ASTR8150/PHYS8150
Neural Networks

Fabien Baron

Georgia State University

fbaron@gsu.edu

Fall 2025

What is a Neural Network?

- A neural network is a computational model inspired by the way biological neural networks in the brain process information.
- Neural network "models" consist of layers of interconnected nodes (neurons) that can learn complex patterns from data
- A well-designed network can approximate any continuous function on compact (finite) domains. Better approximations will require larger networks.
- Models are optimized by adjusting weights based on errors between predicted and actual values, using gradient descent variants adapted to large number of parameters.
- These models are the foundation of deep learning models
- Typical tasks: classification, regression, pattern recognition, regularization, generation of patterns.

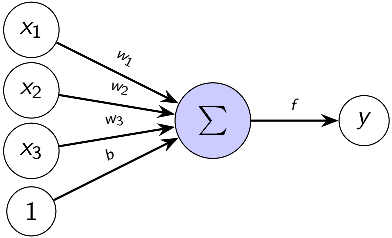
What is a neuron?

- A neuron, is a weighted sum of the inputs, followed by an activation function.
- A neuron first computes a linear transformation known as preactivation:

$$a = \sum_i w_i x_i + b$$

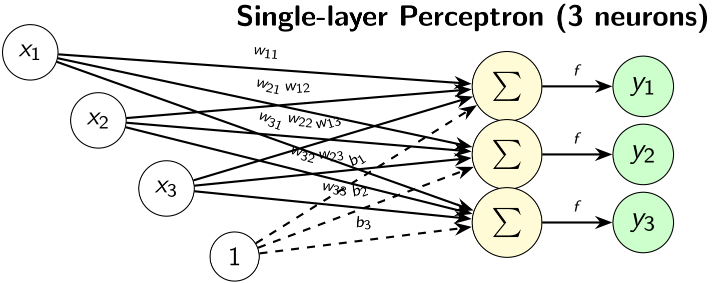
- Nonlinearity is introduced by using an activation function f:

$$y = f(a)$$



- The single-perceptron was the first neuron (developed for classification by Rosenblatt in 1958).

The Single-layer Perceptron



- Single layer of neurons that compute a weighted sum of inputs and apply an activation
- Suitable for classification tasks.

Weights and Biases

- **Weights:** Parameters that determine the strength of the connection between neurons.
- **Biases:** the bias $b = w_0$ allows shifting of activation thresholds and this added flexibility helps the model make better predictions.
- The output of a neuron is calculated as:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

where:

  - w_i are the weights
  - x_i are the inputs
  - b is the bias
  - f is the activation function

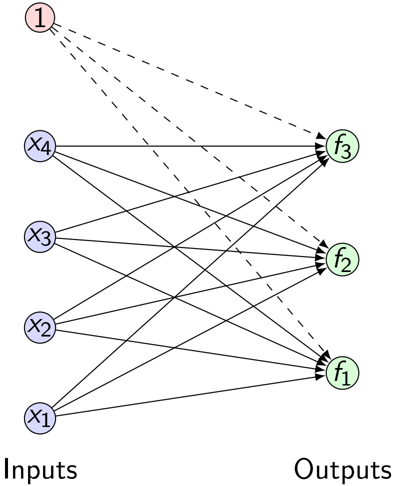
Activation Functions

- Activation functions introduce nonlinearity and enable networks to model complex mappings. Which ones to use depend on what the layer is designed to do (comes with experience).
- **Sigmoid:**
$$f(x) = \frac{1}{1 + e^{-x}}$$
Maps input values to a range between 0 and 1.
- **Tanh:**
$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$
Maps input values to a range between -1 and 1.
- **ReLU (Rectified Linear Unit):**
$$f(x) = \max(0, x)$$
Allows only positive values to pass through.
- **Softmax:**
$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Dense Layers

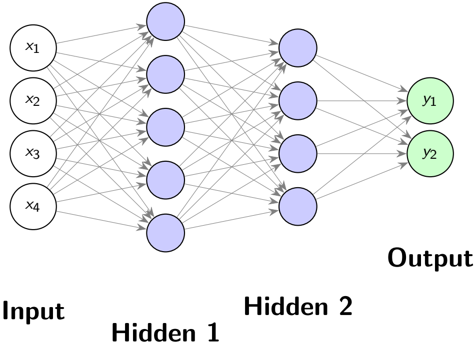
- A **dense layer** (fully connected layer) connects every neuron in the current layer to every neuron in the next layer.
- Each neuron in a layer is connected to neurons in adjacent layers through weighted connections.
- Each connection has an associated weight and each neuron has an associated bias.

