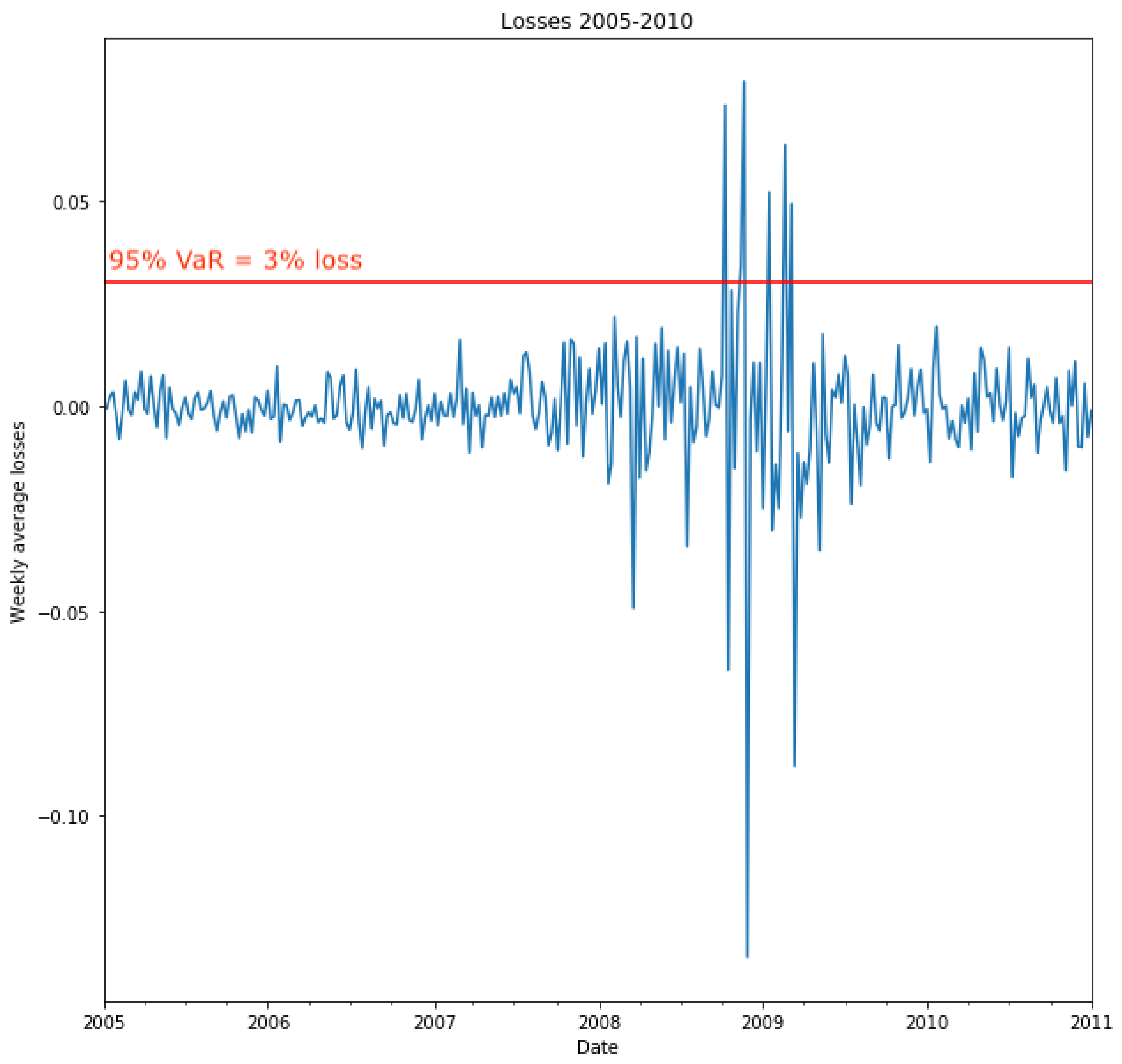
Extreme value theory

QUANTITATIVE RISK MANAGEMENT IN PYTHON

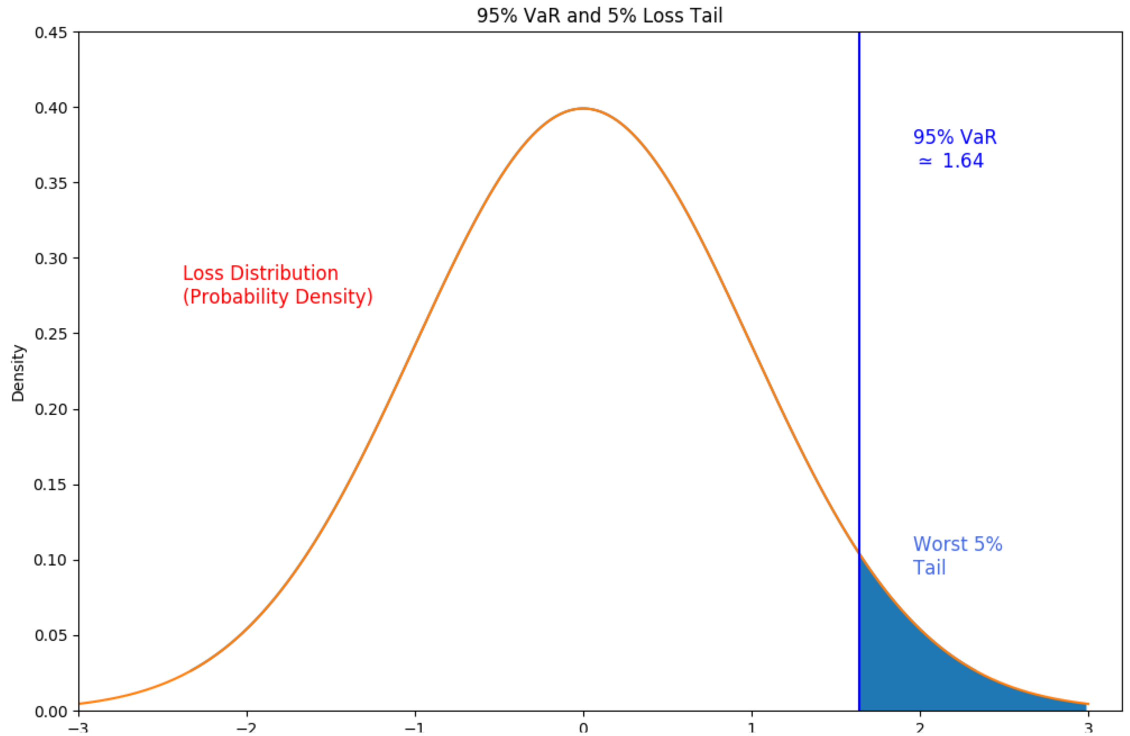
Jamsheed Shorish
Computational Economist

Extreme values

- Portfolio losses: extreme values



- Extreme values: from tail of distribution
  - Tail losses: losses exceeding some value
  - Model tail losses => better risk management

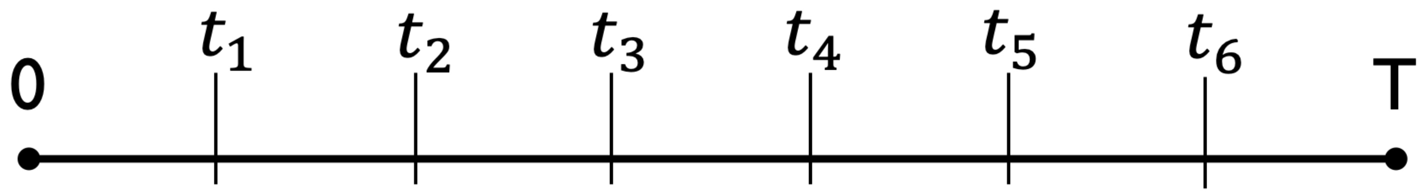


Extreme value theory

- **Extreme value theory**: statistical distribution of extreme values
- **Block maxima**

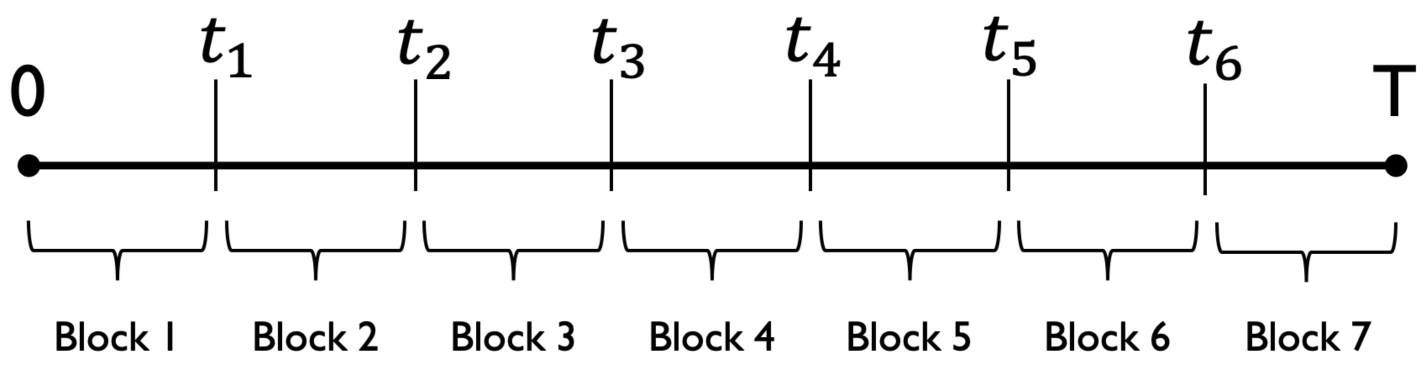
Extreme value theory

- **Extreme value theory**: statistical distribution of extreme values
- **Block maxima**:
  - Break period into sub-periods



