Deep Learning for Financial Econometrics\textsuperscript{2}

Matthew Dixon
matthew.dixon@iit.edu
Illinois Institute of Technology

Machine Learning in Finance Bootcamp
Fields Institute
University of Toronto
September 26th, 2019
Code: https://github.com/mfrdixon/ML_Finance_Codes

\textsuperscript{2}Slides with an * in the title are advanced material for mathematicians. ** denotes advanced programming details.
Overview

- Deep Learning for cross-sectional data
  - Background: review of supervised machine learning
  - Introduction to feedforward neural networks
  - Related approximation and learning theory
  - Training and back-propagation
  - Interpretability with factor modeling example

- Deep Learning for time-series data prediction
  - Understand the conceptual formulation and statistical explanation of RNNs
  - Understand the different types of RNNs, including plain RNNs, GRUs and LSTMs
  - Understand the conceptual formulation and statistical explanation of CNNs.
  - Understand 1D CNNs for time series prediction.
  - Understand how dilation CNNs capture multi-scales.
Overview

- Deep Learning for dimensionality reduction
  - Understand principal component analysis for dimension reduction.
  - Understand how to formulate a linear autoencoder.
  - Understand how a linear autoencoder performs PCA.
Quick Quiz

Which of the following statements are true?

A. Machine learning is different from statistics: Machine learning methods assume the data generation process is unknown.
B. The output from neural network classifiers are the probabilities of an input belonging to a class.
C. We need multiple hidden layers in the neural network to capture non-linearity.
D. Neural networks provide no statistical interpretability, they are ‘black-boxes’ and the importance of the features is unknown.
E. Neural networks can only be fitted to stationary data.
Quick Quiz

Which of the following statements are true?

A Machine learning is different from statistics: Machine learning methods assume the data generation process is unknown [True]

B The output from neural network classifiers are the probabilities of an input belonging to a class [True]

C We need multiple hidden layers in the neural network to capture non-linearity [False]

D Neural networks provide no statistical interpretability, they are ’black-boxes’ and the importance of the features is unknown [False]

E Neural networks can only be fitted to stationary data [False]

---

3 The Universal representation theorem suggests that we only need one hidden layer: Andrei Nikolaevich Kolmogorov, On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. AMS Translation, 28(2):55-59, 1963.
## Quick Overview

<table>
<thead>
<tr>
<th>Property</th>
<th>Statistical Inference</th>
<th>Machine Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>Causal models with explanatory power</td>
<td>Prediction performance, often with limited explanatory power</td>
</tr>
<tr>
<td>Data</td>
<td>The data is generated by a model</td>
<td>The data generation process is unknown</td>
</tr>
<tr>
<td>Framework</td>
<td>Probabilistic</td>
<td>Algorithmic &amp; Probabilistic</td>
</tr>
<tr>
<td></td>
<td>Typically linear</td>
<td>Non-linear</td>
</tr>
<tr>
<td>Expressability</td>
<td>Based on information criteria</td>
<td>Numerical optimization</td>
</tr>
<tr>
<td>Model selection</td>
<td>Limited to lower dimensional data</td>
<td>Scales to higher dimensional input data</td>
</tr>
<tr>
<td>Scalability</td>
<td>Prone to over-fitting</td>
<td>Designed for out-of-sample performance</td>
</tr>
<tr>
<td>Robustness</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Supervised Machine Learning

• Machine learning addresses a fundamental prediction problem: Construct a nonlinear predictor, \( \hat{Y}(X) \), of an output, \( Y \), given a high dimensional input matrix \( X = (X_1, \ldots, X_P) \) of \( P \) variables.

• Machine learning can be simply viewed as the study and construction of an input-output map of the form

\[
Y = F(X) \quad \text{where} \quad X = (X_1, \ldots, X_P).
\]

• The output variable, \( Y \), can be continuous, discrete or mixed.

• For example, in a classification problem, \( F : X \rightarrow Y \) where \( Y \in \{1, \ldots, K\} \) and \( K \) is the number of categories.

• We will denote the \( i^{th} \) observation of the data \( \mathcal{D} := (X, Y) \) as \( (x_i, y_i) \) - this is a “feature-vector” and response (a.k.a. “label”). Note that lower caps refer to a specific observation in \( \mathcal{D} \).
Taxonomy of Most Popular Neural Network Architectures

- **feed forward**
- **auto-encoder**
- **convolution**
- **recurrent**
- **Long / short term memory**
- **neural Turing machines**

*Figure:* Most commonly used deep learning architectures for modeling. Source: [http://www.asimovinstitute.org/neural-network-zoo](http://www.asimovinstitute.org/neural-network-zoo)
**FFWD Neural Networks**

Neural network model:

\[ Y = F_{W,b}(X) + \epsilon \]

where \( F_{W,b} : \mathbb{R}^p \rightarrow \mathbb{R}^d \) is a deep neural network with \( L \) layers

\[ \hat{Y}(X) := F_{W,b}(X) = f^{(L)}_{W^{(L)},b^{(L)}} \circ \cdots \circ f^{(1)}_{W^{(1)},b^{(1)}}(X) \]

- \( W = (W^{(1)}, \ldots, W^{(L)}) \) and \( b = (b^{(1)}, \ldots, b^{(L)}) \) are weight matrices and bias vectors.
- For any \( W^{(i)} \in \mathbb{R}^{m \times n} \), we can write the matrix as \( n \) column \( m \)-vectors \( W^{(i)} = [w^{(i)}_1, \ldots, w^{(i)}_n] \).
- Denote each weight as \( w^{(\ell)}_{i,j} := (W^{(\ell)})_{i,j} \).
- \( X \) is a \( N \times p \) matrix of observations in \( \mathbb{R}^p \).
- No assumptions are made on the distribution of the error, \( \epsilon \), other than it is independently distributed.
Deep Neural Networks*

• Let $\sigma : \mathbb{R} \to B \subset \mathbb{R}$ denote a continuous, monotonically increasingly function.

• A function $f_{W^{(\ell)},b^{(\ell)}}^{(\ell)} : \mathbb{R}^n \to \mathbb{R}^m$, given by
  
  $$f(v) = W^{(\ell)} \sigma^{(\ell-1)}(v) + b^{(\ell)}, \quad W^{(\ell)} \in \mathbb{R}^{m \times n} \text{ and } b^{(\ell)} \in \mathbb{R}^m,$$

  is a semi-affine function in $v$, e.g. $f(v) = wtanh(v) + b$.

• $\sigma(\cdot)$ are known activation functions of the output from the previous layer, e.g. $\sigma(x) := \max(x, 0)$.

• $F_{W,b}(X)$ is a composition of semi-affine functions.

• If all the activation functions are linear, $F_{W,b}$ is just linear regression, regardless of the number of layers $L$. 
Geometric Interpretation of FFWD Neural Networks

No hidden layers  One hidden layer  Two hidden layers

$X_1$  $X_2$  $X_1$  $X_2$  $X_1$  $X_2$
Geometric Interpretation of FFWD Neural Networks

Half-Moon Dataset

No hidden layers

One hidden layer

Two hidden layers
Why do we need more Neurons?
Figure: The number of hidden units is adjusted according to the requirements of the classification problem and can be very high for data sets which are difficult to separate.
Geometric Interpretation of Neural Networks

Figure: Hyperplanes defined by three activated neurons in the hidden layer.
Universal Representation Theorem [1989]*

- Let $C_p := \{f : \mathbb{R}^p \to \mathbb{R} \mid f(x) \in C(\mathbb{R})\}$ be the set of continuous functions from $\mathbb{R}^p$ to $\mathbb{R}$.
- Denote $\Sigma^p(\sigma)$ as the class of functions
  \[ \{F_{W,b} : \mathbb{R}^p \to \mathbb{R} : F_{W,b}(x) = W^{(2)}\sigma(W^{(1)}x + b^{(1)}) + b^{(2)}\} \].
- Consider $\Omega := (0, 1]$ and let $C_0$ be the collection of all open intervals in $(0, 1]$.
- Then $\sigma(C_0)$, the $\sigma$-algebra generated by $C_0$, is the Borel $\sigma$-algebra, $\mathcal{B}((0, 1])$.
- Let $M^p := \{f : \mathbb{R}^p \to \mathbb{R} \mid f(x) \in \mathcal{B}(\mathbb{R})\}$ denote the set of all Borel measurable functions from $\mathbb{R}^p$ to $\mathbb{R}$.
- Denote the Borel $\sigma$-algebra of $\mathbb{R}^p$ as $\mathcal{B}^p$. 
Universal Representation Theorem*

Theorem (Hornik, Stinchcombe & White, 1989)

For every monotonically increasing activation function $\sigma$, every input dimension size $p$, and every probability measure $\mu$ on $(\mathbb{R}^p, \mathcal{B}^p)$, $\Sigma^p(g)$ is uniformly dense on compacta in $C^p$ and $\rho_\mu$-dense in $M^p$. 

Shattering

- What is the maximum number of points that can be arranged so that $F_{W,b}(X)$ shatters them?
- i.e. for all possible assignments of binary labels to those points, does there exist a $W, b$ such that $F_{W,b}$ makes no errors when classifying that set of data points?
- Every distinct pair of points is separable with the linear threshold perceptron. So every data set of size 2 is shattered by the perceptron.
- However, this linear threshold perceptron is incapable of shattering triplets, for example $X \in \{-0.5, 0, 0.5\}$ and $Y \in \{0, 1, 0\}$.
- In general, the VC dimension of the class of halfspaces in $\mathbb{R}^k$ is $k + 1$. For example, a 2-d plane shatters any three points, but can not shatter four points.
- This maximum no. of points is the Vapnik-Chervonenkis (VC) dimension and is one characterization of learnability of a classifier.
Example of Shattering over the Interval $[-1, 1]$

Figure: For the points $\{-0.5, 0.5\}$, there are weights and biases that activates only one of them ($W = 1, b = 0$ or $W = -1, b = 0$), none of them ($W = 1, b = -0.75$) and both of them ($W = 1, b = 0.75$).
VC Dimension Example

• Determine the VC dimension of the indicator function where $\Omega = [0, 1]$

$$F(x) = \{ f : \Omega \rightarrow \{0, 1\}, \ f(x) = \mathbb{1}_{x \in [t_1, t_2]}, \ \text{or} \ f(x) = 1 - \mathbb{1}_{x \in [t_1, t_2]}, \ t_1 < t_2 \in \Omega \}.$$  

• Suppose there are three points $x_1, x_2$ and $x_3$ and assume $x_1 < x_2 < x_3$ without loss of generality. All possible binary labeling of the points is reachable, therefore we assert that $VC(F) \geq 3$.

• With four points $x_1, x_2, x_3$ and $x_4$ (assumed increasing as always), you cannot label $x_1$ and $x_3$ with the value 1 and $x_2$ and $x_4$ with the value 0 for example. So $VC(F) = 3$.

