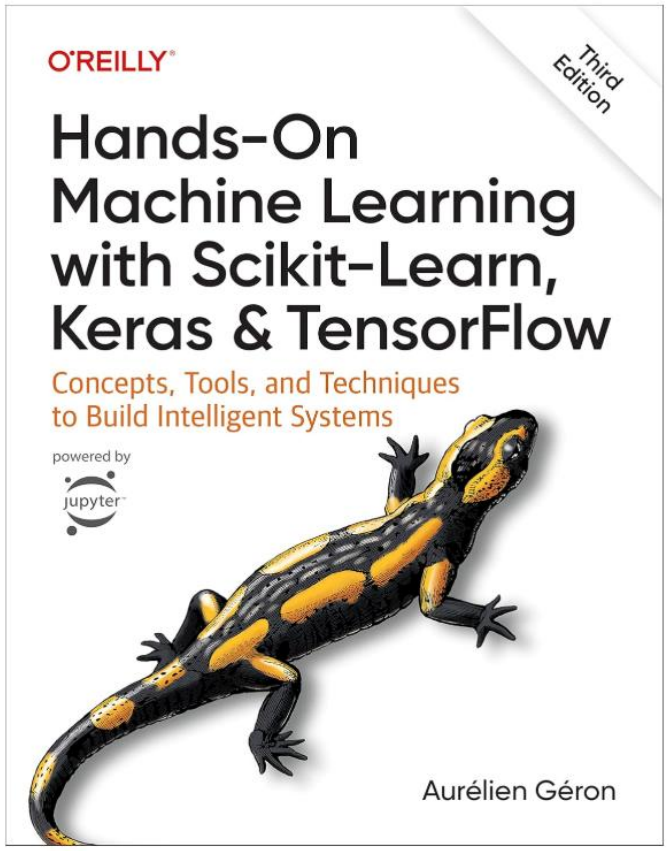
University of BRISTOL

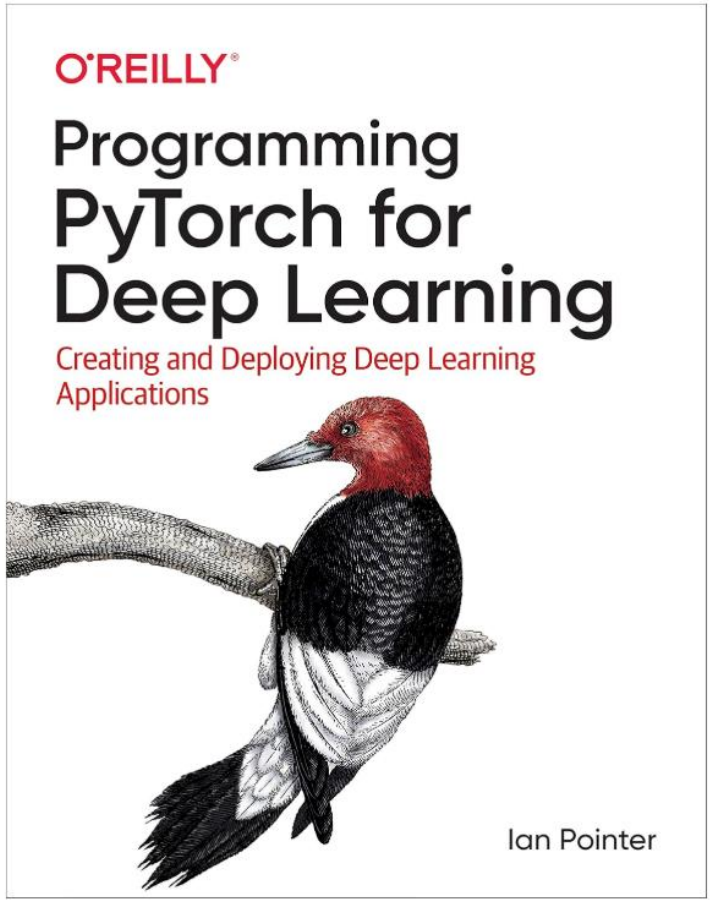
NEURAL NETWORKS CONTINUED

Introduction to AI and Text Analytics

bristol.ac.uk

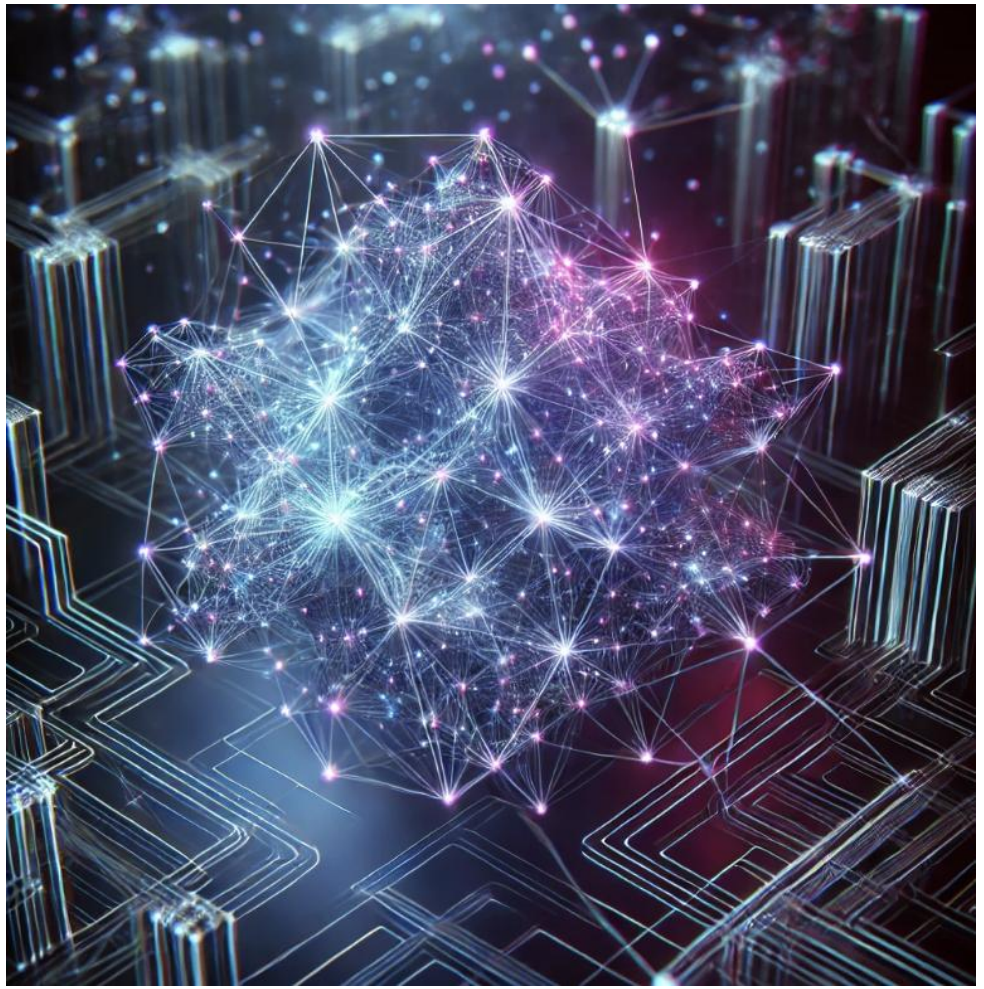
Readings


O'REILLY
Third Edition
Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow
Concepts, Tools, and Techniques to Build Intelligent Systems
powered by
jupyter
Aurélien Géron


O'REILLY
Programming PyTorch for Deep Learning
Creating and Deploying Deep Learning Applications
Ian Pointer

What is Deep Learning

- Deep Learning uses **multi-layered** neural networks
- Learns from **large** datasets to make predictions
- Training Deep NNs takes much **long** due to many layers and often requires **GPUs**



When to implement Deep Neural Networks

1. **Non-linear Relationships**: The data has complex, non-linear patterns.
2. **High-dimensional Data**: The data has many features (e.g., images, text).
3. **Large Datasets**: You have vast amounts of data to train on.
4. **Complex Tasks**: Tasks like image recognition, speech recognition, or language translation.