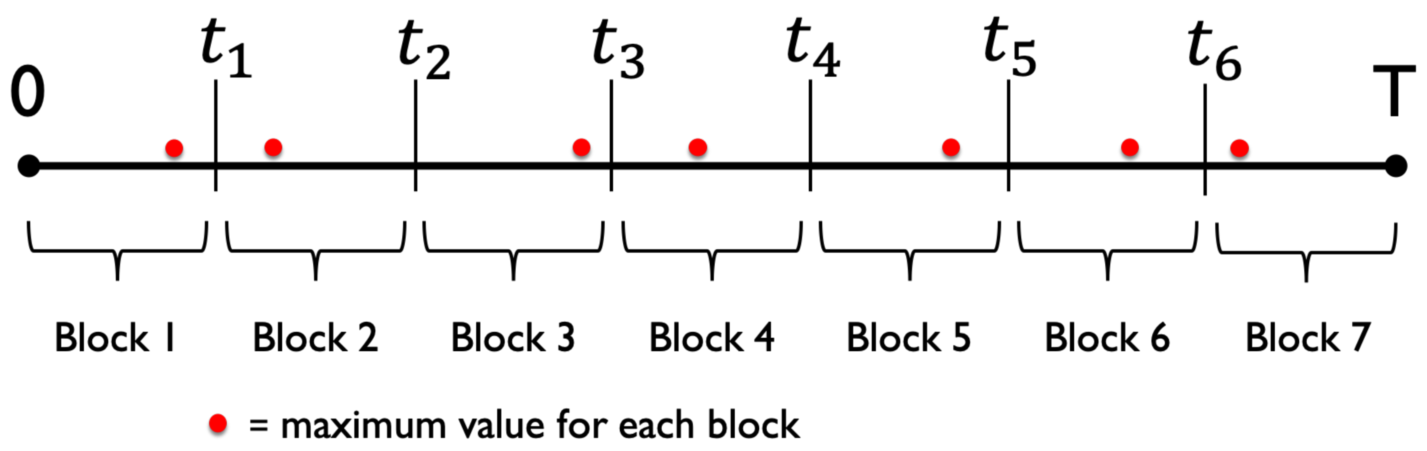
Extreme value theory

- **Extreme value theory**: statistical distribution of extreme values
- **Block maxima**:
  - Break period into sub-periods
  - Form *block* from each sub-period



Extreme value theory

- **Extreme value theory**: statistical distribution of extreme values
- **Block maxima**:
  - Break period into sub-periods
  - Form *blocks* from each sub-period
  - Set of block maxima = dataset
- **Peak over threshold** (POT):
  - Find all losses over given level
  - Set of such losses = dataset



Generalized Extreme Value Distribution

- **Example**: Block maxima for 2007 - 2009
  - Resample losses with desired period (e.g. weekly)

```
maxima = losses.resample("W").max()
```

- **Generalized Extreme Value Distribution (GEV)**
  - Distribution of *maxima* of data
  - Example: parametric estimation using `scipy.stats.genextreme`

```
from scipy.stats import genextreme
params = genextreme.fit(maxima)
```

VaR and CVaR from GEV distribution

- **99% VaR** from GEV distribution
  - Use `.ppf()` percent point function to find 99% VaR
  - Requires `params` from fitted GEV distribution
  - Finds maximum loss over one week period at 99% confidence
- **99% CVaR** from GEV distribution
  - CVaR is conditional expectation of loss *given* VaR as minimum loss
  - Use `.expect()` method to find expected value

```
VaR_99 = genextreme.ppf(0.99, *params)
```

```
CVar_99 = ( 1 / (1 - 0.99) ) * genextreme.expect(lambda x: x, *params, lb = VaR_99)
```

Covering losses

- **Risk management**: *covering losses*
  - Regulatory requirement (banks, insurance)
  - Reserves must be available to cover losses
    - For a specified period (e.g. one week)
    - At a specified confidence level (e.g. 99%)
- **VaR from GEV distribution**:
  - estimates maximum loss
    - given period
    - given confidence level

Covering losses

- **Example**: Initial portfolio value = $1,000,000
- **One week reserve requirement** at 99% confidence
  - VaR_{99} from GEV distribution: maximum loss over one week at 99% confidence
- **Reserve requirement**: Portfolio value x VaR_{99}
  - Suppose VaR_{99} = 0.10, i.e. 10% maximum loss
  - Reserve requirement = $100,000
- Portfolio value *changes* => reserve requirement *changes*
- **Regulation** sets frequency of reserve requirement updating

Let's practice!

QUANTITATIVE RISK MANAGEMENT IN PYTHON

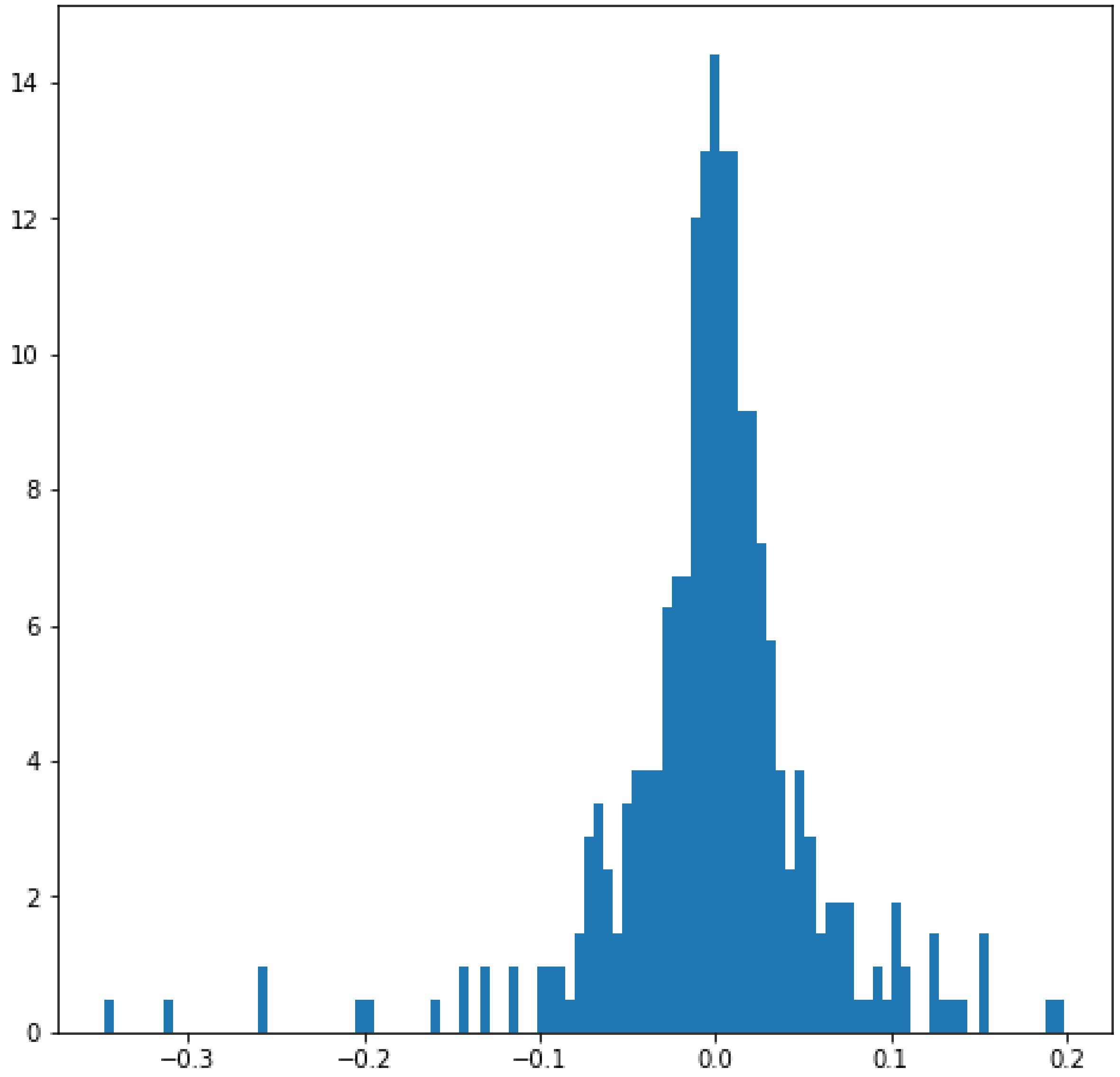
Kernel density estimation

QUANTITATIVE RISK MANAGEMENT IN PYTHON

Jamsheed Shorish
Computational Economist

The histogram revisited

- **Risk factor distributions**
  - *Assumed* (e.g. Normal, T, etc.)
  - *Fitted* (parametric estimation, Monte Carlo simulation)
  - *Ignored* (historical simulation)
- **Actual data**: histogram
- How to represent histogram by probability distribution?
  - *Smooth* data using **filtering**
  - **Non-parametric estimation**

