Portfolio Optimization

Deep Learning Portfolios

Daniel P. Palomar (2025). *Portfolio Optimization: Theory and Application.*
Cambridge University Press.

portfoliooptimizationbook.com

Latest update: March 14, 2025

Outline

Abstract

Artificial intelligence (AI) broadly refers to the ability of computers to emulate human thought and perform tasks in real-world environments, with machine learning (ML) being a subset that enables systems to identify patterns, make decisions, and improve through experience and data. Neural networks (NN), which emulate the structure of human brain neurons, are powerful tools for automatic learning from examples, contrasting with traditional pre-programmed computer tasks. Deep neural networks, or deep learning (DL), have revolutionized many domains, achieving superhuman performance in image recognition, natural language processing, board and video games, protein folding, and close-to-human performance in video processing and self-driving cars. The potential for DL to transform financial systems is significant, with successful applications in sentiment analysis, credit default detection, and satellite image analysis. These slides explore the application of DL in portfolio design, aiming to create a system that processes financial data to generate optimal portfolios, a field still in development with ongoing research and emerging open-source software (Palomar 2025, chap. 16).

Outline

Machine learning (ML)

- **Traditional approach:**
  - First model the data.
  - Then design a portfolio based on the model.
- **Machine learning (ML) approach:**
  - More direct and algorithmic.
  - Attempts to learn a “black-box” model of reality (Breiman 2001).
  - Textbooks: (Hastie, Tibshirani, and Friedman 2009; James et al. 2013; Shalev-Shwartz and Ben-David 2014).
- **Fundamental paradigms of machine learning:**
  - **Supervised learning:** Learn a function that maps an input to an output based on example input-output pairs.
  - **Reinforcement learning:** Learn a mapping function without explicit input-output examples, based on a reward function that evaluates the performance.
  - **Unsupervised learning:** Learn relationships and structure from data (e.g., clustering or graphs), without supervised output (labels) or reward function.

Black-box modeling

- **Historical context of statistical learning:**
  - Introduction of least squares in the early nineteenth century, known as *linear regression*.
  - By the end of the 1970s, many techniques for learning from data were available, mostly linear methods due to computational constraints.
  - Mid-1980s: Introduction of classification and regression trees, practical implementations of *nonlinear methods*.
  - Emergence of machine learning as a subfield in statistics, focusing on supervised and unsupervised modeling and prediction.
  - Recent progress marked by powerful and user-friendly software.
- **Basic idea of machine learning (ML):**
  - Model and learn the input-output relationship or mapping of a system.
  - Input or p *features* denoted as $\boldsymbol{x} = (x_1, \dots, x_p)$.
  - Output denoted as y.
  - Noisy relationship modeled as:

    $$y = f(\boldsymbol{x}) + \epsilon,$$

    where f is the unknown function to be learned, and ϵ denotes random noise.
  - "Black box" modeling: No attempt to understand the mapping f, only to learn it.

Black-box modeling

- **Purpose of estimating or learning the function f:**
  - **Prediction (forecast if in future time):** Given input $\boldsymbol{x}$, predict output y as:
$$\hat{y} = f(\boldsymbol{x}).$$

- **Types of ML systems** based on the nature of the output y:
  - **Regression:** Output is a real-valued number.
  - **Classification:** Output takes discrete values, such as $\{0, 1\}$ or $\{\text{cat}, \text{dog}\}$.

Measuring performance

- **Learning the function f:**
  - Evaluate the goodness of a candidate model using an error function (loss or cost function).
  - Typical choices for error functions:
    - **Mean squared error (MSE) for regression:**

$$\mathrm{MSE} \triangleq \mathbb{E}\left[(y-\hat{y})^2\right] = \mathbb{E}\left[(y-f(\boldsymbol{x}))^2\right];$$

    - **Accuracy (ACC) for classification:** (many others like precision, recall, sensitivity, specificity, etc.)

$$\mathrm{ACC} \triangleq \mathbb{E}\left[I\left(y=\hat{y}\right)\right] = \mathbb{E}\left[I\left(y=f(\boldsymbol{x})\right)\right],$$

      where $I\left(y=\hat{y}\right)$ is the indicator function that equals 1 if $y=\hat{y}$ (correct classification) and 0 otherwise (classification error).
- **Expectation operator $\mathbb{E}\left[\cdot\right]$:**
  - Over the distribution of random input-output pairs $(\boldsymbol{x}, y)$.
  - In practice, approximated via the sample mean over observations.

Measuring performance

- **Supervised learning:**
  - Estimation or learning of the function f based on n observations or *training data*:

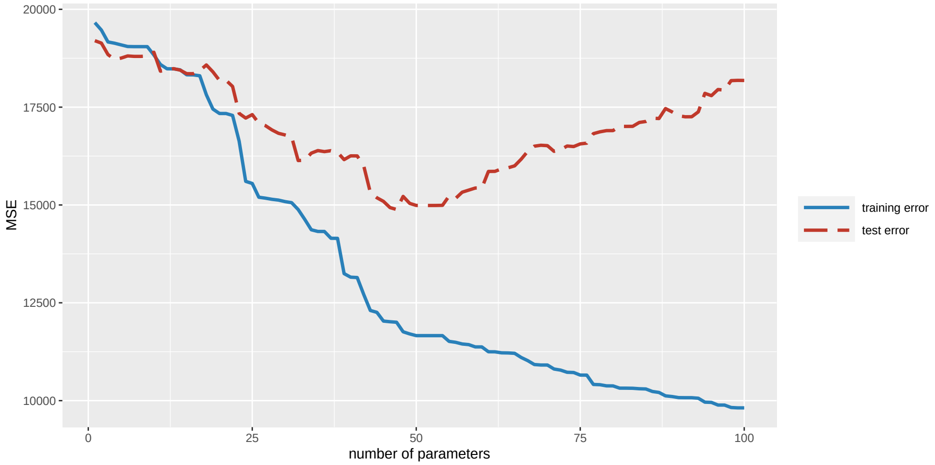
$$\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_n, y_n)\}.$$

  - Performance or error measure computed using training data.
  - Training error may not represent the true error due to *overfitting*.

- **Overfitting:**
  - Occurs when the model fits the noise in the training data, not representative of the test data.
  - Proper evaluation requires test data (new data not used during learning).
  - Test performance (e.g., test MSE or test ACC) is representative of true performance.
  - Divergence between training MSE and test MSE indicates overfitting.

Measuring performance

Overfitting: training error and test error:

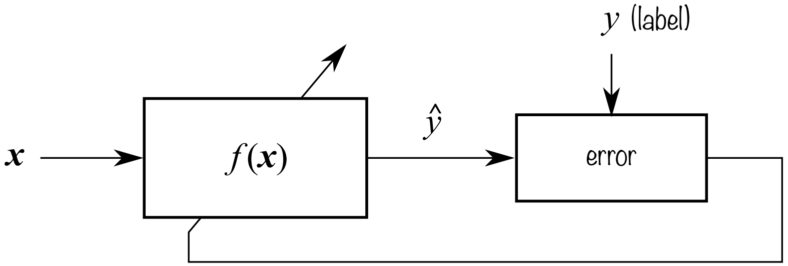


Learning the model f

- **Learning the black-box model f:**
  - Based on training data by minimizing training error or optimizing a performance measure.
  - Specific mechanism depends on the particular black-box model.
  - Example: In artificial neural networks, training algorithms are variations of stochastic gradient descent.
- **Supervised learning:**
  - Learn the function f based on input-output pairs (with labels).
  - Minimize the error (e.g., MSE or ACC).
- **Reinforcement learning:**
  - Used when explicit labels are not available but model performance can be measured.

