Here's a comprehensive set of exam notes for Machine Learning, based on the provided lecture material.

Machine Learning Exam Notes 📝

These notes cover fundamental concepts, algorithms, and applications in Machine Learning.

---

1. Introduction to Machine Learning

What is Machine Learning (ML)?
Machine learning is a subfield of artificial intelligence (AI) that focuses on building systems that can learn from data, identify patterns,1 make decisions, and improve with experience, without being explicitly programmed for each task. A key aspect is the ability to generalize to new, unseen data. The core task often involves learning a function ($Y=f(X)$) to predict outputs (Y) from inputs (X). This learned function is an estimate and inherently imperfect, so much of applied ML is about refining this estimate.

Types of Machine Learning:

- **Supervised Learning**:
  - Trained on **labelled data**, meaning input-output pairs are provided.
  - Goal: Learn a mapping from inputs to outputs to generalize to new data.
  - **Types**:
    - **Regression**: Output variable is continuous (e.g., stock prices).
    - **Classification**: Output variable is categorical (e.g., spam/not spam).
  - Examples: ANNs, Decision Trees, KNN, SVMs, Random Forests.
  - Drawbacks: Requires training data, prone to **overfitting** (good on training data, poor on test data), susceptible to bias in training data, scalability issues with large datasets.
- **Unsupervised Learning**:
  - Trained on **unlabelled data**; no target outputs are provided.
  - Goal: Find hidden patterns, structures, or relationships within the input data.
  - **Types**:
    - **Clustering**: Grouping similar data points (e.g., K-Means).
    - **Dimensionality Reduction**: Reducing features while preserving information (e.g., PCA, Autoencoders).
    - **Anomaly Detection**: Identifying unusual data points.
    - **Association Rule Learning**: Discovering relationships between variables.
  - Disadvantages: Ambiguity in results (hard to evaluate objectively), interpretability of discovered patterns can be challenging.
- **Semi-Supervised Learning**:
  - Uses a combination of labelled and unlabelled data for training.
  - Labelled data guides the learning process, while unlabelled data helps improve generalization.

- **Self-Supervised Learning**:
    - A type of unsupervised learning where supervision signals are artificially generated from the data itself.
    - Goal: Learn useful representations without explicit human annotations.
    - Examples: BERT, GPT models for language understanding.
- **Reinforcement Learning (RL)**:
    - An **agent** learns by interacting with an **environment**.
    - Goal: Learn a **policy** (strategy) to maximize cumulative **rewards** over time through trial and error.
    - Inspired by human/animal learning through feedback.

Data Formulation:

- **Object**: An item of interest.
- **Variables/Attributes**: Properties describing an object.
- **Instance**: Set of attribute values for an object.
- **Dataset**: Complete set of data for an application.
    - **Labelled Dataset**: One attribute is designated as the **class** to be predicted.
    - **Unlabelled Dataset**: No specific attribute is designated for prediction.

Types of Variables:

- **Nominal**: Categorical, no order (e.g., color).
    - **Binary**: Nominal with two values (e.g., true/false).
- **Ordinal**: Categorical with a meaningful order (e.g., small, medium, large).
- **Integer**: Whole numbers (e.g., number of children).
- **Interval-scaled**: Numerical values with equal intervals from an arbitrary zero point (e.g., temperature in Celsius).
- **Ratio-scaled**: Similar to interval-scaled, but zero indicates absence of the characteristic (e.g., weight).
- Variables can be grouped as:
    - **Categorical**: Nominal, binary, ordinal.
    - **Continuous**: Integer, interval-scaled, ratio-scaled.

---

2. Data Pre-processing and Normalization

Pre-processing:

- Goal: Get data into a standard, analyzable form.
- Real-world data may have erroneous values due to measurement errors, subjective judgments, or equipment malfunctions.
- **Missing Values**:
    - Delete the instance.
    - Replace with an average or frequently appearing value from other instances.

Normalization:

- Scaling data to a specific range, typically 0-1.

- Equation: $x' = \frac{x - x_{min}}{x_{max} - x_{min}}$.
    - x: Original value.
    - xmin: Minimum value in the dataset.
    - xmax: Maximum value in the dataset.
    - x′: Normalized value (0 to 1).

---

3. K-Nearest Neighbor (KNN) 📍

A simple algorithm for classification or regression that memorizes training data instead of building a model.

