0.55 -0.11 0.33
-0.11 1.00 -0.11 0.33 -0.11 0.11 -0.11
0.11 -0.11 1.00 -0.33 0.11 -0.11 0.55
0.33 0.33 -0.33 0.55 -0.33 0.33 0.33
0.55 -0.11 0.11 -0.33 1.00 -0.11 0.11
-0.11 0.11 -0.11 0.33 -0.11 1.00 -0.11
0.33 -0.11 0.55 0.33 0.11 -0.11 0.77
RELU Activation
0.77

Activation

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

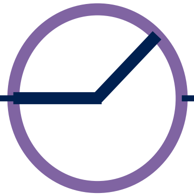


RELU Activation

0.77	0					

Activation

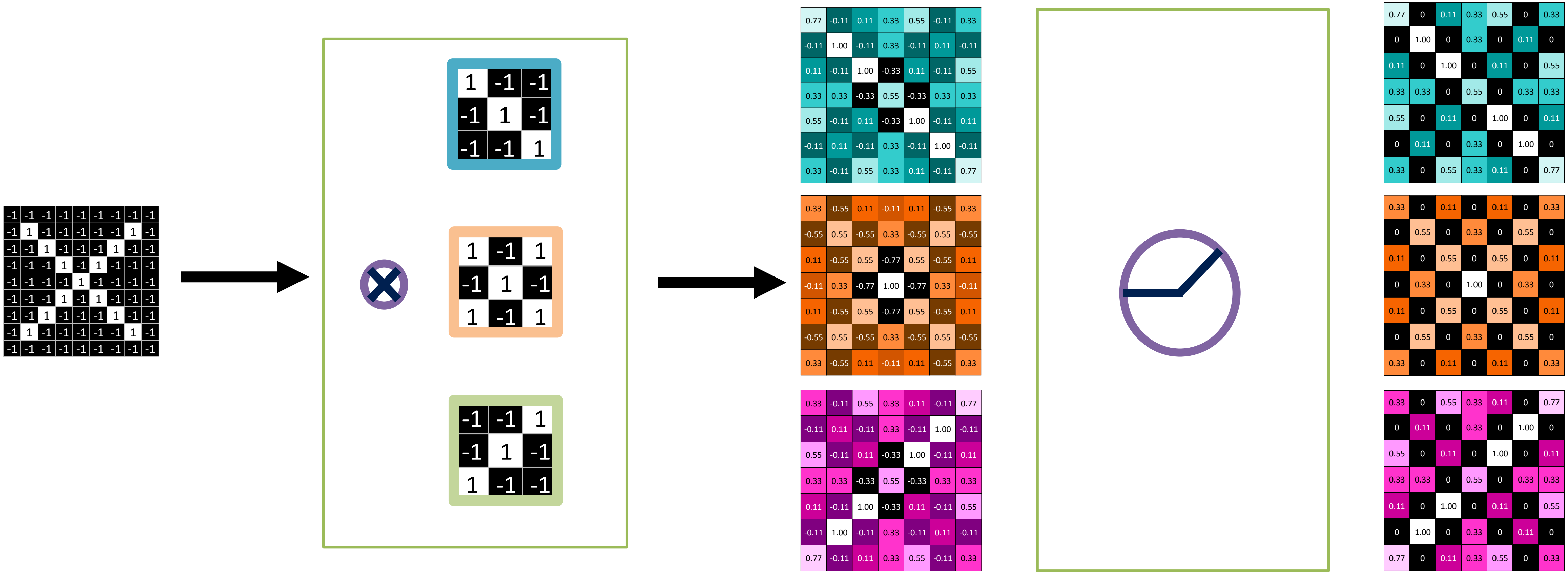
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77



RELU Activation

0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

Convolution and Activation



Pooling

0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

maximum

$$max\{0.77, 0, 0, 1.00\} \rightarrow 1.00$$

1.00			

Pooling

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

maximum

$max\{0.11, 0.33, -0.11, 0.33\} \rightarrow 0.33$

1.00	0.33		

Pooling

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

Maximum

1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

Pooling on All Channels

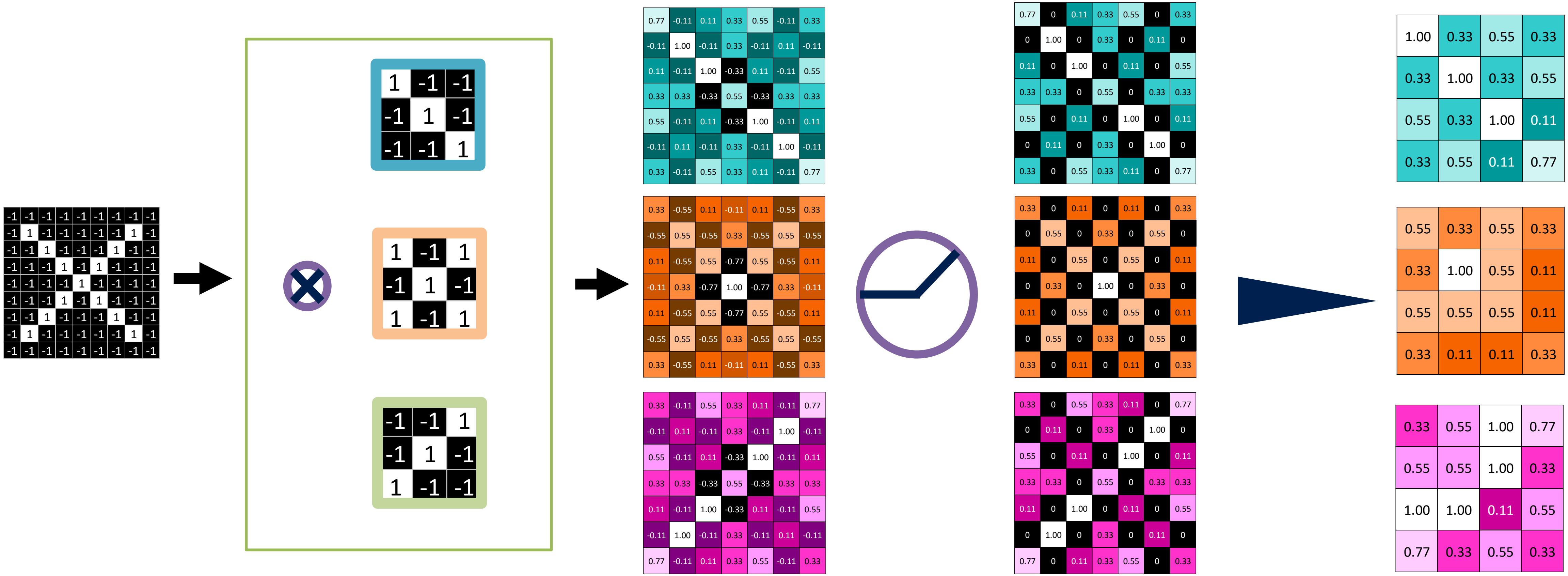
0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.11	0.33	-0.77	1.00	-0.77	0.33	-0.11
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33

0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

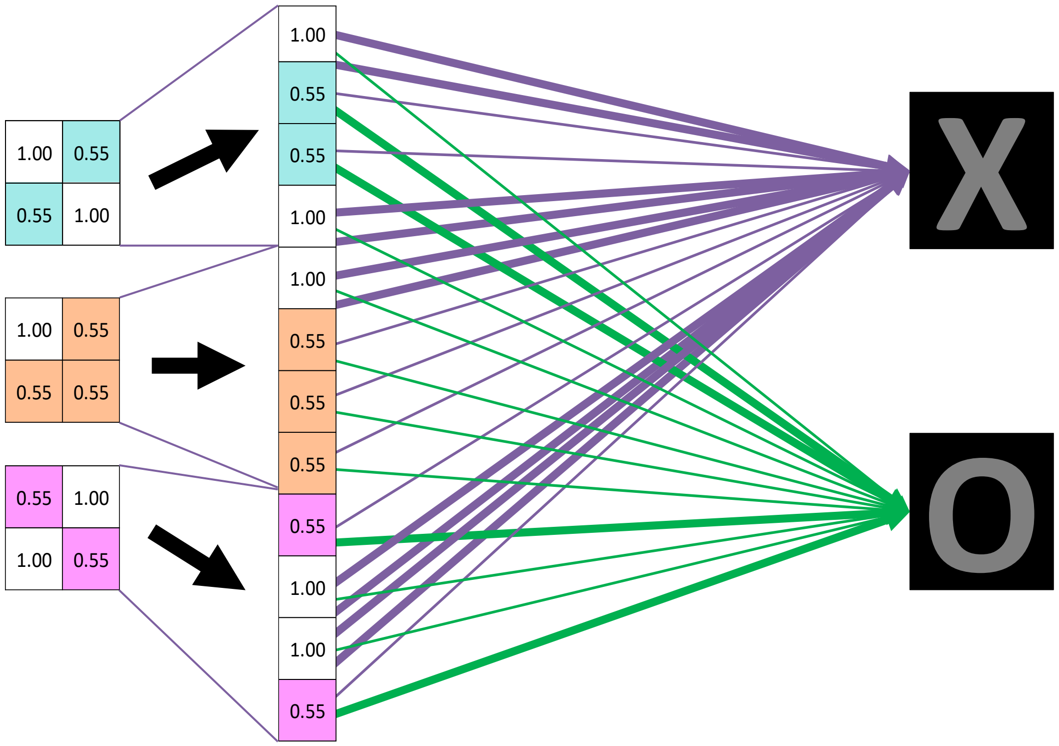
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33