Dense Layer



Neural network architecture: the Multilayer Perceptron

- A neural network consists of three types of layers
- **Input layer:** Receives the input data (contain features).
- **Hidden layers:** Intermediate layers where features are extracted but outputs are not directly observed.
- **Output layer:** Produces the final output (prediction).



- MLPs can model complex relationships and solve problems that are not linearly separable.
- MLPs are widely used in tasks like image recognition, speech processing, and time series prediction.

Training Neural Networks

- The goal of training a neural network is to minimize the error between the predicted and actual outputs.
- Training involves:
  - Feeding input data into the network.
  - Computing the output through forward propagation.
  - Comparing the output with the true label (using a loss function).
  - Adjusting weights to minimize the error using an optimization algorithm.
- Loss functions depend on the task
  - Regression: squared error.
  - Classification: cross-entropy.
- **Epochs:** The number of times the network is trained on the entire dataset.

Gradient Descent for neural networks

- Variants of gradient descent are used to minimize the loss function.
- The weight update rule for gradient descent is:

$$w_i = w_i - \eta \frac{\partial L}{\partial w_i}$$

  where:
  - η is the learning rate.
  - $\frac{\partial L}{\partial w_i}$ is the gradient of the loss function with respect to weight w_i.
- There are different variants of gradient descent:
  - **Stochastic Gradient Descent (SGD):** Uses a single data point for each update.
  - **Batch Gradient Descent:** Uses the entire dataset for each update.
  - **Mini-batch Gradient Descent:** Uses a small batch of data for each update.

Backpropagation

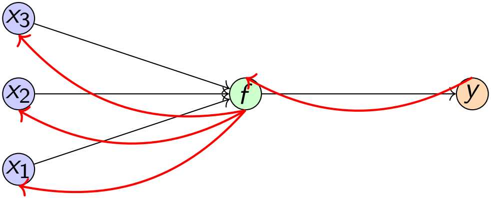
- How do we calculate the gradient of the network with respect to weights? Via backpropagation.
- This led to the development of automatic differentiation
- The algorithm involves two main steps:
  - **Forward Propagation:** Input data is passed through the network to get the output.
  - **Backward Propagation:** The error is calculated, and the gradients of the loss function with respect to the weights are computed.

Backpropagation: Intuition

- Goal: Compute $\frac{\partial L}{\partial w_{ij}}$ for all weights w_{ij} in the network.
- Uses the **chain rule** to propagate gradients from the output layer back to hidden layers.
- For a single hidden neuron f_j:

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial f_j}\frac{\partial f_j}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}$$

where $a_j = \sum_i w_{ji}x_i + b_j$ is the pre-activation.



Backpropagation: Algorithm

1. **Forward pass:** Compute all activations a_j and outputs f_j, y_k.
2. **Compute output layer gradient:**
$$\delta_k = \frac{\partial L}{\partial y_k} \odot g'(a_k)$$
where g is the **output layer activation function** (e.g., softmax for classification).
3. **Propagate to hidden layers:**
$$\delta_j = f'(a_j) \sum_k w_{kj} \delta_k$$
4. **Compute weight gradients:**
$$\frac{\partial L}{\partial w_{ji}} = \delta_j x_i$$
5. **Update weights:**
$$w_{ji} \leftarrow w_{ji} - \eta \frac{\partial L}{\partial w_{ji}}$$

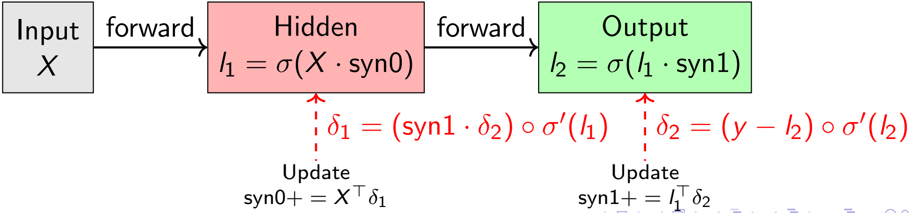
Backpropagation in Trask Toy Network (Sigmoid Activation)

- https://iamtrask.github.io/2015/07/12/basic-python-network/
- Input $X \to$ Hidden $l_1 = \sigma(X \cdot \text{syn0}) \to$ Output $l_2 = \sigma(l_1 \cdot \text{syn1})$
- Forward pass computes activations.
- Output error: $l_{2_{\text{error}}} = y - l_2$. Backprop computes:

$$\delta_2 = l_{2_{\text{error}}} \circ \sigma'(l_2), \quad \delta_1 = (\text{syn1} \cdot \delta_2) \circ \sigma'(l_1)$$

- Weight updates:

$$\text{syn1}+ = l_1^\top \delta_2, \quad \text{syn0}+ = X^\top \delta_1$$



From MLPs to CNNs

- MLPs (fully connected/dense layers):
  - Flatten image into a long vector; some of the spatial layout is lost.
  - Many parameters and this scales with input size.
- Convolutional Neural Networks (CNNs):
  - Local connectivity: neurons look at small spatial patches.
  - Weight sharing: same convolution filter applied across the image.
  - Preserve H$\times$W layout and are parameter-efficient.