• A related, but alternative, approach to the VC dimension is a numerical analysis perspective.
When is a neural network a spline?*

- Under certain choices of the activation function, we can construct NNs which are piecewise polynomial interpolants referred to as 'splines'.
- Let $f : \Omega \to \mathbb{R}$ be any Lipschitz continuous function over a bounded domain $\Omega \subset \mathbb{R}^p$:

$$\forall x, x' \in \Omega, \ |f(x') - f(x)| \leq L|x' - x|,$$

for some constant $L \geq 0$.
- Suppose that the values $f_k := f(x_k)$ are known only at distinct points $\{x_k\}_{k=1}^n$ in $\Omega$.
- Form an orthogonal basis over $\Omega$ to give the interpolant

$$\hat{f}(x) = \sum_{k=1}^n \phi_k(x) f_k, \ \forall x \in \Omega,$$

where the $\{\phi_k(x)\}_{k=1}^n$ are piecewise constant (PC) orthogonal basis functions, $\phi_i(x_j) = \delta_{ij}$ (Kronecker delta).
When is a neural network a spline?*

- If the data points \( \{x_k\}_{k=1}^n \) are (almost) uniformly distributed in the bounded domain \( \Omega \), then the error of our PC interpolant scales as

\[
\| f - \hat{f} \|_{L^2_{p,\mu}(\Omega)} \sim O(n^{-\frac{1}{p}}). \tag{1}
\]
Piecwise basis functions as differences of Heaviside functions*

- Heaviside functions (a.k.a. step functions) over $\mathbb{R}$ are defined as
  $$H(x) = \begin{cases} 
  1 & x \geq 0, \\
  0 & x < 0,
  \end{cases}$$

- Choose the domain to be the unit interval in $\mathbb{R}$ and restrict the data to a uniform grid of width $2\epsilon$ so that $x_k = (2k - 1)\epsilon, \ k = 1 \ldots n$.

- Construct a piecewise constant basis from the difference of Heaviside functions:
  $$\phi_k(x) = H(x - (x_k - \epsilon)) - H(x - (x_k + \epsilon)).$$

- Basis functions are indicator functions:
  $$\phi_k = \mathbb{1}_{[x_k-\epsilon,x_k+\epsilon]}, \ \forall k = 1, \ldots, n,$$
  with compact support over $[x_k - \epsilon, x_k + \epsilon)$ and centered about $x_k$. 

*Note: This is part of a larger discussion on Neural Networks and Splines.
Piecewise constant basis functions*

Figure: The first three piecewise constant basis functions produced by the difference of neighboring step functions, $\phi_k(x) = H(x_k - \epsilon) - H(x_k + \epsilon)$. 
Single Layer NNs Activated with Heaviside functions*

- **Key Idea:** Choose the bias \( b_k^{(1)} = -2(k - 1)\epsilon \) and weights \( W^{(1)} = 1 \) then

\[
H(W^{(1)}x + b^{(1)}) = \begin{bmatrix}
H(x) \\
H(x - 2\epsilon) \\
\vdots \\
H(x - 2(k - 1)\epsilon) \\
\vdots \\
H(x - (2n - 1)\epsilon)
\end{bmatrix}
\]

- The two layer NN approximation,
\[
\hat{f}(x) = W^{(2)}H(W^{(1)}x + b^{(1)}),
\]
is a PC interpolant when

\[
W^{(2)} = [f(x_1), f(x_2) - f(x_1), \ldots, f(x_K) - f(x_{n-1})].
\]
Single Layer NNs Activated with Heaviside functions*

• This single layer neural network gives the following output:

\[
\hat{f}(x) = \begin{cases} 
  f(x_1) & x \leq 2\epsilon, \\
  f(x_2) & 2\epsilon < x \leq 4\epsilon, \\
  \ldots & \ldots \\
  f(x_k) & 2(k)\epsilon < x \leq 2(k + 1)\epsilon, \\
  \ldots & \ldots \\
  f(x_{n-1}) & 2(n - 1)\epsilon < x \leq 2n\epsilon. 
\end{cases}
\]

• \(\hat{f}(x)\) is a linear interpolant over \([0, 1]\).
Single Layer NNs Activated with Heaviside functions

Figure: The approximation of $\sin(2\pi x)$ using gridded input data and Heaviside activation functions.
Single Layer NNs Activated with Heaviside functions *

Theorem

\[ |f(x) - \hat{f}(x)| \leq \delta, \forall x \in [0, 1] \text{ if there are at least } n = \left\lceil \frac{1}{2\epsilon} + 1 \right\rceil \text{ hidden units and } \delta := L\epsilon. \]

Proof.

Since \( x_k = (2k - 1)\epsilon \), we have that \( \hat{f}(x) = f(x_k) \) over the interval \([x_k - \epsilon, x_k + \epsilon]\), which is the support of \( \phi_k(x) \). By the Lipschitz continuity of \( f(x) \), it follows that the worst case error appears at the mid-point of any interval \([x_k, x_{k+1}]\)

\[ |f(x_k + \epsilon) - \hat{f}(x + \epsilon)| = |f(x + \epsilon) - f(x_k)| \leq |f(x_k)| + L\epsilon - |f(x_k)| = \delta. \]
Training with Regularization under IID noise

- With a training set \( D = \{\mathbf{x}_i, y_i\}_{i=1}^N \), solve the constrained optimization

\[
(\hat{W}, \hat{b}) = \arg \min_{(W, b)} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y_i, \hat{Y}(x_i))
\]

- Often, the \( L_2 \)-norm for a traditional least squares problem is chosen as an error measure, and we then minimize the loss function combined with a penalty

\[
\mathcal{L}(y_i, \hat{y}_i) = \|y_i - \hat{y}_i\|_2^2 + \lambda_1 \|W\|_1
\]

- \( \nabla \mathcal{L} \) is given in closed form by a chain rule and, through back-propagation, each layer's weights \( W^{(\ell)} \) and biases \( b^{(\ell)} \) are fitted with stochastic gradient descent.
Equivalence with OLS estimation

- Without activation and no hidden layers, the optimal parameters \( \hat{\beta} := [\hat{b}, \hat{W}] \) are an OLS estimator.

- If we assume that the errors \( \epsilon_1, \epsilon_2, \ldots, \epsilon_N \) are white noise and let \( y := Y \in \mathbb{R} \) so that the linear model is

\[
y = X\beta + \epsilon,
\]

then

\[
\hat{\beta} = (X'X)^{-1}X'y
\]

where the augmented matrix is

\[
X = \begin{bmatrix}
1 & x_{1,1} & \ldots & x_{1,p} \\
\vdots & \vdots & \ddots & \vdots \\
1 & x_{N,1} & \ldots & x_{N,p}
\end{bmatrix}
\]
Training

• Solve the constrained optimization

$$(\hat{W}, \hat{b}) = \arg \min_{W, b} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y_i, \hat{Y}^{W, b}(x_i))$$

• If $Y$ is continuous then the $L_2$-norm for a traditional least squares problem is chosen as an error measure, and we then minimize the loss function

$$\mathcal{L}(Y, \hat{Y}(X)) = \|Y - \hat{Y}\|_2^2$$

• A 1-of-$K$ encoding is used for a categorical response, so that $Y$ is a K-binary vector $Y \in [0, 1]^K$ and the value $k$ is presented as $Y_k = 1$ and $Y_j = 0, \forall j \neq k$. The loss function is the cross-entropy term

$$\mathcal{L}(Y, \hat{Y}(X)) = -Y^T \ln \hat{Y}.$$
Derivative of the softmax function

Using the quotient rule \( f'(x) = \frac{g'(x)h(x) - h'(x)g(x)}{[h(x)]^2} \), the derivative \( \sigma := \sigma(x) \) can be written as:

\[
\begin{align*}
\frac{\partial \sigma_i}{\partial x_i} &= \frac{\exp(x_i)\|\exp(x)\|_1 - \exp(x_i)\exp(x_i)}{\|\exp(x)\|^2_1} \\
&= \frac{\exp(x_i)}{\|\exp(x)\|_1} \cdot \frac{\|\exp(x)\|_1 - \exp(x_i)}{\|\exp(x)\|_1} \\
&= \sigma_i(1 - \sigma_i)
\end{align*}
\]
Derivative of the softmax function

- For the case $i \neq j$, the derivative of the sum is:

$$\frac{\partial \sigma_i}{\partial x_j} = \frac{0 - \exp(x_i) \exp(x_j)}{||\exp(x)||^2_1}$$

$$= -\frac{\exp(x_j)}{||\exp(x)||_1} \cdot \frac{\exp(x_i)}{||\exp(x)||_1}$$

$$= -\sigma_j \sigma_i$$

- This can be written compactly as $\frac{\partial \sigma_i}{\partial x_j} = \sigma_i (\delta_{ij} - \sigma_j)$. 
Updating the Weight Matrices

- In machine learning, the goal is to find the weights and biases given an input $X$. We can express $\hat{Y}$ as a function of the weight matrix $W \in \mathbb{R}^{K \times M}$ and bias $b \in \mathbb{R}^{K}$ so that

$$
\hat{Y}(W, b) = \sigma \circ I(W, b),
$$

where the input function $I : \mathbb{R}^{K \times M} \times \mathbb{R}^{K} \rightarrow \mathbb{R}^{K}$ is of the form $I(W, b) := WX + b$

- Applying the multivariate chain rule to give the Jacobian of $\hat{Y}(W, b)$ gives

$$
\nabla \hat{Y}(W, b) = \nabla(\sigma \circ I)(W, b) \quad (8)
$$

$$
= \nabla \sigma(I(W, b)) \cdot \nabla I(W, b) \quad (9)
$$
Updating the Weight Matrices

- Recall that the cross entropy function is given by
  \[ L(Y, \hat{Y}(X)) = -Y_k^T \ln \hat{Y}_k \]

- Since \( Y \) is a constant we can express the cross-entropy as a function of \((W, b)\)
  \[ L(W, b) = L \circ \sigma(I(W, b)). \]

- Applying the multivariate chain rule gives
  \[
  \nabla L(W, b) = \nabla (L \circ \sigma)(I(W, b))
  = \nabla L(\sigma(I(W, b))) \cdot \nabla \sigma(I(W, b)) \cdot \nabla I(W, b)
  \]
Back-propagation

- Use stochastic gradient descent to find the minimum

\[(\hat{W}, \hat{b}) = \arg \min_{W, b} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y_i, \hat{y}_i)\]

- Because of the compositional form of the model, the gradient can be easily derived using the chain rule for differentiation.
- This can be computed by a forward and then a backward sweep (‘back-propagation’) over the network, keeping track only of quantities local to each neuron.
Back-propagation

Forward Pass:
- Set $Z^{(0)} = X$ and for $\ell = 1, \ldots, L$ set
  $$Z^{(\ell)} = f_{W^{(\ell)}, b^{(\ell)}}(Z^{(\ell-1)}) = \sigma^{(\ell)}(W^{(\ell)}Z^{(\ell-1)} + b^{(\ell)}).$$

Back-propagation:
- Define the back-propagation error $\delta^{(\ell)} := \nabla_{b^{(\ell)}} \mathcal{L}$, where given $\delta^{(L)} = \hat{Y} - Y$ and for $\ell = L - 1, \ldots, 1$ set
  $$\delta^{(\ell)} = (\nabla_{I^{(\ell)}} \sigma^{(\ell)}) (W^{(\ell+1)})^T \delta^{(\ell+1)}, \quad (10)$$
  $$\nabla_{W^{(\ell)}} \mathcal{L} = \delta^{(\ell)} \otimes Z^{(\ell-1)}, \quad (11)$$
  and $\otimes$ is the outer product of two vectors.
### Updating the weights

- The weights and biases are updated according to the expression:

\[
\Delta W^{(\ell)} = -\gamma \nabla_{W^{(\ell)}} \mathcal{L} = -\gamma \delta^{(\ell)} \otimes Z^{(\ell-1)}, \quad (12)
\]

\[
\Delta b^{(\ell)} = -\gamma \delta^{(\ell)}, \quad (13)
\]

where \(\gamma\) is a learning rate parameter. See back-propagation notebook example and Appendix of Chpt 2 lecture notes for more details.

