Lucerne University of
Applied Sciences and Arts
HOCHSCHULE
LUZERN
Business
Institute of Financial Services Zug
IFZ

IFZ Working Paper Series ISSN 1662-520X

IFZ Working Paper No. 0008/2019
March 2019

Re-inforcement Learning for Pricing & Hedging of Derivatives - A Simplified Showcase

Autor

Thomas Krabichler — Lucerne University of Applied Sciences and Arts, Institute of Financial Services Zug IFZ, Grafenauweg 10, 6300 Zug, Switzerland. thomas.krabichler@hslu.ch

Abstract:

This work is inspired by [1]. We provide a technical documentation of how one can exploit re-inforcement learning in order to price & hedge financial derivatives in the presence of transaction cost. The concepts are illustrated exemplarily for a European call option in the one-step case.

Key Words: Hedging, Pricing, Transaction cost, Neural networks, Re-inforcement learning

JEL Classification: C02, C45, C63, G13

Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Re-inforcement Learning for Pricing & Hedging of Derivatives
A Simplified Showcase

Thomas Krabichler, March 11, 2019

Abstract. This work is inspired by [1]. We provide a technical documentation of how one can exploit re-inforcement learning in order to price & hedge financial derivatives in the presence of transaction cost. The concepts are illustrated exemplarily for a European call option in the one-step case.

1. Setup

We consider a discrete time environment over one period with respect to a deterministic initial state. An agent sells/buys a European call option with strike K, maturity T on an asset with the fixed current price S_{T-dt}. We use the following notation.

valuation date	$T-dt$
maturity	T
risk-free short rate	r
discount factor	$P(T-dt,T)=e^{-r\,dt}$
dividend yield	q
price process of underlying	$S=(S_t)_{t\in\{T-dt,T\}}$
forward price of S_T at $T-dt$	F_{T-dt}
proportional transaction cost	tc
holding in bank account before/after re-allocation	m^{pre}_{T-dt}, m^{post}_{T-dt}
initial holding in underlying/hedging strategy	$\Delta^{\text{pre}}_{T-dt}$, $\Delta^{\text{post}}_{T-dt}$
wealth before/after re-allocation	w^{pre}_{T-dt}, w^{post}_{T-dt}
terminal wealth	w_T
strike of the European call	K
payoff of the European call	$\pi_T=\max\{S_T-K,0\}$
holding (short, long)	$h\in\{-1,1\}$
profit & loss of netting the hedge and the payoff	PL_T
present value of the payoff	PV_{T-dt}

2. Valuation & Hedging Scheme

$$
\begin{aligned}
F_{T-dt} &= S_{T-dt}P(T-dt,T)^{-1}e^{-q}\\
m^{\text{post}}_{T-dt}\big(\Delta^{\text{post}}_{T-dt}\big) &= m^{\text{pre}}_{T-dt}\underbrace{-\big(\Delta^{\text{post}}_{T-dt}-\Delta^{\text{pre}}_{T-dt}\big)S_{T-dt}-\big|\Delta^{\text{post}}_{T-dt}-\Delta^{\text{pre}}_{T-dt}\big|S_{T-dt}\,tc}_{\text{self-financing strategy}}\\
w_T\big(\Delta^{\text{post}}_{T-dt}\big) &= \Delta^{\text{post}}_{T-dt}S_T+m^{\text{post}}_{T-dt}\big(\Delta^{\text{post}}_{T-dt}\big)P(T-dt,T)^{-1}\underbrace{-\big|\Delta^{\text{post}}_{T-dt}\big|S_T\,tc}_{\text{unwinding cost}}\\
PL_T\big(\Delta^{\text{post}}_{T-dt}\big) &= w_T\big(\Delta^{\text{post}}_{T-dt}\big)+h\pi_T
\end{aligned}
$$

3. Monte-Carlo-Simulation

We simulate a fixed seed of N different price evolutions $\{S_T^{(k)}\}_{k\in\{1,2,\dots,N\}}$.

