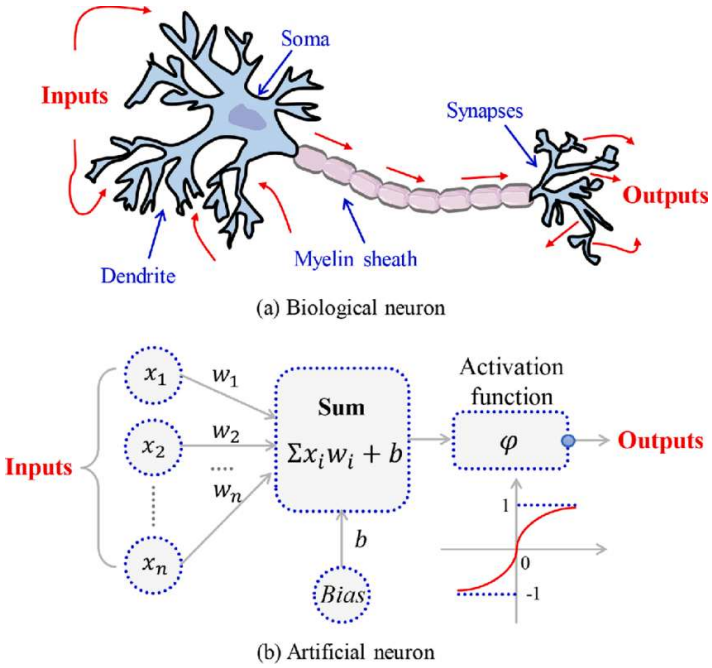
All lectures on neural networks

Introduction to Neural Networks

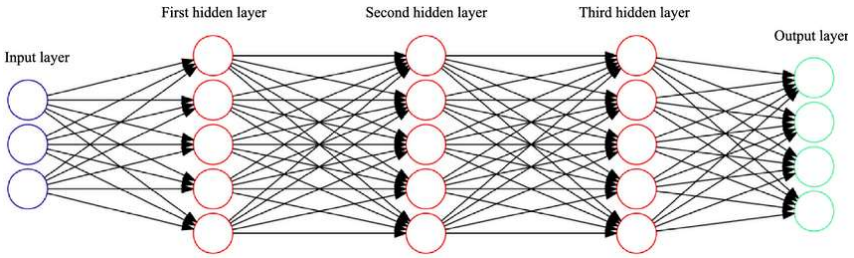
- Biological inspiration: Neurons, synapses, brain structure
- Artificial components: Nodes, edges, layers (input, hidden, output)
- Key concepts: Weights, biases, feedforward propagation



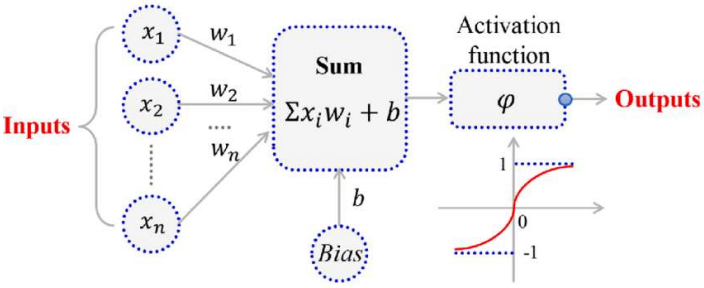
https://www.researchgate.net/profile/Xianlin-Wang/publication/351372032/figure/fig4/AS:1020744041525248@1620375752492/Comparison-between-biological-neuron-and-artificial-neuron-40.png

Basic Architecture of Neural Networks

- Architecture: Input layer, hidden layers, output layer
  - Input Layer: Receives initial data, normalization
  - Hidden Layers: Process information, multiple layers in deep networks
  - Output Layer: Produces final prediction or classification
- Neurons and Connections:
- Dense, fully connected vs. partial connected...
- Mathematical representation: $z_j^{(l)} = \sum_{i=1}^{n} w_{ji}^{(l)} a_i^{(l-1)} + b_j^{(l)}$



https://www.researchgate.net/figure/Fully-connected-deep-neural-network_fig1_332197484



https://www.researchgate.net/profile/Xianlin-Wang/publication/351372032/figure/fig4/AS:1020744041525248@1620375752492/Comparison-between-biological-neuron-and-artificial-neuron-40.png

$a_j = \sum_i x_i w_{ji}$ and $z_j = h(a_j)$

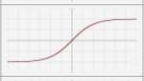
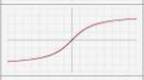
Practical Applications of Neural Networks

- Shape fitting: Regression, curve fitting, function approximation
- Classification: Binary, multi-class, hierarchical
- Real-world examples: Image recognition, speech recognition, natural language processing
- Industry applications: Healthcare, finance, autonomous vehicles, robotics

---

Activation Functions

- Properties: Non-linearity, Differentiability, Range
- Considerations: Shaping vs learning, Vanishing Gradient, Freezing neurons

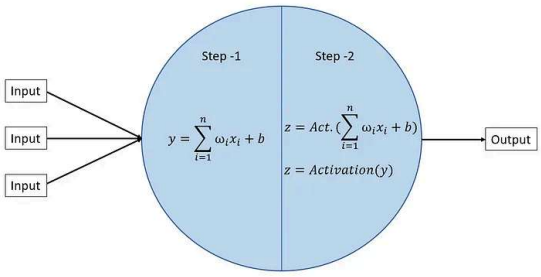
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

https://miro.medium.com/max/1400/1*p_hyqAtyI8pbt2kEI6siOQ.png

---

Forward Propagation

- Process of passing input data through the network
- Steps:
  1. Input data fed into input layer
  2. Compute weighted sum and apply activation for each layer
  3. Final layer produces output



https://medium.com/analytics-vidhya/what-do-you-mean-by-forward-propagation-in-ann-9a89c80dac1b

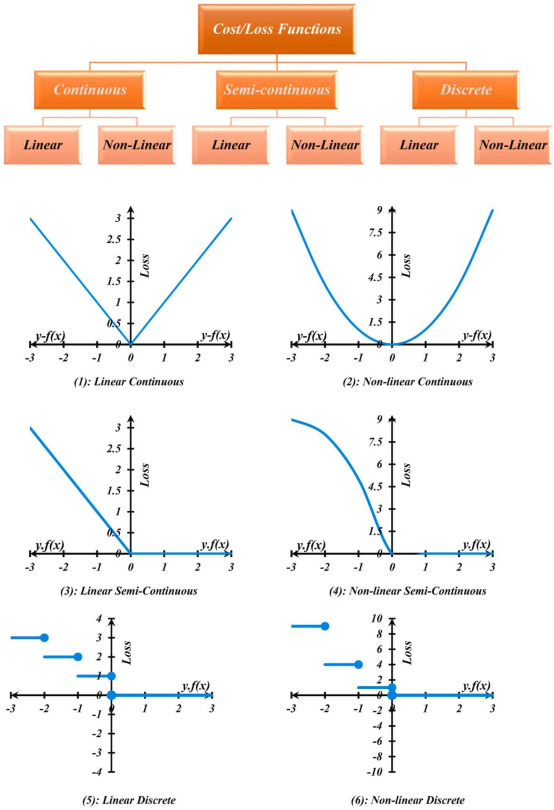
---

Loss Functions

- Measure difference between predicted and actual outputs
- Common loss functions:

