DEEP LEARNING

Paul Bilokon, PhD
CEO, Thalesians Ltd.
Visiting Professor, Imperial College London

Joseph Simonian, PhD
Senior Affiliate Researcher, CFA Institute

Introduction

Deep learning refers to a type of machine learning algorithm that uses multiple layers to progressively extract higher-level features from input data. For example, in image processing, lower layers may identify edges, while higher layers may identify more sophisticated concepts, such as digits, letters, or faces. The first major business application of deep learning was to check processing in the early 2000s. The modern deep learning revolution builds on connectionism—an approach in cognitive science that seeks to explain mental phenomena using artificial neural networks. Connectionism took its rise from the work of Warren McCulloch, Walter Pitts, Donald Olding Hebb, and Karl Lashley. Modern neural networks can be thought of as generalizations of the "perceptron" introduced by Frank Rosenblatt in 1957. In this chapter, we explore the foundations of deep learning and its applications to finance and investing.

Background: Deciphering the Human Brain

Neuron architecture, in various degrees, forms the basis of deep learning algorithms. The detailed study of neurons commenced in the early 1900s, when anatomists began using microscopes and new staining methods to study the microscopic parts of the brain. It was around this time that neuroanatomists Santiago Ramón y Cajal and Camillo Golgi discovered "that nerve cells (neurons) are the building blocks of the brain and showing there are many different types" of neurons (Jones 1999).

Neuroscience progressed significantly through discoveries about how neurons interact. Researchers eventually identified the synapse as the point of connection where nerve cells communicate, leading to major insights into the workings of the central nervous system. Later work revealed that neurons transmit signals through both electrical impulses and chemical processes. The understanding of how neural activity strengthens connections between cells introduced the concept often summarized as "neurons that fire together become more strongly linked," forming the basis for associative or Hebbian learning, where repeated activation strengthens connections between neurons involved in the same process.

Around the same period, Alan Turing developed the idea of a mechanical model of computation, now known as the Turing machine. His work provided a mathematical framework for defining what it means for a task or function to be computationally solvable. This led to the principle

 •

that any process considered effectively computable can be represented by a Turing machine, forming a cornerstone of modern computer science. In 1943, McCulloch and Pitts published "A Logical Calculus of the Ideas Immanent to Nervous Activity," which described a mathematical model of the nervous system as a network of simple logical elements, known as *artificial neurons*, or later as *McCulloch–Pitts neurons*. These neurons take inputs, calculate a weighted sum, and produce an output signal based on a threshold function.

In 1957, Frank Rosenblatt at the Cornell Aeronautical Laboratory simulated a simple artificial neuron called a *perceptron* on an IBM 704. Later, he obtained funding from the Information Systems Branch of the United States Office of Naval Research and the Rome Air Development Center to build a custom-made computer, the Mark I Perceptron. It was first publicly demonstrated on 23 June 1960. The machine was part of a previously secret four-year NPIC (US National Photographic Interpretation Center) project that ran from 1963 through 1966, with the goal of developing the Mark I into a useful tool for photo-interpreters. Indeed, the Mark I was a fairly powerful pattern learning and recognition device for its time and was able to reliably learn to classify visual patterns into groups on the basis of certain geometric similarities and differences, utilizing properties such as position in the retinal field of view, geometric form, occurrence frequency, and size.

Perceptrons and feed forward neural networks (FFNNs) feed information from the front to the back (respectively, input and output). A common characteristic of FFNNs is that in them, two adjacent layers are "fully connected," which means that every neuron from one layer is connected to every neuron from another layer. FFNNs are typically trained through backpropagation, giving the network paired datasets of "what goes in" and "what we want to have coming out." This is called supervised learning, as opposed to unsupervised learning, where we only give it input and let the network fill in the blanks. The error being backpropagated is often some variation of the difference between the input and the output (such as mean squared error, or MSE) or just the linear difference. Given that the network has enough hidden neurons, it can theoretically always model the relationship between the input and output. Practically, their use is a lot more limited, but they are popularly combined with other networks to form new networks. In **Exhibit 1**, we show a perceptron, and in **Exhibit 2**, we show an FFFN. (Note that all exhibits in this chapter were created by the authors).

Extreme learning machines (ELMs) (Huang 2015) are similar to FFNNs but have random connections. They have many similarities to liquid state machines and echo state networks but are neither recurrent nor spiking and do not use backpropagation. Instead of backpropagation, ELMs start with random weights and train the weights in a single step according to the least-squares fit (lowest error across all functions). This results in a considerably less expressive network but one that is also significantly faster than backpropagation.

Deep residual networks (DRNs) (He, Zhang, Ren, and Sun 2016) are very deep FFNNs with additional connections passing input from one layer to one or more further layers.

Exhibit 1. Perceptron Architecture

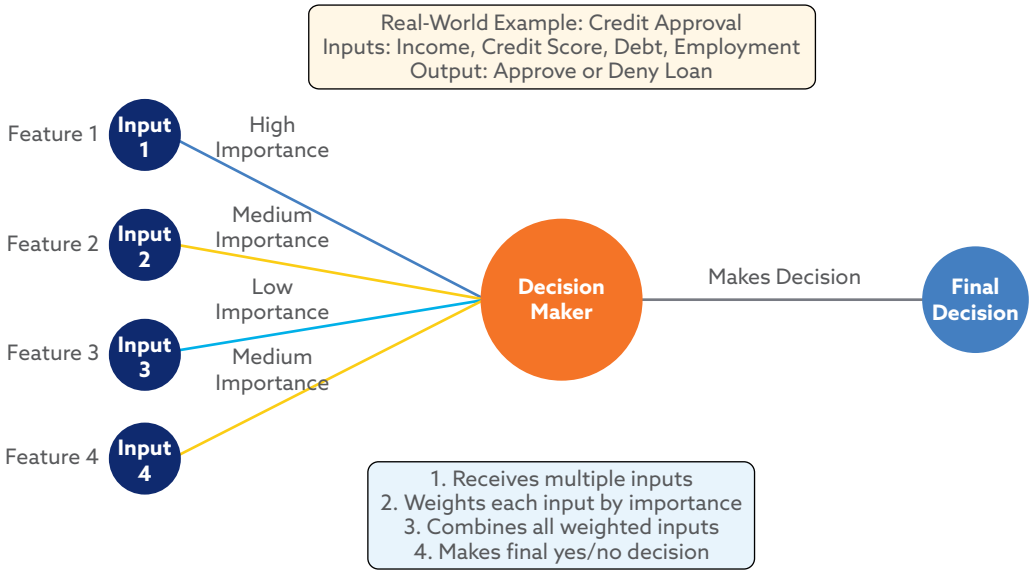
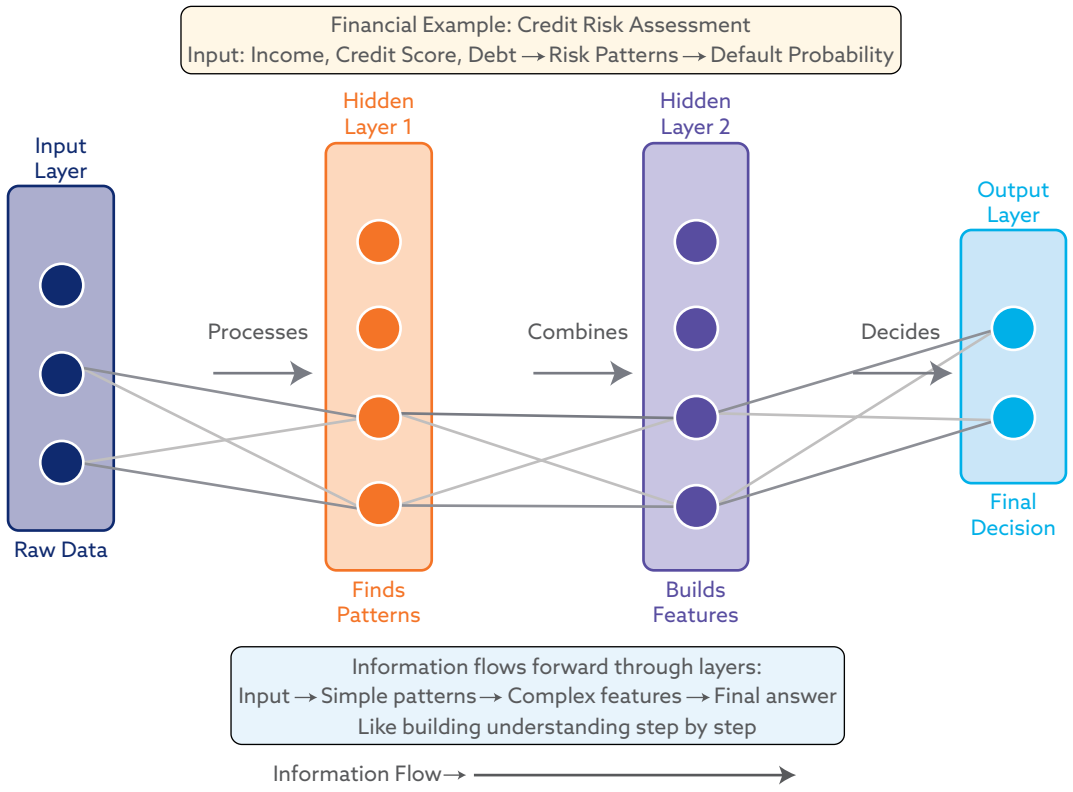