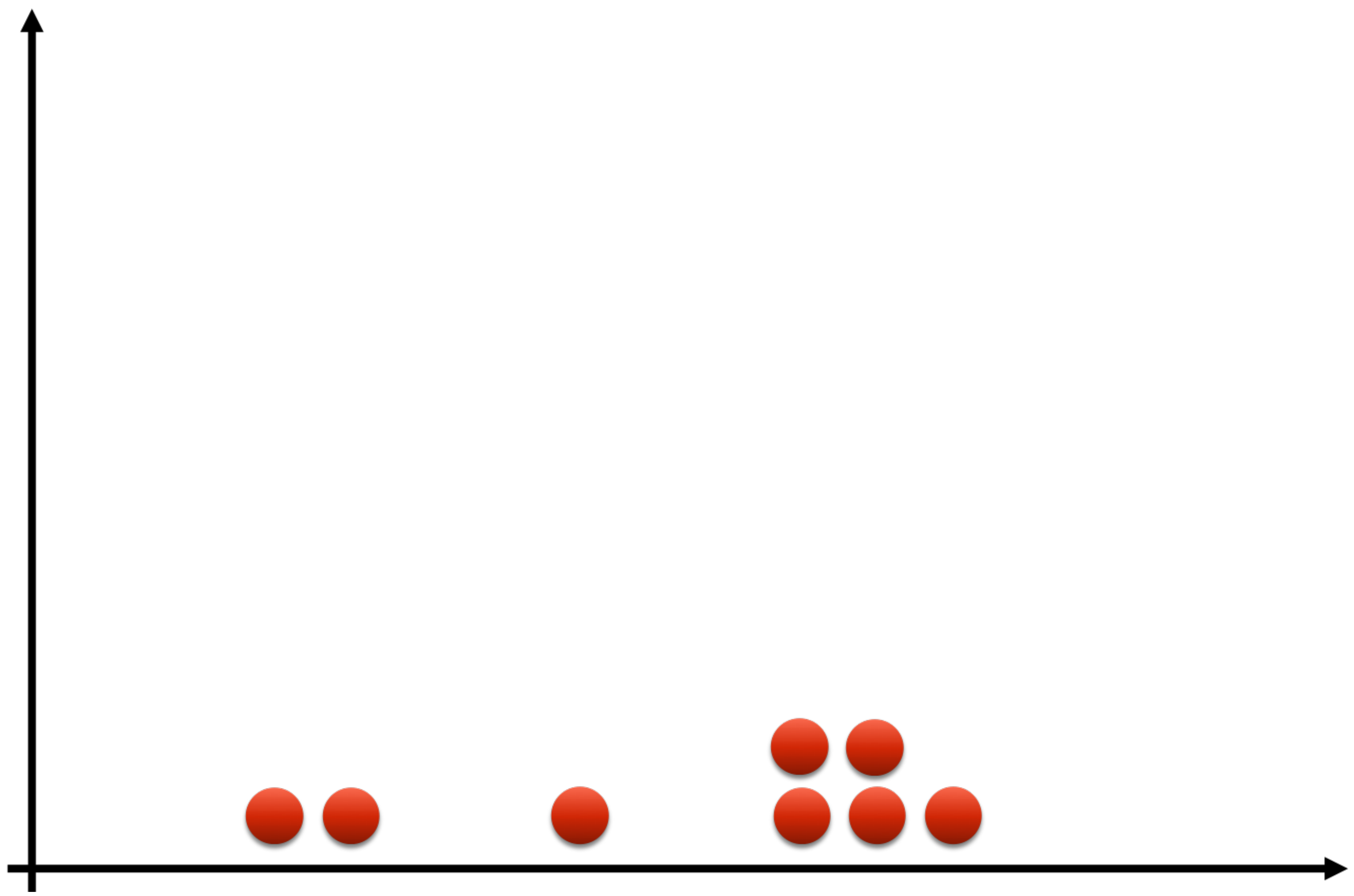


Data smoothing

- **Filter**: smoothen out 'bumps' of histogram

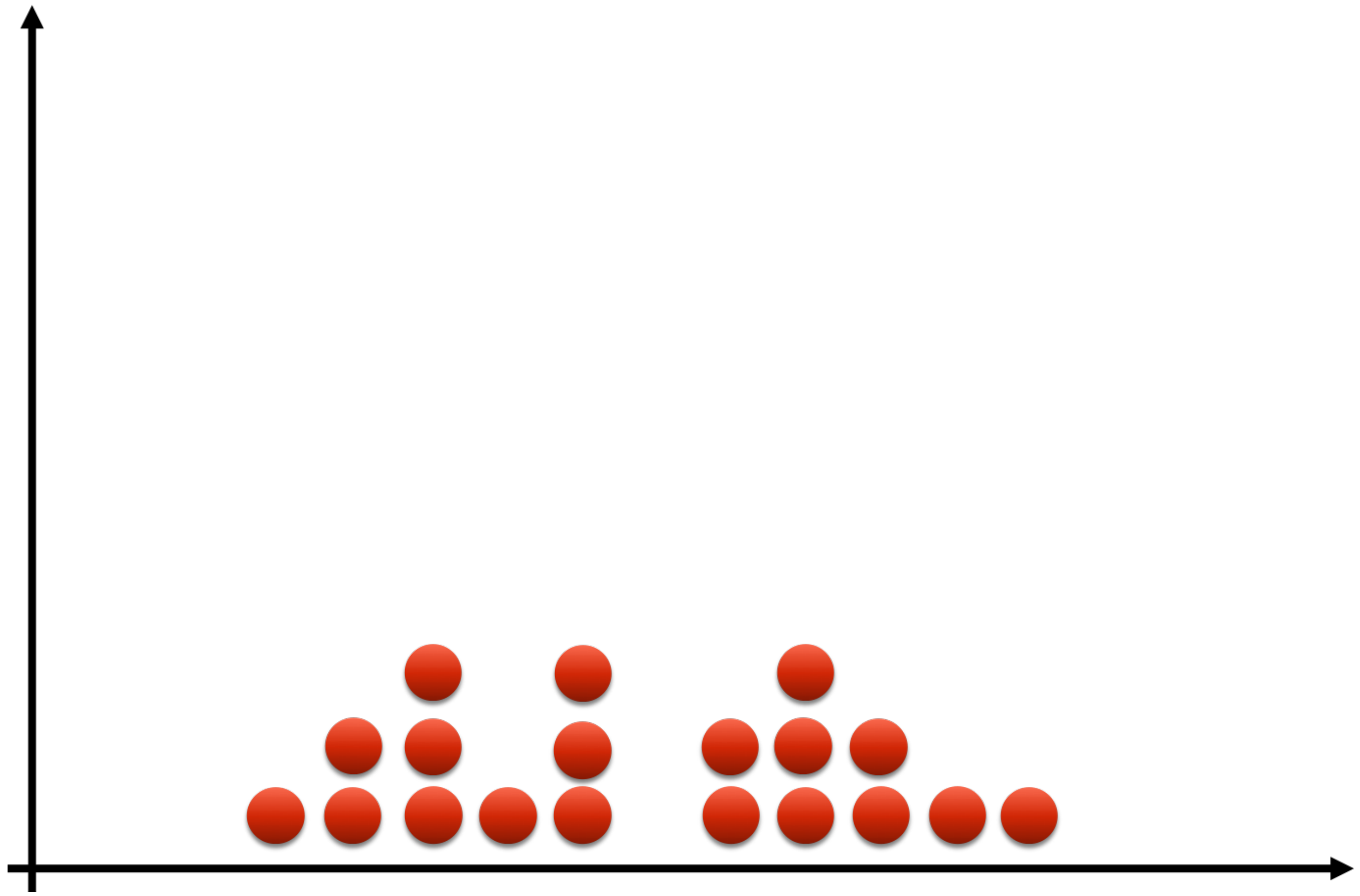
Data smoothing

- Filter: smoothen out 'bumps' of histogram
- Observations accumulate in over time



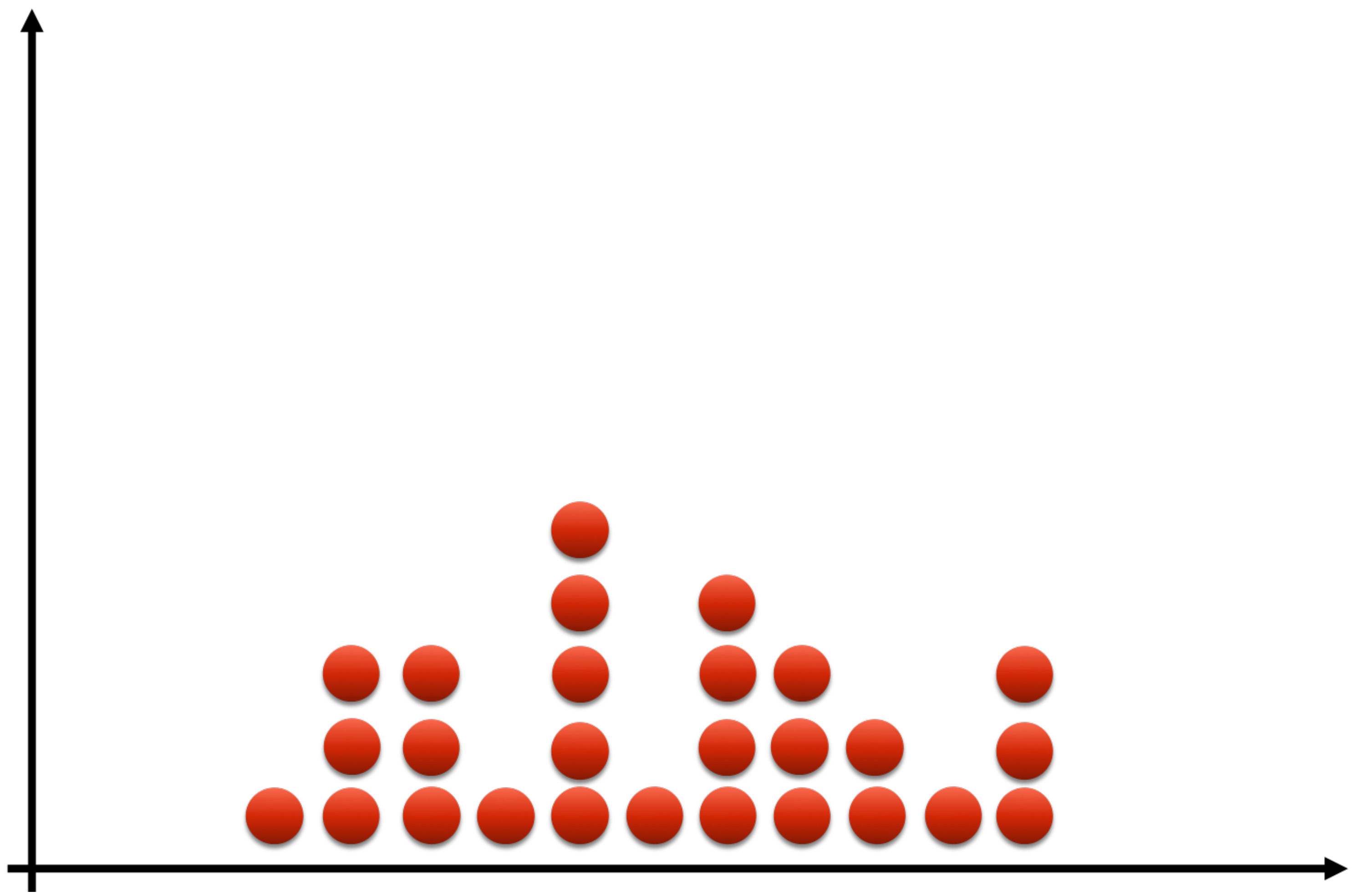
Data smoothing

- Filter: smoothen out 'bumps' of histogram
- Observations accumulate in over time