- **How it works**:
    1. **Data Preparation**: Normalize/scale features for equal contribution to distance calculation.
    2. **Choose K**: Select the number of neighbors (k).
    3. **Calculate Distances**: For a new data point, compute its distance to all training points using a chosen metric (e.g., Euclidean, Manhattan).
    4. **Find k-Nearest Neighbors**: Identify the k closest training points.
    5. **Make a Prediction**:
        - **Classification**: Assign the majority class among the k neighbors.
        - **Regression**: Use the average of the target values of the k neighbors.
- **Principle**: Makes predictions based on proximity to 'k' most similar data points.
- **Pros**: Easy to understand and implement.
- **Cons**: Resource-intensive with large datasets, vulnerable to noisy or poorly scaled data.
- **Applications**: Recommendation systems.

KNN Example:
Classify point Q(6,4) with k=3.
Class A points: P1(2,4),P2(5,8),P3(1,6)
Class B points: P4(7,2),P5(9,5)

1. **Choose k**: k=3.
2. **Calculate Euclidean Distances** from Q(6,4):
    - $d(Q,P1) = \sqrt{(6-2)^2 + (4-4)^2} = 4.0$ (Class A)
    - $d(Q,P2) = \sqrt{(6-5)^2 + (4-8)^2} \approx 4.12$ (Class A)
    - $d(Q,P3) = \sqrt{(6-1)^2 + (4-6)^2} \approx 5.39$ (Class A)
    - $d(Q,P4) = \sqrt{(6-7)^2 + (4-2)^2} \approx 2.24$ (Class B)
    - $d(Q,P5) = \sqrt{(6-9)^2 + (4-5)^2} \approx 3.16$ (Class B)
3. **Identify 3 Nearest Neighbors**:
    - P4 (Distance: 2.24, Class: B)
    - P5 (Distance: 3.16, Class: B)
    - P1 (Distance: 4.00, Class: A)
4. **Make Prediction**: Majority class among P4 (B), P5 (B), P1 (A) is Class B (2 out of 3).
    - Q(6,4) is classified as **Class B**.

---

4. Artificial Neural Networks (ANNs) 🧠

Computational models inspired by the brain's parallel processing.

The Computational Neuron:

- Inspired by biological neurons (dendrites, cell body, axon, synapse).
- **Components** (Figure 2):
  - **Inputs (p)**: Vector p=p1,....,pm.
  - **Weights (W)**: Vector W=W1,...,Wm.
  - **Bias (b)**: An additional parameter with an input of 1.
  - **Weighted Sum (n)**: n=Σi=1mwipi+b.
  - **Activation Function (f)**: Determines the neuron's output (e.g., 0 or 1, indicating fire/not fire) based on n and b. Examples: hard limit, linear, sigmoid.
- **Training**: Process of determining weights and biases using a training set and a learning algorithm (e.g., perceptron, backpropagation).

Neural Network Architectures:

- **Single neuron, single layer**: One neuron with multiple inputs (Figure 3).
- **Single-layer network**: Multiple neurons in one layer (Figure 4, s-neurons).
- **Two-layer network**: Example: 2 neurons in layer 1, 1 neuron in layer 2 (Figure 5).
- **Multilayer network**: Multiple layers, each with multiple neurons (Figure 6, 3 layers with S neurons each).

History of Neural Networks:

- 1940s: McCulloch-Pitts Neuron, Hebbian Learning.
- 1950s-1960s (First Golden Age): Perceptrons, Adaline.
- 1970s (Quiet Years): Kohonen, Anderson, Grossberg, Carpenter.
- 1980s (Renewed Enthusiasm): Backpropagation, Hopfield nets, Neocognitron.
- 1990s: LSTMs, gradient-based learning.
- Currently: Deep learning, CNNs.

McCulloch-Pitts Neuron (1940s):

- First neuron model; simple binary neuron (output 0 or 1).
- Uses a **threshold (θ)**: Tuned problem-dependent parameter.
- Activation function: f(n)={10if n≥θ,if n<θ..
- Example: Emulating OR function (2 inputs x1,x2).
  - If θ=2, w1=2, w2=2, the neuron produces OR logic (Table 3).
  - Weights determined manually due to simplicity. For complex networks, learning algorithms are needed.

Linear Separability:

- A function is linearly separable if its output classes can be separated by a single straight line (or hyperplane in higher dimensions).
- Example: OR function is linearly separable. XOR function is **not** linearly separable

(Figure 8).
- Non-linearly separable problems require multilayer neural networks.

---

5. The Single Layer Perceptron

A feedforward neural network for **pattern classification**.

- **Training**: Involves determining weights and bias for each layer using a learning algorithm.
  - **Epochs**: Iterations over the training set.
  - **Convergence**: Algorithm converges when weights/biases produce target output for all training instances.
- Inputs are attributes, output is the class.
  - **Binary classification**: Output is 0 (or -1) or 1 (output vector length 1).
  - **Multiclass classification**: (n>2 classes), output vector length > 1.
  - Input/output can be **binary** (0,1) or **bipolar** (-1,1).

