TRANSFORMERS PARA GERAÇÃO DE SUPERFÍCIES SUAVES DE VOLATILIDADE IMPLÍCITA

Rafael Zimmer
Institute of Mathematical Science and Computing
University of São Paulo - Brazil
rafael.zimmer@usp.br

Adalton Sena Filho
Institute of Mathematical Science and Computing
University of São Paulo - Brazil
adaltonsena@usp.br

Eduardo Brito
Institute of Mathematical Science and Computing
University of São Paulo - Brazil
eduardobritocusp.br

4 de junho de 2024

ABSTRACT

Neste trabalho implementamos uma rede neural com arquitetura baseada em *Transformers* para otimização de valores P, Q, R para diversos modelos *SABR* paramétricos (*parametric stochastic* α, β, ρ) para geração de superfícies de volatilidade implicita no contexto de opções financeiras. A motivação por trás do trabalho é permitir a interpolação e extrapolação suave da superfície, ou seja, tornando-a derivável em todos os pontos. O desenvolvimento da arquitetura tem como principal contribuição solucionar o problema na escolha dos candidatos $\alpha, \rho, volvol$ cujos parâmetros P, Q, R otimizados gerem a superfície com menor oportunidade de arbitragem. Por fim, utilizaremos os valores históricos observados de $\alpha, \rho, volvol$ e a capacidade inata das arquiteturas de transformers para processar observações de sequências com tamanhos não fixos, ou seja, para volatilidades de contratos com data de vencimento variadas e não fixas e gerar as superfícies, comparando os resultados com métodos de interpolação clássicos (linear e *splines*).

Keywords First keyword · Second keyword · More

1 Introdução

1.1 Uma breve introdução à volatilidade implícita e a reconstrução de superfícies

A superfície de volatilidade implícita (SVI) desempenha um papel crítico em várias aplicações financeiras, desde o cálculo de requisitos de margem até a precificação de derivativos exóticos. Garantir previsões precisas da SVI é fundamental, pois até pequenos erros podem levar a implicações financeiras significativas. Modelos tradicionais para a SVI muitas vezes lutam para capturar efetivamente a dinâmica do mercado e carecem da flexibilidade para valorar instrumentos sem preços cotados.

Neste artigo, propomos uma abordagem para enfrentar os desafios de interpolação e extrapolação da SVI, reduzindo oportunidades de arbitragem. O método escolhido visa implementar arquiteturas de *transformers*, que ao contrário das redes neurais convencionais, aceitam quantidades variáveis de parâmetros históricos para modelos de superfícies suaves. A motivação para a abordagem escolhida decorre da dificuldade na escolha dos candidatos para otimização do modelo *Stochastic Alpha, Beta, Rho* (*SABR*), que usualmente é feita manualmente por analistas. O uso de modelos capazes de receber como entrada pontos em nuvem (*point clouds* em inglês, quando abordado no contexto de arquiteturas de Transformers) permite filtrar pontos candidatos para serem utilizados pelos otimizadores não-lineares. O processo de otimização e filtragem será abordado mais a fundo na seção de desenvolvimento.

1.2 Contexto e antecedentes

Em finanças, opções representam contratos financeiros que concedem ao titular o direito de comprar (opção de compra) ou vender (opção de venda) um ativo a um preço e maturidade (ou data) predeterminados. A precificação de opções tradicionalmente depende de modelos como a fórmula de Black-Scholes, que, embora amplamente utilizada, possui limitações e pressupostos inerentes que podem não capturar totalmente as realidades do mercado. O desafio está em construir um mapeamento contínuo da volatilidade, conhecida como SVI, a partir de um conjunto finito de preços de opções observados, garantindo a ausência de oportunidades de arbitragem.

A literatura existente explora o uso de Redes Neurais (*Neural Networks* ou NN em inglês) para suavização da volatilidade implícita, no entanto, abordagens convencionais de NNs aproximam os valores da volatilidade diretamente, o que gera oportunidades de arbitragem por gerar superfícies cujos pontos observáveis não condizem com os existentes no mercado.

O modelo SABR é uma alternativa para esse problema, sendo possível aproximar os parâmetros α, β, ρ em vez da volatilidade em si. Para isso, contudo é necessário ter-se um conjunto de pontos observados para realizar uma otimização desses valores em função da volatilidade observada. Essa questão gera diversos problemas, devido a necessidade de uma escolha boa (que minimize o erro e desvio padrão para com a volatilidade observada) de observações para ser utilizada. Essa dificuldade será o foco no desenvolvimento do modelo deste trabalho, e será discutida mais a fundo na seção de desenvolvimento.