1. Mean Squared Error (regression): $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
2. Mean Absolute Error (MAE) $MAE = \frac{1}{n}\sum_{i=1}^{n}|(y_i - \hat{y}_i)|$
3. Binary Cross-Entropy (binary classification): $BCE = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$

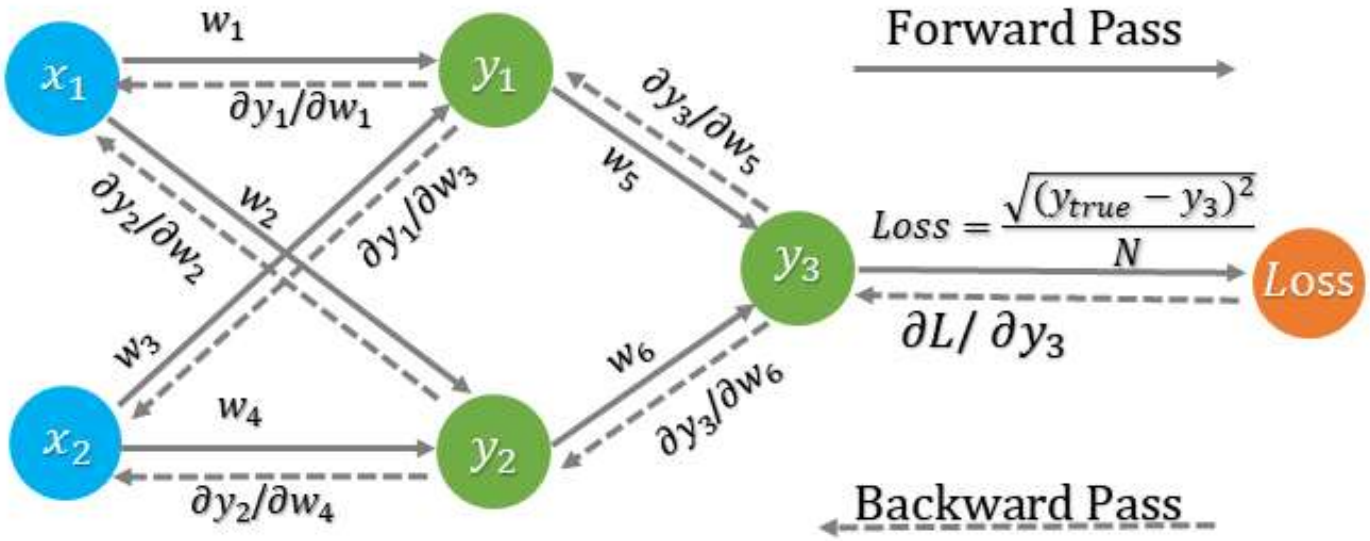
   Uses
4. Categorical Cross-Entropy (multiclass classification): $CCE = -\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}\log(\hat{y}_{ij})$



https://www.sciencedirect.com/science/article/pii/S0952197623015993#fig1

Backpropagation and Training

- Backpropagation algorithm
- Gradient Descent and Optimization, Basic algorithm



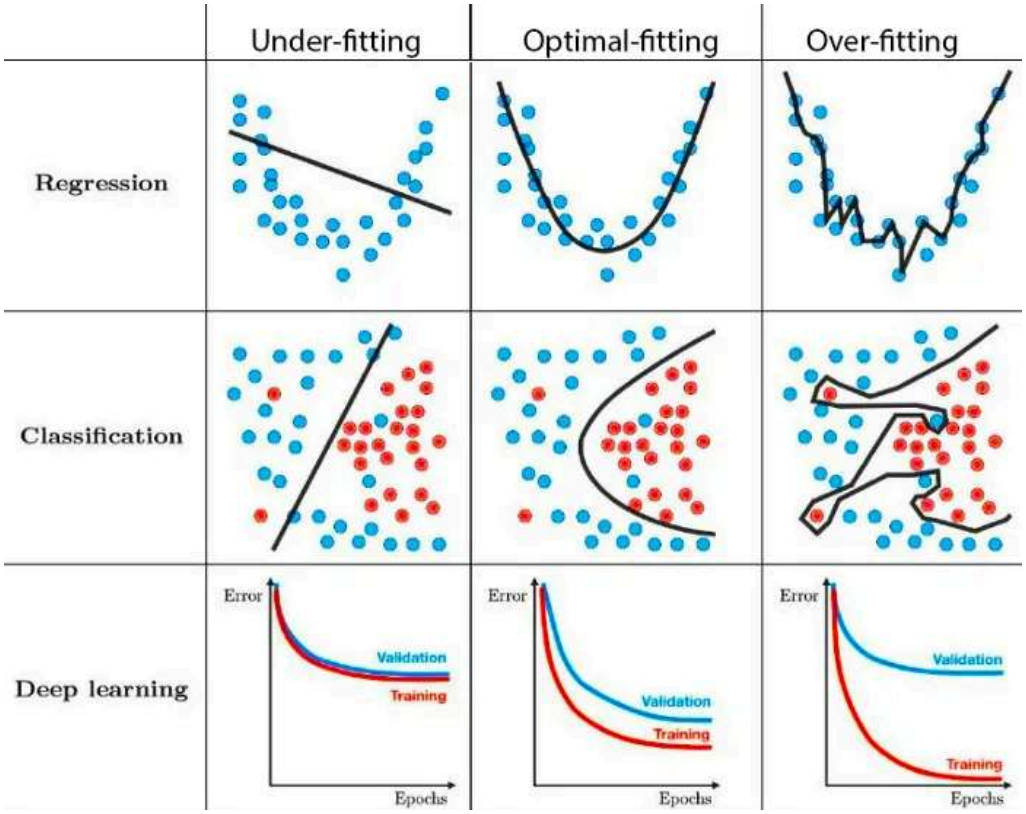
https://www.researchgate.net/figure/An-overview-of-backpropagation-a-Role-of-backpropagation-in-a-neural-network-b_fig2_356390636

