بسمه تعالی

تمرین ۲ شبکه های عصبی

سید محمدرضا حسینی

۹۷۲۴۳۱۲۹

۱) در این تمرین یک شبکه عصبی یک لایه پنهان از پایه با استفاده از numpy نوشته شده.

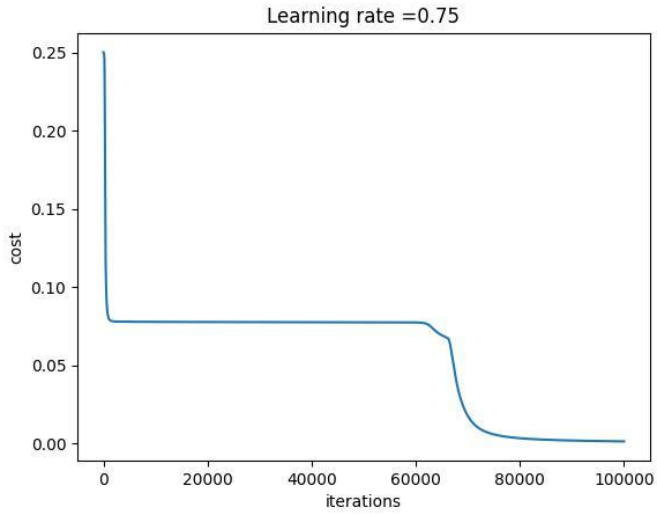
```
x, y = datasets.make_moons(n_samples=1000, shuffle=True, noise=0.1, random_state=None)
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)
hidden_layer_activations = {0: "sigmoid", 1: "tanh"}
output_layer_activations = {0: "sigmoid", 1: "tanh", 2: "linear"}
hidden_activation = hidden_layer_activations.get(1)  # choosing activation function of hidden layer
output_activation = output_layer_activations.get(1)  # choosing activation function of hidden layer
learning_processes = {0: "Batch", 1: "Online"}
learning = learning_processes.get(0)  # choosing learning process
Use_momentum = False
Use_decay_rate = True
```

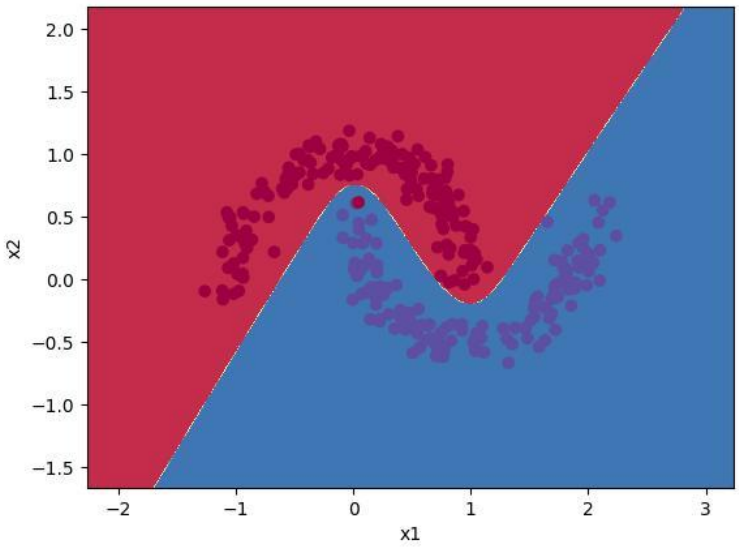
در ابتدای کد متغیر های قرار داده شده که به توان به راحتی تابع فعالیت هر لایه و نحوه آموزش را تنظیم کرد . دو متغیر بولین نیز قرار داده شده که که ثابت ماندن یا تغییر کردن نرخ یادگیری و استفاده از مومنتوم در آن ست میشود.

```
parameters = initialize(n_x, n_h, n_y)
for i in range(0, num_iterations):
    A2, cache = forward_propagation(X, parameters)
    cost = compute_cost(A2, Y)
    costs.append(cost)
    grads = backward_propagation(parameters, cache, X, Y)
    parameters, momentum = update_parameters(parameters, grads, learning_rate, momentum)
    if i != 0 and i % 1000 == 0 and Use_decay_rate:
        learning_rate = learning_rate * lr_decay
        # learning_rate =initial_learning_rate*(1/(1+lr_decay*i))
        # learning_rate = (lr_decay ** i) * initial_learning_rate
    # if cost < learning_bias:
    #     break
    if print_cost and i % 100 == 0:
        print("Cost after iteration %i: %f" % (i, cost))
        # if i > 1000:
        #     diff = 0
        #     for j in (reversed(range(len(costs) - 15, len(costs)))):
        #         diff += abs(costs[j] - costs[j - 1])
        #     if diff < 0.0001 and costs[len(costs) - 1] < 0.01:
        #         break
```

