Option pricing case

Alexandre Carbonneau, M. Sc.

Objectives

- Part 1: Introduction to the deep learning library Keras.
- Part 2: Supervised learning task of option pricing.
- Part 3: Two popular hyperparameter tuning approach in deep learning: grid search and random search.
- Part 4: Implementation of dropout and batchnormalization.

Part 1) Deep learning library Keras

Recall: for a pair (x, y), a single layer feedforward neural network $f_\theta(x) : \mathbb{R}^d \to \mathbb{R}^K$ is defined as follows:

$$h(x) = g\left(W^{(1)}x + b^{(1)}\right)$$
$$f_\theta(x) = o\left(W^{(2)}h(x) + b^{(2)}\right)$$

where

$$\theta = \{W^{(1)}, W^{(2)}, b^{(1)}, b^{(2)}\},$$
$$x \in \mathbb{R}^{d\times 1}, \quad W^{(1)} \in \mathbb{R}^{d_1\times d}, \quad b^{(1)} \in \mathbb{R}^{d_1\times 1},$$
$$W^{(2)} \in \mathbb{R}^{K\times d_1}, \quad b^{(2)} \in \mathbb{R}^{K\times 1}.$$

- Total number of parameters: $d_1 \times d + (K+1) \times d_1 + K$.
- Hyperparameters: **number of neurons** of the **hidden layer** d_1 and the activation function $g(.)$.
- K is the output size (depends on the type of problem).

Deep neural networks (1)

For a pair (x, y) with $L \geq 2$ layers:

$$
\begin{aligned}
h^{(1)}(x) &= g^{(1)}\left(W^{(1)}x + b^{(1)}\right) \\
h^{(2)}(x) &= g^{(2)}\left(W^{(2)}h^{(1)}(x) + b^{(2)}\right) \\
&\dots \\
h^{(L)}(x) &= g^{(L)}\left(W^{(L)}h^{(L-1)}(x) + b^{(L)}\right) \\
f_\theta(x) &= o\left(W^{(L+1)}h^{(L)}(x) + b^{(L+1)}\right)
\end{aligned}
$$

where:

$$\theta = \{W^{(1)}, \dots, W^{(L+1)}, b^{(1)}, \dots, b^{(L+1)}\},$$

- Hyperparameters: L, dimension of each hidden layer, activation functions $g^{(1)}, \dots, g^{(L)}$.

Deep neural networks (2)

```python
# 0) Sequential(): to start the compilation of the neural network with Keras
model = Sequential()

# 1) First hidden layer
model.add(Dense(units = 50, activation = 'relu', input_dim=8))

# Output layer
model.add(Dense(units=1, activation = 'relu'))

# Compile the model
model.compile(loss='mse', optimizer = 'adam')
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 50)                450
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 51
=================================================================
Total params: 501
Trainable params: 501
Non-trainable params: 0
_________________________________________________________________
```

Figure: Single layer feedforward neural network with Keras (regression task).

Deep neural networks (3)

```python
model = Sequential()

# 1) First hidden layer
model.add(Dense(units=30, activation = 'relu', input_dim = 5))

# 2) Output layer
model.add(Dense(units=5, activation = 'softmax'))

# 3) Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam')
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 30)                180
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 155
=================================================================
Total params: 335
Trainable params: 335
Non-trainable params: 0
_________________________________________________________________
```

Figure: Single layer feedforward neural network with Keras for a multiclass classification (5 classes).

Deep neural networks (4)

```python
model = Sequential()

# 1) Three hidden layers
model.add(Dense(units = 80, activation = 'relu', input_dim = 8))
model.add(Dense(units = 100, activation = 'tanh'))
model.add(Dense(units = 120, activation = 'sigmoid'))

# 2) Output layer
model.add(Dense(units = 1, activation = 'relu'))

# 3) Compile the model
model.compile(loss='mse', optimizer='adam')
model.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 80)                720

dense_2 (Dense)              (None, 100)               8100

dense_3 (Dense)              (None, 120)               12120

dense_4 (Dense)              (None, 1)                 121
=================================================================
Total params: 21,061
Trainable params: 21,061
Non-trainable params: 0
```

Figure: Multilayer feedforward neural network with Keras (regression).

Deep neural networks (5)

```
# 0) Sequential(): to start the compilation of the neural network with Keras
model = Sequential()

# 1) Compilation
model.add(Dense(units = 50, activation = 'relu', input_dim=8))
model.add(Dense(units = 70, activation = 'tanh'))
model.add(Dense(units = 110, activation = 'sigmoid'))
model.add(Dense(units = 1, activation = 'linear'))

# 2) Optimizer and loss function
model.compile(loss = 'mse', optimizer = 'sgd')
model.summary()

# 3) train the model
model.fit(X_train, Y_train, epochs = 100, batch_size = 64)
```

Figure: Training of a deep neural network with Keras

Link to the notebook

1) On the google drive folder, open Option Pricing - Day 1.pdf and go to the page 9/30.
2) Click on the following link for the notebook: `https://colab.research.google.com/drive/1Rn5pFZqvcsRfCFe6Iyxp2cY3wTPsdje9`.
3) On google colab, save the notebook on your google drive: click 'File' (top left), 'Save a copy in Drive'.