- **Mini-Batch** or **offline** updates involve using many observations of \(X\) at the same time. The **batch size** refers to the number of observations of \(X\) used in each pass.

- **An epoch** refers to a round-trip (forward+backward sweep) of all training examples.
Back-propagation example with three layers

- Suppose that a feed-forward network classifier has two sigmoid activated hidden layers and a softmax activated output layer.
- After a forward pass, the values of \( \{ Z^{(\ell)} \}_{\ell=1}^{3} \) are stored and the error \( \hat{Y} - Y \), where \( \hat{Y} = Z^{(3)} \), is calculated for one observation of \( X \).
- The back-propagation and weight updates in the final layer are evaluated:

\[
\delta^{(3)} = \hat{Y} - Y \\
\nabla_{W^{(3)}} \mathcal{L} = \delta^{(3)} \otimes Z^{(2)}.
\]
From Equations 10 and 11, we update the back-propagation error and weight updates for Hidden Layer 2

\[
\delta^{(2)} = Z^{(2)}(1 - Z^{(2)})(W^{(3)})^T \delta^{(3)}, \\
\nabla_{W^{(2)}} \mathcal{L} = \delta^{(2)} \otimes Z^{(1)}.
\]

Repeating for Hidden Layer 1

\[
\delta^{(1)} = Z^{(1)}(1 - Z^{(1)})(W^{(2)})^T \delta^{(2)}, \\
\nabla_{W^{(1)}} \mathcal{L} = \delta^{(1)} \otimes X.
\]

We update the weights and biases using Equations 12 and 13, so that \( b^{(3)} \rightarrow b^{(3)} - \gamma \delta^{(3)}, \ W^{(3)} \rightarrow W^{(3)} - \gamma \delta^{(3)} \otimes Z^{(2)} \) and repeat for the other weight-bias pairs, \( \{ (W^{(\ell)}, b^{(\ell)}) \}_{\ell=1}^2 \).
We can use callbacks in Keras to terminate the training based on stopping criteria.

```python
from keras.callbacks import EarlyStopping
...
callbacks = [EarlyStopping(monitor='loss', min_delta=0, patience=2)]
...
model.fit(X,Y, callbacks=callbacks)
```
Explanatory Power of Neural Networks

• In a linear regression model

\[ \hat{Y} = F_\beta(X) := \beta_0 + \beta_1 X_1 + \cdots + \beta_K X_K \]

the sensitivities are

\[ \partial_{X_i} \hat{Y} = \beta_i \]

• In a FFWD neural network, we can use the chain rule

\[ \partial_{X_i} \hat{Y} = \partial_{X_i} F_{W,b}(X) = \partial_{X_i} f_{W(L),b(L)} \circ \cdots \circ f_{W(1),b(1)}(X) \]

• For example, with one hidden layer, \( \sigma(x) := \tanh(x) \) and 
\[ f_{W(\ell),b(\ell)}(X) := \sigma(Z^{(\ell)}) := \sigma(W^{(\ell)}X + b^{(\ell)}) : \]

\[ \partial_{X_j} \hat{Y} = \sum_i w^{(2)}_i (1 - \sigma^2(l^{(1)}_i))w^{(1)}_{i,j} \quad \text{where} \quad \partial_x \sigma(x) = (1 - \sigma^2(x)) \]
Explanatory Power of Neural Networks

- Or in matrix form, using the Jacobian $J = D(I^{(1)})W^{(1)}$ of some continuous $\sigma$:
  \[
  \partial_X \hat{Y} = W^{(2)}J(I^{(1)}) = W^{(2)}D(I^{(1)})W^{(1)},
  \]
  where $D_{i,i}(l) = \sigma'(l_i), \ D_{i,j\neq i} = 0$.

- Bounds on the sensitivities are given by the product of the weight matrices
  \[
  \min(W^{(2)}W^{(1)},0) \leq \partial_X \hat{Y} \leq \max(W^{(2)}W^{(1)},0).
  \]

---

$^4$When $\sigma$ is an identity function, the Jacobian $J(I^{(1)}) = W^{(1)}$. 
Example: Step test

- The model is trained to the following data generation process where the coefficients of the features are stepped:

\[
\hat{Y} = \sum_{i=1}^{10} iX_i, \quad X_i \sim U(0, 1).
\]
Example: Step test

The figure shows the ranked importance of the input variables in a fitted neural network with one hidden layer.

**Figure:** Step test: This figure shows the ranked importance of the input variables in a fitted neural network with one hidden layer.
Example: Friedman test

\[ Y = 10 \sin (\pi X_1 X_2) + 20 (X_3 - 0.5)^2 + 10X_4 + 5X_5 + \epsilon, \]
Figure: Friedman test: Estimated sensitivities of the fitted neural network to the input.
Interaction Terms Introduced by Activation

- Even with one hidden layer, the activation function introduces interaction terms \( X_i X_j \).
- To see this, consider the partial derivative

\[
\partial_{X_j} \hat{Y} = \sum_i w^{(2)}_i \sigma'(I^{(1)}_i) w^{(1)}_{i,j}
\]

- and differentiate again with respect to \( X_k \), \( k \neq i \) to give the Hessian:

\[
\partial^2_{X_j, X_k} \hat{Y} = -2 \sum_i w^{(2)}_i \sigma''(I^{(1)}_i) w^{(1)}_{i,j} w^{(1)}_{i,k}
\]

- which is not zero everywhere unless \( \sigma \) is (piecewise) linear or constant.
Interaction Terms Introduced by Activation

Figure: Friedman test: Estimated Interaction terms in the fitted neural network to the input.
The Jacobian using ReLU activation

- In matrix form, with $\sigma(x) = \max(x, 0)$, the Jacobian, $J$, can be written as linear combination of Heaviside functions

$$ J := J(X) = \partial_X \hat{Y}(X) = W^{(2)} J(I^{(1)}) = W^{(2)} H(W^{(1)} X + b^{(1)}) W^{(1)}, $$

where $H_{ii}(I^{(1)}) = H(I_i^{(1)}) = \mathbb{1}_{I_i^{(1)}>0}$, $H_{ij} = 0$, $j \geq i$.

- This can be written in matrix element form as

$$ J_{ij} = [\partial_X \hat{Y}]_{ij} = \sum_{k=1}^{n} W_{ik}^{(2)} W_{kj}^{(1)} H(I_k^{(1)}) = \sum_{k=1}^{n} c_k H_k(I^{(1)}) $$

where $a_k := W_{ik}^{(2)} W_{kj}^{(1)}$.

- As a linear combination of indicator functions, we have

$$ J_{ij} = \sum_{k=1}^{n-1} a_k \mathbb{1}_{\{I_k^{(1)}>0, I_{k+1}^{(1)} \leq 0\}} + a_n \mathbb{1}_{\{I_n^{(1)}>0\}}, \quad a_k := \sum_{i=1}^{k} c_i. $$
The Jacobian using ReLU activation

- Alternatively, the Jacobian can be expressed in terms of $X$

$$J_{ij} = \sum_{k=1}^{n-1} a_k \mathbb{I}\{W_k^{(1)}X > -b_k^{(1)}, W_{k+1}^{(1)}X \leq -b_{k+1}^{(1)}\} + a_n \mathbb{I}\{W_n^{(1)}X > -b_n^{(1)}\}.$$ 

- wlog: In the case when $p = 1$, this simplifies to a weighted sum of independent Bernoulli trials

$$J_{ij} = \sum_{k=1}^{n-1} a_k \mathbb{I}x_k < X \leq x_{k+1} + a_n \mathbb{I}x_n < X, \ j = 1.$$ 

where $x_k := \frac{b_k^{(1)}}{W_k^{(1)}}$. The expectation of the Jacobian is given by

$$\mu_{n-1} := \mathbb{E}[J_{ij}] = \sum_{k=1}^{n} a_k p_k, \ p_k := \Pr[x_k < X \leq x_{k+1}] \forall k = 1, \ldots, n-1, \ p_n$$

- For finite weights, the expectation is bounded above by $\sum_{k=1}^{n} a_k$. 
Bounds on the Variance

- The variance is
  \[
  \mathbb{V}[J_{ij}] = \sum_{k=1}^{n-1} a_k \mathbb{V}[\mathbb{1}\{I_k^{(1)}>0, I_{k+1}^{(1)}\leq 0\}] + a_n \mathbb{V}[\mathbb{1}\{I_n^{(1)}>0\}] = \sum_{k=1}^{n} a_k p_k (1-p_k).
  \]

- If the weights are constrained so that the mean is constant, \( \mu_{ij}^n = \mu_{ij} \), then the weights are \( a_k = \frac{\mu}{np_k} \). Then the variance is monotonically increasing with the number of hidden units:
  \[
  \mathbb{V}[J_{ij}] = \mu \frac{n-1}{n} < \mu.
  \]

- Otherwise, in general, we have
  \[
  \mathbb{V}[J_{ij}] \leq \mu_{ij} \leq \sum_{i=1}^{n} a_k.
  \]

- Increasing variance with more hidden units is undesirable for interpretability - we want the opposite effect!
Robustness of Interpretability

- If the data is generated from a linear model, can the neural network recover the correct weights?

\[ Y = \beta_1 X_1 + \beta_2 X_2 + \epsilon, \quad X_1, X_2, \epsilon \sim \mathcal{N}(0, 1), \quad \beta_1 = 1, \beta_2 = 1 \]

- Compare OLS estimators with zero hidden layer NNs and single hidden layers NNs.

- Use \textit{tanh} activation functions for smoothness of the Jacobian because \( \max(x, 0) \) gives a piecewise continuous Jacobian.

