UNIT–4 : Artificial Neural Networks and Deep Learning (PYQ Solution)

AKTU · Machine Learning Techniques (BCDS062) · Solutions by **Chirag Jain**, Ycotes

SECTION A Answers (≈70 Words Each)

Q1. What is a Perceptron? Explain the role of the activation function. (May 2026, Jan 2025)

A perceptron is the simplest neural unit: it computes a weighted sum of inputs plus bias, z = Σ wi xi + b, and passes it through an activation function to give the output, o = 1 if z > 0 else −1 (or 0). The activation introduces non-linearity and decides whether the neuron "fires"; without it, stacked layers would collapse into one linear map. Common choices: step, sigmoid, tanh, ReLU.

Q2. What is backpropagation in neural networks? (Jan 2026)

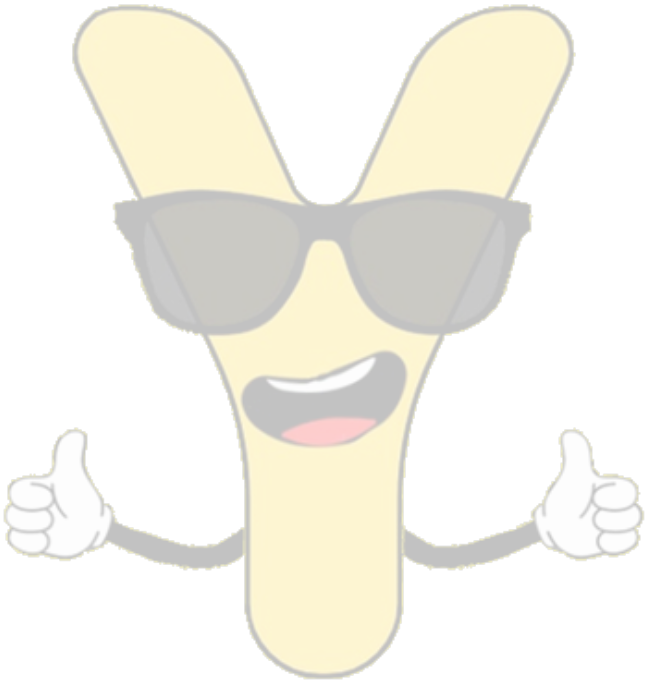
Backpropagation is the algorithm that trains multilayer networks. A forward pass computes outputs and the error; the error is then propagated backwards using the chain rule to compute the gradient of the loss with respect to every weight. Weights are updated by gradient descent, w ← w − η ∂E/∂w, and the cycle repeats until the error is small.

Q3. What is the gradient descent delta rule? (Jan 2024)

The delta rule trains a linear unit by minimising E(w) = ½ Σ (td − od)². Weights move opposite to the gradient: Δwi = η Σ (td − od) xid, where η is the learning rate, t the target and o the output. Incremental (stochastic) form: Δwi = η (t − o) xi, updated after each example. It converges towards the minimum-error weights.

Q4. Define ANN and CNN. (Jan 2024)

An Artificial Neural Network (ANN) is a computing model inspired by the brain, made of interconnected neurons arranged in input, hidden and output layers, whose weights are learned from data. A Convolutional Neural Network (CNN) is a deep ANN for grid-like data such as images; it uses convolution layers with shared filters, pooling and fully connected layers to learn spatial features automatically.



SECTION B Answers (≈250 Words Each)

Q1. Derive and explain the Backpropagation algorithm for training multilayer neural networks. (May 2026)

Backpropagation (BP) trains a feed-forward network by gradient descent on the squared error E = ½ Σk (tk − ok)².

Notation: sigmoid units, o = σ(net) = 1 / (1 + e^-net), with derivative σ′ = o(1 − o). Weight wji connects input i to unit j; η is the learning rate.

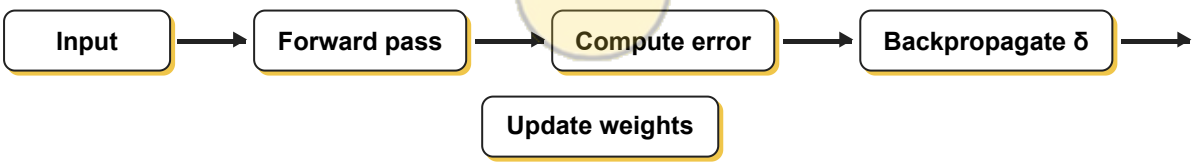
Output unit k: by the chain rule, ∂E/∂wkj = ∂E/∂ok × ∂ok/∂netk × ∂netk/∂wkj = −(tk − ok) · ok(1 − ok) · xkj. Define the error term **δk = ok(1 − ok)(tk − ok)**. Then

