تکلیف اول درس شبکه‌های عصبی

وحید ملکی
شماره دانشجویی: ۴۰۳۱۳۰۰۴

۷ دی ۱۴۰۴

سوال ۵

مقدمه و هدف پروژه

در این بخش از تکلیف، هدف تولید ۱۰۰ نمونه داده از تابع داده‌شده $F(x)$، ذخیره آن‌ها در فایل اکسل، و سپس استفاده از این داده‌ها برای بهینه‌سازی پارامترهای یک مدل پارامتریک با استفاده از ده الگوریتم بهینه‌سازی مختلف است. مدل پارامتریک برای تخمین $F(x)$ تعریف شده و بهینه‌سازی در سه حالت انجام می‌شود: بهینه‌سازی فقط پارامترهای α، فقط پارامترهای θ، و بهینه‌سازی همزمان هر دو گروه. معیار ارزیابی، خطای میانگین مربعات (MSE) است و نتایج در جدولی خلاصه می‌شوند. در نهایت، الگوریتم‌ها مقایسه شده و بهترین روش مشخص می‌شود.
تابع اصلی تولید داده:

$$F(x) = 3\log\mathrm{sig}(1.7x) + 4\tan\mathrm{sig}(3x) + 4\,\mathrm{Swish}(2.5x) + x^4 - 2x^2 + 0.1x + 1$$

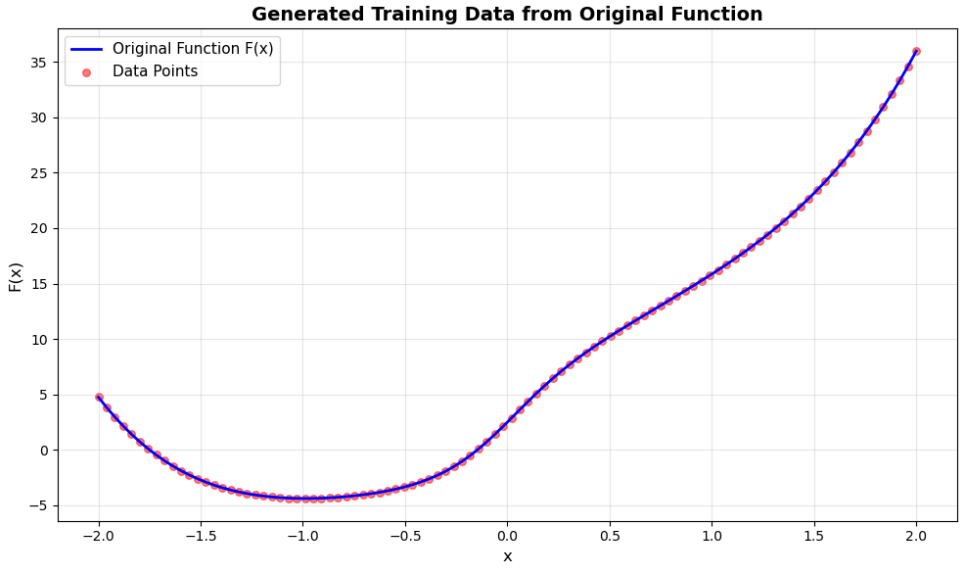
مدل پارامتریک برای تخمین:

$$\hat{F}(x) = \alpha_1\log\mathrm{sig}(\theta_1 x) + \alpha_2\tan\mathrm{sig}(\theta_2 x) + \alpha_3\,\mathrm{Swish}(\theta_3 x) + (\alpha_4 x^2 - 1) + \alpha_5 x$$

روش‌شناسی

تولید و ذخیره داده‌ها

۱۰۰ نمونه داده در بازه $[-2, 2]$ از تابع $F(x)$ تولید شد. این داده‌ها شامل مقادیر x و $y = F(x)$ هستند و در فایل generated_data.xlsx ذخیره شدند. برای تولید داده‌ها از تابع generate_and_save_data در فایل utils.py استفاده شد که داده‌ها را به صورت تصادفی اما یکنواخت تولید می‌کند.
نمودار داده‌های تولید شده در شکل ۱ نشان داده شده است.



شکل ۱: داده‌های تولید شده از تابع اصلی $F(x)$

تعریف مدل و تابع هزینه

مدل $\hat{F}(x)$ با ۵ پارامتر α و ۳ پارامتر θ تعریف شد. مقدار اولیه تمام پارامترها به صورت تصادفی در بازه $[0, 1]$ انتخاب می‌شود. تابع هزینه، خطای میانگین مربعات (MSE) است:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

گرادیان‌ها نسبت به پارامترها با استفاده از قاعده زنجیره‌ای و مشتقات تحلیلی محاسبه می‌شوند تا دقت و سرعت افزایش یابد.