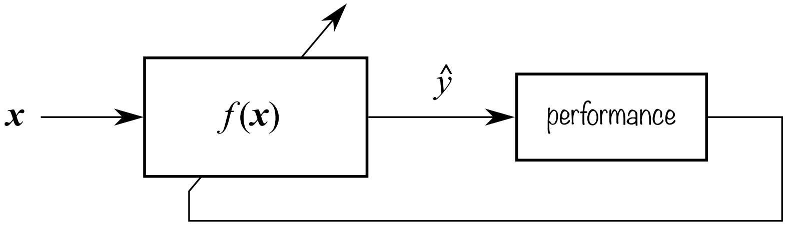
Learning the model f

Supervised learning in ML via error minimization:



Learning the model f

Reinforcement learning in ML via performance optimization:



Learning the model f

- **Overfitting:**
  - Overfitting occurs when the model fits the noise in the training data, not representative of the rest of the data.
  - Happens with too little training data or when the number of parameters (degrees of freedom) characterizing f is too large.
- **Avoiding overfitting:**
  - **Empirical cross-validation methods:**
    - Assess performance of a learned model f (estimated from training data) on new data termed *cross-validation data*.
    - Final performance assessed on yet new data termed *test data*.
  - **Statistical penalty methods:**
    - Avoid reserving data for cross-validation.
    - Rely on mathematically-derived penalty terms on the degrees of freedom.
    - Examples: Bayesian information criterion (BIC), minimum description length (MDL), Akaike information criterion (AIC).

Types of ML models

- **Constraining the function f:**
  - Cannot handle a totally arbitrary function f in the whole space of possible functions.
  - Constrain the search to some class of functions f.
  - Use finite-dimensional parameters, denoted by $\boldsymbol{\theta}$, to characterize f.
  - Example: Linear models have the form $f(\boldsymbol{x}) = \alpha + \boldsymbol{\beta}^\mathsf{T}\boldsymbol{x}$ with parameters $\boldsymbol{\theta} = (\alpha, \boldsymbol{\beta})$.
  - Nonlinear models can have more complicated structures but still with a finite number of parameters.
- **Diversity of statistical learning approaches:**
  - Since the 1970s, many classes of functions have been proposed.
  - "No free lunch theorem" in statistics: No single method dominates all others over all possible data sets.
  - Different methods may work best on different data sets.
  - Selecting the best method for a given data set is a challenging task.

Types of ML models

- **Successful ML models** (Vapnik 1999; Bishop 2006; Hastie, Tibshirani, and Friedman 2009; Shalev-Shwartz and Ben-David 2014):
  - Linear models
  - Sparse linear models
  - Decision trees
  - K-nearest neighbors (KNN)
  - Bagging
  - Boosting
  - Random forests (boosting applied to decision trees)
  - Support vector machines (SVM)
  - Neural networks (foundation of deep learning)
- **Universal function approximators:**
  - Some complex models, such as random forests and neural networks, are capable of approximating any nonlinear smooth function to any desired accuracy.
  - Require enough parameters to achieve the desired accuracy.

Applications of ML in finance

- **Applications of machine learning (ML) in finance:**
  - Time series forecasting
  - Portfolio design
  - Other applications (López de Prado 2019):
    - Credit risk
    - Sentiment analysis
    - Outlier detection
    - Asset pricing
    - Bet sizing
    - Feature importance
    - Order market execution
    - Big data analysis
- **References:**
  - Recent ML advances in finance (López de Prado 2018a).
  - Overview of reasons why most machine learning funds fail (López de Prado 2018b).
  - Overview of ML techniques for time series forecasting (Ahmed et al. 2010) (Bontempi, Taieb, and Borgne 2012).

Outline

Deep learning (DL)

- **Limitations of conventional machine learning:**
  - Limited ability to process raw data in black-box modeling $f(\cdot)$.
  - Required careful engineering and domain expertise to transform raw data into suitable feature vectors $\boldsymbol{x}$.
  - Features known as *handcrafted features*.
- **Representation learning:**
  - *Learned features* automatically obtained by deep architectures.
  - Process referred to as *representation learning*.
- **Deep learning:** (LeCun, Bengio, and Hinton 2015)
  - Methods and architectures that automatically learn features.
  - Uses multiple simple—but nonlinear—modules or layers.
  - Each layer transforms the representation at one level into a higher, more abstract level.
  - Layers of features are learned from data using a general-purpose learning procedure, not designed by human engineers.

Deep learning (DL)

- **Recognition of deep learning pioneers:**
  - Yoshua Bengio, Geoffrey Hinton, and Yann LeCun received the 2018 ACM A. M. Turing Award for breakthroughs in deep neural networks (LeCun, Bengio, and Hinton 2015).
- **Success of deep learning:**
  - Referred to as "the unreasonable effectiveness of deep learning."
  - Questions like "Why does deep and cheap learning work so well?" (Lin, Tegmark, and Rolnick 2017).
- **Resources on deep learning:**
  - Concise account: (LeCun, Bengio, and Hinton 2015)
  - Comprehensive textbook: (Goodfellow, Bengio, and Courville 2016)
  - Short online introductory book: (Nielsen 2015)

Historical snapshot

- **Historical ingredients of neural networks:**
  - 1676: Chain rule of differential calculus.
  - 1847: Gradient descent.
  - 1951: Stochastic gradient descent.
  - 1970: Backpropagation algorithm.
- **Pivotal historical moments in deep learning (DL):**
  - **1962:** Rosenblatt introduces the *multilayer perceptron*.
  - **1967:** Amari suggests training multilayer perceptrons with many layers via *stochastic gradient descent*.
  - **1970:** Linnainmaa publishes what is now known as *backpropagation* (reverse mode of automatic differentiation).
  - **1974 to 1980:** First major “AI winter” (reduced funding and interest in AI).
  - **1987 to 1993:** Second major “AI winter”.
  - **1997:** Introduction of “LSTM” networks (Hochreiter and Schmidhuber 1997).
  - **1998:** Establishment of “CNN” networks (LeCun et al. 1998).
  - **2010:** Start of “AI spring”.

Historical snapshot

- **Pivotal historical moments in deep learning (DL):**
  - **2012:** AlexNet achieves a top-5 error of 15.3% in the ImageNet 2012 Challenge (Krizhevsky, Sutskever, and Hinton 2012).
  - **2014:** Establishment and popularity of "GAN" networks.
  - **2015:** AlphaGo by DeepMind beats a professional Go player.
  - **2016:** Google Translate switches to a neural machine translation engine.
  - **2017:** AlphaZero by DeepMind achieves superhuman play in Chess, Shogi, and Go.
  - **2017:** Proposal of Transformer architecture based on the self-attention mechanism (Vaswani et al. 2017).
  - **2018:** GPT-1 (Generative Pre-Trained Transformer) for natural language processing with 117 million parameters.
  - **2019:** GPT-2 with 1.5 billion parameters.
  - **2020:** GPT-3 with 175 billion parameters.
  - **2022:** ChatGPT, a popular chatbot built on GPT-3, sparks discussions on AI safety.
  - **2023:** GPT-4 with approximately 1 trillion parameters (OpenAI 2023), showing potential signs of artificial general intelligence (AGI) (Bubeck et al. 2023).
- Detailed historical accounts: (Schmidhuber 2015, 2022).

