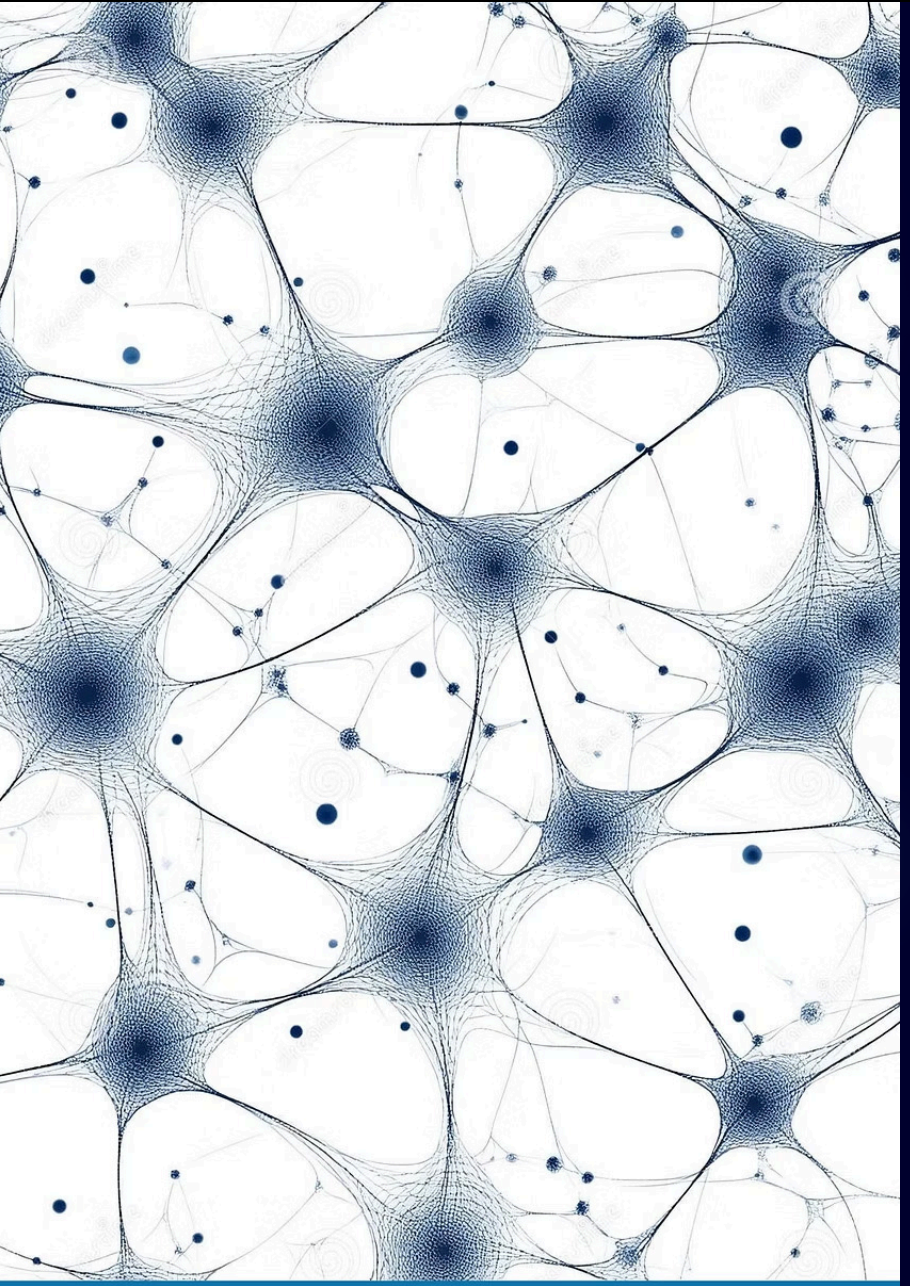


Deep Neural Architectures: CNNs, Autoencoders, RNNs & Transformers

Course: Machine Learning and Data Analysis

Instructor: Hernán Andrés Morales-Navarrete

Beyond Fully Connected Networks

Classical deep networks treat all inputs uniformly, ignoring the inherent spatial and temporal structures present in real-world data. This fundamental limitation has driven the development of specialized architectures that can capture **patterns, hierarchies, and dependencies** in diverse data modalities.

Spatial Invariance

Images contain local patterns that repeat across locations. CNNs address this through shared convolutional filters that detect features regardless of position.