Challenges with Deep Neural Networks

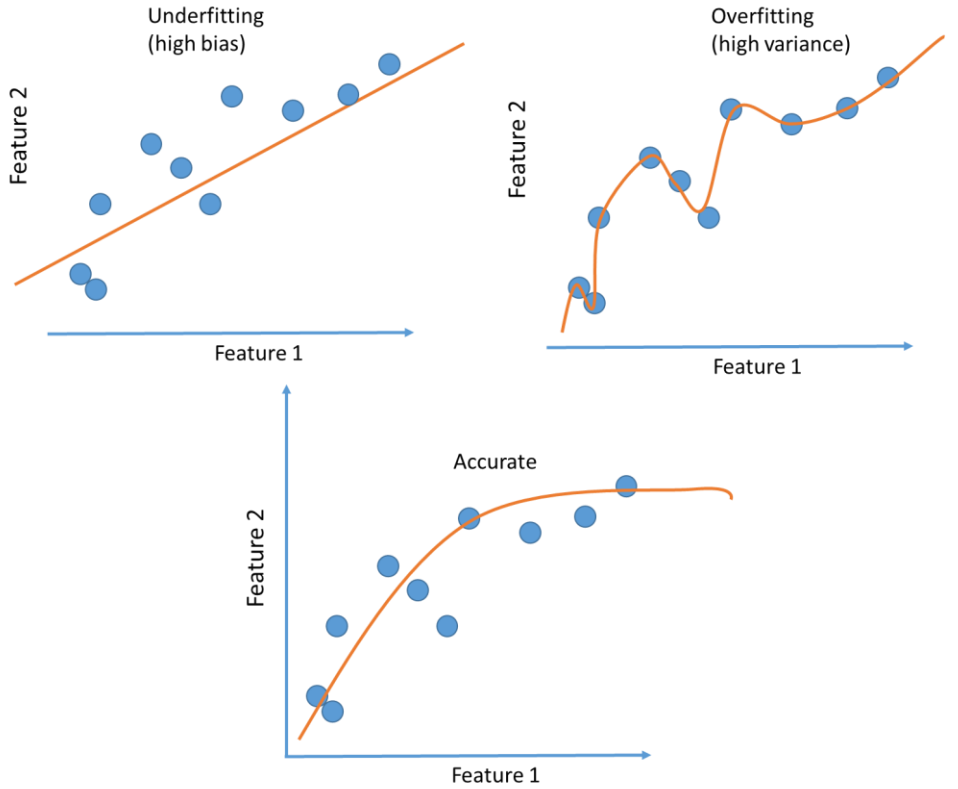
- **Needs a lot of data to train**
- **Overfitting:** The model memorises training data, failing to generalise.
- **Computational Complexity:** Training neural networks requires significant computational resources, especially with large datasets.
- **Neural networks** are often seen as black boxes.

Learning & generalisation

- On one hand, we want to get as **high classification** (or low error) **rate** as possible for the training data set.
- On the other hand, we would also like the trained network to have **good performance** on unseen (**testing**) data.

Low variance, but large bias (error)!

Low bias but large variance! Poor generalisation!



Regularization Techniques

Regularization is a technique used to **prevent overfitting** by adding constraints or penalties to a neural network.

- **Purpose**:
  - Helps models generalize to **unseen data**.
  - Reduces the impact of **noise and irrelevant patterns**.
  - Prevents **overly complex models** from memorizing training data.
- **Why is it needed?**
  - Deep neural networks **tend to overfit** when trained on limited data.
  - Regularization ensures **better generalization performance**.

Regularization Techniques

Dropout

- At each training step, a set percentage of neurons **is temporarily disabled**.
- The network learns **redundant representations**, improving robustness.

- **Pros**:
  - Reduces overfitting.
  - Works well in deep networks.
- **Cons**:
  - Increases training time.

Regularization Techniques

Early Stopping

- **Stops training when validation loss stops improving**, preventing overfitting.
- **How it works**:
  - Monitors the **validation loss** during training.
  - If the loss **stops decreasing**, training **terminates early**.
- **Pros**:
  - Saves computation time.
  - Improves generalization.
- **Cons**:
  - Needs careful monitoring to ensure stopping **at the right time**.

Regularization Techniques

Data Augmentation

- Generates **synthetic training examples** by transforming existing data.
- Used heavily in **computer vision** to improve generalization.
- Common Techniques:
  - **Rotation, flipping, cropping, brightness adjustments**.
  - **Adding noise** to simulate real-world variations.
- Pros:
  - Prevents overfitting in image-based models.
  - Expands dataset size **without collecting more data**.

Strategies to tackle

Underfitting

1. Increase Model Capacity
2. Train the Model Longer
3. Reduce Regularization
4. Improve Data Quality & Features
5. Adjust Training Settings
6. Provide More or Better Data

Overfitting

1. Simplify the Model
2. Improve Training Process
3. Use Regularization
4. Reduce Noise in Labels
5. Use Cross-Validation
6. Get More Training Data

Why NNs Are Seen as a Black Box

- Complex internal structure
- Lack of interpretability
- Distributed decision-making
- Non-transparent training process
- Difficult to explain individual predictions

Where This Matters for Applications

When Black-Box Nature Is Acceptable:

- High-accuracy tasks where interpretability is less critical (e.g., image classification)
- Consumer applications where decisions have low risk (e.g., product suggestions)

When It Becomes a Problem:

- **High-stakes domains** requiring transparency (e.g., healthcare)
- **Regulated environments** Laws may require explainability (e.g., GDPR “right to explanation”).
- **Ethical or fairness concerns** Need to detect bias, justify decisions, and ensure accountability.

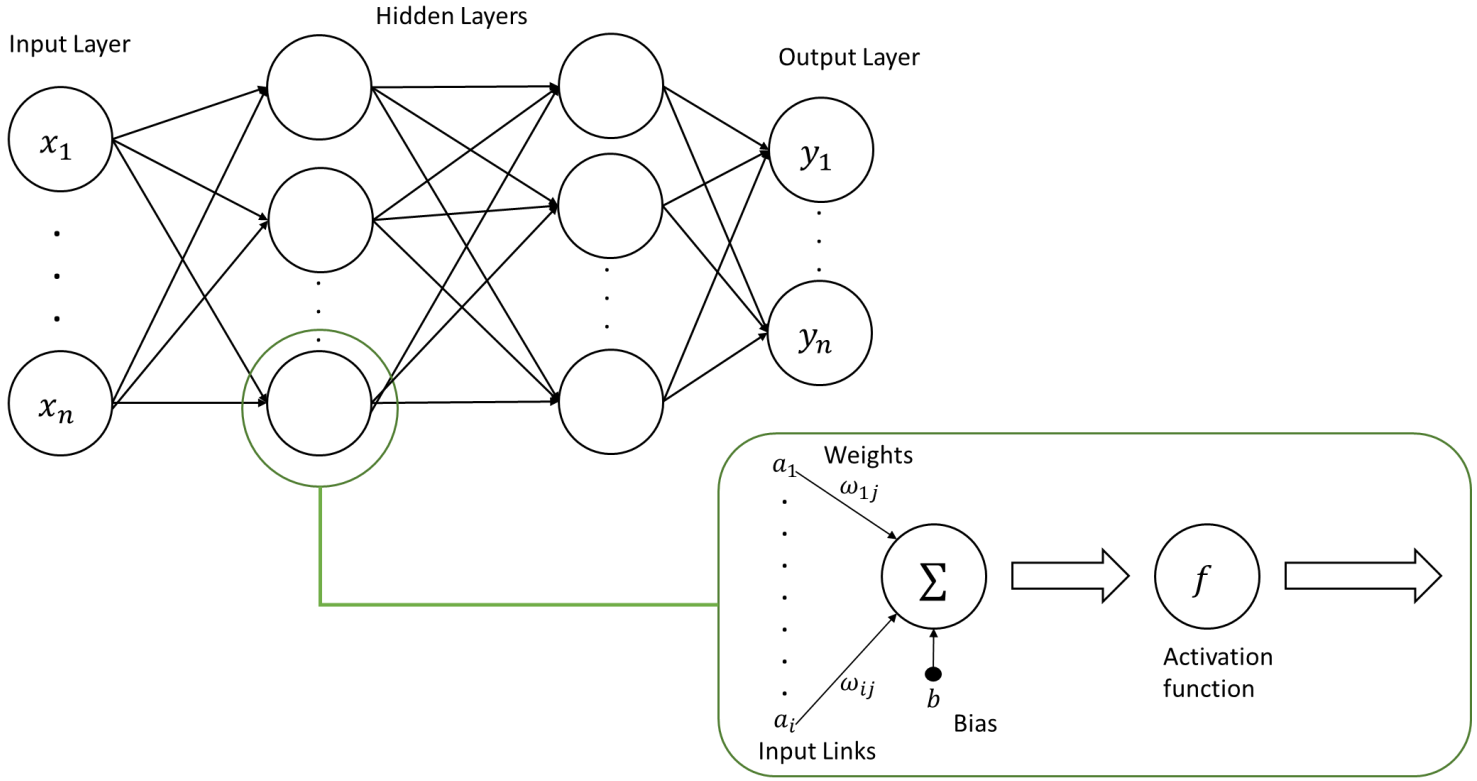
What is Deep Learning

- **Feedforward networks (MLPs)**
- **Convolutional Neural Networks (CNNs)**
- **Recurrent Neural Networks (RNNs)**
  - LSTM
  - GRU
- **Transformers**

Outline

- **Feedforward networks (MLPs) – covered last week**
- **Convolutional Neural Networks (CNNs)**
- **Recurrent Neural Networks (RNNs)**
  - LSTM
  - GRU
- **Transformers – will be covered in text analytics part**

Feed-Forward Neural Networks



Deep NNs have many hidden layers, but they are not always dense layers (fully connected) and often of different kinds

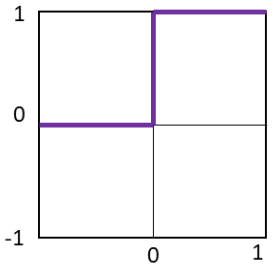
Activation Functions

$$y = f(\sum_{i=1}^{n}(\omega_i x_i + b))$$

Purpose: to introduce non-linearity, allowing the network to learn complex patterns.