Perceptron and sigmoid neuron

- **Perceptron:**
  - Maps input vector $\boldsymbol{x}$ to a binary output value.
  - Function definition:

$$f(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{w}^\top \boldsymbol{x} + b \geq 0 \\ 0 & \text{otherwise,} \end{cases}$$

  - Composition of affine function $\boldsymbol{w}^\top \boldsymbol{x} + b$ with the nonlinear binary step function (Heaviside function) $H(z)$.
  - Alternatively written with the indicator function $I(\cdot)$:

$$f(\boldsymbol{x}) = I(\boldsymbol{w}^\top \boldsymbol{x} + b \geq 0).$$

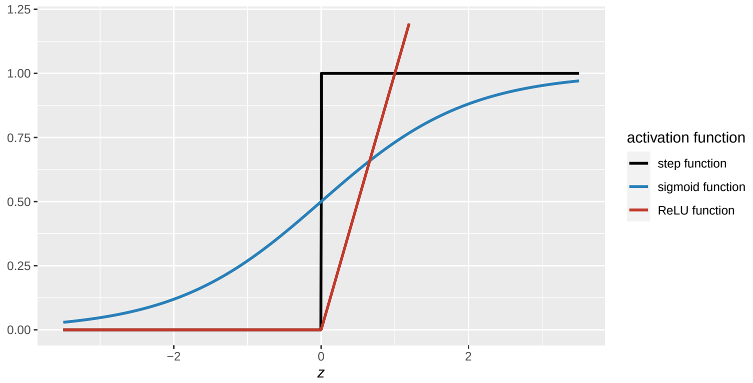
  - Weights $\boldsymbol{w}$ give different importance to inputs.
  - Bias b is equivalent to a nonzero activation threshold.
  - Minimal approximation to how a biological neuron works.
  - Developed in the 1950s and 1960s by Frank Rosenblatt, inspired by earlier work by Warren McCulloch and Walter Pitts.

Perceptron and sigmoid neuron

- **Sigmoid neurons:**
  - Similar to perceptrons but modified for small changes in weights and bias to cause small changes in output.
  - Crucial for allowing a network of sigmoid neurons to learn.
  - Function definition:
$$f(\boldsymbol{x}) = \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b),$$
  - Sigmoid function σ defined as:
$$\sigma(z) = \frac{1}{1 + e^{-z}}.$$
  - Behavior:
    - For large positive z, $e^{-z} \approx 0$ and $\sigma(z) \approx 1$.
    - For very negative z, $e^{-z} \to \infty$ and $\sigma(z) \approx 0$.
    - Resembles the behavior of the perceptron.
- **Nonlinear activation functions:**
  - **Heaviside function** $H(z)$ (step function): Used in perceptrons.
  - **Sigmoid function** $\sigma(z)$: Used in sigmoid neurons.
  - **ReLU function** $\mathrm{ReLU}(z) = \max(0, z)$: Popular activation function to be discussed later.

Perceptron and sigmoid neuron

Activation functions: step function (for the perceptron), sigmoid function (for the sigmoid neuron), and ReLU (popular in neural networks):

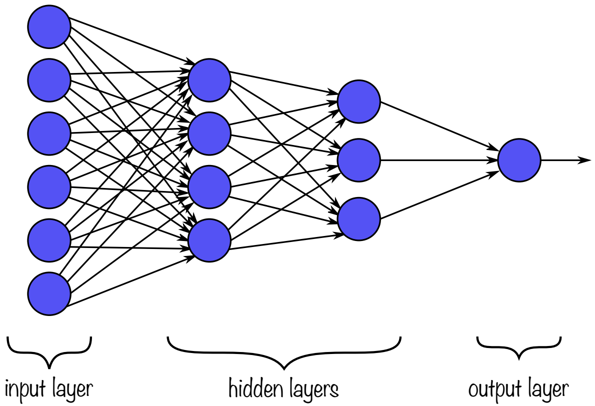


Neural networks (NN)

- **Multilayer perceptron (MLP):**
  - Combination of perceptrons in multiple layers.
  - Perceptrons in subsequent layers make decisions at more complex and abstract levels.
  - MLPs are universal function approximators (can approximate arbitrary functions).
- **Neural network:**
  - Several layers of neurons of any type (often referred to as MLPs).
  - **Layers:**
    - **Input layer:** Contains the input vector $\boldsymbol{x}$ (not an operational layer).
    - **Hidden layers:** Intermediate layers between input and output.
    - **Output layer:** Contains the output neurons.
  - Goal: Approximate some (possibly vector-valued) function $\boldsymbol{f}$.
  - **Feedforward neural networks:** Information flows from input $\boldsymbol{x}$ to output $\boldsymbol{y}$ without feedback connections.
  - **Recurrent neural networks:** Include feedback connections where outputs are fed back into the model.

Neural networks (NN)

Example of a multilayer perceptron with two hidden layers:



Neural networks (NN)

- **Deep neural network:**
  - Network with a large number of layers (depth).
  - Also referred to as "deep feedforward neural network."
- **Mathematical representation:**
  - Each layer i implements a vector function $\boldsymbol{f}^{(i)}$.
  - Connected chain of functions (composition of functions):

    $$\boldsymbol{f} = \boldsymbol{f}^{(n)} \circ \cdots \circ \boldsymbol{f}^{(2)} \circ \boldsymbol{f}^{(1)},$$

    where $\circ$ denotes composition of functions.
  - Each hidden layer i produces an intermediate vector $\boldsymbol{h}^{(i)}$ from the previous vector $\boldsymbol{h}^{(i-1)}$ (with $\boldsymbol{h}^{(0)} \triangleq \boldsymbol{x}$):

    $$\boldsymbol{h}^{(i)} = \boldsymbol{f}^{(i)}\left(\boldsymbol{h}^{(i-1)}\right) = \boldsymbol{g}^{(i)}\left(\boldsymbol{W}^{(i)}\boldsymbol{h}^{(i-1)} + \boldsymbol{b}^{(i)}\right),$$

    where $\boldsymbol{f}^{(i)}$ is the composition of an affine function with the elementwise nonlinear activation function $\boldsymbol{g}^{(i)}$.

Neural networks (NN)

- **Common elementwise nonlinear activation functions:**
  - **Sigmoid function:**
$$\sigma(z) = \frac{1}{1+e^{-z}};$$
  - **Hyperbolic tangent:**
$$\tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}};$$
  - **Rectified linear unit (ReLU) function:**
$$\mathrm{ReLU}(z) = \max(0, z),$$
    which typically learns much faster in networks with many layers.
- **Output layer:**
  - **Classification problems:** Employs the *softmax function*:
$$\mathrm{softmax}(\boldsymbol{z}) = \frac{e^{\boldsymbol{z}}}{\mathbf{1}^{\mathsf{T}} e^{\boldsymbol{z}}}.$$
  - **Regression problems:** Typically a simple affine mapping without activation function, i.e., $\boldsymbol{g}(\boldsymbol{z}) = \boldsymbol{z}$.

Learning via backpropagation

- **Supervised learning in deep learning:**
  - Developing a black-box model by training the system with data and minimizing an error function.
  - Adjusts specific parameters (weights) to determine the input-output function.
  - Deep learning systems may have hundreds of millions (or even billions) of adjustable weights and labeled examples for training.
- **Gradient method for learning:**
  - Adjusts the weight vector $\boldsymbol{w}$ by computing the gradient vector of the error function $\xi(\boldsymbol{w})$.
  - Gradient vector $\partial\xi/\partial\boldsymbol{w}$: how the error would change with small weight adjustments.
  - Weight vector adjustment formula:

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^{k} - \kappa\frac{\partial\xi}{\partial\boldsymbol{w}},$$

    where κ is the *learning rate*.
- **Error or cost function:**
  - Defined via a mathematical expectation over the distribution of possible input-output pairs.
  - In practice, the expectation operator is approximated using *stochastic gradient descent*.

Learning via backpropagation

- **Stochastic gradient descent (SGD):**
    - Involves presenting the input vector for several examples, computing outputs and errors, determining the average gradient, and adjusting weights.
    - Repeated for numerous small sets of examples from the training set until the average of the objective function ceases to decrease.
    - Called stochastic because each small set of examples provides a noisy estimate of the average gradient.
    - Typically identifies a good set of weights faster than more complex optimization methods.
- **Backpropagation:**
    - Practical implementation of the chain rule for derivatives.
    - Used to compute gradients in multilayer architectures.
    - Discovered independently by several groups during the 1970s and 1980s.
- **Historical context:**
    - In the late 1990s, neural nets and backpropagation were largely forsaken.
    - Thought that simple gradient descent would get trapped in poor local minima.
    - Recent theoretical and empirical results suggest that local minima are not a serious problem in general.