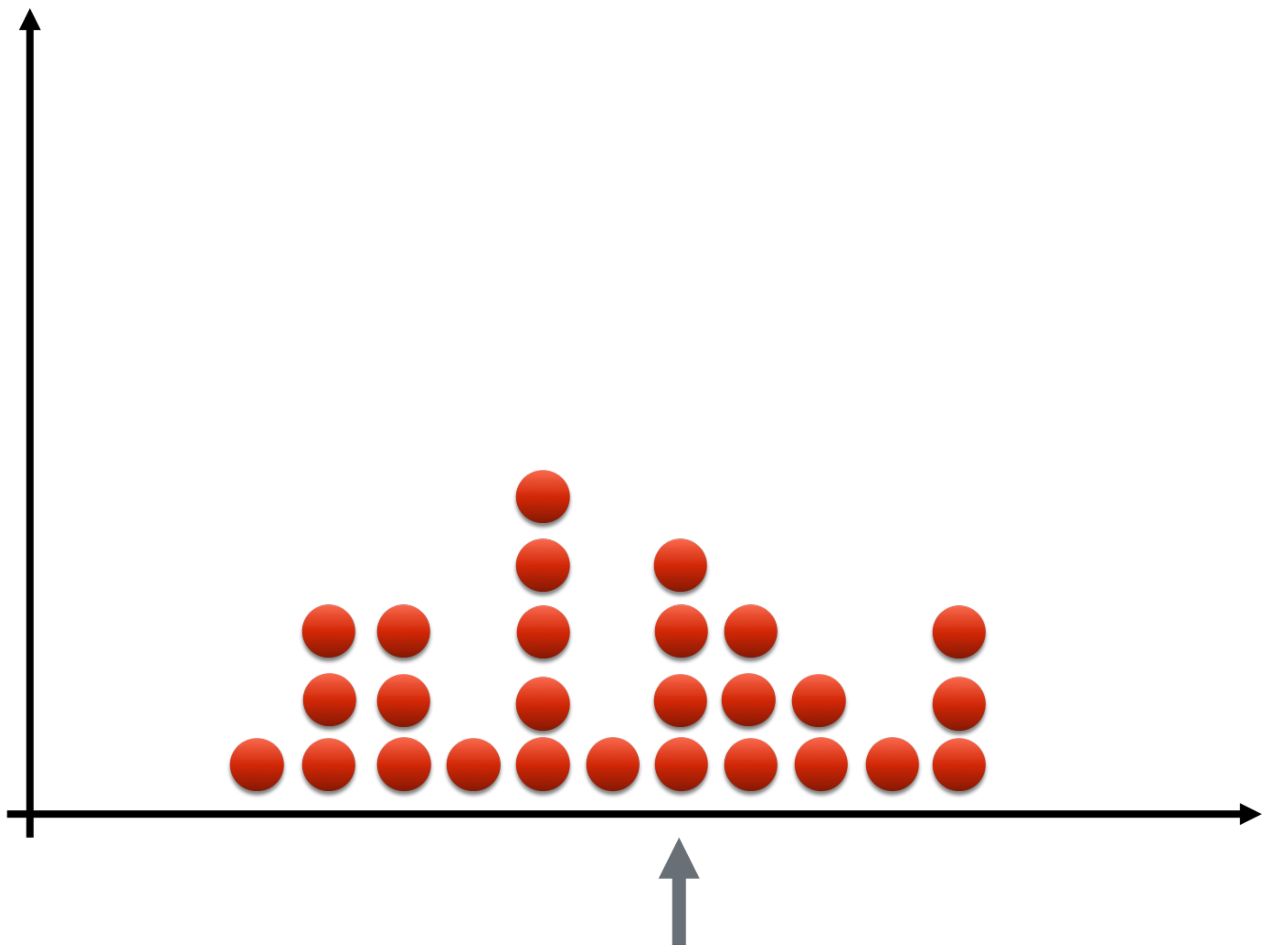
Data smoothing

- Filter: smoothen out 'bumps' of histogram
- Observations accumulate in over time



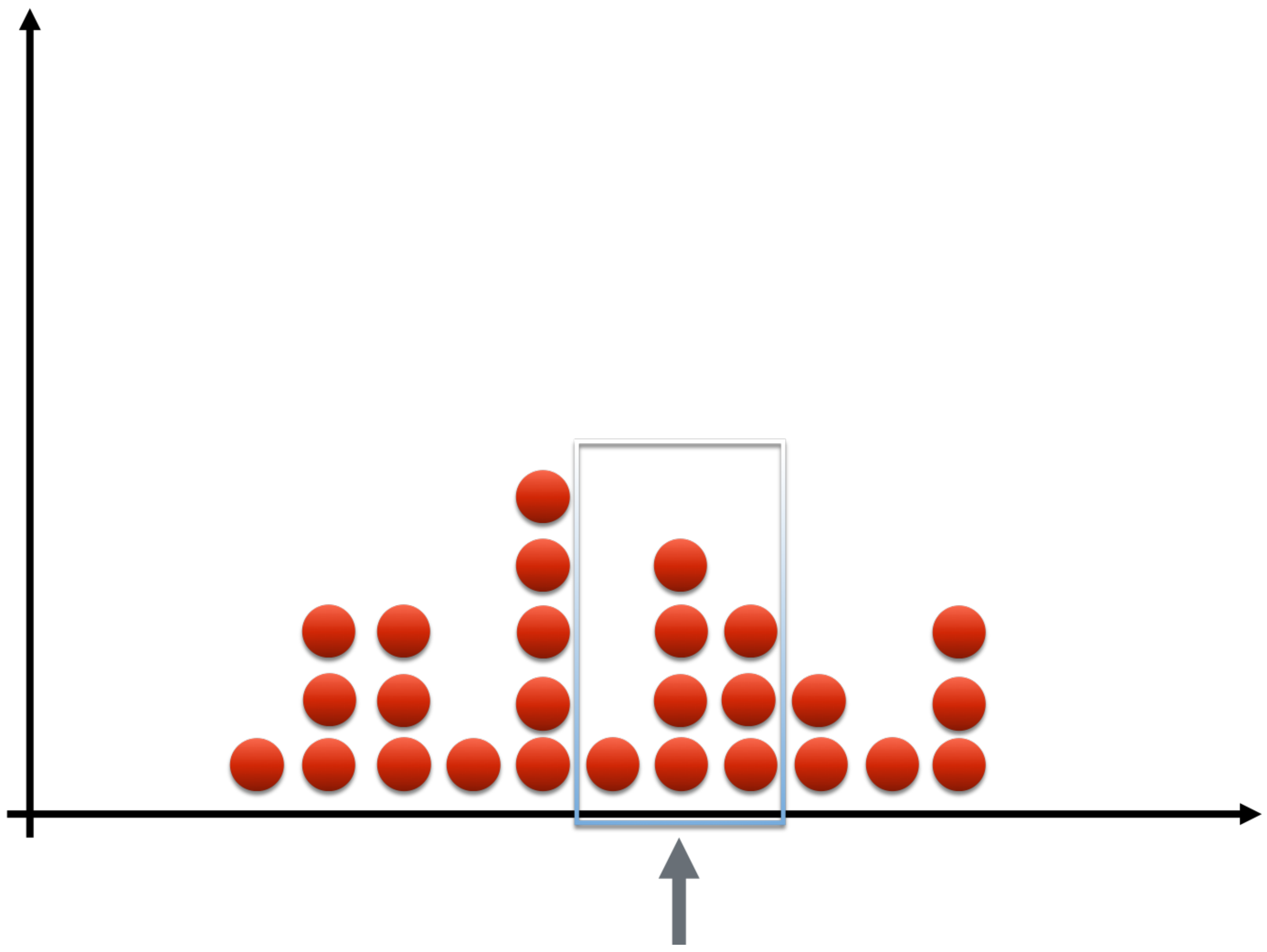
Data smoothing

- Filter: smoothen out 'bumps' of histogram
- Observations accumulate in over time
- Pick particular portfolio loss



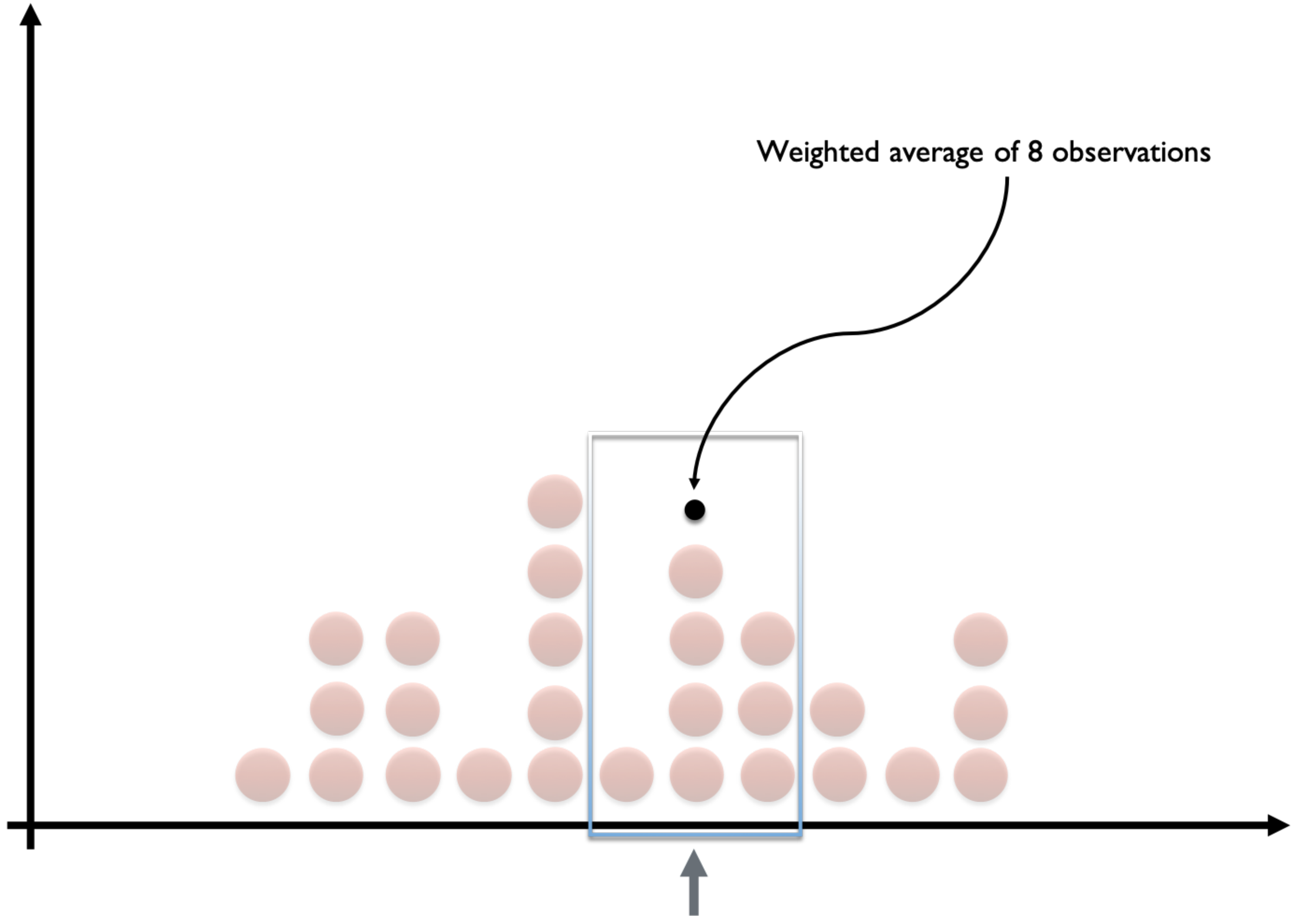
Data smoothing

- Filter: smoothen out 'bumps' of histogram
- Observations accumulate in over time
- Pick particular portfolio loss
  - Examine nearby losses