Linear

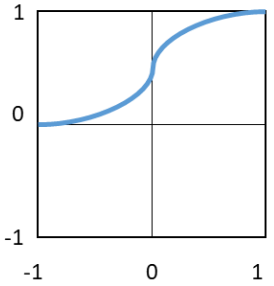
Step



$$f(z) = \begin{cases} 1 \text{ if } z \geq 0 \\ 0 \text{ otherwise} \end{cases}$$

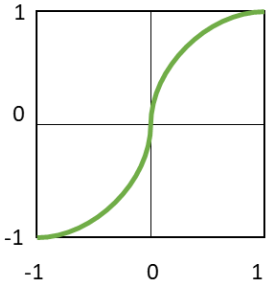
Nonlinear

Sigmoid



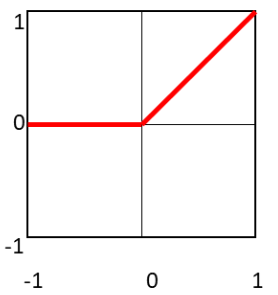
$$f(z) = \frac{1}{1+e^{-z}}$$

Hyperbolic Tangent



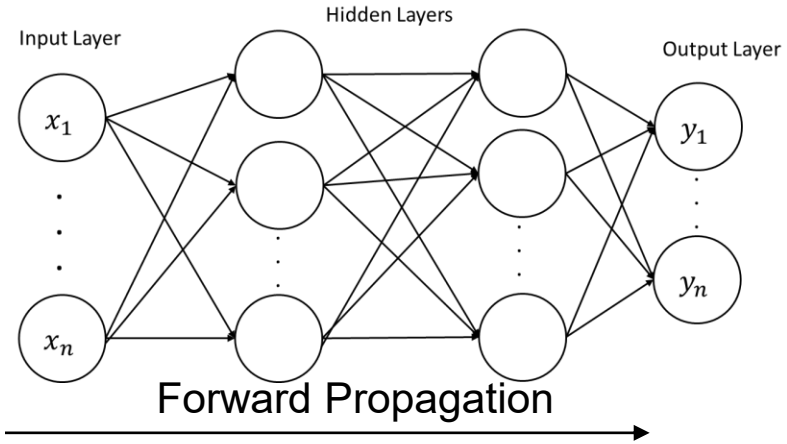
$$f(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

ReLU



$$f(z) = \max(0, z)$$

Forward Propagation

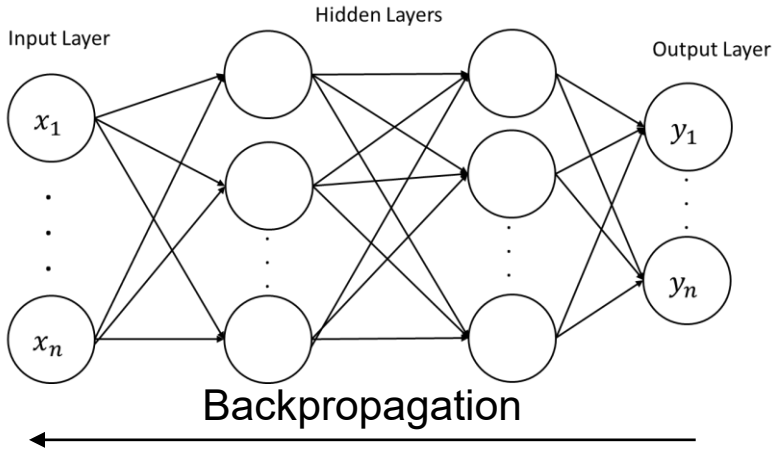


1. Initialise weights: Set the all the initial weights and nodes' biases to small random values (or zeros)
2. For each training sample, calculate the outputs of the hidden and output layers

$$y_k = f_k^0\left(\sum_{j=1}^{n_h}(\omega_{jk}^0 v_j + b_k^o)\right) \qquad v_j = f_j^h\left(\sum_{i=1}^{n}(\omega_{ij}^h x_i + b_j^h)\right)$$

where f_j^h is the activation function applied at hidden layers. (e.g relu) and f_k^0 is the activation function at the output. (depends on if classification or regression)

Backpropagation



4. Calculate error at the output.

5. Propagate the error the back through the network, layer by layer.

6. Adjust the weights/bias of all neurons of each layer to reduce the loss.

7. Repeat steps 2 to 6 until the total error reaches the required level or stopping criteria.

https://www.youtube.com/watch?v=Ilg3gGewQ5U

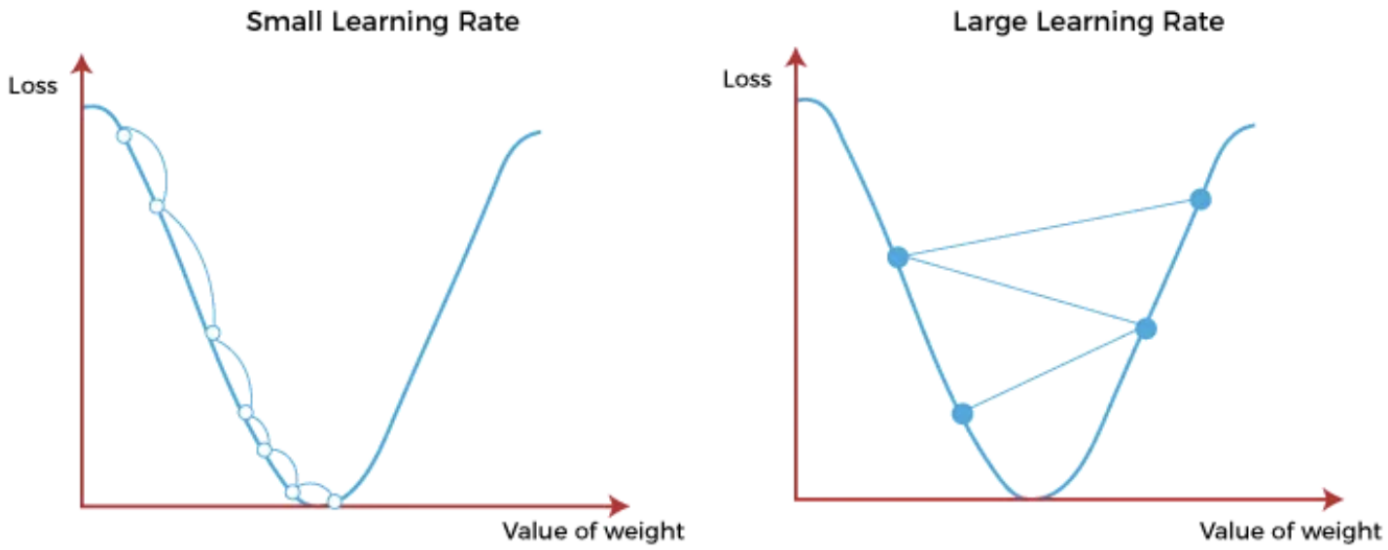
Gradient Descent

Gradient Descent is the most fundamental **optimisation algorithm** in deep learning.

- It calculates the gradient (**derivative**) of the loss function **with respect to the model's parameters** and updates them in the direction that reduces the loss.
- The update rule:
  - **New weight = Old weight - Learning rate × Gradient**
- The learning rate controls how big the update steps are.

Learning Rate

- The **learning rate** is a crucial **hyperparameter** that controls the **step size** during weight updates in gradient descent.
- Determines how **quickly or slowly** a neural network learns from data.
- It **affects model convergence, accuracy, and stability**.



Types of Loss Functions

Regression Loss Functions (Used for predicting continuous values)

- **Mean Squared Error** (MSE): Penalizes larger errors more heavily, useful when large deviations matter.
- **Mean Absolute Error** (MAE): Treats all errors equally, making it more robust to outliers.

Classification Loss Functions (Used for predicting categories)

- **Binary Cross-Entropy:** Used for binary classification tasks like predicting disease vs. no disease.
- **Categorical Cross-Entropy**: Used for multi-class classification problems, such as detecting different tumour types.