DL architectures:

Research in DL is extremely vibrant and new architectures are constantly being explored by practitioners and academics. In the following we describe some of the most relevant paradigms:

- Fully-connected neural networks
- Convolutional neural networks (CNNs)
- Recursive neural networks (RNNs), e.g., LSTM.
- Transformers
- Autoencoder networks
- Generative adversarial networks (GANs)
- Diffusion models

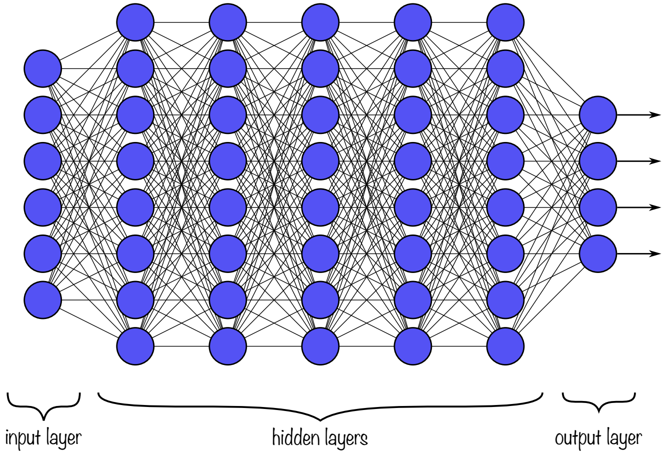
DL architectures: Fully-connected neural networks

- **Fully connected neural networks:**
    - Each neuron takes inputs from all outputs of the previous layer and combines them with weights.
    - Results in a rapid increase in the number of weights to be trained.
    - Illustrated in next figure with a simple MLP having five hidden layers.

- **Reducing the number of weights:**
    - Incorporate meaningful structure into the network tailored to the specific application.
    - Reduce the number of weights per layer.
    - Allows for many layers to express computationally large models.
    - Produces high levels of abstraction while keeping the number of actual parameters manageable.

DL architectures: Fully-connected neural networks

Large number of weights in a simple multilayer perceptron with five hidden layers:



DL architectures: Convolutional neural networks (CNNs)

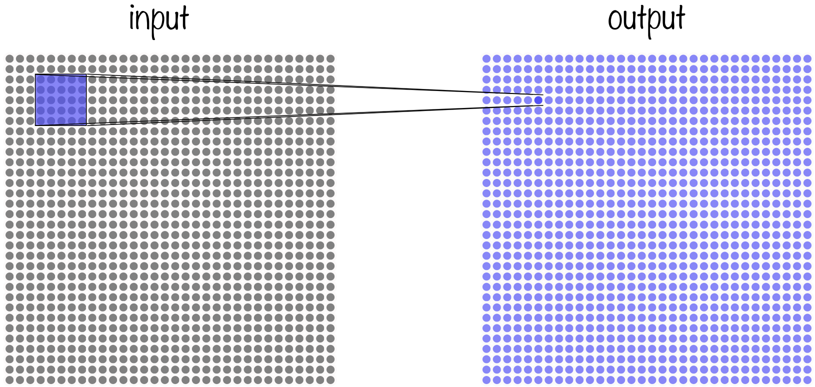
- **Convolutional neural networks (CNNs):**
  - Based on the concept of convolution from signal processing.
  - Achieved many practical successes, especially in computer vision.
  - Origins date back to the 1970s.
  - Seminal paper: 1998 “LeNet-5” architecture (LeCun et al. 1998) with 7 layers.
  - Major achievement: 2012 “AlexNet” architecture (Krizhevsky, Sutskever, and Hinton 2012) revolutionized image classification.
- **Concept of CNNs:**
  - Originated from image processing.
  - Input is a two-dimensional image.
  - Pixels are processed with their nearby pixels rather than distant ones (local connectivity).
  - Weights are shifted across the image, allowing sharing and reuse by all neurons in the hidden layer.
  - Introduces structure in the matrix $\boldsymbol{W}^{(i)}$ in the affine mapping $\boldsymbol{W}^{(i)}\boldsymbol{h}^{(i-1)} + \boldsymbol{b}^{(i)}$.
  - Matrix $\boldsymbol{W}^{(i)}$ is highly sparse with repeated nonzero elements.

DL architectures: Convolutional neural networks (CNNs)

- **Example of CNN efficiency:**
  - For a 100×100 image (10,000-dimensional input vector):
    - Fully connected approach: Requires 10^8 weights.
    - CNN approach: Requires coefficients of a 5×5 filter (25 weights) plus 1 for the bias.
  - CNNs use translation invariance to reduce the number of parameters.
  - Results in faster training and enables building deep networks with convolutional layers.
- **Reducing network complexity in CNNs:**
  - Use different stride lengths when shifting the filter.
  - Incorporate pooling layers (e.g., max-pooling) after convolutional layers.
  - Pooling layers condense feature maps and retain information about the presence of features without precise location details.
- **Graph CNNs:**
  - Extension of CNNs based on processing neighboring pixels.
  - Generalize the concept of neighborhood using a connectivity graph on the input elements.

DL architectures: Convolutional neural networks (CNNs)

CNN filtering layer:

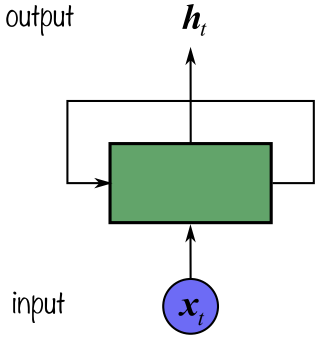


DL architectures: Recursive neural networks (RNNs)

- **Feedforward neural networks:**
  - Produce an output solely dependent on the current input.
  - Do not have internal memory.
- **Recursive neural networks (RNNs):**
  - Neural networks with loops, allowing information to persist (memory).
  - Can implement functions of all previous inputs $\boldsymbol{f}(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_t)$.
- **Advantages of RNNs:**
  - Potential to link past information to current tasks.
  - Effective when the gap between relevant information and its required location is small.
- **Challenges with RNNs:**
  - Struggle to learn long-term dependencies when the gap widens significantly.
  - Theoretically capable of managing long-term dependencies but often struggle in practice.
- **Long short-term memory (LSTM) networks:**
  - Introduced in 1997 (Hochreiter and Schmidhuber 1997).
  - Capable of learning long-term dependencies.
  - Refined and popularized in subsequent works.
  - Demonstrated success in memory-dependent tasks, e.g., natural language processing.

DL architectures: Recursive neural networks (RNNs)

RNN layer with a loop:



DL architectures: Transformers

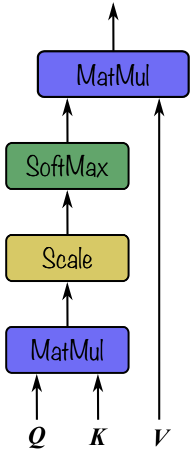
- **Transformer architecture:**
  - Introduced in 2017 (Vaswani et al. 2017).
  - Revolutionized natural language processing tasks.
  - Relies on self-attention mechanisms to process input sequences simultaneously.
  - Outperforms existing architectures in most applications.
- **Comparison with other architectures:**
  - **CNNs:**
    - Handle spatial data like images.
  - **RNNs:**
    - Process sequential data but struggle with vanishing and exploding gradient issues.
  - **Transformers:**
    - Overcome limitations of RNNs by employing self-attention.
    - Enable parallel processing, faster training, and improved handling of long-range dependencies.
    - Use position encoding to incorporate positional information.