Example: RGB image $32\times32\times3$

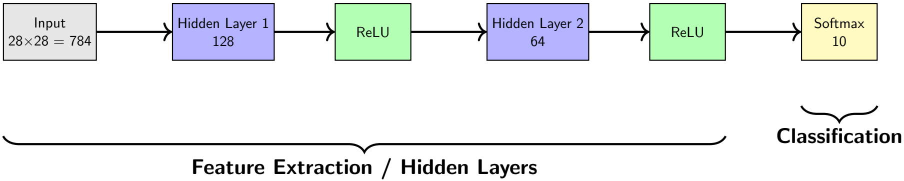
Dense layer (100 units):
parameters $= (32 \cdot 32 \cdot 3) \cdot 100 \approx 307{,}200$

Conv layer (3×3 kernels, 64 filters):
parameters $= 3 \cdot 3 \cdot 3 \cdot 64 = 1{,}728$

$\Rightarrow$ Conv layer uses $\sim$**180$\times$ fewer** params

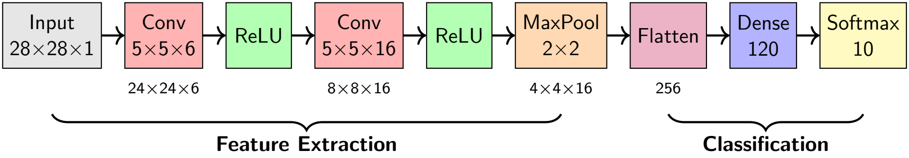
Basic MLP architecture (MNIST classification)

- Fully connected layers followed by activation functions
- Classification done by softmax
- Example below for MNIST classification



Basic CNN architecture (MNIST classification)

- Typical block: [Conv → ReLU → Conv → ReLU → Pool]
- Repeat several blocks to build hierarchy.
- Final: [Flatten → Dense layer → Softmax] for classification
- Design choices: kernel sizes, number of filters, strides, pooling policy.
- Example below for MNIST classification



Convolution and Feature Maps

- Convolution extracts local patterns (edges, textures) from input image patches.
- Just a discrete convolution (no flip, implemented as cross-correlation in most libraries)!
- Kernel slides over the image (stride) producing a smaller output (downsampling).
- Each kernel has depth = number of input channels; sums over channels to create a single feature map.
- Stacking multiple kernels produces multiple feature maps.
- Each kernel produces one feature map; multiple kernels produce multiple maps.

Convolution and Feature Maps

- Input: an image of size $H \times W \times C$, where $H =$ height, $W =$ width, $C =$ number of channels (e.g., 3 for RGB).
- A convolutional kernel/filter of size $k \times k$ has depth $= C$ (matches the input channels).
- At each spatial location, the kernel performs an element-wise multiplication with the corresponding $k \times k \times C$ patch of the input, then sums over all elements to produce a single scalar.
- This scalar becomes one element of the output feature map.
- The kernel slides across the input with a given **stride** S, producing a smaller output in height and width:

$$H_{\text{out}} = \frac{H - k + 2P}{S} + 1, \quad W_{\text{out}} = \frac{W - k + 2P}{S} + 1$$

  where P is the padding applied to the input.
- Each kernel produces one feature map of size $H_{\text{out}} \times W_{\text{out}}$.
- Multiple kernels (K of them) produce K feature maps, which can be stacked to form an output tensor of size $H_{\text{out}} \times W_{\text{out}} \times K$.