الگوریتم‌های بهینه‌سازی

ده الگوریتم بهینه‌سازی از پایه پیاده‌سازی شدند. در ادامه، توضیح کامل هر الگوریتم همراه با فرمول‌ها آورده شده است. تمام الگوریتم‌ها با نرخ یادگیری اولیه ۰.۰۵۰۰ و ۵۰۰۰ تکرار (Epochs) اجرا شدند.

۱. **SGD (Stochastic Gradient Descent):** ساده‌ترین روش: وزن‌ها مستقیماً با گرادیان به‌روزرسانی می‌شوند.

$$w_{t+1} = w_t - \eta \nabla_w \mathcal{L}$$

که η نرخ یادگیری است. ساده اما ممکن است در مینیمم محلی گیر کند.

۲. **Momentum:** مانند SGD اما با اضافه کردن ممنتم برای عبور از مینیمم‌های محلی.

$$v_{t+1} = \gamma v_t - \eta \nabla_w \mathcal{L}, \quad w_{t+1} = w_t + v_{t+1}$$

$\gamma = 0.9$ برای حفظ ممنتم.

۳. **Nesterov (Nesterov Accelerated Gradient)**: نسخه بهبودیافته Momentum با نگاه به جلو.

$$v_{t+1} = \gamma v_t - \eta \nabla_w \mathcal{L}(w_t + \gamma v_t), \quad w_{t+1} = w_t + v_{t+1}$$

همگرایی سریع‌تر.

۴. **AdaGrad (Adaptive Gradient)**: نرخ یادگیری برای هر پارامتر تطبیقی می‌شود.

$$G_{t+1} = G_t + (\nabla_w \mathcal{L})^2, \quad w_{t+1} = w_t - \frac{\eta}{\sqrt{G_{t+1} + \epsilon}} \nabla_w \mathcal{L}$$

مناسب برای داده‌های پراکنده اما نرخ یادگیری ممکن است خیلی کوچک شود.

۵. **RMSProp (Root Mean Square Propagation)**: بهبود AdaGrad با میانگین متحرک.

$$E[g^2]_{t+1} = \beta E[g^2]_t + (1-\beta)(\nabla_w \mathcal{L})^2, \quad w_{t+1} = w_t - \frac{\eta}{\sqrt{E[g^2]_{t+1} + \epsilon}} \nabla_w \mathcal{L}$$

$\beta = 0.999$.

۶. **AdaDelta**: مانند RMSProp اما بدون نیاز به نرخ یادگیری اولیه.

$$E[\Delta w^2]_{t+1} = \beta E[\Delta w^2]_t + (1-\beta)(\Delta w)^2$$

نرخ یادگیری تطبیقی بر اساس تغییرات گذشته.

۷. **Adam (Adaptive Moment Estimation)**: ترکیب Momentum و RMSProp.

$$m_{t+1} = \beta_1 m_t + (1-\beta_1)\nabla_w \mathcal{L}, \quad v_{t+1} = \beta_2 v_t + (1-\beta_2)(\nabla_w \mathcal{L})^2$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_{t+1}} + \epsilon} \hat{m}_{t+1}$$

$\beta_2 = 0.999$ ،$\beta_1 = 0.9$.

۸. **AdaMax**: نسخه Adam با نرم بی‌نهایت.

$$u_{t+1} = \max(\beta_2 u_t, |\nabla_w \mathcal{L}|)$$

۹. **Nadam (Nesterov-accelerated Adaptive Moment Estimation)**: Adam با Nesterov Momentum.
مشابه Adam اما با نگاه به جلو.

۱۰. **AMSGrad**: بهبود Adam برای جلوگیری از همگرایی ضعیف.

$$\hat{v}_{t+1} = \max(\hat{v}_t, v_{t+1})$$

پیاده‌سازی پروژه

پروژه به صورت ماژولار در چهار فایل پیاده‌سازی شد:
- activations.py: توابع فعال‌سازی و مشتقات (logsig، tansig، swish و مشتقات).
- Optimizers.py: کلاس‌های بهینه‌ساز (هر الگوریتم در یک کلاس Stateful).
- utils.py: توابع تولید داده، مدل، MSE و محاسبه گرادیان.
- main.ipynb: اجرای آزمایش‌ها، ذخیره نتایج و رسم نمودارها.

کد اصلی در main.ipynb با ۵۰۰۰ تکرار و نرخ یادگیری . اجرا شد.