1.3 Contribuições

1. **Desenvolvimento de um modelo SABR com cálculo vetorizado de volatilidade implícita**: A implementação proposta do modelo SABR calcula eficientemente as volatilidades implícitas usando operações vetorizadas.
2. **Integração com uma abordagem de ajuste de parâmetros baseada em transformer**: O modelo transformer é empregado para ajustar os parâmetros do SABR, capturando padrões complexos nos dados históricos de opções.
3. **Geração de superfícies de volatilidade suaves**: O modelo combinado gera superfícies de volatilidade diárias para o conjunto de dados de opções do ativo americano SPY, replicando a dinâmica das superfícies históricas.

2 Desenvolvimento

2.1 Formulação do Modelo SABR

O modelo SABR é definido por um conjunto de equações diferenciais estocásticas que descrevem a dinâmica do preço a termo e sua volatilidade. Os parâmetros do modelo incluem:

- **Alpha (α)**: O nível inicial da volatilidade.
- **Beta (β)**: O parâmetro de elasticidade que determina a dependência da volatilidade em relação ao preço do ativo.
- **Rho (ρ)**: A correlação entre o preço do ativo e sua volatilidade.
- **Volvol (ν)**: A volatilidade do processo de volatilidade.

A função de volatilidade implícita do modelo SABR é dada por:

$$\sigma_{BS}(f,K) = \alpha \frac{(fK)^{\frac{1-\beta}{2}}}{1 + \left(\frac{(1-\beta)^2\rho^2}{24} + \frac{(1-\beta)^2(2-3\rho^2)}{1920}\right)(fK)^{1-\beta}} \tag{1}$$

2.2 Modelo Transformer para Ajuste de Parâmetros

Transformers são um tipo de arquitetura de rede neural conhecida por sua capacidade de capturar dependências complexas em dados sequenciais. Neste projeto, um transformer encoder é usado para ajustar os parâmetros do modelo SABR (P, Q e R), correspondendo a α, ρ e ν respectivamente. O transformer processa dados históricos de opções e produz estimativas de parâmetros que minimizam o erro entre as volatilidades implícitas do modelo e as volatilidades de mercado observadas.

2.3 Detalhes da Implementação

2.3.1 Classe do Modelo SABR

A classe ‘SABRModel‘ implementa o modelo SABR, fornecendo métodos para calcular preços a termo, volatilidades implícitas e ajustar os parâmetros do modelo. Os métodos principais incluem:

- ‘forward()‘: Calcula o preço a termo dado o tempo até o vencimento, preço à vista, taxa livre de risco e rendimento de dividendos.
- ‘z()‘: Calcula a variável intermediária z usada na fórmula de volatilidade do SABR.
- ‘x()‘: Calcula a variável intermediária x usada na fórmula de volatilidade do SABR.
- ‘ivol()‘: Calcula a volatilidade implícita para parâmetros de modelo e condições de mercado dadas.
- ‘fit()‘: Ajusta os parâmetros do modelo SABR aos dados de mercado observados usando técnicas de otimização.

2.4 Arquitetura do Transformer

A arquitetura do transformer é conhecida por sua capacidade de capturar dependências complexas em dados sequenciais, utilizando um mecanismo de atenção para ponderar a importância de diferentes partes da sequência. A seguir, descrevemos a implementação de um modelo transformer em PyTorch, destacando os componentes principais: a atenção própria (Self-Attention), o bloco do transformer (Transformer Block) e o encoder do transformer (Transformer Encoder).

2.4.1 Atenção Própria

A classe `SelfAttention` implementa o mecanismo de atenção própria, que aceita dados contínuos e calcula a atenção para diferentes cabeças. A atenção é calculada através das matrizes de valor (`value`), chave (`key`) e consulta (`query`), resultando em uma saída ponderada pela importância de cada parte da sequência.

$$\textit{Attention}(Q,K,V) = \textit{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2}$$