Binary Classification Example (Fruit Sorting):

- Sensors for shape (round=1, elliptical=-1), texture (smooth=1, rough=-1), weight (>0.5kg=1, $\le$0.5kg=-1).
- Apples (class 1) vs. Oranges (class -1).
  - Orange: p=[1,−1,−1] (round, rough, <0.5kg). (Note: Text says "first -1 representing a rough texture", but then uses [1 1 -1] for orange in Table 4, and [1 1 1] for apple. Let's follow Table 4 for consistency in the algorithm example).
  - Table 4:
    - Orange: p=[1,−1,−1], t=−1.
    - Apple: p=[1,1,−1], t=1.
- Architecture: 3 inputs, 1 output (Figure 9).
- Weight matrix W=(w11,w21,w31)T, bias vector b=[b].

Multiclass Classification Example (Fruit Sorting):

- Apples, Oranges, Grapefruit.
- Grapefruit: p=[1,1,1] (round, rough, >0.5kg). (Note: text says "1 representing a rough texture" for grapefruit, which might be a typo if 1 usually means smooth).
- Table 5 input/output vectors.
- Architecture: 3 inputs, 2 outputs (Figure 10).
- Weight matrix W (3×2), bias vector b (1×2).

Perceptron Learning Algorithm (Algorithm 1):

- Applies to linearly separable binary classification problems (single layer). Multiclass usually needs multilayer.
- **Activation Functions**:
  - Binary data: f(n)={10if n≥0,if n<0..
  - Bipolar data: f(n)={1−1if n≥0,if n<0..

    - Where $n=\Sigma w_i p_i+b$ (for single output j=1).
- **Algorithm Steps**:
    1. Initialize weights (wi) and bias (b) to zero or small random values.
    2. **while** algorithm has not converged **do**: 3. **for** each training instance (p,t) **do**: 4. Calculate output f(n). 5. **if** f(n)≠t **then**: 6. Update weights: $w_i=w_i+(t-f(n))\cdot p_i$. 7. Update bias: $b=b+(t-f(n))$. 8. **end if** 9. **end for**
    3. **end while** (An epoch is one pass through all training instances).
- **Alternative update rules with learning rate (α)**:
    - $w_i=w_i+\alpha\cdot(t-f(n))\cdot p_i$.
    - $b=b+\alpha\cdot(t-f(n))$.

Example Walkthrough (Binary Classification, Table 4, Bipolar output -1/1):
Initialize w=[0,0,0], b=0. Using bipolar activation f(n)=1 if n≥0, else −1.

- **Epoch 1**:
    - **Instance 1**: p=[1,−1,−1], t=−1.
        - n=(1)(0)+(−1)(0)+(−1)(0)+0=0.
        - f(n)=1. f(n)≠t.
        - w1=0+(−1−1)(1)=−2.
        - w2=0+(−1−1)(−1)=2.
        - w3=0+(−1−1)(−1)=2.
        - b=0+(−1−1)=−2.
        - Current: w=[−2,2,2], b=−2.
    - **Instance 2**: p=[1,1,−1], t=1.
        - n=(1)(−2)+(1)(2)+(−1)(2)+(−2)=−2−2=−4.
        - f(n)=−1. f(n)≠t.
        - w1=−2+(1−(−1))(1)=−2+2=0.
        - w2=2+(1−(−1))(1)=2+2=4.
        - w3=2+(1−(−1))(−1)=2−2=0.
        - b=−2+(1−(−1))=−2+2=0.
        - Current: w=[0,4,0], b=0.
- **Epoch 2**:
    - **Instance 1**: p=[1,−1,−1], t=−1.
        - n=(1)(0)+(−1)(4)+(−1)(0)+0=−4.
        - f(n)=−1. f(n)=t. No change.
    - **Instance 2**: p=[1,1,−1], t=1.
        - n=(1)(0)+(1)(4)+(−1)(0)+0=4.
        - f(n)=1. f(n)=t. No change.
- Algorithm converges as no changes in Epoch 2. Final: w=[0,4,0], b=0.

Single-layer perceptrons are limited; multi-layer perceptrons (MLPs) with backpropagation are often needed.

---

6. Multilayer Perceptron (MLP)

An ANN with multiple layers: input, one or more hidden layers, and an output layer. Data flows one way (feedforward).

