Deep Learning in Actuarial Science

Overview Lecture
36th International Summer School SAA
University of Lausanne

Ronald Richman, Salvatore Scognamiglio, Mario V. Wüthrich

8 September 2025

Course Overview

Today's Overview Lecture:

- Historical perspective and motivation
- Representation learning as core principle
- Feed-forward networks (FNNs)
- Combined Actuarial Neural Networks
- CNNs, RNNs, Transformers (in brief)
- Applications and future directions

Future Lectures:

- Detailed review of FNN architecture
- Transformers and attention mechanisms
- LocalGLMnet for interpretability
- Training and optimization
- Foundation Models

Why Deep Learning in Actuarial Science?

- **Traditional approach**: Manual feature engineering, model specification
- **Modern challenges**: High-dimensional data, complex interactions, unstructured data
- **Deep learning promises**: Automated feature learning, universal approximation
- **Actuarial opportunity**: Better risk assessment, improved predictions, novel data sources

Deep learning allows us to move from hand-crafted features to learned representations

Historical Perspective: The Evolution - 1

- **1943**: McCulloch–Pitts neuron formalizes a logical model of a neuron (McCulloch & Pitts, 1943).
- **1958**: Rosenblatt's perceptron (Rosenblatt, 1958).
- **1969**: Minsky–Papert highlight perceptron limits and help trigger the first AI winter (Minsky & Papert, 1969).
- **1986**: Backpropagation popularized (Rumelhart, Hinton, & Williams, 1986).
- **1989, 1991**: Universal Approximation Theorem for MLPs (Cybenko, 1989; Hornik, 1991).
- **1997, 1998**: LSTM for long-range sequence memory (Hochreiter & Schmidhuber, 1997); LeNet-5 for digit recognition (LeCun, Bottou, Bengio, & Haffner, 1998).

Historical Perspective: The Evolution - 2

- **2012**: AlexNet revolutionizes vision with GPU training (Krizhevsky, Sutskever, & Hinton, 2012).
- **2014, 2015-2016**: Seq2Seq and residual networks extend depth and stability (Sutskever, Vinyals, & Le, 2014; He, Zhang, Ren, & Sun, 2016).
- **2017**: **Unifying architecture:**Transformer architecture and scaled dot-product attention (Vaswani et al., 2017).

The Transformer model started off as a clever way to perform Seq2Seq tasks. Decomposing it into the encoder and decoder pieces led to decoder only models, which are now the standard within LLMs.

Historical Perspective: The Evolution - 3

- **2018+**: Deep learning enters actuarial practice
- **2020–2022**: Foundation models as a new paradigm
  - GPT-3 shows few-shot generalization at scale (Brown et al., 2020); BERT popularizes bidirectional pretraining and CLS token.(Devlin, Chang, Lee, & Toutanova, 2018).
  - Foundation model framing and risks (Bommasani et al., 2021).
  - Instruction following with human feedback (Ouyang et al., 2022).
- **2022–2024**: Reasoning-focused training and prompting
  - Chain-of-thought and zero-shot CoT (Wei et al., 2022; Kojima, Gu, Reid, Matsuo, & Iwasawa, 2022).
  - Self-consistency improves multi-step reasoning (Wang et al., 2022).

Scaling Laws

Key insight: Depth + data + compute (GPUs/TPUs) → predictable scaling. Compute-optimal training laws guide model and dataset sizing (Kaplan et al., 2020; Hoffmann et al., 2022).

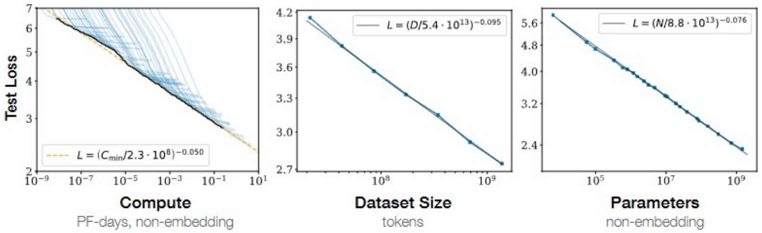


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

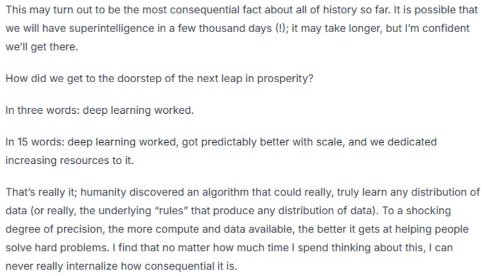
Practical Successes of Deep Learning

- Computer vision starting with AlexNet architecture of Krizhevsky, Sutskever and Hinton (2012)
- Speech recognition (Hannun, Case, Casper et al. 2014).
- Natural language processing, e.g. Google's neural translation machine (Wu, Schuster, Chen et al. 2016)
- Analysis of GPS data (Brébisson, Simon, Auvolat et al. 2015)
- Winning method in 2018 M4 time series forecasting competition (Makridakis, Spiliotis and Assimakopoulos 2018a).
- Analysis of tabular data (Guo and Berkhahn 2016) (plus other Kaggle competitions)

Deep Learning Worked!