Important: **work as a team!**

Implementation

Part 1 in the notebook.

Part 2) Option pricing: a supervised learning approach

In the second part of the tutorial, we price options with neural networks under the Black-Scholes model (BSM).

- Many papers can be found doing a similar exercise.
- See e.g. `https://srdas.github.io/Papers/BlackScholesNN.pdf`

Part 2) Option pricing: a supervised learning approach

Under the BSM, the price of a European call option is known in closed-form:

$$C = Se^{-qT}\mathcal{N}(d_1) - Ke^{-rT}\mathcal{N}(d_2),$$

$$d_1 = \frac{\ln(S/K) + (r - q + \sigma^2/2)T}{\sigma\sqrt{T}}, \quad d_2 = d_1 - \sigma\sqrt{T},$$

where

- $\mathcal{N}$ is the CDF of the standardized Gaussian distribution;
- S is the current price of the underlying;
- K and T are the strike price and the maturity of the option;
- r and q are the continuous risk-free rate and dividend yield;
- σ is the volatility parameter.

Fin ML

Part 2) Option pricing: a supervised learning approach

In Culkin and Ras (2017), the authors are asking the following question:

- Can a Neural Network be used to learn the Black-Scholes European call option price?

Part 2) Option pricing: a supervised learning approach

In Culkin and Ras (2017), the authors are asking the following question:

- Can a Neural Network be used to learn the Black-Scholes European call option price?
- i.e. the price of a European call option is a function of the features $[S, K, T, r, q, \sigma]$, can we learn the mapping from these features to the price C?

Methodology

Table 1: The range of parameters used to simulate 300,000 call option prices. The strike prices K were chosen to lie within the vicinity of the stock price S, so as to be realistic.

Parameter	Range
Stock price (S)	\$10 – \$500
Strike price (K)	\$7 – \$650
Maturity (T)	1 day to 3 years
Dividend rate (q)	0% – 3%
Risk free rate (r)	1% – 3%
Volatility (σ)	5% – 90%
Call price (C)	\$0 – \$328

- Simulate 100,000 pairs (X, Y) where $X = [S, K, T, q, r, \sigma]$ and the target Y is the call price under BSM.

Methodology

Table 1: The range of parameters used to simulate 300,000 call option prices. The strike prices K were chosen to lie within the vicinity of the stock price S, so as to be realistic.

Parameter	Range
Stock price (S)	\$10 – \$500
Strike price (K)	\$7 – \$650
Maturity (T)	1 day to 3 years
Dividend rate (q)	0% – 3%
Risk free rate (r)	1% – 3%
Volatility (σ)	5% – 90%
Call price (C)	\$0 – \$328

- Simulate 100,000 pairs (X, Y) where $X = [S, K, T, q, r, \sigma]$ and the target Y is the call price under BSM.
- Split into 60,000 call options for the Training set (in-sample) to fit the neural network, 20,000 for the Valid set for hyperparameters tuning and 20,000 for the Test set (out-of-sample) to evaluate the performance of the model.

Implementation

Part 2 in the notebook.

Part 3) Optimization of deep neural networks

The most important objective in machine learning: to be able to **generalize** over new (unseen) examples.

- In-sample performance is **not** the primary objective.

Part 3) Optimization of deep neural networks

The most important objective in machine learning: to be able to **generalize** over new (unseen) examples.

- In-sample performance is **not** the primary objective.
- **Out-of-sample** performance is the most important in machine learning!

Hyperparameter search in deep learning (1)

Suppose you have a dataset $\mathcal{D} = (X^{(i)}, Y^{(i)})_{i=1:N}$.

$$\mathcal{D} \longrightarrow \{\mathcal{D}_{train}, \mathcal{D}_{valid}, \mathcal{D}_{test}\}.$$

- $\mathcal{D}_{train}$: training of the **parameters** (i.e. fit the model).

Hyperparameter search in deep learning (1)

Suppose you have a dataset $\mathcal{D} = (X^{(i)}, Y^{(i)})_{i=1:N}$.

$$\mathcal{D} \longrightarrow \{\mathcal{D}_{train}, \mathcal{D}_{valid}, \mathcal{D}_{test}\}.$$

- $\mathcal{D}_{train}$: training of the **parameters** (i.e. fit the model).
- $\mathcal{D}_{valid}$: **hyperparameter tuning**.

