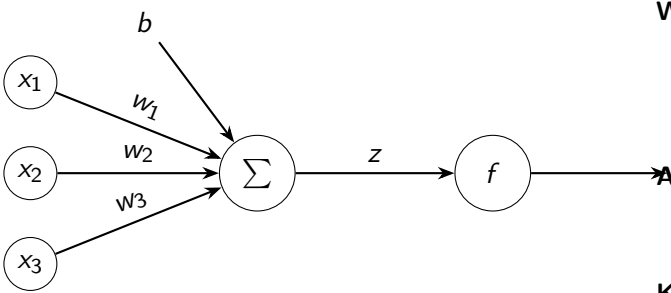
Deep Learning: Essential Notes

November 12, 2025

Slide 1: Nodes, Edges, and Sum/Activation



Activation: sum of the connected edges satisfies a threshold (know as activation fucntion), this activates the neuron at the next layer

Minimise cost value:

Back-progragation

Weighted sum:

$$z = \sum_{i=1}^{3} w_i x_i + b = \mathbf{w}^\top \mathbf{x} + b$$

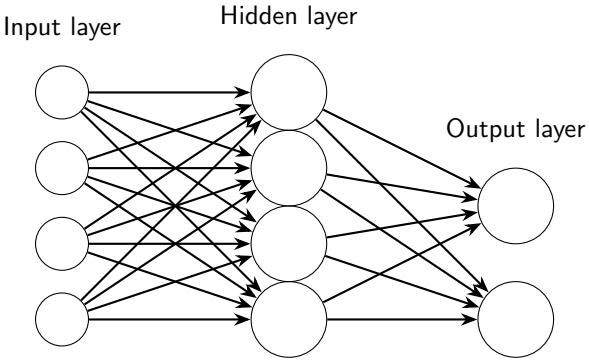
Activation:

$$y = f(z) \quad \text{(e.g., ReLU, } \sigma\text{, tanh)}$$

Key pieces:

- **Nodes**: inputs/neurons
- **Edges**: weights w_i
- **Bias**: b
- **Activation**: nonlinearity $f(\cdot)$

Slide 2: The Three General Layers



Input layer: raw features $\mathbf{x}$

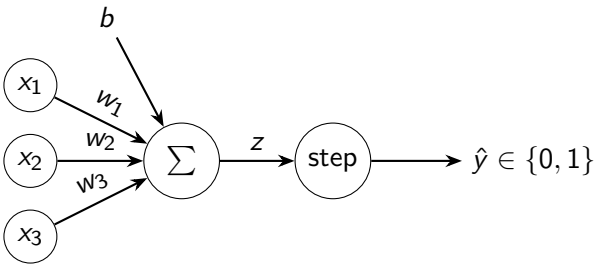
Hidden layers: learn internal representations via nonlinear units

Output layer: task-specific mapping (class scores, regression)

Forward pass:

$$\mathbf{h} = f(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1), \quad \mathbf{y} = g(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2)$$

Slide 3: Perceptron Visual and Prediction Steps



step 1: imputs are fed to into processor
step 2: peceptron aplies weiht to estiamte output
step 3: perceptron computes error
step 4: perceptron adjusts error by back propagation
step 5: repeat 1-4 until desired model accuracy

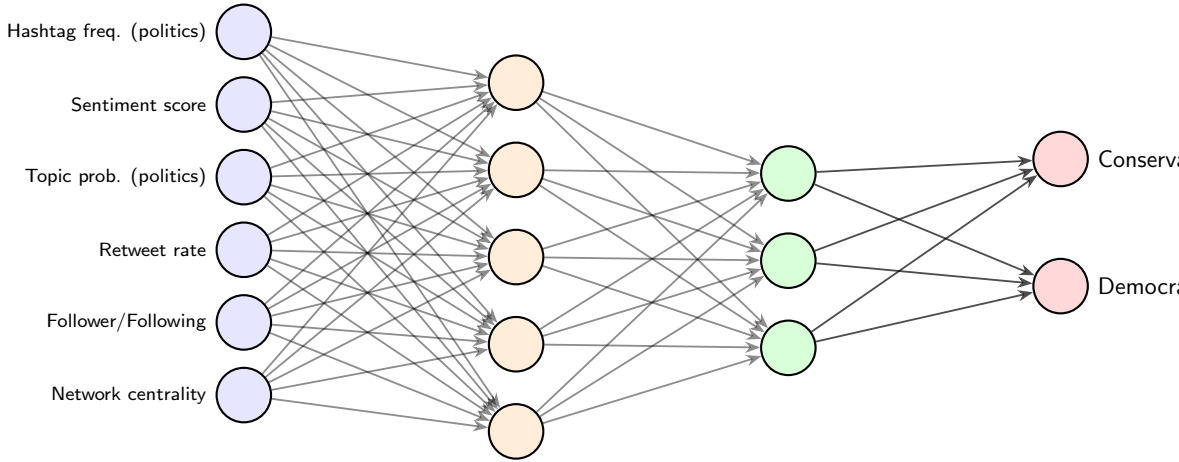
Prediction steps:

1. Take inputs $\mathbf{x}$.
2. Compute $z = \mathbf{w}^\top \mathbf{x} + b$.
3. Apply step: $\hat{y} = \mathbb{1}[z \geq 0]$.
4. Return class label (e.g. 1 = positive, 0 = negative).

Decision boundary:

$$\mathbf{w}^\top \mathbf{x} + b = 0$$

Slide 4: MLP Example Political Influence (Visual Only)

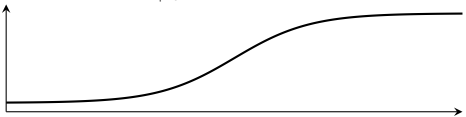


Slide 5: MLP Classification vs Regression (At a Glance)

Aspect	Classification MLP	Regression MLP
Target & Output	Discrete classes; output layer = *sigmoid* (binary) or *softmax* (multi-class)	Continuous value(s); output layer = *linear (identity)*
Loss	Binary/multi-class cross-entropy	MSE / MAE / Huber
Common Metrics	Accuracy, F1, Precision/Recall, AUC	RMSE, MAE, R^2
Typical Features (examples)	Counts/frequencies, TFIDF/topic probs, network stats, categorical dummies, normalized numeric signals	Scaled numerics, engineered ratios, moving averages, lagged values, interaction terms, splines/bins
Output Interpretation	Class probabilities (confidence)	Point estimate (with optional intervals)
Last-layer Activation	Sigmoid / Softmax	Identity

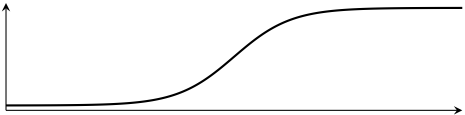
Slide 7: Activation Functions Shapes & When to Use

Sigmoid $\sigma(z) = \dfrac{1}{1+e^{-z}}$



Map to $(0, 1)$ for probabilities (binary outputs). Can saturate/vanish.

Tanh $\tanh(z)$

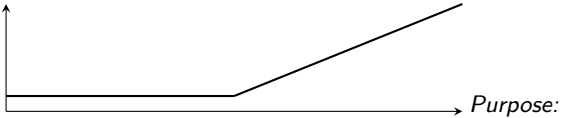


Purpose: Zero-centered activations; useful in shallow nets; still saturates.

Identity $f(z) = z$

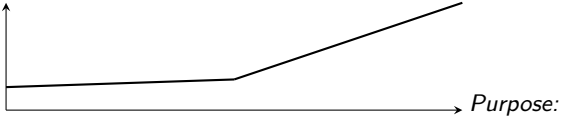
Linear output for regression (last layer).

ReLU $\max(0, z)$



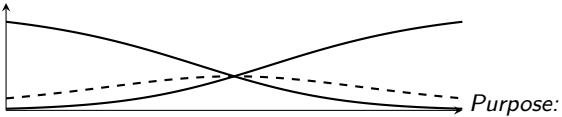
Default for hidden layers; sparse/fast; beware dead neurons.

Leaky ReLU $\max(\alpha z, z)$ (e.g. $\alpha = 0.01$)



Fix ReLU dead units with small negative slope.

Softmax $\text{softmax}_k(\mathbf{z}) = \dfrac{e^{z_k}}{\sum_j e^{z_j}}$



Turn logits into class probabilities (multi-class outputs).