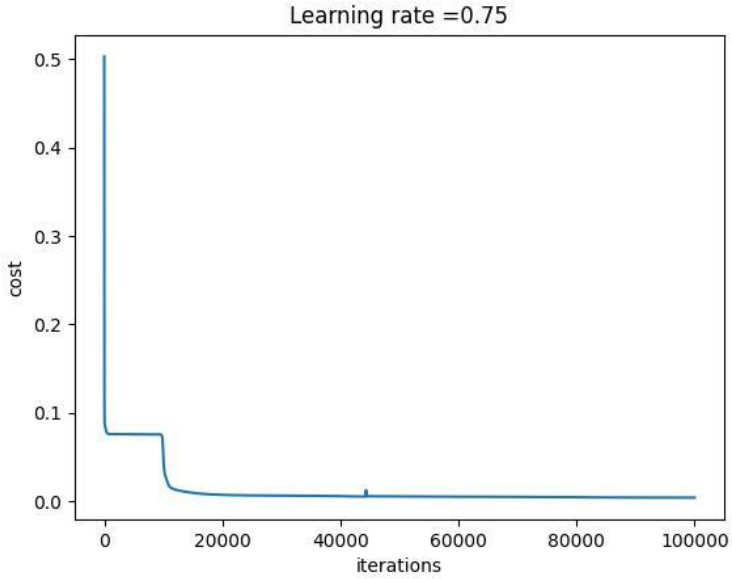
شیوه کار بدین صورت است که ابتدا مقادیر وزن ها و بایاس مقدار دهی اولیه شده و در هر حلقه بسته به نوع آموزش ابتدا forward propagation و سپس cost محاسبه میشود . بعد از آن نیز در تابع backward_propagation گرادیان نسبت به هر یک از w, b ها محاسبه میشود . در انتها نیز در تابع update_parameters نرخ w,b ها با توجه به backpropagation آپدیت میشود.

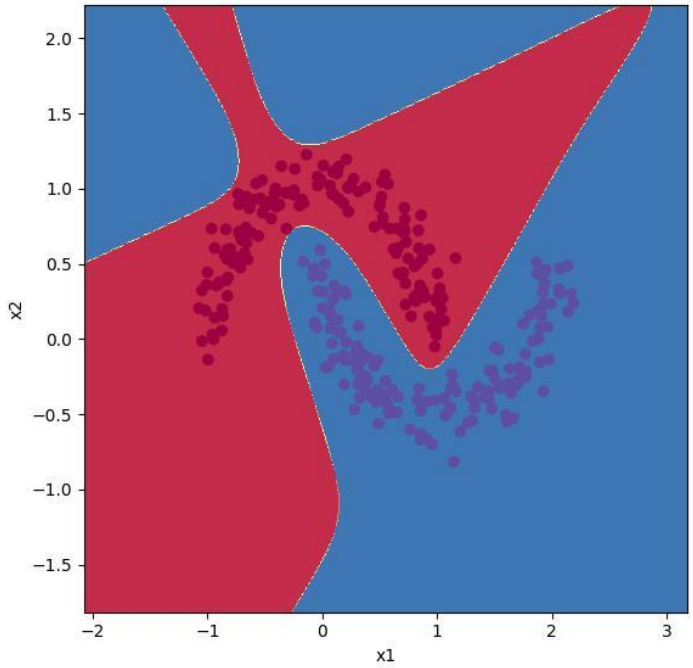
در توابع بخشی پیاده سازی شده که در صورت همگرا شدن آموزش و تغییر کم در epoch ۱۵ آموزش به اتمام برسد ولی با توجه به توضیحات این بخش کامنت شده است.