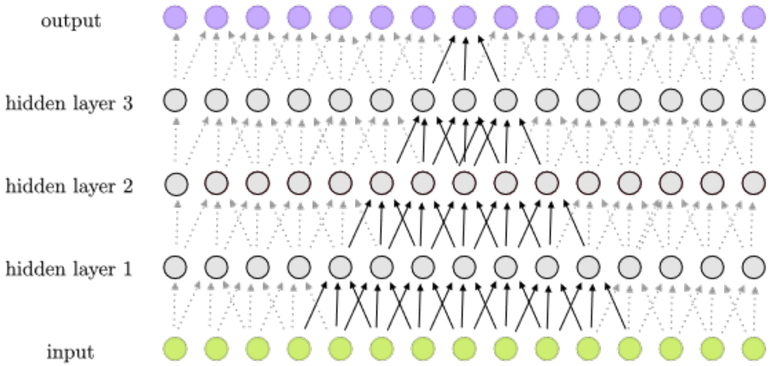
Convolution, then pooling

- The convolution unit is the first building block of a CNN. It requires some kernel specifications:
  - Padding P: add border (often zeros) to control edge behavior.
  - Stride S: how far the kernel moves each step.
  - Output size:

$$H_{\text{out}} = \left\lfloor \frac{H - k + 2P}{S} \right\rfloor + 1.$$

- Pooling reduces spatial dimensions and keeps important signals.
  - Max pooling (e.g. 2×2, stride 2) picks the strongest activation in each window.
  - Average pooling keep the average of the window.
  - Reduces the number of variables: faster training.

1D CNN Example: Feed-forward Illustration



- This diagram shows a 3-layer "1D" convolutional network with kernel size = 3 and stride = 1.
- Convolution in each layer: window slides over the input, producing a feature map. After convolution, the activation function is applied (not shown).

Receptive Field in CNNs

- The receptive field of a neuron is the region of the previous layer it "sees."
- In dense (fully connected) layers, each neuron sees the entire previous layer.
- In a convolutional layer, each neuron sees only a local patch (e.g., 5×5).
- Stacking convolutional layers increases the effective receptive field:
  - Neurons consider larger areas of the input progressively.
- "Dilated convolutions" expand receptive field without increasing parameters:
  - Sparse sampling of input
  - Can combine multiple dilation rates for variable receptive field sizes

Flattening in CNNs

- Flattening is typically used right before a dense classification layer in CNNs.
- Fully connected/dense layers expect a 1D vector input.
- Convolution and pooling layers output the feature maps with shape $H \times W \times C$, $H =$ height, $W =$ width, $C =$ number of channels
- Flattening therefore reshapes the 3D feature maps into a 1D vector:

$$\text{Flatten} : H \times W \times C \quad \rightarrow \quad H \cdot W \cdot C$$

Overview of all Neural Network Architectures

- The following 20 slides were partially generated by Claude + ChatGPT from my instructions, then I edited them for clarity
- Architectures organized chronologically by development
- Importance rating:

  *** = foundational/very important
  ** = widely used
  * = specialized/emerging
- Building on CNN knowledge to explore modern architectures
- Focus on practical applications in physical sciences
- Understanding when to use (and when not to use) each architecture

Recurrent Neural Networks - 1986 - **

- **Purpose:** Process sequential data with temporal dependencies
- **Architecture:** Hidden state loops back into itself at each time step
- **Key difference from CNNs:** Maintains memory of previous inputs
- **Applications:**
  - Light curve analysis (variable stars, transients)
  - Time-series photometry from surveys
  - Spectral sequence analysis
- **Limitation:** Struggles with long sequences (vanishing gradients)
- **Modern alternatives:** LSTM, GRU, Transformers often preferred

Autoencoders - 1980s - **

- **Architecture:** Encoder compresses to bottleneck, decoder reconstructs
- **Training:** Minimize reconstruction error (unsupervised)
- **Purpose:** Learn compressed representations, dimensionality reduction
- **Applications:**
  - Compressing spectral data for efficient storage and analysis
  - Denoising telescope images
  - Feature extraction for clustering and classification
- **Variants:**
  - Denoising autoencoders: trained with corrupted inputs
  - Sparse autoencoders: enforce sparsity in latent space
  - Convolutional autoencoders: for image data
- **Foundation for:** VAEs and other generative models

LSTM and GRU Networks - 1997/2014 - **

- **LSTM (Long Short-Term Memory) - 1997:**
  - Cell state (persistent memory) with three gates (learned filters controlling information flow)
  - Gates decide: what to forget, what to store, what to output
  - Maintains context over thousands of time steps
  - Applications: long-duration transient classification, multi-wavelength correlation
- **GRU (Gated Recurrent Unit) - 2014:**
  - Simpler than LSTM with only two gates (update and reset)
  - Faster training, similar performance for many tasks
  - Often preferred when computational efficiency matters
- **When to use:** Sequential astronomical data where order matters
- **Trend:** Being replaced by Transformers for many applications