0.33	0.55	1.00	0.77
0.55	0.55	1.00	0.33
1.00	1.00	0.11	0.55
0.77	0.33	0.55	0.33

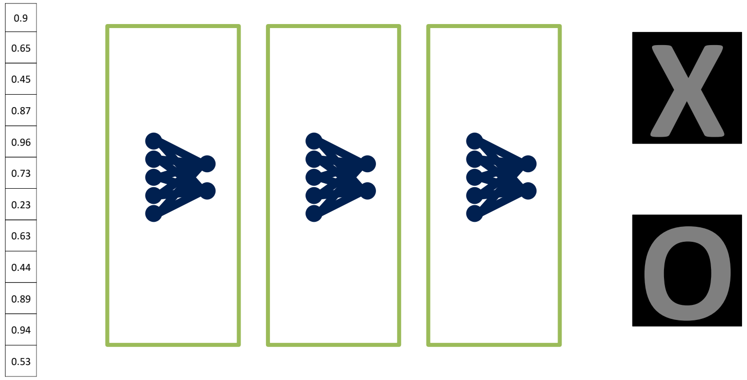
Convolution and Pooling over 3 Channels



Classification

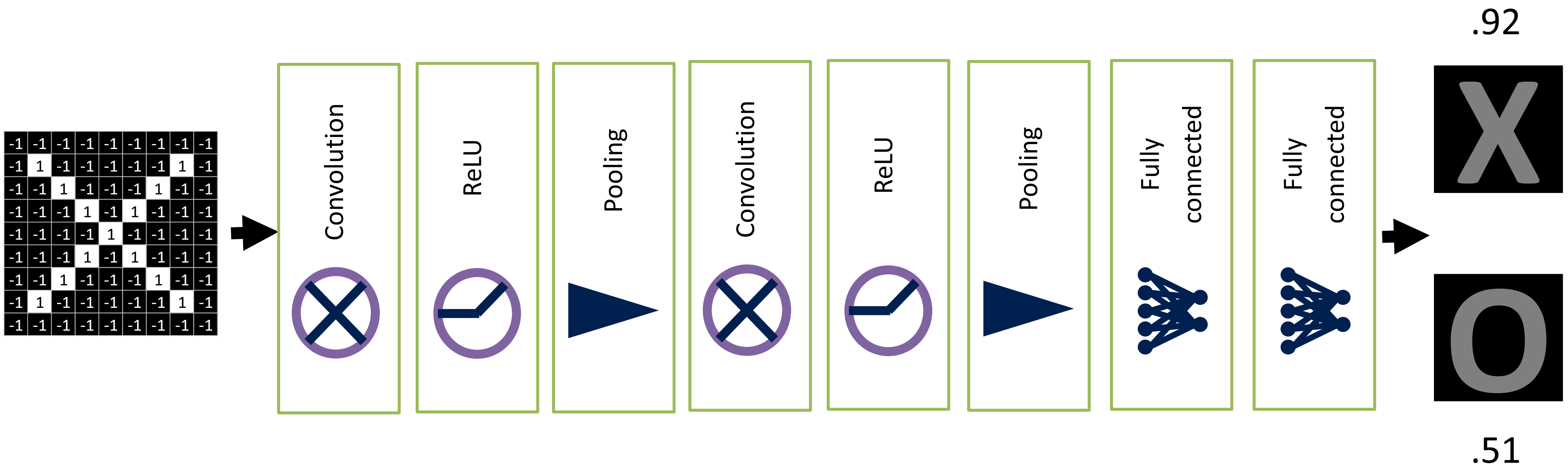


- Fully Connected Layer
  - Flattening
  - Binary classification
  - Cross entropy loss
  - Multiple layers before output



Steps in CNN

- Architecture of CNN



What if the image has 3 dimensions?

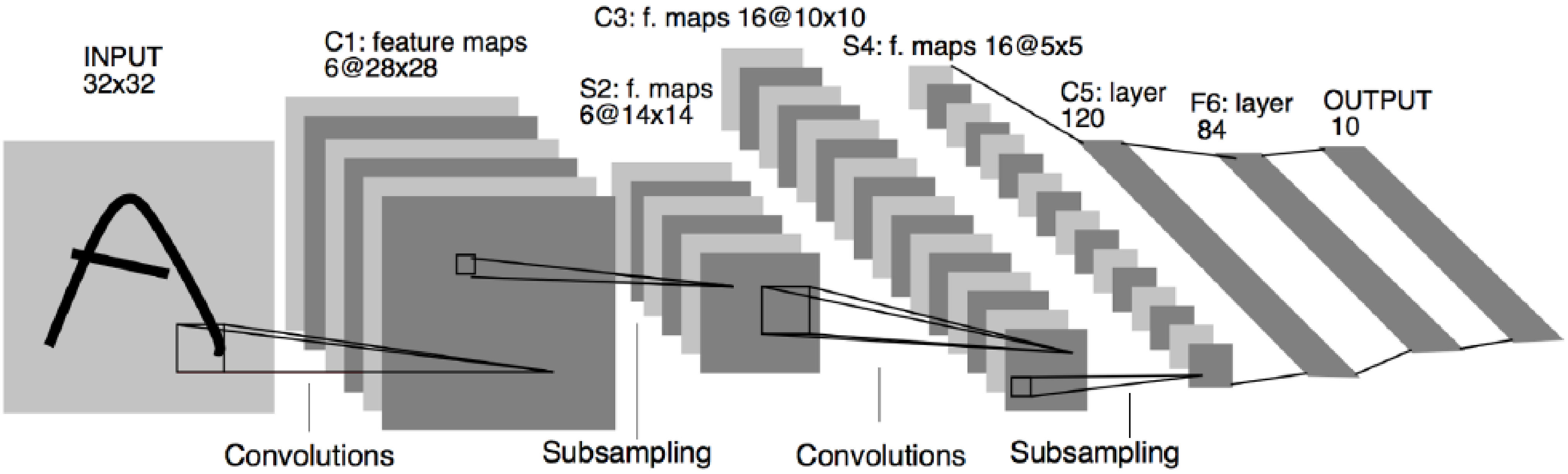
- The kernel becomes a 3-dimensional.

- The output would still be a single value.