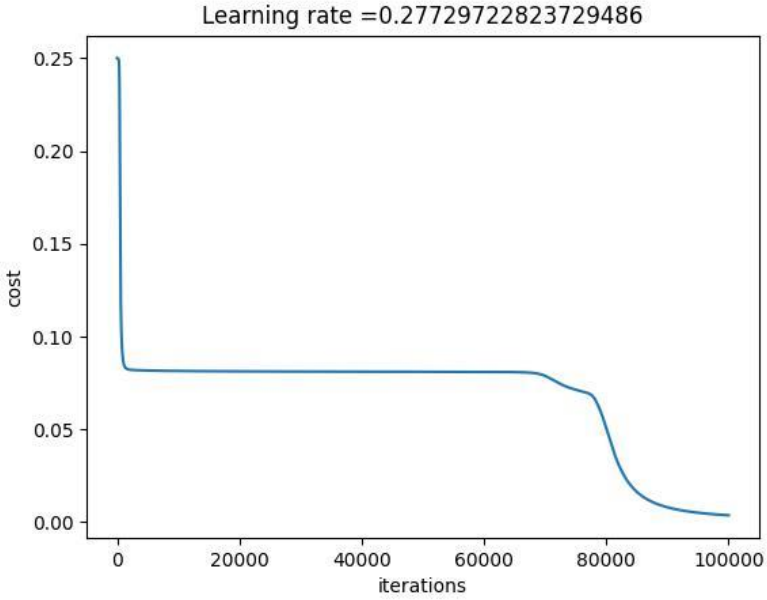


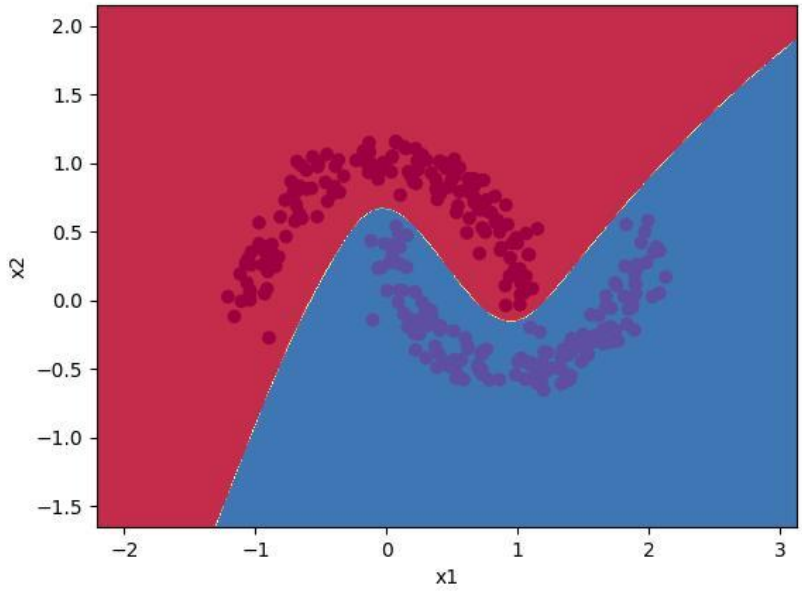
تابع فعالیت ها سیگموییید بوده ، آموزش به صورت batch و نرخ یادگیری ثابت



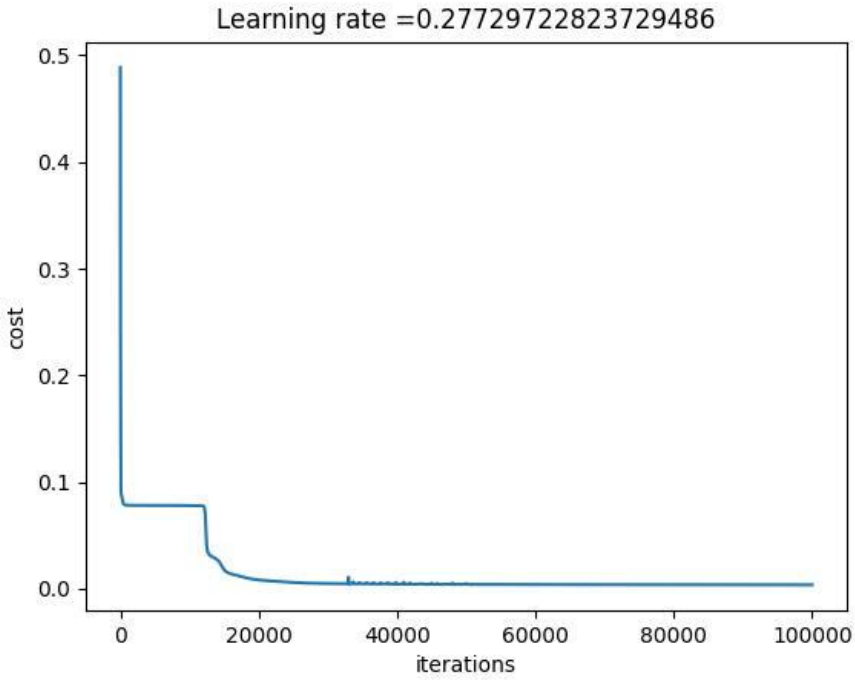


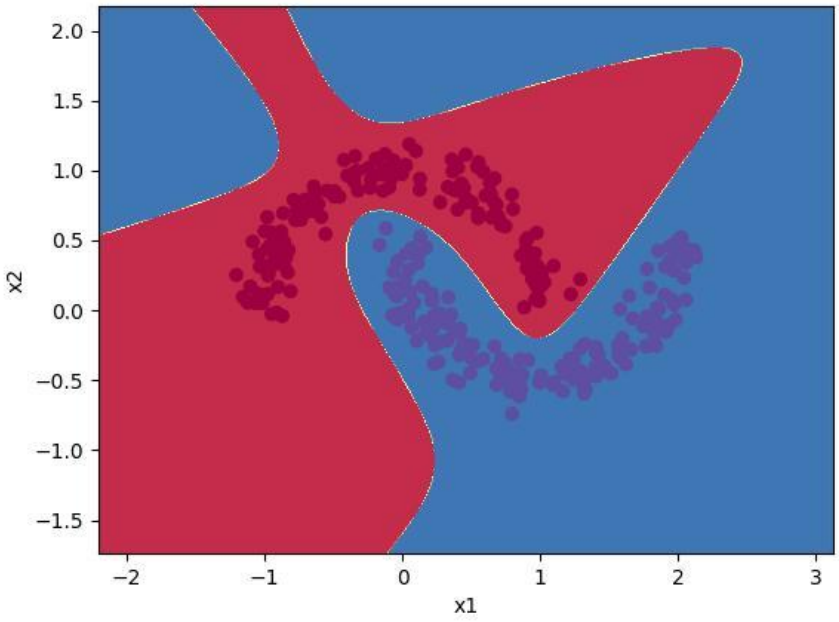
تابع فعالیت tanh ، batch learning و نرخ یادگیری ثابت