Dimensionality Reduction

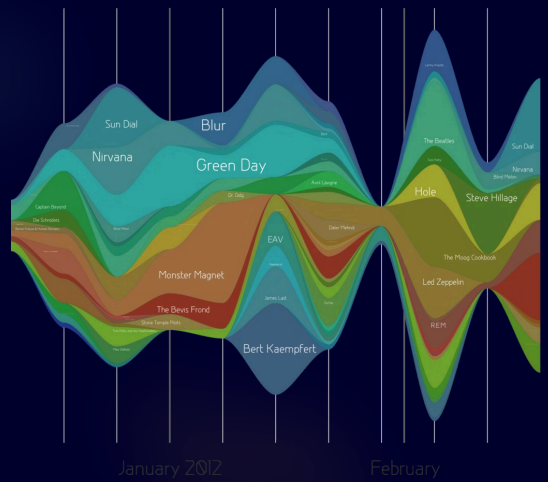
High-dimensional data often contains redundancy. Autoencoders learn compact representations that preserve essential information while reducing complexity.

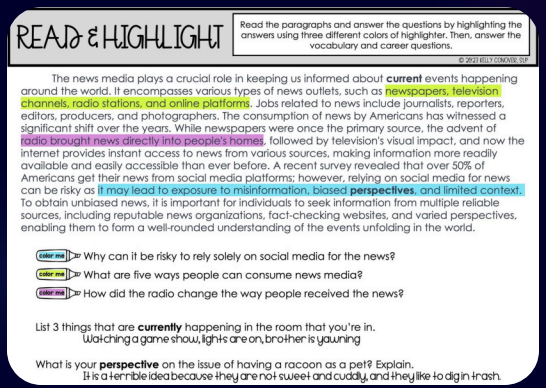
Sequential Dependencies

Time-series and sequential data require memory of past inputs. RNNs maintain hidden states to capture temporal relationships.

Long-Range Context

Many tasks require understanding relationships across distant elements. Transformers use attention mechanisms to model global dependencies efficiently.





READ & HIGHLIGHT

Read the paragraphs and answer the questions by highlighting the answers using three different colors of highlighter. Then, answer the vocabulary and career questions.

© 2021 KELLY CONOVER TEP

The news media plays a crucial role in keeping us informed about **current** events happening around the world. It encompasses various types of news outlets, such as newspapers, television channels, radio stations, and online platforms. Jobs related to news include journalists, reporters, editors, producers, and photographers. The consumption of news by Americans has witnessed a significant shift over the years. While newspapers were once the primary source, the advent of radio brought news directly into people's homes, followed by television's visual impact, and now the internet provides instant access to news from various sources, making information more readily available and easily accessible than ever before. A recent survey revealed that over 50% of Americans get their news from social media platforms; however, relying on social media for news can be risky as it may lead to exposure to misinformation, biased **perspectives**, and limited context. To obtain unbiased news, it is important for individuals to seek information from multiple reliable sources, including reputable news organizations, fact-checking websites, and varied perspectives, enabling them to form a well-rounded understanding of the events unfolding in the world.

color one Why can it be risky to rely solely on social media for the news?

color one What are five ways people can consume news media?

color one How did the radio change the way people received the news?

List 3 things that are **currently** happening in the room that you're in.
Watching a game show, lights are on, brother is yawning

What is your **perspective** on the issue of having a racoon as a pet? Explain.
It is a terrible idea because they are not sweet and cuddly, and they like to dig in trash.

Comparing Modern Neural Architectures

Each architecture is designed to exploit specific structural properties of data. Understanding their core principles helps us choose the right tool for each problem.

Architecture	Data Type	Core Idea	Example Applications
CNN	Images, spatial data	Shared convolutional filters detect local patterns	Microscopy analysis, particle tracking, medical imaging
Autoencoder	Any modality	Compression and reconstruction through latent space	Denoising, anomaly detection, feature learning
RNN	Sequential data	Hidden state maintains temporal memory	Time-series forecasting, sensor data, video analysis
Transformer	Sequential data	Attention mechanism models global relationships	Natural language, physics signals, molecular dynamics

While each architecture has a primary use case, modern applications often combine multiple approaches to leverage complementary strengths.