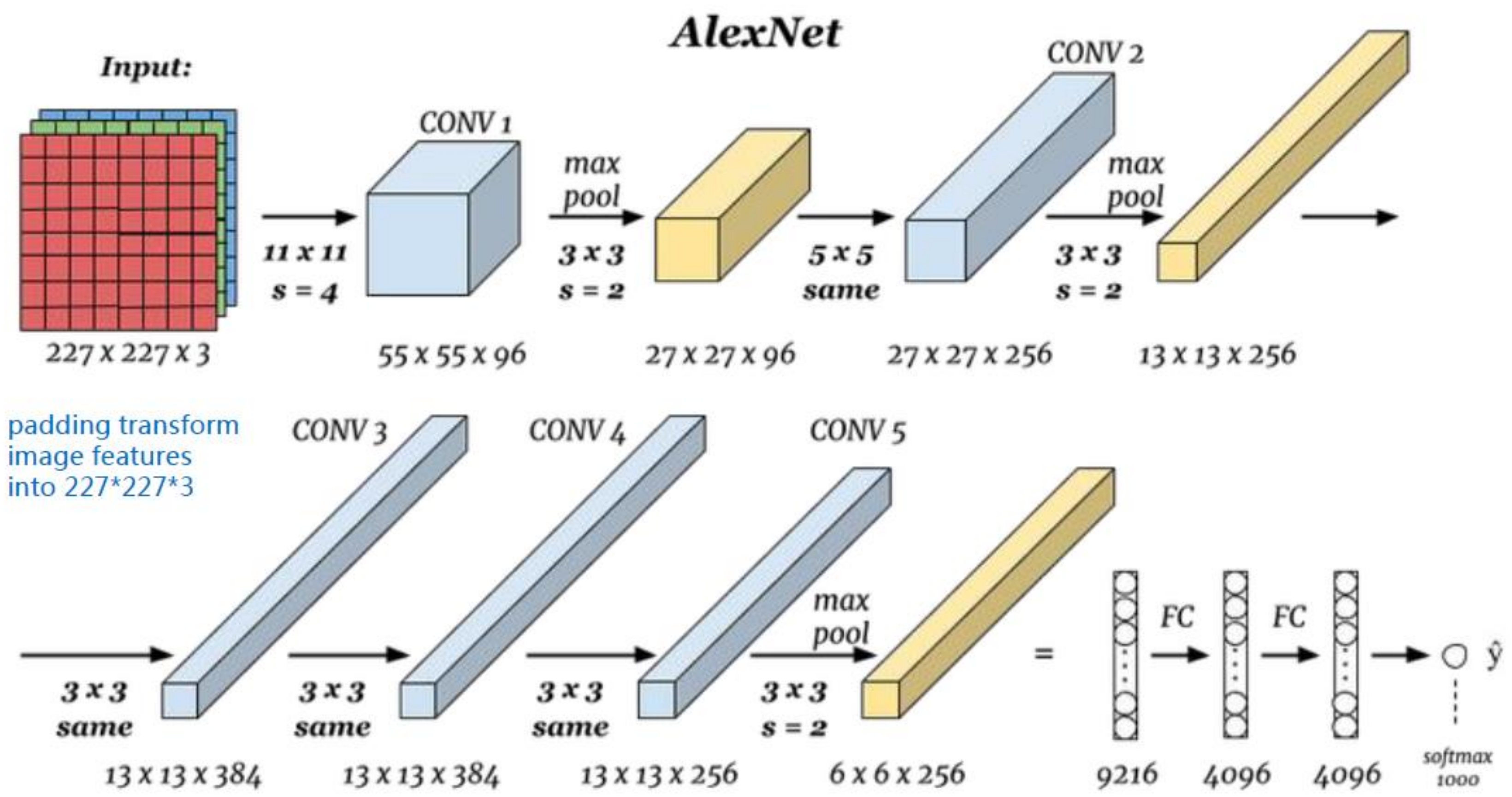
CNN Architectures

- LeNet Architecture: Introduced by Yann LeCunn in 1989
- AlexNet Architecture: Developed at University of Toronto for 2012 ImageNet competition (winning entry).
- GoogLeNet (Inception) Architecture: 2014 ImageNet winning entry.
- VGGNet Architecture: 2014 ImageNet runners up.
- ResNet Architecture: 2015 ImageNet winner.