University of BRISTOL

CONVOLUTIONAL NEURAL NETWORKS

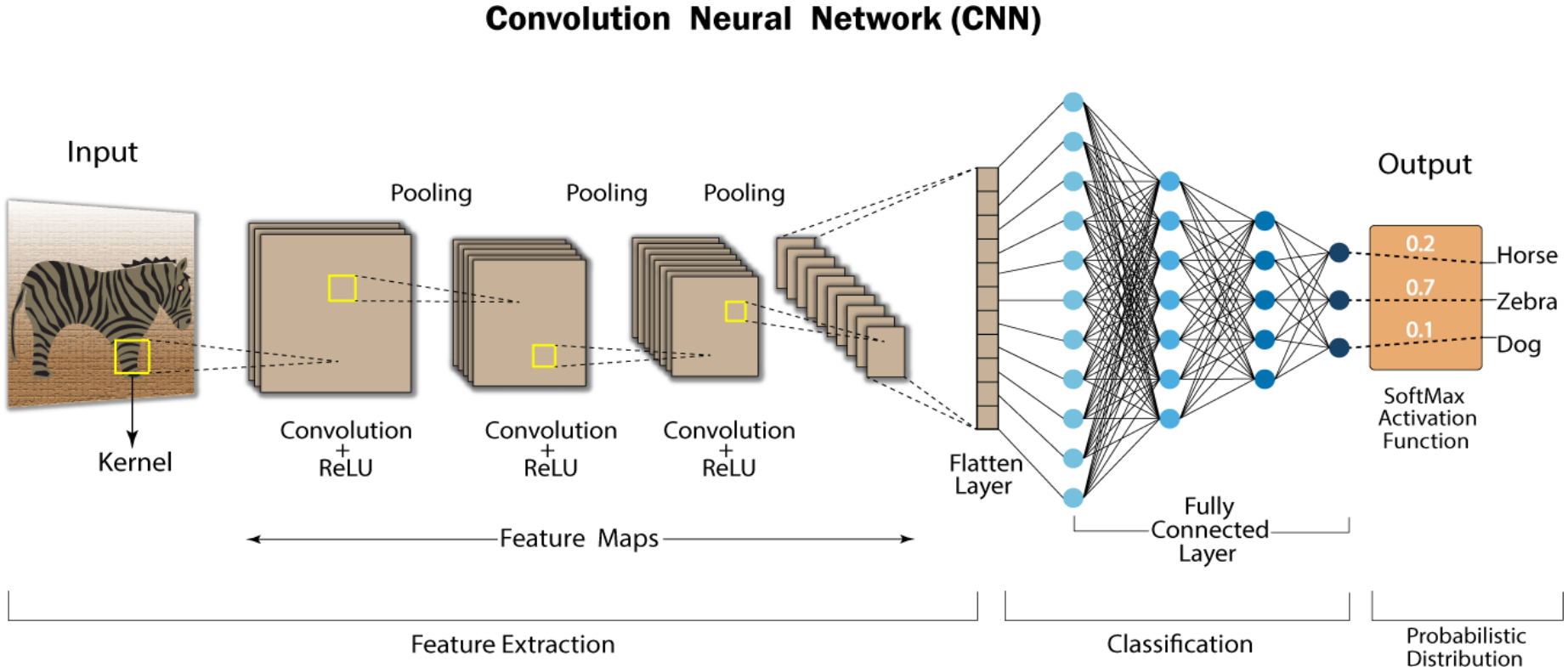
bristol.ac.uk

Why Convolutional Neural Networks?

- Images are high-dimensional and contain spatial structure
- Fully connected networks ignore spatial relationships
- CNNs learn local patterns first, then complex features
- Parameter sharing makes CNNs efficient and scalable
- Foundation of modern computer vision applications

Convolutional Neural Networks (CNN)

- A Convolutional Neural Network (CNN) is a deep learning architecture designed to process structured grid data, such as images.
- CNNs automatically learn spatial hierarchies of features from input data.



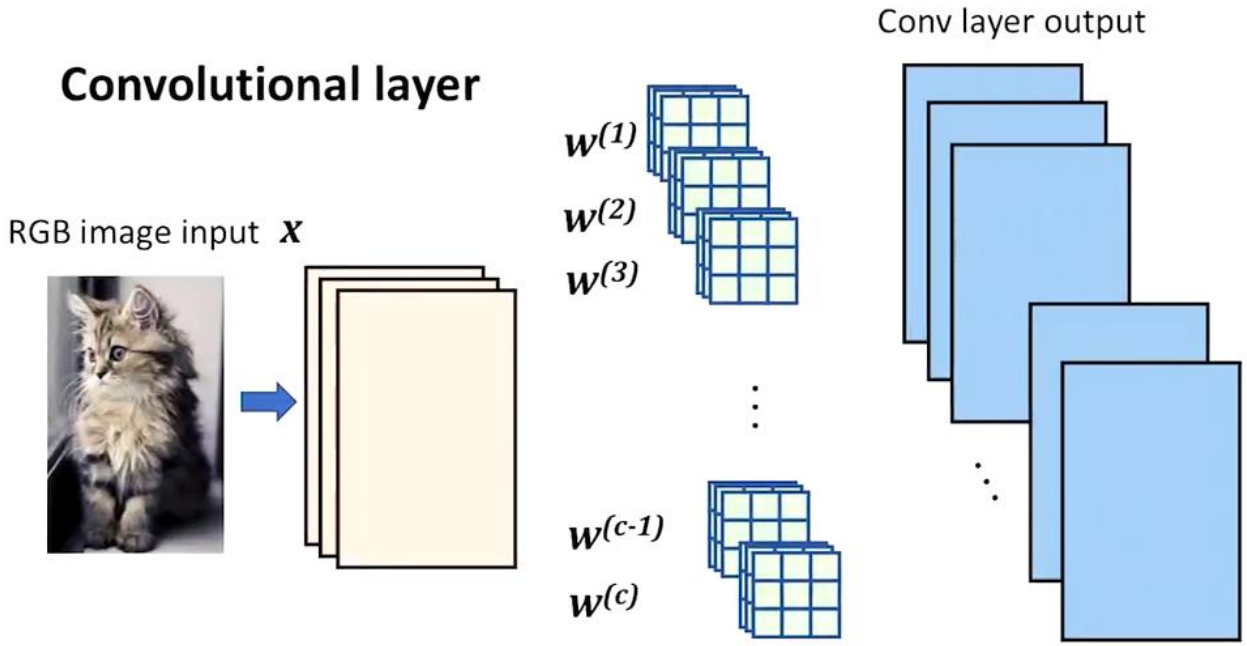
bristol

CNN Architecture Overview

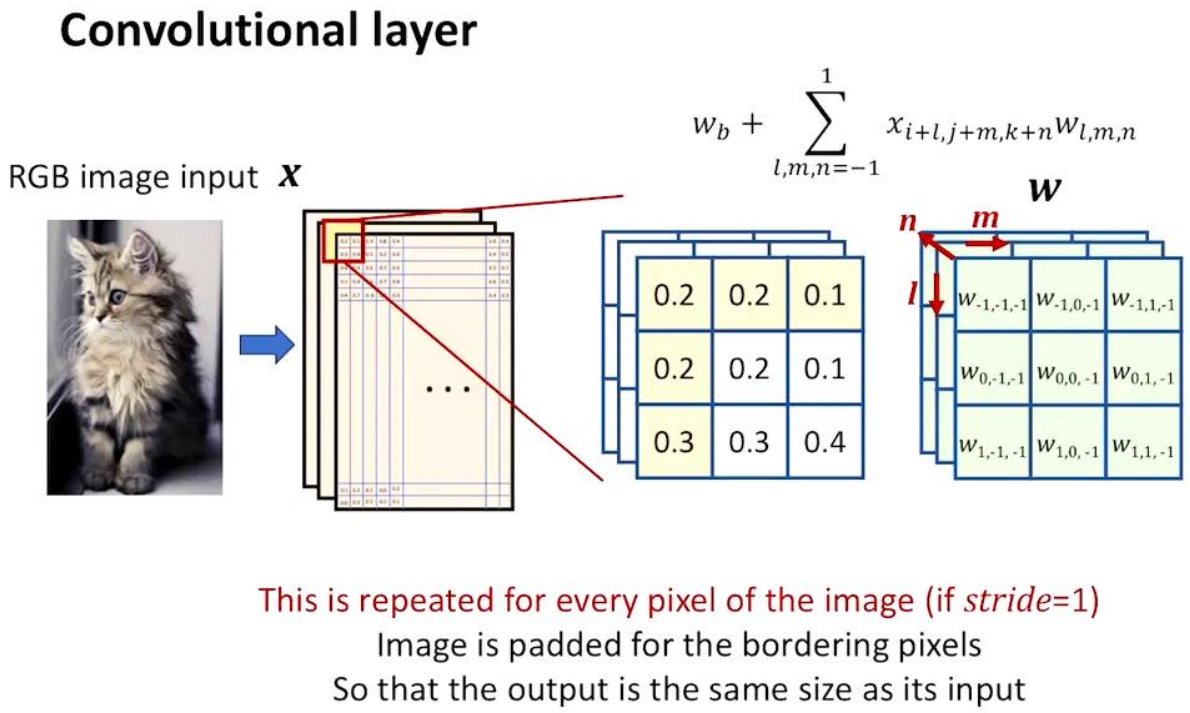
Main Components of CNN:

- **Convolutional Layers:** Extract features using filters/kernels.
- **Pooling Layers:** Reduce dimensionality while retaining key information.
- **Fully Connected Layers:** Flattened feature maps are connected to output neurons.
- **Activation Functions:** Introduce non-linearity for better learning.