Exhibit 2. Feed Forward Neural Network Architecture



Now, let us implement a perceptron in Python. Going through the steps manually will give us a good idea about how neural networks solve problems. First, we generate some data:

```
X = []
for x1 in [0., 1.]:
for x2 in [0., 1.]:
X. append ([ x1 , x2 ])
y = []
for x in X:
y. append ( x[0] and x [1])
```

It is more convenient to work with NumPy arrays than native Python lists, so we convert accordingly:

```
X = np. array ( X) y = np. array ( y)
```

We initialize the weights and bias to (pseudo)random values sampled from the standard normal distribution:

```
weights = np. random . normal( size =(2 , 1)) bias = np. random . normal ()
```

The function "predict" will predict *y* given *X* as well as fixed weights and bias:

```
def predict(X, weights, bias):
    y_pred = []
    for x in X:
        x = x.reshape(-1, 1)
        v = weights.T @ x + bias
        y_pred.append(0. if v < 0. else 1.)
    return np.array(y_pred)
```

The function fit updates the weights and bias using gradient descent with γ set to the learning_rate:

```
def fit(X, y, weights, bias, learning_rate=.01, epochs=1):
    for i in range(epochs):
        for x, target in zip(X, y):
            x = x.reshape(-1, 1)
            v = weights.T @ x + bias
            y_pred = 0. if v < 0. else 1.
            if target != y_pred:
                bias -= learning_rate * (y_pred - target)
                weights -= learning_rate * (y_pred - target) * x
    return weights, bias
```

The epoch parameter represents the number of complete cycles (*epochs*) through the entire training dataset and indicates the number of passes that the machine learning algorithm must complete during that training. We proceed as follows:

```
weights, bias = fit(X, y, weights, bias, epochs=100)
y_pred = predict(X, weights, bias)
y_pred
# array([0., 0., 0., 1.])

y
# array([0., 0., 0., 1.])

weights, bias
# (array([[0.12671415],
#       [0.0117357]]),

# -0.13231146189930793)
```

These weights and bias have been found by the gradient descent algorithm. Our *procedural* code is a bit haphazard. It would be cleaner to use the *object-oriented* approach and encapsulate the notion of a perceptron in a dedicated class:

```
class Perceptron(object):
  def __init__(self, dim):
    self.dim = dim
    self.weights = np.random.normal(size=(self.dim, 1))
    self.bias = np.random.normal()
```

```
def fit(self, X, y, learning_rate=.01, epochs=1):
    for i in range(epochs):
        for x, target in zip(X, y):
            x = x.reshape(-1, 1)
            v = self.weights.T @ x + self.bias
            y_pred = 0. if v < 0. else 1.
            if target != y_pred:
                self.bias -= learning_rate * (y_pred - target)
                self.weights -= learning_rate * (y_pred - target) * x


def predict(self, X):
    y_pred = []
    for x in X:
        x = x.reshape(-1, 1)
        v = self.weights.T @ x + self.bias
        y_pred.append(0. if v < 0. else 1.)
    return np.array(y_pred)
```

The code we have provided is for pedagogical purposes; as such, a class already exists in *scikit-learn*, the popular free software machine learning library for Python. Scikit-learn grew out of a June 2007 Google Summer of Code project by David Cournapeau and now features various classification, regression, and clustering algorithms, including support vector machines, random forests, gradient boosting, *k*-means, and DB-SCAN. Scikit-learn is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Although individual perceptrons turned out to be of limited practical use, networks of perceptrons (or feed forward neural networks, FFNNs) were soon recognized as powerful universal function approximators. Their calibration in practice was impeded by computational restrictions, which were overcome algorithmically using the backpropagation algorithm (Rumelhart, Hinton, and Williams 1986), a major computational advance, and improvements in hardware, such as the emergence of GPUs. The progress was not uniform, and this academic area

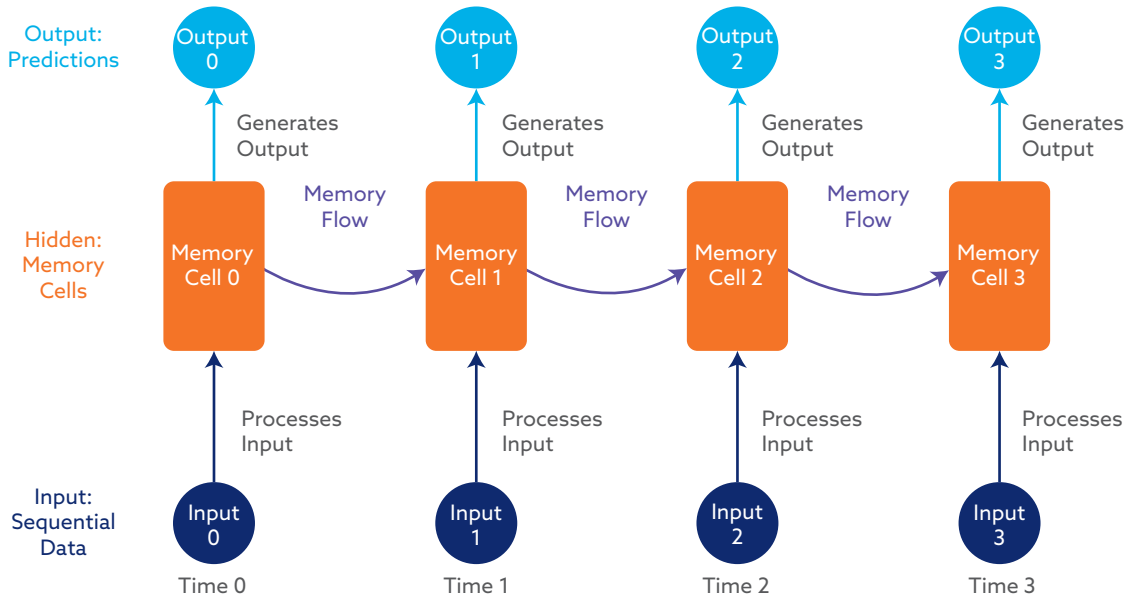
went through several periods of funding cuts, which are sometimes referred to as *AI winters* (Harguess and Ward 2022).