Hyperparameter search in deep learning (1)

Suppose you have a dataset $\mathcal{D} = (X^{(i)}, Y^{(i)})_{i=1:N}$.

$$\mathcal{D} \longrightarrow \{\mathcal{D}_{train}, \mathcal{D}_{valid}, \mathcal{D}_{test}\}.$$

- $\mathcal{D}_{train}$: training of the **parameters** (i.e. fit the model).
- $\mathcal{D}_{valid}$: **hyperparameter tuning**.
- $\mathcal{D}_{test}$: estimate the **generalization performance** (i.e. out-of-sample performance).

Hyperparameter search in deep learning (2)

- D_{valid} is used to **choose all hyperparameters**: number of hidden layers, number of neurons per layer, activation function for each hidden layer, learning rate, stochastic gradient descent (SGD) optimizer, number of epochs, etc.

Hyperparameter search in deep learning (2)

- D_{valid} is used to **choose all hyperparameters**: number of hidden layers, number of neurons per layer, activation function for each hidden layer, learning rate, stochastic gradient descent (SGD) optimizer, number of epochs, etc.
- Conclusion: the hyperparameter space is **extremely large** and testing every combination is **not** computationally feasible.

Hyperparameter search in deep learning (3)

General optimization procedure:

1) Choose a set of hyperparameters (important step, will be covered in the next few slides).
2) Train the parameters θ **only** with $\mathcal{D}_{train}$ (SGD).
3) Repeat steps 1) – 2) for a fixed number of combinations of hyperparameters.
4) Choose the model that **minimizes the loss** on D_{valid}.
5) Finally, evaluate the generalization performance on D_{test}.

Grid search

Method 1: **Grid search**. Define a grid of possible combinations of hyperparameters and test **each combination** of this grid.
Example:

- Number of hidden layers: $\{2, 3, 4\}$; number of neurons: $\{100, 105, 110, \ldots, 195\}$; activation functions: $\{\text{RELU}, \text{sigmoid}, \tanh\}$; optimizer: $\{\text{Adam}, \text{RMSprop}, \text{Adagrad}\}$.

Grid search

Method 1: **Grid search**. Define a grid of possible combinations of hyperparameters and test **each combination** of this grid.
Example:

- Number of hidden layers: $\{2, 3, 4\}$; number of neurons: $\{100, 105, 110, \ldots, 195\}$; activation functions: $\{\text{RELU}, \text{sigmoid}, \tanh\}$; optimizer: $\{\text{Adam}, \text{RMSprop}, \text{Adagrad}\}$.
- Simple to implement, but **computationally very expensive**.

Grid search

Method 1: **Grid search**. Define a grid of possible combinations of hyperparameters and test **each combination** of this grid.
Example:

- Number of hidden layers: $\{2, 3, 4\}$; number of neurons: $\{100, 105, 110, \ldots, 195\}$; activation functions: $\{\text{RELU}, \text{sigmoid}, \text{tanh}\}$; optimizer: $\{\text{Adam}, \text{RMSprop}, \text{Adagrad}\}$.
- Simple to implement, but **computationally very expensive**.
- For more information: see Goodfellow et al. [4], chapter 11.4.3.

Random search

Method 2: **Random search**. Define the boundaries of each hyperparameter. This method simply consists in choosing **a set of hyperparameters randomly** among these boundaries (i.e. random sampling of a subset of hyperparameters).

- Difference with **Grid search**: does not test all combinations of the grid, **only a subset**, i.e. a maximum number of iterations.

Random search

Method 2: **Random search**. Define the boundaries of each hyperparameter. This method simply consists in choosing **a set of hyperparameters randomly** among these boundaries (i.e. random sampling of a subset of hyperparameters).

- Difference with **Grid search**: does not test all combinations of the grid, **only a subset**, i.e. a maximum number of iterations.
- Conclusion: **simple to implement and powerful**. Known to be a favorable method over Grid search.

Random search

Method 2: **Random search**. Define the boundaries of each hyperparameter. This method simply consists in choosing **a set of hyperparameters randomly** among these boundaries (i.e. random sampling of a subset of hyperparameters).

- Difference with **Grid search**: does not test all combinations of the grid, **only a subset**, i.e. a maximum number of iterations.
- Conclusion: **simple to implement and powerful**. Known to be a favorable method over Grid search.
- For more information: original paper of Bergstra and Bengio [1]. Also Goodfellow et al. [4], chapter 11.4.4.

Implementation

Part 3 in the notebook.

Part 4) Dropout and batchnormalization

For this part, go directly in section 4 of the notebook.

Bibliography

1) Bergstra. J. and Bengio, Y. (2012) *Random search for hyper-parameter optimization*.
2) Goodfellow, I., Bengio, Y. and Courville, A. (2015). *Deep Learning*.

Fin ML