Δwkj = η δk xkj

Hidden unit h: its error is the weighted sum of the errors of the units it feeds:
δh = oh(1 − oh) Σk wkh δk
Δwhi = η δh xhi

Algorithm:

1. Initialise all weights to small random values.
2. Until the stopping condition is met, for each training example:
   - Forward pass: compute the output of every unit.
   - For each output unit compute δk = ok(1 − ok)(tk − ok).
   - For each hidden unit compute δh = oh(1 − oh) Σ wkh δk.
   - Update every weight: wji ← wji + η δj xji.



Backpropagation cycle

Notes: a momentum term Δw(n) = η δ x + α Δw(n−1) speeds convergence; BP can get stuck in local minima, but works well in practice, and can represent any function given enough hidden units.

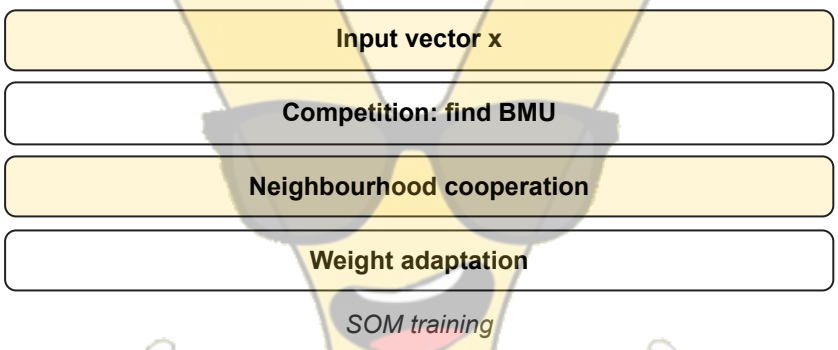
Q2. Discuss the Self-Organizing Map (SOM) algorithm. How does it perform clustering and dimensionality reduction, and how does it differ

from supervised models? (Jan 2025)

A **Self-Organizing Map (Kohonen map)** is an unsupervised neural network that maps high-dimensional input onto a low-dimensional (usually 2-D) grid of neurons while preserving **topological neighbourhood** relations.

Algorithm:

1. Initialise the weight vector wj of each grid neuron randomly.
2. Pick an input vector x.
3. **Competition:** find the **best matching unit (BMU)**, the neuron with minimum distance ‖x − wj‖.
4. **Cooperation:** define a neighbourhood around the BMU that shrinks with time.
5. **Adaptation:** update neighbours: wj(t+1) = wj(t) + η(t) h(j, t) (x − wj(t)), where h is the neighbourhood function (largest at the BMU).
6. Reduce η and the neighbourhood radius; repeat for many epochs.



SOM training

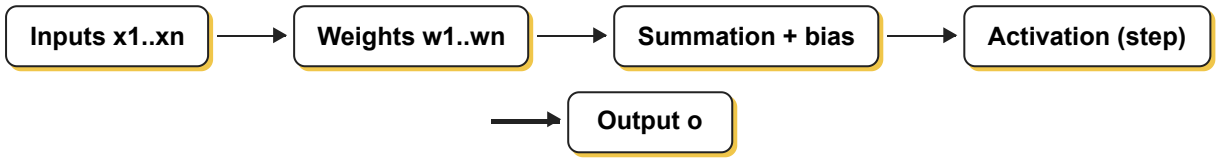
Clustering: similar inputs win neighbouring neurons, so groups of neurons form clusters. **Dimensionality reduction:** each n-dimensional input is represented by its 2-D BMU position, keeping similarity relations (U-matrix visualisation).