- Increase the number of hidden layers and show that the variance of the sensitivities convergence with.
## Robustness of Interpretability

<table>
<thead>
<tr>
<th>Method</th>
<th>$\hat{\beta}_0$</th>
<th>$\hat{\beta}_1$</th>
<th>$\hat{\beta}_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLS</td>
<td>0</td>
<td>1.0154</td>
<td>1.018</td>
</tr>
<tr>
<td>NN$_0$</td>
<td>$b^{(1)} = 0.03$</td>
<td>$W_1^{(1)} = 1.0184141$</td>
<td>$W_2^{(1)} = 1.02141815$</td>
</tr>
<tr>
<td>NN$_1$</td>
<td>$W^{(2)}\sigma(b^{(1)}) + b^{(2)} = 0.02$</td>
<td>$\mathbb{E}[W^{(2)}D^{(1)}(I^{(1)})W_1^{(1)}] = 1.013887$</td>
<td>$\mathbb{E}[W^{(2)}D^{(1)}(I^{(1)})W_2^{(1)}] = 1.02224$</td>
</tr>
</tbody>
</table>
Robustness of Interpretability

(a) density of $\hat{\beta}_1$

(b) density of $\hat{\beta}_2$
## Robustness of Interpretability ($\hat{\beta}_1$)

<table>
<thead>
<tr>
<th>Hidden Units</th>
<th>Mean</th>
<th>Median</th>
<th>Std.dev</th>
<th>1% C.I.</th>
<th>99% C.I.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.980875</td>
<td>1.0232913</td>
<td>0.10898393</td>
<td>0.58121675</td>
<td>1.0729908</td>
</tr>
<tr>
<td>10</td>
<td>0.9866159</td>
<td>1.0083131</td>
<td>0.056483902</td>
<td>0.76814914</td>
<td>1.0322522</td>
</tr>
<tr>
<td>50</td>
<td>0.9918353</td>
<td>1.0029879</td>
<td>0.03123002</td>
<td>0.8698967</td>
<td>1.0182846</td>
</tr>
<tr>
<td>100</td>
<td>1.0071343</td>
<td>1.0175397</td>
<td>0.028034585</td>
<td>0.89689034</td>
<td>1.0296803</td>
</tr>
<tr>
<td>200</td>
<td>1.0152218</td>
<td>1.0249312</td>
<td>0.026156902</td>
<td>0.9119074</td>
<td>1.0363332</td>
</tr>
</tbody>
</table>

The confidence interval narrows with increasing number of hidden units.
# Robustness of Interpretability ($\hat{\beta}_2$)

<table>
<thead>
<tr>
<th>Hidden Units</th>
<th>Mean</th>
<th>Median</th>
<th>Std.dev</th>
<th>1% C.I.</th>
<th>99% C.I.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.98129386</td>
<td>1.0233982</td>
<td>0.10931312</td>
<td>0.5787732</td>
<td>1.073728</td>
</tr>
<tr>
<td>10</td>
<td>0.9876832</td>
<td>1.0091512</td>
<td>0.057096474</td>
<td>0.76264584</td>
<td>1.0339714</td>
</tr>
<tr>
<td>50</td>
<td>0.9903236</td>
<td>1.0020974</td>
<td>0.031827927</td>
<td>0.86471796</td>
<td>1.0152498</td>
</tr>
<tr>
<td>100</td>
<td>0.9842479</td>
<td>0.9946766</td>
<td>0.028286876</td>
<td>0.87199813</td>
<td>1.0065105</td>
</tr>
<tr>
<td>200</td>
<td>0.9976638</td>
<td>1.0074166</td>
<td>0.026751818</td>
<td>0.8920307</td>
<td>1.0189484</td>
</tr>
</tbody>
</table>

The confidence interval narrows with increasing number of hidden units.
Linear Cross-sectional Factor Models

- Consider the linear cross-sectional factor model:

\[ r_t = B_{t-1}f_t + \epsilon_t, \quad t = 1, \ldots, T \]

- \( B_t = [1 \mid b_{1,t} \mid \cdots \mid b_{K,t}] \) is the \( N \times K + 1 \) matrix of factor exposures
- \( (b_{i,t})_k \) is the exposure (a.k.a. loading) of asset \( i \) to factor \( k \) at time \( t \)
- \( f_t = [\alpha, f_{1,t}, \ldots, f_{K,t}] \) is the \( K + 1 \) vector of factor returns
- \( r_t \) is the \( N \) vector of asset returns
- \( \rho(f_t, \epsilon_t) = 0 \) and cross-sectional heteroskedasticity
  \[ \mathbb{E}[\epsilon_{t,i}^2] = \sigma_i^2, \quad i \in \{1, \ldots, N\}. \]
Consider the non-linear cross-sectional factor model:

\[ r_t = F(B_t) + \epsilon_t \]

- \( F : \mathbb{R}^K \to \mathbb{R} \) is a differentiable non-linear function
- Do not assume that \( \epsilon_t \) is Gaussian or require any other restrictions on \( \epsilon_t \), i.e. error distribution is non-parametric
- The model shall just be used to predict the next period returns only and stationarity of the factor returns is not required.
Neural Network Cross-sectional Factor Models\textsuperscript{5}

- Neural network cross-sectional factor model:

\[ r_t = F_{W,b}(B_t) + \epsilon_t \]

where \( F_{W,b}(X) \) is a neural network with \( L \) layers

\[ \hat{r} := F_{W,b}(X) = f_{W^{(L)},b^{(L)}}^{(L)} \circ \cdots \circ f_{W^{(1)},b^{(1)}}^{(1)}(X) \]

\textsuperscript{5}Dixon & Polson, Deep Fundamental Factor Models, 2019, arxiv.org/abs/1903.07677.
Experimental Setup

- Here, we define the universe as the top 250 stocks from the S&P 500 index, ranked by market cap.
- Factors are given by Bloomberg and reported monthly.
- Remove stocks with too many missing factor values, leaving 218 stocks.
- Use a 100 period experiment, fitting a cross-sectional regression model at each period and using the model to forecast the next period monthly returns.
- At each time step, the canonical experimental design over $t = 2, \ldots, T$ is
  - $(X_{t}^{\text{train}}, Y_{t}^{\text{train}}) = (B_{t-1}, r_{t-1})$
  - $(X_{t}^{\text{test}}, Y_{t}^{\text{test}}) = (B_{t}, r_{t})$
MSEs

(a) Linear Regression

(b) FFWD Neural Network
Overfitting

**Figure:** In sample MSE as a function of the number of neurons in the hidden layer. The models are trained here without L1 regularization.
Managing the Bias-Variance Tradeoff with $L_1$ Regularization

Figure: The effect of $L_1$ regularization on the MSE errors for a network with 50 neurons in the hidden layer.
Information Ratios (Random Selection of Stocks)

Figure: Random selection of $x$ stocks from the universe in each period.
**Information Ratios (In-Sample)**

![Graphs showing Information Ratios](image)

(a) Linear Regression  
(b) FFWD Neural Network

**Figure:** Selection of the top performing stocks from the universe in each in-sample period.
Information Ratios (Out-of-Sample)

Figure: Selection of the $x$ top performing stocks from the universe in each out-of-sample period.
**Figure:** The sensitivities and their uncertainty can be estimated from data (black) or derived analytically (red).
Recurrent Neural Networks

- Understand the conceptual formulation and statistical explanation of RNNs
- Understand the different types of RNNs
- Two notebook examples with limit order book data and Bitcoin history
- Navigate further learning resources
Recurrent Neural Network Predictors

• Input-output pairs $D = \{x_t, y_t\}_{t=1}^{N}$ are auto-correlated observations of $X$ and $Y$ at times $t = 1, \ldots, N$

• Construct a nonlinear times series predictor, $\hat{y}_{t+h}(\mathcal{X}_t)$, of an output, $y_{t+h}$, using a high dimensional input matrix of $p$ length sub-sequences $\mathcal{X}_t$:

$$\hat{y}_{t+h} = F(\mathcal{X}_t) \quad \text{where} \quad \mathcal{X}_t = seq_{t-p+1,t}(X) = (x_{t-p+1}, \ldots, x_t)$$

• $x_{t-j}$ is a $j^{th}$ lagged observation of $x_t$, $x_{t-j} = L^j[x_t]$, for $j = 0, \ldots, p - 1$. 
Recurrent Neural Networks (p=6)
Recurrent Neural Networks

• For each time step \( s = t - p + 1, \ldots, t \), a function \( F_h \) generates a hidden state \( z_s \):

\[
    z_s = F_h(x_s, z_{s-1}) := \sigma(W_h x_s + U_h z_{s-1} + b_h), \quad W_h \in \mathbb{R}^{H \times d}, \quad U_h \in \mathbb{R}^{H \times H}
\]

• When the output is continuous, the model output from the final hidden state is given by:

\[
    \hat{y}_{t+h} = F_y(z_t) = W_y z_t + b_y, \quad W_y \in \mathbb{R}^{K \times H}
\]

• When the output is categorical, the output is given by

\[
    \hat{y}_{t+h} = F_y(z_t) = \text{softmax}(W_y z_t + b_y)
\]

• Goal: find the weight matrices \( W = (W_h, U_h, W_y) \) and biases \( b = (b_h, b_y) \).
Univariate Example

- Consider the univariate, step-ahead, time series prediction $\hat{y}_t = F(x_{t-1})$, using $p$ previous observations $\{y_{t-i}\}_{i=1}^p$.
- Because this is a special case when no input is available at time $t$ (since we are predicting it), we form the hidden states to time $z_{t-1}$.
- The simplest case of a RNN with one hidden unit, $H = 1$, no activation function and the dimensionality of the input vector is $d = 1$. 
Univariate Example

• If $W_h = U_h = \phi, |\phi| < 1$, $W_y = 1$ and $b_h = b_y = 0$.

• Then we can show that $\hat{y}_t = F(\chi_{t-1})$ is a zero-drift auto-regressive, $AR(p)$, model with geometrically decaying weights:

$$
\begin{align*}
  z_{t-p} &= \phi y_{t-p}, \\
  z_{t-p+1} &= \phi (z_{t-p} + y_{t-p+1}), \\
  \ldots &= \ldots \\
  z_{t-1} &= \phi (z_{t-2} + y_{t-1}), \\
  y_t &= z_{t-1},
\end{align*}
$$

where

$$
\hat{y}_t = (\phi L + \phi^2 L^2 + \cdots + \phi^p L^p)[y_t].
$$
Stability

- Given a lag 1 unit innovation, $1$, the output from the hidden layer is

$$z_t = \sigma(W_h0 + U_h1 + b_h).$$

- For strict stationarity, we require that the absolute value of each component of the hidden variable under a unit impulse at lag 1 is less than 1

$$|z_t|_j = |\sigma(U_h1 + b_h)|_j < 1,$$

which is satisfied if $|\sigma(x)| \leq 1$ and each element of $U_h1 + b_h$ is finite.

- Additionally if $\sigma$ is strictly monotone increasing then $|z_t|_j$ under a lag two unit innovation is less than $|z_t|_j$ under a lag one unit innovation

$$|\sigma(U_h1 + b_h)|_j > |\sigma(U_h\sigma(U_h1 + b_h) + b_h)|_j. \quad (14)$$

In particular, the choice $\tanh$ satisfies the requirement on $\sigma$. 
Multiple Choice Question 1

Identify the following correct statements:

• A linear recurrent neural network with a memory of \( p \) lags is an autoregressive model \( AR(p) \) with non-parametric error.