DL architectures: Transformers

- **Relevance of transformers:**
  - Exceptional performance on a wide range of tasks such as machine translation, sentiment analysis, conversational AI, automated content generation, etc.
  - Basis of state-of-the-art models like GPT (OpenAI 2023).
- **Self-attention mechanism:**
  - Presents the network with all inputs at once, allowing it to decide which parts should influence others automatically.
  - For n inputs, each of dimension d, arranged in an $n \times d$ matrix $\boldsymbol{V}$, the goal is to replace each row of $\boldsymbol{V}$ with a linear weighted combination of all rows.
  - Weights are computed using "query" matrix $\boldsymbol{Q}$ and "key" matrix $\boldsymbol{K}$ by calculating the inner product of their rows to form a similarity matrix $\boldsymbol{Q}\boldsymbol{K}^\mathsf{T}$.
  - The similarity matrix is scaled by $\sqrt{d_k}$ and normalized using the softmax operator:
$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\mathsf{T}}{\sqrt{d_k}}\right)\boldsymbol{V}.$$
  - In practice, $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$ are linear transformations of the inputs, and multiple self-attention mechanisms are typically used in parallel.

DL architectures: Transformers

Self-attention mechanism (scaled dot-product attention):

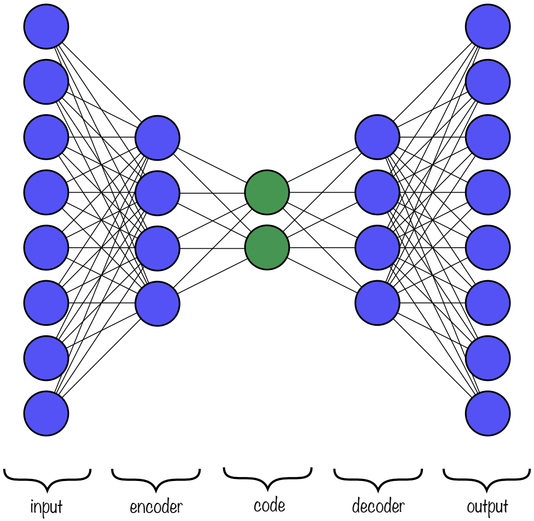


DL architectures: Autoencoder networks

- **Autoencoder networks:**
  - Used in deep learning for data representation, feature extraction, and dimensionality reduction (Kramer 1991).
  - Perform an unsupervised feature learning process.
- **Architecture:**
  - Consists of an input layer, one or more hidden layers, and an output layer.
  - Symmetrical structure divided into the encoder and decoder.
  - Input and output layers have the same number of nodes.
  - Central bottleneck called *code* or *latent features*.
- **Training process:**
  - Network is trained to make the output as close as possible to the input.
  - Forces the central bottleneck to condense information, performing feature extraction in an unsupervised manner.

DL architectures: Autoencoder networks

Autoencoder structure:



DL architectures: Generative adversarial networks (GANs)

- **Generative adversarial networks (GANs):**
  - Developed in 2014 (Goodfellow et al. 2014).
  - Consist of two adversarial neural networks: a generator and a discriminator.
  - Generator aims to produce realistic data (e.g., images, text).
  - Discriminator's role is to distinguish between real and fake data.
- **Training process:**
  - Generator and discriminator are trained simultaneously.
  - Generator creates synthetic data and presents it to the discriminator.
  - Discriminator evaluates whether the data is real or fake and provides feedback.
  - Generator modifies its output based on feedback to create more realistic data.
  - Process continues until the generator produces data indistinguishable from real data.
- **Applications of GANs:**
  - Image generation
  - Text-to-image synthesis
  - Generating realistic music
  - Generating artificial time series with asset prices for backtesting and stress testing (Takahashi, Chen, and Tanaka-Ishii 2019; Yoon, Jarrett, and Schaar 2019).

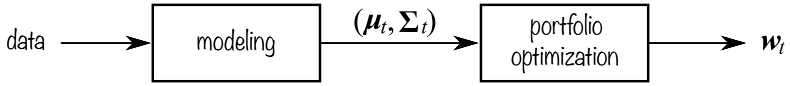
DL architectures: Diffusion models

- **Diffusion models:**
  - A type of generative model in deep learning.
  - Use a diffusion process to generate samples from a target distribution.
  - First introduced in 2015 (Sohl-Dickstein et al. 2015).
  - Gained popularity in the 2020s (Song and Ermon 2019; Ho, Jain, and Abbeel 2020).
- **Concept:**
  - Different from GANs, which use two adversarial networks (generator and discriminator).
  - Iteratively transform an initial noise signal using a series of neural networks.
  - Generate samples that resemble the target distribution.
  - At each iteration step, the model estimates the conditional distribution of the data given the current level of noise.
- **Comparison with GANs:**
  - Diffusion models are more stable during training compared to GANs.
  - Both are generative models that generate high-quality samples from complex distributions.
  - GANs are more flexible and can generate a wider variety of samples, including those not present in the training data.

Outline

DL for portfolio design

- Block diagram of data modeling and portfolio optimization:



- **Usage of Deep learning (DL) in portfolio design:**
  - **DL in modeling or time series forecasting:**
    - Use DL for data modeling while keeping traditional portfolio optimization.
  - **DL in portfolio component:**
    - Use DL for portfolio optimization while keeping traditional data modeling.
    - Not considered further as traditional optimization is well understood and efficient.
  - **End-to-end modeling:**
    - Use DL for both data modeling and portfolio optimization.

DL for portfolio design

- **Input data for DL systems:**
  - **Raw time series data:**
    - Price data (e.g., open, high, low, close).
    - Volume.
  - **Alternative data sources:**
    - Technical analysis.
    - Fundamental analysis.
    - Macroeconomic data.
    - Financial statements.
    - News.
    - Social media feeds.
    - Investor sentiment analysis.
  - **Time horizon and data frequency:**
    - High frequency data and intraday price movements.
    - Daily, weekly, or monthly stock prices.

Challenges of DL in finance

While deep neural networks excel in many domains, their application to financial systems remains uncertain. Key challenges:

- **Data scarcity**: Financial time series are extremely limited compared to other DL applications (e.g., two years of daily stock prices yield only 504 observations)
- **Low signal-to-noise ratio**: Financial signals are extremely weak and completely submerged in noise, unlike high signal-to-noise applications like image recognition
- **Data nonstationarity**: Financial distributions change over time (bull/bear markets), contrasting with applications where distributions remain constant

Challenges of DL in finance

- **Data adaptive feedback loop**: Once patterns are discovered and exploited, they tend to disappear due to market participants' reactions
- **Lack of prior human evidence**: Unlike other DL success areas, no human has demonstrated consistent ability to forecast financial markets, as Malkiel noted: "a blindfolded chimpanzee throwing darts at the stock listings can select a portfolio that performs as well as those managed by the experts"

Despite increasing research and applications since the 2010s in areas like algorithmic trading, portfolio management, and financial sentiment analysis, it's too early to determine if DL's success in other domains will fully extend to financial systems.

Challenges of DL in finance

Can you spot the cat? And the octopus? Financial data is more like an octopus.

Cat

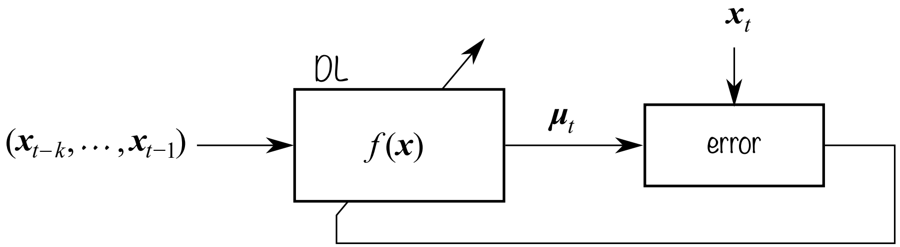
Octopus

Standard time series forecasting

- **Deep learning in portfolio design:**
  - Commonly used in time series modeling or forecasting (ignoring subsequent portfolio optimization component).
  - Intensively explored since 2015 (Sezer, Gudelek, and Ozbayoglu 2020).
- **LSTM in financial time series forecasting:**
  - Utilizes temporal characteristics due to inherent memory.
  - Initially dominated the domain (Fischer and Krauss 2018).
- **Transformers in financial time series forecasting:**
  - Shown to handle long-term memory more efficiently.
- **General process of time series forecasting:**
  - Input: lookback of past k time series values $(\boldsymbol{x}_{t-k}, \dots, \boldsymbol{x}_{t-1})$.
  - Desired output: next value of the time series $\boldsymbol{x}_t$.
  - Produced output (forecast): denoted by $\boldsymbol{\mu}_t$.
  - Error measure between $\boldsymbol{\mu}_t$ and $\boldsymbol{x}_t$ drives the learning process.
  - Forecast horizon can be further into the future, not just the next time index t.