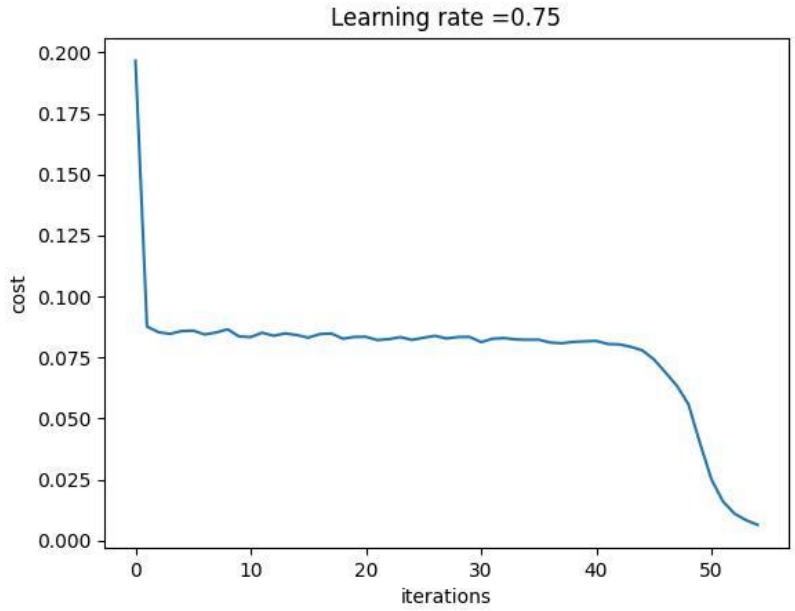


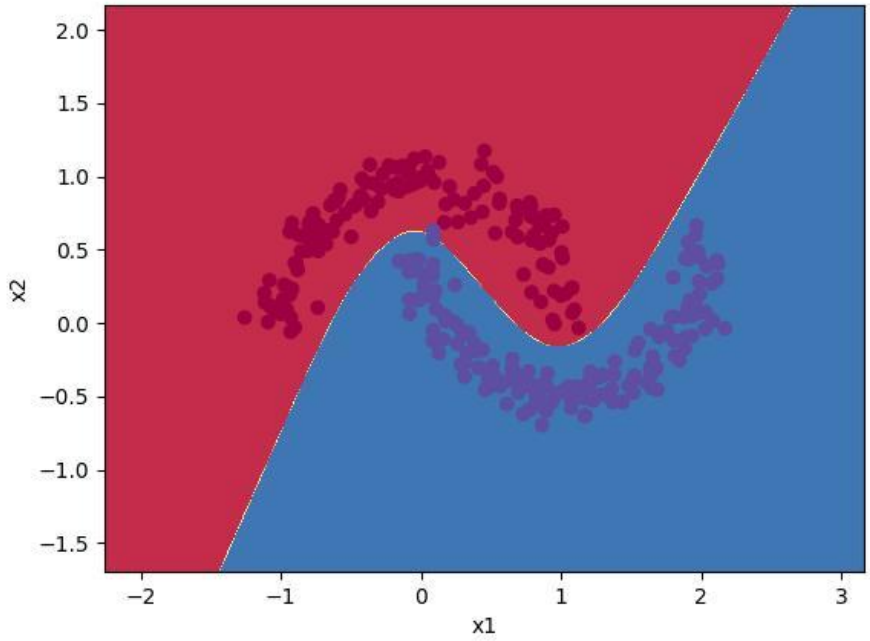
تابع فعالیت sigmoid ، batch learning ، نرخ یادگیری متغیر