• Recurrent neural networks, as time series models, are guaranteed to be stable, for any choice of weights.

• The amount of memory in a shallow recurrent network corresponds to the number of times a single perceptron layer is unfolded.

• The amount of memory in a deep recurrent network corresponds to the number of perceptron layers.
Overview

- RNNs assume that the data is covariance stationary. The autocovariance structure of the RNN is fixed and therefore so must it be in the data.

- RNNs using a version of back-propagation known as “back-propagation through time” (BPTT) which is prone to a vanishing gradient problem when the number of lags in the model is large.

- We will see extensions to these plain-RNNs which maintain a long-term memory through an exponentially smoothed hidden state.
Exponential Smoothing

- Exponential smoothing is a type of forecasting or filtering method that exponentially decreases the weight of past and current observations to give smoothed predictions $\tilde{y}_{t+1}$.
- It requires a single parameter, $\alpha$, also called the smoothing factor or smoothing coefficient.
- This parameter controls the rate at which the influence of the observations at prior time steps decay exponentially.
- $\alpha$ is often set to a value between 0 and 1. Large values mean that the model pays attention mainly to the most recent past observations, whereas smaller values mean more of the history is taken into account when making a prediction.
Exponential Smoothing

- Exponential smoothing takes the forecast for the previous period $\tilde{y}_t$ and adjusts with the forecast error, $y_t - \tilde{y}_t$. The forecast for the next period becomes:

$$\tilde{y}_{t+1} = \tilde{y}_t + \alpha(y_t - \tilde{y}_t)$$

or equivalently

$$\tilde{y}_{t+1} = \alpha y_t + (1 - \alpha)\tilde{y}_t.$$

- Writing this as a geometric decaying autoregressive series back to the first observation:

$$\tilde{y}_{t+1} = \alpha y_t + \alpha(1 - \alpha)y_{t-1} + \alpha(1 - \alpha)^2y_{t-2} + \alpha(1 - \alpha)^3y_{t-3}$$

$$+ \cdots + \alpha(1 - \alpha)^{t-1}y_1 + \alpha(1 - \alpha)^t\tilde{y}_1,$$
Exponential Smoothing

- We observe that smoothing introduces long-term model, the entire observed data, not just a sub-sequence used for prediction in a plain-RNN, for example.

- For geometrically decaying models, it is useful to characterize it by the “half-life” - the lag $k$ at which its coefficient is equal to a half:

\[ \alpha(1 - \alpha)^k = \frac{1}{2}, \]

or

\[ k = -\frac{\ln(2\alpha)}{\ln(1 - \alpha)}. \]
Dynamic Smoothing

- Dynamic exponential smoothing is a time dependent, convex, combination of the smoothed output, $\tilde{y}_t$, and the observation $y_t$:

$$\tilde{y}_{t+1} = \alpha_t y_t + (1 - \alpha_t)\tilde{y}_t,$$

where $\alpha_t \in [0, 1]$ denotes the dynamic smoothing factor which can be equivalently written in the one-step-ahead forecast of the form

$$\tilde{y}_{t+1} = \tilde{y}_t + \alpha_t(y_t - \tilde{y}_t).$$

- Hence the smoothing can be viewed as a form of dynamic forecast error correction; When $\alpha_t = 1$, the one-step ahead forecast merely repeats the current observation $y_t$, and when $\alpha_t = 0$, the forecast error is ignored and the forecast is the current model output.
Dynamic Smoothing

- The smoothing can also be viewed a weighted sum of the lagged observations, with lower or equal weights, $\alpha_{t-s} \prod_{r=1}^{s}(1 - \alpha_{t-r+1})$ at the lag $s \geq 1$ past observation, $y_{t-s}$:

$$
\tilde{y}_{t+1} = \alpha_{t} y_{t} + \sum_{s=1}^{t-1} \alpha_{t-s} \prod_{r=1}^{s}(1 - \alpha_{t-r+1})y_{t-s} + \prod_{r=0}^{t-1}(1 - \alpha_{t-r})\tilde{y}_{1},
$$

where the last term is a time dependent constant and typically we initialize the exponential smoother with $\tilde{y}_{1} = y_{1}$.

- Note that for any $\alpha_{t-r+1} = 1$, the prediction $\tilde{y}_{t+1}$ will have no dependency on all lags $\{y_{t-s}\}_{s \geq r}$. The model simply forgets the observations at or beyond the $r^{th}$ lag.

- In the special case when the smoothing is constant and equal to 1 - $\alpha$, then the above expression simplifies to

$$
\tilde{y}_{t+1} = \alpha^{-1} y_{t},
$$
Dynamic Smoothing

Putting this together gives the following dynamic RNN:

\[
\text{smoothing: } \tilde{h}_t = \hat{\alpha}_t \hat{h}_t + (1 - \hat{\alpha}_t) \tilde{h}_{t-1} \quad (15)
\]

\[
\text{smoother update: } \hat{\alpha}_t = \sigma^{(1)}(U_\alpha \tilde{h}_{t-1} + W_\alpha x_t + b_\alpha) \quad (16)
\]

\[
\text{hidden state update: } \hat{h}_t = \sigma(U_h \tilde{h}_{t-1} + W_h x_t + b_h), \quad (17)
\]

where \(\sigma^{(1)}\) is a sigmoid or heaviside function and \(\sigma\) is any activation function.
Figure: An illustrative example of the response of a simplified GRU and comparison with a plain-RNN and a RNN with an exponentially smoothed hidden state, under a constant $\alpha$. The RNN loses memory of the unit impulse after three lags, whereas the RNNs with smooth hidden states maintain memory of the first unit impulse even when the second unit impulse arrives. The difference between the dynamically smoothed RNN (the toy GRU) and smoothed RNN with a fixed smoothing parameter appears insignificant. Keep in mind however that the dynamical smoothing model has much more flexibility in how controlling the sensitivity of the smoothing to the unit impulses.
Gated Recurrent Units (GRUs)

- We introduce a reset, or switch, $r_t$, in order to forget the dependence of $\hat{h}_t$ on previous data.
- Effectively, we turn $\hat{h}_t$ from a RNN to a FNN.

\[
\begin{align*}
    h_t &= \hat{Z}_t \hat{h}_t + (1 - \hat{Z}_t) h_{t-1} \\
    \hat{Z}_t &= \sigma(U_Z h_{t-1} + W_Z x_t + b_Z) \\
    \hat{h}_t &= \tanh(U_\hat{h} \hat{r}_t \cdot h_{t-1} + W_h x_t + b_h) \\
    \hat{r}_t &= \sigma(U_r h_{t-1} + W_r x_t + b_r) \\
    \hat{y}_t &= W_Y h_t + b_Y
\end{align*}
\]

- $\hat{r}_t$ is the switching (reset) parameter which determines how much recurrence is used for the candidate hidden state $\hat{h}_t$. 

References


A linear recurrent neural network with a memory of \( p \) lags is an autoregressive model \( AR(p) \) with non-parametric error.

Recurrent neural networks, as time series models, are guaranteed to be stable, for any choice of weights.

The amount of memory in a shallow recurrent network corresponds to the number of times a single perceptron layer is unfolded.

The amount of memory in a deep recurrent network corresponds to the number of perceptron layers.

Answer: 1, 2, 3
Learning Objectives

• Understand the conceptual formulation and statistical explanation of CNNs.

• Understand 1D CNNs for time series prediction.

• A notebook example on univariate time series prediction with 1D CNNs.

• Understand how dilation CNNs capture multi-scales.

• Understand 2D CNNs for image processing. Supplementary notebook.
Multiple Choice Question 2

Identify the following correct statements:

• A feedforward network can, in principle, always be used for any supervised learning problem.

• A linear recurrent neural network with a memory of \( p \) lags is an autoregressive model \( AR(p) \).

• Recurrent neural networks, as time series models, are guaranteed to be stable, for any choice of weights and activation.

• Feedforward and recurrent neural network predictors require that the data is stationary.

• GRUs allow the hidden state to persist beyond a sequence by using smoothing.
Multiple Choice Question 2

Identify the following correct statements:

- A feedforward network can, in principle, always be used for any supervised learning problem.
- A linear recurrent neural network with a memory of $p$ lags is an autoregressive model $AR(p)$.
- Recurrent neural networks, as time series models, are guaranteed to be stable, for any choice of weights and activation.
- Feedforward and recurrent neural predictors require that the data is stationary.
- GRUs allow the hidden state to persist beyond a sequence by using smoothing.

Answer: 1,2,4,5.
Convolutional neural networks (CNN)

- Convolutional neural networks (CNN) are feedforward neural networks that can exploit local spatial structures in the input data.
- CNNs attempt to reduce the network size by exploiting spatial or temporal locality.
- CNNs use sparse connections between the different layers.
- Convolution is frequently used for image processing, such as for smoothing, sharpening, and edge detection of images.
Weighted Moving Average Smoothers

- A common technique in time series analysis and signal processing is to filter the time series.
- We have already seen exponential smoothing as a special case of a class of smoothers known as “weighted moving average (WMA)” smoothers.
- WMA smoothers take the form
  \[ \tilde{x}_t = \frac{1}{\sum_{i \in I} w_i} \sum_{i \in I} w_i x_{t-i} \]
  where \( \tilde{x}_t \) is the local mean of the time series. The weights are specified to emphasize or deemphasize particular observations of \( x_{t-i} \) in the span \(|I|\).
- Examples of well known smoothers include the Hanning smoother \( h(3) \):
  \[ \tilde{x}_t = (x_{t-1} + 2x_t + x_{t+1})/4. \]
- Such smoothers have the effect of reducing noise in the time series.
Moving Average Smoothers

Figure: Example of a moving average filter applied to time series.
Convolutions for Time Series

• The moving average filter is in fact a (discrete) convolution using a very simple filter kernel.

• The simplest discrete convolution gives the relation between $x_i$ and $x_j$:

$$x_t - i = \sum_{j=0}^{t-1} \delta_{ij} x_t - j, \ i \in \{0, \ldots, t - 1\}$$

where we have used the Kronecker delta $\delta$.

• The kernel filtered time series is a convolution\(^7\)

$$\tilde{x}_t - i = \sum_{j \in J} K_{j + k + 1} x_t - i - j, \ i \in \{k + 1, \ldots, p - k\},$$

\(^7\)For simplicity, the ends of the sequence are assumed to be unsmoothed but for notational reasons we set $\tilde{x}_t - i = x_t - i$ for $i \in \{1, \ldots, k, p - k + 1, \ldots, p\}$. 