Standard time series forecasting

Block diagram of standard time series forecasting via DL:



Standard time series forecasting

- **Error measures in learning process:**
  - **Regression setting:**
    - Forecast value is a number or vector of values.
    - Error vector: $\boldsymbol{e}_t = \boldsymbol{\mu}_t - \boldsymbol{x}_t$.
    - Quantities: mean square error (MSE), mean absolute error (MAE), median absolute deviation (MAD), mean absolute percentage error (MAPE), etc.
  - **Classification setting:**
    - Forecast is the trend (e.g., up/down).
    - Measures: accuracy, error rate, cross-entropy, etc.
    - Reference: (Goodfellow, Bengio, and Courville 2016).
- **Mathematical formulation of DL network:**
  - Implements function $\boldsymbol{f}_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{t-k}, \ldots, \boldsymbol{x}_{t-1}\right)$ with parameters $\boldsymbol{\theta}$.
  - Produces estimate of $\boldsymbol{x}_t$ as $\boldsymbol{\mu}_t = \boldsymbol{f}_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{t-k}, \ldots, \boldsymbol{x}_{t-1}\right)$.
  - Optimization problem:

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \mathbb{E}\left[\ell\left(\boldsymbol{f}_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{t-k}, \ldots, \boldsymbol{x}_{t-1}\right), \boldsymbol{x}_t\right)\right]$$

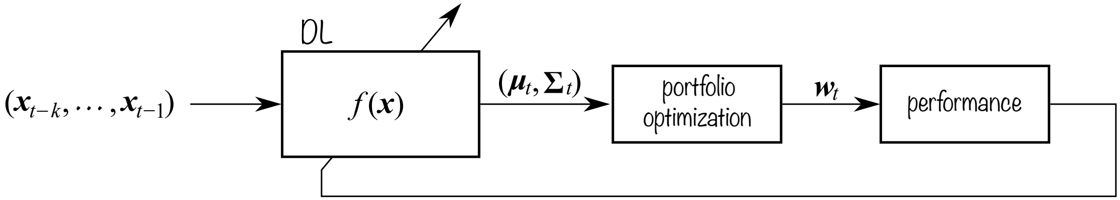
  - $\ell(\cdot, \cdot)$ denotes the prediction error function to be minimized (e.g., MSE or cross-entropy).

Portfolio-based time series forecasting

- **Standard time series modeling limitations:**
  - Ignores subsequent portfolio optimization component.
  - Performance measure defined in terms of error between forecast $\boldsymbol{\mu}_t$ and label $\boldsymbol{x}_t$.
  - Suitable error definition for portfolio optimization is unclear and heuristic.
- **Holistic approach:**
  - Incorporates portfolio optimization component to measure overall performance.
  - Avoids arbitrary error definitions.
- **Process of time series forecasting with portfolio optimization:**
  - Output $\boldsymbol{\mu}_t$ fed into portfolio optimization block to produce portfolio $\boldsymbol{w}_t$.
  - Performance measured meaningfully, e.g., Sharpe ratio.

Portfolio-based time series forecasting

Block diagram of portfolio-based time series forecasting via DL:



$(\boldsymbol{x}_{t-k}, \dots, \boldsymbol{x}_{t-1})$
DL
$f(\boldsymbol{x})$
$(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$
portfolio optimization
$\boldsymbol{w}_t$
performance

Portfolio-based time series forecasting

- **Mathematical formulation:**
  - DL network function to produce estimate: $\boldsymbol{\mu}_t = \boldsymbol{f}_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{t-k}, \dots, \boldsymbol{x}_{t-1}\right)$.
  - Portfolio $\boldsymbol{w}_t$ designed by minimizing objective function $f_0(\cdot)$.
  - Holistic optimization problem:

$$\begin{array}{ll} \underset{\boldsymbol{\theta}}{\text{minimize}} & \mathbb{E}\left[\xi\left(\boldsymbol{w}_t, \boldsymbol{x}_t\right)\right] \\ \text{subject to} & \boldsymbol{w}_t = \arg\min_{\boldsymbol{w}} f_0\left(\boldsymbol{w}; \boldsymbol{\mu}_t = \boldsymbol{f}_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{t-k}, \dots, \boldsymbol{x}_{t-1}\right)\right), \end{array}$$

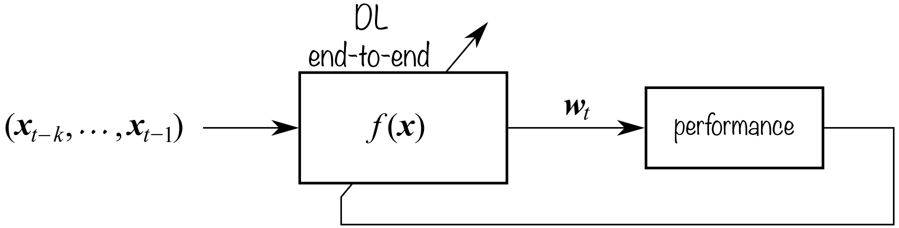
  - $\xi(\cdot, \cdot)$: error function measuring overall system performance (e.g., negative Sharpe ratio).
  - Parameters $\boldsymbol{\theta}$ optimized to minimize overall system performance.
- **Learning process challenges:**
  - Backpropagation requires partial derivatives of output with respect to input.
  - Closed-form expression for portfolio optimization block simplifies computation (e.g., $\boldsymbol{w}_t = \Sigma_t^{-1}\boldsymbol{\mu}_t$).
  - Alternative approach requires computing partial derivatives via optimality conditions.
  - Recent developments and open-source libraries for this (Amos and Kolter 2017).

End-to-end portfolio design

- **Standard DL time series modeling:**
    - Ignores subsequent portfolio optimization component and uses some heuristic error measure.
- **DL portfolio-based time series forecasting:**
    - Takes into account subsequent portfolio optimization block.
    - Measures overall performance using a meaningful performance measure.
- **End-to-end DL design:**
    - Makes full use of the black-box philosophy.
    - Models the whole process with a single DL component.

End-to-end portfolio design

Block diagram of end-to-end portfolio design via DL:



End-to-end portfolio design

- **Mathematical formulation of end-to-end DL portfolio:**
  - DL network function: $\boldsymbol{f}_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{t-k},\dots,\boldsymbol{x}_{t-1}\right)$.
  - Produces portfolio $\boldsymbol{w}_t$ directly.
  - Optimization problem:

$$\begin{array}{ll} \underset{\boldsymbol{\theta}}{\textsf{minimize}} & \mathbb{E}\left[\xi\left(\boldsymbol{w}_t,\boldsymbol{x}_t\right)\right] \\ \textsf{subject to} & \boldsymbol{w}_t = \boldsymbol{f}_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{t-k},\dots,\boldsymbol{x}_{t-1}\right), \end{array}$$

  - $\xi(\cdot,\cdot)$: error function measuring overall system performance (e.g., negative Sharpe ratio).
- **Advantages of end-to-end architecture:**
  - Most powerful as it can learn to optimize the overall performance measure of interest.
- **Challenges of end-to-end architecture:**
  - Requires huge amounts of data.
  - Needs to be deep enough to properly optimize the performance measure, implying more weights to be learned.
  - Financial applications may not have the luxury of huge amounts of data.
  - Feasibility in practice is uncertain due to data constraints.