More Sophisticated Deep Learning Frameworks

In this section, we describe more sophisticated deep learning frameworks, several of which have become popular in finance. We begin with *recurrent neural networks* (RNNs) (Rumelhart et al. 1986), which are FFNNs that are not stateless—rather, they have connections between passes, connections through time. They are popular in financial applications. Neurons take input information not only from previous layers but also from themselves on previous passes. Thus, the order in which the input is fed into and trained in the network matters. One major challenge with RNNs is the vanishing gradient problem, where, depending on the activation functions used, information rapidly gets lost over time. This is similar to how some FFNNs lose information in depth. Nevertheless, RNNs are a good choice for many time series applications. We show an RNN in **Exhibit 3**.

Echo state networks (Jaeger and Haas 2004) are another type of (recurrent) network. They set themselves apart by having random connections between neurons (i.e., they are not organized into neat sets of layers). Instead of feeding input and backpropagating the error, they feed the input, update the neurons, and observe the output over time. The input and output layers have a somewhat unconventional role as the input layer is used to prepare the network and the output layer acts as an observer of the activation patterns that develop over time. During the training period, only the connections between the observer and the hidden units are changed.

Exhibit 3. Recurrent Neural Network Architecture Through Time

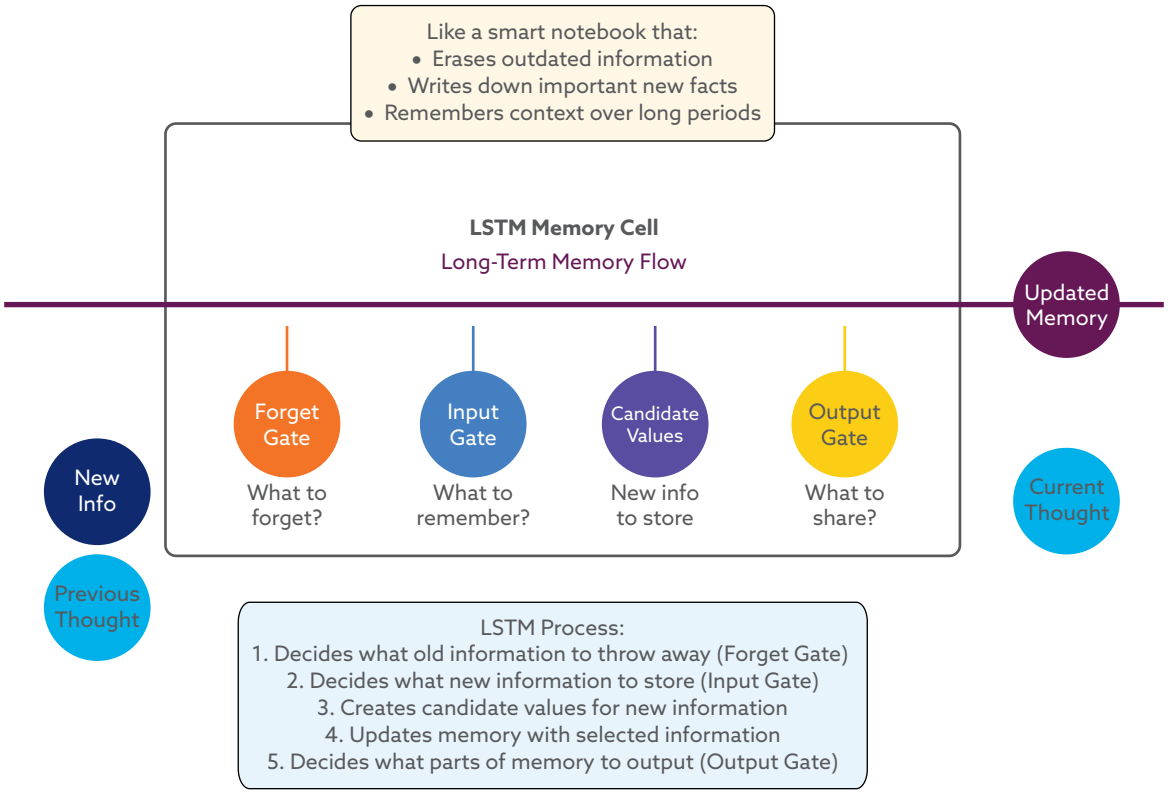


Key Concept: Memory cells remember information from previous time steps
This allows the network to understand sequences and context

Long short-term memory (LSTM) (Hochreiter and Schmidhuber 1997) networks try to combat the vanishing/exploding gradient problem by introducing gates and an explicitly defined memory cell. These are inspired mostly by circuitry, not so much biology. Each neuron has a memory cell and three gates: input, output, and forget. The function of these gates is to safeguard the information by stopping or allowing the flow of it. The input gate determines how much of the information from the previous layer is stored in the cell. The output layer takes the job on the other end and determines how much of the next layer gets to know about the state of this cell. The forget gate, as the name suggests, enables the network to forget. LSTMs have been shown to be able to learn complex sequences, such as writing prose or composing music. We show an LSTM in **Exhibit 4**.

Neural Turing machines (NTMs) (Graves, Wayne, and Danihelka 2014) can be understood as an abstraction of LSTMs and an attempt to undo the black-box nature of neural networks (and provide insight into what is going on in there). NTMs augment the traditional neural network architecture with an external memory bank, allowing it to perform tasks that require both computation and flexible memory manipulation, such as copying, sorting, and associative recall. The architecture consists of three main components: a controller, which processes inputs and determines how to interact with memory; a memory matrix, which serves as the external storage for information; and read/write heads, which focus attention on specific memory locations for reading or writing data.