The Intelligence Age

September 23, 2024



This may turn out to be the most consequential fact about all of history so far. It is possible that we will have superintelligence in a few thousand days (!); it may take longer, but I'm confident we'll get there.

How did we get to the doorstep of the next leap in prosperity?

In three words: deep learning worked.

In 15 words: deep learning worked, got predictably better with scale, and we dedicated increasing resources to it.

That's really it; humanity discovered an algorithm that could really, truly learn any distribution of data (or really, the underlying "rules" that produce any distribution of data). To a shocking degree of precision, the more compute and data available, the better it gets at helping people solve hard problems. I find that no matter how much time I spend thinking about this, I can never really internalize how consequential it is.

The Feature Engineering Problem

Traditional Actuarial Approach:

- Manual variable selection
- Domain expertise required
- Interactions specified by hand
- Time-consuming process

Example: French MTPL pricing

- Must decide: include Age^2? Age $\times$ Region?
- Hundreds of possible interactions

The Curse of Dimensionality:

- Telematics: 1000s of variables
- Text data: unstructured
- Images: pixel-level data
- Time series: temporal dependencies

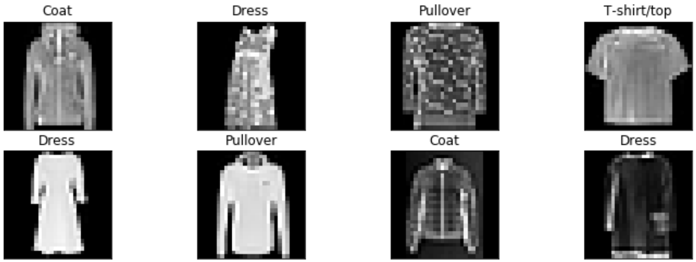
Manual feature engineering becomes infeasible!

Representation Learning: Automated Feature Discovery

- **Definition**: Learning transformations of data that make it easier to extract useful information
- **Key idea**: Let the model discover the features
- **Traditional examples**: PCA and Partial Least Squares (PLS)
- **Deep learning approach**: Hierarchical feature learning

Fashion-MNIST Example: Dataset Overview

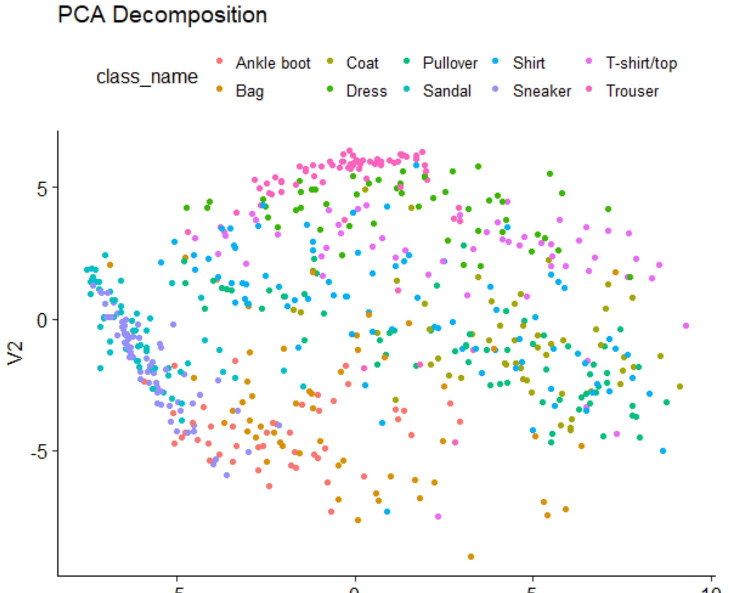
- Fashion-MNIST dataset contains 70,000 grayscale images (28x28 pixels) of clothing items.
- Task: Classify the type of clothing.
- Ideal testbed for comparing representation learning approaches.



Fashion-MNIST: Traditional PCA Limitations

- Applying PCA directly to the images does not show much differentiation between classes.
- Linear dimensionality reduction fails to capture the complex patterns in image data.
- Manual feature engineering would be required to improve performance.

Fashion-MNIST: Traditional PCA Results



Representation Learning

- **Representation Learning** is an ML technique where algorithms automatically design features that are optimal for a particular task.
- Traditional examples are PCA (unsupervised) and PLS (supervised).
- The feature space is then comprised of learned features which can be fed into an ML/DL model.
- BUT: Simple representation learning approaches often fail when applied to high dimensional data.

Representation learning at a glance: Given inputs $X \in \mathbb{R}^p$ and target Y, learn a mapping $\phi : \mathbb{R}^p \to \mathbb{R}^d$ with $d \ll p$ so that simple predictors on $Z = \phi(X)$ perform well (Bengio, Courville, & Vincent, 2013).