Slide 6: MLP Model Components

Data pipeline

- Preprocess: cleaning, normalization/standardization
- Train/validation/test split; batching and shuffling

Architecture

- Layer sizes, depth, activation functions
- Regularization: dropout, weight decay, batch norm

Optimization

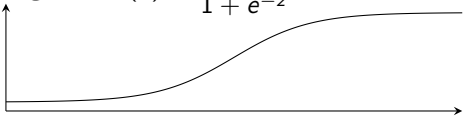
- Loss (BCE/MSE), optimizer (SGD/Adam), learning rate schedule
- Early stopping, checkpoints

Evaluation

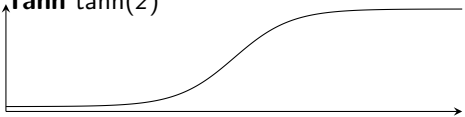
- Metrics (task-specific), error analysis, robustness checks

Slide 7: Typical Activation Functions (with sketches)

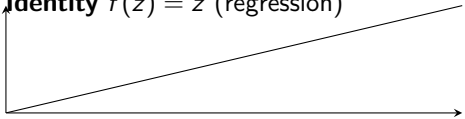
Sigmoid $\sigma(z) = \dfrac{1}{1 + e^{-z}}$



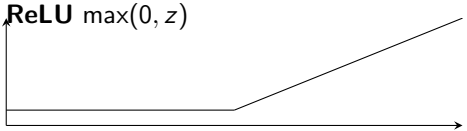
Tanh $\tanh(z)$



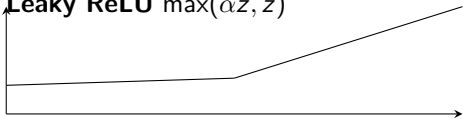
Identity $f(z) = z$ (regression)



ReLU $\max(0, z)$



Leaky ReLU $\max(\alpha z, z)$



Softmax (multi-class output)

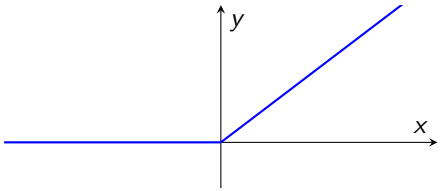
$$\text{softmax}_k(\mathbf{z}) = \frac{e^{z_k}}{\sum_j e^{z_j}}$$

Slide 8: Effect of Bias in a Neuron

Purpose: The bias allows a neuron to shift its activation threshold, enabling more flexible decision boundaries.

Without Bias: $y = f(wx)$

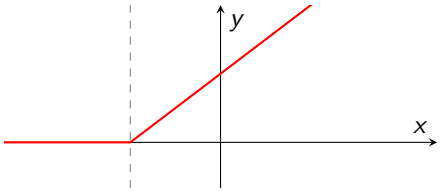
ReLU: $f(wx)$



Activates at $x = 0$ decision boundary fixed at origin.

With Bias: $y = f(wx + b)$, here $b = +1.5$

ReLU: $f(wx + b)$



Activation threshold shifts left; decision boundary moves.

Key Takeaway

- **Without bias:** all activations/lines pass through origin limited flexibility.
- **With bias:** neurons can learn thresholds and offsets more expressive model.

Agenda

1. Typical CNN workflow (data features classifier)
2. What is a convolution? Padding, stride, and why odd kernels
3. Max pooling: purpose and effect
4. Build a tiny CNN for MNIST
5. Your code: line-by-line concepts
6. Parameter counts, shapes, and sanity checks
7. (Optional) Feature maps visualization tips

Typical CNN Workflow

1. **Data prep:** load images, normalize to $[0, 1]$, set shape $(H, W, \text{channels})$.
2. **Convolutions:** learn local patterns (edges, strokes) with shared kernels.
3. **Pooling:** downsample, keep strongest activations, add translational tolerance.
4. **Flatten/Global pooling:** convert feature maps to vectors.
5. **Dense + Softmax:** map features to class probabilities.
6. **Train/Evaluate:** choose loss/metrics, iterate for few epochs.

MNIST specifics

- Grayscale digits: $28 \times 28 \times 1$
- 10 classes (0–9)
- Simple dataset: two conv blocks often suffice

What is a Convolution?

- A **kernel** (e.g., 3×3) slides over the image. Each output pixel is a weighted sum of a local neighborhood.
- For one channel: ((I * K)(i,j) $= \sum_{u=-r}^{r} \sum_{v=-r}^{r} K(u,v); I(i+u, j+v)), where = \frac{k-1}{2}$ for odd k.
- With C input channels, each filter has $k \times k \times C$ weights $+1$ bias; outputs one feature map. Using F filters yields F maps.

Why odd kernels (3,5,7...)?

They have a **well-defined center**. Padding and alignment stay symmetric; gradients are stable.