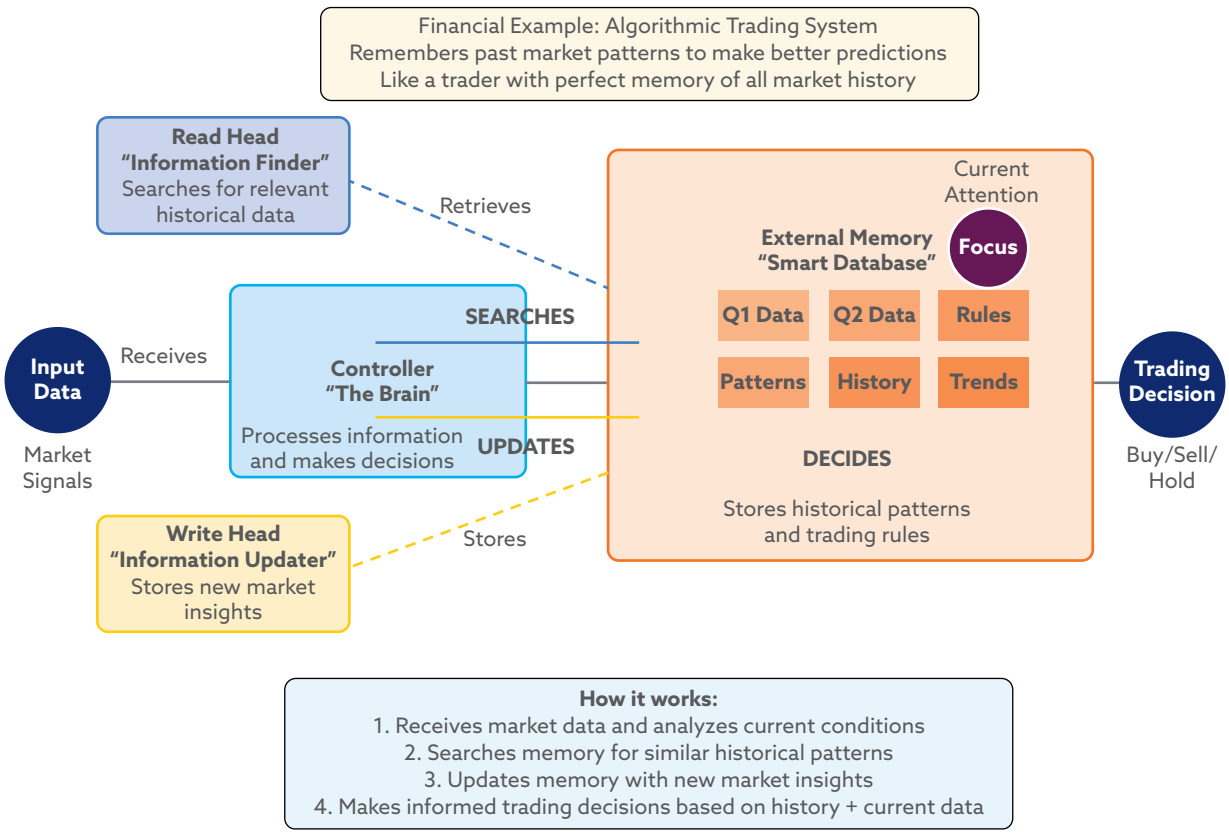
Exhibit 4. LSTM Architecture



The controller, often implemented as a recurrent neural network, receives both the current input and the previous memory readout, enabling it to make informed decisions about memory access. The read and write heads use attention mechanisms to determine how much focus to place on each memory location, allowing the NTM to interact with memory in a smooth, differentiable way. This differentiability means the entire system can be trained end to end using gradient descent, just like standard neural networks. The result is a model that combines the pattern recognition strengths of neural networks with the algorithmic flexibility of a Turing machine, making NTMs particularly suited for tasks that require reasoning over sequences and manipulating stored data in complex ways. *Differentiable neural computers* (Graves, Wayne, Reynolds, Harley, Danihelka, Grabska-Barwińska, Colmenarejo, Grefenstette, and Ramalho 2016) are enhanced neural Turing machines with scalable memory, inspired by how memories are stored by the human hippocampus. We show the architecture of an NTM in **Exhibit 5**.

Gated recurrent units (GRUs) (Cho, van Merrienboer, Bahdanau, and Bengio 2014) are a variation on LSTMs. They contain one less gate and are wired slightly differently: instead of an input, output, and forget gate, they have an update gate. The update gate determines both how much information to keep from the last state and how much information to let in from the previous layer. The reset gate functions much like the forget gate of an LSTM but is located at different points in the decision-making process. In most cases, they function similarly to LSTMs but are slightly faster and easier to run (albeit also slightly less expressive).

Exhibit 5. Neural Turing Machine Architecture



Bidirectional recurrent neural networks and *bidirectional long short-term memory networks* (BNs) (Schuster and Paliwal 1997) look identical to their unidirectional counterparts. The main difference between them is that BNs are not just connected to the past but also connected to the future. This means that during training, the network fills in gaps instead of simply advancing information. For example, instead of advancing an image on the edge, it could fill a hole in the middle of an image.

Autoencoders (AEs) represent a different use of FFNNs rather than a fundamentally different architecture. In autoencoders, we compress information. In AEs, the entire network resembles an hourglass, having smaller hidden layers relative to the input and output layers. AEs can be trained using backpropagation by feeding input and setting the error to be the difference between the input and what came out (Hinton and Salakhutdinov 2006). *Variational autoencoders* (VAEs) have the same architecture as AEs but are "taught" an approximated probability distribution of the input samples data (Kingma and Welling 2014). *Denoising autoencoders* are AEs where we feed in the input data with noise. The output of the network is compared with the original input without the noise, which encourages the network to learn broader features instead of details (Vincent, Larochelle, Bengio, and Manzagol 2008).

With *sparse autoencoders* (SAEs) (Makhzani and Frey 2013) we encode information in more space. So instead of the network converging in the middle and then expanding back to the input size, the middle of the network is the zone of expansion. SAEs are useful in extracting small features from a dataset. Instead of simply feeding back the input as in some other networks, we feed back the input with the addition of a sparsity driver. This sparsity driver is often a "threshold filter," where only a certain error is passed back and trained; other errors will be "irrelevant" for that pass and set to zero. This is somewhat similar to spiking neural networks, where not all neurons fire all the time. Among the various types of encoders, VAEs in particular have become popular in finance because of their utility in anomaly detection and generating synthetic data. We show VAE architecture in **Exhibit 6**.

Generative adversarial networks (GANs) (Goodfellow, Pouget-Abadie, Mirza, Xu, Warde-Farley, Ozair, Courville, and Bengio 2014) are a class of generative models that use a game-theoretic framework to learn and generate new data that mimics the distribution of a given dataset. GANs consist of two neural network twins: the *generator* and the *discriminator*. The generator creates synthetic data from random noise, attempting to mimic the real data distribution. The discriminator distinguishes between real data (from the dataset) and fake data (produced by the generator). The discriminator receives either training data or generated content from the generator. Information regarding how well the discriminator is able to correctly predict the data source is then used as part of the error for the generating network. This process in essence creates a competitive game in which the discriminator gets better at distinguishing real data from generated data and the generator learns to become less predictable to the discriminator. GANs have become popular in finance as a means to generate synthetic data. We show the GAN architecture in **Exhibit 7**.

Liquid state machines (Maass, Natschläger, and Markram 2002) are a type of spiking neural network that replace the usual sigmoid activation functions with discrete threshold mechanisms, where each neuron also maintains an internal state or accumulated potential. Instead of overwriting the neuron's current value with the weighted sum of its neighbors, the input is incrementally added to the neuron's stored energy. When this accumulated value surpasses a defined threshold, the neuron emits a spike, transferring energy to connected units.