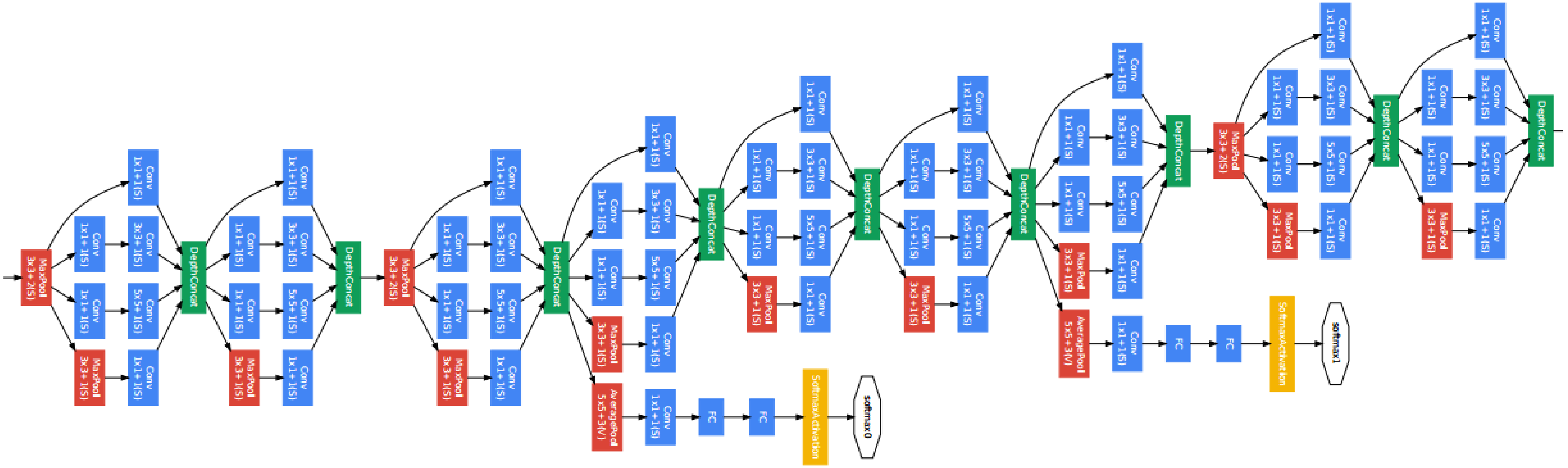
LeNet Architecture



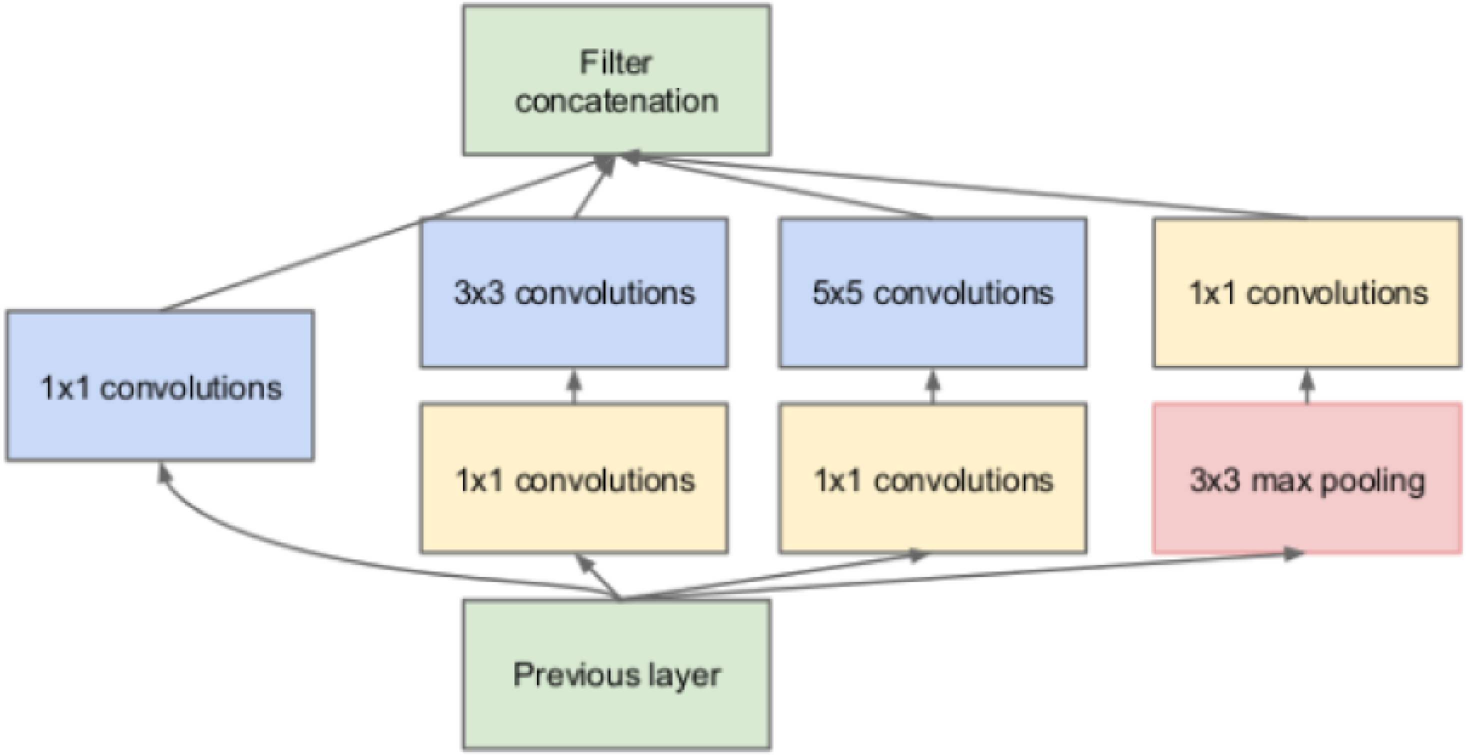
AlexNet Architecture



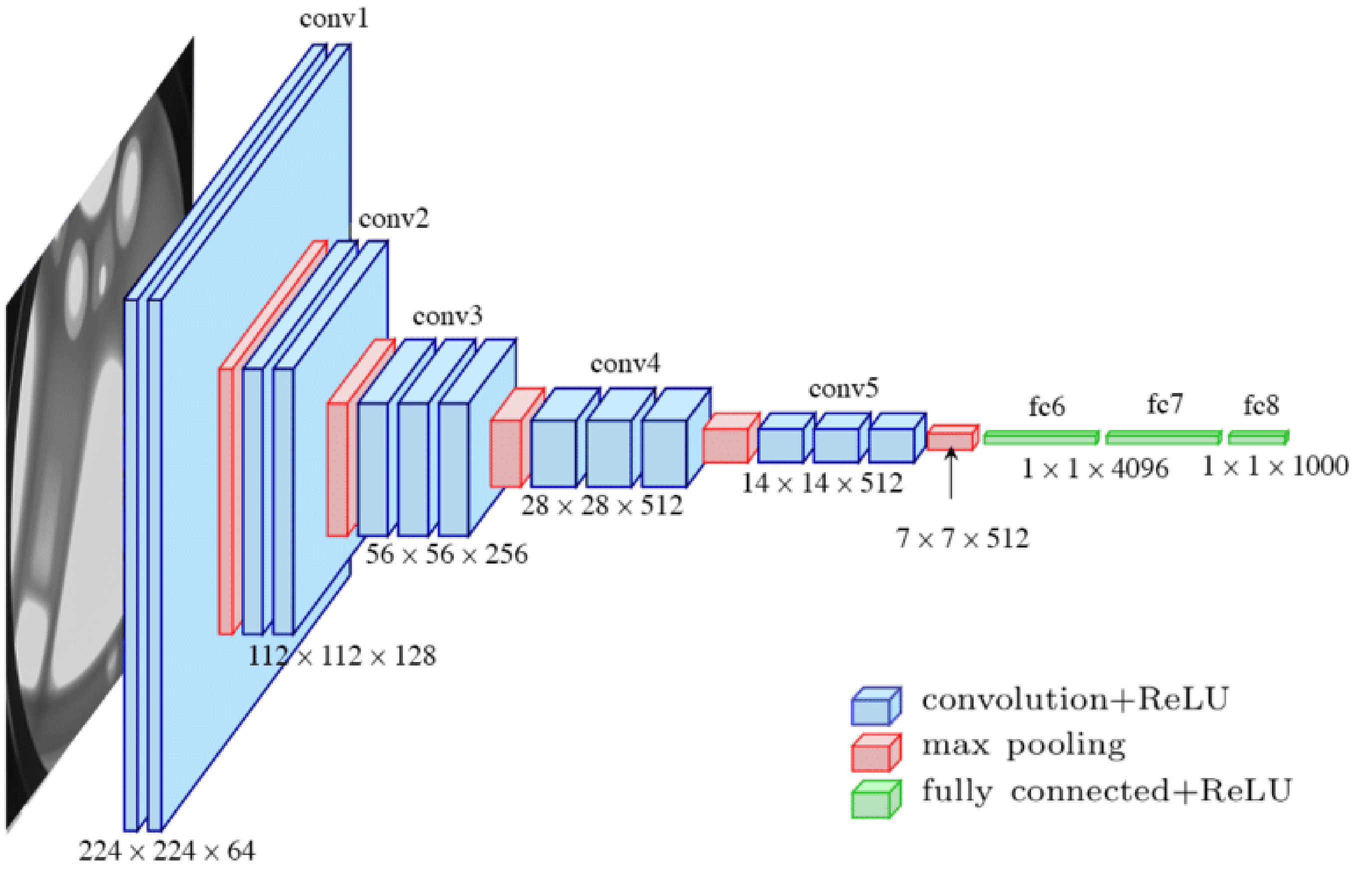
Inception Architecture



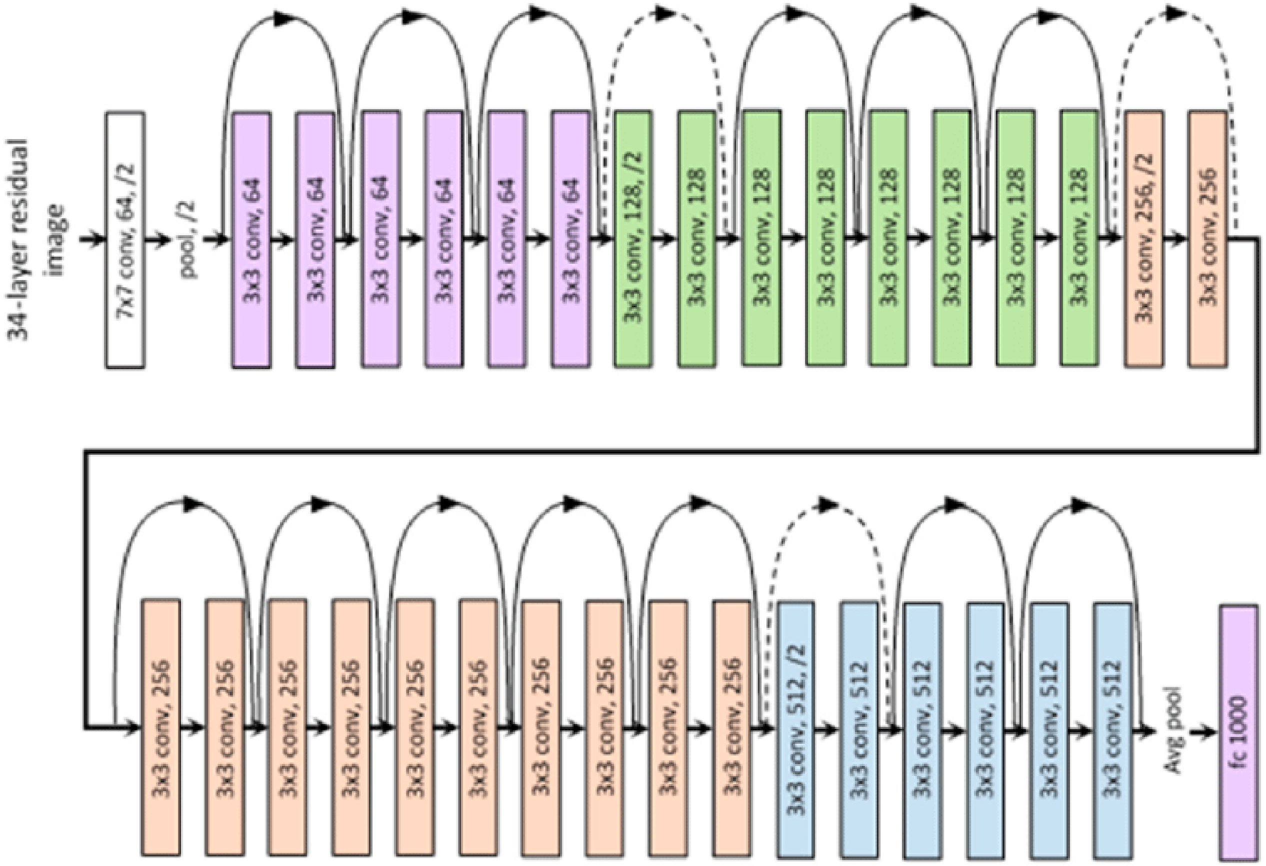
Inception Module

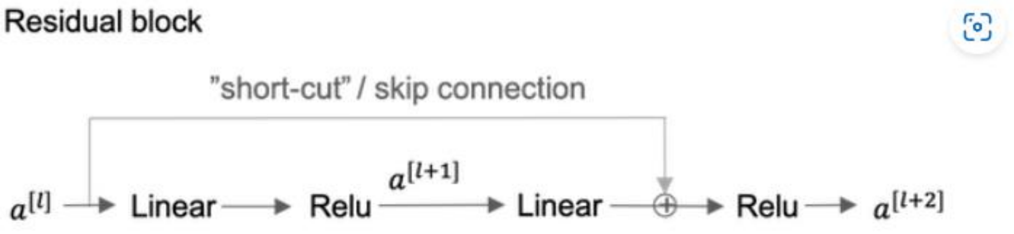


VGGNet Architecture



ResNet Architecture





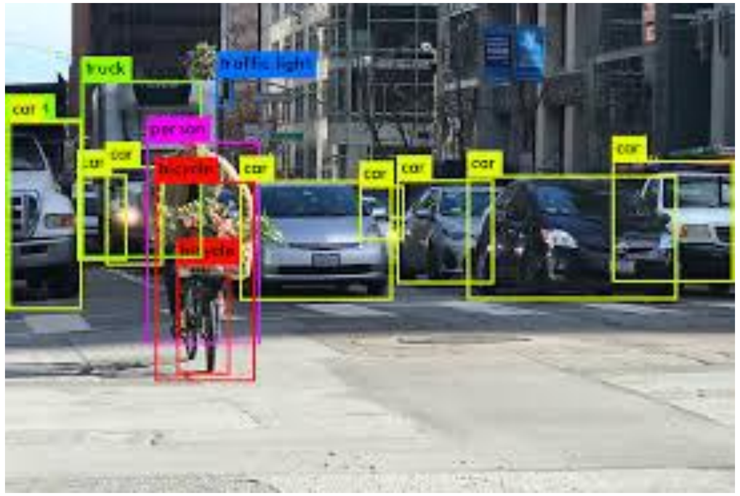
$$z^{[l+1]} = W^{[l+1]}\, a^{[l]} + b^{[l+1]}$$
$$a^{[l+1]} = g(z^{[l+1]})$$
$$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$$
$$\boxed{a^{[l+2]} = g(z^{[l+2]})}$$

$$\Rightarrow$$

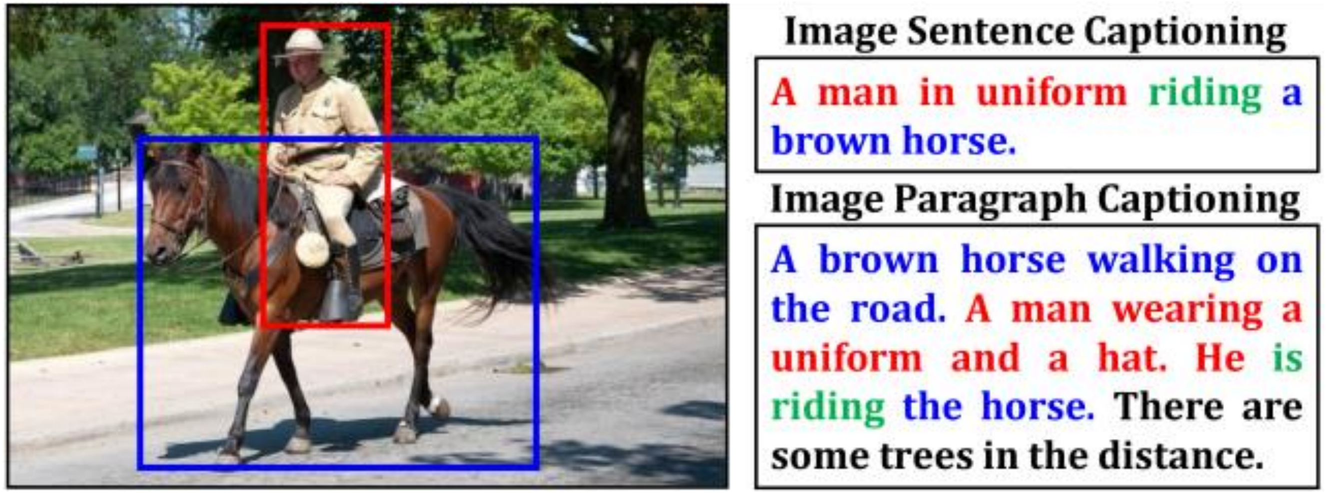
$$z^{[l+1]} = W^{[l+1]}\, a^{[l]} + b^{[l+1]}$$