- **Structure**:
  - **Input Layer**: Receives features.
  - **Hidden Layers**: Perform computations. Neurons apply weighted sum, add bias, pass through non-linear activation function (e.g., sigmoid, ReLU).
  - **Output Layer**: Produces final output, with activation suited to task (e.g., softmax for classification, linear for regression).
- **Activation Functions**: Non-linear functions (sigmoid, tanh, ReLU) enable modeling complex, non-linear relationships.
- **Training**: Uses **backpropagation** and **gradient descent** to minimize a loss function (e.g., MSE for regression, cross-entropy for classification). Gradients update parameters iteratively.
- **Applications**: Classification, regression, pattern recognition, image/speech processing, NLP, financial forecasting.
- **Advantages**:
  - Can model complex, non-linear relationships.
  - **Universal Approximation Theorem**: With enough neurons, MLPs can approximate any continuous function.
- **Challenges**:
  - Prone to **overfitting**, especially with large networks.
  - Requires careful **hyperparameter tuning** (layers, neurons, learning rate).
  - Computationally expensive for deep networks.
- MLPs are foundational for modern deep learning models.

Backpropagation Algorithm:
First algorithm for training MLPs; also called generalized delta rule.

- **Main Processes**:
  1. Feedforward training (input to output layer).
  2. Error at output layer is propagated backward.
  3. Weights and biases updated based on propagated error.
- A gradient descent algorithm; versions vary (e.g., weight update procedures).
- Architecture: Single input layer, one/more hidden layers, single output layer. Nodes in input/output layers match number of inputs/outputs. Hidden layer nodes are a design decision.
- **Activation Functions (f(n))**: Must be differentiable.
  - Binary Sigmoid: $f(n)=1+\exp(-n)1$, $f'(n)=f(n)(1-f(n))$.
  - Bipolar Sigmoid: $f(n)=1+\exp(-n)2-1$, $f'(n)=21(1+f(n))(1-f(n))$.
- **Example Architecture (Figure 11)**: 1 input layer (3 inputs), 1 hidden layer (2 nodes, h1,h2), 1 output layer (2 outputs, t1,t2). Biases v0j to hidden, w0k to output.
- **Notation for one hidden layer**:

- pl: input l.
- vlj: weight from input l to hidden node j. v0j is bias to hidden node j.
- wjk: weight from hidden node j to output node k. w0k is bias to output node k.
- n1j: weighted sum at hidden node j. f(n1j): activation of hidden node j.
- n2k: weighted sum at output node k. f(n2k): activation of output node k.
- tk: target output for output node k.
- α: learning rate.

Backpropagation Algorithm Steps (Conceptual):

- **Algorithm 2: Main Loop**

  1. Initialize weights and biases (small random values). Range e.g., -0.5 to 0.5.
  2. **while** stopping condition not met **do** (e.g., no error change, max epochs): 3. Perform feedforward learning (Algorithm 3). 4. Backpropagate error (Algorithm 4). 5. Update weights (Algorithm 5).
  3. **end while**

- **Algorithm 3: Feedforward Learning**

  1. Calculate $n1j=v0j+\Sigma l=1nvlj \cdot pl$ for each hidden node j. (n=num inputs)
  2. Calculate f(n1j) for each hidden node j.
  3. Calculate $n2k=w0k+\Sigma j=1hwjk \cdot f(n1j)$ for each output node k. (h=num hidden nodes)
  4. Calculate f(n2k) for each output node k.

- **Algorithm 4: Backpropagation of Error** (j=num hidden nodes, k=num output nodes, l=num inputs)

  1. **Output Layer Error**:
     - Error term: $\delta k=(tk-f(n2k))f'(n2k)$ for each output node k.
     - Weight correction: $\Delta wjk=\alpha \delta k f(n1j)$ for each weight wjk.
     - Bias correction: $\Delta w0k=\alpha \delta k$ for each bias w0k.
  2. **Hidden Layer Error**:
     - Sum of delta inputs: $\delta nj=\Sigma k=1m \delta kwjk$ for each hidden node j. (m=num outputs)
     - Error term: $\delta j=\delta njf'(n1j)$ for each hidden node j.
     - Weight correction: $\Delta vlj=\alpha \delta jpl$ for each weight vlj. (Equation 15 is repeated in source)
     - Bias correction: $\Delta v0j=\alpha \delta j$ for each bias v0j.

- **Algorithm 5: Updating Weights**

  1. Update output layer weights/biases: $wjk(new)=wjk(old)+\Delta wjk$ (includes bias w0k).
  2. Update hidden layer weights/biases: $vlj(new)=vlj(old)+\Delta vlj$ (includes bias v0j).