$$\begin{aligned} w_T^{(i)}(\Delta_{T-dt}^{\text{post}}) &= \Delta_{T-dt}^{\text{post}} S_T^{(i)} + m_{T-dt}^{\text{post}}(\Delta_{T-dt}^{\text{post}}) P(T-dt,T)^{-1} - \left|\Delta_{T-dt}^{\text{post}}\right| S_T^{(i)}\, tc \\ \pi_T^{(i)} &= \max\{S_T^{(i)} - K, 0\} \\ PL_T^{(i)}(\Delta_{T-dt}^{\text{post}}) &= w_T^{(i)}(\Delta_{T-dt}^{\text{post}}) + h\pi_T^{(i)} \end{aligned}$$

4. Minimisation of the Expected Shortfall

The agent aims at minimising the expected shortfall to a significance/risk appetite α by optimally delta-hedging the exposure. This entails the convex risk measure $ESF_{T,\alpha}$, the certainty equivalent $VaR_{T,\alpha}$ and the indifference price PV_{T-dt} satisfying the relations

$$\begin{aligned} ESF_{T,\alpha} &= \inf_{VaR_{T,\alpha},\Delta_{T-dt}^{\text{post}}} VaR_{T,\alpha} + \frac{1}{\alpha}\mathbb{E}\Big[\max\Big\{-PL_T(\Delta_{T-dt}^{\text{post}}) - VaR_{T,\alpha}, 0\Big\}\Big], \\ PV_{T-dt} &= P(T-dt,T)VaR_{T,\alpha}\big| m_{T-dt}^{\text{pre}} = 0. \end{aligned}$$

Minimising the expected shortfall is equivalent to solving the equation $\nabla J(v,\delta) = 0$, where

$$\begin{aligned} PL_T^{(i)}(\delta) &= \delta S_T^{(i)} + \Big(m_{T-dt}^{\text{pre}} - (\delta - \Delta_{T-dt}^{\text{pre}})S_{T-dt} - \left|\delta - \Delta_{T-dt}^{\text{pre}}\right| S_{T-dt}\, tc\Big) P(T-dt,T)^{-1} - |\delta| S_T^{(i)}\, tc + h\pi_T^{(i)}, \\ J(v,\delta) &= v + \frac{1}{\alpha}\frac{1}{N}\sum_{i=1}^{N} \max\{-PL_T^{(i)}(\delta) - v, 0\}, \\ \frac{\partial J}{\partial v} &= 1 - \frac{1}{\alpha}\frac{1}{N}\sum_{i=1}^{N} \mathbb{1}_{\{-PL_T^{(i)} - v > 0\}}, \\ \frac{\partial J}{\partial \delta} &= \frac{1}{\alpha}\frac{1}{N}\sum_{i=1}^{N} \mathbb{1}_{\{-PL_T^{(i)} - v > 0\}} \Bigg[S_T^{(i)}\Big(tc\big(\mathbb{1}_{\{\delta>0\}} - \mathbb{1}_{\{\delta<0\}}\big) - 1\Big) \\ &\qquad + S_{T-dt} P(T-dt,T)^{-1}\Bigg(tc\Big(\mathbb{1}_{\{\delta > \Delta_{T-dt}^{\text{pre}}\}} - \mathbb{1}_{\{\delta < \Delta_{T-dt}^{\text{pre}}\}}\Big) + 1\Bigg)\Bigg]. \end{aligned}$$

Since PL_T is unaffected by the level of v, we achieve $\partial_v J = 0$ simply by choosing v as the α-quantile of the set $\{PL_T^{(i)}\}_{i=1}^N$. Denote by $I = I(\Delta_{T-dt}^{\text{post}}) \subset \{1,2,\dots,N\}$ the subset of indices that correspond to the worst α profit & loss scenarios. Regarding the second equation, the optimal δ satisfies the equation

$$\sum_{i\in I(\delta)} S_T^{(i)}\Big(1 - tc\big(\mathbb{1}_{\{\delta>0\}} - \mathbb{1}_{\{\delta<0\}}\big)\Big) = \alpha N S_{T-dt} P(T-dt,T)^{-1}\Bigg(tc\Big(\mathbb{1}_{\{\delta > \Delta_{T-dt}^{\text{pre}}\}} - \mathbb{1}_{\{\delta < \Delta_{T-dt}^{\text{pre}}\}}\Big) + 1\Bigg).$$