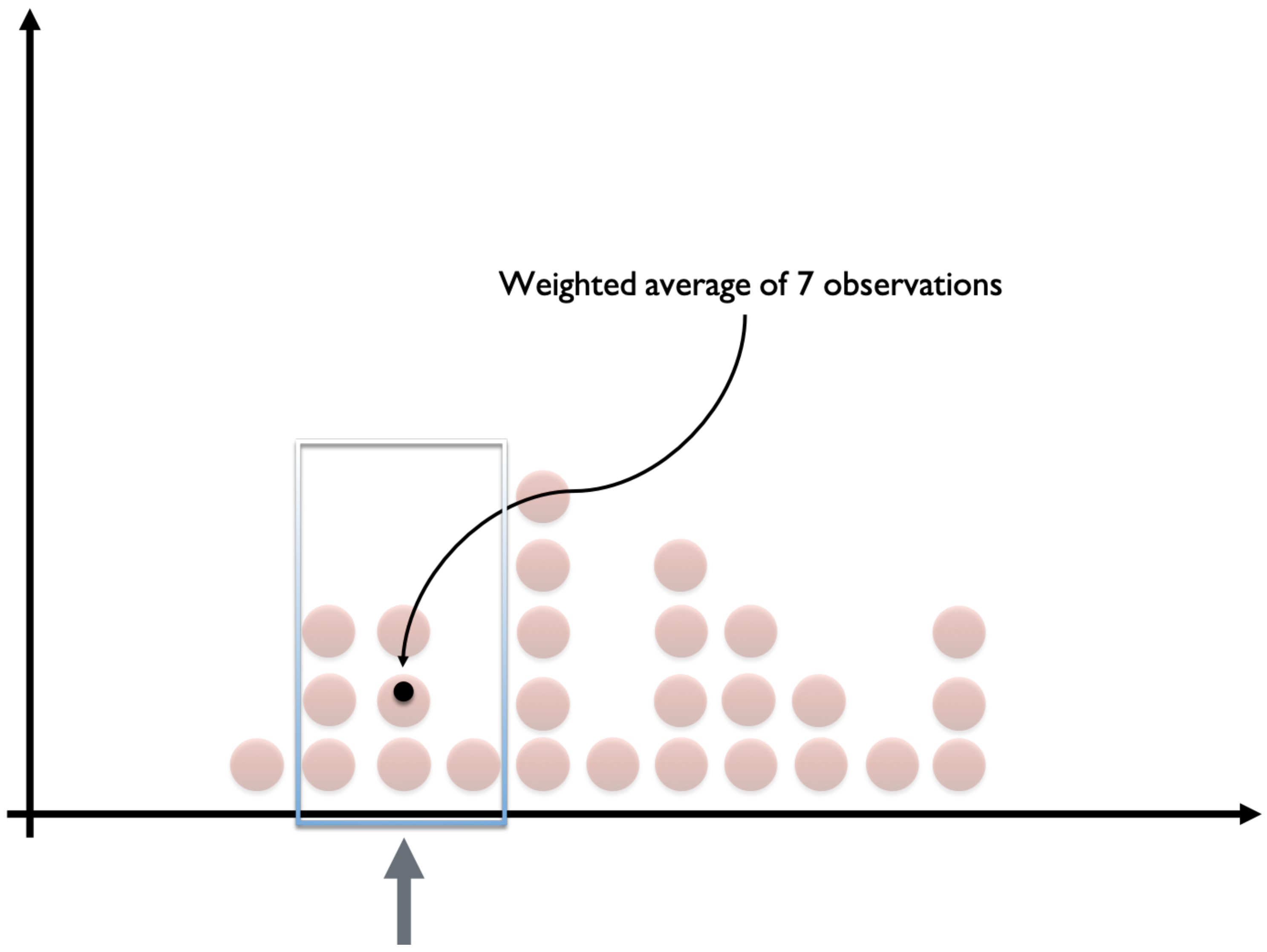
Data smoothing

- Filter: smoothen out 'bumps' of histogram
- Observations accumulate in over time
- Pick particular portfolio loss
  - Examine nearby losses
  - Form "weighted average" of losses
- **Kernel**: filter choice; determines "window"



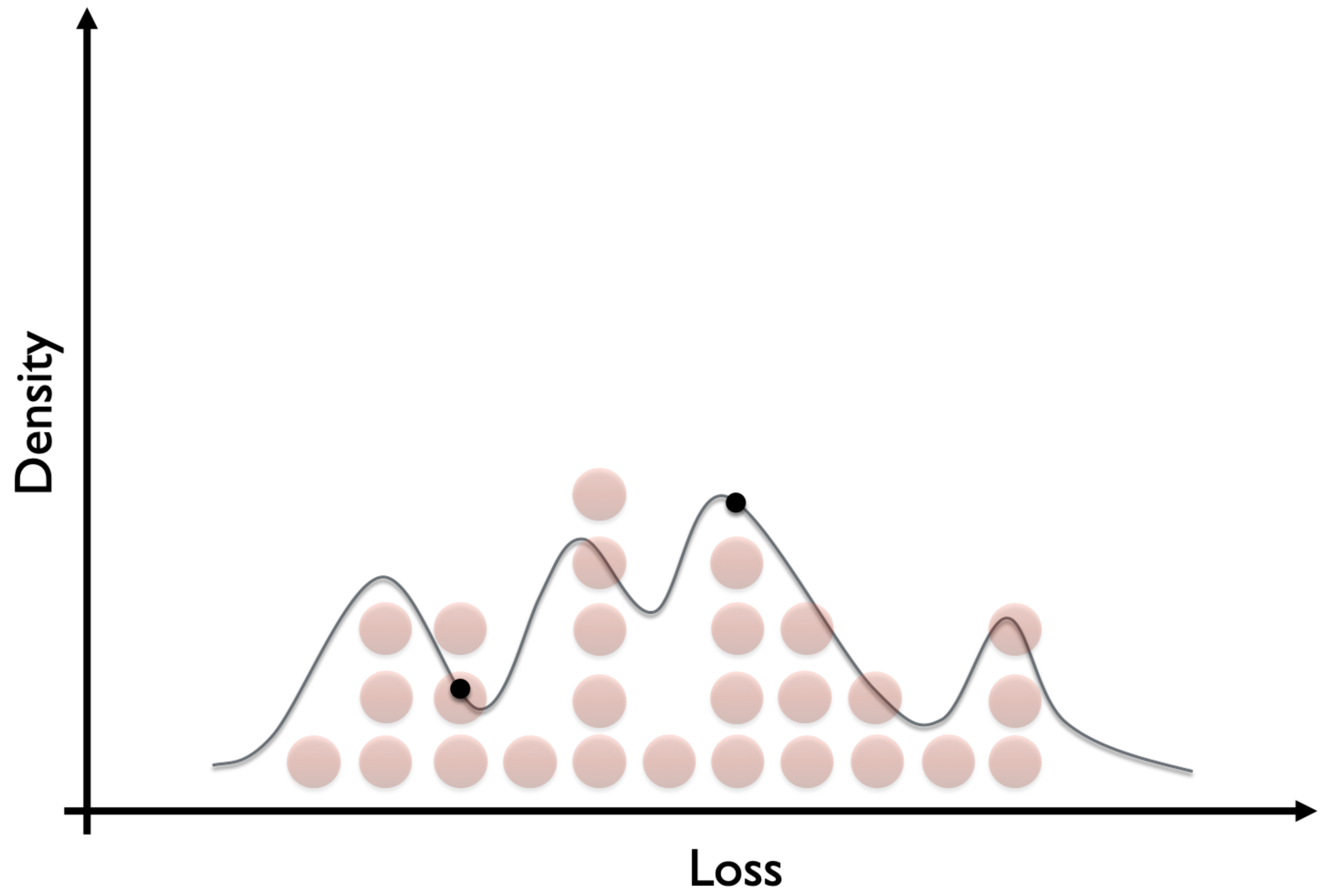
Data smoothing

- Filter: smoothen out 'bumps' of histogram
- Observations accumulate in over time
- Pick particular portfolio loss
  - Examine nearby losses
  - Form "weighted average" of losses
- **Kernel**: filter choice; determines "window"
  - Move window to another loss



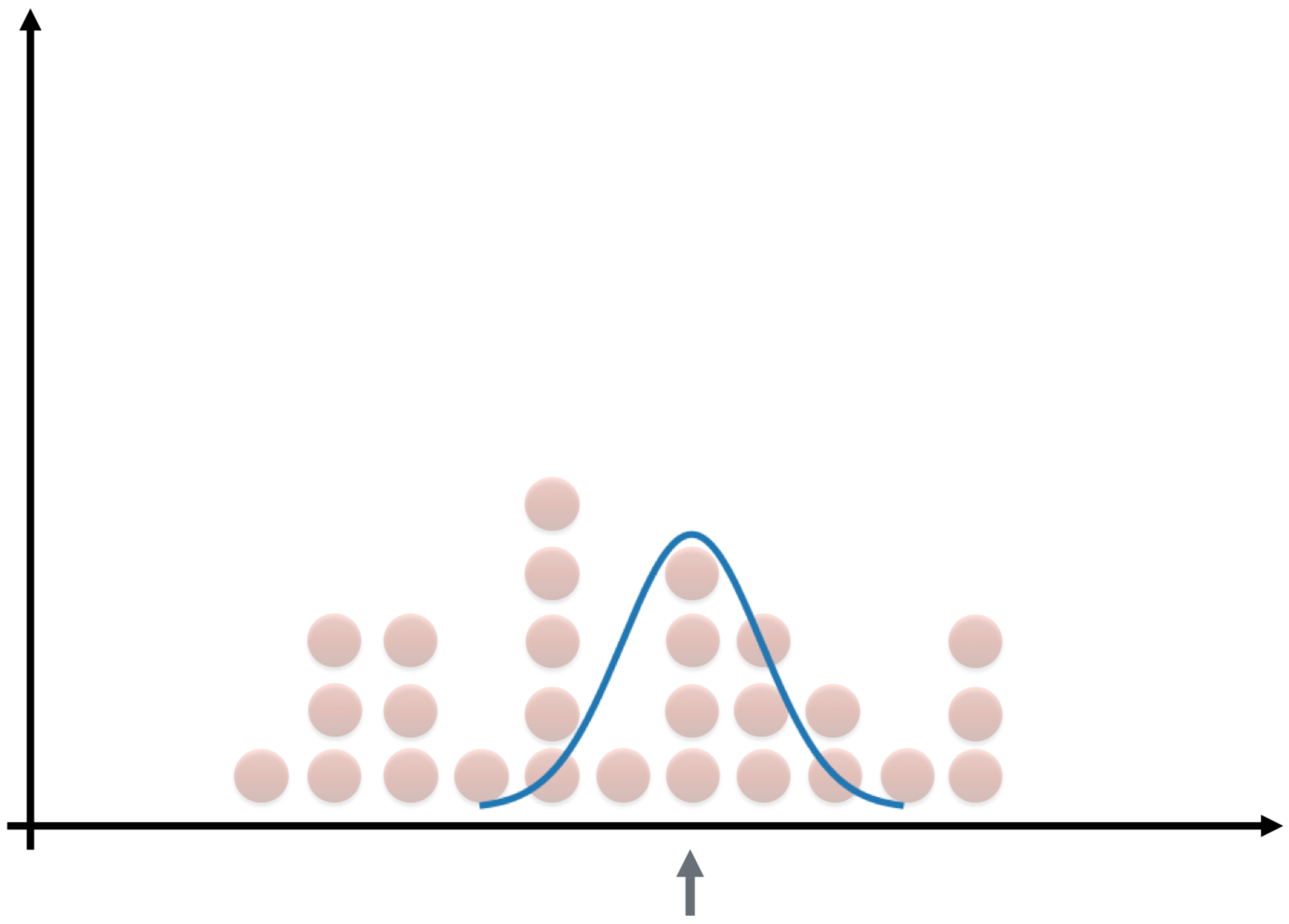
Data smoothing

- Filter: smoothen out 'bumps' of histogram
- Observations accumulate in over time
- Pick particular portfolio loss
  - Examine nearby losses
  - Form "weighted average" of losses
- **Kernel**: filter choice; determines "window"
  - Move window to another loss
- **Kernel density estimate**: probability density



