Deep Learning

Short notes for B.Tech CSE / IT · MCA · M.Tech

Uncodemy Handwritten Notes · free for students

1. Introduction

- **Deep Learning (DL)** = a part of machine learning that uses neural networks with many hidden layers.
- It learns features automatically from raw data (images, audio, text), so less manual feature engineering is needed.
- Works well because of big data, GPUs and better training methods.

➤ ML vs DL

Point	Machine Learning	Deep Learning
Features	Mostly hand-made	Learned automatically
Data needed	Less	Large
Hardware	CPU is often enough	GPU / TPU preferred
Interpretability	Easier	Harder (black box)
Examples	SVM, decision tree	CNN, RNN, Transformer

2. Neural Network Basics

➤ Biological vs artificial neuron

- An artificial neuron takes inputs x, multiplies by weights w, adds bias b, then applies an activation f.

FORMULA Neuron output $y = f(w1x1 + w2x2 + \dots + wnxn + b) = f(w \cdot x + b)$.

➤ Perceptron

- Single neuron with a step function. Learning rule: $w = w + \eta(t - y)x$.
- Can solve AND, OR. Cannot solve XOR because XOR is not linearly separable. XOR needs a hidden layer.

EXAMPLE AND gate with w1 = w2 = 1, b = −1.5, step(z) = 1 if z >= 0. Input (1,1): z = 0.5 → 1. Input (1,0): z = −0.5 → 0. Works for all four inputs.

➤ Multi-layer perceptron (MLP)

```
input layer   hidden layer(s)   output layer
  x1 ---\      ( h1 )  ---\
  x2 ----+---> ( h2 )  ----+--->  ( y )
  x3 ---/      ( h3 )  ---/
```

- **Feedforward**: data moves input → hidden → output.
- **Universal approximation theorem**: one hidden layer with enough neurons can approximate any continuous function.

3. Activation Functions

Function	Formula	Range	Note
Sigmoid	1 / (1 + e^(−x))	0 to 1	Vanishing gradient, used for binary output
Tanh	(e^x − e^(−x)) / (e^x + e^(−x))	−1 to 1	Zero-centred
ReLU	max(0, x)	0 to ∞	Fast, default for hidden layers, "dying ReLU"
Leaky ReLU	x if x > 0 else 0.01x	−∞ to ∞	Fixes dying ReLU
Softmax	e^(xi) / Σ e^(xj)	0 to 1, sum = 1	Multi-class output

REMEMBER Without a non-linear activation, many layers collapse into one linear layer.

EXAMPLE Softmax of [1, 2]: e^1 = 2.718, e^2 = 7.389, sum = 10.107. Output = [0.269, 0.731].

4. Training a Network

➤ Loss functions

- Regression: **MSE** = (1/n) Σ $(y - \hat{y})^2$.

- Binary classification: **binary cross-entropy** = $-[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$.
- Multi-class: **categorical cross-entropy** = $-\Sigma\, y \log \hat{y}$.

➤ Backpropagation

1. Forward pass: compute the output and loss.
2. Backward pass: use the chain rule to find $\partial Loss/\partial w$ for every weight, from output back to input.
3. Update each weight with gradient descent.

FORMULA $w_{new} = w_{old} - \eta \cdot \partial L/\partial w$, where η = learning rate.

➤ Gradient descent variants

Type	Data per update	Note
Batch GD	Whole dataset	Smooth but slow
Stochastic GD (SGD)	1 sample	Fast, noisy
Mini-batch GD	Small batch (e.g. 32)	Most used

- **Epoch** = one full pass over the training data. **Iteration** = one update.

EXAMPLE 1000 samples, batch size 50 → 1000 / 50 = 20 iterations per epoch.

➤ Optimizers

- **Momentum**: adds a fraction of the previous update to move faster in one direction.
- **AdaGrad**: per-weight learning rate, shrinks over time.
- **RMSProp**: moving average of squared gradients.
- **Adam**: momentum + RMSProp. Most common default.

➤ Problems in training

- **Vanishing gradient**: gradients become tiny in deep nets (sigmoid / tanh). Fix: ReLU, good initialisation, batch norm, residual links.
- **Exploding gradient**: gradients become huge. Fix: gradient clipping.
- Weight initialisation: **Xavier / Glorot** (tanh, sigmoid), **He** (ReLU).

5. Regularisation

- **L1 / L2** weight penalty (L2 is also called weight decay).