- **Testing Phase**: Apply trained network to unseen instances. Calculate weighted sums and apply activation functions to get output.

---

7. Hopfield Neural Network

Used for **pattern association**, memorizing patterns and recognizing them even if noisy or incomplete.

- **Associative Memory**:
  - **Autoassociative**: Input and output vectors are the same (network remembers the input pattern).
  - **Heteroassociative**: Input and output vectors are different (network memorizes association between two patterns).
- Hopfield network is a **recurrent, autoassociative** memory network. It's a single-layer network.
- Example (Figure 12): 4 inputs p1–p4, 4 outputs y1–y4.
- **Training**: Determine the n×n weight matrix W (where n is length of input vector).
  - Diagonal weights wii=0.
  - For **binary** patterns (0,1): wij=Σe[2pi(e)−1][2pj(e)−1] for i偃=j, over all training patterns e. (This converts binary to bipolar then calculates outer product sum).
  - For **bipolar** patterns (-1,1): wij=Σepi(e)pj(e) for i偃=j, over all training patterns e. (Sum of outer products).
- **Application (Algorithm 6 - Pattern Recall)**: Corrects a noisy/incomplete input pattern p.
  1. Initialize output y=p.
  2. **while** algorithm has not converged (no change in y) **do**: 3. **while** all components yi not updated in this epoch **do**: 4. Randomly select a component yi to update. 5. Calculate input to neuron i: si=pi+Σj=1,j偃=inyjwji (Note: Source has pi+Σyjwji, but standard Hopfield often uses Σyjwji or similar. Let's follow source si=pi+Σj=1nyjwji. The provided sum is for j=1 to n, which includes wii which should be 0. The problem description example implies pi from the *current test pattern* is added). 6. Apply activation function (threshold logic): * **if** si>θi **then** yi=1 * **else if** si=θi **then** yi=yi (no change) * **else** (si<θi) yi=0 (for binary output). * θi is a threshold, often 0. 7. Update input vector p according to changes in y (for next component update in same epoch). 8. **end while** (epoch ends)
  3. **end while** (convergence check)

Hopfield Example: Train to store binary pattern P=[1,1,1,0]. Assume θi=0 for all i.

1. Training: Convert P to bipolar: Pbipolar=[1,1,1,−1].
   $W = P_{bipolar}^T P_{bipolar} - I \cdot (\text{num_patterns})$. For one pattern, W=111−1(111−1)=111−1111−1111−1−1−1−11. Set diagonals to 0: W=011−1101−1110−1−1−1−10.
2. Application: Correct input pattern ptest=[0,0,1,0].
   Initialize y=[0,0,1,0]. Let pcurrent_state=y. (The example uses pi from the initial ptest for si calculation. This seems to be a specific variant).
   - **Epoch 1**:

        - Update y1: $s1 = p_{test,1} + (y1w11 + y2w21 + y3w31 + y4w41) = 0 + (0 \cdot 0 + 0 \cdot 1 + 1 \cdot 1 + 0 \cdot (-1)) = 1$. Since $s1 = 1 > \theta 1 = 0 \Rightarrow y1 = 1$. Update y=[1,0,1,0], pcurrent_state=[1,0,1,0].
        - Update y4: $s4 = p_{test,4} + (y1w14 + y2w24 + y3w34 + y4w44) = 0 + (1 \cdot (-1) + 0 \cdot (-1) + 1 \cdot (-1) + 0 \cdot 0) = -2$. Since $s4 = -2 < \theta 4 = 0 \Rightarrow y4 = 0$. Update y=[1,0,1,0], pcurrent_state=[1,0,1,0].
        - Update y2: $s2 = p_{test,2} + (y1w12 + y2w22 + y3w32 + y4w42) = 0 + (1 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 + 0 \cdot (-1)) = 2$. Since $s2 = 2 > \theta 2 = 0 \Rightarrow y2 = 1$. Update y=[1,1,1,0], pcurrent_state=[1,1,1,0].
        - Update y3: $s3 = p_{test,3} + (y1w13 + y2w23 + y3w33 + y4w43) = 1 + (1 \cdot 1 + 1 \cdot 1 + 1 \cdot 0 + 0 \cdot (-1)) = 1 + 2 = 3$. Since $s3 = 3 > \theta 3 = 0 \Rightarrow y3 = 1$. Update y=[1,1,1,0], pcurrent_state=[1,1,1,0].
    - Output y=[1,1,1,0]. This is the stored pattern. Another epoch would show no changes, so convergence.

---

8. Deep Neural Networks (DNNs) 🚀

Neural networks with several hidden layers, potentially hundreds.