Basis	SOM	Supervised models
Labels	Not used	Required
Goal	Structure, clusters, visualisation	Predict targets
Learning	Competitive, no error signal	Error-driven (backprop)
Output	Grid map	Class/value

SECTION C Answers (≈500 Words Each)

Q1. Explain perceptron and multilayer perceptron with a neat diagram. (Jan 2026)

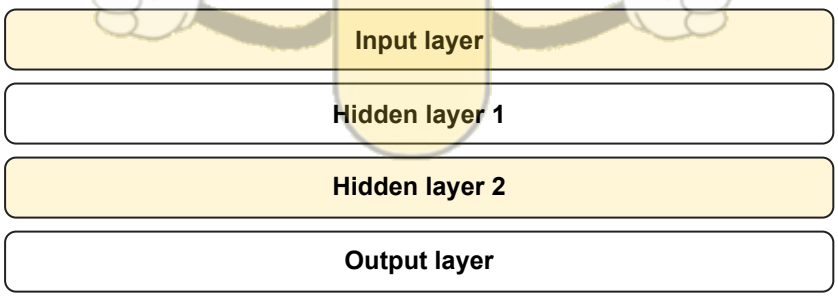
Perceptron: a single neuron proposed by Rosenblatt. Inputs x1…xn are multiplied by weights w1…wn, summed with bias w0, and passed through a threshold: o = +1 if Σ wi xi + w0 > 0, else −1.



Perceptron

Learning rule: wi ← wi + η (t − o) xi. If the output is correct nothing changes; otherwise the weights are nudged towards the target. It converges only for **linearly separable** data (AND, OR) and cannot solve **XOR**.

Multilayer perceptron (MLP): a feed-forward network with an **input layer**, one or more **hidden layers** and an **output layer**. Each neuron is connected to all neurons of the next layer, and hidden units use **non-linear activations** (sigmoid, tanh, ReLU). Hidden layers build intermediate feature representations, so an MLP can learn **non-linear decision boundaries** such as XOR and, in theory, approximate any continuous function (universal approximation).



Multilayer perceptron

Training: the MLP is trained with **backpropagation**: forward pass, error computation, backward propagation of error, weight update by gradient descent.

Basis	Perceptron	MLP
Layers	One	Two or more weighted layers
Boundary	Linear	Non-linear
XOR	Cannot solve	Can solve
Training	Perceptron rule	Backpropagation

Applications: the MLP is used in classification, speech and character recognition, and prediction.

Conclusion: the perceptron is the building block; stacking such units with non-linear activations gives the MLP its power.

Q2. Derive the Backpropagation algorithm (mathematical steps), with the training rules for output-unit and hidden-unit weights. (Jan 2026, Jan 2025, Jan 2024)

Consider a network of sigmoid units trained to minimise $E(w) = \frac{1}{2} \Sigma$ over outputs k of $(tk - ok)^2$. For unit j, $netj = \Sigma i\ wji\ xji$ and $oj = \sigma(netj) = 1 / (1 + e^{-netj})$, with $\sigma'(net) = oj(1 - oj)$. Gradient descent gives $\Delta wji = -\eta\ \partial E/\partial wji$. Since wji affects E only through netj:

$\partial E/\partial wji = (\partial E/\partial netj) \cdot xji$. Let $\delta j = -\partial E/\partial netj$. Then **$\Delta wji = \eta\ \delta j\ xji$.**

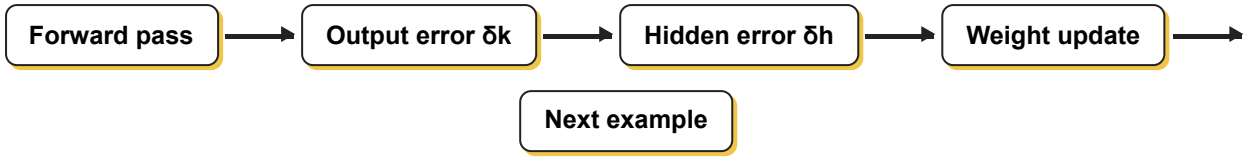
Case 1: output unit k. netk affects E only through ok:
$\partial E/\partial netk = (\partial E/\partial ok)(\partial ok/\partial netk) = -(tk - ok) \cdot ok(1 - ok)$
So **$\delta k = ok(1 - ok)(tk - ok)$**, and $\Delta wkj = \eta\ \delta k\ xkj$.

Case 2: hidden unit h. netj influences E through every unit k that receives the output of h (Downstream(h)):
$\partial E/\partial neth = \Sigma k\ (\partial E/\partial netk)(\partial netk/\partial neth) = \Sigma k\ -\delta k \cdot wkh \cdot oh(1 - oh)$
So **$\delta h = oh(1 - oh)\ \Sigma k\ wkh\ \delta k$**, and $\Delta whi = \eta\ \delta h\ xhi$.

Algorithm summary:

1. Initialise weights with small random numbers.
2. For each training example:
   - Propagate the input forward and compute the output o of every unit.
   - For each output unit k: $\delta k = ok(1 - ok)(tk - ok)$.
   - For each hidden unit h: $\delta h = oh(1 - oh)\ \Sigma\ wkh\ \delta k$.
   - Update each weight: $wji \leftarrow wji + \eta\ \delta j\ xji$.