The filtered $AR(p)$ model is

$$\hat{x}_t = \mu + \sum_{i=1}^{p} \phi_i \tilde{x}_{t-i}$$

(23)

$$= \mu + (\phi_1 L + \phi_2 L^2 + \cdots + \phi_p L^p)[\tilde{x}_t]$$

(24)

$$= \mu + [L, L^2, \ldots, L^p] \phi[\tilde{x}_t],$$

(25)

with coefficients $\phi := [\phi_i, \ldots, \phi_p]^T$.  

---

Note that there is no look-ahead bias because we do not smooth the last $k$ values of the observed data $\{x_s\}_{s=1}^t$. 

---

\(^8\)Note that there is no look-ahead bias because we do not smooth the last $k$ values of the observed data $\{x_s\}_{s=1}^t$. 

---
1D CNN with one hidden unit

- A simple 1D CNN consisting of a feedforward output layer and a non-activated hidden layer with one unit (i.e. kernel):

\[ \hat{x}_t = W_y z_t + b_y, \quad z_t = [\tilde{x}_{t-1}, \ldots, \tilde{x}_{t-p}]^T, \quad W_y = \phi^T, \quad b_y = \mu, \]

where \( \tilde{x}_{t-i} \) is the \( i^{th} \) output from a convolution of the \( p \) length input sequence with a kernel consisting of \( 2k + 1 \) weights.

- These weights are fixed over time and hence the CNN is only suited to prediction from stationary time series.

- Note also, in contrast to a RNN, that the size of the weight matrix \( W_y \) increases with the number of lags in the model.
1D CNN with H hidden units

- The univariate CNN predictor with \( p \) lags and \( H \) activated hidden units (kernels) is

\[
\hat{x}_t = W_y \text{vec}(z_t) + b_y \tag{26}
\]
\[
[z_t]_{i,m} = \sigma \left( \sum_{j \in J} K_{m,j+k+1} x_{t-i-j} + [b_h]_m \right) \tag{27}
\]
\[
= \sigma(K \ast x_t + b_h) \tag{28}
\]

where \( m \in \{1, \ldots, H\} \) denotes the index of the kernel and the kernel matrix \( K \in \mathbb{R}^{H \times 2k+1} \), hidden bias vector \( b_h \in \mathbb{R}^{H} \) and output matrix \( W_y \in \mathbb{R}^{1 \times pH} \).
Remarks on CNNs

- Since the size of $W_y$ increases with both the number of lags and the number of kernels, it may be preferable to reduce the dimensionality of the weights with an additional layer and hence avoid over-fitting.

- Convolutional neural networks are not limited to sequential models. One might, for example, sample the past lags non-uniformly so that $l = \{2^i\}_{i=1}^{p}$ then the maximum lag in the model is $2^p$. Such a non-sequential model allows a large maximum lag without capturing all the intermediate lags.
Stationarity

• A univariate linear CNN predictor, with one kernel and no activation, can be written in the canonical form

\[
\hat{x}_t = \mu + (1 - \Phi(L))[K \ast x_t] = \mu + K \ast (1 - \Phi(L))[x_t] \quad (29)
\]

\[
= \mu + (\tilde{\phi}_1 L + \ldots \tilde{\phi}_p L^p)[x_t] := \mu + (1 - \tilde{\Phi}(L))[x_t], \quad (30)
\]

where, by the linearity of \( \Phi(L) \) in \( x_t \), the convolution commutes and thus we can write \( \tilde{\phi} := K \ast \phi \).

• Provided that \( \tilde{\Phi}(L)^{-1} \) forms a divergent sequence in the noise process \( \{\epsilon_s\}_{s=1}^t \) then the model is stable.

• Finding the roots of the characteristic equation

\[
\tilde{\Phi}(z) = 0,
\]

it follows that the linear CNN is strictly stationary and ergodic if all the roots lie outside the unit circle in the complex plane,

\[
|\lambda_i| > 1, \; i \in \{1, \ldots, p\}.
\]
**Equivalent Eigenvalue Problem**

- Finding roots of polynomials is equivalent to finding eigenvalues of the “companion matrix”.
- A famous theorem states that the roots of any polynomial can be found by turning it into a matrix and finding the eigenvalues.
- Given the $p$ degree polynomial:
  \[ q(z) = c_0 + c_1 z + \ldots + c_{p-1} z^{p-1} + z^p, \]
  we define the $p \times p$ companion matrix
  \[
  C := \begin{pmatrix}
  0 & 1 & 0 & \ldots & 0 \\
  0 & 0 & 1 & 0 & \vdots \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  0 & 0 & 0 & 0 & 1 \\
  -c_0 & -c_1 & \ldots & -c_{p-2} & -c_{p-1}
  \end{pmatrix}
  \]
  then the characteristic polynomial $\text{det}(C - \lambda I) = q(\lambda)$, and so the eigenvalues of $C$ are the roots of $q$. \(^{10}\)
Stridred Convolution

• In the above slides, we performed the convolution by sliding the image area by a unit increment. This is known as using a **stride**, $s$, of one$^{11}$:

• More generally, given an integer $s \geq 1$, a convolution with stride $s$ for input (feature) $f \in \mathbb{R}^m$ is defined as:

$$[K *_s f]_i = \sum_{p=-k}^{k} K_pf_{s(i-1)+1+p}, \quad i \in \{1, \ldots, \left\lceil \frac{m}{s} \right\rceil \}.$$  \hspace{1cm} (31)

Here $\left\lceil \frac{m}{s} \right\rceil$ denotes the smallest integer greater than $\frac{m}{s}$.

---

$^{11}$A common choice in CNNs is to take $s = 2$
Dilated Convolution

- In addition to image processing, CNNs have also been successfully applied to time series. WaveNet, for example, is a CNN developed for audio processing.
- Time series often display long-term correlations. Moreover, the dependent variable(s) may exhibit non-linear dependence on the lagged predictors.
- Recall that a CNN for times series is a non-linear $p$-auto-regression of the form

$$y_t = \sum_{i=1}^{p} \phi_i(x_{t-i}) + \epsilon_t$$

(32)

where the coefficient functions $\phi_i$, $i \in \{1, \ldots, p\}$ are data-dependent and optimized through the convolutional network.
- A dilated convolution effectively allows the network to operate on a coarser scale than with a normal convolution. This is similar to pooling or stridded convolutions, but here the output has the same size as the input.
Dilated convolution

- In a dilated convolution the filter is applied to every $d^{th}$ element in the input vector, allowing the model to efficiently learn connections between far-apart data points.

- For an architecture with $L$ layers of dilated convolutions $\ell \in \{1, \ldots, L\}$, a dilated convolution outputs a stack of “feature maps” given by

$$[K^{(\ell)} *_{d^{(\ell)}} f^{(\ell-1)}]_i = \sum_{p=-k}^k K^{(\ell)}_p f^{(\ell-1)}_{d^{(\ell)}(i-1)+1+p}, \quad i \in \{1, \ldots, \lceil \frac{m}{d^{(\ell)}} \rceil \}.$$  \hspace{1cm} (33)

where $d$ is the dilation factor and we can choose the dilations to increase by a factor of two: $d^{(\ell)} = 2^{\ell-1}$.

- The filters for each layer, $K^{(\ell)}$, are chosen to be of size $1 \times k' = 1 \times 2$, where $k' := 2k + 1$.
Dilated Convolution

- The output is the forecasted time series \( \hat{Y} = (\hat{y}_t)_{t=0}^{N-1} \).

\[ r = 2^{L-1} k'. \tag{34} \]

Figure: A dilated convolutional neural network with three layers. The receptive field is given by \( r = 8 \), i.e. one output value is influenced by eight input neurons.

- The receptive field depends on the number of layers \( L \) and the filter size \( k' \), and is given by
Learning Objectives

• Understand principal component analysis for dimension reduction.
• Understand how to formulate a linear autoencoder.
• Understand how a linear autoencoder performs PCA.
• Notebook example using dimension reduction on the yield curve.
Preliminaries

- Let \( \{y_i\}_{i=1}^N \) be a set of \( N \) observation vectors, each of dimension \( n \). We assume that \( n \leq N \).
- Let \( Y \in \mathbb{R}^{n \times N} \) be a matrix whose columns are \( \{y_i\}_{i=1}^N \),
  \[
  Y = \begin{bmatrix}
  y_1 & \cdots & y_N \\
  \end{bmatrix}.
  \]
- The element-wise average of the \( N \) observations is an \( n \) dimensional signal which may be written as:
  \[
  \bar{y} = \frac{1}{N} \sum_{i=1}^{N} y_i = \frac{1}{N} Y 1_N,
  \]
  where \( 1_N \in \mathbb{R}^{N \times 1} \) is a column vector of all-ones.
- Let \( Y_0 \) be a matrix whose columns are the demeaned observations (we center each observation \( y_i \) by subtracting \( \bar{y} \) from it):
  \[
  Y_0 = Y - \bar{y} 1_N^T.
  \]
Projection

- A linear projection from $\mathbb{R}^m$ to $\mathbb{R}^n$ is a linear transformation of a finite dimensional vector given by a matrix multiplication:

$$x_i = W^T y_i,$$

where $y_i \in \mathbb{R}^n$, $x_i \in \mathbb{R}^m$, and $W \in \mathbb{R}^{n \times m}$.

- Each element $j$ in the vector $x_i$ is an inner product between $y_i$ and the $j$-th column of $W$, which we denote by $w_j$.

- Let $X \in \mathbb{R}^{m \times N}$ be a matrix whose columns are the set of $N$ vectors of transformed observations, let $\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i = \frac{1}{N} X \mathbf{1}_N$ be the element-wise average, and $X_0 = X - \bar{x} \mathbf{1}_N^T$ the de-meaned matrix.

- Clearly, $X = W^T Y$ and $X_0 = W^T Y_0$. 

 Principal Component Projection*

- When the matrix $W^T$ represents the transformation that applies principal component analysis, we denote $W = P$, and the columns of the orthonormal matrix\textsuperscript{12}, $P$, denoted $\{p_j\}_{j=1}^n$, are referred to as \textit{loading vectors}.

- The transformed vectors $\{x_i\}_{i=1}^N$ are referred to as \textit{principal components} or \textit{scores}.

- The first loading vector is defined as the unit vector with which the inner products of the observations have the greatest variance:

$$p_1 = \max_{w_1} w_1^T Y_0 Y_0^T w_1 \text{ s.t. } w_1^T w_1 = 1.$$  \hspace{1cm} (35)

- The solution to (35) is known to be the eigenvector of the sample covariance matrix $Y_0 Y_0^T$ corresponding to its largest eigenvalue\textsuperscript{13}

\textsuperscript{12}i.e. $P^{-1} = P^T$.

\textsuperscript{13}We normalize the eigenvector and disregard its sign.
Next, \( p_2 \) is the unit vector which has the largest variance of inner products between it and the observations after removing the orthogonal projections of the observations onto \( p_1 \). It may be found by solving:

\[
P_2 = \max_{w_2} w_2^T \left( Y_0 - p_1 p_1^T Y_0 \right) \left( Y_0 - p_1 p_1^T Y_0 \right)^T w_2 \text{ s.t. } w_2^T w_2 = 1.
\]

The solution to (36) is known to be the eigenvector corresponding to the largest eigenvalue under the constraint that it is not collinear with \( p_1 \).