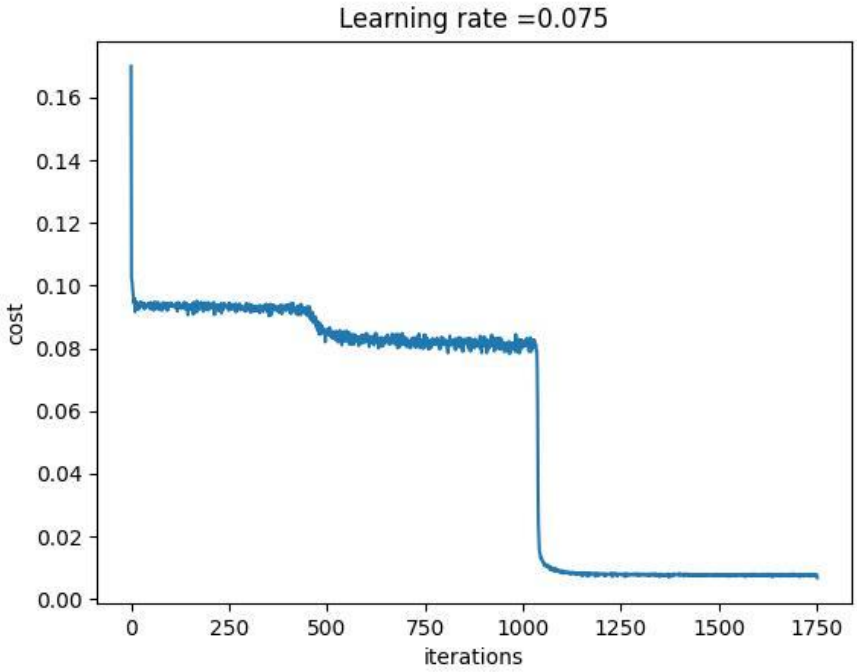


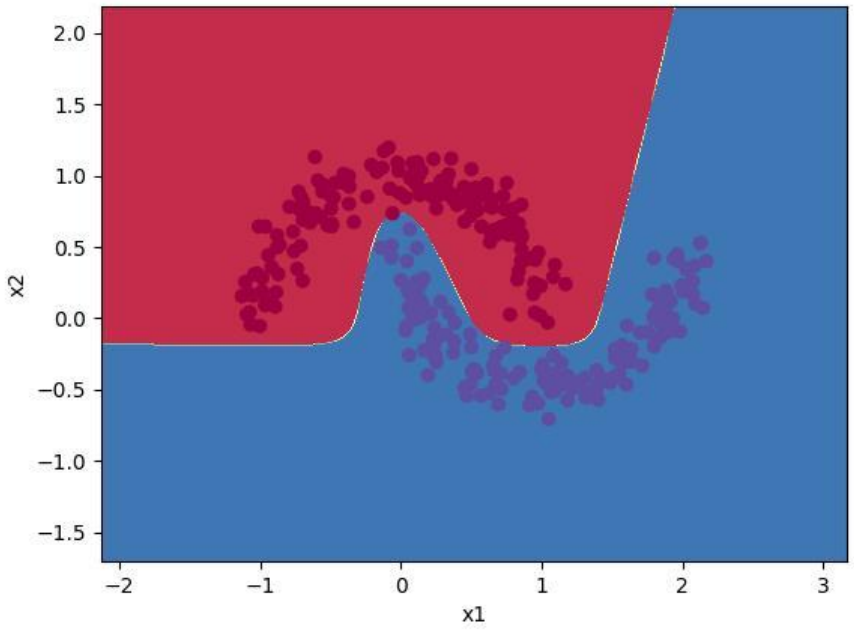
تابع فعالیت tanh ، batch learning ، نرخ یادگیری متغیر