Convolutional Neural Networks (CNNs)

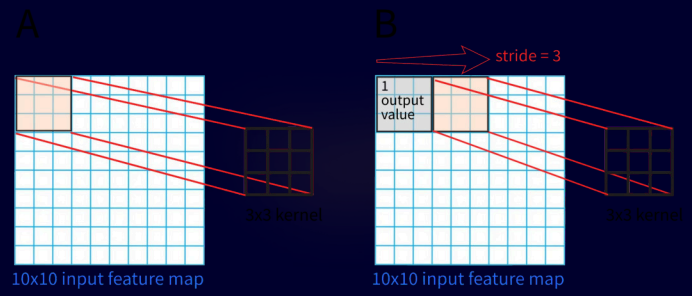
Local Receptive Fields & Shared Weights

Convolutional layers apply sliding filters across input data to detect **local patterns**. Unlike fully connected layers, convolutions exploit two key principles:

- **Local connectivity:** Each neuron connects only to a small spatial region
- **Parameter sharing:** The same filter weights are applied across all locations
- **Translation invariance:** Features are detected regardless of position

Pooling layers then reduce spatial dimensionality while preserving the most salient features, creating a hierarchical representation that becomes increasingly abstract at deeper layers.

This architecture achieves remarkable computational efficiency compared to fully connected networks—a 100×100 image would require 10,000 weights per neuron in a dense layer, but only 9 weights for a 3×3 convolutional filter.



$$y_{ij} = f\left(\sum_{m,n} w_{mn} x_{i+m,j+n} + b\right)$$

The output at position (i,j) is computed by applying the nonlinear activation function f to a weighted sum over the local receptive field, plus bias b.

Typical CNN Structure

CNNs build hierarchical feature representations through alternating convolutional and pooling operations. Early layers detect simple patterns like edges and textures, while deeper layers combine these into increasingly complex and abstract features.

1

Input Image

Raw pixel values (e.g., 28×28×1 grayscale)

2

Conv + ReLU

Extract low-level features: edges, corners, color gradients

3

Pooling

Reduce spatial dimensions, retain important features

4

Conv + ReLU

Combine features into mid-level patterns: textures, shapes

5

Dense Layers

High-level reasoning and classification

6

Output

Class probabilities or predictions

This hierarchical processing mirrors biological visual systems, where simple cells in V1 detect edges, while higher visual areas recognize complete objects.

CNN Example in PyTorch

Here's a practical implementation of a convolutional neural network for image classification. This architecture demonstrates the key components: feature extraction through convolutional layers, and decision-making through fully connected layers.

```
import torch.nn as nn

class CNNExample(nn.Module):
    def __init__(self):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(1, 16, 3, padding=1),  # 1 channel -> 16 filters
            nn.ReLU(),
            nn.MaxPool2d(2),                 # Reduce spatial size by 2x
            nn.Conv2d(16, 32, 3, padding=1), # 16 channels -> 32 filters
            nn.ReLU(),
            nn.MaxPool2d(2)                  # Further reduction
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(32*7*7, 10)            # 10 output classes
        )

    def forward(self, x):
        return self.classifier(self.features(x))
```

01

Feature Extraction

Two convolutional blocks progressively extract spatial features. The first block creates 16 feature maps, the second increases to 32, capturing increasingly complex patterns.

02

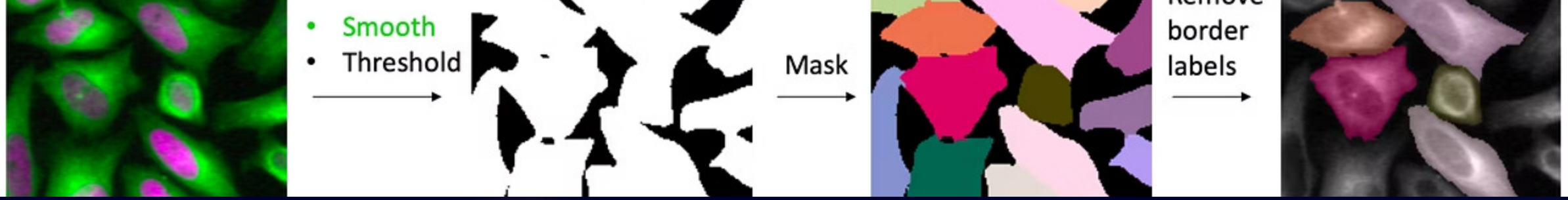
Spatial Reduction

MaxPooling operations reduce the 28×28 input to 7×7, decreasing computation while maintaining discriminative information.