Similarly, the remaining loading vectors are equal to the remaining eigenvectors of \( Y_0 Y_0^T \) corresponding to descending eigenvalues.
Principal Components

- The eigenvalues of $Y_0 Y_0^T$, which is a positive semi-definite matrix, are non-negative.
- They are not necessarily distinct, but since it is a symmetric matrix it has $n$ eigenvectors that are all orthogonal, and it is always diagonalizable.
- Thus, the matrix $P$ may be computed by diagonalizing the covariance matrix:

$$Y_0 Y_0^T = P \Lambda P^{-1} = P \Lambda P^T,$$

where $\Lambda = X_0 X_0^T$ is a diagonal matrix whose diagonal elements $\{\lambda_i\}_{i=1}^n$ are sorted in descending order.
- The transformation back to the observations is $Y = PX$.
- The fact that the covariance matrix of $X$ is diagonal means that PCA is a decorrelation transformation and is often used to denoise data.
Dimensionality reduction

- PCA is often used as a method for dimensionality reduction, the process of reducing the number of variables in a model in order to avoid the curse of dimensionality.

- PCA gives the first $m$ principal components ($m < n$) by applying the truncated transformation

$$X_m = P_m^T Y,$$

where each column of $X_m \in \mathbb{R}^{m \times N}$ is a vector whose elements are the first $m$ principal components, and $P_m$ is a matrix whose columns are the first $m$ loading vectors,

$$P_m = \begin{bmatrix} p_1 & \cdots & p_m \end{bmatrix} \in \mathbb{R}^{n \times m}.$$

- Intuitively, by keeping only $m$ principal components, we are losing information, and we minimize this loss of information by maximizing their variances.
Minimum total squared reconstruction error

- $P_m$ is also a solution to:

$$\min_{W \in \mathbb{R}^{n \times m}} \left\| Y_0 - WW^T Y_0 \right\|_F^2 \quad s.t. \quad W^T W = I_{m \times m}, \quad (37)$$

where $F$ denotes the Frobenius matrix norm.

- The $m$ leading loading vectors are an orthonormal basis which spans the $m$ dimensional subspace onto which the projections of the demeaned observations have the minimum squared difference from the original demeaned observations.

- In other words, $P_m$ compresses each demeaned vector of length $n$ into a vector of length $m$ (where $m \leq n$) in such a way that minimizes the sum of total squared reconstruction errors.

- The minimizer of (37) is not unique: $W = P_m Q$ is also a solution, where $Q \in \mathbb{R}^{m \times m}$ is any orthogonal matrix, $Q^T = Q^{-1}$. Multiplying $P_m$ from the right by $Q$ transforms the first $m$ loading vectors into a different orthonormal basis for the same subspace.
Minimum total squared reconstruction error

- A neural network that is trained to learn the identity function $f(Y) = Y$ is called an autoencoder. Its output layer has the same number of nodes as the input layer, and the cost function is some measure of the reconstruction error.
- Autoencoders are unsupervised learning models, and they are often used for the purpose of dimensionality reduction.
- A simple autoencoder that implements dimensionality reduction is a feed-forward autoencoder with at least one layer that has a smaller number of nodes, which functions as a bottleneck.
- After training the neural network using backpropagation, it is separated into two parts: the layers up to the bottleneck are used as an encoder, and the remaining layers are used as a decoder.
- In the simplest case, there is only one hidden layer (the bottleneck), and the layers in the network are fully connected.
Linear autoencoders*

In the case that no non-linear activation function is used, $x_i = W^{(1)} y_i + b^{(1)}$ and $\hat{y}_i = W^{(2)} x_i + b^{(2)}$. If the cost function is the total squared difference between output and input, then training the autoencoder on the input data matrix $Y$ solves:

$$\min_{W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}} \left\| Y - \left( W^{(2)} \left( W^{(1)} Y + b^{(1)} 1_N^T \right) + b^{(2)} 1_N^T \right) \right\|^2_F.$$  \hspace{1cm} (38)

If we set the partial derivative with respect to $b_2$ to zero and insert the solution into (38), then the problem becomes:

$$\min_{W^{(1)}, W^{(2)}} \left\| Y_0 - W^{(2)} W^{(1)} Y_0 \right\|^2_F$$

Thus, for any $b_1$, the optimal $b_2$ is such that the problem becomes independent of $b_1$ and of $\bar{y}$. Therefore, we may focus only on the weights $W^{(1)}$, $W^{(2)}$. 
Linear autoencoders as orthogonal projections

- Setting the gradients to zero, $W^{(1)}$ is the left Moore-Penrose pseudoinverse of $W^{(2)}$ (and $W^{(2)}$ is the right pseudoinverse of $W^{(1)}$):

  $$W^{(1)} = (W^{(2)})^\dagger = \left(W^{(2)}^T W^{(2)}\right)^{-1} (W^{(2)})^T$$

- The minimization with respect to a single matrix is:

  $$\min_{W^{(2)} \in \mathbb{R}^{n \times m}} \left\| Y_0 - W^{(2)} (W^{(2)})^\dagger Y_0 \right\|_F^2 \tag{39}$$

  The matrix $W^{(2)} (W^{(2)})^\dagger = W^{(2)} ((W^{(2)})^T W^{(2)})^{-1} (W^{(2)})^T$ is the orthogonal projection operator onto the column space of $W^{(2)}$ when its columns are not necessarily orthonormal.

- This problem is very similar to (37), but without the orthonormality constraint.
Linear autoencoders: mathematical properties*

- It can be shown that $W^{(2)}$ is a minimizer of (39) if and only if its column space is spanned by the first $m$ loading vectors of $Y$.
- The linear autoencoder is said to apply PCA to the input data in the sense that its output is a projection of the data onto the low dimensional principal subspace.
- However, unlike actual PCA, the coordinates of the output of the bottleneck are correlated and are not sorted in descending order of variance.
- The solutions for reduction to different dimensions are not nested: when reducing the data from dimension $n$ to dimension $m_1$, the first $m_2$ vectors ($m_2 < m_1$) are not an optimal solution to reduction from dimension $n$ to $m_2$, which therefore requires training an entirely new autoencoder.
Equivalence of linear autoencoders and PCA*

Hypothesis: The first \( m \) loading vectors of \( Y \) are the first \( m \) left singular vectors of the matrix \( W^{(2)} \) which minimizes (39).

- Train the linear autoencoder on the original dataset \( Y \) and then compute the first \( m \) left singular vectors of \( W^{(2)} \in \mathbb{R}^{n \times m} \), where typically \( m \ll N \).
- The loading vectors may also be recovered from the weights of the hidden layer by a singular value decomposition, \( W^{(1)} \).
- If \( W^{(2)} = U \Sigma V^T \), which we assume is full-rank, then

\[
W^{(1)} = (W^{(2)})^\dagger = V \Sigma^\dagger U^T
\]

and

\[
W_2 W_2^\dagger = U \Sigma V^T V \Sigma^\dagger U^T = U \Sigma \Sigma^\dagger U^T = U_m U_m^T, \tag{40}
\]

where we used the fact that \( (V^T)^\dagger = V \) and that \( \Sigma^\dagger \in \mathbb{R}^{m \times n} \) is a matrix whose diagonal elements are \( \frac{1}{\sigma_j} \) (assuming \( \sigma_j \neq 0 \), and 0 otherwise).
**Figure:** This figure shows the yield curve over time, each line corresponds to a different maturity in the term-structure of interest rates.
Figure: The covariance matrix of the data in the transformed coordinates, according to (a) the loading vectors computed by applying SVD to the entire dataset, (b) the weights of the linear autoencoder, and (c) the left singular vectors of the autoencoder weights.
Figure: The first two principal components of $\Delta Y_0$, projected using $P_m$ are shown in (a). The first two approximated principal components (up to a sign change) using $U_m$. The first principal component is represented by the x-axis and the second by the y-axis.
Summary

- Deep Learning for cross-sectional data
  - Background: review of supervised machine learning
  - Introduction to feedforward neural networks
  - Related approximation and learning theory
  - Training and back-propagation
  - Interpretability with factor modeling example

- Deep Learning for time-series data prediction
  - Understand the conceptual formulation and statistical explanation of RNNs
  - Understand the different types of RNNs, including plain RNNs, GRUs and LSTMs
  - Two notebook examples with limit order book data and Bitcoin history
  - Understand the conceptual formulation and statistical explanation of CNNs.
  - Understand 1D CNNs for time series prediction.
Summary

- Deep Learning for dimensionality reduction
  - Understand principal component analysis for dimension reduction.
  - Understand how to formulate a linear autoencoder.
  - Understand how a linear autoencoder performs PCA.
  - Notebook example using dimension reduction on the yield curve.
Review of time series analysis

- Characterize the properties of time series
- Problem formulation in time series prediction
- Practical challenges that frequently arise in time series prediction
**Time Series Data**

Time series data:

- Consists of observations on a variable over time, e.g. stock prices
- Observations are not assumed to be independent over time, rather observations are often strongly related to their recent histories.
- For this reason, the ordering of the data matters (unlike cross-sectional data).
- Pay special attention to the *data frequency*. 
Classical Time Series Analysis

- Box-Jenkins linear time series analysis, e.g. ARIMA
- Non-linear time series analysis, e.g. GARCH
- Dynamic state based models, e.g. Kalman filters.
Autoregressive Processes \( AR(p) \)

- The \( p^{th} \) order autoregressive process of a variable \( y \) depends only on the previous values of the variable plus a white noise disturbance term

\[
y_t = \mu + \sum_{i=1}^{p} \phi_i y_{t-i} + \epsilon_t
\]

- Defining the polynomial function \( \phi(L) := (1 - \phi_1 L - \phi_2 L^2 - \cdots - \phi_p L^p) \), the AR(p) process can be expressed in the more compact form

\[
\phi(L)y_t = \mu + \epsilon_t
\]

**Key point:** an AR(p) process has a geometrically decaying acf.
Stationarity

**Stationarity**

- An important property of AR(p) processes is whether they are stationary or not. A process which is stationary will have error disturbance terms $\epsilon_t$ will have a declining impact on the current value of $y$ as the lag increases.