Convolutional Layers



Convolutional Layers



This is repeated for every pixel of the image (if *stride*=1)
Image is padded for the bordering pixels
So that the output is the same size as its input

Convolutional Layers

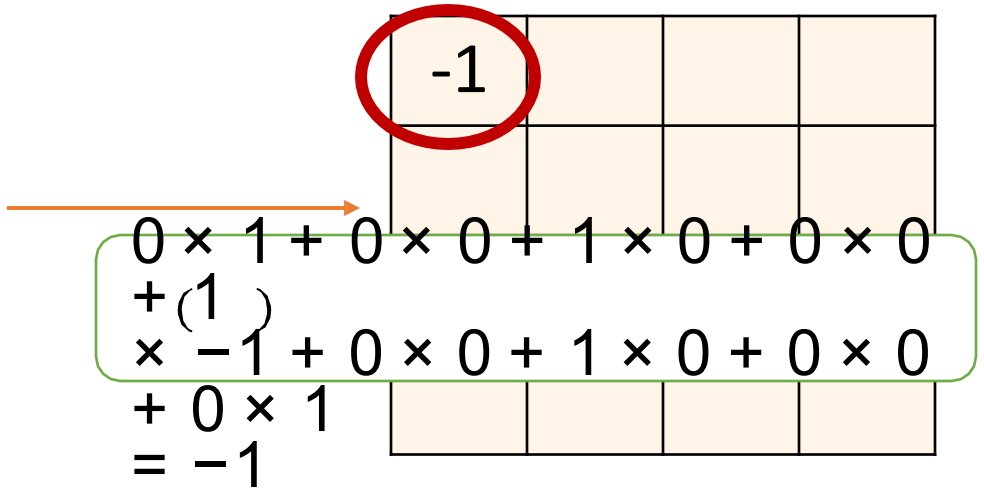
Input Image

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

Kernel

1	0	0
0	-1	0
0	0	1

Feature Map



Convolutional Layers

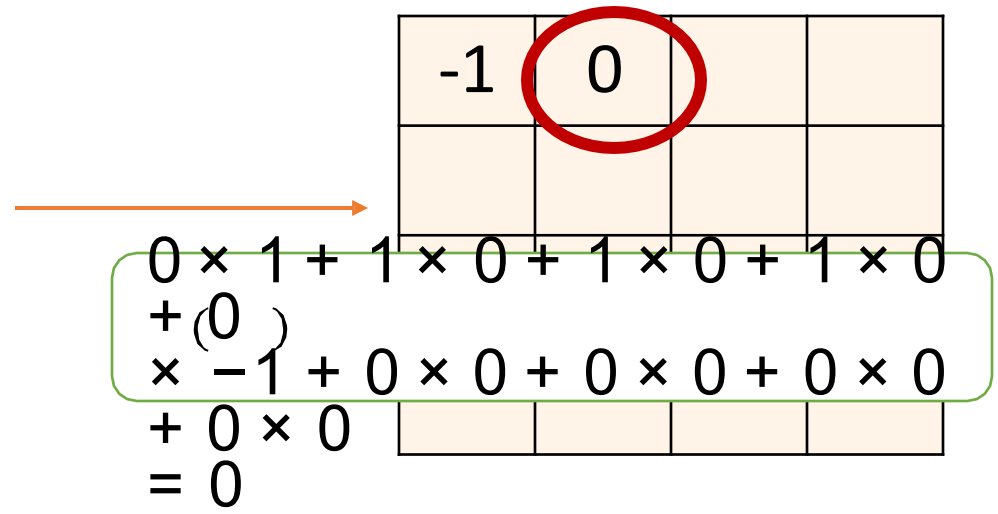
Input Image

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

Kernel

1	0	0
0	-1	0
0	0	1

Feature Map



Convolutional Layers

Input Image

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

Kernel

1	0	0
0	-1	0
0	0	1

Feature Map

-1	0	1	1
0	1	0	1
1	0	1	0
1	1	0	-1

Convolutional Layers

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

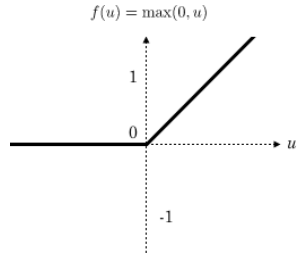
0	-1	0
-1	5	-1
0	-1	0

114				

Pooling Layers

Feature Map

-1	0	1	1
0	1	0	1
1	0	1	0
1	1	0	-1



Feature Map
Post Relu

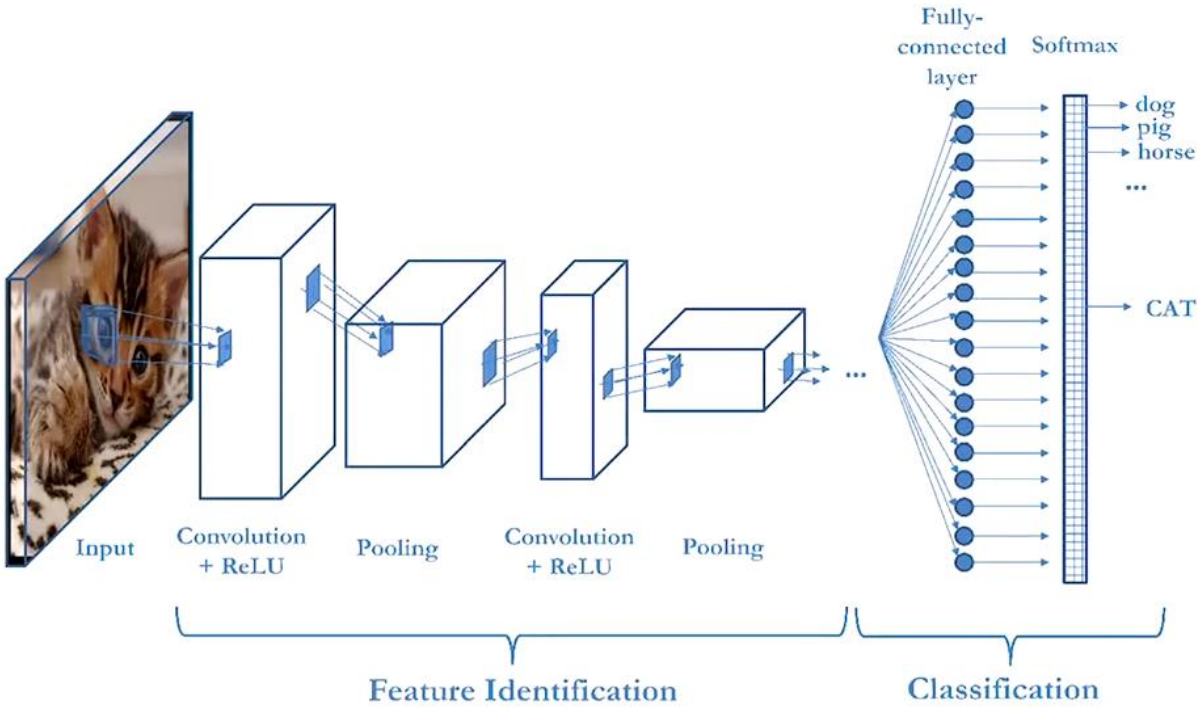
0	0	1	1
0	1	0	1
1	0	1	0
1	1	0	0

Avg pool

0.25	0.75
0.75	0.25

- **Max Pooling** – Select maximum value in the section
- **Min Pooling** – Select minimum value in the section
- **Average Pooling** – Average all values in the section
- **Sum Pooling** – Summing all values in the section

Fully Connected Layer



The generic processing flow of deep convolutional neural network, whose architecture involves transforming input signals through many sequences of locally-connected convolutional, RELU, and pooling nodes, before finally passing them to a fully connected classification layer that assigns finally category labels.