03

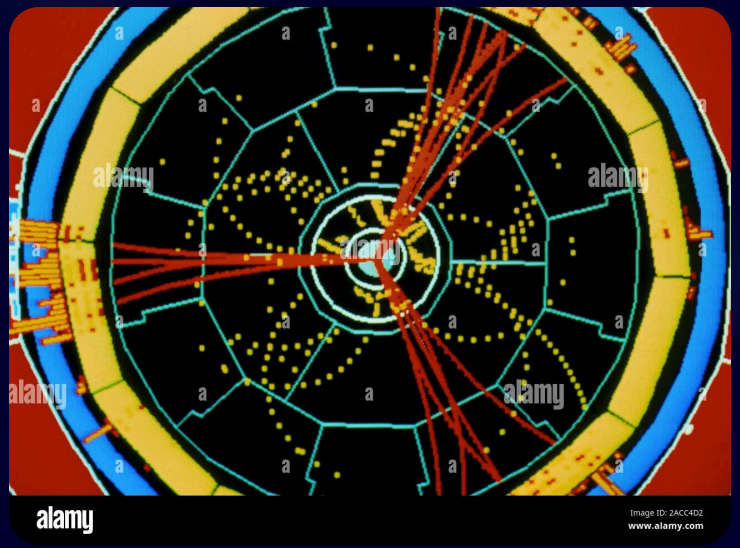
Classification

The flattened feature vector (32×7×7 = 1,568 dimensions) is mapped to 10 output classes through a fully connected layer.



CNNs in Physics

Real-World Scientific Applications



Convolutional neural networks have revolutionized data analysis across physics and related sciences, handling complex pattern recognition tasks that were previously intractable.

- **Particle Physics:** Track reconstruction in detector events at CERN, identifying particle trajectories from millions of collisions per second
- **Microscopy:** Automated cell segmentation and tracking in fluorescence imaging, enabling high-throughput biological analysis
- **Astronomy:** Galaxy classification and gravitational lens detection in large sky surveys
- **Materials Science:** Phase identification in electron microscopy images of crystalline structures

These applications share a common challenge: extracting meaningful patterns from high-dimensional, noisy data where spatial relationships are critical. CNNs excel by learning task-specific features directly from data, often surpassing hand-crafted feature engineering approaches.

Autoencoders

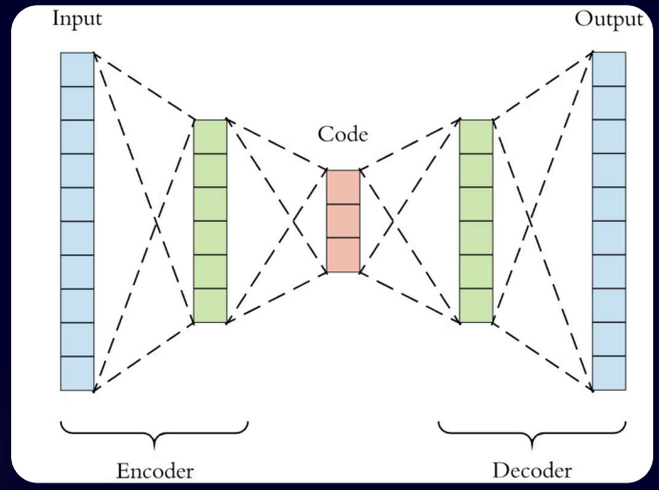
Learning Compact Representations

Autoencoders are unsupervised learning models that compress input data into a low-dimensional **latent space** and then reconstruct it. This process forces the network to capture the most essential information needed to recreate the input.

1. **Input Data**
   High-dimensional observations (images, signals, measurements)
2. **Encoder**
   Compresses input into compact latent representation z
3. **Latent Space**
   Low-dimensional bottleneck captures essential structure
4. **Decoder**
   Reconstructs original input from latent code
5. **Output**
   Reconstruction that preserves key features

Key Applications

- **Denoising:** Train on noisy inputs to reconstruct clean signals
- **Dimensionality Reduction:** Alternative to PCA with nonlinear transformations
- **Anomaly Detection:** Normal data reconstructs well; anomalies have high reconstruction error
- **Data Generation:** Sample from latent space to create new examples



Autoencoder Equations & Loss Function

The autoencoder training objective is to minimize the reconstruction error between input and output, forcing the model to learn an efficient encoding.

Reconstruction Loss

$$\mathcal{L} = \|x - \hat{x}\|^2$$

where $\hat{x} = D(E(x))$ is the reconstructed input

Encoder Function

$$E(x) = f(Wx + b)$$

Maps input x to latent representation z through linear transformation and nonlinearity f (typically ReLU or sigmoid). The weight matrix W defines the learned compression.