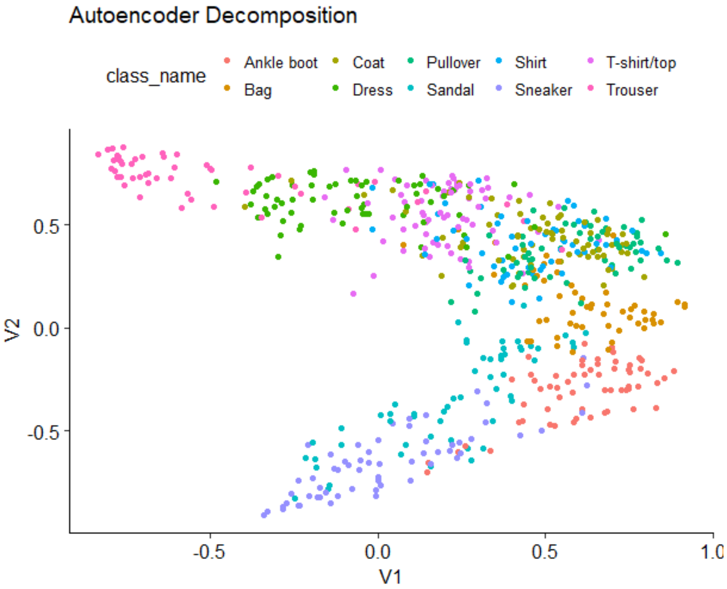
Deep Learning

- **Deep Learning** is a representation learning technique that automatically constructs hierarchies of complex features to represent abstract concepts.
- Features in higher layers (closer to outputs) are composed of simpler features from lower (closer to inputs) layers.
- A typical example is a feed-forward neural network.
- **The principle**: Provide raw data to the network and let it figure out what and how to learn.

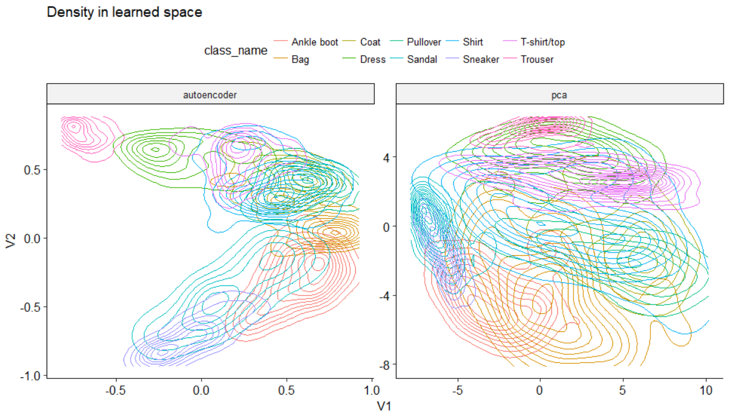
Fashion-MNIST: Deep Learning Solution

- We applied a deep autoencoder (a type of non-linear PCA) to the same data.
- Differences between some classes are now much more clearly emphasized.
- The deep representation automatically captures meaningful differences between the images without much human input.
- This is an example of automated feature/model specification.

Fashion-MNIST: Deep Learning Results



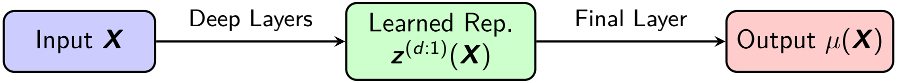
Fashion-MNIST: Learned Feature Density



Density plot of learned features from a deep autoencoder on the Fashion-MNIST dataset, showing clear separation between clothing categories.

From Representation to Prediction

- **Step 1**: Deep layers learn representations
- **Step 2**: Final layer performs prediction using learned features
- **Key insight**: Deep network = Feature extractor + Predictor


Input $\boldsymbol{X}$
Deep Layers
Learned Rep. $\boldsymbol{z}^{(d:1)}(\boldsymbol{X})$
Final Layer
Output $\mu(\boldsymbol{X})$

Evaluating representation quality: linear probing

- One can report **Linear probe** accuracy to gauge quality of the representation $Z = \phi(X)$ (Alain & Bengio, 2017):

- For multi-class, softmax head with cross-entropy

$$\min_{W,b} \; -\frac{1}{n}\sum_{i=1}^{n} \log \frac{\exp(w_{y_i}^\top z_i + b_{y_i})}{\sum_{c=1}^{C} \exp(w_c^\top z_i + b_c)}.$$

- For regression, linear head with loss L

$$\min_{W,b} \; \frac{1}{n}\sum_{i=1}^{n} L(y_i,\; Wz_i + b).$$

- Other types of probes can be used too.

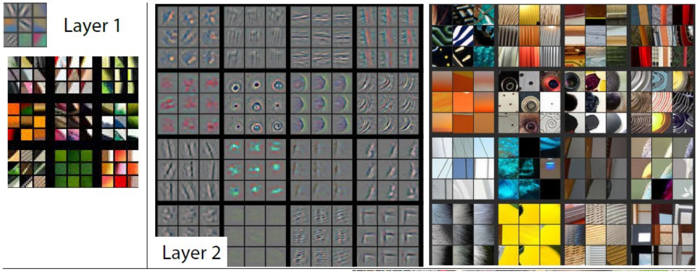
Why Depth is Important - 1

- Depth in neural networks refers to the number of layers, allowing for hierarchical feature learning.
- **Capturing deep interactions**: Shallow networks can only model simple, direct relationships between inputs and outputs.
- Deeper layers build upon lower-level features: early layers detect basic patterns (e.g., edges in images), while deeper layers combine them into complex abstractions (e.g., objects or concepts).
- This hierarchy enables the network to capture intricate, non-linear interactions and dependencies in data that would require exponentially more parameters in a shallow model.