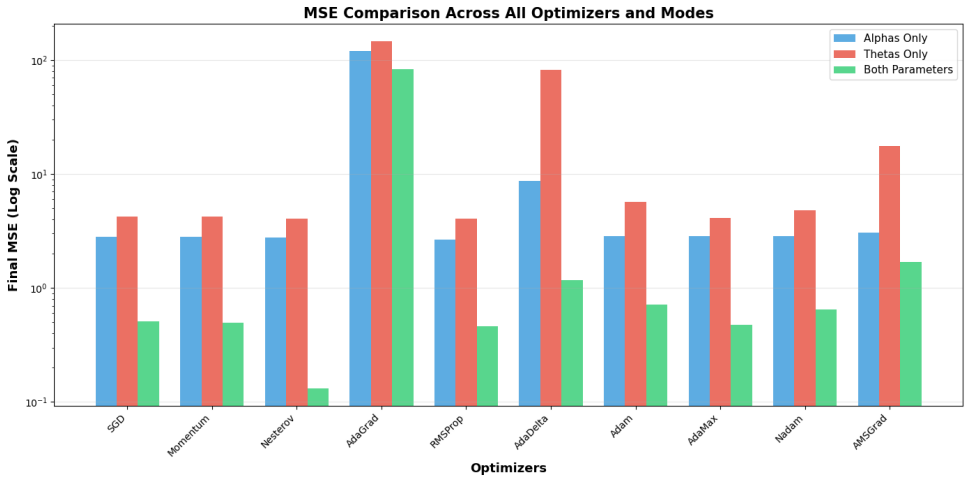
نتایج

جدول نتایج MSE:

جدول ۱: نتایج خطای MSE برای هر الگوریتم و حالت

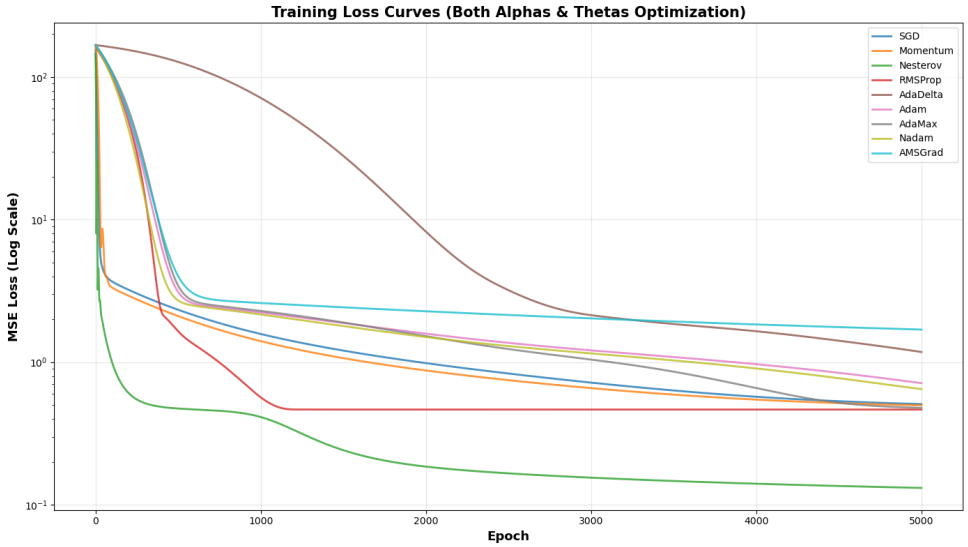
MSE (Both Alphas & Thetas)	MSE (Thetas Only)	MSE (Alphas Only)	Optimizer
0.508155	4.218111	2.821886	SGD
0.498051	4.216349	2.821888	Momentum
0.131215	4.086758	2.786854	Nesterov
83.280737	146.319219	119.956749	AdaGrad
0.465062	4.049449	2.647066	RMSProp
1.178379	82.237825	8.753923	AdaDelta
0.712995	5.697448	2.844417	Adam
0.477942	4.124761	2.835573	AdaMax
0.646712	4.824294	2.843917	Nadam
1.693381	17.705722	3.067494	AMSGrad

نمودار مقایسه MSE در شکل ۲.