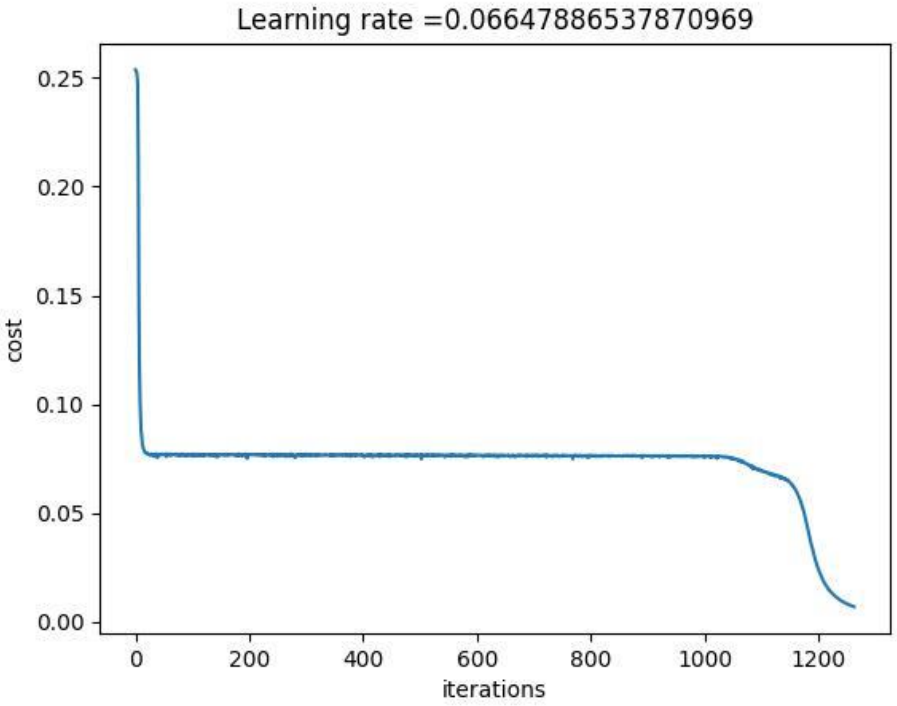


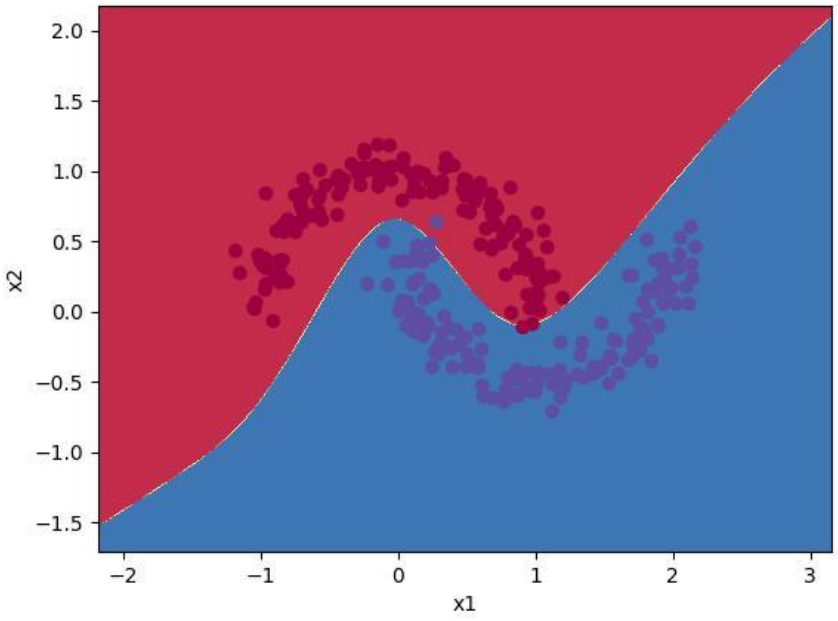
تابع فعالیت sigmoid ، online learning ، نرخ یادگیری ثابت



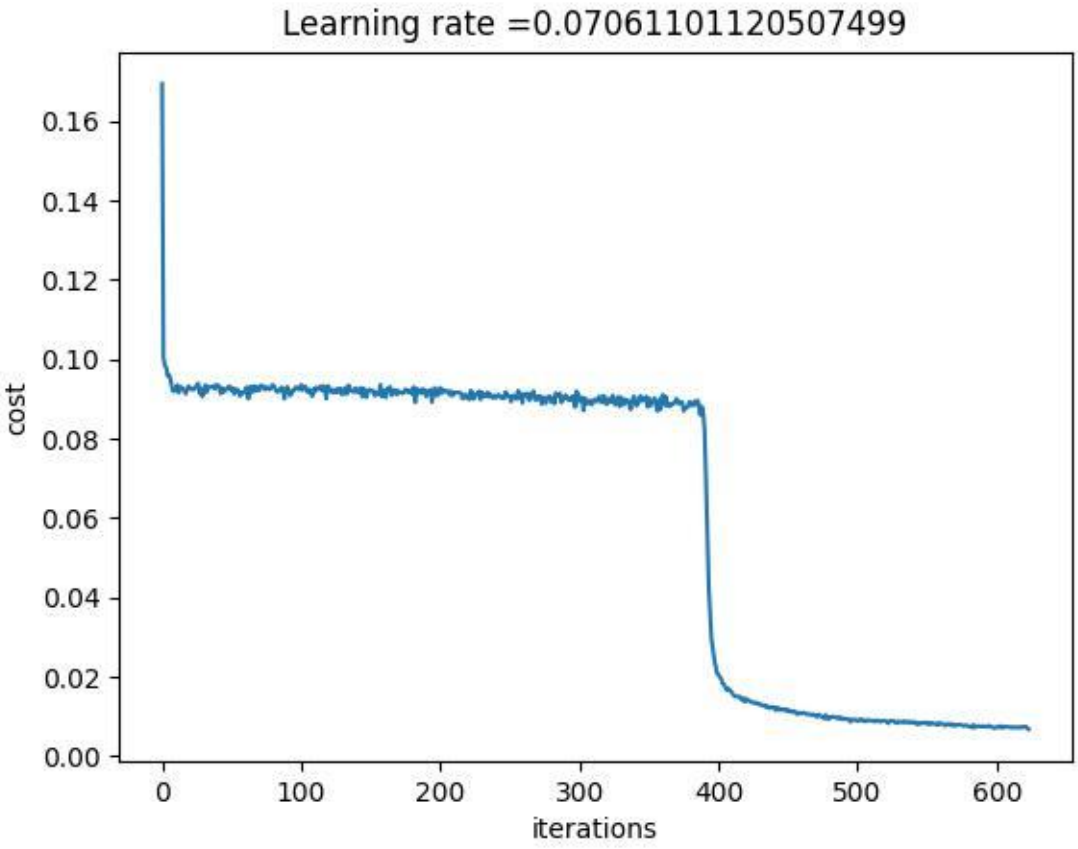


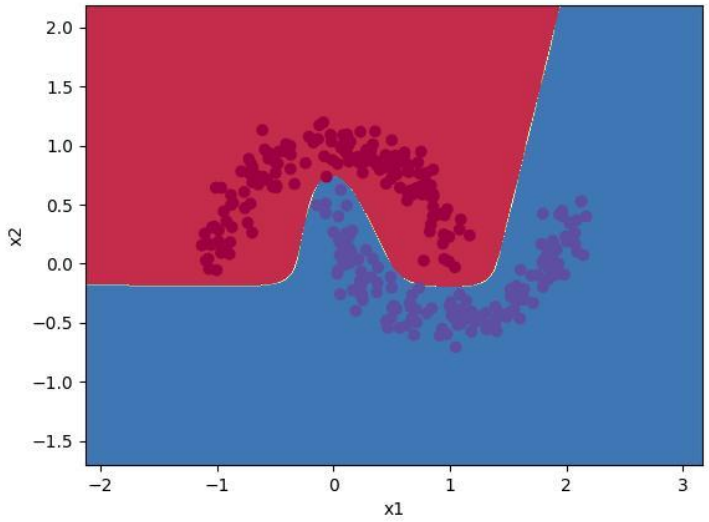
تابع فعالیت tanh ، online learning ، نرخ یادگیری ثابت