- Conversely, non-stationary AR(p) processes exhibit the counter-intuitive behavior that the error disturbance terms become increasingly influential as the lag increases. If the roots of the characteristic equation

$$\phi(z) = (1 - \phi_1 z - \phi_2 z^2 - \cdots - \phi_p z^p) = 0$$

*all* lie outside the unit circle, then the AR(p) process is stationary.
Exercise 1

Is the following random walk (zero mean AR(1) process) stationary?

\[ y_t = y_{t-1} + \epsilon_t \]
Exercise 2

Compute the mean, variance and autocorrelation function of the following zero-mean AR(1) process:

\[ y_t = \phi_1 y_{t-1} + \epsilon_t. \]
MA(q) process The $q^{th}$ order moving average process is the linear combination of the white noise process $\{\epsilon_t\}_{t=1}^{q+1}$

$$y_t = \mu + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} + \epsilon_t$$

It’s convenient to define the Lag or Backshift operator $L^i y_t := y_{t-i}$ and write the MA(q) process as

$$y_t = \mu + \theta(L) \epsilon_t$$

where the polynomial function $\theta(L) := 1 + \theta_1 L + \theta_2 L^2 + \cdots + \theta_q L^q$. 
The distinguishing properties of the $MA(q)$ process are the following:

- **Constant mean:** $\mathbb{E}[y_t] = \mu$
- **Constant variance:** $\mathbb{V}[y_t] = (1 + \theta_1^2 + \theta_2^2 + \cdots + \theta_q^2)\sigma^2$
- **Autocovariances which may be non-zero to lag $q$ and zero thereafter:**
  \[
  \gamma_s = \begin{cases} 
  (\theta_s + \theta_{s+1}\theta_1 + \theta_{s+2}\theta_2 + \cdots + \theta_q\theta_{q-s})\sigma^2 & s = 1, 2, \ldots, q, \\
  0 & s > q.
  \end{cases}
  \]

**Key point:** a $MA(q)$ process has non-zero points of acf $\equiv q$. Above $q$ there is a sharp cut-off.
Consider the following zero-mean MA(2) process:

\[ y_t = \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} \]

Show that:

1. \( \mathbb{E}[y_t] = 0, \forall t \)
2. \( \mathbb{V}[y_t] = \gamma_0 = (1 + \theta_1^2 + \theta_2^2) \sigma^2, \forall t \)
3. \( \epsilon_1 = \frac{\theta_1 + \theta_1 \theta_2}{1 + \theta_1^2 + \theta_2^2}, \epsilon_2 = \frac{\theta_2}{1 + \theta_1^2 + \theta_2^2} \) and \( \epsilon_s = 0, \ s > q = 2. \)

If \( \theta_1 = -0.5 \) and \( \theta_2 = 0.25 \), sketch the autocorrelation function (acf) of the sample MA(2) process.
The $p^{th}, q^{th}$ order ARMA process of a variable $y$ combines the AR($p$) and MA($q$) processes:

\[ \phi(L)y_t = \mu + \theta(L)\epsilon_t \]

where

\[
\phi(L) := (1 - \phi_1 L - \phi_2 L^2 - \cdots - \phi_p L^p) \\
\theta(L) := 1 + \theta_1 L + \theta_2 L^2 + \cdots + \theta_q L^q
\]

The mean of a ARMA series depends only on the coefficients of the autoregressive process and not the coefficients of the moving average process:

\[ \mathbb{E}[y_t] = \frac{\mu}{1 - \phi_1 - \phi_2 - \cdots - \phi_p} \]
Forecasting with a $MA(q)$ model

- Predict the value $y_{t+1}$ given information $\Omega_t$ at time $t$ using the equation:

$$y_{t+1} = \mu + \theta_1 \epsilon_t + \theta_2 \epsilon_{t-1} + \theta_3 \epsilon_{t-2} + \epsilon_{t+1}$$

- The forecast is the conditional expectation of the prediction. The one-step ahead forecast at time $t$ is denoted $f_{t,1}$:

$$f_{t,1} = \mathbb{E}[y_{t+1}|\Omega_t] = \mu + \theta_1 \epsilon_t + \theta_2 \epsilon_{t-1} + \theta_3 \epsilon_{t-2}$$

since only the conditional expectation $\mathbb{E}[\epsilon_{t+1}|\Omega_t] = 0$.

- The two-step ahead forecast $f_{t,2}$ is given by:

$$f_{t,2} = \mathbb{E}[y_{t+2}|\Omega_t] = \mu + \theta_2 \epsilon_t + \theta_3 \epsilon_{t-1}$$

- In general, a $MA(q)$ process has a memory of $q$ periods, so forecasts $q + 1$ steps ahead collapse to an intercept $\mu$. 
Forecasting with a $AR(p)$ model

- Recall that an AR($p$) process has infinite memory.
- In general, the $s$-step ahead forecast for an AR(2) process is given by

$$f_{t,s} = \mathbb{E}[y_{t+s} | \Omega_t] = \mu + \phi_1 f_{t,s-1} + \phi_2 f_{t,s-2}$$

thus we need to compute the one-period lag and two-period lag first.

- $ARMA(p,q)$: Putting the MA($q$) and AR($p$) $s$-step ahead forecasting equations together, we get

$$f_{t,s} = \sum_{i=1}^{p} a_i f_{t,s-i} + \sum_{j=1}^{q} b_j \epsilon_{t+s-j}$$

where $f_{t,s} = y_{t+s}, s \leq 0$ and $\epsilon_{t+s} = 0, s > 0$. 
Exercise 4

Consider the following ARMA(2,2) 1-step ahead forecast

\[ f_{t,1} = a_1 f_{t,0} + a_2 f_{t,-1} + b_1 \epsilon_t + b_2 \epsilon_{t-1} \]  
\[ = a_1 y_t + a_2 y_{t-1} + b_1 \epsilon_t + b_2 \epsilon_{t-1} \]  

where \( a_i \) and \( b_j \) are the autoregressive and moving average coefficients respectively.
Parametric tests: time series analysis

- Augmented Dickey Fuller test for time series stationarity, with the Schwert (heuristic) estimate for the maximum lag
- Durbin-Watson tests for co-integration of two variables
- Granger causality tests
Identification

- it may be difficult to observe the order of the MA and AR processes from the acf and pacf plots.
- It is often preferable to use the Akaike Information Criteria (AIC) to measure the quality of fit:

\[
AIC = \ln(\hat{\sigma}^2) + \frac{2k}{T}
\]

where \(\hat{\sigma}^2\) is the residual variance (the residual sums of squares divided by the number of observations \(T\)) and \(k = p + q + 1\) is the total number of parameters estimated.

- This criterion expresses a trade-off between the first term, the quality of fit, and the second term, a penalty function proportional to the number of parameters.
- Adding more parameters to the model reduces the residuals but increases the right hand-term, thus the AIC favors the best fit with the fewest number of parameters.

AIC is limited to asymptotic normality assumptions on the error.
In Sample: Diagnostics

- A fitted model must be examined for underfitting with a white noise test. Box and Pierce propose the Portmanteau statistic
  \[ Q^*(m) = T \sum_{l=1}^{m} \hat{\rho}_l^2 \]
- As a test statistic for the null hypothesis
  \[ H_0: \rho_1 = \cdots = \rho_m = 0 \]
  against the alternative hypothesis
  \[ H_a: \rho_i \neq 0 \]
  for some
  \[ i \in \{1, \ldots, m\} \]
  \( \hat{\rho}_i \) are the sample autocorrelations of the residual.
- The Box-Pierce statistic follows an asymptotically chi-squared distribution with \( m - p - q \) degrees of freedom.
  The decision rule is to reject \( H_0 \) if \( Q(m) > \chi^2_{\alpha} \) where \( \chi^2_{\alpha} \) denotes the 100(1 - \( \alpha \))th percentile of a chi-squared distribution with \( m \) degrees of freedom and is the significance level for rejecting \( H_0 \).
• The Ljung-Box test statistic increases the power of the test in finite samples:

\[ Q(m) = T(T + 2) \sum_{l=1}^{m} \frac{\hat{\rho}_l^2}{T - l} \]

• This statistic also follows an asymptotically chi-squared distribution with \( m \) degrees of freedom.

• The Ljung-Box statistic follows asymptotically a chi-squared distribution with \( m - p - q \) degrees of freedom.
**Figure:** Times series cross validation, also referred to as walk forward optimization, is used to avoid introducing look-ahead bias into the model.
Forecasting Accuracy

• The forecasting accuracy may be measured using the mean square error (MSE) or the mean absolute error (MAE) of the out-of-sample forecasts with the actual observations.

• Let’s suppose that we have an trained the model with $T$ observations, then the MSE evaluated over the next $m$ observations:

$$MSE = \frac{1}{m} \sum_{t=0}^{m-1} (y_{T+t+s} - f_{T+t,s})^2$$

• The MAE is given by

$$MAE = \frac{1}{m} \sum_{t=0}^{m-1} |y_{T+t+s} - f_{T+t,s}|$$

**Question:** Why use the MSE in preference to the MAE?
Predicting Events

- Suppose we have conditionally i.i.d. Bernoulli r.v.s $X_t$ with $p_t := P[X_t = 1|\Omega_t]$ representing a binary event
- $\mathbb{E}[X_t | \Omega] = 0 \cdot (1 - p_t) + 1 \cdot p_t = p_t$
- $\mathbb{V}[X_t | \Omega] = p_t(1 - p_t)$
- Under an ARMA model:
  \[
  \ln \left( \frac{p_t}{1 - p_t} \right) = \phi^{-1}(L)(\mu + \theta(L)\epsilon_t)
  \]
Entropy

### Directional Performance: Confusion Matrix

<table>
<thead>
<tr>
<th>Actual</th>
<th>Predicted</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>up</td>
<td></td>
<td></td>
</tr>
<tr>
<td>down</td>
<td></td>
<td></td>
</tr>
<tr>
<td>up</td>
<td>$m_{11}$</td>
<td>$m_{12}$</td>
</tr>
<tr>
<td>down</td>
<td>$m_{21}$</td>
<td>$m_{22}$</td>
</tr>
<tr>
<td></td>
<td>$m_{01}$</td>
<td>$m_{02}$</td>
</tr>
</tbody>
</table>

\[
\chi^2 = \sum_{i=1}^{2} \sum_{j=1}^{2} \frac{(m_{ij} - m_{i0}m_{0j}/m)^2}{m_{i0}m_{0j}/m}
\]

with 1 degree of freedom.
Example: Confusion Matrix

<table>
<thead>
<tr>
<th>Actual</th>
<th>Predicted</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>up</td>
<td>down</td>
</tr>
<tr>
<td>up</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>down</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>4</td>
</tr>
</tbody>
</table>

\[ \chi^2 = 0.137 \]

with p-value of 0.71.
ROC chart

ROC Space

Perfect Classification

Random guess

TPR or sensitivity

FPR or (1 - specificity)

Better

Worse
Performance Terminology

- **Precision** is $TP/(TP + FP)$: fraction of records that were positive from the group that the classifier predicted to be positive.

- **True Positive Rate** is $TP/(TP + FN)$: fraction of positive examples the classifier correctly identified. This is also known as Recall or Sensitivity.

- **False Positive Rate** is $FP/(FP + TN)$: fraction of positive examples which the classifier mis-identified.

- **Specificity** is $TN/(FP + TN) = 1 − FPR$: fraction of negative examples which the classifier correctly identified.
Practical Prediction Issues

• How to predict inherently rare events, e.g. black swans? Ans: over-sampling?
• How much training history?
• How frequently to retrain the model?
Figure: Normalized confusion matrices (left) without and (right) with oversampling.
Figure: ROC curves with and without oversampling.