دانشگاه صنعتی امیرکبیر
(پلی‌تکنیک تهران)
دانشکده ی ریاضی و علوم کامپیوتر

پایان‌نامه کارشناسی‌ارشد

گرایش علوم کامپیوتر

مقایسه روشهای فراابتکاری برای بهینه سازی ضرایب شبکه عصبی

گزارش هشتم

نگارش

پویا پارسا

استاد راهنما

دکتر قطعی

خرداد ۱۴۰۰


بسم الله الرحمن الرحيم

فصل اول

دیتاست

۱-۱ مقدمه

در این گزارش سعی بر بررسی ساز و کار بهینه ساز های متفاوت و هم چنین استفاده از آن ها در آموزش یک شبکه عصبی ساده پرداخته ایم. دو بهینه ساز مد نظر PSO که جزء روش های شاخص بدون گرادیان و از آن طرف بهینه ساز Adam که در زمره ی بهترین روش گرادیان محور قرار می گیرند بوده اند.

۱-۲ اطلاعات کلی

به دلیل تمرکز این گزارش بر روی بهینه ساز ها ، از استفاده ی دیتاست پیچیده پرهیز کرده ام و از دیتاست معروف و قدیمی Iris استفاده کرده ام. این دیتاست اطلاعات سه نوع گل را دربردارد و شامل ۱۵۰ رکورد می باشد و برای هر رکورد ۴ فیچر شامل اطلاعاتی در مورد طول گلبرگ و کاسبرگ و ... را داراست.

۱-۳ معیار استفاده شده

معیار استفاده شده برای ارزیابی نتایج برای هر دو بهینه سازی Accuracy بوده است. که در گزارش قبل به دلیل انتخاب آن اشاره شده است.

فصل دوم

پیاده سازی

۲-۱ بهینه سازی ازدحام ذرات یا PSO

کد های استفاده شده در این گزارش از طریق این لینک در دسترس است.

Particle swarm optimization یا PSO [۵] یک روش جمعیت محور تصادفی برای بهینه سازی است در این روش از گرادیان استفاده نمی شود و در زمره ی روش های Swarm Intelligence قرار می گیرد. مزایای این روش به طور موردی به شرح زیر است :

- لزومی ندارد تابع هدف مشتق پذیر باشد.
- تعداد هایپر پارامتر های خیلی کمی دارد
- به راحتی می توان آن را درک کرد
- و بر روی طیف وسیعی از مسائل می توان آن را به کار برد.

به نظر می رسد این روش الهام گرفته از رفتار اجتماعی پرندگان و دسته های بزرگ ماهی برای پیدا کردن غذا است.

کلیت این روش به این شکل است که تعدادی ذره به طور تصادفی در فضای جستجو پراکنده می شوند؛ در ابتدا جهت حرکت و سرعت آن ها نیز به طور تصادفی تعیین می شود سپس در هر مرحله جهت و سرعت ذرات با توجه به بهترین جواب پیدا شده توسط خود ذره تا آن لحظه و همچنین بهترین جواب پیدا شده توسط کل جمعیت به روز می شود. به بیان دقیق تر:

$$P_i^{t+1} = P_i^t + V_i^{t+1}$$

$$V_i^{t+1} = \underbrace{wV_i^t}_{\text{Inertia}} + \underbrace{c_1 r_1 (P_{best(i)}^t - P_i^t)}_{\text{Cognitive (Personal)}} + \underbrace{c_2 r_2 (P_{bestglobal}^t - P_i^t)}_{\text{Social (Global)}}$$

شکل ۲-۱: نحوه تعیین سرعت ذره i در iteration t+1 .

در فرمول بالا r1, r2 ضرایبی تصادفی هستند که رفتار ذره ما را تعیین می کنند در واقع چه میزان به ندای درون و چه میزان به پیام های جمعی توجه کند! دقت داشته باشید که r1, r2 هایپرپارمتر نیستند و برای هر ذره و در هر مرحله متفاوتند