Practical considerations and terms

1. Validation vs. Training data
    1. Randomly divide your dataset
    2. Consider second testing Validation
2. Epochs
3. Number of Training epochs (define or interrupt)
4. Learning rate? ..... Time!
5. Forgetting rate?... Why? - Freezing neuron
6. Dropping rate?.... Why? - Local minima
7. Data Filtering/Augmenting

Overfitting and Underfitting

- Bias-variance tradeoff (to simple vs to complex)
- Training error vs. validation error (training to long on specific set)
- Capacity: Model complexity vs. dataset size (to small/biased datasets)
- Early stopping: Monitoring validation performance



https://towardsdatascience.com/techniques-for-handling-underfitting-and-overfitting-in-machine-learning-348daa2380b9

Practical Implementation

- (Professional) Frameworks: TensorFlow, PyTorch, Keras
- Key components: Model definition, data loading, training loop, evaluation
- GPU acceleration: CUDA, cuDNN
- Debugging techniques: Gradient checking, learning rate scheduling

Non professional: Excel, Numpy

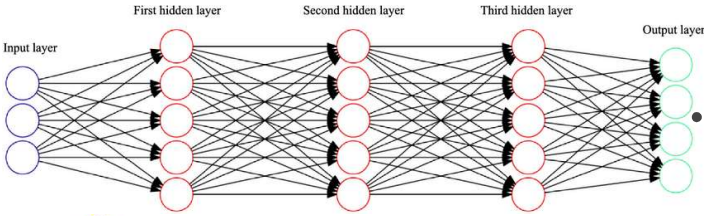
(show excel Neural Network)

Advanced Topics in Neural Networks

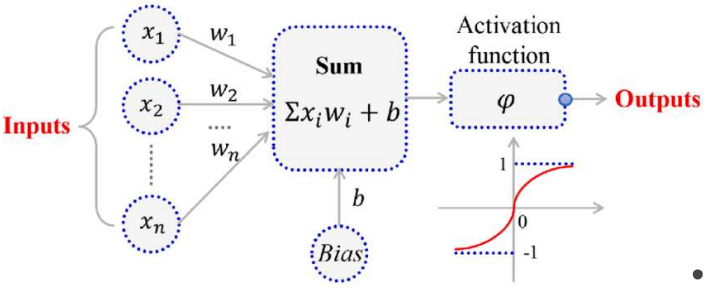
This is the second lecture in our series on Neural Networks, building upon the foundational concepts discussed in the first lecture.

Review of Fundamentals

- Basic Architecture of Neural Networks



https://www.researchgate.net/figure/Fully-connected-deep-neural-network_fig1_332197484



https://www.researchgate.net/profile/Xianlin-Wang/publication/351372032/figure/fig4/AS:1020744041525248@1620375752492/Comparison-between-biological-neuron-and-artificial-neuron-40.png

- Forward Propagation Or how the data flows through the network

$a_j = \sum_i x_i w_{ji}$ and $z_j = h(a_j)$

- Error Generation Difference between predicted outputs and actual targets.
  - Loss Function (e.g., Mean Squared Error): $E = \frac{1}{n}\sum_{i=1}^{n}(z_i - \hat{z}_i)^2$
- Backward Propagation Error gradients flow backward to update weights.
  - Gradient of Error with respect to weights: $\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \delta_j \frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i$ or $\delta_j = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j}$ for hidden units
  - The local δ_j adjusts weights to minimize the loss function.

Alternative Optimization Algorithms

- Stochastic Gradient Descent (SGD) (mini batch)
  - Updates weights using a small batch of data. (not after one epoch)
  - Can be noisy but often leads to faster convergence as local minimas are avoided
- Momentum
  - Incorporates past gradients to smooth updates by adding a little bit of the previous update on top of the currently calculated one.
- RMSProp (Root Mean Square Propagation)
  - adaptive learning rate optimization algorithm designed to address the problem of vanishing or exploding gradients by adjusting the learning rate for each parameter based on the magnitude of recent gradients.
- Adam Optimizer (Adaptive Moment Estimation)
  - Combines the benefits of Momentum and RMSProp (adjusts the learning rate)
  - Adjusts learning rates individually for each parameter.

Optimizers play a crucial role in how efficiently and effectively a neural network learns.

Adam Optimizer

1. First Moment (Momentum-like)

Formula	Explanation
$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \frac{\partial L}{\partial \theta_t}$	- m_t = first moment (mean of gradients) - $\frac{\partial L}{\partial \theta_t}$ = gradient of the loss function L with respect to parameters θ_t - β_1 = decay rate for the first moment, typically set to **0.9**. It controls how much of the past gradients are retained.

2. Second Moment (RMSProp-like)

Formula	Explanation
$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left(\frac{\partial L}{\partial \theta_t}\right)^2$	- v_t = second moment (mean of squared gradients) - $\left(\frac{\partial L}{\partial \theta_t}\right)^2$ = element-wise square of the gradient - β_2 = decay rate for the second moment, typically set to **0.999**. It controls how much of the past squared gradients are retained.

3. Bias Correction

Formula	Explanation
$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$	- $\hat{m}_t$ = bias-corrected first moment - β_1 helps to correct bias toward zero in the early stages.
$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$	- $\hat{v}_t$ = bias-corrected second moment - β_2 helps to correct bias toward zero in the early stages.

4. Parameter Update

Formula	Explanation
$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$	- θ_{t+1} = updated parameter at time step $t + 1$ - α = learning rate - $\hat{m}_t$ = bias-corrected first moment - $\hat{v}_t$ = bias-corrected second moment - ϵ = small constant to prevent division by zero.

Hyperparameter Tuning

Hyperparameters are settings that control the behavior of the training process but are not learned from the data.

- Examples include:
  - Learning Rate: Controls how large each update step is.
  - Batch Size: Determines the number of samples used in each training iteration.
  - Number of Layers/Neurons: Defines the architecture of the model.
  - Regularization Parameters: Helps prevent overfitting by penalizing complex models.
  - Dropout Rate: Probability of dropping neurons during training.

Regularization

- Prevent Overfitting: Discourage complexity during training.
- Encourage Simplicity: Penalizing large weights and complex models

L1 and L2 Regularization

- L1 Regularization: Adds the absolute value of weights to the loss function.
  $Loss = Loss_{original} + \lambda \sum |w_i|$
  Encourages sparsity — many weights become zero.
- L2 Regularization: Adds the squared value of weights to the loss function.
  $Loss = Loss_{original} + \lambda \sum w_i^2$
  Encourages smaller weights overall.