Strengths of CNNs

- **High Accuracy**: They can achieve high accuracy in various image recognition tasks.
- **Efficiency**: They are efficient, especially when implemented on GPUs.
- **Robustness**: They are robust to noise and variations in input data.
- **Adaptability**: It can be adapted to different tasks by modifying their architecture.

Limitations of CNNs

- **Complexity**: It can be complex and difficult to train, especially for large datasets.
- **Resource-Intensive**: It require significant computational resources for training and deployment.
- **Data Requirements**: They need large amounts of labelled data for training.
- **Interpretability**: They can be difficult to interpret making it challenging to understand their predictions.

Applications of CNNs

- Image classification
- Object detection
- Image segmentation
- Video analysis

What Is ImageNet?

- Large-scale visual database for training and benchmarking CNNs
- Contains **14+ million labelled images**
- Organized into **20,000+ categories** (WordNet hierarchy)

Why It Matters

- Enabled deep learning breakthroughs in computer vision
- Foundation of the **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)**
- Drove development of iconic CNN architectures

Impact on the Field

- Standard benchmark for image classification
- Pushed models toward higher accuracy and deeper architectures
- Accelerated adoption of GPUs for deep learning

ImageNet

Popular CNN Architectures

AlexNet (2012)

- First deep CNN to win ImageNet
- Used ReLU, dropout, and GPU training
- Sparked the deep learning revolution in vision

VGGNet (2014)

- Very deep (16–19 layers)
- Simple architecture: stacks of 3×3 convolutions
- Known for clean, uniform design

GoogLeNet / Inception (2014–2015)

- Introduced **Inception modules**
- Efficient: fewer parameters than VGG
- Deeper and wider architecture

ResNet (2015)

- Introduced **residual connections**
- Enabled extremely deep networks (50–152 layers)
- Solved vanishing gradient problem

DenseNet (2017)

- Each layer connects to all previous layers
- Very parameter-efficient
- Encourages feature reuse

Example of the output

Figure: test images and the five labels considered most probable by the AlexNet model

Performance Comparison

Model	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
AlexNet (2012)	~80%	60M	8 layers	~2.0–2.5 ms	~0.7 ms
VGG-16 (2014)	~89.8%	138M	16 layers	~15–20 ms	~4–5 ms
GoogLeNet / Inception v1 (2014)	~89.6%	6.8M	22 layers	~7–9 ms	~2–3 ms
ResNet-50 (2015)	~93%	25.6M	50 layers	~10–12 ms	~1.5–2 ms
ResNet-152 (2015)	~94.1%	60M	152 layers	~20–25 ms	~3–4 ms
DenseNet-201 (2017)	~93.7%	20M	201 layers	~12–15 ms	~2–3 m

Transfer Learning

- **Definition**: Transfer learning is a deep learning technique where a **pre-trained model** is used as a starting point for a new task. Instead of training from scratch, we **fine-tune** an existing model to a new dataset.
- **Why Use Transfer Learning?**
  - **Reduces Training Time** – Models have already learned useful features.
  - **Requires Less Data** – Pre-trained models generalize well to new tasks.
  - **Improves Accuracy** – Beneficial for small datasets.
  - **Works Across Domains** – Can be applied to medical imaging, NLP, and other domains.
- **How Does It Work?**
  - Choose a **pre-trained model** (e.g., ResNet, VGG).
  - **Freeze lower layers** to retain learned features.
  - **Fine-tune upper layers** with the new dataset.
  - Train on **task-specific data** with fewer epochs.

University of BRISTOL

RECURRENT NEURAL NETWORKS

bristol.ac.uk

What Are Recurrent Neural Networks

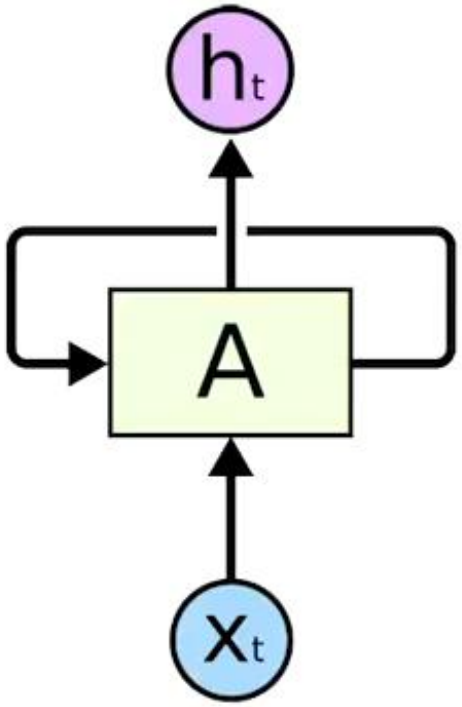
- Neural networks designed for sequential data
- Maintain a hidden state that carries information across time
- Process inputs one step at a time
- Useful when order and context matter

Recurrent Neural Networks

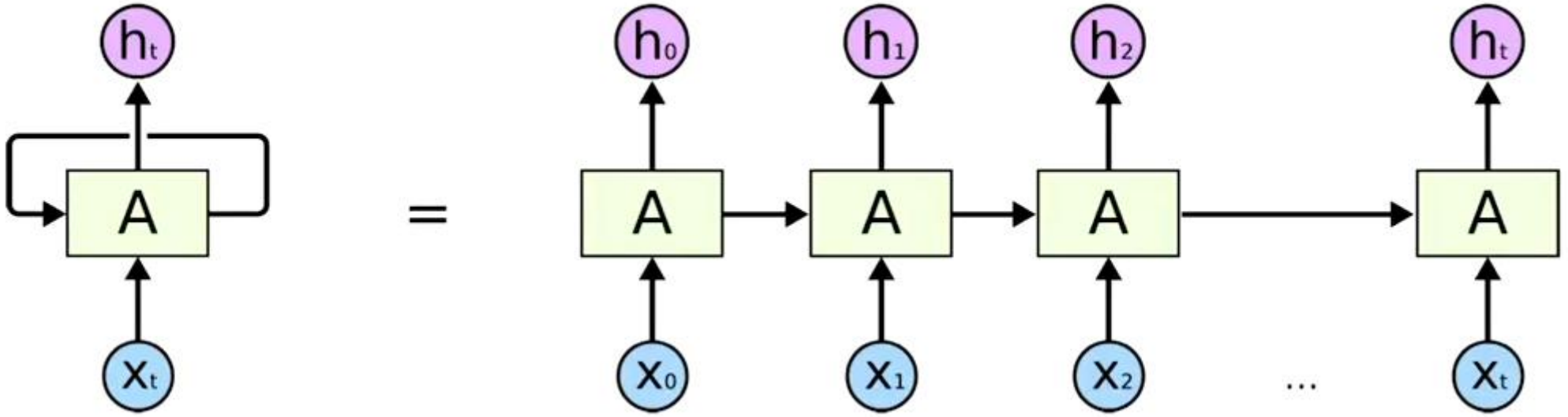
Instead of just feeding forward, the output of the network is fed back in the next timestep

$$h_t = \sigma(A \cdot [h_{t-1}, x_t] + b)$$

where $[\mathrm{h}, \mathrm{x}]$ indicates the concatenation of these vectors

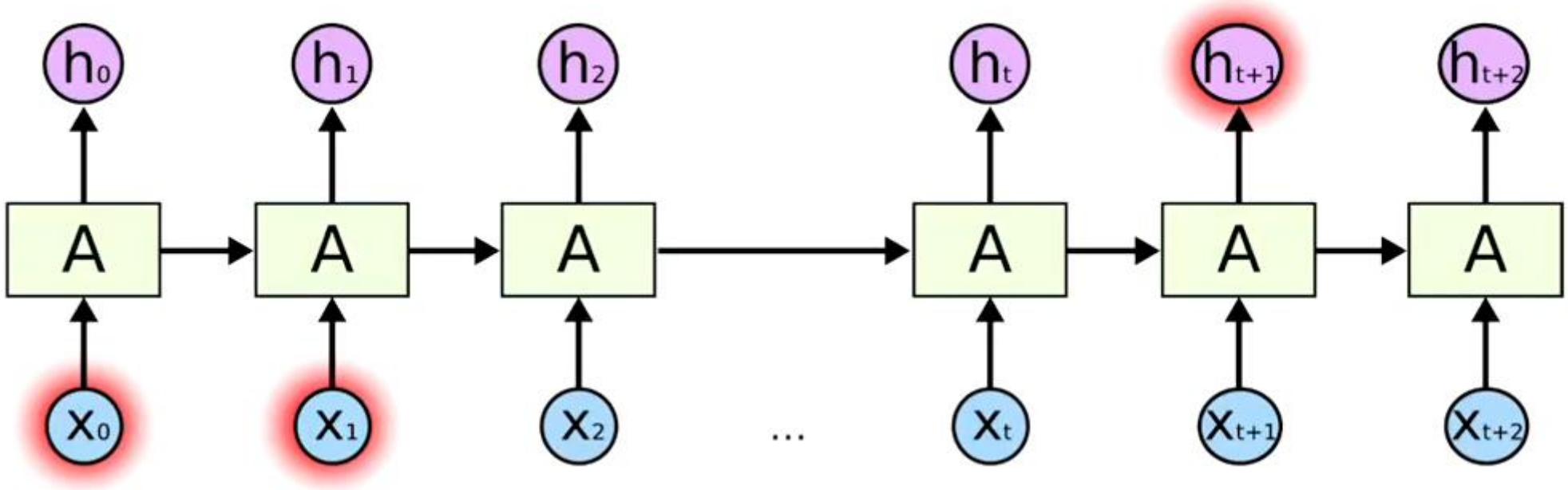


Recurrent Neural Networks



Training via backpropagation through time (BPTT)
Standard propagation keeps weights equal at each layer