Encoder-Decoder Architectures - 2014 - **

- **Purpose:** Map input sequences to output sequences of different lengths
- **Architecture:** Encoder compresses input to latent representation, decoder generates output
- **Implementation:** Can use RNNs, LSTMs, or Transformers for both components
- **With attention:** Decoder can focus on relevant encoder states dynamically
- **Applications:**
  - Spectrum-to-parameter estimation
  - Image captioning for astronomical objects
  - Translating between observational domains
- **Key advantage:** Flexible input-output lengths and modalities

Generative Adversarial Networks (GANs) - 2014 - ***

- **Architecture:** Generator creates samples from noise, discriminator judges authenticity
- **Training:** Adversarial min-max game between two networks
- **Variants:** DCGAN (convolutional), StyleGAN (style control), conditional GAN
- **Applications:**
  - High-quality synthetic galaxy images for training data augmentation
  - Simulating realistic telescope PSFs and observing conditions
  - Generating rare object classes to balance datasets
- **Challenges:** Training instability, mode collapse, difficult hyperparameter tuning
- **Advantage:** Sharp, realistic samples (better than VAEs)
- **Status:** Increasingly replaced by diffusion models for image generation

Variational Autoencoders (VAEs) - 2014 - **

- **Extension of autoencoders:** Encoder outputs distribution parameters, not fixed vectors
- **Architecture:** Encoder maps to mean and variance, sampling layer, decoder reconstructs
- **Loss:** Reconstruction + KL divergence (regularizes latent space)
- **Key advantage:** Smooth, continuous latent space enables generation
- **Applications:**
  - Unsupervised morphology classification
  - Anomaly detection in survey data
  - Generating physically plausible galaxy simulations
  - Interpolating between observed object types
- **Vs GANs:** More stable training, structured latent space, but blurrier samples

Attention Mechanisms - 2014 - ***

- **Core idea:** Allow models to focus on relevant parts of input dynamically
- **Mechanism:** Compute weighted sum of values based on query-key similarity (matching what you're looking for with what's available)
- **Advantages over RNNs:**
  - Can attend to any position directly (no sequential bottleneck: all positions accessible at once)
  - Parallelizable training (much faster)
  - Better at capturing long-range dependencies
- **Applications:**
  - Focusing on spectral features in high-resolution spectra
  - Identifying relevant time steps in irregular light curves
  - Multi-modal data fusion (images + spectra + metadata)
- **Foundation for:** Transformers, which use self-attention

UNet Architecture - 2015 - ***

- **Purpose:** Dense prediction tasks requiring spatial precision
- **Architecture:** Symmetric encoder-decoder with skip connections at each level
- **Key feature:** Skip connections preserve fine spatial details lost during downsampling
- **Why it works:** Combines high-level semantic info (what objects are) with low-level spatial detail (where exactly they are)
- **Applications:**
  - Source detection and deblending in crowded fields
  - Pixel-level morphological segmentation of galaxies
  - Artifact and satellite trail masking
  - Point spread function deconvolution
- **Variants:** 3D UNet (for volumetric data), attention UNet, residual UNet
- **Status:** Gold standard for segmentation tasks

Residual Networks (ResNets) - 2015 - ***

- **Problem solved:** Degradation problem (very deep networks performed worse than shallow ones)
- **Innovation:** Skip connections that add input to output: $y = F(x) + x$
- **Key benefit:** Enables training of networks with 100+ layers
- **Why it works:** Easier to learn residual mapping (small corrections to input) than full mapping
- **Applications:**
  - Deep feature extraction from images
  - Backbone for object detection and segmentation
  - Base architecture for many specialized models
- **Variants:** ResNeXt, Wide ResNet, DenseNet (dense connections)
- **Legacy:** Skip connections now ubiquitous (UNet, Transformers, etc.)

Normalizing Flows & Invertible Networks - 2015 - **

- **Core property:** Bijective (invertible) mappings between data and latent space
- **Architecture:** Coupling layers, affine transformations, carefully designed invertible operations
- **Key advantage:** Can compute exact likelihoods (unlike VAEs/GANs)
- **Applications:**
  - Uncertainty quantification in parameter inference
  - Bayesian posterior estimation with guaranteed coverage
  - Inverse problems: deconvolution, denoising with uncertainty
  - Simulator-based inference for complex physical models
- **Examples:** RealNVP, Glow, Neural Spline Flows
- **Trade-off:** Architectural constraints for invertibility vs flexibility