تابع فعالیت sigmoid ، online learning ، نرخ یادگیری متغیر



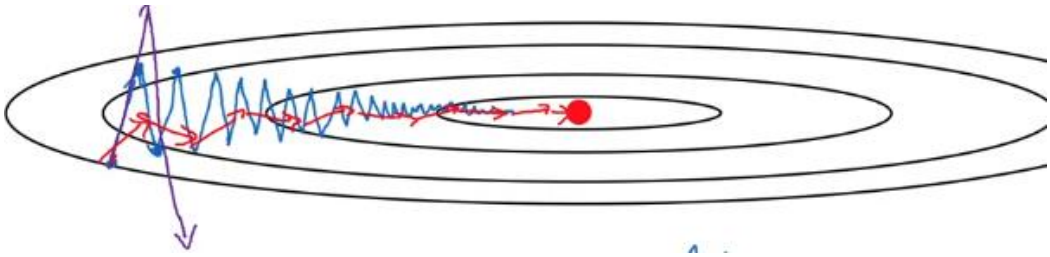


تابع فعالیت tanh ، online learning ، نرخ یادگیری متغیر

به خوبی مشخص است که تقریبا در تمامی روش ها توانسته ایم شبکه را به خوبی بر روی داده های آموزشی ، آموزش دهیم و نمودار نهایی نشان دهنده دقت خوب شبکه بر روی داده های تست دارد.

در یادگیری Online مقدار زمان بیشتری نسبت به batch زمان صرف میشود چون batch به صورت vectorized شده انجام میشود ولی Online هر epoch مقدار زمان بیشتری نیاز دارد.

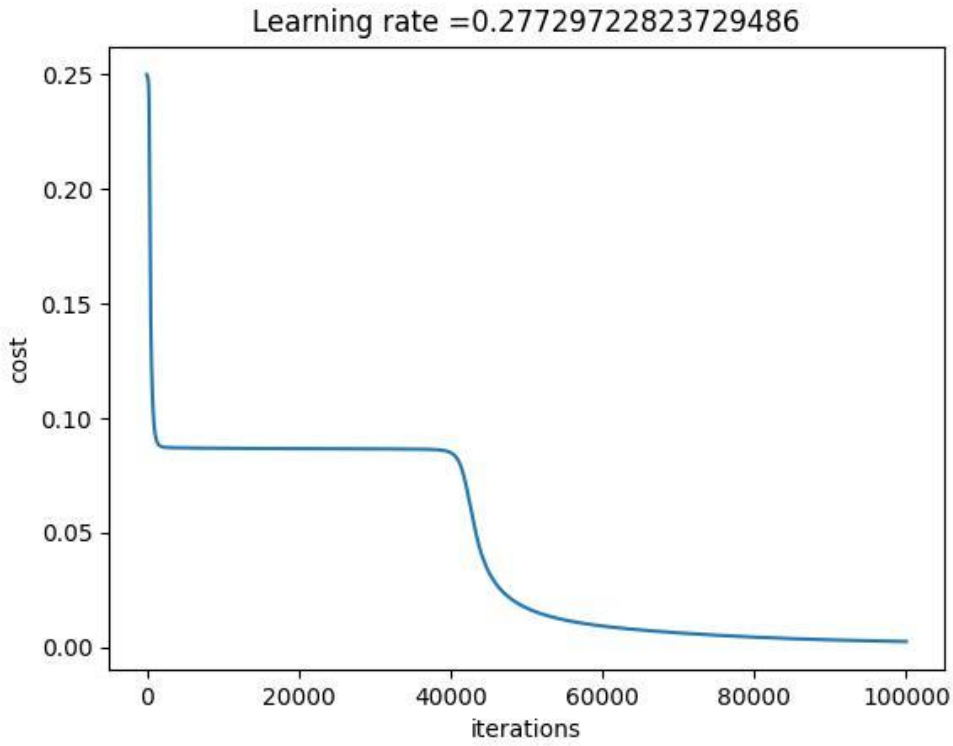
در روش ممنتوم هدف محاسبه یک میانگین وزن دار از گرادیان ها و استفاده از آن ها برای آپدیت کردن وزن های میباشد. در ممنتوم هدف آن است که اگر هدف در مرکز کانتور ها باشد ، آموزش در جهت افقی بیشتر و در جهت عمودی کندتر باشد به نحوی که برخلاف تنزل گرادیان معمول oscilliate زیادی نداشته باشد.