Why Depth is Important - 2

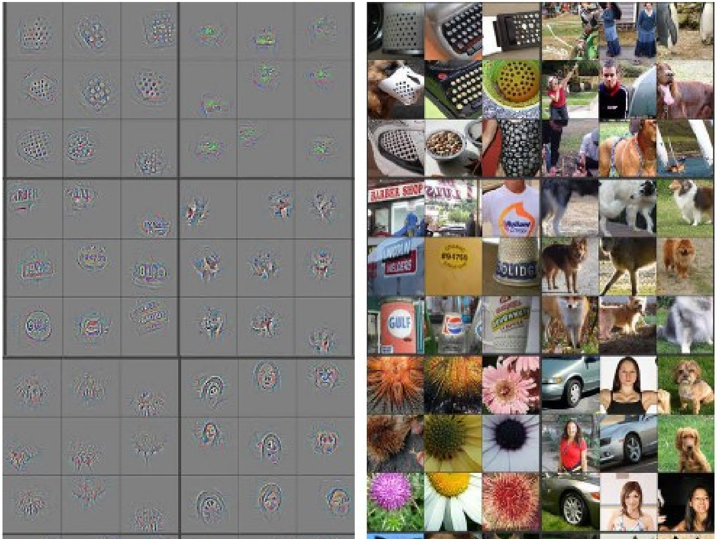
- Theoretical support: Deep networks can approximate complex functions more efficiently than shallow ones (universal approximation theorem extensions).

Visualization of Layers 1 and 2 of a CNN (Zeiler & Fergus, 2014)



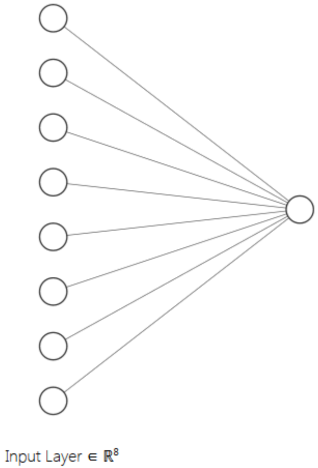
Why Depth is Important - 3

Visualization of Layer 5 of a CNN (Zeiler & Fergus, 2014)



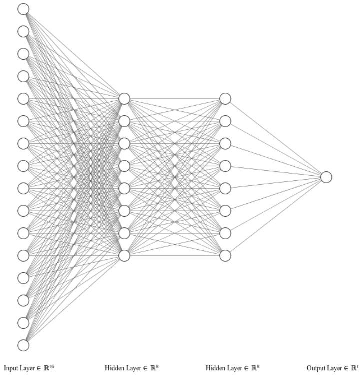
Single Layer NN = GLM

- **Single layer neural network**
- Circles = variables
- Lines = connections
- The input layer holds the variables...
- ...which are multiplied by weights (coefficients) to get the result.
- A single layer neural network is essentially a GLM!



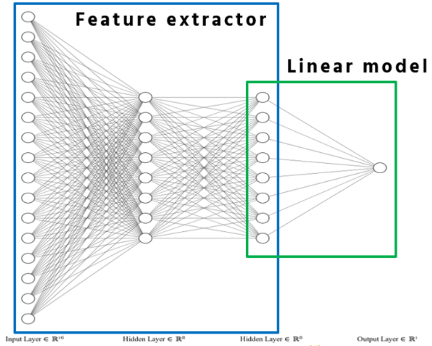
Deep Feedforward Net

- **Deep** = multiple layers
- **Feedforward** = data travels from left to right
- More complicated representations of input data are learned in the hidden layers.


Input Layer $\in \mathbb{R}^{16}$
Hidden Layer $\in \mathbb{R}^{8}$
Hidden Layer $\in \mathbb{R}^{8}$
Output Layer $\in \mathbb{R}^{1}$

FCN generalizes GLM

- Intermediate layers perform representation learning.
- The last layer is a (generalized) linear model, where input variables are the new representation of the data.
- You can strip off the last layer and use the learned features in another model, like XGBoost.



[Diagram: FCN generalizes GLM]

Mathematical Foundation: FNN Architecture

Notation:

- Input: $\boldsymbol{X} \in \mathbb{R}^{q_0}$
- Hidden layers: $\ell = 1, \dots, d$ with q_ℓ neurons each
- Layer ℓ transformation: $\boldsymbol{z}^{(\ell)} : \mathbb{R}^{q_{\ell-1}} \to \mathbb{R}^{q_\ell}$
- Composition: $\boldsymbol{z}^{(d:1)} = \boldsymbol{z}^{(d)} \circ \cdots \circ \boldsymbol{z}^{(1)}$

FNN regression function:

$$\mu_{\vartheta}(\boldsymbol{X}) = g^{-1}\left(\langle \boldsymbol{w}^{(d+1)}, \boldsymbol{z}^{(d:1)}(\boldsymbol{X})\rangle + w_0^{(d+1)}\right) \tag{1}$$

where g^{-1} is the inverse link function and ϑ contains all network parameters.