$$a^{[l+1]} = g(z^{[l+1]})$$
$$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$$
$$\boxed{a^{[l+2]} = g(z^{[l+2]} + a^{[l]})}$$

Other applications

- **Object detection**



- **Image captioning**



Autoencoders

Motivation

- What if we don’t have labels, how to train the models then?
- Unsupervised Learning:
  - We try to learn some structure and pattern in the data.
  - Cluster the data that have similar features.
  - Goal: To learn representations of input data (latent space).

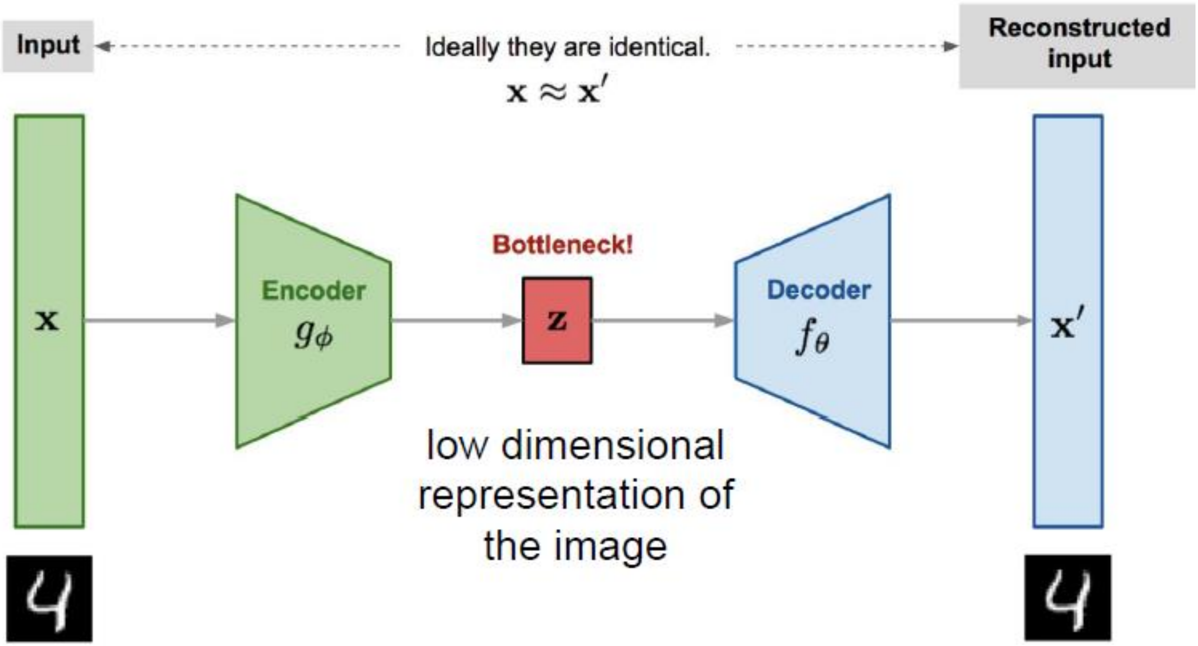
Applications

- Feature Extraction
- Dimensionality Reduction
- Image Denoising
- **Anomaly Detection**