Think order of polynomial for fitting data

Dropout

- Randomly drops neurons during training to prevent over-reliance on specific neurons.
  Think the random locking (during backpropagation) decouples neurons, breaks linearity

Batch Normalization

- Normalizes inputs to each layer to have zero mean and unit variance, stabilizing learning.
- Decouples layers, reduces initial parameter sensitivity

(expensive) Tuning Techniques

Grid Search **Tests combinations of predefined hyperparameter values.**

- Pros: Systematic exploration.
- Cons: Computationally expensive.

Random Search **Random search values from a predefined range.**

- Pros: More efficient than grid search.
- Cons: Might miss optimal combinations.
- Hyperband Variant that stops the search early

Bayesian Optimization **Uses past results to predict promising hyperparameter values.**

- Pros: Finds good combinations with fewer evaluations.
- Cons: Complex to implement.

Modern methods use instead block types that improve propagation and learning (see end)

Its all about the vanishing gradients, freezing and local minima

---

Types of Neural Networks

We focus on (see also larger list at the end)

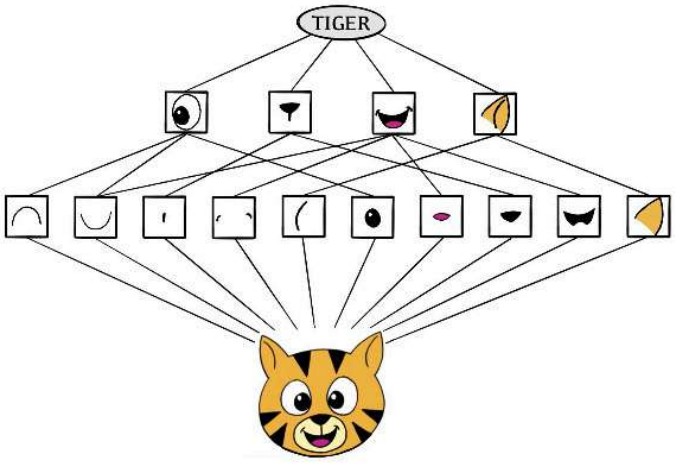
- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM) Networks as implementation
- GAN

Convolutional Neural Networks (CNNs)

Strong for image recognition e.g. OCR, or facial recognition

- Convolution Layers
  - Apply filters/kernels to input data.
  - Capture local spatial patterns, extracts features using learnable kernels (weights)
- Feature Hierarchies

- Shallow layers detect edges and textures.
- Deeper layers recognize complex shapes and objects.



Convolution Operation

- Mathematical Representation

Formula	Explanation
$(I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$	I is the input image K is the kernel/filter

```
from scipy import misc
im = misc.ascent()
from scipy import ndimage
```

edge_detection= $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

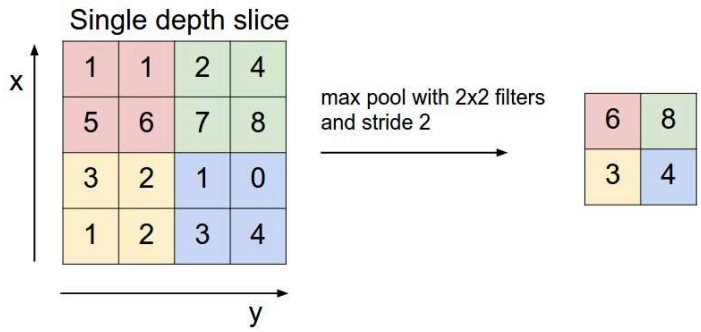
```
ndimage.convolve(im, edge_detection, mode="constant", cval=0)
```

- Padding: Adds zeros around input to control output size.

- Stride: Steps the filter moves across the input.

Pooling Layers in Neural Networks

- Purpose of Pooling
  - Reduces spatial dimensions of data.
  - Controls overfitting by reducing parameters.
  - Aggregates features to make the model more robust to translations.
- Max Pooling
  - Operation: Takes the maximum value in each pooling window.
  - Effect: Captures the most prominent features in each region. (think what happens in the edges or center...)



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

Convolution layers learn to detect patterns, while pooling layers simplify the data by summarizing these patterns.

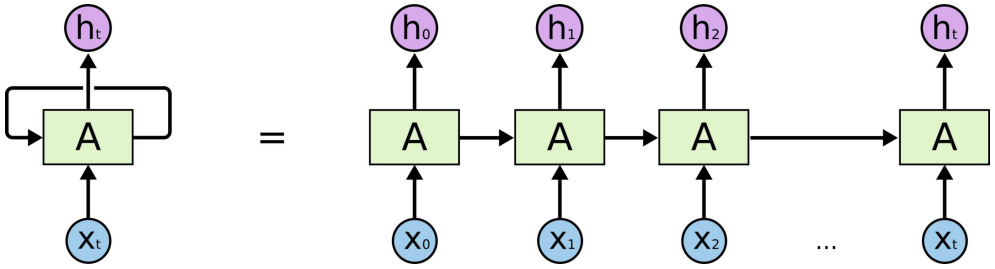
Convolution layers extract local features from the input, such as edges, textures, or simple patterns. As more convolution layers are stacked, these local features are combined to detect more complex patterns or higher-level features.
1. In earlier layers, convolution filters might detect edges, corners, or simple shapes.
2. In deeper layers, the features become more abstract, capable of detecting complex structures such as parts of objects or even entire objects.

Pooling layers (typically MaxPooling) are applied after convolution layers to downsample the feature maps, reducing their spatial dimensions. This allows the network to progressively focus on the most important features while discarding unnecessary details.

By stacking convolution and pooling layers, the network learns a hierarchical representation of the input,
1. initial layers focus on low-level features (think ears, eyes)
2. deeper layers capture high-level concepts. (Think shape distribution)

Recurrent Neural Networks (RNNs)

- Designed for Sequential Data
  - Processes sequences, maintaining context through time.
- Applications
  - Language modeling, translation.
  - Time-series prediction.
  - Language models like ChatGPT utilize architectures inspired by RNNs to handle sequences.
- Vanilla RNN Equation
  - $h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$ where x_t is current input h_{t-1} output from previous input (in time)
  - $y_t = W_{hy}h_t + b_y$



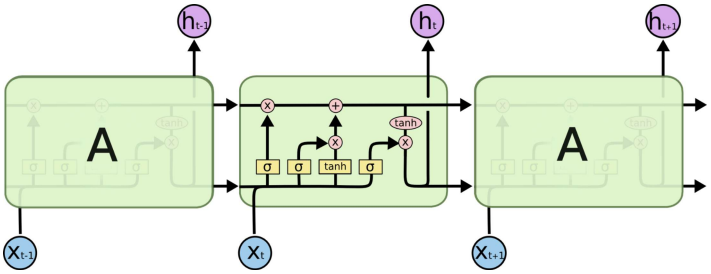
https://colah.github.io/posts/2015-08-Understanding-LSTMs

Long Short-Term Memory (LSTM) Networks

Controll the Cell State (C_t): Memory of the network.
Gates:
- New state: $h_t = o_t * \tanh(f_t * C_{t-1} + i_t * \tilde{C}_t)$
- Candidate cell state: C_t

Gates that regulate the time each :
- Forget Gate: $f_t = Sigmodial(W_f[h_{t-1}, x_t] + b_f)$
- Input Gate: $i_t = Sigmodial(W_i[h_{t-1}, x_t] + b_i)$
- Output Gate: $o_t = Sigmodial(W_o[h_{t-1}, x_t] + b_o)$

LSTMs can capture dependencies over long sequences, making them suitable for complex language models like ChatGPT.