Exhibit 6. Variational Autoencoder Architecture

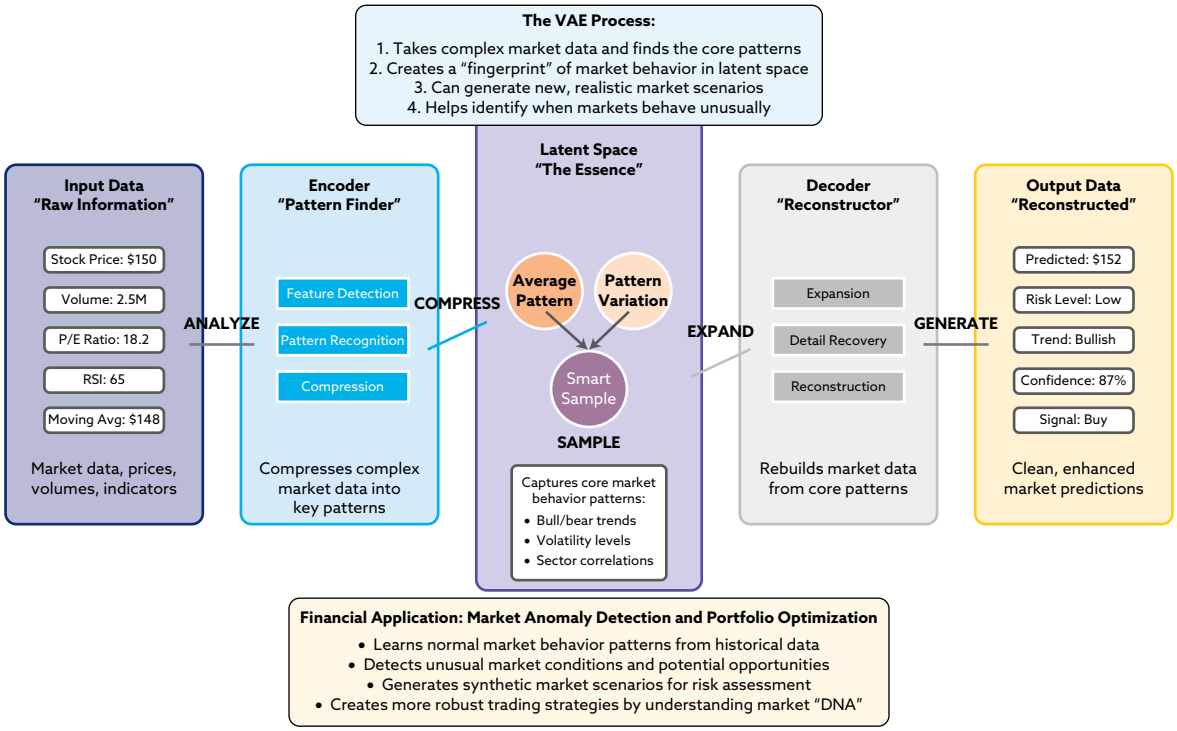
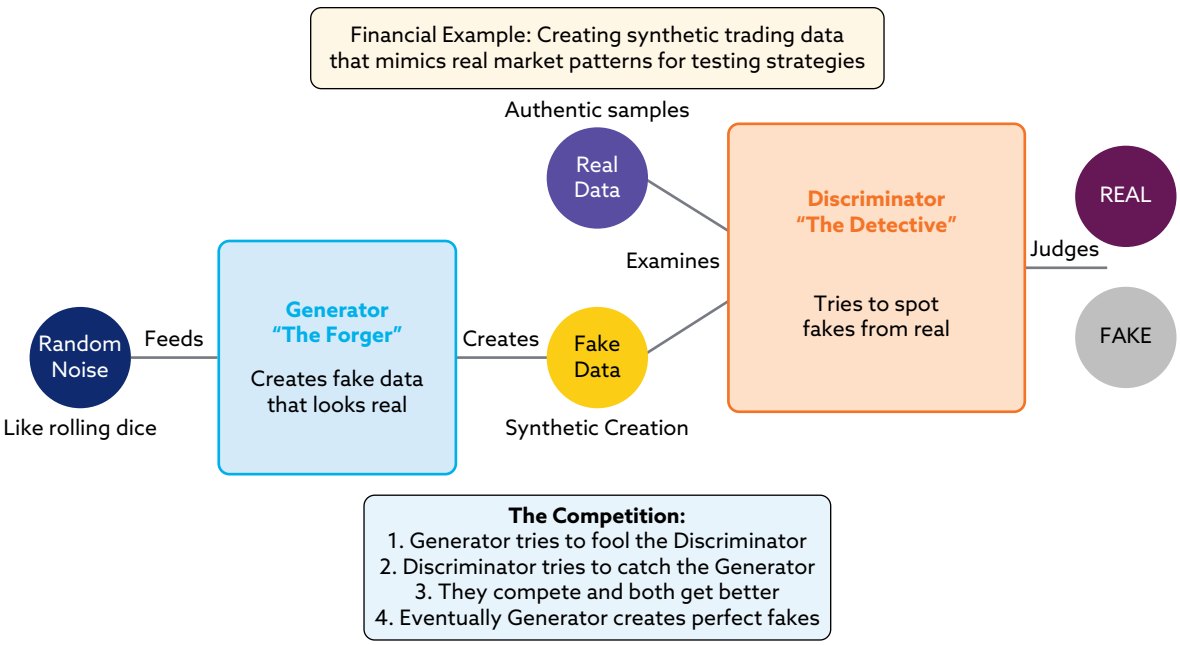


Exhibit 7. GAN Architecture



This produces a characteristic firing pattern with long periods of inactivity punctuated by sudden bursts of activity, behavior that is typical of spiking behaviors. A *Hopfield network* (HN) is a network in which each neuron is connected to every other neuron. Every node is input before training, hidden during training, and output afterward. These networks are trained by setting the neurons' value to the desired pattern; then, the weights can be computed. These networks are often called *associative memory* because they converge to the most similar state as the input. *Boltzmann machines* (BMs) are similar to HNs, except that some neurons are marked as input neurons and others remain "hidden." The algorithm begins by assigning weights randomly and learns through backpropagation, or *contrastive divergence*, where a Markov chain is used to determine the gradients between two informational gains. At the end of a full network update, the input neurons become output neurons. In contrast to HNs, in BMs, the neurons mostly have binary activation patterns.

Convolutional neural networks (CNNs) (LeCun, Bottou, Bengio, and Haffner 1998) or *deep convolutional neural networks* (DCNNs) are different from most other deep learning algorithms. They are primarily used for image processing but can also be applied to other types of data such as audio. A typical use case for CNNs is where you input network images and the network classifies the data—for example, "cat" versus "dog." CNNs tend to start with an input "scanner" that is not intended to parse all training data at once. The input data are then processed through convolutional layers, where not all nodes are connected to all nodes. Each node only concerns itself with neighboring cells in close proximity (how close varies by application). Convolutional layers also tend to shrink as they deepen.

Aside from these convolutional layers, CNNs also frequently feature pooling layers. Pooling is a way to filter out details. One commonly used pooling technique is max pooling, where we take, for example, three pixels and pass on the pixel with the most amount of red. Real-world implementations of CNNs often attach an FFNN to the end of the algorithm to further process the data, a maneuver that allows for highly nonlinear abstractions.

Deconvolutional networks (DNs), also known as *inverse graphics networks* (IGNs), are reversed convolutional neural networks. For example, consider the case where we feed a network the word "dog" (or a binary classification input vector) and train it to produce dog-like pictures by comparing what it generates to real pictures of dogs. DNNs can also be combined with FFNNs just like regular CNNs can. When this is done, the pooling layers often found in CNNs are frequently replaced with analogous inverse operations, primarily interpolation and extrapolation with biased assumptions.