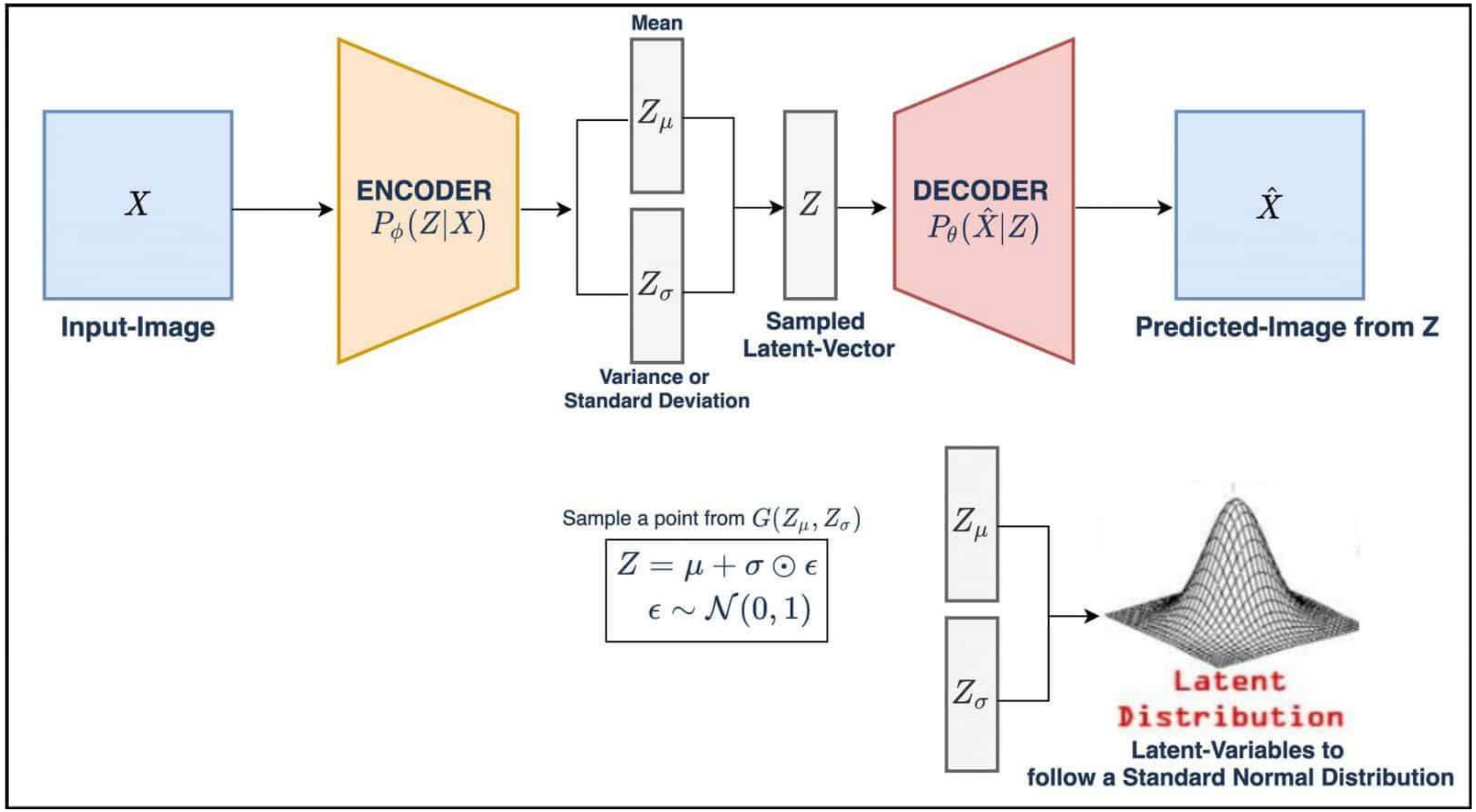
Components

- **Encoder:** Performs dimensionality reduction and creates a latent space.

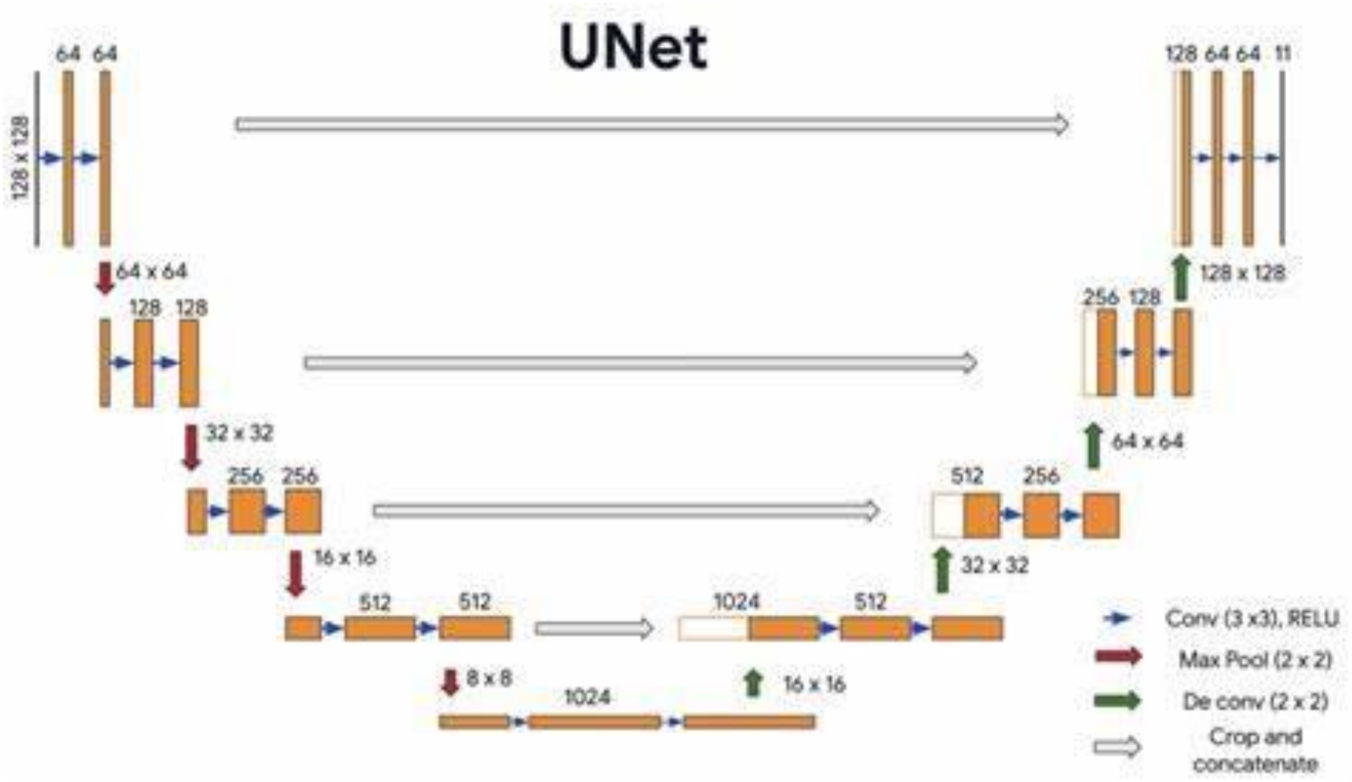
- **Decoder:** Reconstructs the input using the latent space.



Architectures: Variational Autoencoders



Architectures: U-Net Architecture



Recurrent Networks

Motivation

- Can we use the previous architectures for sequential inputs?
- Networks should be able to remember the data over time.
- Networks should handle varying input length.

Examples

- Speech Recognition
- Music Generation
- Sentiment Classification
- DNA Sequence analysis
- Machine translation
- Named Entity recognition