Bayesian Neural Networks (BNNs) - 1990s/2010s - **

- **Core idea:** Treat network weights as probability distributions instead of fixed values
- **Key difference:** Provides uncertainty estimates for predictions (epistemic uncertainty: model uncertainty)
- **Training:** Variational inference, sampling methods (MCMC), or ensemble approximations
- **Output:** Predictive distribution rather than single point estimate
- **Applications:**
  - Critical decisions requiring confidence estimates (safety-critical systems)
  - Active learning: identify which data points to measure next
  - Out-of-distribution detection (knowing when model is uncertain)
  - Experimental design in physics (optimal measurement planning)
- **Challenge:** Computationally expensive, approximate inference needed
- **Modern approaches:** Dropout as Bayesian approximation, deep ensembles, Laplace approximation

Transformer Networks - 2017 - ***

- **Architecture:** Self-attention layers (attention within same sequence) + feed-forward networks (no recurrence)
- **Key innovation:** Process entire sequences in parallel using attention
- **Components:** Multi-head attention (multiple parallel attention operations), positional encoding (adds position info), layer normalization
- **Variants:**
  - Encoder-only (BERT-style): classification, feature extraction
  - Decoder-only (GPT-style): generation, autoregressive tasks
  - Encoder-decoder: sequence-to-sequence tasks
- **Applications:**
  - Photometric redshift estimation from multi-band data
  - Time-series classification of variable objects
  - Foundation models for astronomical catalogs
- **Status:** Currently dominant architecture for sequence modeling

Graph Neural Networks (GNNs) - 2017 - **

- **Purpose:** Process data with graph structure (nodes and edges)
- **Key idea:** Nodes aggregate information from neighbors iteratively (message passing)
- **Difference from CNNs:** Handle irregular, non-Euclidean structures (not on regular grids)
- **Applications:**
  - Modeling cosmic web structure (galaxies as nodes)
  - Processing point clouds from fiber spectroscopy
  - Predicting properties in galaxy merger trees
  - Analyzing gravitational lens systems (multiple images as graph)
- **Variants:** GCN (Graph Convolutional), GraphSAGE, GAT (Graph Attention)
- **Advantage:** Natural representation for many astronomical datasets

Neural ODEs & Continuous Models - 2018 - *

- **Concept:** Treat neural network layers as continuous transformations (differential equations)
- **Key idea:** Replace discrete layers with continuous-time dynamics (infinitely many infinitesimal steps)
- **Benefits:** Memory-efficient training, adaptive computation (adjust precision as needed), theoretical elegance
- **Applications:**
    - Modeling time evolution of physical systems
    - Trajectory prediction for moving objects (asteroids, satellites)
    - Continuous-time series analysis with irregular sampling
- **Related:** Physics-informed neural networks (PINNs)
- **Status:** Active research area, growing adoption for physics problems

Physics-Informed Neural Networks (PINNs) - 2019 - **

- **Innovation:** Incorporate physical laws directly into loss function
- **Loss components:** Data loss + physics loss (how well equations are satisfied)
- **Advantage:** Can learn with sparse data by leveraging known physics
- **Applications:**
  - Solving radiative transfer equations
  - Stellar structure and evolution modeling
  - Orbital dynamics with observational constraints
  - Inverse problems in astrophysics (inferring hidden parameters)
- **Key benefit:** Solutions respect physical laws by construction
- **Challenge:** Requires differentiable physics equations (must be smooth for gradient computation)
- **Trend:** Growing interest for scientific machine learning

Neural Radiance Fields (NeRFs) - 2020 - **

- **Purpose:** 3D scene reconstruction from 2D images
- **Representation:** Neural network encodes volumetric scene (3D volume) as continuous function
- **Input:** Spatial coordinates (x,y,z) and viewing direction
- **Output:** Density and color at that point (raytracing through volume)
- **Applications:**
  - 3D reconstruction of extended objects (nebulae, galaxies)
  - Modeling complex morphologies from multi-angle observations
  - Virtual observatory: generate novel views of objects
- **Challenge:** Requires multiple viewpoints (limited for distant objects)
- **Variants:** Instant-NGP (faster training), conditional NeRFs
- **Status:** Emerging application area for physical sciences

Vision Transformers (ViT) - 2020 - ***

- **Extension:** Apply Transformers to images by treating patches as tokens (small image squares as sequence elements)
- **Process:** Split image into patches, embed them, apply Transformer
- **Advantages:**
  - Can capture global context better than CNNs
  - Scales well with data and compute
  - Effective for large datasets
- **Disadvantages:**
  - Requires more data than CNNs to train from scratch
  - Less inductive bias (doesn't assume spatial locality: nearby pixels are related)
- **Applications:**
  - Galaxy morphology classification at scale
  - Multi-scale feature extraction from survey images
  - Transfer learning from pre-trained models