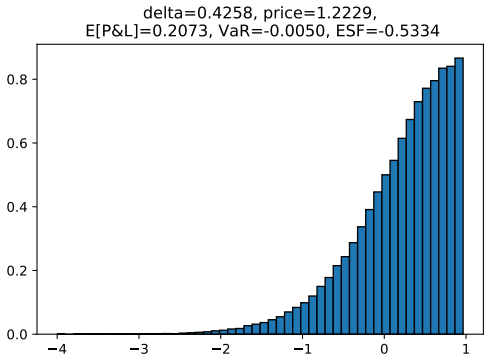
Note that the right-hand side of this equation is predictable at time $T - dt$. Therefore, an optimal hedging strategy can be found iteratively. Realistically it holds $1 - tc\big(\mathbb{1}_{\{\delta>0\}} - \mathbb{1}_{\{\delta<0\}}\big) \approx 1$. Hence, the algorithm may terminate once the deviance between the left- and the right-hand side is smaller than S_{T-dt}.

5. Validation Results

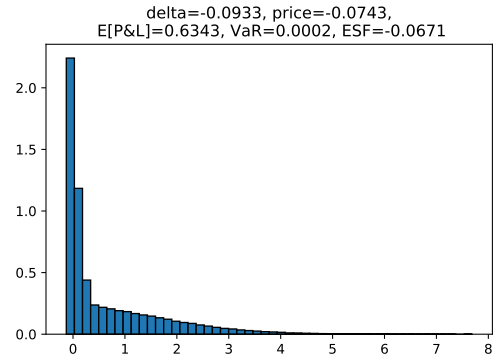
As specified in the code below, gradient descent is implemented with an exponentially decaying learning rate and with line search. The accuracy of the valuation is validated using another independent sample of 100 000 paths.

valuation date	$T - dt = 0$
maturity	$T = 1/252$
risk-free interest rate	$r = 2.00\%$
dividend rate	$q = 0$
initial price process of underlying	$S_{T-dt} = 100$
future price dynamics	GBM
drift	$\mu = 7.00\%$
volatility	$\sigma = 30.00\%$
proportional transaction cost	$tc = 0.30\%$
holding in bank account before re-allocation	$m^{\text{pre}}_{T-dt} = 0$
initial holding in underlying	$\Delta^{\text{pre}}_{T-dt} = 0$
strike	$K = 100$
holding (short, long)	$h = \pm 1$
significance (risk appetite)	$\alpha = 30\%$
number of simulations	$N = 100\,000$
learning rate in gradient descent	$\lambda = 0.2$

$h = -1$:



$h = 1$:



6. Outlook

By utilising recurrent neural networks and techniques inspired from re-inforcement learning, this simplified showcase can be generalised to the multi-step case and to more realistic features. These comprise more risk factors, stochastic rates, stochastic volatilities, traded options as well as variance swaps and, as the case may be, path-dependent payoffs. Furthermore, different target functions may be considered such as, for instance, the L^2-norm of PL_T. This would certainly imply a different concept for PV_{T-dt}.

References

[1] Bühler, H., Gonon, L., Teichmann, J., Wood B.
Deep Hedging (2018).
arXiv:1802.03042.