Pre- and Post-Activation in Neural Networks

- **Pre-activation**: In each layer of a neural network, the inputs are multiplied by the layer's weights and added to biases, forming a linear combination. This step computes the weighted sum: $z = Wx + b$, where x is the input, W are weights, and b is the bias.
- This linear transformation aggregates information from the previous layer but remains linear without further processing.
- **Post-activation**: The pre-activation output z is passed through a non-linear activation function $a = \sigma(z)$, such as ReLU or sigmoid.
- The activation introduces non-linearity, enabling the network to model complex patterns. Without it, the network would behave like a single linear model regardless of depth.

Importance of Non-Linearity in Activation Functions

- **Why non-linearity matters**: Non-linear activations (e.g., ReLU, sigmoid) allow networks to model arbitrary complex functions by introducing bends and thresholds in the decision boundaries.
- Without non-linearity, each layer would only perform linear transformations, and stacking layers would collapse to a single linear model—no matter the depth.
- **Contrast with linear activations**: If all activations are linear (e.g., $\sigma(z) = z$), the entire network reduces to $y = W_n \cdots W_1 x + b$, equivalent to a single-layer linear regression. This limits the model to linear relationships, failing on non-linear problems like XOR or image classification.
- Non-linearity enables universal approximation: Deep non-linear networks can **theoretically** approximate any continuous function.

From Vectors to Tensors

Traditional tabular data: $\boldsymbol{X}_i \in \mathbb{R}^q$ (1D tensor)

Extended tensor structures:

- **Time series**: $\boldsymbol{X}_{1:t} \in \mathbb{R}^{t \times q}$ (2D tensor)
- **Images**: $\boldsymbol{X}_{1:t,1:s} \in \mathbb{R}^{t \times s \times 3}$ (3D tensor, RGB)
- **Panel data**: Multiple instances over time (4D tensor)

Key challenge: How to design architectures that respect structure?

- FNNs: Ignore spatial/temporal structure
- CNNs: Exploit local patterns via convolution
- RNNs: Capture sequential dependencies

Convolutional Neural Networks: Motivation

Problem with FNNs for spatial/temporal data:

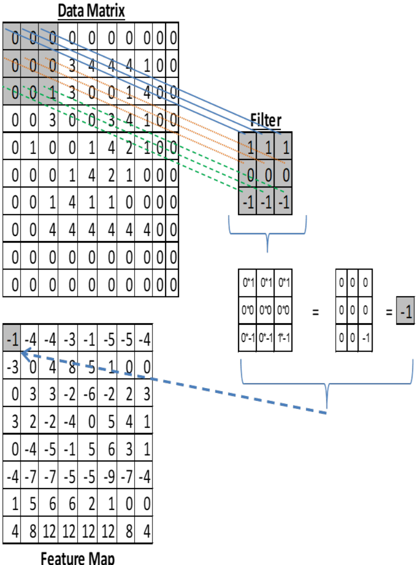
- Full connectivity: Every input connects to every neuron
- Parameter explosion: Image $100 \times 100 \times 3 \rightarrow 30,000$ inputs!
- No spatial awareness: Adjacent pixels treated independently

CNN solution (LeCun et al., 1998):

- **Local connectivity**: Neurons connect to small regions (receptive fields)
- **Parameter sharing**: Same filter applied across all positions
- **Hierarchical learning**: Low-level $\rightarrow$ High-level features

Convolutional Neural Networks: Image

=



Data Matrix

0	0	0	0	0	0	0	0	0	0
0	0	0	3	4	4	4	1	0	0
0	0	1	3	0	0	1	4	0	0
0	0	3	0	0	3	4	1	0	0
0	1	0	0	1	4	2	1	0	0
0	0	0	1	4	2	1	0	0	0
0	0	1	4	1	1	0	0	0	0
0	0	4	4	4	4	4	4	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Filter

1	1	1
0	0	0
-1	-1	-1

$$\begin{bmatrix} 0*1 & 0*1 & 0*1 \\ 0*0 & 0*0 & 0*0 \\ 0*-1 & 0*-1 & 1*-1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} = \boxed{-1}$$

Feature Map

-1	-4	-4	-3	-1	-5	-5	-4
-3	0	4	8	5	1	0	0
0	3	3	-2	-6	-2	2	3
3	2	-2	-4	0	5	4	1
0	-4	-5	-1	5	6	3	1
-4	-7	-7	-5	-5	-9	-7	-4
1	5	6	6	2	1	0	0
4	8	12	12	12	12	8	4

1D CNN: Mathematical Formulation

Input: Time series $\boldsymbol{X}_{1:t} \in \mathbb{R}^{t \times q}$

1D CNN layer with q_1 filters:

$$\boldsymbol{z}^{(1)} : \mathbb{R}^{t \times q} \to \mathbb{R}^{t' \times q_1} \tag{2}$$

where $t' = \lfloor \frac{t-K}{\delta} + 1 \rfloor$ (number of windows)

Each unit computed as:

$$z_{u,j}^{(1)} = \phi\left(w_{0,j}^{(1)} + \sum_{k=1}^{K} \langle \boldsymbol{w}_{k,j}^{(1)}, \boldsymbol{X}_{(u-1)\delta+k} \rangle \right) \tag{3}$$

- K: Kernel size (window width)
- δ: Stride (step size)
- $\boldsymbol{w}_{k,j}^{(1)} \in \mathbb{R}^q$: Filter weights

1D CNN: Intuition and Applications

Rolling window analogy:

- Kernel = Window size
- Stride = Step size
- Filter = Pattern detector

Example parameters:

- $K = 3$, $\delta = 1$: Overlapping
- $K = 3$, $\delta = 3$: Non-overlapping

Actuarial applications:

- **Claims triangles**: Detect development patterns
- **Telematics**: Speed-acceleration patterns
- **Time series**: Seasonal effects in frequency

Advantages:

- Captures local patterns
- Translation equivariant
- Efficient computation

2D CNN: Spatial Pattern Recognition

Input: Spatial data $\boldsymbol{X}_{1:t,1:s} \in \mathbb{R}^{t\times s\times q}$

2D CNN layer mapping:

$$\boldsymbol{z}^{(1)} : \mathbb{R}^{t\times s\times q} \to \mathbb{R}^{t'\times s'\times q_1} \tag{4}$$

Convolution operation:

$$z^{(1)}_{u,v,j} = \phi\left(w^{(1)}_{0,j} + \sum_{k_t=1}^{K_t}\sum_{k_s=1}^{K_s} \langle \boldsymbol{w}^{(1)}_{k_t,k_s,j}, \boldsymbol{X}_{(u-1)\delta_t+k_t,(v-1)\delta_s+k_s} \rangle \right) \tag{5}$$

- Kernel: (K_t, K_s) - height and width
- Stride: (δ_t, δ_s) - vertical and horizontal steps
- Parameters: $(1 + K_t K_s q) \times q_1$

Deep CNN: Hierarchical Feature Learning

Composing multiple CNN layers:

- **Layer 1**: Low-level features (edges, simple patterns)
- **Layer 2**: Mid-level features (combinations)
- **Layer 3+**: High-level abstractions

Recurrent Neural Networks: Sequential Memory

Motivation: Process sequences of variable length with memory

Plain-vanilla RNN layer (final state):

$$\boldsymbol{z}^{(1)} : \mathbb{R}^{t\times q} \to \mathbb{R}^{q_1} \tag{6}$$

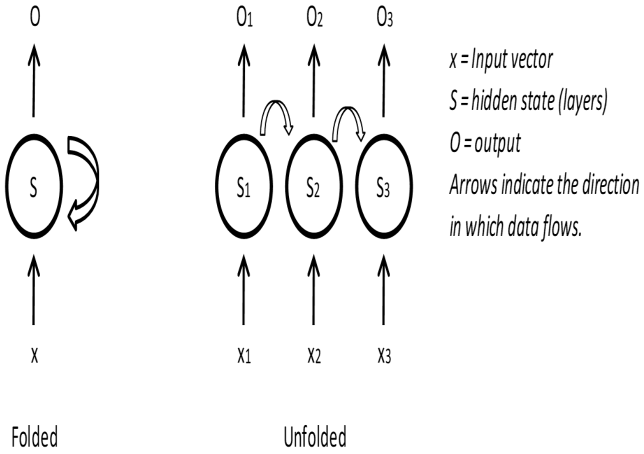
Recurrent update at time u:

$$z_{u,j}^{(1)} = \phi\left(w_{0,j}^{(1)} + \langle \boldsymbol{w}_j^{(1)}, \boldsymbol{X}_u\rangle + \langle \boldsymbol{v}_j^{(1)}, \boldsymbol{z}_{u-1}^{(1)}\rangle\right) \tag{7}$$

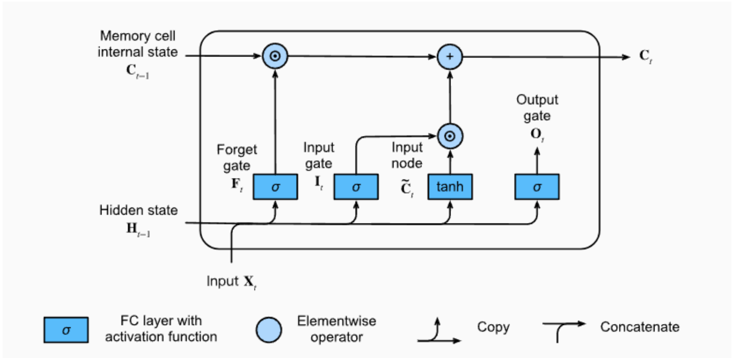
- $\boldsymbol{X}_u$: Current input
- $\boldsymbol{z}_{u-1}^{(1)}$: Previous hidden state (memory)
- Parameters: $q_1(1+q+q_1)$ (shared across time)

Key insight: Same as FNN but with recurrent connection!

RNN Diagram


O
S
X
Folded
O1
O2
O3
S1
S2
S3
X1
X2
X3
Unfolded
x = Input vector
S = hidden state (layers)
O = output
Arrows indicate the direction
in which data flows.