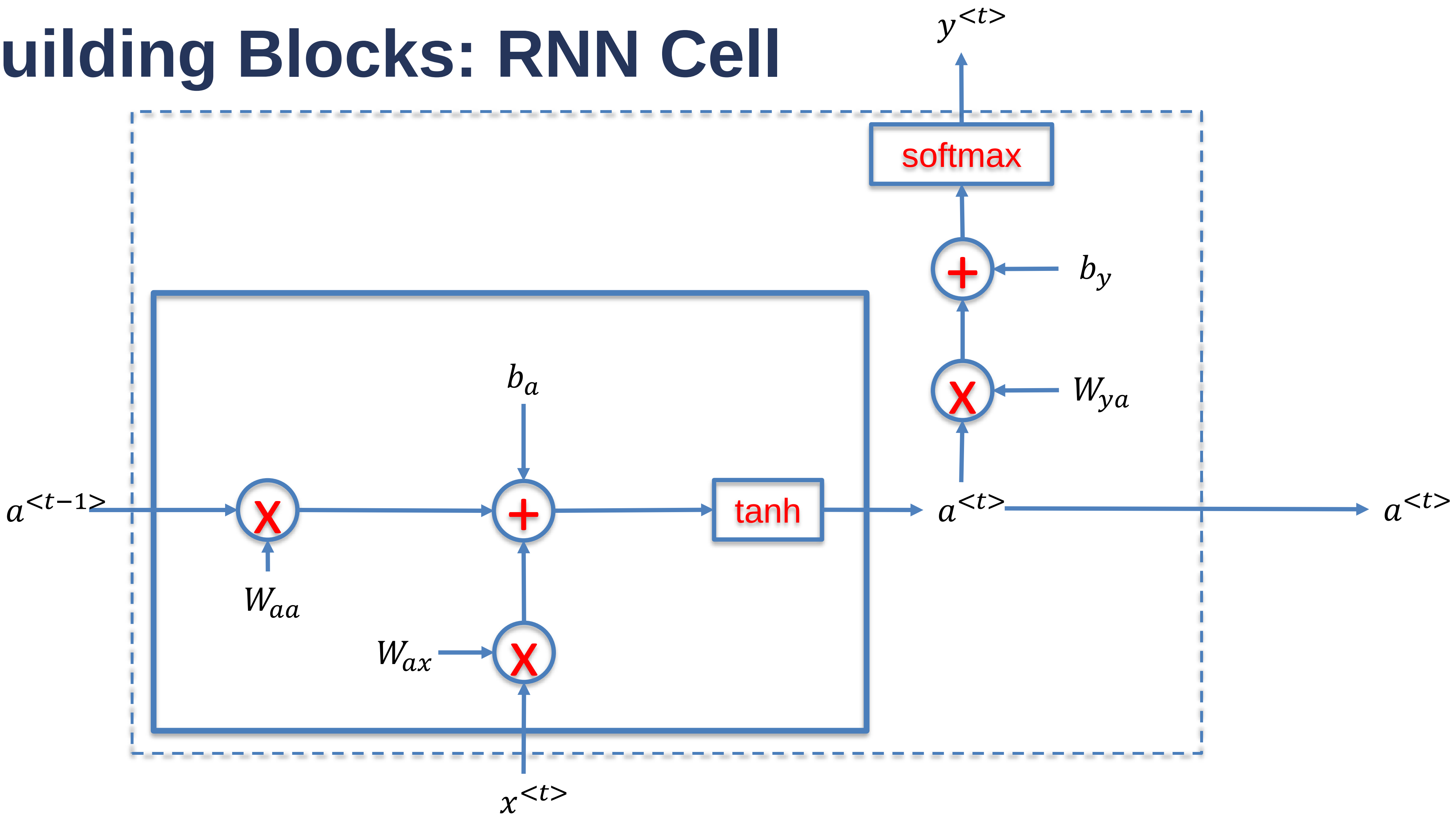
Notation

- Example: Named Entity Recognition
  - Input: Harry Potter and Hermione Granger invented a new spell.
  - Output: 1 1 0 1 1 0 0 0 0
  - $x^{<t>}$ denotes the input in the index t and $y^{<t>}$ denotes the corresponding output.
  - In this case, the input and the output length is the same.

Notation

- Example: Sentiment Classification
  - Input: The current government is not working to benefit its public.
  - Output (0 or 1): 0
  - $x^{<t>}$ denotes the input in the index t, and y denotes the corresponding output.
  - In this case, we only have a single output (i.e., the sentiment).

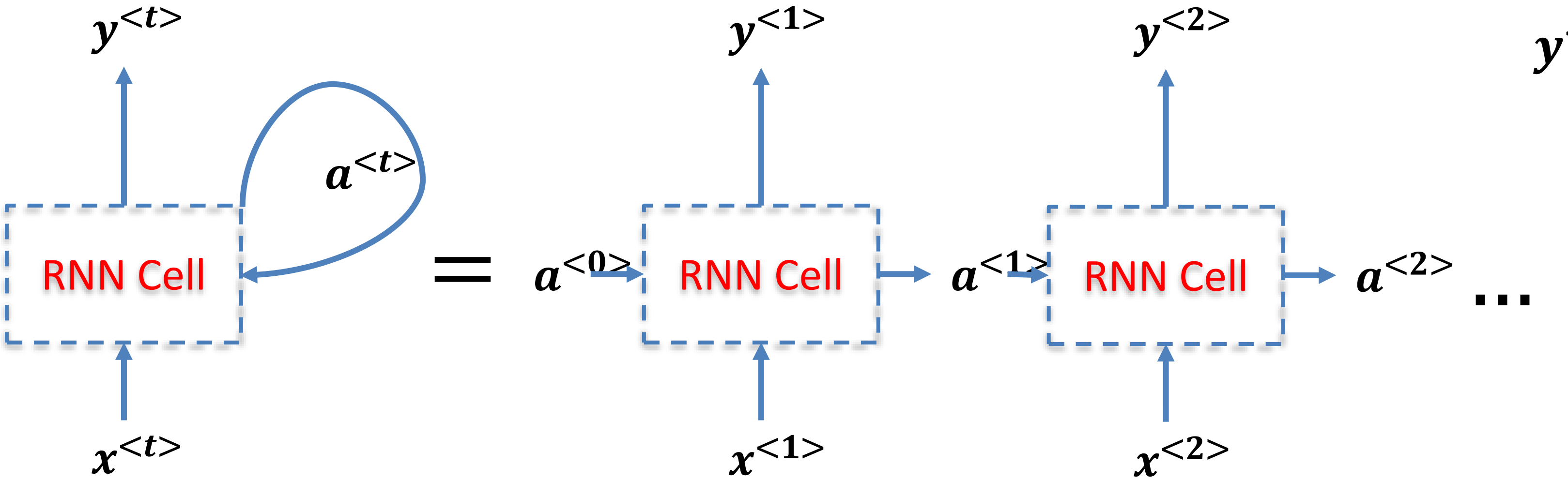
Building Blocks: RNN Cell

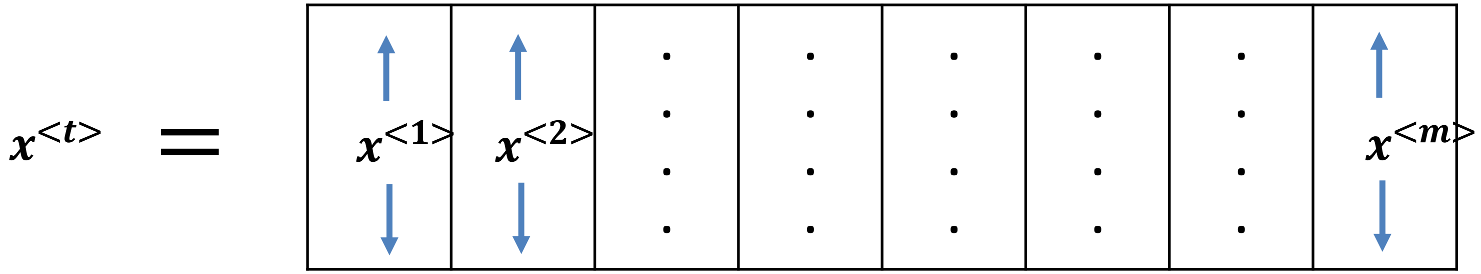


Building Blocks: Unfold and Parameter Sharing

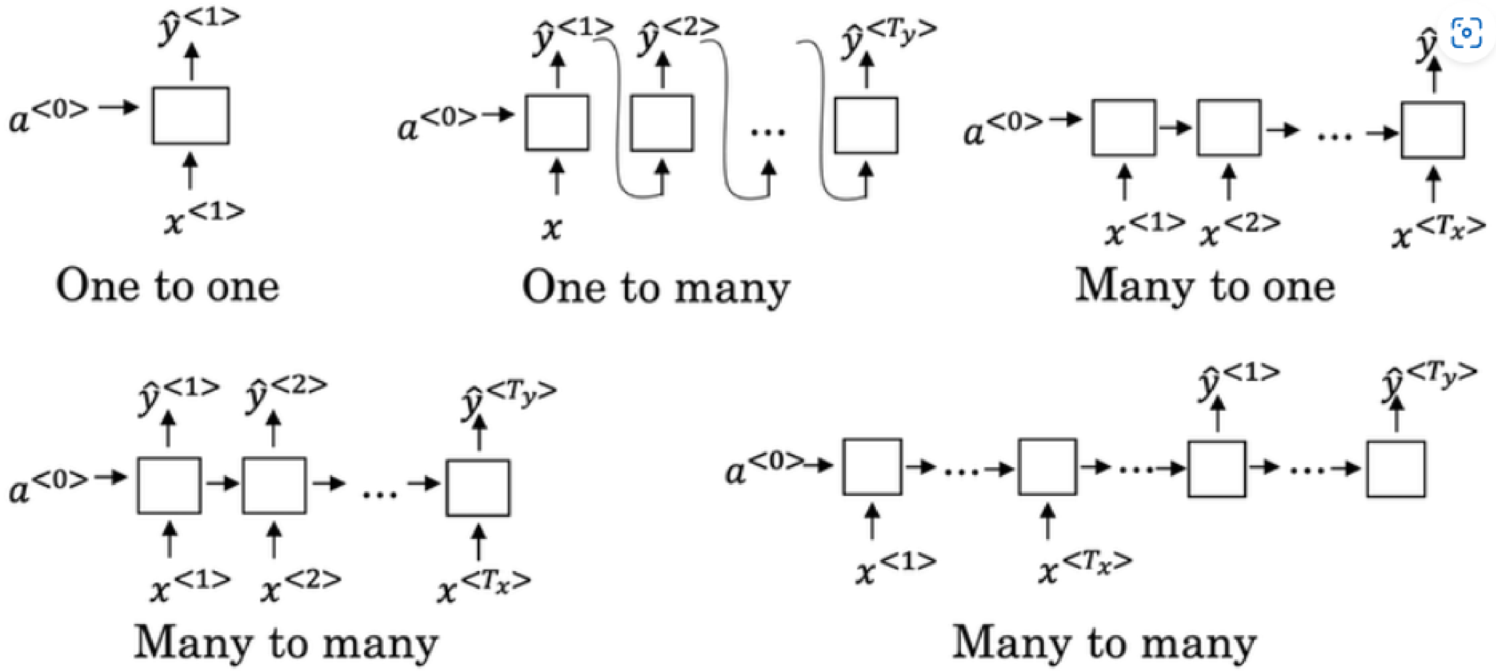
$$a^{<t>} = \tanh\left(W_{ax}x^{<t>} + W_{aa}a^{<t-1>} + b_a\right)$$