3. Repeat until the error is below a threshold.



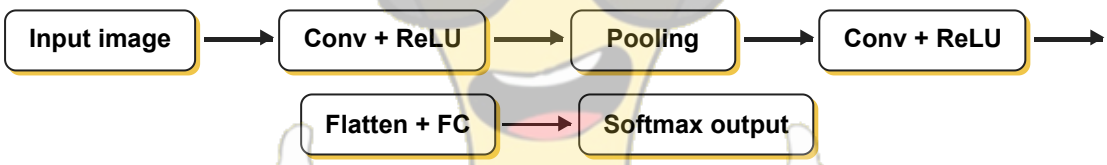
Backpropagation steps

Worked mini-example: one output unit with o = 0.6, t = 1, input x = 0.5, η = 0.1. δ = 0.6 × 0.4 × 0.4 = 0.096, so Δw = 0.1 × 0.096 × 0.5 = 0.0048.

Conclusion: the error term of a hidden unit is the weighted sum of downstream error terms, which is why the error is said to be propagated backwards.

Q3. Explain the architecture and functioning of Convolutional Neural Networks, with the roles of convolution, activation, pooling and fully connected layers. (May 2026, Jan 2026, Jan 2025)

A **CNN** is a deep network designed for grid-like data (images). It exploits **local connectivity**, **weight sharing** and **spatial hierarchy**, which reduces parameters compared with a fully connected network.



CNN architecture

Layers:

1. **Input layer:** raw pixels, e.g. 32 × 32 × 3 (width, height, RGB).
2. **Convolution layer:** small learnable filters (kernels, e.g. 3 × 3) slide over the image with a given **stride** and **padding**, computing dot products to produce **feature maps** (edges, textures, then shapes). Output size = (W − F + 2P)/S + 1. Filters share weights across positions.
3. **Activation layer:** usually **ReLU**, f(x) = max(0, x), adds non-linearity and avoids vanishing gradients.
4. **Pooling layer:** down-samples feature maps (**max pooling** 2 × 2 keeps the maximum), reducing size and computation, and giving small translation invariance and less overfitting.
5. **Fully connected layer:** the flattened features go to dense layers that combine them for decisions.

6. **Output layer: softmax** gives class probabilities.

Functioning: early layers detect low-level features (edges), middle layers parts (eyes, wheels), deep layers whole objects. Training uses **backpropagation** with gradient descent on cross-entropy loss; dropout and batch normalisation help regularise.

Example: a 28 × 28 digit image with 6 filters of 5 × 5 gives 24 × 24 × 6 maps; 2 × 2 pooling gives 12 × 12 × 6, and so on to 10 output classes (LeNet).

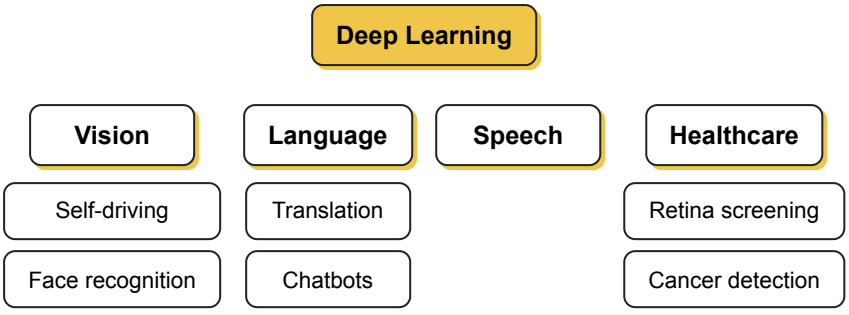
Applications: image classification, object detection, medical imaging, face recognition.

Conclusion: CNNs learn hierarchical features automatically, making them the standard for vision tasks.