Appendix: Implementation in Python

```
from collections import deque
import matplotlib.pyplot as plt
import numpy

# ================================================================================="
# CLASSES & METHODS
# ================================================================================

class model_config:
    def __init__(self,dt,r,S0,mu,sigma,q,tc,alpha,nSim,episodes,precision,lr,
            lr_decay,line_search=None):
        """ discretisation, risk-free rate, initial price, drift, vol,\
        dividend, transaction cost, alpha, nSim, steps in gradient  descent,\
        minimal sup-norm of gradient, learning rate, learning rate decay,\
        optional list of learning rates"""
        self.dt=dt
        self.r=r
        self.S0=S0
        self.mu=mu
        self.sigma=sigma
        self.q=q
        self.tc=tc
        self.alpha=alpha
        self.nSim=nSim
        self.episodes=episodes
        self.precision=precision
        self.lr=lr
        self.lr_decay=lr_decay
        self.line_search=line_search

    def get_prices(self,nPaths=None,seed_user_value=1):
        if nPaths==None:
            nPaths=self.nSim
        numpy.random.seed(seed_user_value)
        return self.S0*numpy.exp((self.mu-1/2*self.sigma**2)*self.dt-self.q+\
            self.sigma*numpy.sqrt(self.dt)*numpy.random.normal(0,1,nPaths))

class state(model_config):
    def __init__(self,model_config,timestep,delta,wealth):
        self.model_config=model_config
        self.timestep=timestep
        self.delta=delta
        self.wealth=wealth

    def re_allocate(self,delta_post):
        self.wealth=self.wealth-numpy.abs(delta_post-self.delta)*\
            self.model_config.S0*self.model_config.tc
        self.delta=delta_post
```



```
    def update(self,price):
        self.wealth+=self.delta*(price-self.model_config.S0)+(self.wealth\
            -self.delta*self.model_config.S0)*(numpy.exp(self.model_config.r*\
            self.model_config.dt)-1)
        self.timestep+=1

    def single_scenario(self,price):
        return self.wealth+self.delta*(price-self.model_config.S0)+\
            (self.wealth-self.delta*self.model_config.S0)*(numpy.exp(\
            self.model_config.r*self.model_config.dt)-1)-\
            numpy.abs(self.delta)*price*self.model_config.tc

    def simulate(self,prices):
        """for a given state, calculate resulting wealth for all simulated
        prices"""
        return list(map(lambda x: self.single_scenario(x),prices))

class valuation_output:
    def __init__(self,v,dv,delta,ddelta,J):
        self.v=v
        self.dv=dv
        self.delta=delta
        self.ddelta=ddelta
        self.J=J

    def DCF(self,r,dt):
        return numpy.exp(-r*dt)*self.v

class European_call(state):
    def __init__(self,state,strike,position=-1,seed_user_value=1):
        """short position/sell option: position=-1,
        long position/buy option: position=1"""
        self.state=state
        self.strike=strike
        self.position=position
        self.seed_user_value=seed_user_value

    def payoff_script(self,prices):
        payoff=prices-self.strike
        payoff[payoff<0]=0
        return payoff

    def target(self,prices,payoff,v=None,delta_post=None):
        if v==None:
            v=-self.position*self.payoff_script(numpy.repeat(\
                self.state.model_config.S0,2))[0]
        if delta_post==None:
            delta_post=-self.position*(1 if self.state.model_config.S0>\
                self.strike else 0)
        initial_state=state(self.state.model_config,0,self.state.delta,\
            self.state.wealth)
        initial_state.re_allocate(delta_post)
```



```
        hedge=initial_state.simulate(prices)
        profit_n_loss=hedge+self.position*payoff
        risk_contribution=1/self.state.model_config.alpha*(-profit_n_loss-v)
        risk_contribution[risk_contribution<0]=0
        dv=1-1/self.state.model_config.alpha*1/self.state.model_config.nSim*\
            sum(-profit_n_loss-v>0)
        ddelta=1/self.state.model_config.alpha*1/self.state.model_config.nSim*\
            sum(numpy.multiply(-profit_n_loss-v>0,prices*(\
            self.state.model_config.tc*(1 if delta_post>0 else -1)-1)+\
            self.state.model_config.S0*numpy.exp(self.state.model_config.r*\
            self.state.model_config.dt)*(self.state.model_config.tc*(1 if \
            delta_post>self.state.delta else -1)+1)))
        return valuation_output(v,dv,delta_post,ddelta,v+\
            numpy.average(risk_contribution))

    def valuation(self,v=None,delta_post=None,nPaths=None,seed_user_value=1,\
            verbatim=0):
        if nPaths==None:
            nPaths=self.state.model_config.nSim
        prices=self.state.model_config.get_prices(nPaths,seed_user_value)