- **Dropout**: randomly turn off a fraction of neurons during training (e.g. $p = 0.5$). Not used at test time.
- **Early stopping**: stop when validation loss starts rising.
- **Data augmentation**: flip, rotate, crop images to create more training data.
- **Batch normalisation**: normalises layer inputs per mini-batch; speeds up training.

TIP Training loss falling but validation loss rising = overfitting. Add dropout, augmentation or stop early.

6. Convolutional Neural Networks (CNN)

- Designed for grid data like images. Uses shared filters, so far fewer weights than a dense net.
- Layers: **Convolution → Activation (ReLU) → Pooling → … → Flatten → Fully connected → Softmax**.

➤ Convolution

- A small **filter / kernel** (e.g. 3×3) slides over the input and makes a **feature map**.
- **Stride** = step size. **Padding** = adding zeros around the border.

FORMULA Output size = $((N - F + 2P) / S) + 1$, where N = input size, F = filter size, P = padding, S = stride.

EXAMPLE Input 32×32, filter 5×5, $P = 0$, $S = 1$ → $(32 - 5 + 0)/1 + 1 = 28$, so 28×28. With $P = 2$ → $(32 - 5 + 4)/1 + 1 = 32$ (same size).

FORMULA Parameters in a conv layer = $(F \times F \times C_{in} + 1) \times K$, where K = number of filters (+1 for bias).

EXAMPLE 3×3 filters, 3 input channels, 64 filters → $(3 \times 3 \times 3 + 1) \times 64 = 28 \times 64 = 1792$ parameters.

➤ Pooling

- **Max pooling**: keeps the largest value in each window. **Average pooling**: keeps the mean.
- 2×2 pooling with stride 2 halves width and height. It has no learnable weights.

Famous architectures

Model	Key idea
LeNet-5	Early CNN for digits
AlexNet	ReLU + dropout + GPU, deep CNN
VGG-16	Many small 3×3 filters
GoogLeNet	Inception module (parallel filters)
ResNet	Skip (residual) connections, very deep nets

- **Transfer learning**: reuse a model trained on a big dataset, fine-tune it on your small dataset.

7. Recurrent Neural Networks (RNN)

- For sequence data (text, speech, time series). Has a loop: hidden state carries memory.

FORMULA $h_t = \tanh(W_h \cdot h_{(t-1)} + W_x \cdot x_t + b)$, $y_t = W_y \cdot h_t$.

- Trained with **Backpropagation Through Time (BPTT)**.
- Problem: vanishing gradients, so it forgets long-term information.

LSTM and GRU

- **LSTM** has a cell state and 3 gates:
  - **Forget gate**: what to remove from memory
  - **Input gate**: what new information to store
  - **Output gate**: what to output
- **GRU** has 2 gates (update, reset). Simpler and faster than LSTM.
- Sequence types: one-to-many (image captioning), many-to-one (sentiment), many-to-many (translation).

8. Advanced Models

Autoencoder

- Encoder compresses input to a small **latent code**; decoder rebuilds the input.
- Uses: dimensionality reduction, denoising, anomaly detection.

- **Variational autoencoder (VAE)**: learns a probability distribution, can generate new data.

➤ GAN (Generative Adversarial Network)

- **Generator** makes fake data. **Discriminator** tells real from fake.
- They train against each other (a min-max game) until fakes look real.
- Problem: mode collapse, unstable training.

➤ Attention and Transformers

- **Attention**: the model learns which parts of the input to focus on.

> **FORMULA** Attention(Q, K, V) = softmax(Q·K^T / sqrt(d_k)) · V.

- **Transformer**: built only on self-attention + feedforward layers, no recurrence, so it trains in parallel.
- Uses **positional encoding** to know word order. **Multi-head attention** looks at many relations at once.
- Encoder models (BERT) are good for understanding; decoder models (GPT) for generation.

9. Most Asked Questions (practice)

1. Difference between ML and DL. Why does DL need more data?
2. Explain the perceptron. Why can it not solve XOR?
3. Compare sigmoid, tanh, ReLU and softmax.
4. Explain backpropagation with the chain rule.
5. What is vanishing gradient? How is it solved?
6. Explain dropout, batch normalisation and early stopping.
7. Find the output size and parameter count of a conv layer.
8. Explain CNN architecture with a neat diagram.
9. Explain LSTM gates. LSTM vs GRU.
10. Explain GAN and the self-attention formula of transformers.

> **TIP** Numericals on conv output size, parameter count and softmax are easy marks. Learn the CNN and LSTM diagrams well.