Diffusion Models - 2020 - ***

- **Process:** Learn to reverse a gradual noising process (progressively add random noise)
- **Training:** Network predicts noise at different corruption levels
- **Generation:** Start with pure noise, iteratively denoise to create sample
- **Advantages over GANs:** More stable training, higher sample quality, better mode coverage (generates diverse outputs)
- **Applications:**
  - State-of-the-art galaxy morphology generation
  - Super-resolution of low-resolution astronomical images
  - Inpainting missing or corrupted telescope data
  - Conditional generation based on physical parameters
- **Drawback:** Slower generation (many denoising steps)
- **Trend:** Current state-of-the-art for image generation tasks

Foundation Models & Transfer Learning - 2020s - ***

- **Concept:** Large models pre-trained on massive datasets, then adapted (fine-tuned for specific tasks)
- **Paradigm shift:** From training from scratch to fine-tuning (adjust pre-trained weights)
- **Benefits:**
  - Leverage knowledge from large datasets
  - Requires less labeled data for specific tasks
  - Often achieves better performance
- **Applications:**
  - Pre-train on large sky surveys, fine-tune for specific objects
  - Transfer from simulations to real observations
  - Multi-modal models combining images, spectra, and text
- **Examples:** Vision-language models for scientific data, cross-domain foundation models
- **Trend:** Increasingly important as datasets grow

Siamese & Contrastive Learning Networks - 2005/2020 - **

- **Architecture:** Twin networks with shared weights (same parameters) processing paired inputs
- **Training:** Learn similarity metric (measure of how alike items are)
- **Mechanism:** Pull similar pairs together in embedding space, push different pairs apart
- **Loss functions:** Contrastive loss (pair-based), triplet loss (anchor-positive-negative), SimCLR (modern contrastive)
- **Applications:**
  - Finding similar spectra or light curves in large databases
  - Few-shot learning (learn from few examples) for rare object classification
  - Cross-matching objects across surveys
  - Change detection between epochs
- **Advantage:** Effective with limited labeled data

Hybrid & Ensemble Architectures - Ongoing - **

- **Hybrid models:** Combine different architecture types
  - CNN backbone (feature extractor) + Transformer head (final processing): ConvViT, CoAtNet
  - CNN for spatial features + RNN for temporal (for video/time-series)
  - Physics-informed layers within standard networks
- **Ensemble methods:** Combine multiple models for robustness
  - Multiple architectures voting on classification
  - Stacking (using outputs of models as inputs to meta-model) for regression
  - Uncertainty estimation through ensemble disagreement (variation across models)
- **Benefit:** Leverage strengths of different approaches for complex problems
- **Trade-off:** Increased complexity and computational cost

Architecture Comparison: Practical Considerations

- **Data requirements:**
  - Low: Transfer learning, Siamese networks, PINNs
  - Medium: CNNs, RNNs, autoencoders, UNet
  - High: Transformers, GANs, diffusion models (from scratch)
- **Computational cost:**
  - Low: Simple RNNs, small CNNs
  - Medium: ResNets, LSTMs, autoencoders, GNNs
  - High: Large Transformers, diffusion models, NeRFs
- **Interpretability:**
  - Higher: Attention mechanisms, PINNs, GNNs (graph structure)
  - Lower: Deep CNNs, complex ensembles, large foundation models

When NOT to Use Deep Learning

- **Limited data:** With <100s of examples, classical methods often better
- **Interpretability required:** When you need to understand every decision
- **Computational constraints:** Real-time processing on limited hardware
- **Well-solved problems:** Established methods may be simpler and sufficient
- **Physical models exist:** When accurate physics-based models are available
- **Consider alternatives:**
  - Traditional statistics for small sample sizes
  - Physics-based models for well-understood processes
  - Simple ML (random forests, gradient boosting) for tabular data
  - Gaussian processes for uncertainty quantification with small data
- **Best practice:** Start simple, add complexity only when justified

Architecture Selection Guide

- **Images (single):** CNN, ResNet, ViT
- **Sequences (time-series, spectra):** LSTM, GRU, Transformers
- **Image segmentation:** UNet, attention UNet
- **Generation:** Diffusion models (best quality), GANs, VAEs
- **Graphs & irregular data:** GNNs
- **With physical constraints:** PINNs, Neural ODEs
- **Limited labels:** Transfer learning, Siamese networks
- **Uncertainty critical:** Normalizing flows, Bayesian approaches, ensembles
- **Multi-modal data:** Transformers with appropriate encoders
- **Key factors:** Data type, quantity, task, computational budget, interpretability needs

Current Trends in Neural Architectures (2024-2025)