LSTM Architecture Diagram


Memory cell internal state $\mathbf{C}_{t-1}$
$\mathbf{C}_t$
Output gate $\mathbf{O}_t$
Forget gate $\mathbf{F}_t$
Input gate $\mathbf{I}_t$
Input node $\tilde{\mathbf{C}}_t$
σ
tanh
Hidden state $\mathbf{H}_{t-1}$
Input $\mathbf{X}_t$
FC layer with activation function
Elementwise operator
Copy
Concatenate

LSTM cell structure showing gates and information flow (Zhang, Lipton, Li, & Smola, 2023)

LSTM: Long Short-Term Memory - 1

Problem: Vanilla RNNs suffer from vanishing gradients

LSTM solution (Hochreiter & Schmidhuber, 1997):

- **Memory cell $\boldsymbol{c}_u$**: Long-term storage
- **Hidden state $\boldsymbol{z}_u$**: Short-term output
- **Gates**: Control information flow

Three gates regulate memory:

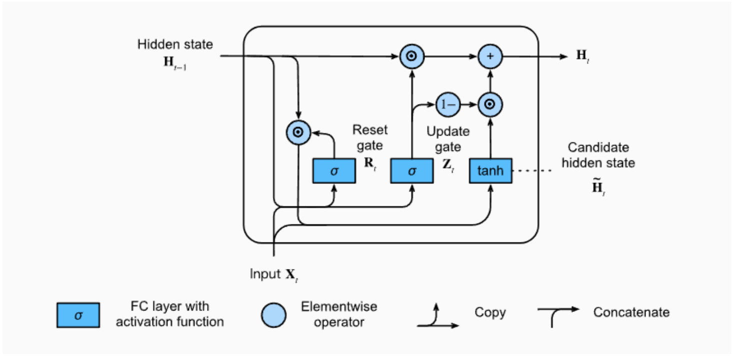
- **Forget gate $\boldsymbol{f}_u = \sigma(W_f \boldsymbol{X}_u + V_f \boldsymbol{z}_{u-1})$**
- **Input gate $\boldsymbol{i}_u = \sigma(W_i \boldsymbol{X}_u + V_i \boldsymbol{z}_{u-1})$**
- **Output gate $\boldsymbol{o}_u = \sigma(W_o \boldsymbol{X}_u + V_o \boldsymbol{z}_{u-1})$**

LSTM: Long Short-Term Memory - 2

Memory update:

$$\boldsymbol{c}_u = \boldsymbol{f}_u \odot \boldsymbol{c}_{u-1} + \boldsymbol{i}_u \odot \tilde{\boldsymbol{c}}_u \tag{8}$$

GRU Architecture Diagram


Hidden state $\mathbf{H}_{t-1}$
$\mathbf{H}_t$
Reset gate $\mathbf{R}_t$
Update gate $\mathbf{Z}_t$
Candidate hidden state $\tilde{\mathbf{H}}_t$
σ
σ
tanh
1−
+
Input $\mathbf{X}_t$
FC layer with activation function
Elementwise operator
Copy
Concatenate

GRU cell structure showing gates and information flow (Zhang et al., 2023)

GRU: Gated Recurrent Unit

Simplified alternative to LSTM (Cho, van Merriënboer, Bahdanau, & Bengio, 2014):

Two gates instead of three:

- **Update gate $\boldsymbol{o}_u$**: How much past to keep
- **Reset gate $\boldsymbol{r}_u$**: How much past to forget

Hidden state update:

$$\boldsymbol{z}_u = (1 - \boldsymbol{o}_u) \odot \boldsymbol{z}_{u-1} + \boldsymbol{o}_u \odot \tilde{\boldsymbol{z}}_u \tag{9}$$

where candidate state:

$$\tilde{\boldsymbol{z}}_u = \tanh(W_z \boldsymbol{X}_u + V_z(\boldsymbol{r}_u \odot \boldsymbol{z}_{u-1})) \tag{10}$$

Advantages: Fewer parameters than LSTM, often similar performance

CNN vs RNN: When to Use Which?

Use CNNs when:

- Local patterns matter
- Translation invariance needed
- Spatial/grid structure exists
- Parallel processing required

Examples:

- Telematics heatmaps
- Geographic risk maps
- Fixed-size triangles

Use RNNs when:

- Sequential order crucial
- Variable lengths common
- Long-term dependencies
- Temporal causality important

Examples:

- Policy history modeling
- Text processing (claims)
- Time series forecasting

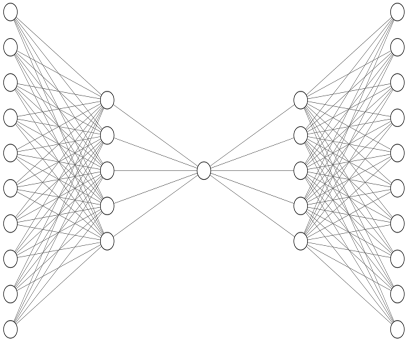
Hybrid approaches: CNN for feature extraction $\rightarrow$ *RNN for sequences*

Embedding Layer – Categorical Data

- One-hot encoding expresses the prior that categories are orthogonal.
- The traditional actuarial solution is credibility.
- **Embedding layer prior**: related categories should cluster together.
  - It learns a dense vector transformation of sparse input vectors and clusters similar categories.
  - Can be pre-calibrated to MLE of GLM models (CANN proposal of Wüthrich and Merz 2019).

Autoencoder – Unsupervised Learning

- An autoencoder is a network trained to produce an output equal to its input.
- A "bottleneck" in the middle restricts the dimension of the encoded data.
- It performs a type of non-linear PCA.
- The bottleneck layer expresses the prior that the data can be summarized in only a few dimensions.