Finally, *capsule networks* (CapsNet) (Sabour, Frosst, and Hinton 2017) are biologically inspired alternatives to pooling, where neurons are connected with a vector of weights instead of just one weight (a scalar). *Kohonen networks* (Kohonen 1990), on the other hand, use competitive learning to classify data without supervision. In the next section, we describe some specific applications of deep learning to finance.

Applications in Finance

Derivatives pricing was one of the early targets of applied neural networks in finance. Early adopters of neural networks in option pricing include Malliaris and Salchenberger (1993); Hutchinson, Lo, and Poggio (1994); Yao, Li, and Tan (2000); Bennell and Sutcliffe (2004); and Gradojevic, Gencay, and Kukolj (2009). With the advent of deep learning

(Goodfellow 2016), neural networks started to become mainstream, and deep learning reentered quants' collective consciousness, particularly following publication of work on *deep learning volatility* (Ferguson and Green 2018; Horvath, Muguruza, and Tomas 2021), which soon became mainstream.

In these articles, the authors presented neural network-based calibration methods that perform the calibration task within a few milliseconds for the full implied surface. These frameworks are applicable across a range of volatility models—including second-generation stochastic volatility models and the rough volatility family—and a range of derivative contracts. Neural networks are being used in offline approximations of complex pricing functions, which are difficult to represent or time consuming to evaluate by other means.

In some instances, finance has generated algorithmic advances, such as *differential deep learning* (Huge and Savine 2020). It combines automatic adjoint differentiation (Capriotti and Giles 2024) with machine learning, where the models are trained on examples of not only inputs and labels but also differentials of labels with regard to inputs, yielding highly effective pricing and risk approximations. More recently, these approaches have been applied in a wider range of settings, such as stochastic volatility (Sridi and Bilokon 2023), including exotic products (Ma, Ventre, Tiranti, and Chen 2025).

Outside the context of derivatives pricing, deep methods have been applied for *alpha generation*. Kolm, Turiel, and Westray (2023) deployed deep learning to forecast high-frequency returns at multiple horizons for 115 stocks traded on Nasdaq using order book information at the most granular level. State-of-the-art predictive accuracy was achieved by running "off-the-shelf" artificial neural networks on stationary inputs derived from the order book. Using cross-sectional regressions, the authors linked an LSTM network's forecasting performance to stock characteristics at the market microstructure level, suggesting "information-rich" equities can be predicted more accurately. The effective horizon of stock-specific forecasts was found to be approximately two average price changes.

Deep econometrics (Bilokon 2025) is a principled rethinking of the classical econometric (Ruud 2000) and time-series (Tsay 2010) analyses using deep learning techniques (Goodfellow 2016; Dixon, Halperin, and Bilokon 2020). The focus of Bilokon (2025) is on the estimation of parameters in various econometric settings. Some applications focus on the rethinking of the Wiener–Kolmogorov filtering theory, the so-called deep stochastic filters (Horvath, Kratsios, Limmer, and Yang 2023; Stok, Bilokon, and Simonian 2024).

Some interesting applications have arisen out of the combination of reinforcement deep learning and reinforcement learning, a framework where agents learn through a system of rewards and punishments, based on their actions in specific states. Reinforcement learning (Sutton 2020) differs from supervised learning in that the ground truth may not necessarily be known. Feedback is often evaluative rather than prescriptive, is often delayed, and may be sourced from the environment. In recent years, this subfield of machine learning/artificial intelligence gained public recognition when a reinforcement-learning-based system beat the human champion at the game of Go (Silver, Huang, Maddison, Guez, Sifre, van den Driessche, Schrittwieser, et al. 2016). Soon after, reinforcement learning began to gain popularity in finance.

Early adopters started to use deep reinforcement learning for hedging derivative contracts, giving rise to *deep hedging* (Halperin 2017; Buehler, Gonon, Teichmann, and Wood 2019; Kolm and Ritter 2019a; Cao, Chen, Hull, and Poulos 2021). Reinforcement learning in this

application was used to derive optimal hedging strategies for derivatives in cases where transaction costs and other frictions are present.

Far from being a novelty, many of these algorithms have been extensively studied and evaluated (see, e.g., Stoiljkovic 2025). Financial applications have led to a cross-pollination of ideas, which has contributed new and enhanced reinforcement learning techniques, such as enhancements to inverse reinforcement learning (Halperin, Liu, and Zhang 2022) and distributional reinforcement learning (Halperin 2024). Other researchers focused on applications of reinforcement learning to wealth management (e.g., Dixon, Gvozdanovic, and O'Kane 2023). This gave rise to G-Learner (Dixon and Halperin 2020), a reinforcement learning algorithm that uses explicitly defined one-step rewards, does not assume a data generation process, and is appropriate for use with noisy data. GIRL (Dixon and Halperin 2020) applies goal-based G-learning to inverse reinforcement learning (IRL) (Dixon et al. 2020), where rewards collected by the agent are not observed but inferred.

Others have combined ideas from the emerging subfields of reinforcement learning, such as multiarmed bandits, to update the now classic Markowitz–Sharpe framework (Varlashova and Bilokon 2025; Bilokon and Varlashova 2025). This framework arose from the rethinking of some of the issues relevant to finance, such as nonstationarity, in novel and nontrivial ways. Needless to say, the extensive work on the uses of machine learning for time-series forecasting (Dixon, Klabjan, and Bang 2017; Stok et al. 2024) is the foundation of many trading applications. Jaddu and Bilokon (2024) combined deep learning on the order books with reinforcement learning and backtested the resulting strategies in the presence of frictions. Zejnullahu, Moser, and Osterrieder (2022) explored in considerable detail the use of double deep Q-networks for trading purposes. Pendharkar and Cusatis (2018) explored applications of reinforcement learning agents to trading financial indexes. We point out that financial applications of reinforcement learning have been extensively reviewed by Kolm and Ritter (2019b); Charpentier, Élie, and Remlinger (2023); and Hambly, Xu, and Yang (2023).

Other advances in machine learning have been applied to create synthetic financial data, which are particularly useful in small data environments (Buehler, Horvath, Lyons, Arribas, and Wood 2020; Bühler, Horvath, Lyons, Arribas, and Wood 2020). Some research has focused on speeding up the calculations on a wider range of devices, such as field-programmable gate arrays (Sobakinskikh and Bilokon 2025), rather than algorithmic advances. There has also been cross-disciplinary work, which is difficult to classify, at the boundaries of finance, machine learning, and physics (Halperin and Dixon 2020). Progress has occurred in one of the most controversial areas of applications of machine learning and artificial intelligence to finance—explainability (Bussmann, Giudici, Marinelli, and Papenbrock 2021). The rise of large language models, such as ChatGPT (OpenAI, Achiam, Adler, Agarwal, Ahmad, Akkaya, Aleman, et al. 2023) and Claude are likely to further revolutionize finance.

Concluding Thoughts

The application of deep learning to finance has evolved from early neural network experiments in derivative pricing to sophisticated deep learning systems that now permeate virtually every aspect of financial markets. Financial applications of deep learning have increasingly focused on practical implementation challenges. Early research often ignored market microstructure effects, transaction costs, and regulatory constraints. Contemporary work in deep hedging,

order flow prediction, and portfolio optimization explicitly incorporates these real-world frictions, making the resulting strategies more robust and implementable.