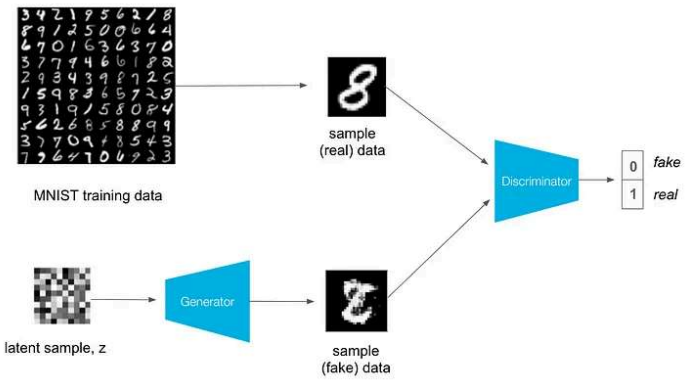


https://colah.github.io/posts/2015-08-Understanding-LSTMs

Generative Adversarial Networks (GANs)

- Components (that are trained together iteratively one step each)
  - Generator Network
    - Creates synthetic data resembling real data.
    - tries to create data that minimizes the Discriminator's ability to correctly classify fake data as fake
  - Discriminator Network
    - Distinguishes between real and synthetic data.
    - tries to maximize its accuracy in distinguishing real data from fake data.
    - Trained on:
      - Real data from the training set, which it should classify as real (output close to 1).
      - Fake data from the Generator, which it should classify as fake (output close to 0).

Challenges: non-convergence or mode collapse (generator produces limited variations of output)



https://medium.com/swlh/gan-generative-adversarial-network-3706ebfef77e

---

GANs in Practice

- Image Generation: Creating realistic faces, art. See e.g. https://github.com/phillipi/pix2pix

- Data Augmentation: Enhancing training datasets. general GAN e.g. https://github.com/tensorflow/gan
- Chess and Game Playing
  - GANs used to generate new strategies or positions.
  - Example: AI models generating novel chess games.
- New Models (e.g., o1 Model)
  - Advanced GAN architectures pushing boundaries in data generation.
  - Call Normal model 20 times and select best answer
- Super-Resolution
- Text-to-Image Generation

GANs have revolutionized how we approach data generation, impacting fields like game theory and creative arts.

Jacobian and Hessian Matrices in Neural Networks

- Jacobian Matrix Represents all first-order partial derivatives of a vector-valued function.
  - $J_{ij} = \frac{\partial y_i}{\partial x_j}$
- Hessian Matrix, Represents all second-order partial derivatives.
  - $H_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j}$
  Usually contains the curvature of the loss function and helps in adjusting learning rates dynamically.
- Example: Pulse Processing/Filtering
  - Using the Jacobian to adjust filter parameters to match desired pulse shapes.
  - Hessian provides curvature information, aiding in refining filter settings and provides cross-input information

---

Practical Implementation Tips 1

Model Initialization (Remember it is a fitting procedure!)

Proper initialization helps prevent vanishing or exploding gradients. Scale weights based on the number of input and output neurons.

- Xavier Initialization: For layers with symmetric activation functions like $\tanh$, use: $W = \sqrt{\frac{1}{n_{in}+n_{out}}}$

- He Initialization: for ReLU activations use: $W = \sqrt{\frac{2}{n_{in}}}$

So for a the relu layer l the weights: W_l=np.random.randn(size$_l$,size$_{l-1}$)*np.sqrt(2/size$_{l-1}$)

Starting with the right weights is like beginning a journey from a favorable location.

Practical Implementation Tips 2

Learning Rate Scheduling

- Step Decay: Reduces learning rate at set intervals: $\alpha = \alpha_0 \cdot \gamma^{\lfloor \frac{epoch}{step} \rfloor}$
- Exponential Decay: Gradually decreases learning rate over time:$\alpha = \alpha_0 \cdot e^{-\lambda \cdot epoch}$
- Adaptive Methods: Adjusts learning rate based on performance metrics.

Monitoring Training

- Plotting training and validation loss curves.
- Using tools like TensorBoard or Matplotlib for visualization.

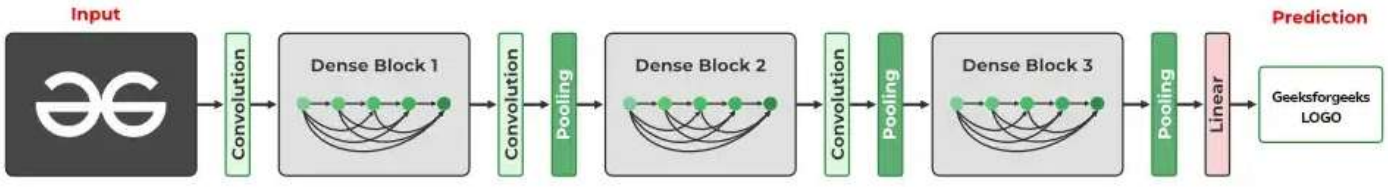
Debugging Techniques

- Gradient Checking: Verifies the correctness of gradients numerically.
- Visualizing Activations: Identify dead neurons or activations saturating