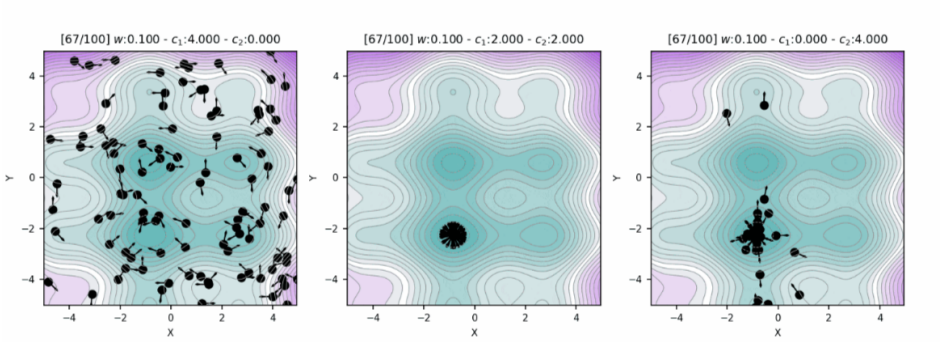
۲-۱-۱ هایپرپارامتر ها

۱. تمایل به تغییر سرعت یا inertia weight

در واقع قدرت تغییر مسیر ذره ها را مشخص می کند هر چه این پارامتر بزرگتر باشد گردش و اکتشاف ذرات بیشتر و هر چه کمتر باشد همگرایی قوی تر و بهره وری از نقاط یافت بیشتر است.

۲. تاثیر پذیری از جمع یا خود c1, c2

c1 میزان تاثیر پذیری از پاسخ یافت شده توسط خود ذره و در مقابل آن c2 میزان تاثیر گرفتن ذرات از بهترین پاسخ یافت شده را مشخص می کنند. یک تعادل مناسب بین این دو مقدار باعث می شود هم ذرات فضای اطراف خود را جستجو کنند و هم به سمت بهترین جواب های پیدا شده همگرا شوند.



شکل ۲-۲: بررسی تاثیر c1 و c2 های متفاوت

۲-۱-۲ کد

کد های استفاده شده از [۳] هستند و در این بخش بیشتر به توصیف آن ها می پردازم.

```python
def logits_function(p):
    """ Calculate roll-back the weights and biases

    Inputs
    ------
    p: np.ndarray
        The dimensions should include an unrolled version of the
        weights and biases.

    Returns
    -------
    numpy.ndarray of logits for layer 2

    """
    # Roll-back the weights and biases
    W1 = p[0:80].reshape((n_inputs,n_hidden))
    b1 = p[80:100].reshape((n_hidden,))
    W2 = p[100:160].reshape((n_hidden,n_classes))
    b2 = p[160:163].reshape((n_classes,))

    # Perform forward propagation
    z1 = X.dot(W1) + b1  # Pre-activation in Layer 1
    a1 = np.tanh(z1)     # Activation in Layer 1
    logits = a1.dot(W2) + b2 # Pre-activation in Layer 2
    return logits          # Logits for Layer 2
```

شکل ۲-۳: نحوه ی بازگردان ذره از حالت برداری به وزن های شبکه

برای ساخت ذره در واقع ما وزن های بین لایه ها (به همراه بایاس) را در یک بردار تک بعدی می ریزیم و به نوعی ساختار شبکه ی عصبی را صاف می کنیم. هر ذره برداری است با ابعاد وزن های شبکه عصبی. و در هر مرحله برای محاسبه ی fitness ذره بردار متناظر با آن را به ساختار شبکه ی عصبی آن برمی گردانیم. این عملیات در تکه کد بالا انجام می شود:

و با انجام یک forward propagation و محاسبه ی خطا میزان بهینگی ذره را به دست می آوریم. همانطور که می بینید ابتدا با استفاده از Softmax خروجی را تبدیل به یک توزیع احتمالی می کنیم و سپس از طریق تابع خطایمان که negative likelihood است و ضابطه ی این چنینی دارد

$$L_i = -\log(p_{y_i})$$

که در آن p_{y_i} مقدار احتمال حساب شده توسط مدل برای کلاس صحیح است.

```python
# Forward propagation
def forward_prop(params):
    """Forward propagation as objective function

    This computes for the forward propagation of the neural network, as
    well as the loss.

    Inputs
    ------
    params: np.ndarray
        The dimensions should include an unrolled version of the
        weights and biases.

    Returns
    -------
    float
        The computed negative log-likelihood loss given the parameters
    """

    logits = logits_function(params)

    # Compute for the softmax of the logits
    exp_scores = np.exp(logits)
    probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)

    # Compute for the negative log likelihood

    corect_logprobs = -np.log(probs[range(num_samples), y])
    loss = np.sum(corect_logprobs) / num_samples

    return loss

```

شکل ۲-۴: محاسبه ی خطا برای یک ذره.