The Gaussian kernel

- Continuous kernel
- Weights all observations by distance from center
- **Generally**: many different kernels are available
  - Used in time series analysis
  - Used in signal processing



KDE in Python

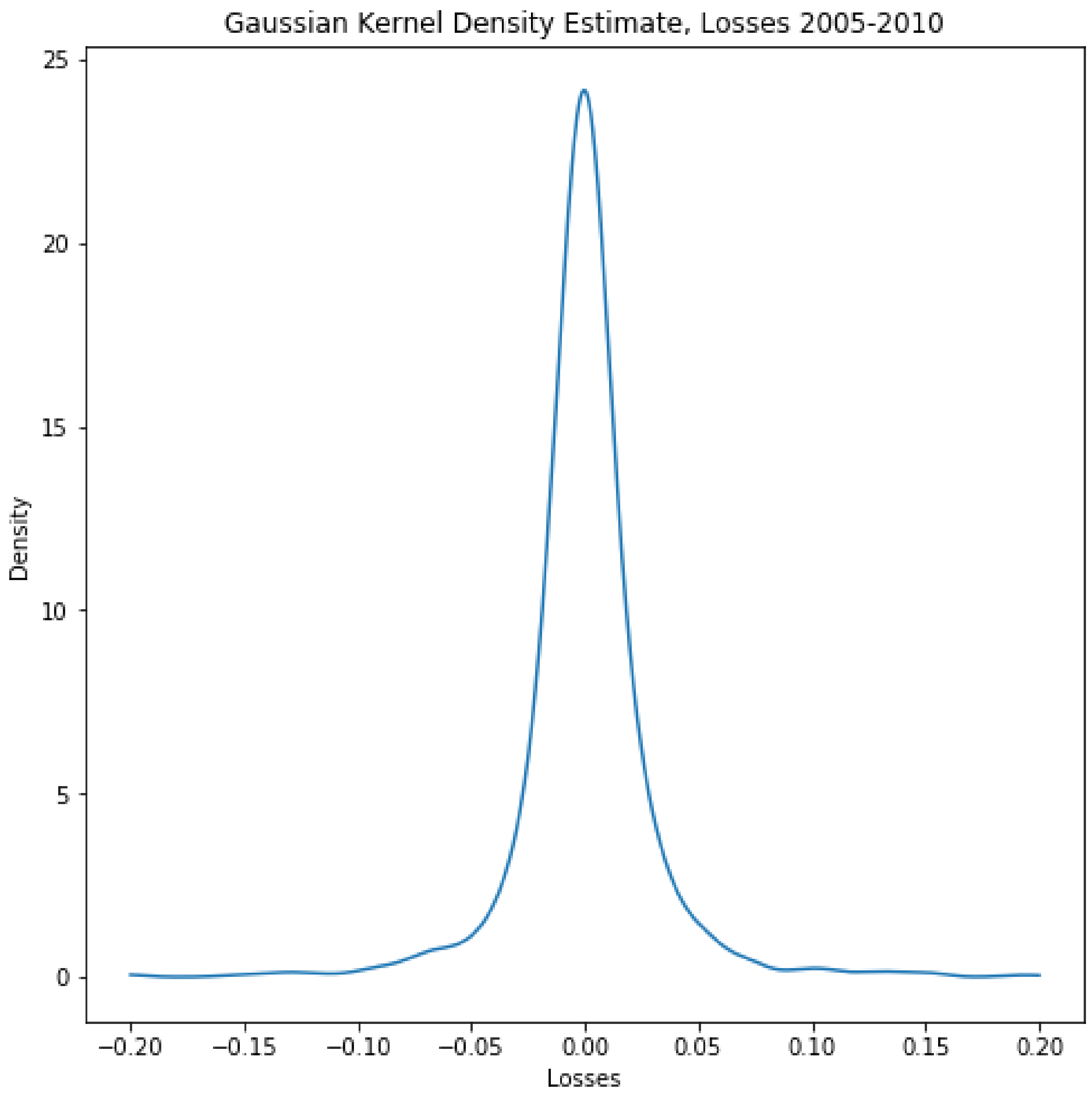
```
from scipy.stats import gaussian_kde

kde = guassian_kde(losses)

x = np.linspace(np.min(losses),
                np.max(losses),
                1000)

plt.plot(x, kde.pdf(x))
```

- Visualization: probability density function from KDE fit



Finding VaR using KDE

- VaR: use `gaussian_kde` `.resample()` method
- Find quantile of resulting sample
- CVaR: expected value as previously encountered, but
  - `gaussian_kde` has no `.expect()` method => compute integral manually
  - special `.expect()` method written for exercise

```
sample = kde.resample(size = 1000)
VaR_99 = np.quantile(sample, 0.99)
print("VaR_99 from KDE: ", VaR_99)
```

```
VaR_99 from KDE: 0.08796423698448601
```

Let's practice!

QUANTITATIVE RISK MANAGEMENT IN PYTHON

Neural network risk management

QUANTITATIVE RISK MANAGEMENT IN PYTHON

Jamsheed Shorish
Computational Economist

Real-time portfolio updating

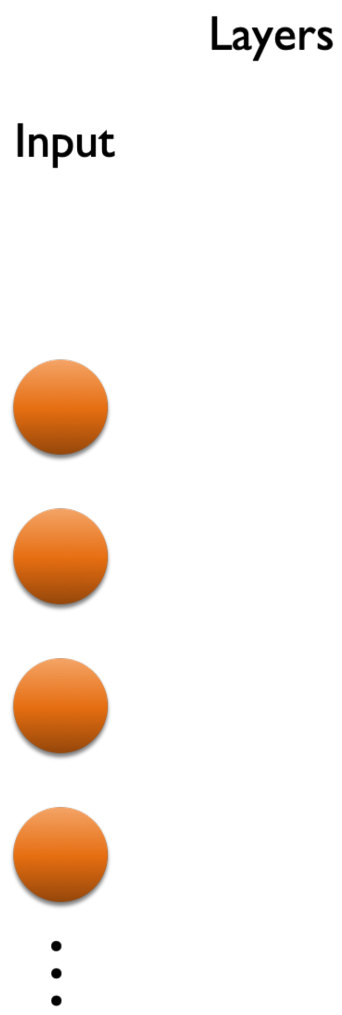
- **Risk management**
  - **Defined** risk measures (VaR, CVaR)
  - **Estimated** risk measures (parameteric, historical, Monte Carlo)
  - **Optimized** portfolio (e.g. Modern Portfolio Theory)
- *New* market information => *update* portfolio weights
  - **Problem**: portfolio optimization costly
  - **Solution**: $\text{weights} = f(\text{prices})$
  - *Evaluate* f in real-time
  - *Update* f only occasionally

Neural networks

- **Neural Network**: $\text{output} = f(\text{input})$
  - **Neuron**: interconnected processing node in function
- Initially developed 1940s-1950s
- **Early 2000s**: application of neural networks to "big data"
  - Image recognition, processing
  - Financial data
  - Search engine data
- **Deep Learning**: neural networks as part of Machine Learning
  - **2015**: Google releases open-source Tensorflow deep learning library for Python