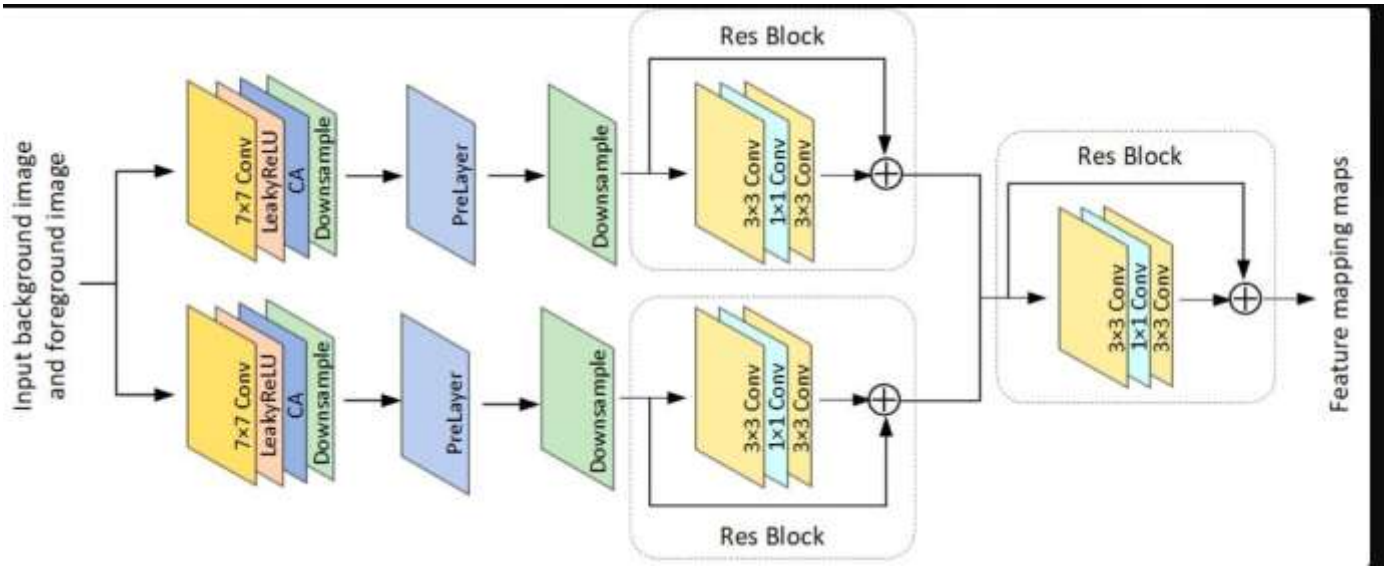
---

Some Network types in the fast developing world

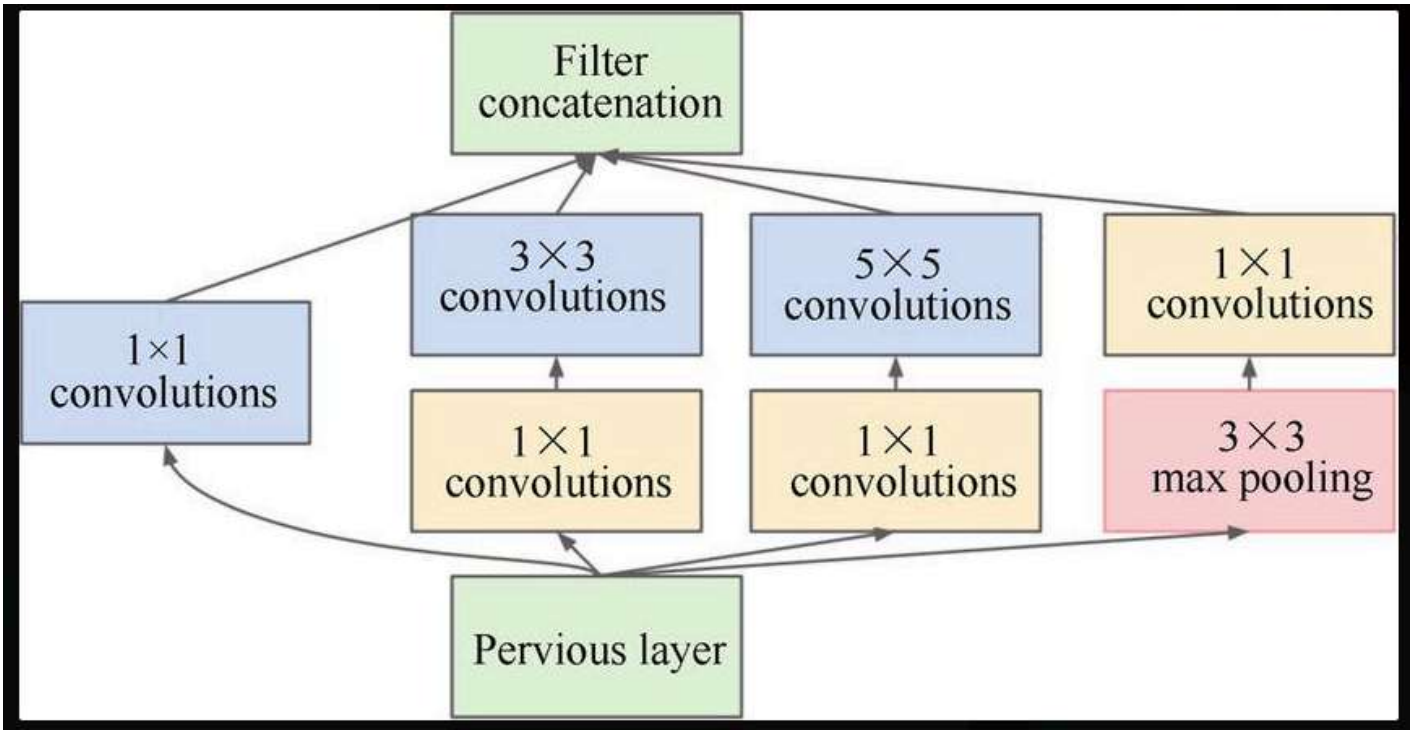
Dense Connections (DenseNets)