در انتها با استفاده از کتاب خانه ی PySwarms به جستجوی وزن های بهینه می پردازیم و با استفاده از معیار Accuracy دقت مدل را می سنجیم همان طور که می بینید بر روی دیتاست ساده ی Iris به دقت ۹۹ درصد رسیدیم که نشان دهنده کارا بودن این الگوریتم است.

```python
# Initialize swarm
options = {'c1': 0.5, 'c2': 0.3, 'w':0.9}

# Call instance of PSO
dimensions = (n_inputs * n_hidden) + (n_hidden * n_classes) + n_hidden + n_classes
optimizer = ps.single.GlobalBestPSO(n_particles=100, dimensions=dimensions, options=options)

# Perform optimization
cost, pos = optimizer.optimize(f, iters=1000)
```

```python
def predict(pos):
    """
    Use the trained weights to perform class predictions.

    Inputs
    ------
    pos: numpy.ndarray
        Position matrix found by the swarm. Will be rolled
        into weights and biases.
    """
    logits = logits_function(pos)
    y_pred = np.argmax(logits, axis=1)
    return y_pred
```

```python
(predict(pos) == y).mean()
```

```
0.9933333333333333
```

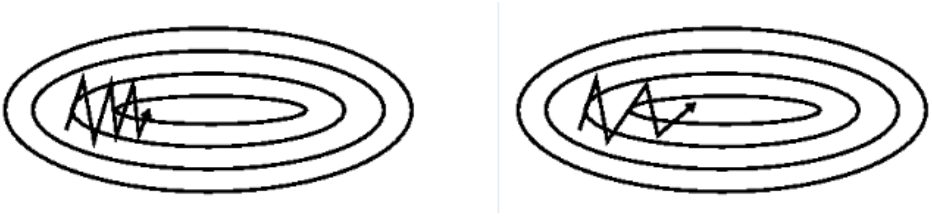
شکل ۲-۵: استفاده از کتاب خانه ی PySwarms و محاسبه ی دقت.

۲-۲ بهینه ساز Adam [۲]

Adam از جمله روش های تدریجی (Iterative) که در واقع بسط داده شده ی روش Stochastic Gradient Descent است. SGD روشی بر پایه گرادیان است که به شدت در کاربر های بینایی ماشین و NLP به کار گرفته شده است. می دانیم که شبکه های عصبی را می توان تابعی از ورودی ها به خروجی ها دید روش SGD با مشتق گیری از تابع خطا نسبت به وزن های شبکه، با استفاده از مشتق زنجیره ای اقدام به آپدیت کردن وزن های شبکه می کند.

تا اینجا همه ی توضیحات درباره ی SGD بود اما Adam دقیقا چگونه کار می کند ؟ Adam که از کلمات adaptive moment estimation برگرفته شده است بر خلاف SGD که learning rate ثابتی را برای کل روند train در نظر می گیرد؛ learning rate متمایزی را بر هر یک از پارامتر های شبکه نگهداری می کند و هر یک از این پارامتر ها را با توجه به تغییرات گرادیان سازگار و آپدیت می کند. در واقع شاید بتوان گفت روش Adam مزیت های روش AdaGrad و RMSProp را با هم داراست

در واقع به جای آن که مانند RMSProp تنها از Momentum مرتبه اول استفاده کند از Momentum مرحله ی دوم نیز استفاده می کند. در توضیح Momentum نیز باید بگوییم به جای آنکه تنها گرادیان در مرحله ی آخر را در نظر بگیریم می توانیم گرادیان های مرحله ی قبل را نیز دخیل کنیم که این کار منجر به حذف نویز از گرادیان می شود و تشکیل یک میانگین هندسی از گرادیان ها می شود.



شکل ۲-۶: استفاده از momentum و تاثیر آن در جهت گرادیان [۴].

هایپر پارامتر ها

۱. آلفا

همان میزان تاثیر گرادیان جدید در محاسبه آپدیت وزن هاست

۲. beta1

ضریب کاهش گشتاور مرتبه ی اول در میانگین هندسی

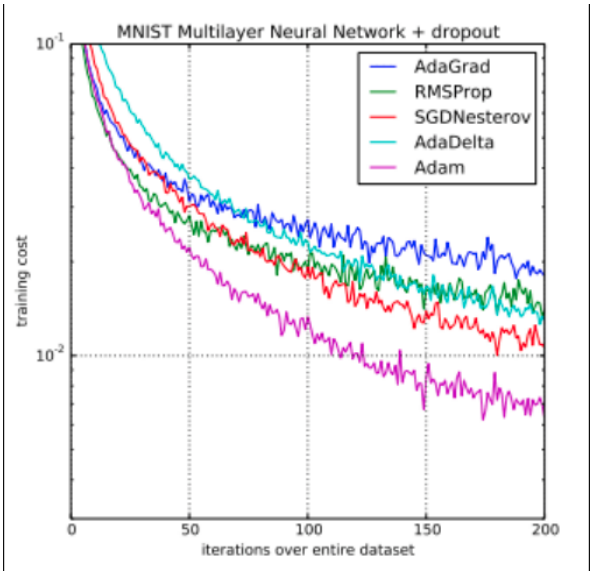
۳. beta2

ضریب کاهش گشتاور مرتبه ی دوم در میانگین هندسی

۴. epsilon

یک عدد بسیار کوچک برای جلوگیری از تقسیم بر صفر در طی عملیات.