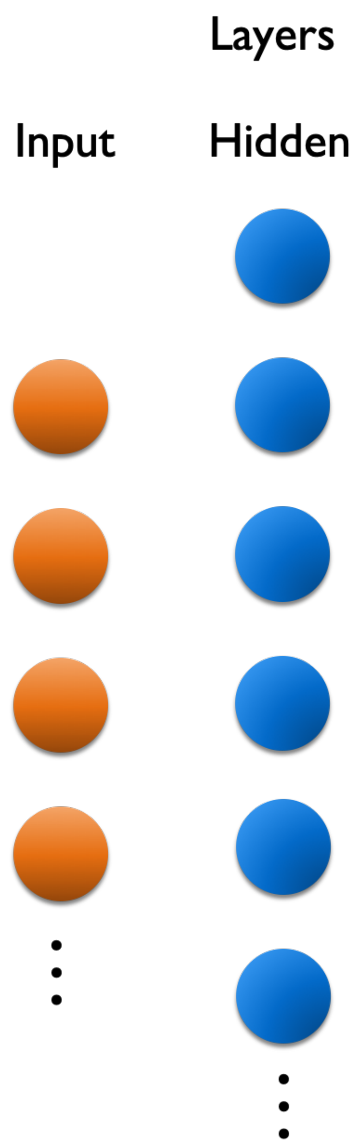
Neural network structure

- **Layers: connected processing neurons**
  - Input layer



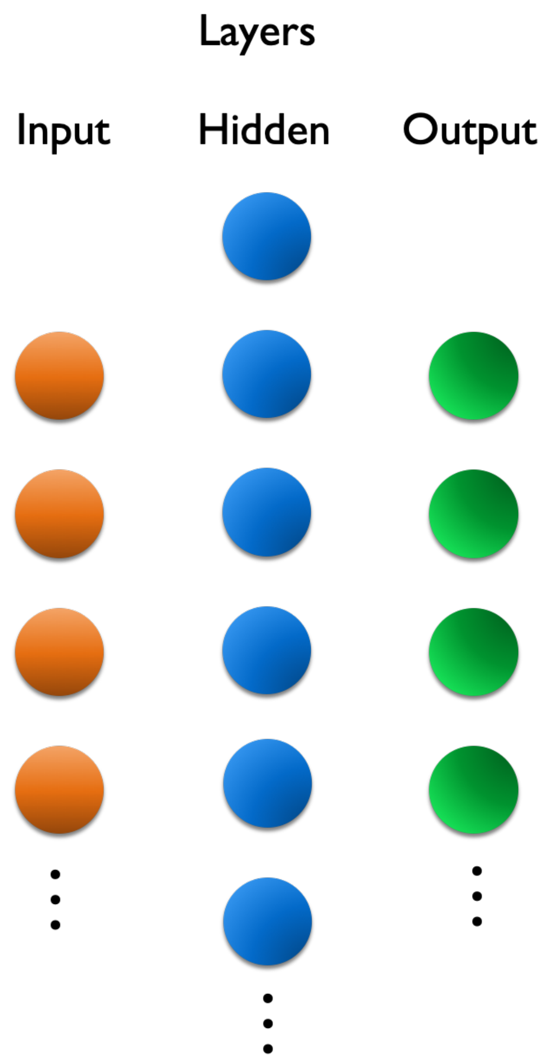
Neural network structure

- **Neural network structure**
  - Input layer
  - Hidden layer



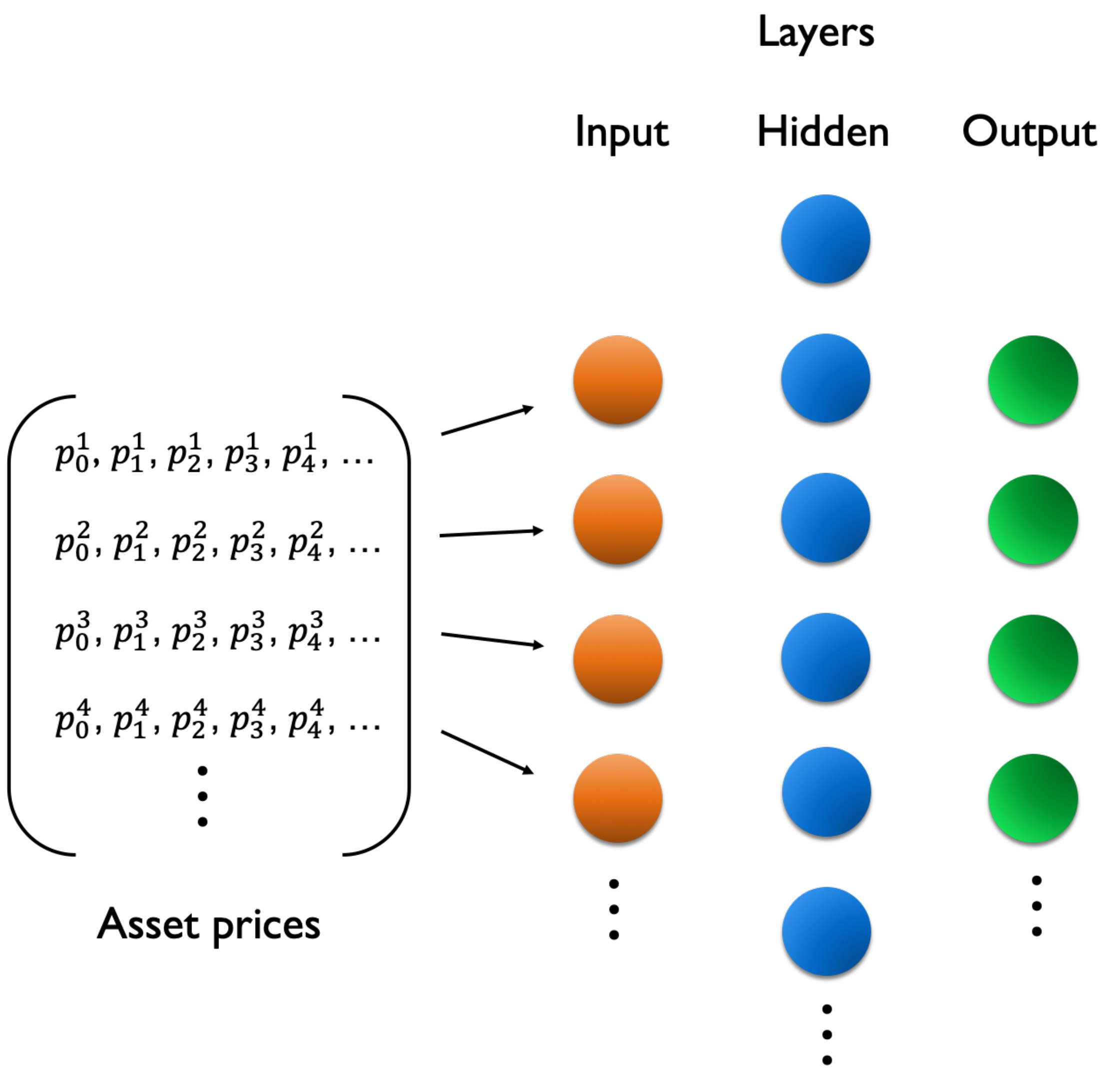
Neural network structure

- **Neural network structure**
  - Input layer
  - Hidden layer
  - Output layer
- **Training**: learn relationship between input and output



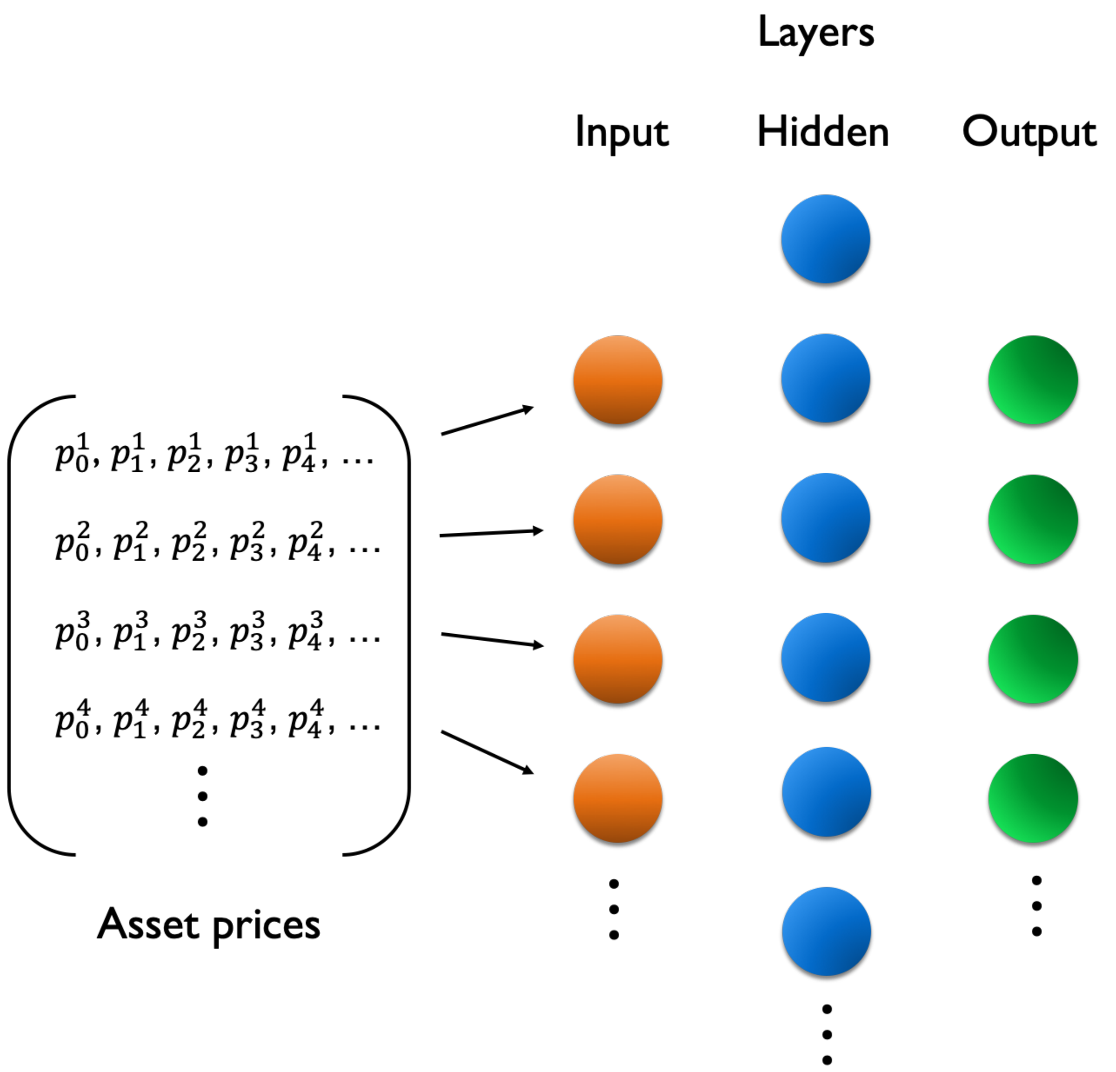
Neural network structure

- **Neural network structure**
  - Input layer
  - Hidden layer
  - Output layer
- **Training**: learn relationship between input and output
  - Asset prices => Input layer