- Can be feedforward or recurrent; perform supervised or unsupervised learning.
- Performance measured by **accuracy** and **loss** (from a loss function).
- Significant contributions in **image processing** and **automatic speech recognition**.
- Learning is incremental, layer by layer.
- **Loss Function**: Measures difference between network activation and target values. Measures "loss" of the network. Weights updated to reduce this loss. Choice depends on task (regression/classification).
    - **Regression**: Mean Squared Error (MSE - common), Mean Squared Logarithmic Error, Mean Absolute Error.
    - **Binary Classification**: Binary Cross-Entropy, Hinge Loss, Squared Hinge Loss.
    - **Multiclass Classification**: Multiclass Cross-Entropy, Sparse Multiclass Cross-Entropy, Kullback-Leibler Divergence.
- **Optimizers**: Update weights to reduce loss.
    - Examples: Backpropagation, Gradient Descent (GD), Stochastic GD (SGD), Mini-batch GD, Momentum, Nesterov Accelerated Gradient, Adagrad, AdaDelta, Adam, RMSProp.
    - **Adam** often best for training time and efficiency.
    - Optimizers usually require a **learning rate**, a problem-dependent hyperparameter.
    - If using backpropagation, loss function must be differentiable.
- **Activation Functions**: Can vary between layers; output layer choice is important.
    - Commonly used: Sigmoid, Softmax, Tanh, ReLU (Rectified Linear Unit), Leaky ReLU, Swish.

- **Types of DNNs**: Convolutional Neural Networks (CNNs), Autoencoders, Long Short-Term Memory (LSTM), Restricted Boltzmann Machines (RBMs), Deep Belief Networks.
- **Overfitting**: High accuracy on training set, low accuracy on test set (poor generalization). Always report training and test set accuracy; base conclusions on test set.
    - **Solutions**:
        - **Regularization**.
        - **Early stopping** of learning.
        - Reducing network size.
        - **Dropout**.
- DNNs require sufficient data. If data is scarce:
    - **Transfer Learning**: Pretrain a network on a large dataset, then adapt it for a specific task with less data.
        - Reuses features and/or weights.
        - Techniques: Keep all layers fixed and retrain only the last layer, or selectively retrain layers.
        - **ImageNet**: Common database for pretraining computer vision models.
        - Pretrained models often available via Python libraries.
        - Examples (Vision): VGG16, VGG19, InceptionV3, XCeption, ResNet50.
        - Examples (NLP): Word2Vec, GloVe, FastText.

---

9. Convolutional Neural Networks (CNNs / ConvNets) 🖼️

Popularized by AlexNet (ImageNet 2012 winner). Highly effective for **image classification**.

- **Input**: Image is converted to a matrix of pixel values (0-255).
    - Color images: 3 (Red, Green, Blue) 2D arrays stacked.
    - Grayscale images: Single 2D array.
    - Matrix size depends on image resolution; preprocessing may reduce scale.
- **Connectivity**: Not all layers are fully connected; some are **sparsely connected** (Figures 13 & 14).
    - Sparsely connected layers lead to **parameter sharing** (shared weights).
- **Typical Layers**:
    - **Convolutional Layers**:
        - Usually the first layer(s) after input, can also be deeper.
        - Applies a **convolution operator** instead of full matrix multiplication.
        - Focuses learning on local image aspects (e.g., edges).
        - **Filter/Kernel**: A small weight matrix (kernel is 2D, filter can be multiple kernels for 3D structure). Values initially random or operator-dependent.
        - Output: **Feature Map** (result of applying filter to image pixels).
        - Convolution operators: Identity, edge detection, sharpening, box blur.

        - Operation: Filter slides over input (image/feature map).
            - **Receptive Field**: Area of input the filter covers (same dimensions as filter).
            - **Stride**: Number of pixels filter moves at a time (usually 1; larger stride reduces dimensionality).
            - **Zero Padding**: Adding a border of zeros to input; a hyperparameter.
        - Sparsely connected; reduces dimensionality.
    - **Nonlinear (ReLU) Layers**:
        - Add nonlinearity. Activation: f(x)=max(0,x).
    - **Pooling (e.g., Max Pooling) Layers**:
        - Reduce number of parameters (downsample feature map size).
        - Helps reduce overfitting.
    - **Fully Connected Layers**:
        - At least the final output layer is fully connected; sometimes last few layers.
        - **Softmax** activation often used in output units for classification.
        - This is often the layer retrained in transfer learning.
- **Architectures**:
    - Can be created from scratch or use existing ones (often pretrained on ImageNet).
    - Examples: LeNet (first CNN), AlexNet, GoogLeNet, VGGNet, ResNet, DenseNet.