Padding and Stride

- **Padding** controls output size:
  - `valid`: no padding; $H \to H - k + 1$ (shrinks)
  - `same`: zero-pad to keep H and W unchanged
- **Stride** s skips positions (e.g., $s = 2$ halves spatial size).
- **Rule of thumb:** start with $k = (3, 3)$, stride $= 1$, padding=`same` or pooling for downsampling.

Max Pooling

- **MaxPooling2D**(2×2): takes the max in each 2×2 window; reduces size by ≈ 2 in H and W.
- Benefits: translation tolerance, fewer parameters, less overfitting, faster compute.
- Analogy (materials): like **coarse-graining**keeping the most prominent response in each small patch.

CNN Layer Pipeline (Simple Explanation)

Conv → BatchNorm → Pool → Conv → Pool → Flatten → Dense128 → Dropout → Dense10

- **Convolution (Conv)**: Learns patterns (edges, textures).
- **BatchNorm**: Normalizes activations for stable, faster training.
- **MaxPooling**: Downsamples spatial size (position invariance).
- **Second Conv + Pool**: Deeper, more abstract features.
- **Flatten**: Converts 2D maps to a 1D vector.
- **Dense(128)**: Combines features into meaning.
- **Dropout(0.3)**: Reduces overfitting.
- **Dense(10)**: Softmax classifier (10 classes).

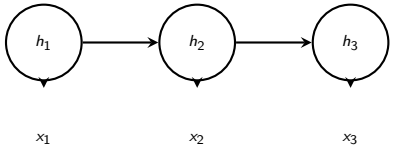
RNN Network: How It Differs from a CNN

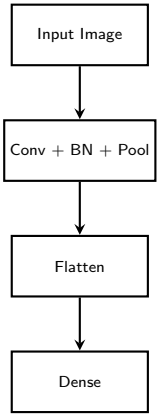
RNN (Recurrent Neural Network)

- Processes data **step-by-step** over time.
- Has **hidden state** that carries memory.
- Good for **time-series, sequences, signals**.
- Output depends on **current input + past inputs**.

CNN (Convolutional Neural Network)

- Processes data **spatially** (images).
- Learns patterns via **filters**.
- Good for **images, microstructures, 2D/3D fields**.
- Output depends on **spatial features**.

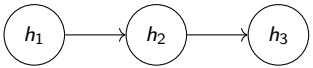




Comparison: SimpleRNN vs LSTM vs GRU

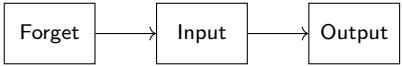
SimpleRNN

- Oldest RNN unit; single state h_t.
- Struggles on long sequences (vanishing gradients).
- Fast and simple.



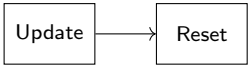
LSTM

- Two states: hidden h_t and cell c_t.
- Three gates: input, forget, output.
- Best for long-term dependencies; heavier.



GRU

- One state: h_t only.
- Two gates: reset, update.
- Faster than LSTM, similar performance.



Model	States	Gates	Best For
SimpleRNN	h_t	0	short patterns
GRU	h_t	2	medium sequences
LSTM	h_t, c_t	3	long-term patterns

ConvLSTM vs LSTM vs CNN (High-Level Comparison)

Model	Input Type	What It Learns	Use Case
CNN	Images (H, W, C)	Spatial features	Microstructures, images
LSTM	Sequences (T, F)	Temporal patterns	Time-series, signals
ConvLSTM	Image sequences (T, H, W, C)	Spatio-temporal features	Videos, microstructure evolution

CNN

- Processes whole image at once.
- Learns patterns in space.
- Uses Conv filters + Pooling.

LSTM

- Reads one time step at a time.
- Memory via hidden state h_t.
- No spatial structure.

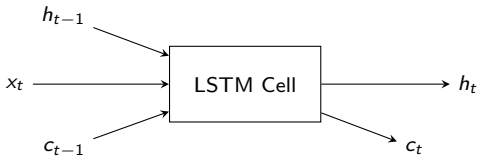
ConvLSTM

- CNN + LSTM ideas combined.
- Learns both space & time.
- Best for evolving microstructures/videos.