Decoder Function

$$D(z) = g(W'z + b')$$

Reconstructs input from latent code z using separate weights W' and activation g. The decoder learns to invert the compression performed by the encoder.

The bottleneck architecture is crucial: by forcing information through a low-dimensional latent space, the network must learn to discard noise and redundancy while preserving essential structure. The dimensionality of z controls the trade-off between compression and reconstruction fidelity.

Training uses standard backpropagation with gradient descent. The network learns both the optimal compression (encoder) and reconstruction (decoder) simultaneously by minimizing the squared difference between input and output.

PyTorch Example: Simple Autoencoder

This implementation demonstrates a basic autoencoder for compressing 28×28 grayscale images (784 pixels) into a 32-dimensional latent representation—a compression ratio of nearly 25:1.

```
class Autoencoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.encoder = nn.Sequential(
            nn.Linear(784, 128),    # First compression layer
            nn.ReLU(),
            nn.Linear(128, 32)      # Bottleneck: 32D latent space
        )
        self.decoder = nn.Sequential(
            nn.Linear(32, 128),     # Expand back from latent
            nn.ReLU(),
            nn.Linear(128, 784),    # Reconstruct to original size
            nn.Sigmoid()            # Output in [0,1] for images
        )

    def forward(self, x):
        z = self.encoder(x)         # Compress
        return self.decoder(z)      # Reconstruct
```

Encoder Path

784 → 128 → 32 dimensions

Progressive compression captures increasingly abstract features in lower-dimensional space

Decoder Path

32 → 128 → 784 dimensions

Symmetric expansion reconstructs original input from compact latent representation

Training tip: The latent space dimensionality (32 in this example) is a critical hyperparameter. Too small, and reconstruction quality suffers; too large, and the model may simply memorize inputs without learning useful structure.

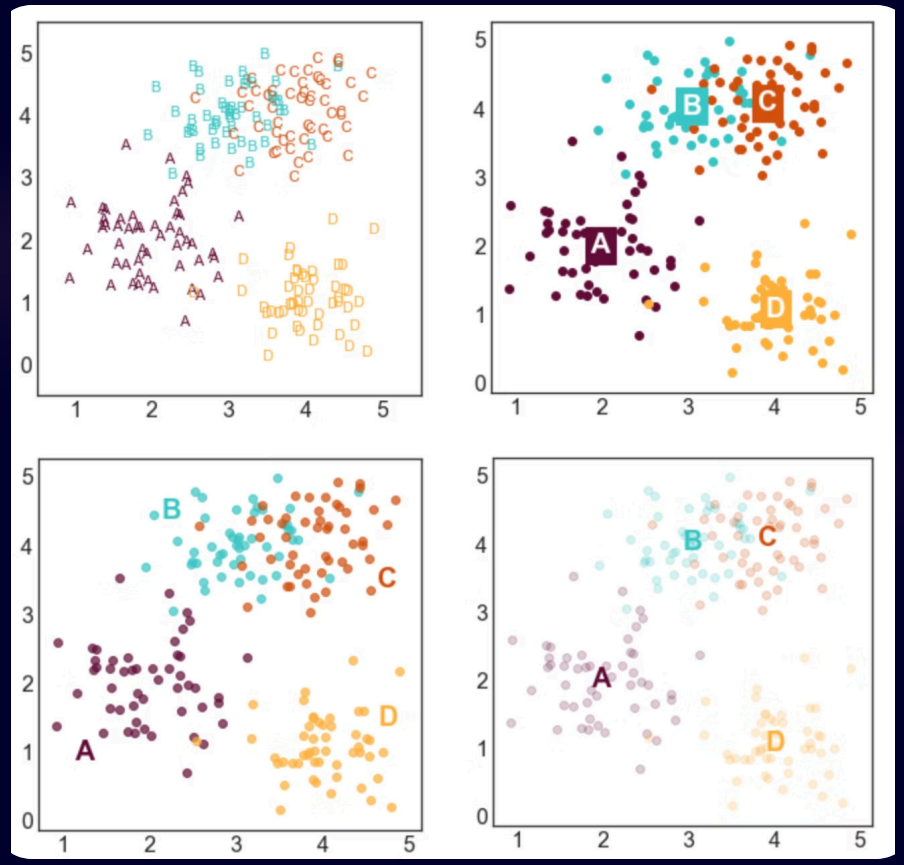
Latent Space in Physics

Understanding Hidden Representations

The latent space learned by autoencoders provides a powerful lens for understanding complex physical systems. By mapping high-dimensional observations into compact representations, autoencoders reveal the **intrinsic dimensionality** and underlying structure of data.