        payoff=self.payoff_script(prices)
        valuation=self.target(prices,payoff,v,delta_post)
        if verbatim==1:
            print("iteration={:}, dv={:.6f}, ddelta={:.6f}, target={:.6f}".\
                format(0,valuation.dv,valuation.ddelta,valuation.J))
        if not isinstance(self.state.model_config.line_search,list):
            lr=self.state.model_config.lr
            for k in range(1,self.state.model_config.episodes):
                valuation=self.target(prices,payoff,valuation.v-\
                    lr*valuation.dv,valuation.delta-lr*valuation.ddelta)
                if verbatim==1:
                    print("iteration={:}, lr={:.6f}, dv={:.6f}, "\
                        "ddelta={:.6f}, target={:.6f}".\
                        format(k,lr,valuation.dv,valuation.ddelta,valuation.J))
                if max(abs(valuation.dv),abs(valuation.ddelta))<\
                    self.state.model_config.precision:
                    break
                lr*=self.state.model_config.lr_decay
        else:
            for k in range(1,self.state.model_config.episodes):
                trials=deque(maxlen=len(self.state.model_config.line_search))
                temp_targets=numpy.repeat(0.,\
                    len(self.state.model_config.line_search))
                for l in self.state.model_config.line_search:
                    trials.append(self.target(prices,payoff,valuation.v-l*\
                        valuation.dv,valuation.delta-l*valuation.ddelta))
                    temp_targets[len(trials)-1]=trials[len(trials)-1].J
                lr=self.state.model_config.line_search[numpy.argmin(\
                    temp_targets)]
                if (k>1 and min(temp_targets)>valuation.J):
                    break
                else:
```

```
                    valuation=trials[numpy.argmin(temp_targets)]
                if verbatim==1:
                    print("iteration={:}, lr={:.6f}, dv={:.6f}, "\
                        "ddelta={:.6f}, target={:.6f}".\
                        format(k,lr,valuation.dv,valuation.ddelta,valuation.J))
                if max(abs(valuation.dv),abs(valuation.ddelta))<\
                    self.state.model_config.precision:
                    break
        return valuation

    def validation(self,v,delta_post,nPaths=None,seed_user_value=2,\
            nbins=50):
        if nPaths==None:
            nPaths=self.state.model_config.nSim
        prices=self.state.model_config.get_prices(nPaths,seed_user_value)
        payoff=self.payoff_script(prices)
        initial_state=state(self.state.model_config,0,self.state.delta,\
            self.state.wealth+v)
        initial_state.re_allocate(delta_post)
        hedge=initial_state.simulate(prices)
        profit_n_loss=hedge+self.position*payoff
        E=numpy.average(profit_n_loss)
        VaR=numpy.percentile(profit_n_loss,100*self.state.model_config.alpha)
        ESF=numpy.average(profit_n_loss[profit_n_loss<VaR])
        plt.hist(profit_n_loss,bins=nbins,density=True,edgecolor='black')
        plt.title("delta={:.4f}, price={:.4f},\n E[P&L]={:.4f}, VaR={:.4f}, "\
                  "ESF={:.4f}".format(delta_post,v,E,VaR,ESF))

# ================================================================================"
# EXAMPLE
# ================================================================================

if __name__ == "__main__":
    example_short=European_call(state(model_config(1/252,0.02,100.,0.07,0.3,0,\
        0.003,0.3,100000,50,5e-3,0.2,0.99,list([0.001,0.005,0.01,0.05,0.1,0.2,\
        0.5])),0,0,0),100,-1,1)
    result_short=example_short.valuation(verbatim=1)
    print("CE/VaR="+str(result_short.v))
    print("dv="+str(result_short.dv))
    print("delta="+str(result_short.delta))
    print("ddelta="+str(result_short.ddelta))
    print("ESF="+str(result_short.J))
    plt.figure(0)
    example_short.validation(result_short.DCF(\
        example_short.state.model_config.r,\
        example_short.state.model_config.dt),result_short.delta,100000,2,50)
    plt.savefig('histogram_short_option.pdf', format='pdf')

    example_long=European_call(state(model_config(1/252,0.02,100.,0.07,0.3,0,\
        0.003,0.3,100000,50,5e-3,0.2,0.99,list([0.001,0.005,0.01,0.05,0.1,0.2,\
        0.5])),0,0,0),100,1,1)
    result_long=example_long.valuation(verbatim=1)
```



```
    print("CE/VaR="+str(result_long.v))
    print("dv="+str(result_long.dv))
    print("delta="+str(result_long.delta))
    print("ddelta="+str(result_long.ddelta))
    print("ESF="+str(result_long.J))
    plt.figure(1)
    example_long.validation(result_long.DCF(\
        example_long.state.model_config.r,\
        example_long.state.model_config.dt),result_long.delta,100000,2,50)
    plt.savefig('histogram_long_option.pdf', format='pdf')
```

thomas.krabichler@hslu.ch