- **Transformers everywhere:** Extending beyond NLP (Natural Language Processing) to all domains
- **Foundation models:** Pre-training on massive datasets becoming standard
- **Efficient architectures:** MobileNets, EfficientNets for resource constraints (mobile devices, edge computing)
- **Neural architecture search:** Automated design of architectures (AI designing AI)
- **Physics integration:** Growing use of PINNs and physics-aware models
- **Multimodal learning:** Models handling multiple data types simultaneously
- **Uncertainty quantification:** More focus on reliable confidence estimates (knowing when model is uncertain)
- **Domain-specific architectures:** Custom designs for experimental physics, materials science, climate
- **Keep learning:** Field evolves rapidly, fundamentals remain important

Learning Resources & Next Steps

- **Getting started:**
  - Identify your problem type and data characteristics
  - Start with established architectures for your domain
  - Use transfer learning when possible
  - Benchmark against simple baselines first
- **Domain-specific resources:**
  - Physics/science-focused ML workshops and tutorials
  - Pre-trained models on scientific data
  - Community benchmarks and datasets
- **Best practices:**
  - Validate carefully with held-out test sets
  - Consider domain-specific evaluation metrics
  - Document architecture choices and hyperparameters
  - Share code and models with the community

Visual Demonstrations: RNNs, LSTMs, Transformers

- **RNN/LSTM Visualizations:**
  - Understanding LSTMs (Colah's Blog) - Excellent visual explanations
  - Memorization in RNNs (Distill) - Interactive visualizations
- **Transformer Visualizations:**
  - Hugging Face Transformer Example - Interactive attention
  - LLM Visualization - 3D visualization of transformer internals
  - Transformer Explainer - Step-by-step interactive guide
- **Attention Mechanism:**
  - Visualizing Attention (Jay Alammar) - Clear visual guide

Visual Demonstrations: Generative Models

- **GANs:**
  - This Person Does Not Exist - StyleGAN face generation
  - GANDissect - Interactive GAN unit visualization
  - GAN Lab - Train GANs in your browser
- **Diffusion Models:**
  - Stable Diffusion Demo - Text-to-image generation
  - Diffusion Explainer - Interactive explanation
- **VAEs:**
  - VAE Explainer - Interactive latent space exploration

Visual Demonstrations: Specialized Architectures

- **Neural Radiance Fields (NeRFs):**
  - NeRF Project Page - Original paper with demos
  - Nerfstudio - Open-source NeRF tools with gallery
- **Graph Neural Networks:**
  - GNN Introduction (Distill) - Interactive visualizations
  - StellarGraph Demos - GNN notebooks
- **Vision Transformers:**
  - ViT Attention Maps - Visualizing what ViTs learn
- **Bayesian Neural Networks:**
  - Bayesian Networks - Bayesian neural nets model uncertainty in predictions.

Visual Demonstrations: CNNs & Architecture Tools

- **CNN Visualizations (for reference):**
  - CNN Explainer - Interactive CNN visualization
  - ConvNetJS - Train CNNs in browser
- **UNet & Segmentation:**
  - UNet Original Page - Medical image examples
- **Neural Network Playgrounds:**
  - TensorFlow Playground - Interactive neural network training
  - Netron and related models - Visualize network architectures from model files

Interactive Learning Platforms & Tutorials

- **Comprehensive Tutorials:**
  - Distill.pub - Research journal with interactive visualizations
  - Dive into Deep Learning - Interactive book with code
  - Jay Alammar's Blog - Visual guides to ML concepts
- **Hands-on Platforms:**
  - Hugging Face Spaces - Pre-trained model demos
- **Papers with Code:**
  - Papers with Code - Research papers with implementations
  - Annotated Implementations - Explained code

Summary: Timeline & Importance

- **1980s-2000s Foundations:** RNN (**), Autoencoders (**), LSTM (**), BNNs (**)
- **2014-2017 Breakthrough:** Attention (***), GANs (***), VAEs (**), ResNets (***), UNet (***), Normalizing Flows (**)
- **2017-2020 Transformer Era:** Transformers (***), GNNs (**), Neural ODEs (*), PINNs (**)
- **2020-Present Modern:** ViT (***), Diffusion (***), NeRFs (**), Foundation Models (***)
- **Ongoing Techniques:** Siamese/Contrastive (**), Hybrid/Ensemble (**)
- *** = **Foundational:** Transformers, Attention, GANs, ResNets, UNet, Diffusion, ViT, Foundation Models
- ** = **Widely Used:** RNNs, LSTMs, Autoencoders, VAEs, GNNs, PINNs, NeRFs, Siamese, Hybrid
- * = **Specialized:** Neural ODEs (emerging in physics applications)