Scientific Applications

- **Phase Transition Identification:** Clustering in latent space reveals distinct physical regimes in simulation data
- **Order Parameter Discovery:** Latent dimensions may correspond to physical quantities characterizing system state
- **Anomaly Detection:** Unusual experimental observations occupy sparse regions of latent space
- **Data Compression:** Efficient storage and transmission of large scientific datasets



For example, in condensed matter simulations, an autoencoder trained on spin configurations can learn a latent representation that naturally separates ordered and disordered phases, even without explicit labels. The latent space coordinates act as learned collective variables that capture essential physics.

Recurrent Neural Networks (RNNs)

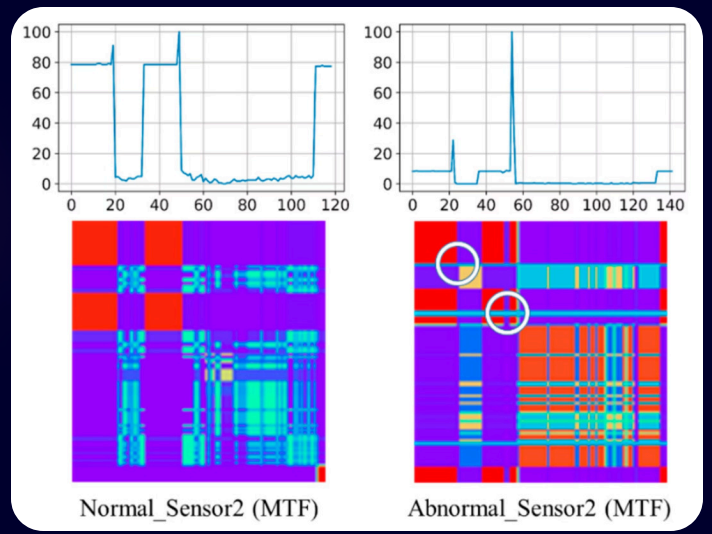
Dealing with Sequences

Traditional feedforward networks treat each input independently, discarding information about temporal order. This limitation becomes critical when analyzing sequential data where past context influences current predictions.

RNNs address this by maintaining a **hidden state** that acts as memory, updated at each time step to incorporate new information while retaining relevant past context.

Why Sequential Models Matter

- **Time Series Forecasting:** Predict future values based on historical patterns
- **Sensor Signal Processing:** Filter noise and detect events in streaming data
- **Experimental Dynamics:** Model temporal evolution in physical systems
- **Video Analysis:** Track objects and recognize actions across frames



Hidden State Update

$$h_t = f(W x_t + U h_{t-1})$$

The hidden state at time t combines current input xt with previous state ht-1 through learned weight matrices W and U, passed through activation f.

This recurrent connection creates an internal memory that can, in principle, capture arbitrarily long-range dependencies. However, standard RNNs struggle with long sequences due to vanishing gradients—a problem addressed by more sophisticated architectures like LSTMs and GRUs.

PyTorch Example: Basic RNN

PyTorch provides built-in RNN modules that handle the recurrent computations efficiently. Here's a simple example processing sequential data:

```
rnn = nn.RNN(
    input_size=10,      # Dimensionality of input features
    hidden_size=20,     # Dimensionality of hidden state
    num_layers=1,       # Single recurrent layer
    batch_first=True    # Input shape: (batch, sequence, features)
)

# Example input: 5 sequences, each 3 time steps, 10 features per step
x = torch.randn(5, 3, 10)

# Forward pass returns outputs and final hidden state
out, h = rnn(x)
```

01

Input Processing

At each time step t, the RNN receives input xt (dimension 10) and combines it with previous hidden state ht-1 (dimension 20)

02

Hidden State Update

The network computes new hidden state by applying learned transformations and nonlinearity, capturing temporal dependencies

03

Output Generation

The RNN produces output at each time step (out) and returns final hidden state (h), which contains summary of entire sequence

The hidden state acts as memory, accumulating information across time steps. For many-to-one tasks (e.g., sequence classification), we typically use only the final hidden state. For many-to-many tasks (e.g., sequence prediction), we use outputs at all time steps.

Variants: LSTM and GRU

Solving the Vanishing Gradient Problem

Standard RNNs struggle to learn long-range dependencies because gradients vanish exponentially with sequence length. Two important variants address this limitation through **gating mechanisms** that control information flow.