Recurrent Neural Networks



When errors backpropagate through a large number of layers, the gradients can become very small

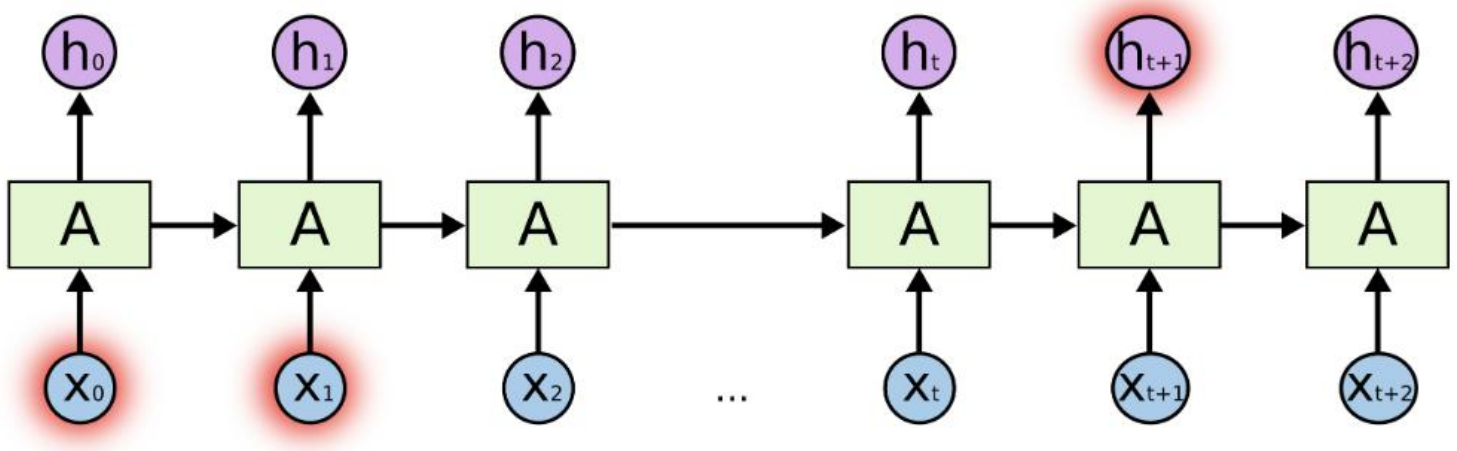
Strengths and limitations of RNNs

- Good for sequential patterns
- Can model variable-length inputs
- Capture temporal relationships

- Vanishing gradients
- Exploding gradients
- Struggle with long-term dependencies
- Training becomes unstable for long sequences

Long Short-Term Memory (LSTM)

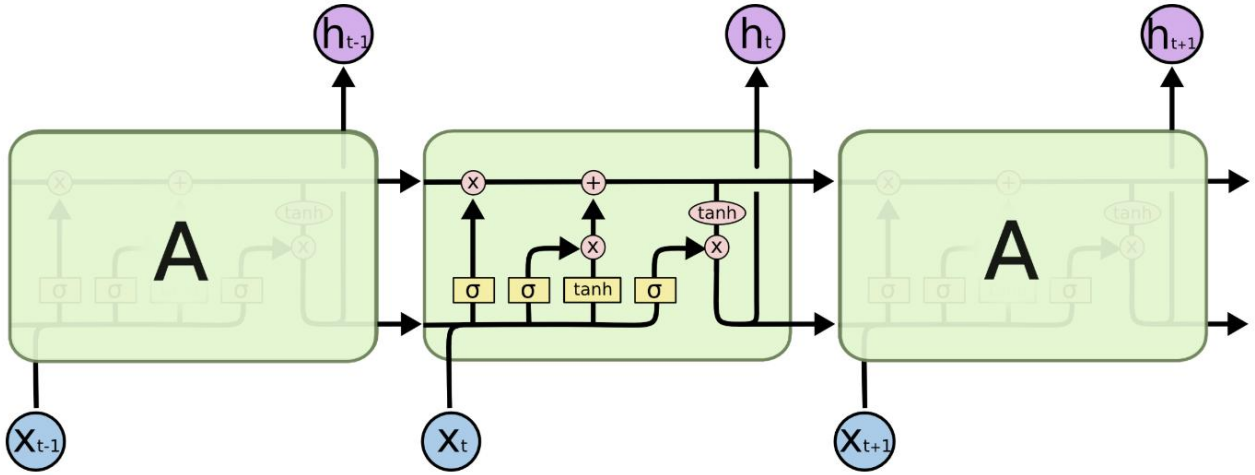
- Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) designed to handle sequential data and long-term dependencies.
- It overcomes the vanishing gradient problem present in standard RNNs.



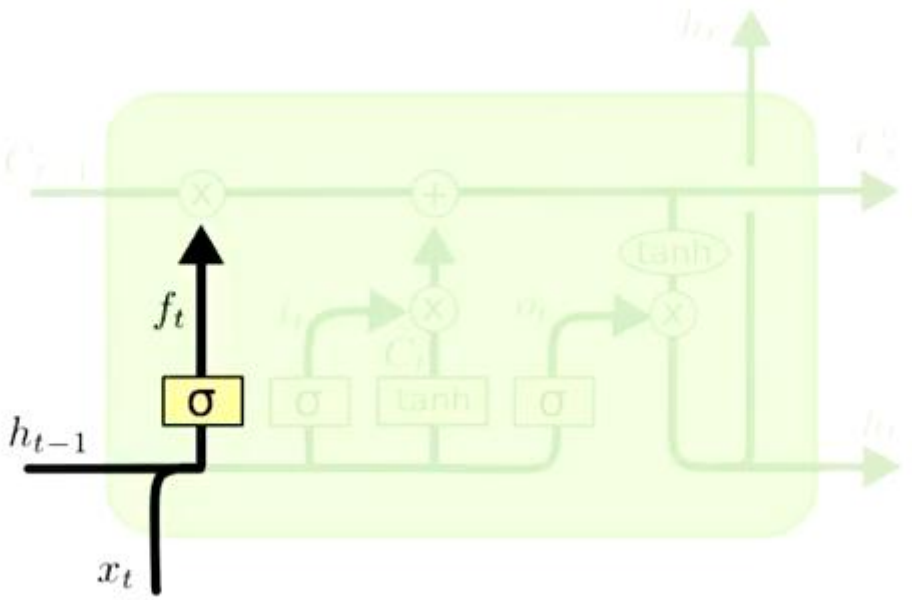
LSTM Architecture Overview

Main Components of LSTM:

- **Cell State:** Stores long-term memory.
- **Forget Gate:** Decides what information to discard.
- **Input Gate:** Determines what new information to store.
- **Output Gate:** Controls the output from the cell.