A implementação inclui:

```
class SelfAttention(nn.Module):
 def __init__(self, in_features, heads):

  self.head_dim = in_features // heads
  self.in_features = in_features
  self.heads = heads

  self.W_v = nn.Linear(self.head_dim, self.head_dim, bias=False)
  self.W_k = nn.Linear(self.head_dim, self.head_dim, bias=False)
  self.W_q = nn.Linear(self.head_dim, self.head_dim, bias=False)

  self.fc_out = nn.Linear(in_features, in_features)

 def forward(self, value, key, query, mask=None):
  N = query.shape[0]
  value_len, key_len, query_len = value.shape[1], key.shape[1], query.shape[1]

  value = value.reshape(N, value_len, self.heads, self.head_dim)
  key = key.reshape(N, key_len, self.heads, self.head_dim)
  query = query.reshape(N, query_len, self.heads, self.head_dim)

  values = self.W_v(value)
  keys = self.W_k(key)
  queries = self.W_q(query)

  energy = torch.einsum("nqhd,nkhd->nhqk", [queries, keys])
  if mask is not None:
  energy = energy.masked_fill(mask == 0, float("-1e20"))

  attention = torch.softmax(energy / (self.head_dim ** (1 / 2)), dim=3)
  out = torch.einsum("nhql,nlhd->nqhd", [attention, values]).reshape(N, query_len, self.in_features)

  out = self.fc_out(out)
  return out
```

2.4.2 Bloco do Transformer

A classe `TransformerBlock` implementa um bloco do transformer, composto por um mecanismo de atenção própria, normalização em camadas (LayerNorm) e uma rede feed-forward. Este bloco é a unidade básica que será repetida várias vezes no encoder:

```
class TransformerBlock(nn.Module):
 def __init__(self, in_features, heads, forward_expansion):
  self.attention = SelfAttention(in_features, heads)
  self.norm1 = nn.LayerNorm(in_features)
  self.norm2 = nn.LayerNorm(in_features)

  self.feed_forward = nn.Sequential(
   nn.Linear(in_features, forward_expansion * in_features),
   nn.ReLU(),
   nn.Linear(forward_expansion * in_features, in_features)
  )

 def forward(self, x, mask=None):
  attention = self.attention(x, x, x, mask)
  x = self.norm1(attention + x)
  forward = self.feed_forward(x)
  out = self.norm2(forward + x)
```

```
  return out
```

2.4.3 Encoder do Transformer

A classe `TransformerEncoder` implementa o encoder do transformer, composto por vários blocos do transformer. Este encoder processa a sequência de entrada, aplicando múltiplas camadas de atenção própria e redes feed-forward:

```
class TransformerEncoder(nn.Module):
 def __init__(self, in_features, heads, num_layers, forward_expansion, dropout, out_features):
  self.layers = nn.ModuleList([
   TransformerBlock(in_features, heads, forward_expansion) for _ in range(num_layers)
  ])

  self.dropout = nn.Dropout(dropout)
  self.fc_out = nn.Linear(in_features, out_features)

 def forward(self, x, mask=None):
  for layer in self.layers:
   x = layer(x, mask)
  return self.fc_out(x)
```

2.4.4 Classe SABR Paramétrica

A classe 'ParametricSABR' estende o modelo SABR introduzindo funções de parâmetros dependentes do tempo para α, ρ e ν. Essas funções são ajustadas aos dados históricos usando técnicas de otimização, e a classe fornece métodos para gerar superfícies de volatilidade suaves.

3 Results

4 Conclusão

4.1 Resumo

Este trabalho apresentou uma abordagem para ajustar os parâmetros do modelo SABR utilizando uma rede neural baseada na arquitetura de transformers (especificamente o elemento do *encoder*). A combinação do modelo SABR com a arquitetura transformer foi utilizada para replicar padrões da superfície de volatilidade implicita histórica, resultando reconstrução suave e sem oportunidade de arbitragem para os preços diários observados. A utilização de operações vetorizadas no cálculo de volatilidades implícitas e a integração com um método de ajuste de parâmetros baseado em transformer demonstraram melhorias significativas na modelagem da volatilidade diária, relativo à eficiência em escolher os candidatos para otimização manualmente.

4.2 Implicações e Trabalhos Futuros

A capacidade de gerar superfícies de volatilidade diárias suaves apoia diversas aplicações, incluindo a precificação de opções, a gestão de riscos e o trading de derivativos. Além disso, a abordagem apresentada destaca o potencial dos transformers na modelagem de dados financeiros complexos, abrindo caminho para novas pesquisas e inovações no campo da engenharia financeira.

Pesquisas futuras podem explorar a integração de fatores de mercado adicionais e arquiteturas de redes neurais alternativas para aprimorar ainda mais a precisão das previsões de volatilidade. Estender o modelo para outros instrumentos financeiros e mercados proporcionará uma validação mais ampla de sua eficácia. Além disso, investigações sobre a adaptação do modelo a diferentes regimes de mercado e condições econômicas variáveis podem revelar novas oportunidades de aplicação e desenvolvimento do método proposto.

Referências