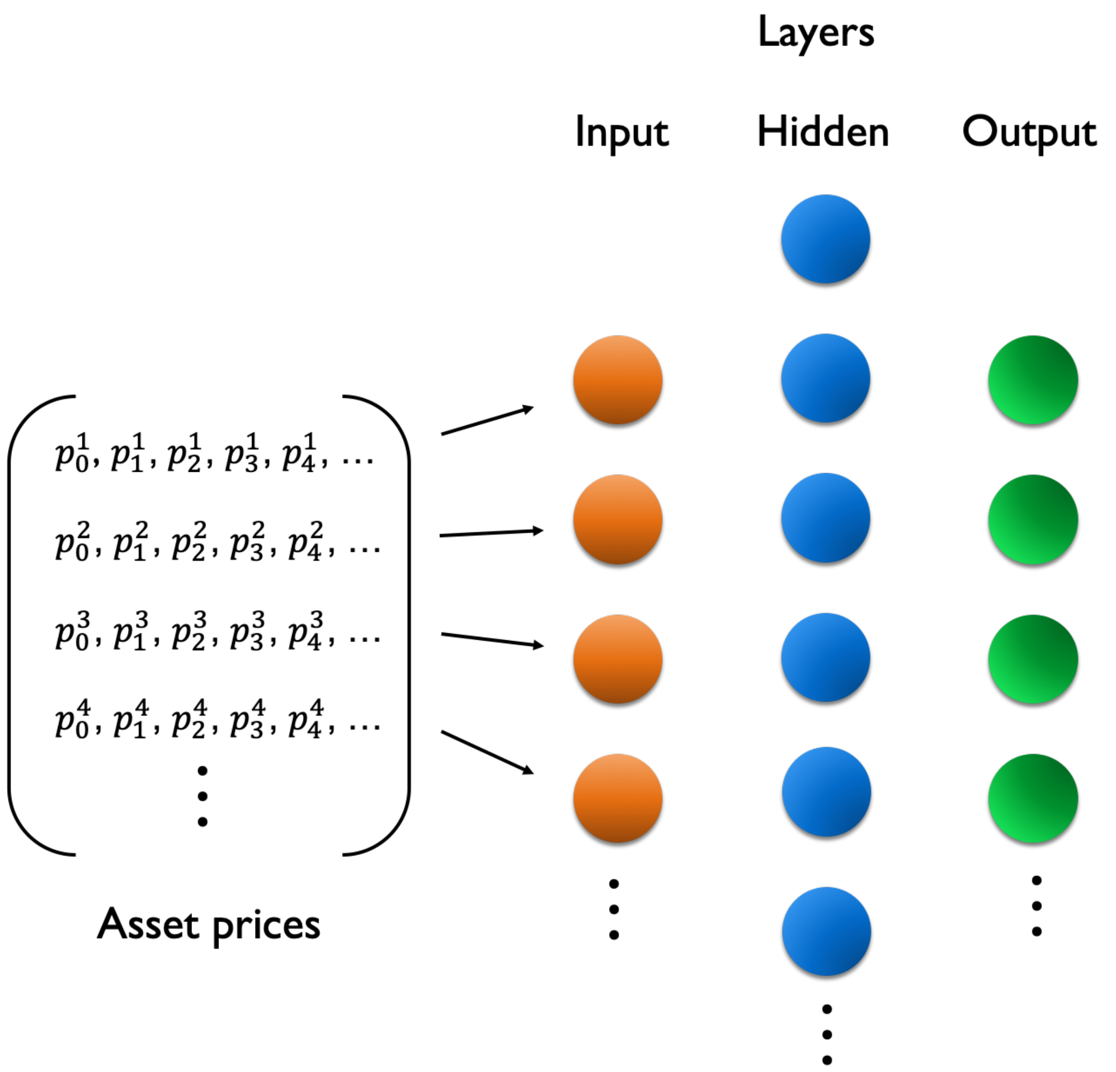
Neural network structure

- **Neural network structure**
  - Input layer
  - Hidden layer
  - Output layer
- **Training**: learn relationship between input and output
  - Asset prices => Input layer
  - Input + hidden layer processing



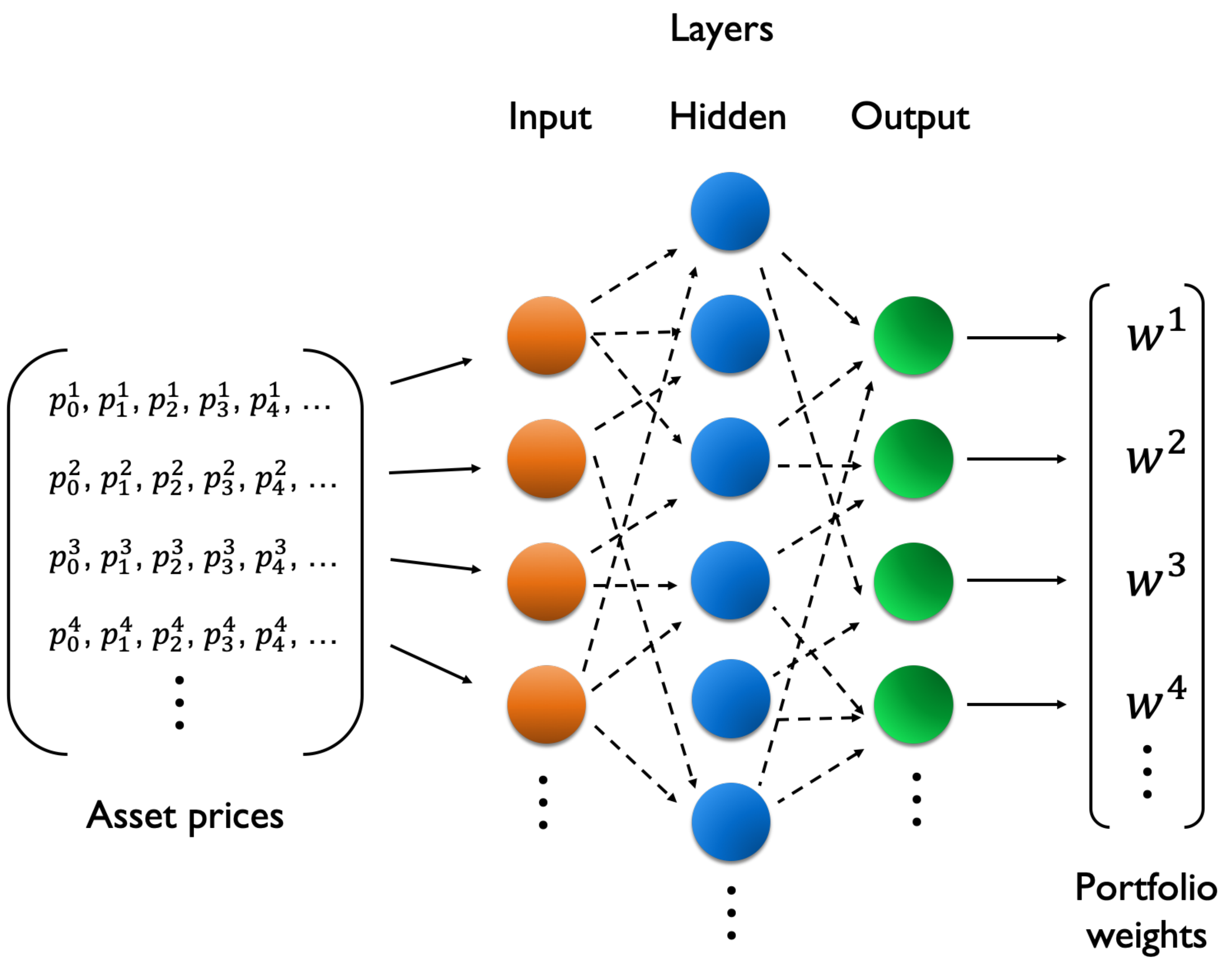
Neural network structure

- **Neural network structure**
  - Input layer
  - Hidden layer
  - Output layer
- **Training**: learn relationship between input and output
  - Asset prices => Input layer
  - Input + hidden layer processing
  - Hidden + output layer processing



Neural network structure

- **Neural network structure**
  - Input layer
  - Hidden layer
  - Output layer
- **Training**: learn relationship between input and output
  - Asset prices => Input layer
  - Input + hidden layer processing
  - Hidden + output layer processing
  - Output => portfolio weights



Using neural networks for portfolio optimization

- **Training**
  - *Compare* output and pre-existing "best" portfolio weights
  - **Goal**: minimize "error" between output and weights
  - Small error => network is *trained*
- **Usage**
  - **Input**: new, unseen asset prices
  - **Output**: predicted "best" portfolio weights for new asset prices
  - Best weights = risk management

Creating neural networks in Python

- **Keras**: high-level Python library for neural networks/deep learning
- Further info: Introduction to Deep Learning with Keras

```
from keras.models import Sequential

from keras.layers import Dense

model = Sequential()
model.add(Dense(10, input_dim=4, activation='sigmoid'))
model.add(Dense(4))
```

Training the network in Python

- Historical asset prices: `training_input` matrix
- Historical portfolio weights: `training_output` vector
- **Compile** model with:
  - given error minimization ('**loss**')
  - given optimization algorithm ('**optimizer**')
- **Fit** model to training data
  - **epochs**: number of training loops to update internal parameters

```
model.compile(loss='mean_squared_error', optimizer='rmsprop')

model.fit(training_input, training_output, epochs=100)
```

Risk management in Python

- **Usage**: provide new (e.g. real-time) asset pricing data
  - **New vector** `new_asset_prices` given to input layer
- Evaluate network using `model.predict()` on new prices
  - **Result:** `predicted` portfolio weights
- Accumulate enough data over time => **re-train** network
  - Test network on previous data => **backtesting**

```
# new asset prices are in the vector new_asset_prices
predicted = model.predict(new_asset_prices)
```

Let's practice!

Wrap-up and Future Steps

QUANTITATIVE RISK MANAGEMENT IN PYTHON

Jamsheed Shorish
Computational Economist

Congratulations!

Chapter 1

Risk and
Return Recap

Return Distribution

Risk Factors

Volatility & Covariance

Modern Portfolio Theory

Efficient Portfolio &
Efficient Frontier

Congratulations!

Chapter 1	Chapter 2
Risk and Return Recap	Goal-oriented Risk Management
Return Distribution	Loss Distribution
Risk Factors	Value at Risk (VaR)
Volatility & Covariance	Conditional VaR
Modern Portfolio Theory	Risk Exposure
Efficient Portfolio & Efficient Frontier	Portfolio Hedging

Congratulations!

Chapter 1	Chapter 2	Chapter 3
Risk and Return Recap	Goal-oriented Risk Management	Estimating & Identifying Risk
Return Distribution	Loss Distribution	Parametric Estimation
Risk Factors	Value at Risk (VaR)	Historical Simulation
Volatility & Covariance	Conditional VaR	Monte Carlo Simulation
Modern Portfolio Theory	Risk Exposure	Structural Breaks
Efficient Portfolio & Efficient Frontier	Portfolio Hedging	Extreme Events

Congratulations!

Chapter 1	Chapter 2	Chapter 3	Chapter 4
Risk and Return Recap	Goal-oriented Risk Management	Estimating & Identifying Risk	Advanced Risk Management
Return Distribution	Loss Distribution	Parametric Estimation	Extreme Value Theory
Risk Factors	Value at Risk (VaR)	Historical Simulation	Kernel Density Estimation
Volatility & Covariance	Conditional VaR	Monte Carlo Simulation	Neural Networks
Modern Portfolio Theory	Risk Exposure	Structural Breaks	Real-time risk management
Efficient Portfolio & Efficient Frontier	Portfolio Hedging	Extreme Events	

Tools in your toolkit

Scipy	Statsmodels	PyPortfolioOpt	Keras
scipy.stats	statsmodels.api	pypfopt	keras
norm()	OLS()	risk_models	models
skewnorm()	add_constant()	cla	layers
t()	.fit()	expected_returns	Sequential()
genextreme()		efficient_frontier	Dense()
gaussian_kde()		objective_functions	.add()
anderson()		EfficientFrontier()	.fit()
skewtest()		mean_historical_return()	.predict()
.pdf()		CovarianceShrinkage()	
.ppf()		.negative_cvar()	
.fit()		.CLA()	
.rvs()		.ledoit_wolf()	

Future steps and reference

- **Upcoming DataCamp courses**
  - Credit Risk Modeling in Python
  - Financial Forecasting in Python
  - Machine Learning for Finance in Python
  - GARCH Models for Finance in Python
- *Quantitative Risk Management: Concepts, Techniques and Tools*, McNeil, Frey & Embrechts, Princeton UP, 2015.

Best of luck on your data science journey!

QUANTITATIVE RISK MANAGEMENT IN PYTHON