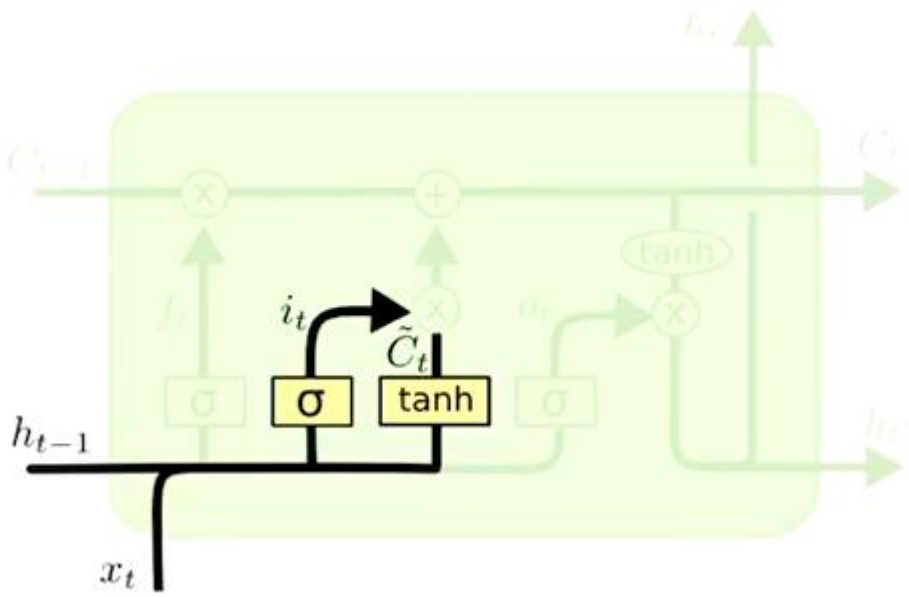
Forget Gate



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

The first part is forget gate, which decide what information to discard

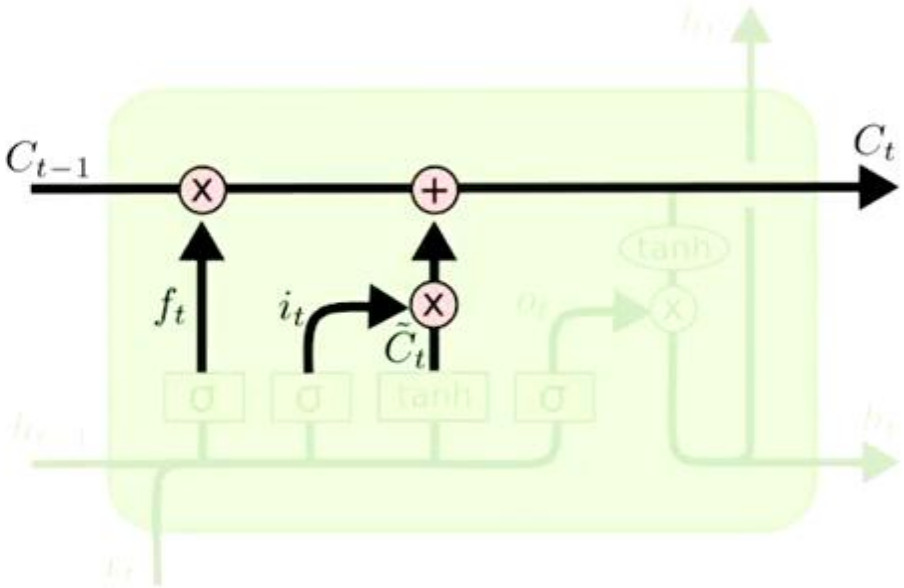
Input Gate



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i$$
$$\widetilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

The input gate decides which values to update, and new candidate cell values are created

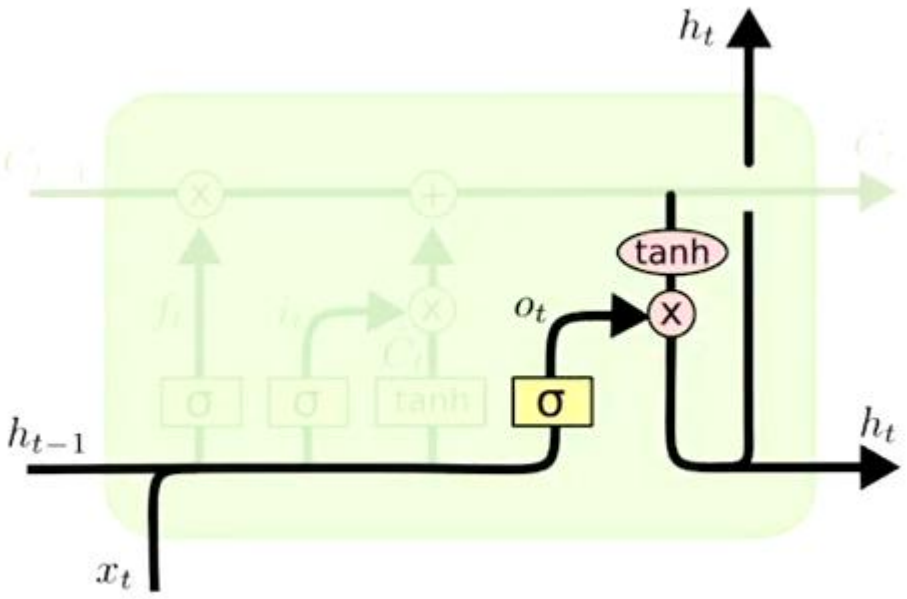
Cell State



$$C_t = f_t * C_{t-1} + i_t * \widetilde{C}_t$$

The cell state is updated

Output Gate



$$\sigma_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh(C_t)$$

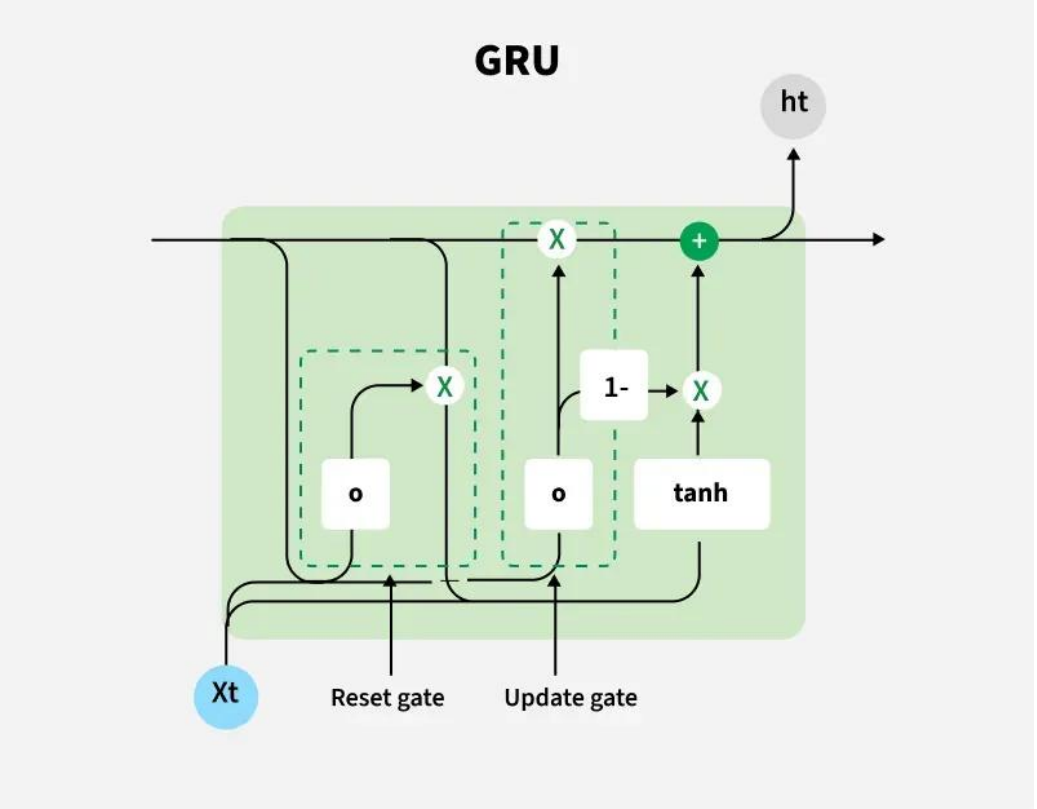
Finally, the output gate decides which values to output

What is a GRU? (Gated Recurrent Unit)

- A simplified variant of LSTM
- Designed to handle long-term dependencies with fewer parameters
- Uses two gates instead of three
- Faster and more efficient than LSTM while maintaining strong performance

Gated Recurrent Units (GRUs)

- **Update Gate**: This gate decides how much information from previous hidden state should be retained for the next time step.
- **Reset Gate**: This gate determines how much of the past hidden state should be forgotten.



LSTM vs GRU

- Both solve the same problem (long-term dependencies)
- GRU is **simpler** and **faster**
- LSTM is more **expressive** and **powerful**
- Choice depends on:
  - Dataset size
  - Sequence length
  - Hardware constraints
  - Accuracy vs. speed trade-offs

Applications of RNNs

- Time Series Forecasting
- Healthcare & Biomedical Signals
- Language Modelling
- Speech Recognition
- Video & Sequential Vision Tasks
- Robotics & Control Systems

In some applications, Transformers replaced RNNs

Summary

Feature	FNN	CNN	RNN
Definition	Feed-forward network of neurons processing input to output, optionally with hidden layers	Convolutional layers for automatic spatial feature extraction in images or videos	Neural network with recurrent connections to process sequential data
Type of Data	Tabular, structured data	Images, videos, spatial data	Sequential, time-series, text, speech
Model Complexity	Simple to moderate	Moderate to high	High
Applications	Pattern recognition, basic computer vision	Image recognition, object detection, text digitization	Text-to-speech, language modelling, time-series prediction