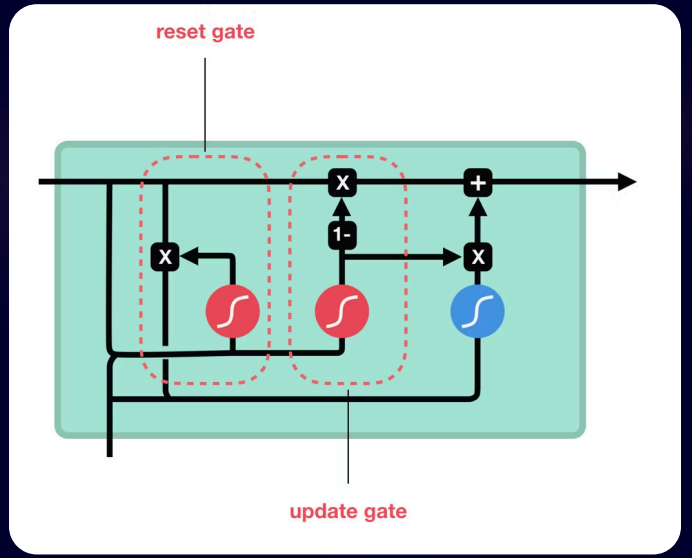
LSTM: Long Short-Term Memory

LSTMs introduce three gates that regulate information:

- **Forget Gate:** Decides what information to discard from cell state
- **Input Gate:** Controls what new information to add
- **Output Gate:** Determines what to output based on cell state

This architecture maintains a separate cell state that can preserve information over many time steps, enabling learning of dependencies spanning hundreds of steps.

```
lstm = nn.LSTM(
    input_size=10,
    hidden_size=32,
    num_layers=2,
    batch_first=True
)
```



GRU: Gated Recurrent Unit

GRUs simplify LSTMs by combining the forget and input gates into a single update gate, and merging cell state with hidden state. This results in fewer parameters while maintaining similar performance on many tasks. GRUs are often faster to train and work well when computational resources are limited.

Sequential Models in Science

Physics Use Cases

Recurrent neural networks have become valuable tools for analyzing temporal data across physics and experimental sciences, where understanding dynamics and making predictions based on history is essential.

Sensor Signal Processing

RNNs filter noise and predict future values in streaming sensor data, enabling real-time anomaly detection and system monitoring in experimental setups.

Microscopy Dynamics

Track cellular movements and morphological changes in time-lapse imaging, predicting future trajectories and identifying behavioral patterns.

Chaotic Systems

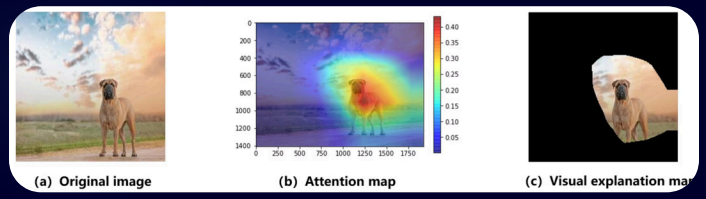
Model and forecast behavior of nonlinear dynamical systems, capturing complex temporal patterns in fluid dynamics and plasma oscillations.

A particularly powerful application is in detecting rare events in high-energy physics experiments, where RNNs can identify anomalous detector signals by learning normal temporal patterns from vast quantities of background data. The network's ability to maintain context allows it to distinguish genuine particle signatures from noise and detector artifacts.

Transformers: The Attention Mechanism

From Recurrence to Attention

While RNNs process sequences sequentially, Transformers introduce a fundamentally different paradigm: the model learns to **attend** to relevant parts of the input regardless of their position. This attention mechanism asks: "Which elements of the input are most important for processing the current element?"



The Attention Operation

Attention computes a weighted sum of values (V), where weights come from comparing queries (Q) with keys (K). This allows each position to directly access information from all other positions.

- **Queries (Q):** "What am I looking for?"
- **Keys (K):** "What do I contain?"
- **Values (V):** "What information should I provide?"

Scaled Dot-Product Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Similarity between queries and keys determines attention weights, which are then applied to values. Scaling by √dk prevents extremely small gradients for large dimensions.

Unlike RNNs, attention can access the entire sequence in parallel, enabling much more efficient training on modern hardware. This parallelization, combined with the ability to model long-range dependencies directly, has made Transformers the dominant architecture for sequence modeling.