$$y^{<t>} = softmax(W_{ya}a^{<t>} + b_y)$$





Different types of RNNs

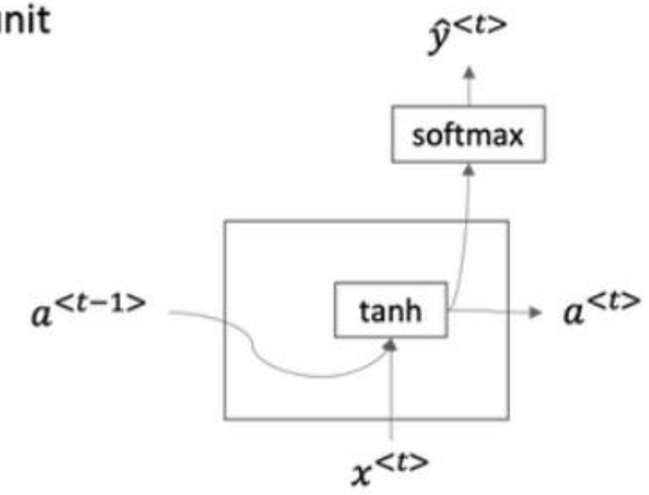


Challenges with RNNs

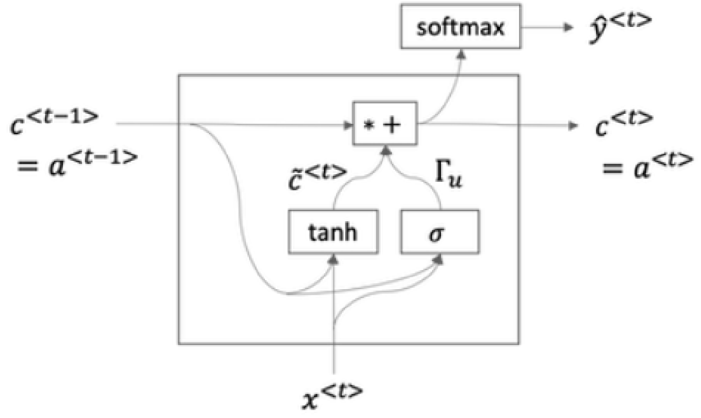
- Vanishing Gradient Problems
- Not good at capturing very long-term dependencies.
- It is possible to solve exploding gradient problems using the technique known as gradient clipping. But it is highly challenging to deal with vanishing gradients.

Gated Recurrent Units

RNN unit



GRU



$$\tilde{c}^{<t>} = \tanh(W_c[c^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[\, c^{<t-1>}, x^{<t>}] + b_u)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

Full GRU:

$$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[\, c^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_r = \sigma(W_r[\, c^{<t-1>}, x^{<t>}] + b_r)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

Long Short Term Memory (LSTM)

LSTM units

GRU	LSTM
$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$	$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$
$\Gamma_u = \sigma(W_u[\, c^{<t-1>}, x^{<t>}] + b_u)$	(update) $\Gamma_u = \sigma(W_u[\, a^{<t-1>}, x^{<t>}] + b_u)$
$\Gamma_r = \sigma(W_r[\, c^{<t-1>}, x^{<t>}] + b_r)$	(forget) $\Gamma_f = \sigma(W_f[\, a^{<t-1>}, x^{<t>}] + b_f)$
	(output) $\Gamma_o = \sigma(W_o[\, a^{<t-1>}, x^{<t>}] + b_o)$
$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$	$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$
$a^{<t>} = c^{<t>}$	$a^{<t>} = \Gamma_o * \tanh(c^{<t>})$

Long Short Term Memory (LSTM)

LSTM

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

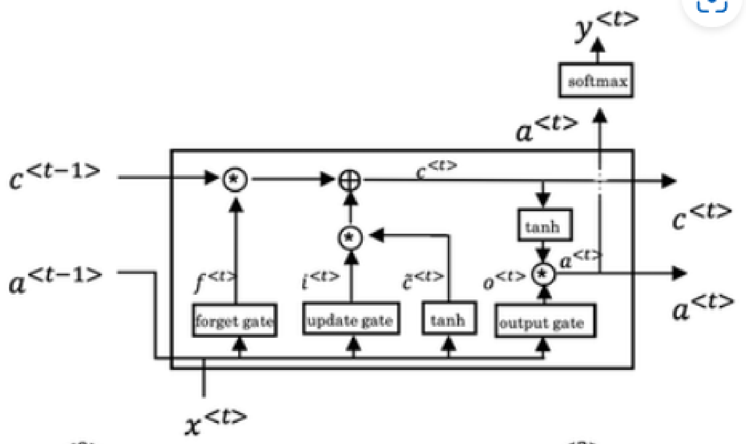
$$\Gamma_u = \sigma(W_u[\, a^{<t-1>}, x^{<t>}] + b_u)$$

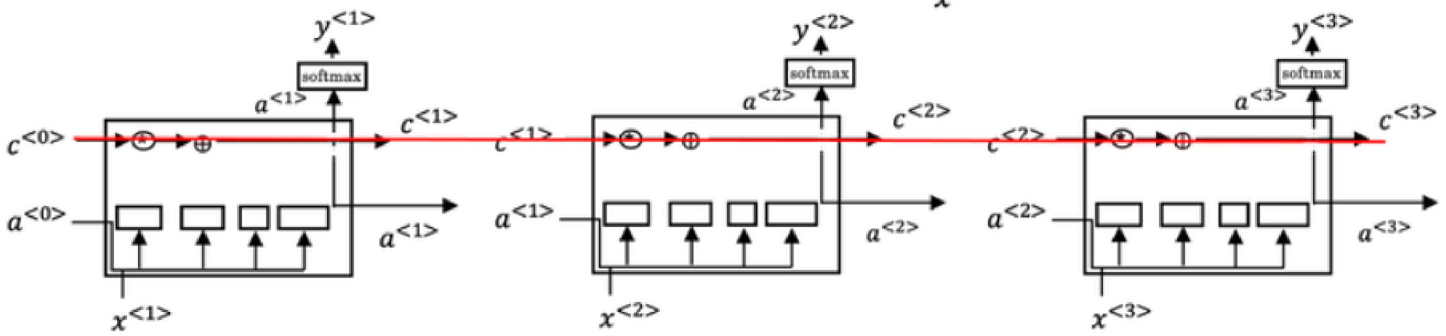
$$\Gamma_f = \sigma(W_f[\, a^{<t-1>}, x^{<t>}] + b_f)$$

$$\Gamma_o = \sigma(W_o[\, a^{<t-1>}, x^{<t>}] + b_o)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * \tanh(c^{<t>})$$





UNIVERSITY OF TORONTO | Engineering

Thank you.

cmore@mie.utoronto.ca
http://cmore.mie.utoronto.ca