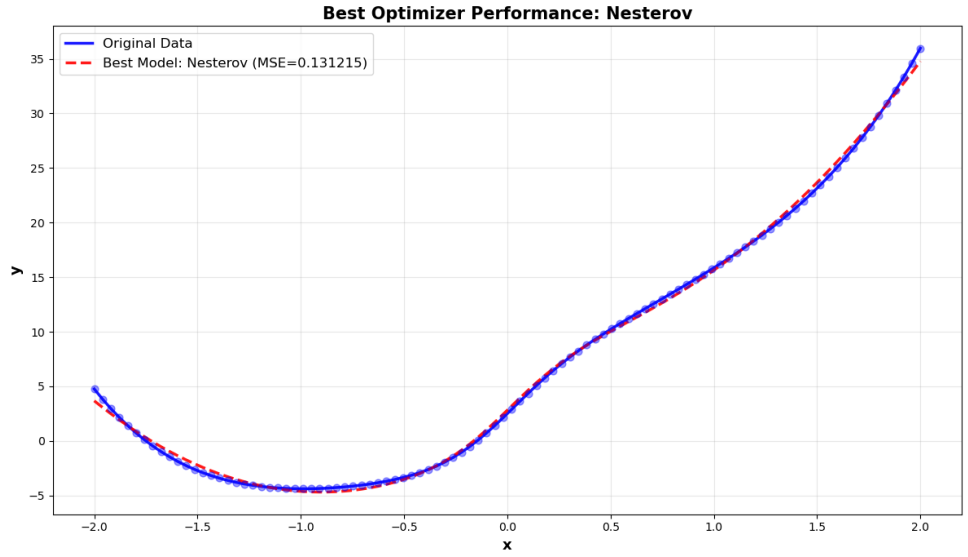
شکل ۲: مقایسه MSE در حالت‌های مختلف

نمودار منحنی‌های خطا در حالت همزمان در شکل ۳.



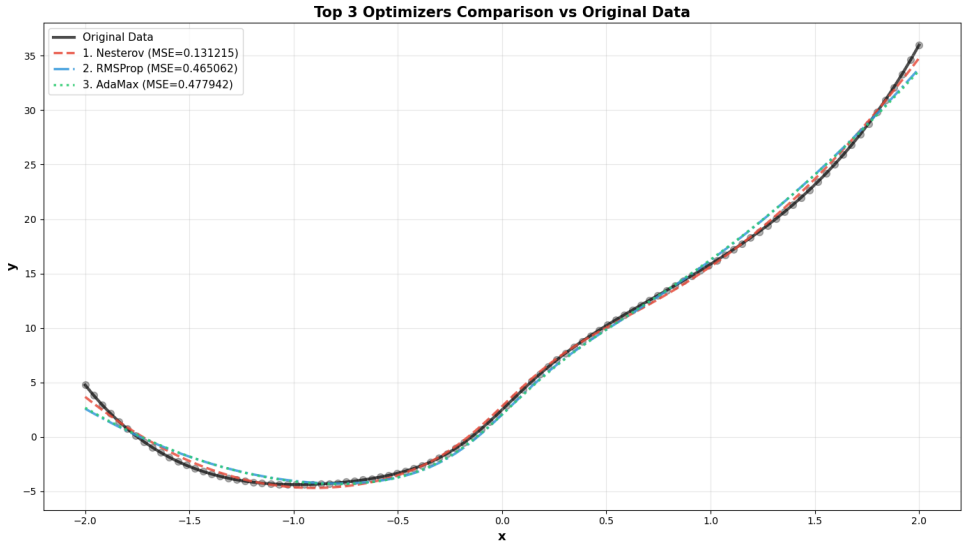
شکل ۳: منحنی‌های خطا برای حالت همزمان

برازش بهترین مدل (Nesterov) در شکل ۰۴



شکل ۴: برازش بهترین مدل

مقایسه سه مدل برتر در شکل ۰۵



شکل ۵: مقایسه سه مدل برتر

تحلیل و مقایسه

- **بهینه‌سازی همزمان بهترین است:** MSE پایین‌تر در حالت همزمان نشان‌دهنده تعامل بین α و θ است.
- **بهترین الگوریتم: Nesterov** با MSE=0.131215 در حالت همزمان، به دلیل ممنتم نگاه به جلو، همگرایی سریع و عبور از مینیمم محلی.
- **عملکرد ضعیف AdaGrad و AdaDelta:** نرخ یادگیری کوچک می‌شود و گیر می‌کنند.
- **الگوریتم‌های تطبیقی مانند RMSProp و AdaMax:** خوب اما نه به اندازه Nesterov.

پارامترهای بهترین مدل: $\alpha_1 = 7.64$، $\alpha_2 = 4.07$، $\alpha_3 = -3.81$، $\alpha_4 = 6.22$، $\alpha_5 = 8.59$، $\theta_1 = 1.04$، $\theta_2 = 2.98$، $\theta_3 = 2.28$.

نتیجه‌گیری

الگوریتم‌های مبتنی بر ممنتم مانند Nesterov بهترین عملکرد را داشتند. پیاده‌سازی از پایه نشان داد که انتخاب الگوریتم تأثیر مستقیم بر کیفیت دارد. برای مسائل مشابه، Nesterov توصیه می‌شود.