Transformer Architecture

Parallel Sequence Modeling

The Transformer architecture consists of encoder and decoder stacks, each built from repeated blocks containing multi-head self-attention and feedforward layers. This design enables modeling complex relationships across entire sequences.

Input Embedding

Convert discrete tokens or continuous features into dense vectors with positional encodings

Multi-Head Self-Attention

Each position attends to all positions in parallel, capturing dependencies at multiple scales

Feed-Forward Networks

Apply nonlinear transformations independently to each position

Layer Normalization

Stabilize training and enable deeper networks through residual connections

Key Advantages

- **Parallelization:** Process entire sequences simultaneously, dramatically reducing training time
- **Long-Range Dependencies:** Direct connections between distant elements avoid vanishing gradients
- **Interpretability:** Attention weights reveal which inputs influence each output

```
from torch.nn import Transformer

model = Transformer(
 d_model=64, # Feature dimension
 nhead=8, # Attention heads
 num_encoder_layers=2 # Encoder depth
)

src = torch.rand(10, 32, 64) # (seq, batch, features)
out = model(src, src)
```

Transformers in Physics

Emerging Applications

While Transformers originated in natural language processing, they are rapidly gaining traction in physics and scientific computing, where their ability to model long-range dependencies and process sequences in parallel offers significant advantages.

Molecular Dynamics

Model interactions between atoms in molecules and materials. Attention mechanisms naturally capture the many-body correlations essential for accurate force prediction, outperforming traditional methods for complex systems.

Quantum State Prediction

Represent and predict quantum many-body states where entanglement creates long-range correlations. Transformers can efficiently encode the exponentially large Hilbert space structure.

Particle Trajectory Analysis

Track particles through detector systems by attending to spatial and temporal relationships. This approach handles complex scenarios with particle interactions and multiple scattering events.

Microscopy Sequence Interpretation

Analyze time-lapse microscopy by modeling dependencies across spatial locations and time. Attention reveals which cellular regions influence others during dynamic processes.

The interpretability of attention weights is particularly valuable in scientific applications, providing insights into which features or time points drive predictions—essentially offering a learned, data-driven hypothesis about causal relationships in the system.

Summary

Architectural Principles & Trade-offs

CNNs

Exploit **spatial structure** through local receptive fields and parameter sharing. Ideal for image and grid-like data where nearby elements interact strongly.

Autoencoders

Learn **compressed latent representations** that capture essential data structure. Enable dimensionality reduction, denoising, and anomaly detection.

RNNs

Model **sequential dependencies** through recurrent hidden states that maintain temporal context. Effective for time-series and streaming data.

Transformers

Capture **global context** via attention mechanisms that directly relate all sequence elements. Parallel processing and long-range modeling.

Universal Challenges

- Gradient-based optimization susceptible to local minima
- Large data requirements for effective learning
- Overfitting mitigation through regularization
- Hyperparameter sensitivity and tuning complexity
- Interpretability versus performance trade-offs
- Computational resource demands for training

The choice of architecture should be guided by the structure inherent in your data: spatial relationships favor CNNs, sequential dependencies suggest RNNs or Transformers, and unsupervised structure discovery points to autoencoders. Modern applications often combine these approaches in hybrid architectures.

Discussion & Outlook

Future Directions in Neural Architecture Design

As neural architectures mature, the frontier of research shifts toward integrating domain knowledge and addressing fundamental questions about their capabilities and limitations in scientific contexts.

Physics-Informed Learning

How can we embed physical laws—conservation principles, symmetries, differential equations—directly into network architectures? This could reduce data requirements and ensure physically plausible predictions.

Universal Architectures

Are Transformers truly universal? Their success across domains suggests they may approximate a general-purpose learning mechanism, but understanding their inductive biases remains an open question.

Generative Modeling

Beyond classification and prediction, generative models like VAEs and diffusion models enable sampling from learned distributions—crucial for simulation and uncertainty quantification.

Emerging Research Directions

- **Physics-Informed Transformers:** Attention mechanisms that respect causality and physical constraints
- **Multimodal Fusion:** Combining heterogeneous data sources (images, signals, text) for richer scientific understanding
- **Uncertainty Quantification:** Developing architectures that provide reliable confidence estimates, essential for scientific decision-making
- **Causal Discovery:** Moving beyond correlation to identify causal relationships in observational data

The convergence of machine learning and physical sciences promises powerful new tools for understanding complex systems, but success requires careful consideration of both statistical learning principles and domain-specific constraints.