Several developments promise to further revolutionize finance. The emergence of large language models creates opportunities for natural language processing of financial documents, automated report generation, and sophisticated conversational interfaces for financial analysis. Quantum computing, although still in its infancy, may eventually enable the solution of optimization problems that are currently intractable. Meanwhile, regulatory developments around algorithmic transparency and explainable AI will likely shape how these technologies are deployed in practice.

The cross-pollination between finance and deep learning has benefited both fields. Finance has provided challenging real-world problems that have spurred methodological advances in such areas as differential machine learning and distributional reinforcement learning. Conversely, techniques developed in computer science have enabled financial practitioners to tackle previously unsolvable problems in risk management, trading, and asset allocation.

As we stand at this inflection point, with deep learning capabilities advancing at an unprecedented pace, the integration of artificial intelligence into financial markets appears not merely inevitable but already well underway. The question is no longer whether deep learning (and AI as a whole) will transform finance but, rather, how quickly and in what specific directions this transformation will proceed.

References

Bennell, Julia, and Charles Sutcliffe. 2004. "Black–Scholes versus Artificial Neural Networks in Pricing FTSE 100 Options." *Intelligent Systems in Accounting, Finance & Management* 12 (4): 243–60. doi:10.1002/isaf.254.

Bilokon, Paul. 2025. "Deep Econometrics." Working paper (9 June). doi:10.2139/ssrn.5286898.

Bilokon, Paul, and Valeria Varlashova. 2025. "Tail-Aware Portfolio Optimization Using Hoeffding-Informed Thresholds." Working paper (22 May). doi:10.2139/ssrn.5265442.

Buehler, Hans, Lukas Gonon, Josef Teichmann, and Ben Wood. 2019. "Deep Hedging." *Quantitative Finance* 19 (8): 1271–91. doi:10.1080/14697688.2019.1571683.

Buehler, Hans, Blanka Horvath, Terry Lyons, Imanol Perez Arribas, and Ben Wood. 2020. "Generating Financial Markets with Signatures." Working paper (21 July). doi:10.2139/ssrn.3657366.

Bühler, Hans, Blanka Horvath, Terry Lyons, Imanol Perez Arribas, and Ben Wood. 2020. "A Data-Driven Market Simulator for Small Data Environments." Working paper (21 June). doi:10.48550/arXiv.2006.14498.

Bussmann, Niklas, Paolo Giudici, Dimitri Marinelli, and Jochen Papenbrock. 2021. "Explainable Machine Learning in Credit Risk Management." *Computational Economics* 57 (1): 203–16. doi:10.1007/s10614-020-10042-0.

Cao, Jay, Jacky Chen, John Hull, and Zissis Poulos. 2021. "Deep Hedging of Derivatives Using Reinforcement Learning." *Journal of Financial Data Science* 3 (1): 10–27. doi:10.3905/jfds.2020.1.052.

Capriotti, Luca, and Mike Giles. 2024. "15 Years of Adjoint Algorithmic Differentiation (AAD) in Finance." *Quantitative Finance* 24 (9): 1353–79. doi:10.1080/14697688.2024.2325158.

Charpentier, Arthur, Romuald Élie, and Carl Remlinger. 2023. "Reinforcement Learning in Economics and Finance." *Computational Economics* 62 (1): 425–62. doi:10.1007/s10614-021-10119-4.

Cho, K., B. van Merrienboer, D. Bahdanau, and Y. Bengio. 2014. "On the Properties of Neural Machine Translation: Encoder–Decoder Approaches." *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*: 103–11.

Dixon, Matthew Francis, Ivan Gvozdanovic, and Dominic O'Kane. 2023. "Time Consistent Reinforcement Learning for Optimal Consumption under Epstein–Zin Preferences. Working paper (14 March). doi:10.2139/ssrn.4388762.

Dixon, Matthew, and Igor Halperin. 2020. "G-Learner and GIRL: Goal Based Wealth Management with Reinforcement Learning." Working paper (25 February). doi:10.48550/arXiv.2002.10990.

Dixon, Matthew F., Igor Halperin, and Paul Bilokon. 2020. *Machine Learning in Finance: From Theory to Practice*. Cham, Switzerland: Springer.

Dixon, Matthew, Diego Klabjan, and Jin Hoon Bang. 2017. "Classification-Based Financial Markets Prediction Using Deep Neural Networks." *Algorithmic Finance* 6 (3–4): 67–77. doi:10.3233/AF-170176.

Ferguson, Ryan, and Andrew Green. 2018. "Deeply Learning Derivatives." Working paper (17 October). doi:10.48550/arXiv.1809.02233.

Goodfellow, Ian J., Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. "Generative Adversarial Nets." *Proceedings of the 27th International Conference on Neural Information Processing Systems* 2: 2672–80.

Goodfellow, Ian. 2016. *Deep Learning: Adaptive Computation and Machine Learning*. Cambridge, MA: MIT Press.

Gradojevic, Nikola, Ramazan Gencay, and Dragan Kukolj. 2009. "Option Pricing with Modular Neural Networks." *IEEE Transactions on Neural Networks* 20 (4): 626–37. doi:10.1109/TNN.2008.2011130.

Graves, A., G. Wayne, and I. Danihelka. 2014. "Neural Turing Machines." arXiv (10 December). doi:10.48550/arXiv.1410.5401.

Graves, A., G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, and T. Ramalho. 2016. "Hybrid Computing Using a Neural Network with Dynamic External Memory." *Nature* 538 (7626): 471–76.

Halperin, Igor. 2017. "QLBS: Q-Learner in the Black–Scholes (–Merton) Worlds." Working paper (17 December). doi:10.48550/arXiv.1712.04609.

Halperin, Igor. 2024. "Distributional Offline Continuous-Time Reinforcement Learning with Neural Physics-Informed PDEs (SciPhy RL for DOCTR-L)." *Neural Computing & Applications* 36 (9): 4643–59. doi:10.1007/s00521-023-09300-7.

Halperin, Igor, and Matthew Dixon. 2020. "'Quantum Equilibrium–Disequilibrium': Asset Price Dynamics, Symmetry Breaking, and Defaults as Dissipative Instantons." *Physica A* 537 (1 January). doi:10.1016/j.physa.2019.122187.

Halperin, Igor, Jiayu Liu, and Xiao Zhang. 2022. "Combining Reinforcement Learning and Inverse Reinforcement Learning for Asset Allocation Recommendations." Working paper (6 January). doi:10.48550/arXiv.2201.01874.

Hambly, Ben, Renyuan Xu, and Huining Yang. 2023. "Recent Advances in Reinforcement Learning in Finance." *Mathematical Finance* 33 (3): 437–503. doi:10.1111/mafi.12382.

Harguess, Josh, and Chris M. Ward. 2022. "Is the Next Winter Coming for AI? Elements of Making Secure and Robust AI." In *2022 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*, 1–7. doi:10.1109/AIPR57179.2022.10092230.

He, K., X. Zhang, S. Ren, and J. Sun. 2016. "Deep Residual Learning for Image Recognition." In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–78.

Hinton, Geoffrey E., and Ruslan Salakhutdinov. 2006. "Reducing the Dimensionality of Data with Neural Networks." *Science* 313 (5786): 504–07. doi:10.1126/science.1127647.

Hochreiter, S., and J. Schmidhuber. 1997. "Long Short-Term Memory." *Neural Computation* 9 (8): 1735–80.