Reinforcement learning end-to-end portfolio design

- **Deep learning architecture requirements:**
    - Needs to be deep enough in terms of layers.
    - Requires huge amounts of training data.
    - More suitable for high-frequency data applications, such as high-frequency trading (HFT).
- **Reinforcement learning (RL) paradigm:**
    - Suitable for scenarios where actions of the agent affect the state of the system.
    - Different from supervised learning.
    - Involves a feedback loop between the learning system and its experiences.
    - Learning is implemented in an online fashion, i.e., in real time as orders are executed.
    - Learning is slower compared to supervised learning.
- **Reinforcement learning in portfolio strategy:**
    - Uncertain usefulness due to slower learning process.
    - Comprehensive overview of RL-based methods for quantitative trading provided in (Sun, Wang, and An 2023).

Outline

Case studies of DL portfolios

- **Current state of DL in portfolio design:**
  - Research flourishing since 2005, but still in early stages.
  - Uncertainty about full integration with financial systems.
- **Continuous flow of research:**
  - Increasing publications on DL in portfolio design.
  - Promising results, but require cautious interpretation.
- **Potential pitfalls in backtesting DL architectures:**
  - **Overfitting:** Multiple adjustments may lead to overfitting; lack of details on final networks.
  - **Look-ahead bias:** Risk of future data leakage during training and incorrect time alignment.
  - **Ignoring transaction costs:** Misleading results if frequent rebalancing costs are ignored; slow rebalancing may initially disregard costs.
- **Overview of DL models for financial applications:**
  - Comprehensive review up to 2020 in (Ozbayoglu, Gudelek, and Sezer 2020).
  - Focus on DL for financial time series forecasting in (Sezer, Gudelek, and Ozbayoglu 2020).

Example #1: LSTM for financial time series forecasting

- **Example of standard time series forecasting:**
  - **Reference:** (Fischer and Krauss 2018).
  - **Method:** LSTM network for binary classification.
  - **Classes:** Return larger or smaller than cross-sectional median return.
  - **Objective:** Minimize cross-entropy.
- **Network structure:**
  - **Input layer:** 1 feature (daily returns), lookback of $k = 240$ timesteps (approx. one trading year).
  - **Hidden layer:** LSTM with 25 hidden neurons (2,752 parameters).
  - **Output layer:** Fully connected with 2 neurons (softmax activation for class probabilities).
- **Portfolio design using DL forecasts:**
  - Predicts probability of each asset outperforming or underperforming the median.
  - Assets ranked by probability of outperforming the median.
  - Long-short quintile portfolio formed.

Example #1: LSTM for financial time series forecasting

- **Empirical results:**
  - **Data:** Daily data of S&P 500 stocks.
  - **Performance:** LSTM with quintile portfolio outperforms benchmarks (random forest, logistic regression, fully connected deep network).
  - **Sharpe ratio (before transaction costs):**
    - LSTM: 5.8
    - Random forest: 5.0
    - Fully connected network: 2.4
  - **Sharpe ratio (after transaction costs, 5 bps):**
    - LSTM: 3.8
    - Random forest: 3.4
    - Fully connected network: 0.9
  - **Market Sharpe ratio:** 0.7
  - **Period performance:** Better during 1993-2009, deteriorated between 2010-2015, profitability fluctuating around zero.

Example #2: Financial time series forecasting integrated with portfolio optimization

- **Example of portfolio-based time series forecasting:**
    - **Reference:** (Butler and Kwon 2023).
    - **Architecture:**
        - Simple linear network for forecasting returns.
        - Mean-variance portfolio (MVP) optimization component (Palomar 2025, chap. 7).
- **Key aspects:**
    - Partial derivatives of the portfolio solution are essential for backpropagation.
    - Derivatives detailed in (Butler and Kwon 2023).
- **Numerical experiments:**
    - **Data:** Daily returns from 24 global futures markets (1986-2020).
    - **Comparison:** Proposed method vs. benchmark minimizing MSE.
    - **Results:** Significant improvement in Sharpe ratio (transaction costs not considered).

Example #3: End-to-end NN-based portfolio

- **Example of portfolio-based and end-to-end DL architectures:**
  - **Reference:** (Uysal, Li, and Mulvey 2023).
  - **Approaches:**
    - **Model-based:** Neural network learns intermediate features fed into a portfolio optimization block.
    - **Model-free:** Neural network directly outputs portfolio allocation.
- **Model-free architecture:**
  - **Input layer:** Raw features (past $k = 5$ daily returns, past 10, 20, and 30-day average returns, volatilities of each asset).
  - **Hidden layer:** Fully-connected with 32 neurons.
  - **Output layer:** 7 neurons (number of assets) with softmax function for normalized portfolio allocation.

Example #3: End-to-end NN-based portfolio

- **Model-based architecture:**
  - **Input layer:** Same raw features as model-free.
  - **Hidden layers:**
    - First fully-connected hidden layer (similar to model-free).
    - Second hidden layer with softmax function for risk budgeting.
  - **Output layer:** Risk-parity portfolio (RPP) optimization block (Palomar 2025, chap. 11).
- **Empirical results:**
  - **Data:** Daily market data of seven ETFs (2011-2021).
  - **Performance (Sharpe ratio):**
    - **Model-based:** $1.10 \sim 1.15$.
    - **Nominal risk-parity portfolio:** $0.62 \sim 0.79$.
    - $1/N$ **portfolio:** $0.41 \sim 0.83$.
    - **Model-free:** $0.31 \sim 0.56$.
  - **Conclusion:** Model-based architecture preferred due to better performance and reduced overfitting. Transaction costs not considered in analysis.

Example #4: End-to-end DL-based portfolio

- **Example of end-to-end DL architecture:**
  - **Reference:** (C. Zhang et al. 2021), building on (Z. Zhang, Zohren, and Roberts 2020a).
  - **Framework:** Bypasses traditional forecasting and covariance matrix estimation.
  - **Objective:** Optimizes various functions like Sharpe ratio and mean-variance trade-off.
  - **Constraints:** Ensures output portfolio satisfies constraints on short selling, cardinality control, maximum positions, and leverage.
- **Architecture components:**
  - **Score block:** Produces raw portfolio scores $\boldsymbol{s}_t$ from market information (e.g., previous k returns $(\boldsymbol{x}_{t-k}, \dots, \boldsymbol{x}_{t-1})$).
  - **Architectures considered:**
    - Linear model.
    - Fully-connected network with 64 units.
    - Single LSTM layer with 64 units.
    - CNN with 4 layers (3 convolutional layers with filters of size 32, 64, 128, and a single LSTM layer with 64 units).

Example #4: End-to-end DL-based portfolio

- **Portfolio block:** Enforces desired structure on scores $\boldsymbol{s}_t$.
  - **No-shorting:** Softmax layer:
    $$\boldsymbol{w}_t = \frac{e^{\boldsymbol{s}_t}}{\mathbf{1}^\mathsf{T} e^{\boldsymbol{s}_t}};$$
  - **Shorting allowed:** Modified softmax:
    $$\boldsymbol{w}_t = \operatorname{sign}(\boldsymbol{s}_t) \times \frac{e^{\boldsymbol{s}_t}}{\mathbf{1}^\mathsf{T} e^{\boldsymbol{s}_t}};$$
  - **Maximum position control:** Generalized sigmoid $\sigma_a(z)$:
    $$\boldsymbol{w}_t = \operatorname{sign}(\boldsymbol{s}_t) \times \frac{\boldsymbol{\sigma}_a(|\boldsymbol{s}_t|)}{\mathbf{1}^\mathsf{T} \boldsymbol{\sigma}_a(|\boldsymbol{s}_t|)};$$
- **Empirical results:**
  - **Data:** Daily data.
  - **Performance:**
    - Without transaction costs: best with single LSTM layer, Sharpe ratio of 2.6.
    - With 2 bps transaction costs: performance similar to simple benchmarks.
  - **Conclusion:** Further work needed to account for transaction costs.