---

10. Autoencoders

Deep learning NNs for **unsupervised learning** (no labelled training data).

- Input is the same as the target output.
- Learns to reconstruct its input, often in a different (e.g., compressed) representation.
- **Applications**: Dimensionality reduction, image compression, information retrieval.
- **Structure**: Feedforward network with typically three parts:
    - **Input Layer**: Receives input (e.g., image to be compressed).
    - **Hidden Layer(s) (Encoder)**: Extracts features, produces a compressed "code" or representation (bottleneck). Can be multiple hidden layers.
    - **Output Layer (Decoder)**: Reconstructs the input from the code.
- Both encoder and decoder are typically fully connected.
- **Training**: Find weights/biases. Weighted sum + bias -> activation.
    - **Activation Functions**: Sigmoid, ReLU commonly used.
    - **Optimizers**: Backpropagation, SGD.
    - **Loss Function**: Measures reconstruction accuracy (e.g., Mean Squared Error, Binary Cross-Entropy).

---

11. Large Language Models (LLMs) 💬

A category of neural networks known for massive scale and impressive language capabilities,

using deep learning (specifically **transformer architectures**).

- Trained on vast amounts of text data to comprehend and generate human-like language.
- Revolutionized Natural Language Processing (NLP) in tasks like text generation, translation, summarization, question answering.
- Examples: GPT-4, PaLM, Llama.
- **Scale**: Enormous size, often billions or trillions of parameters, enabling capture of language complexity and nuanced patterns.
- **Transformer Architecture**: Core design using **self-attention mechanisms** to process input sequences efficiently.
    - Handles long-range dependencies in text, crucial for context and coherence.
    - Scalability allows utility across diverse domains (creative writing to coding).
- **Training Process**:
    1. **Pretraining**: Model exposed to massive unlabeled text data to learn general linguistic patterns.
    2. **Fine-tuning**: Foundational knowledge honed by training on labeled, task-specific datasets to optimize for particular applications.
- **Key Attributes**:
    - **Effective Generalization**: Excel at tasks not explicitly trained for, adaptable.
    - **Emergent Abilities**: As size grows, exhibit new capabilities like multi-step reasoning, code generation, few-shot/zero-shot learning (performing tasks with little/no examples).

---

12. Decision Trees 🌳

Supervised learning models, essentially classifiers, trained to generalize to unseen data.

- Advantage: **Interpretable** (Figure 15).
- Can overfit if allowed infinite size; size is often limited, or tree is **pruned**.
- Created using **induction algorithms** (e.g., ID3).

Dominance Measurements (for attribute selection):

- Used by induction algorithms to pick the best attribute at each tree node. Illustrated with Netball dataset (Table 6).
    - Attributes: Outlook, Temperature, Humidity, Wind. Decision D: Play Netball (Yes/No).
- **Entropy E(D)**: Measures impurity/uncertainty of a set of examples D.
    - $E(D)=\Sigma i=1n-p(i)\log 2p(i)$.
    - p(i): probability of class i (Num instances of class i / Total instances).
    - Example (Netball): 9 Yes, 5 No. Total 14.
        - $p(yes)=149$, $p(no)=145$.
        - $E(D)=-(149)\log 2(149)-(145)\log 2(145)\approx 0.94$.

- **Information Gain G(D,A)**: Expected reduction in entropy by splitting on attribute A.
  - $G(D,A) = E(D) - \Sigma_{j=1}^{m} p(D|A=j)E(D|A=j)$.
  - j: j-th value of attribute A. m: number of values for A.
  - $p(D|A=j)$: (Num instances with A=j) / (Total instances).
  - $E(D|A=j)$: Entropy of subset of D where attribute A has value j.
  - Example (Gain for Wind):
    - $E(D) = 0.94$.
    - Wind values: Weak (8 instances), Strong (6 instances).
    - $p(D|wind=weak) = \frac{8}{14}$, $p(D|wind=strong) = \frac{6}{14}$.
    - $E(D|wind=weak)$: For 8 'Weak' instances (6 Yes, 2 No) $\Rightarrow -\frac{6}{8}\log_2\frac{6}{8} - \frac{2}{8}\log_2\frac{2}{8} \approx 0.81$.
    - $E(D|wind=strong)$: For 6 'Strong' instances (3 Yes, 3 No) $\Rightarrow -\frac{3}{6}\log_2\frac{3}{6} - \frac{3}{6}\log_2\frac{3}{6} = 1$.
    - $G(D,wind) = 0.94 - [(\frac{8}{14})(0.81) + (\frac{6}{14})(1)] \approx 0.048$.