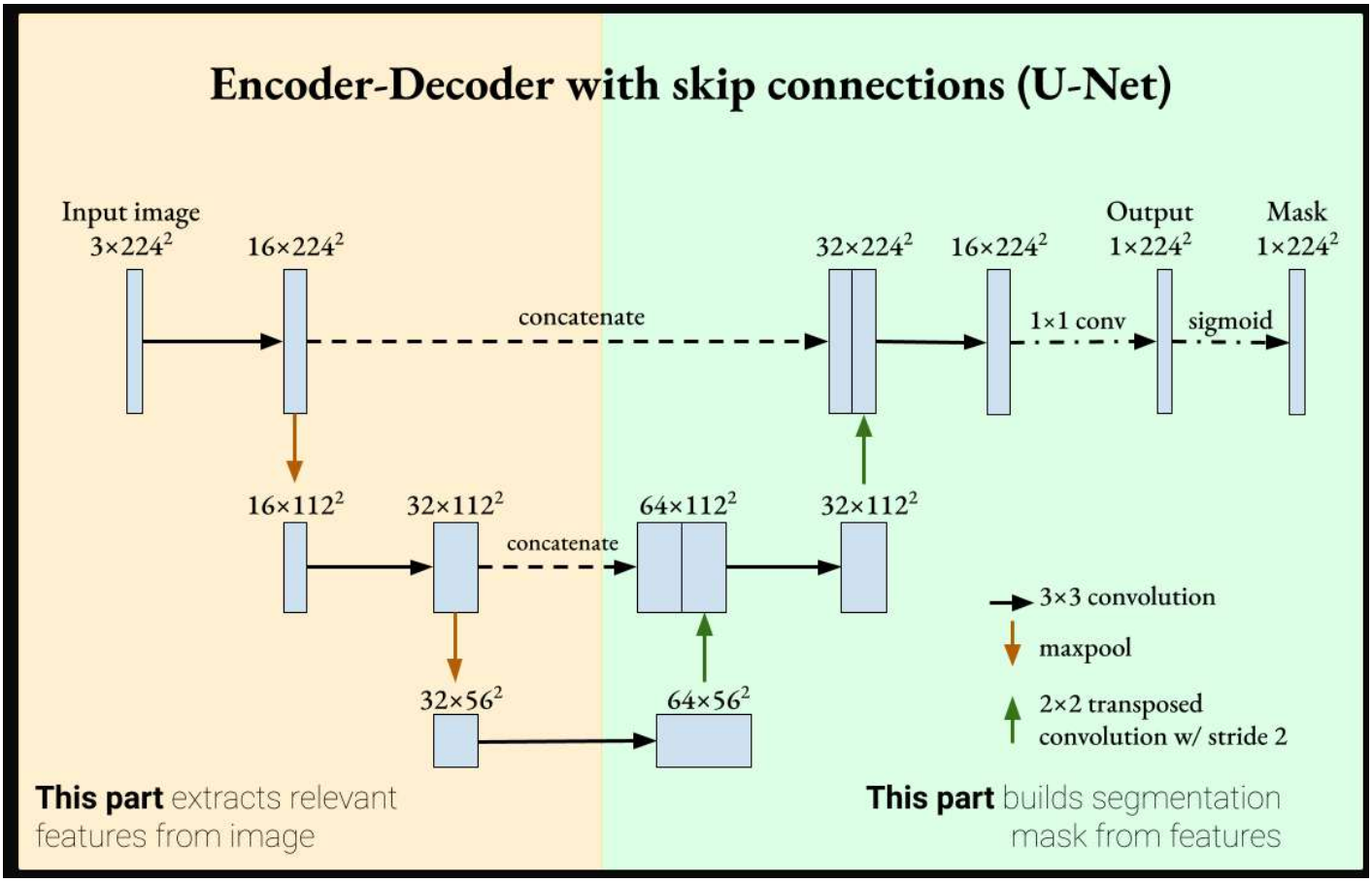
2. Highway Networks



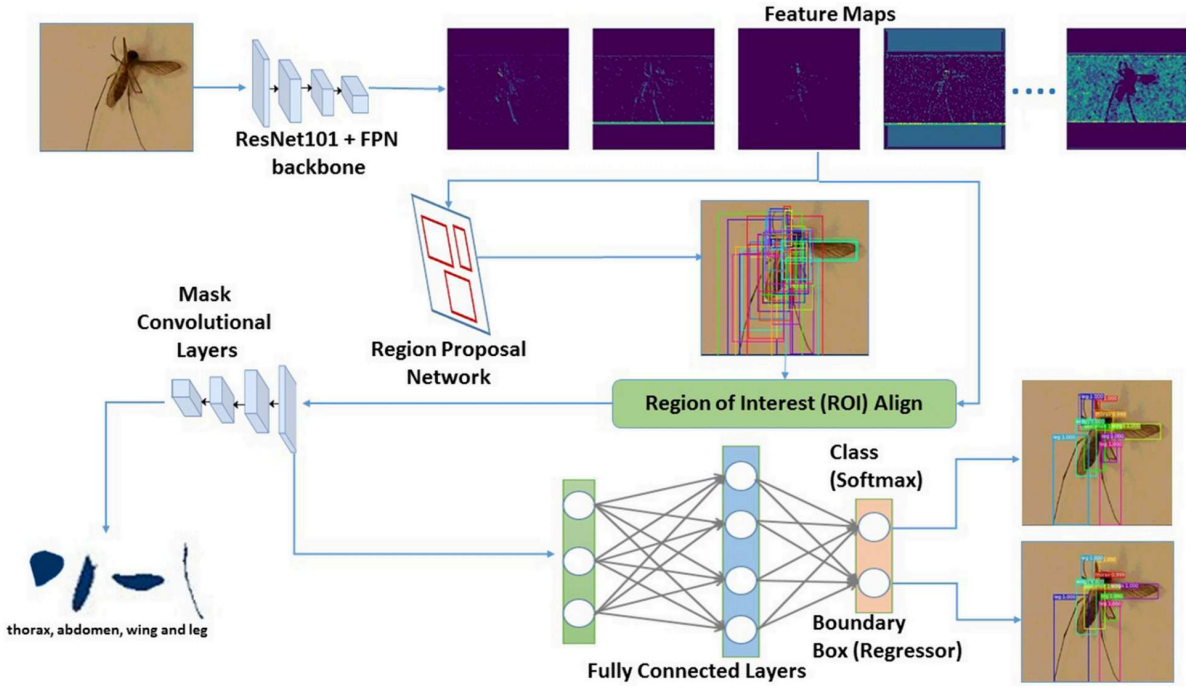
3. Inception/Parallel (GoogleNet/Inception Modules)



4. U-Net and Encoder-Decoder Skip Connections



5. Masked Connections and Attention



Crazy list of Network types (not complete)