با وجود الگوریتم های مشابه که نام آنان در بالا ذکر شد توصیه ی بسیاری از متخصصان بر استفاده از Adam در وهله ی اول می باشد [۲] در تصویر پایین عملکرد بهینه ساز های مختلف را در آموزش یک شبکه ی پرسپترون چند لایه می بینید.



شکل ۲-۷: عملکرد بهینه ساز های متفاوت در آموزش.

۲-۲-۱ کد

کد استفاده شده از [۱] می باشد. با استفاده از یک ساختار شبکه عصبی مشابه اما این بار در pytorch به استفاده از بهینه ساز Adam می پردازیم

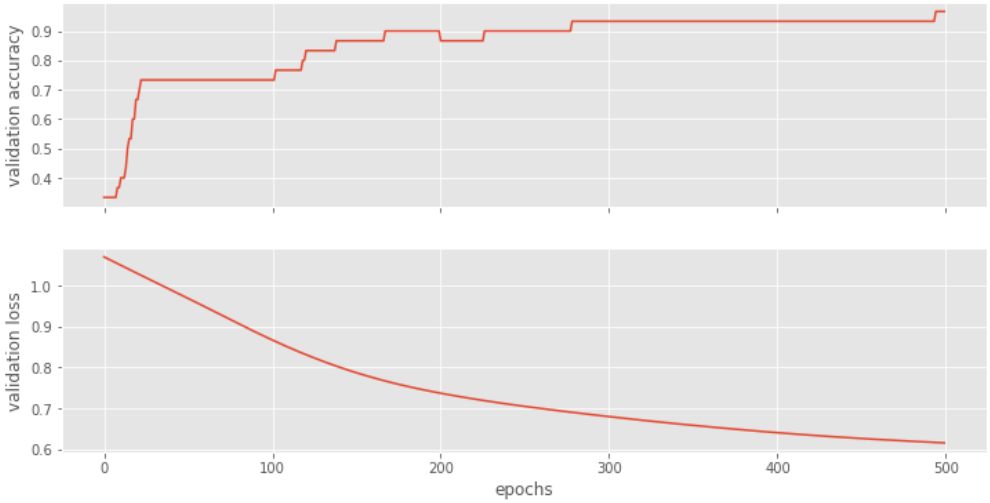
```
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.layer1 = nn.Linear(n_inputs, n_hidden)
        self.layer2 = nn.Linear(n_hidden, n_classes)

    def forward(self, x):
        x = F.relu(self.layer1(x))
        x = F.softmax(self.layer2(x), dim=1)
        return x
```

```
model     = Model()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
loss_fn   = nn.CrossEntropyLoss()
model
```

```
Model(
  (layer1): Linear(in_features=4, out_features=20, bias=True)
  (layer2): Linear(in_features=20, out_features=3, bias=True)
)
```

شکل ۲-۸: پیاده سازی شبکه و استفاده از بهینه ساز adam

همچنین روند یادگیری مدل را در نمودار زیر می توانیم مشاهده کنیم. همان طور که دیده می شود دقت تقریبا مشابه ای را بر روی داده های تست به دست می دهد.



شکل ۲-۹: روند بهبود در آموزش توسط Adam

۲-۳ جمع بندی

در آزمایش های انجام شده در این گزارش روش PSO هم سرعت همگرایی بیشتری و هم دقت بالاتری را به نمایش گذاشت البته به دلیل کوچک بودن مدل و دیتاست باید آزمایش های تکمیلی انجام داد.

منابع و مراجع

[1] Brownlee, Jason. Classifying the iris data set with pytorch. https://janakiev.com/blog/pytorch-iris/.

[2] Brownlee, Jason. Gentle introduction to the adam optimization algorithm for deep learning. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/: :text=Adam

[3] pyswarms. Pyswarms training a neural network. https://pyswarms.readthedocs.io/en/latest/examples/usecases/train_neural_network.html.

[4] RUDER, SEBASTIAN. An overview of gradient descent optimization algorithms. https://ruder.io/optimizing-gradient-descent/index.htmladam.

[5] Thevenot, Axel. Particle swarm optimization (pso) visually explained. https://towardsdatascience.com/particle-swarm-optimization-visually-explained-46289eeb2e14.