Horvath, Blanka, Anastasis Kratsios, Yannick Limmer, and Xuwei Yang. 2023. "Deep Kalman Filters Can Filter." Working paper (27 October). doi:10.13140/RG.2.2.22953.57445.

Horvath, Blanka, Aitor Muguruza, and Mehdi Tomas. 2021. "Deep Learning Volatility: A Deep Neural Network Perspective on Pricing and Calibration in (Rough) Volatility Models." *Quantitative Finance* 21 (1): 11–27. doi:10.1080/14697688.2020.1817974.

Huang, Guang-Bin. 2015. "What are Extreme Learning Machines? Filling the Gap between Frank Rosenblatt's Dream and John von Neumann's Puzzle." *Cognitive Computation* 7 (3): 263–78.

Huge, Brian, and Antoine Savine. 2020. "Differential Machine Learning." Working paper (30 September). doi:10.48550/arXiv.2005.02347.

Hutchinson, James M., Andrew W. Lo, and Tomaso Poggio. 1994. "A Nonparametric Approach to Pricing and Hedging Derivative Securities via Learning Networks." *Journal of Finance* 49 (3): 851–89. doi:10.1111/j.1540-6261.1994.tb00081.x.

Jaddu, Koti S., and Paul A. Bilokon. 2024. "Deep Learning with Reinforcement Learning on Order Books." *Journal of Financial Data Science* 6 (1): 61–84. doi:10.3905/jfds.2024.1.149.

Jaeger, H., and H. Haas. 2004. "Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication." *Science* 304 (5667): 78–80.

Jones, Edward G. 1999. "Golgi, Cajal and the Neuron Doctrine." *Journal of the History of the Neurosciences* 8 (2): 170–78. doi:10.1076/jhin.8.2.170.1838.

Kingma, D. P., and M. Welling. 2014. "Auto-Encoding Variational Bayes." arXiv (1 May). doi:10.48550/arXiv.1312.6114.

Kohonen, T. 1990. "The Self-Organizing Map." *Proceedings of the IEEE* 78 (9): 1464–80.

Kolm, Petter N., and Gordon Ritter. 2019a. "Dynamic Replication and Hedging: A Reinforcement Learning Approach." *Journal of Financial Data Science* 1 (1): 159–71. doi:10.3905/jfds.2019.1.1.159.

Kolm, Petter N., and Gordon Ritter. 2019b. "Modern Perspectives on Reinforcement Learning in Finance." Working paper (6 September). doi:10.2139/ssrn.3449401.

Kolm, Petter N., Jeremy Turiel, and Nicholas Westray. 2023. "Deep Order Flow Imbalance: Extracting Alpha at Multiple Horizons from the Limit Order Book." *Mathematical Finance* 33 (4): 1044–81. doi:10.1111/mafi.12413.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner. 1998. "Gradient-Based Learning Applied to Document Recognition." *Proceedings of the IEEE* 86 (11): 2278–324.

Ma, Yanqing, Carmine Ventre, Renzo Tiranti, and Aiming Chen. 2025. "Deep Generative Calibration on Stochastic Volatility Models with Applications in FX Barrier Options." In *SAC '25: Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing*, 122–30.

Maass, W., T. Natschläger, and H. Markram. 2002. "Real-Time Computing without Stable States: A New Framework for Neural Computation Based on Perturbations." *Neural Computation* 14 (11): 2531–60.

Makhzani, A., and B. J. Frey. 2013. "k-Sparse Autoencoders." arXiv (19 December). doi:10.48550/arXiv.1312.5663.

Malliaris, Mary, and Linda Salchenberger. 1993. "A Neural Network Model for Estimating Option Prices." *Applied Intelligence* 3 (3): 193–206. doi:10.1007/BF00871937.

McCulloch, Warren, and Walter Pitts. 1943. "A Logical Calculus of the Ideas Immanent to Nervous Activity." *Bulletin of Mathematical Biophysics* 5 (4): 115–33.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, et al. 2023. "GPT-4 Technical Report." Working paper (15 March). doi:10.48550/arXiv.2303.08774.

Pendharkar, Parag C., and Patrick Cusatis. 2018. "Trading Financial Indices with Reinforcement Learning Agents." *Expert Systems with Applications* 103 (August): 1–13. doi:10.1016/j.eswa.2018.02.032.

Rosenblatt, F. 1957. "The Perceptron—A Perceiving and Recognizing Automaton." *Cornell Aeronautical Laboratory* 85 (460-1).

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. 1986. "Learning Representations by Back-Propagating Errors." *Nature* 323 (6088): 533–36. doi:10.1038/323533a0.

Ruud, Paul Arthur. 2000. *An Introduction to Classical Econometric Theory*. New York: Oxford University Press.

Sabour, S., N. Frosst, and G. E. Hinton. 2017. "Dynamic Routing between Capsules." arXiv (26 October). doi:10.48550/arXiv.1710.09829.

Schuster, M., and K. K. Paliwal. 1997. "Bidirectional Recurrent Neural Networks." *IEEE Transactions* on *Signal Processing* 45 (11): 2673–81.

Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, et al. 2016. "Mastering the Game of Go with Deep Neural Networks and Tree Search." *Nature* 529 (7587): 484–89. doi:10.1038/nature16961.

Sobakinskikh, Ilia, and Paul Alexander Bilokon. 2025. "Optimizing Transformer Neural Network for Real-Time Outlier Detection on FPGAs." *Journal of FinTech* 5 (1). doi:10.1142/S2705109925500014.

Sridi, Abir, and Paul Bilokon. 2023. "Applying Deep Learning to Calibrate Stochastic Volatility Models." Working paper (25 September). doi:10.48550/arXiv.2309.07843.

Stoiljkovic, Zoran. 2025. "Advanced Option Pricing and Hedging with Q-Learning: Performance Evaluation of the QLBS Algorithm." *Journal of Derivatives* 32 (3): 48–79. doi:10.3905/jod.2025.1.222.

Stok, Robert, Paul Bilokon, and Joseph Simonian. 2024. "From Deep Learning to Deep Econometrics." *Journal of Financial Data Science* 6 (2): 54–73. doi:10.3905/jfds.2024.1.155.

Sutton, Richard S. 2020. *Reinforcement Learning: Adaptive Computation and Machine Learning*, 2nd ed. Cambridge, MA: MIT Press.

Tsay, Ruey S. 2010. *Analysis of Financial Time Series*, 3rd ed. Hoboken, NJ: Wiley.

Varlashova, Valeria, and Paul Alexander Bilokon. 2025. "Optimal Allocation with Continuous Sharpe Ratio Covariance Bandits." *Journal of Financial Data Science* 7 (3): 171–91. doi:10.3905/jfds.2025.1.191.

Vincent, P., H. Larochelle, Y. Bengio, and P.-A. Manzagol. 2008. "Extracting and Composing Robust Features with Denoising Autoencoders." In *ICML '08: Proceedings of the 25th International Conference on Machine Learning*, 1096–103.

Yao, Jingtao, Yili Li, and Chew Lim Tan. 2000. "Option Price Forecasting Using Neural Networks." *Omega* 28 (4): 455–66. doi:10.1016/S0305-0483(99)00066-3.

Zejnullahu, Frensi, Maurice Moser, and Joerg Osterrieder. 2022. "Applications of Reinforcement Learning in Finance—Trading with a Double Deep Q-Network." Working paper (28 June). doi:10.48550/arXiv.2206.14267.