Example #5: End-to-end deep reinforcement learning portfolio

- **Example of deep reinforcement learning (DRL):**
  - **Reference:** (Z. Zhang, Zohren, and Roberts 2020b).
  - **Objective:** Maximize expected cumulative return, aligning with RL framework to maximize expected cumulative rewards through agent-environment interaction.
  - **Features:** Past prices, returns over varying time frames, technical indicators.
  - **Action space:**
    - Discrete set: $\{-1, 0, 1\}$ (short, no holding, long positions).
    - Continuous set: Entire $[-1, 1]$ interval.
  - **Reward function:** Volatility-adjusted return after accounting for transaction costs.
  - **Model architecture:** Two-layer LSTM networks with 64 and 32 units.
- **Evaluation:**
  - **Data:** 50 highly liquid futures contracts (2011-2019).
  - **Asset classes:** Commodities, equity indexes, fixed income, foreign exchange markets.
  - **Comparison:** Traditional time series momentum strategies.
  - **Results:**
    - Proposed algorithms generate positive profits despite substantial transaction costs.
    - Effectively track major market trends without altering positions.
    - Scale down or maintain positions during consolidation periods.

Outline

Summary

Deep neural networks excel in many domains, but will this revolution extend to financial systems?

While NLP advances benefit news sentiment analysis for trading, financial forecasting faces unique challenges:

- **Data scarcity**: Limited financial data (e.g., 504 observations for 2 years of daily prices)
- **Low signal-to-noise ratio**: Alpha-generating signals buried in overwhelming noise
- **Data nonstationarity**: Shifting statistical properties complicate learning
- **Adaptive feedback loops**: Discovered patterns quickly disappear when exploited
- **Lack of human success**: As Malkiel noted, even experts can't consistently outperform random selection

The jury remains out on deep learning's ultimate impact on financial systems.

References I

Ahmed, N. K., A. F. Atiya, N. E. Gayar, and H. El-Shishiny. 2010. "An Empirical Comparison of Machine Learning Models for Time Series Forecasting." *Econometric Reviews* 29 (5-6): 594–621.

Amos, B., and J. Z. Kolter. 2017. "OptNet: Differentiable Optimization as a Layer in Neural Networks." In *Proceedings of the International Conference on Machine Learning (ICML)*, 70:136–45.

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning*. Springer.

Bontempi, G., S. B. Taieb, and Y.-A. Le Borgne. 2012. "Machine Learning Strategies for Time Series Forecasting." In *European Business Intelligence Summer School*, 62–77. Springer.

Breiman, L. 2001. "Statistical Modelling: The Two Cultures." *Statistical Science* 16 (3): 199–231.

Bubeck, S., V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, et al. 2023. "Sparks of Artificial General Intelligence: Early Experiments with GPT-4." *Available at arXiv*. `https://doi.org/10.48550/arXiv.2303.12712`.

Butler, A., and R. H. Kwon. 2023. "Integrating Prediction in Mean-Variance Portfolio Optimization." *Quantitative Finance* 23 (3): 429–52.

References II

Fischer, T., and C. Krauss. 2018. "Deep Learning with Long Short-Term Memory Networks for Financial Market Predictions." *European Journal of Operational Research* 270 (2): 654–69.

Goodfellow, I., Y. Bengio, and A. Courville. 2016. *Deep Learning*. MIT Press.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. 2014. "Generative Adversarial Nets." In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 27.

Hastie, T., R. Tibshirani, and J. Friedman. 2009. *The Elements of Statistical Learning*. 2nd ed. Springer.

Ho, J., A. Jain, and P. Abbeel. 2020. "Denoising Diffusion Probabilistic Models." In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*. Virtual.

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

James, G., D. Witten, T. Hastie, and R. Tibshirani. 2013. *An Introduction to Statistical Learning: With Applications in R*. Springer.

References III

Kramer, M. A. 1991. "Nonlinear Principal Component Analysis Using Autoassociative Neural Networks." *AIChE Journal* 37 (2): 233–43.

Krizhevsky, A., I. Sutskever, and G. E. Hinton. 2012. "ImageNet Classification with Deep Convolutional Neural Networks." In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 25.

LeCun, Y., Y. Bengio, and G. Hinton. 2015. "Deep Learning." *Nature* 521: 436–44.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner. 1998. "Gradient-Based Learning Applied to Document Recognition." *Proceedings of the IEEE* 86 (11): 2278–2324.

Lin, H. W., M. Tegmark, and D. Rolnick. 2017. "Why Does Deep and Cheap Learning Work so Well?" *Journal of Statistical Physics* 168: 1223–47.

López de Prado, M. 2018a. *Advances in Financial Machine Learning*. Wiley.

———. 2018b. "The 10 Reasons Most Machine Learning Funds Fail." *Journal of Portfolio Management* 44 (6): 120–33.

References IV

———. 2019. “Ten Applications of Financial Machine Learning.” *SSRN Electronic Journal*. `https://dx.doi.org/10.2139/ssrn.3365271`.

Nielsen, M. A. 2015. *Neural Networks and Deep Learning*. Determination Press. `http://neuralnetworksanddeeplearning.com`.

OpenAI. 2023. “GPT-4 Technical Report.” *Available at arXiv*. `https://doi.org/10.48550/arXiv.2303.08774`.

Ozbayoglu, A. M., M. U. Gudelek, and O. B. Sezer. 2020. “Deep Learning for Financial Applications : A Survey.” *Available at arXiv*. `https://doi.org/10.48550/arXiv.2002.05786`.

Palomar, D. P. 2025. *Portfolio Optimization: Theory and Application*. Cambridge University Press.

Schmidhuber, J. 2015. “Deep Learning in Neural Networks: An Overview.” *Neural Networks* 61: 85–117.

———. 2022. “Annotated History of Modern AI and Deep Learning.” *Available at arXiv*. `https://doi.org/10.48550/arXiv.2212.11279`.

Sezer, O. B., M. U. Gudelek, and A. M. Ozbayoglu. 2020. “Financial Time Series Forecasting with Deep Learning : A Systematic Literature Review: 2005–2019.” *Applied Soft Computing* 90.

References V

Shalev-Shwartz, S., and S. Ben-David. 2014. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.

Sohl-Dickstein, J., E. A. Weiss, N. Maheswaranathan, and S. Ganguli. 2015. "Deep Unsupervised Learning Using Nonequilibrium Thermodynamics." In *Proceedings of the International Conference on Machine Learning (ICML)*, 37:2256–65.

Song, Y., and S. Ermon. 2019. "Generative Modeling by Estimating Gradients of the Data Distribution." In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*. Vancouver, Canada.

Sun, S., R. Wang, and B. An. 2023. "Reinforcement Learning for Quantitative Trading." *ACM Transactions on Intelligent Systems and Technology* 13 (3): 1–29.

Takahashi, S., Y. Chen, and K. Tanaka-Ishii. 2019. "Modelling Financial Time-Series with Generative Adversarial Networks." *Physica A: Statistical Mechanics and Its Applications* 527: 1–12.

Uysal, A. S., X. Li, and J. M. Mulvey. 2023. "End-to-End Risk Budgeting Portfolio Optimization with Neural Networks." *Annals of Operations Research*.

Vapnik, V. 1999. *The Nature of Statistical Learning Theory*. 2nd ed. Springer.

References VI

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. 2017. “Attention Is All You Need.” In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*.

Yoon, J., D. Jarrett, and M. van der Schaar. 2019. “Time-Series Generative Adversarial Networks.” In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*. Vancouver, Canada.

Zhang, C., Z. Zhang, M. Cucuringu, and S. Zohren. 2021. “A Universal End-to-End Approach to Portfolio Optimization via Deep Learning.” *Available at arXiv*. `https://doi.org/10.48550/arXiv.2111.09170`.

Zhang, Z., S. Zohren, and S. Roberts. 2020a. “Deep Learning for Portfolio Optimization.” *The Journal of Financial Data Science* 2 (4): 8–20.

———. 2020b. “Deep Reinforcement Learning for Trading.” *The Journal of Financial Data Science* 4 (1): 1–16.