ID3 (Iterative Dichotomizer 3) Algorithm (Algorithm 7):

- Greedy algorithm that iteratively builds a decision tree top-down.
- **Steps**:
  1. If all examples S have the same classification c, return a leaf node labeled c.
  2. If attribute set A is empty, return a leaf node labeled with the most common class in S.
  3. Else: a. Select best_attribute from A that has the **highest information gain** on S. b. Create a decision node for best_attribute. c. Remove best_attribute from A. d. For each possible value vi of best_attribute: i. Let Si be the subset of S where best_attribute =vi. ii. If Si is empty, add a leaf labeled with the most common class in S as a child. iii. Else, add the subtree ID3(Si,A) as a child.
- Example tree for Netball data (Figure 16).
- Shortcoming: Can produce large trees (overfitting). Addressed by limiting tree depth or pruning.

---

13. Random Forests 🌲🌲🌲

An **ensemble learning** method using multiple decision trees for classification.

- **Process**:
  1. Choose multiple subsets Si from the training set T (often with replacement - bootstrapping).
  2. Create a decision tree classifier Ci for each subset Si. Different induction algorithms can be used for each tree. Feature randomness is also often introduced (each tree considers only a subset of features for splitting).
  3. Classify a new instance using **majority voting** among all trees in the forest. The class predicted by the most trees is the final output.

14. Unsupervised Learning (Clustering) KMeans

Focuses on finding patterns in unlabelled data.

- **Clustering**: Divides data into groups (clusters) where instances within a cluster are similar, and clusters are dissimilar to each other.
- Similarity is measured using a distance metric (e.g., Euclidean distance), dependent on the algorithm.
- K-Means is a common clustering algorithm.

K-Means Algorithm (Algorithm 8):

- Clusters data into K predefined clusters. Standard version for numerical data.
- **K**: Number of clusters. Can be chosen randomly or empirically (e.g., elbow method).
- **Algorithm Steps**:
    1. Given data D=d1,...,dn.
    2. Determine K (number of clusters).
    3. Randomly select K initial **centroids** (c1,...,cK) from the data points or space.
    4. **while** algorithm has not converged (centroids don't change much, or cluster assignments stabilize) **do**: a. **Assignment Step**: For each data instance di: i. Calculate Euclidean distance ej=Σ(dil−cjl)2 from di to each centroid cj. ii. Assign di to the cluster kj with the smallest ej. b. **Update Step**: For each cluster kj: i. Recalculate its centroid cj as the mean of all data instances di assigned to cluster kj. (cjl=∣Sj∣1Σdi∈Sjdil for each dimension l).
    5. **end while**.
- Convergence: When no data instances move between clusters after an iteration, or centroids stabilize.

K-Means Example (Table 7 data, K=2).
Entities: E1(1,1), E2(1.5,2), E3(3,4), E4(5,7), E5(3.5,5).
Initial centroids: $C_1=$E2(1.5,2), $C_2=$E4(5,7).

- **Iteration 1 - Assignment** (Table 8):
    - E1 to C1: dist=1.12; E1 to C2: dist=7.21 ⟹ E1 in Cluster 1.
    - E3 to C1: dist=2.5; E3 to C2: dist=3.61 ⟹ E3 in Cluster 1.
    - E5 to C1: dist=3.61; E5 to C2: dist=2.5 ⟹ E5 in Cluster 2.
    - Clusters: Cluster 1 = {E1, E2, E3}, Cluster 2 = {E4, E5}.
- **Iteration 1 - Update Centroids**:
    - New C1: mean(E1,E2,E3) = (31+1.5+3, 31+2+4) = (1.83, 2.33).
    - New C2: mean(E4,E5) = (25+3.5, 27+5) = (4.25, 6).
- **Iteration 2 - Assignment** (Tables 9 & 10):
    - Distances from E1, E2, E3 to new C1 are smaller than to new C2.
    - Distances from E4, E5 to new C2 are smaller than to new C1.
    - No data instances change clusters.
- **Convergence**: Algorithm converged.

K-Means Advantages:

- Simplicity and ease of use; intuitive iterative process.
- Scalability and rapid convergence, especially with distinct clusters or good initial centroids. Efficient for large datasets.

K-Means Limitations:

- Assumes clusters are spherical and evenly distributed; may perform poorly on non-spherical/irregular clusters.
- Sensitive to initial placement of centroids, can lead to varied outcomes.
- Requires the number of clusters (K) to be specified beforehand, which can be difficult.
- Prone to influence by noise and outliers.
- Understanding limitations is crucial for effective use.

Good luck with your exam! 🍀