LSTM Memory: Hidden State vs Cell State

- **Hidden State (h_t)**: short-term memory; used for output.
- **Cell State (c_t)**: long-term memory; carried across timesteps.
- Together:

$$(h_t, c_t) = \text{LSTM}(x_t, h_{t-1}, c_{t-1})$$



RNN Input Shapes: 2D vs 3D

MLP / Dense Layers: 2D Input

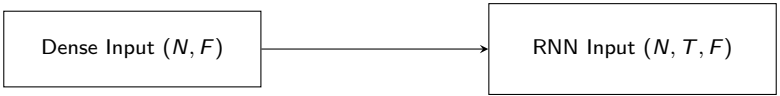
$$(N, F)$$

- N = number of samples
- F = number of features
- No time dimension

RNN / LSTM / GRU: 3D Input

$$(N, T, F)$$

- T = time steps
- F = features per timestep
- Needed for sequence modeling



Inside an LSTM Cell: Input, Forget, Output Gates

- **Forget Gate** f_t: remove old info.

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

- **Input Gate** i_t: write new info.

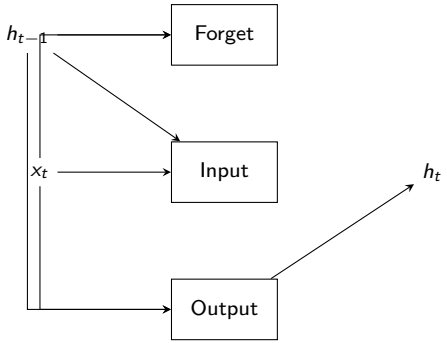
$$i_t = \sigma(W_i[x_t, h_{t-1}] + b_i)$$

- **Output Gate** o_t: expose memory as output.

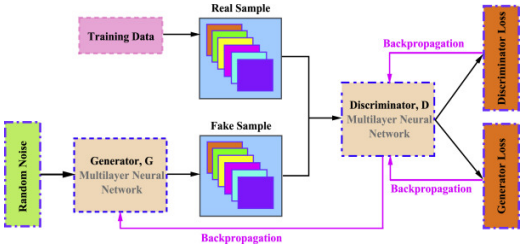
$$o_t = \sigma(W_o[x_t, h_{t-1}] + b_o)$$

- **Cell Update**

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t$$



Generative Adversarial Networks (GANs)



- The **Generator (G)** creates new data (e.g., images) from random noise.
- The **Discriminator (D)** evaluates the data, distinguishing between real samples from the training set and fake samples produced by the generator.
- Both networks are trained in an **adversarial** setup:
  - The generator tries to produce data that can fool the discriminator.
  - The discriminator tries to correctly classify real and fake data.
- Training continues until the generator produces data that is **indistinguishable** from real training samples.

Key Components of a GAN

- **Generator**
  - Takes a random noise vector (z)
  - Learns to produce synthetic samples (fake images)
  - Objective: *fool the discriminator*
- **Discriminator**
  - Takes real or generated samples
  - Outputs probability of being real
  - Objective: *correctly classify real vs fake*
- **Adversarial Framework**
  - Generator and Discriminator compete
  - Training improves both progressively

Generator and Discriminator Architecture

Generator Network

- Input: Noise vector (z), e.g., 100 dimensions
- Hidden Layers: Dense + LeakyReLU activations + Batch Normalization (stabilize training and improve convergence)
- Output Layer: 784 units (flattened 28×28 image)
- Activation: **tanh** (outputs in $[-1, 1]$), suitable for image data

Discriminator Network

- Input: Flattened image (784 dimensions)
- Hidden Layers: Dense + LeakyReLU
- Output Layer: 1 unit (real/fake)
- Activation: **sigmoid** probabiity of likelihood of input image being real

GAN Training Algorithm

1. **Generate fake images** using the generator by passing random noise as input.
2. **Sample real images** by selecting a random batch from the training dataset.
3. **Concatenate real and fake images** to form a combined batch for the discriminator.
4. **Create discriminator labels:** (Optional)
   - Real images $\rightarrow$ 0.9 (one-sided label smoothing)
   - Fake images $\rightarrow$ 0
5. **Train the discriminator** on the combined batch of real and fake images.
6. Generate a new batch of **random noise** for the generator.
7. **Create generator labels:** 1 (real), so the generator attempts to fool the discriminator.
8. **Train the combined GAN model** (generator + frozen discriminator) using the updated noise and the generator labels.
9. **Monitor training** by printing discriminator and GAN losses.
10. Repeat for the specified number of iterations. The generator improves at producing realistic images, while the discriminator gets better at distinguishing real from fake.
11. **Final Goal:** The generator should create images that are **indistinguishable** from real data.

latent_dim (100) $\longrightarrow$ Generator $\longrightarrow$ data_dim (784)
data_dim (784) $\longrightarrow$ Discriminator $\longrightarrow$ real/fake score