Algorithm	Purpose
Autoencoder (AE)	You typically use AEs to reduce the number of random variables under consideration, so the system can learn a representation for a set of data and, therefore, process generative data models.
Bidirectional Recurrent Neural Network (BRNN)	The goal of a BRNN is to increase the information inputs available to the network by connecting two hidden, directionally opposing layers to the same output. Using BRNNs, the output layer can get information from both past and future states.
Boltzmann Machine (BM)	A recurrent neural network, this algorithm is capable of learning internal representations and can represent and solve tough combined problems.
Convolutional Neural Network (CNN)	Most commonly used to analyze visual imagery, CNNs are a feed-forward neural network designed to minimize pre-processing.
Deconvolutional Neural Network (DNN)	DNNs enable unsupervised construction of hierarchical image representations. Each level of the hierarchy groups information from the preceding level to add more complex features to an image.
Deep Belief Network (DBN)	When trained with an unsupervised set of examples, a DBN can learn to reconstruct its inputs probabilistically by using layers as feature detectors. Following this process, you can train a DBN to perform supervised classifications.
Deep Convolutional Inverse Graphics Network (DCIGN)	A DCIGN model aims to learn an interpretable representation of images that the system separates according to the elements of three-dimensional scene structure, such as lighting variations and depth rotations. A DCIGN uses many layers of operators, both convolutional and deconvolutional.
Deep Residual Network (DRN)	DRNs assist in handling sophisticated deep learning tasks and models. By having many layers, a DRN prevents the degradation of results.
Denoising Autoencoder (DAE)	You use DAEs to reconstruct data from corrupted data inputs; the algorithm forces the hidden layer to learn more robust features. As a result, the output yields a more refined version of the input data.
Echo State Network (ESN)	An ESN works with a random, large, fixed recurrent neural network, wherein each node receives a nonlinear response signal. The algorithm randomly sets and assigns weights and connectivity in order to attain learning flexibility.
Extreme Learning Machine (ELM)	This algorithm learns hidden node output weightings in one step, creating a linear model. ELMs can generalize well and learn many times faster than backpropagation networks.
Feed Forward Neural Network (FF or FFNN) and Perceptron (P)	These are the basic algorithms for neural networks. A feedforward neural network is an artificial neural network in which node connections don’t form a cycle; a perceptron is a binary function with only two results (up/down; yes/no, 0/1).
Gated Recurrent Unit (GRU)	GRUs use connections through node sequences to perform machine learning tasks associated with clustering and memory. GRUs refine outputs through the control of model information flow.

Algorithm	Purpose
Generative Adversarial Network (GAN)	This system pits two neural networks — discriminative and generative — against each other. The objective is to distinguish between real and synthetic results in order to simulate high-level conceptual tasks.
Hopfield Network (HN)	This form of recurrent artificial neural network is an associative memory system with binary threshold nodes. Designed to converge to a local minimum, HNs provide a model for understanding human memory.
Kohonen Network (KN)	A KN organizes a problem space into a two-dimensional map. The difference between self-organizing maps (SOMs) and other problem-solving approaches is that SOMs use competitive learning rather than error-correction learning.
Liquid State Machine (LSM)	Known as third-generation machine learning (or a spiking neural network), an LSM adds the concept of time as an element. LSMs generate spatiotemporal neuron network activation as they preserve memory during processing. Physics and computational neuroscience use LSMs.
Long/Short-Term Memory (LSTM)	LSTM is capable of learning or remembering order dependence in prediction problems concerning sequence. An LSTM unit holds a cell, an input gate, an output gate, and a forget gate. Cells retain values over arbitrary time intervals. Each unit regulates value flows through LSTM connections. This sequencing capability is essential in complex problem domains, like speech recognition and machine translation.
Markov Chain (MC)	An MC is a mathematical process that describes a sequence of possible events in which the probability of each event depends exclusively on the state attained in the previous event. Use examples include typing-word predictions and Google PageRank.
Neural Turing Machine (NTM)	Based on the mid-20th-century work of data scientist Alan Turing, an NTM performs computations and extends the capabilities of neural networks by coupling with external memory. Developers use NTM in robots and regard it as one of the means to build an artificial human brain.
Radial Basis Function Networks (RBF nets)	Developers use RBF nets to model data that represents an underlying trend or function. RBF nets learn to approximate the underlying trend using bell curves or non-linear classifiers. Non-linear classifiers analyze more deeply than do simple linear classifiers that work on lower dimensional vectors. You use these networks in system control and time series predictions.
Recurrent Neural Network (RNN)	RNNs model sequential interactions via memory. At each time step, an RNN calculates a new memory or hidden state reliant on both the current input and previous memory state. Applications include music composition, robot control, and human action recognition.
Restricted Boltzmann Machine (RBM)	An RBM is a probabilistic graphical model in an unsupervised environment. An RBM consists of visible and hidden layers as well as the connections between binary neurons in each of these layers. RBNs are useful for filtering, feature learning, and classification. Use cases include risk detection and business and economic analyses.
Support Vector Machine (SVM)	Based on training example sets that are relevant to one of two possible categories, an SVM algorithm builds a model that assigns new examples

Algorithm	Purpose
	to one of two categories. The model then represents the examples as mapped points in space while dividing those separate category examples by the widest possible gap. The algorithm then maps new examples in that same space and predicts what category they belong to based on which side of the gap they occupy. Applications include face detection and bioinformatics.
Variational Autoencoder (VAE)	A VAE is a specific type of neural network that helps generate complex models based on data sets. In general, an autoencoder is a deep learning network that attempts to reconstruct a model or match the target outputs to provided inputs through backpropagation. A VAE also yields state-of-the-art machine learning results in the areas of image generation and reinforcement learning.

Further Reading and Resources

- Books
  - *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by Aurélien Géron
  - *Pattern Recognition and Machine Learning* by Christopher M. Bishop
  - *Deep Learning* by Ian Goodfellow, Yoshua Bengio, and Aaron Courville
- Online Courses : https://www.coursera.org/specializations/deep-learning
- LLM Tools : Ask AI how to use AI, Phrase full sentences and then ask more. Chat GPT and CO are your private tutors

Training Questions:

1. What are the essential components of a NN, explain the function of the different layers, and the key lingo
2. Name and explain the two most used Activation functions ("Leaky" ReLu, Sigmoid or Tanh)
    1. when to use which
    2. why an activation function
3. Name and explain the main Loss functions, why do we need them, what do they do
4. Explain Backpropagation and the vanishing gradient concept
5. Name and Explain alternative training methods, SGD, Momentum, RMSProp, ADAM
6. What is hyperparameter tuning
7. What is regularization, how does it work, why do you need it?
8. what are CNN's why and how do they work
9. Why would you stack convolution and MaxPooling?

10. What is the principle of RNN and how does LSTM work?
11. What is the principle of GANs and where are they used?
12. Why would you use dropouts?
13. How do you generate your own training data?
14. Name a few biases that you have personally encountered when working with NN. How could you have discovered them before.
15. What are good feedback tools that enable you train NN more efficiently
16. Name at least 5 different uses of NN in DataProcessing
17. What differs unsupervised learning from the two other main groups of machine Learning.
18. Why is it a challenge to use for a large number of dimensions.
19. Describe in a few words the main concept of clustering
    1. name at least 3 different clustering methods.
    2. What is the common main challenge in clustering?
    3. How could you overcome this challenge
20. Dimensionality reduction:
    1. Name the key concept
    2. explain the concept of one linear and one non linear
    3. Explain the key concept of PCA
    4. Why can SVD be used to solve PCA?
    5. Why are in most cases the achieve vectors not representative of the reality.
    6. What is the relation between PCA and linear regression?