Q4. Discuss applications of Deep Learning with suitable case studies. (Jan 2026)

Deep learning uses many-layered neural networks to learn features directly from large data. Major applications with case studies:

- **Computer vision:** image classification and detection. Case study: **AlexNet** on ImageNet (2012) cut error sharply; self-driving cars use CNNs to detect lanes, pedestrians and signs (Tesla, Waymo).
- **Healthcare:** detecting diabetic retinopathy from retinal images and cancer from X-rays and pathology slides, with accuracy comparable to specialists (Google's retinal screening system).
- **Natural language processing:** machine translation (Google Translate uses transformer models), chatbots, sentiment analysis, large language models.
- **Speech recognition:** Siri, Alexa and Google Assistant use deep recurrent and transformer models for voice commands.
- **Recommendation systems:** Netflix, YouTube and Amazon use deep models to personalise content.
- **Games and robotics: AlphaGo** (deep reinforcement learning) defeated the Go world champion in 2016.
- **Finance and security:** fraud detection, credit scoring, face-based authentication.
- **Generative AI:** GANs and diffusion models create images, music and text.



Deep learning applications

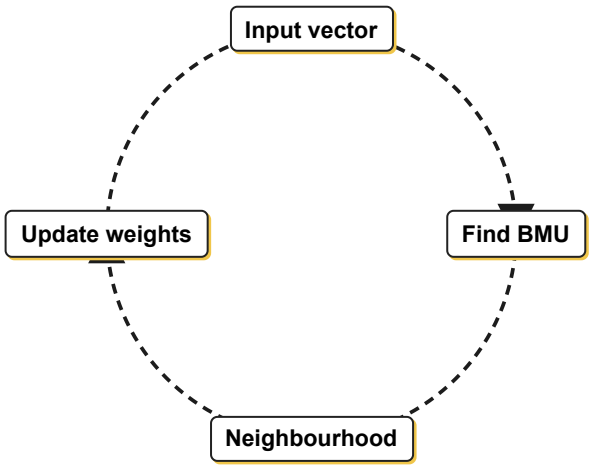
Conclusion: deep learning excels where large data and complex patterns exist.

Q5. What is a Self-Organizing Map (SOM), and how does it perform unsupervised learning? (May 2026)

A **Self-Organizing Map** (Kohonen network) is an unsupervised neural network with an input layer fully connected to a 2-D grid of output neurons; it produces a low-dimensional, **topology-preserving** representation of the input data.

Learning (competitive learning):

1. **Initialise** each neuron's weight vector w_j randomly.
2. **Present** an input vector x.
3. **Competition:** the neuron whose weight vector is closest to x (smallest Euclidean distance $\|x - w_j\|$) is the **winner (BMU)**.
4. **Cooperation:** neurons near the BMU on the grid form its neighbourhood, with a Gaussian strength $h(j) = \exp(-d^2/2\sigma^2)$.
5. **Adaptation:** $w_j \leftarrow w_j + \eta\, h(j)(x - w_j)$, pulling the winner and its neighbours towards x.
6. Decrease the learning rate η and the radius σ over iterations and repeat.



SOM learning loop

No target labels or error signal are used, so learning is unsupervised: the map self-organises so that similar inputs activate neighbouring neurons, forming **clusters**.

Example: a SOM given colour (R, G, B) vectors arranges similar colours next to each other on the grid.

Applications: clustering, data visualisation, dimensionality reduction, speech recognition, customer segmentation.

Conclusion: SOM discovers the structure of unlabelled data using competition and neighbourhood cooperation.

Q6. Write short notes on the Probably Approximately Correct (PAC) learning model. (Jan 2024)

PAC learning (Valiant, 1984) is a framework that asks how many training examples a learner needs so that, with high probability, it outputs a hypothesis with low error.

Setup: instances X drawn from an unknown but fixed distribution D; the target concept c comes from a concept class C; the learner L outputs a hypothesis h from the hypothesis space H.

- **True error:** error_D(h) = Pr over x drawn from D [c(x) ≠ h(x)].
- **Approximately correct:** error_D(h) ≤ ε.
- **Probably:** this holds with probability at least 1 − δ.

Definition: C is PAC-learnable by L using H if, for all c in C, all distributions D, and any ε, δ in (0, 1/2), the learner outputs with probability ≥ 1 − δ a hypothesis with error ≤ ε, in time polynomial in 1/ε, 1/δ, n and size(c).

Sample complexity (finite H, consistent learner):
$m \geq (1/\varepsilon)(\ln|H| + \ln(1/\delta))$
Example: $|H| = 2^{10} = 1024$, $\varepsilon = 0.1$, $\delta = 0.05 \rightarrow m \geq 10\,(6.93 + 3.0) = 10 \times 9.93$, about **100 examples**.

Significance: it formalises generalisation, shows that more complex hypothesis spaces (larger $\ln|H|$) need more data, and links to VC dimension for infinite H.

Conclusion: PAC gives a theoretical bound on the data needed for reliable learning.