Transformers and Attention - Vaswani et al. (2017)

Key idea: Attention mechanism - “focus on what’s relevant”

Self-attention formula:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{11}$$

where Q, K, V are query, key, value matrices

Advantages:

- Parallel processing (unlike RNNs) = efficient computation on GPUs
- Long-range dependencies
- Somewhat interpretable attention weights

Full coverage in dedicated transformer lecture

LocalGLMnet: Interpretability Focus

Key idea: Local linear models defined by neural network

Architecture:

- Neural network learns coefficients of local GLM
- Each combination of covariates recieves own GLM
- Predicions made as sum over $\beta_j(X_j)X_j$

Benefits:

- Full interpretability for predictions
- Captures heterogeneity
- More regulatory compliance friendly than FNN

Detailed treatment in LocalGLMnet lecture

CANN: Bridging Classical and Modern

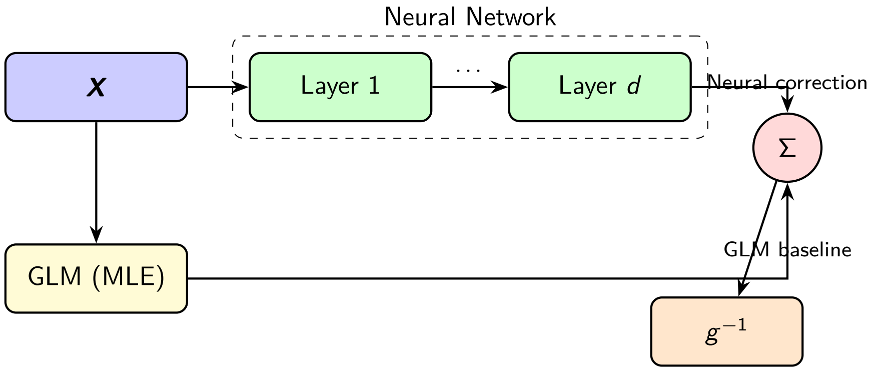
Motivation:

- GLMs are interpretable but may miss complex patterns
- Deep networks are powerful but less interpretable
- Can we combine the best of both worlds?

CANN Architecture (Wüthrich & Merz 2019):

$$\mu^{\mathrm{CANN}}(\boldsymbol{X}) = g^{-1}\left(\underbrace{\langle \widehat{\boldsymbol{\vartheta}}^{\mathrm{MLE}}, \boldsymbol{X}\rangle}_{\text{GLM baseline}} + \underbrace{\langle \boldsymbol{w}^{(d+1)}, \boldsymbol{z}^{(d:1)}(\boldsymbol{X})\rangle}_{\text{Neural correction}}\right) \tag{12}$$

CANN: Implementation Details



Training procedure:

1. Fit GLM using MLE, freeze parameters $\widehat{\boldsymbol{\vartheta}}^{\mathrm{MLE}}$
2. Train neural network to learn residual patterns; this adds corrections to GLM predictions

CANN: Advantages

- **Interpretability**: GLM component provides baseline understanding
- **Performance**: Neural network captures non-linear patterns
- **Stability**: GLM ensures reasonable predictions even if NN fails
- **Diagnostic**: NN contribution shows where GLM is insufficient

Connection to ResNet:

- First term: residual/skip connection (input $\boldsymbol{X}$ directly to output)
- Second term: classic FNN architecture for non-linear corrections
- Interpretation: Linear GLM term + non-linear FNN for interactions not captured by GLM

Key Takeaways

- **Core principle**: Representation learning automates feature engineering
- **Mathematical foundation**: FNNs generalize GLMs through composition
- **Advanced architectures**: CNNs, RNNs, Transformers for specific data types
- **Practical approach**: CANN bridges traditional and modern methods

References I

Alain, G., & Bengio, Y. (2017). Understanding intermediate layers using linear classifier probes. In *International conference on learning representations (iclr), workshop track.* Retrieved from https://arxiv.org/abs/1610.01644 (arXiv:1610.01644)

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(8), 1798–1828.

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., . . . others (2021). On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., . . . others (2020). Language models are few-shot learners. *Advances in neural information processing systems*, *33*, 1877–1901.

References II

Cho, K., van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). Learning phrase representations using rnn encoder–decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*. Retrieved from `https://arxiv.org/abs/1406.1078`

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, *2*(4), 303–314.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 770–778).

Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, *18*(7), 1527–1554.

References III

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., . . . others (2022). Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, *4*(2), 251–257.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., . . . Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., & Iwasawa, Y. (2022). Large language models are zero-shot reasoners. *Advances in neural information processing systems*, *35*, 22199–22213.

References IV

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*, 1097–1105.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115–133.

Minsky, M., & Papert, S. (1969). *Perceptrons: An introduction to computational geometry*. MIT press.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., … others (2022). Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, *35*, 27730–27744.

References V

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, *65*(6), 386.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, *323*(6088), 533–536.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, *27*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., . . . Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Sharan, N., . . . Zhou, D. (2022). Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

References VI

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., ... others (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, *35*, 24824–24837.

Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Computer vision – eccv 2014* (pp. 818–833). Springer. Retrieved from `https://doi.org/10.1007/978-3-319-10590-1_53` doi: 10.1007/978-3-319-10590-1_53

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2023). *Dive into deep learning*. Cambridge University Press.