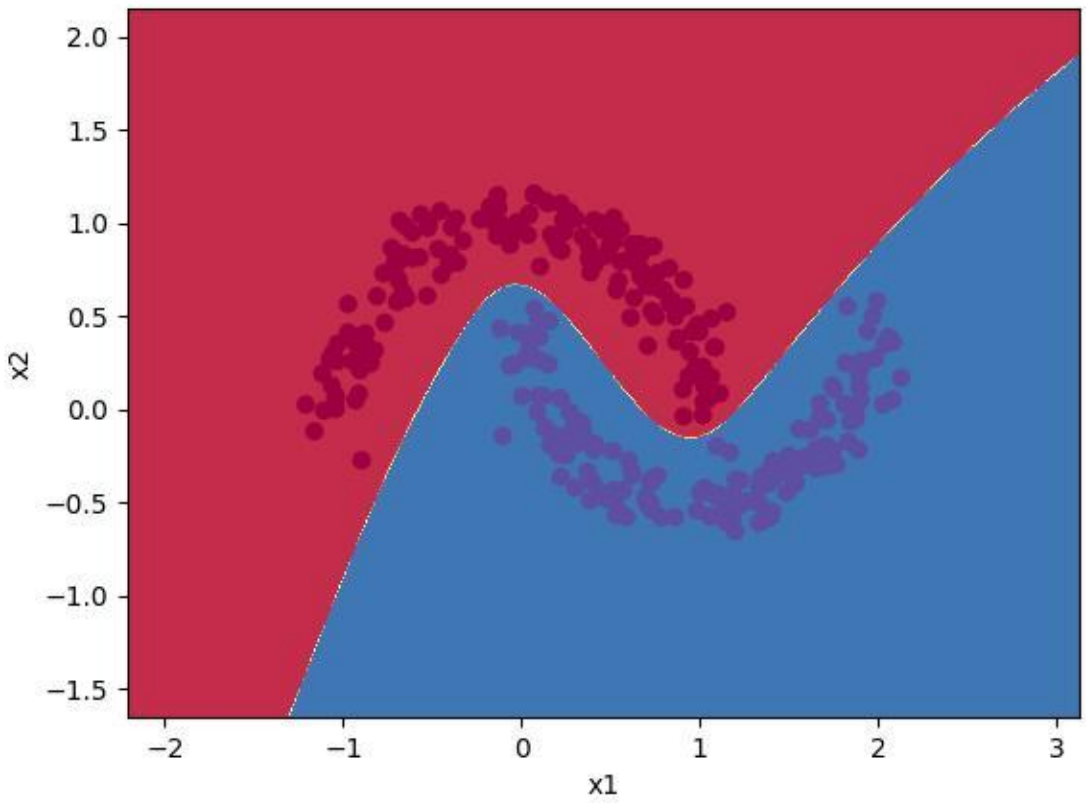


خطوط آبی و بنفش تنزل گرادیان معمول با نرخ یادگیری را نشان میدهند و خط قرمز استفاده از ممنتوم را به خوبی نمایش میدهد که به صورت یک میانگین وزن دار به سرعت به هدف نزدیک میشود.

بهترین شبکه به دست آمده شبکه با تابع فعالیت سیگموید با نرخ یادگیری متغیر بود .

پس از استفاده از روش ممنتوم ، همانطور که در شکل زیر مشخص است سریع تر و در epoch های کمتری به local optima بعدی همگرا شد.


Learning rate =0.27729722823729486
0.25
0.20
0.15
0.10
0.05
0.00
cost
0
20000
40000
60000
80000
100000
iterations


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
x2
−2
−1
0
1
2
3
x1

۲) Adam یا adaptive moment estimation یکی از optimization algorithm هایی است که به خوبی در بازه زیادی از برنامه ها قابل استفاده است . این الگوریتم به زبان ساده به صورت همزمان RMSprop و momentum را در کنار هم قرار داده و از آن ها استفاده میکند. Adam توانسته سرعت روش ممنتوم را با قدرت RMSprop که انطباق گرادیان ها در جهت های مختلف است ترکیب کند.

این الگوریتم در هنگامی که نرخ های یادگیری پارامتر ها منطبق میکند علاوه بر استفاده از میانگین گشتاور اول ما از میانگین گشتاور دوم گرادیان ها هم استفاده میکند. به این معنی که میانگین حرکت نمایی گرادیان و مربع گرادیان را محاسبه میکند. این کار سبب میشود که نرخ کاهش گرادیان به نحوی کنترل شود که کمترین حرکت زیگزاگی oscillation در هنگام رسیدن به گلوبال مینیمم داشته و در همین حال بتوان گام های به اندازه کافی بزرگ برای عبور از مینیمم های محلی داشت .

$V_{dw} = beta1 * V_{dw} + (1-beta1)dw$ $S_{dw} = beta2 * S_{dw} + (1-beta2)\ dw**2$

$V_{db} = beta1 * V_{db} + (1-beta1)db$ $S_{db} = beta2 * S_{db} + (1-beta2)db**2$

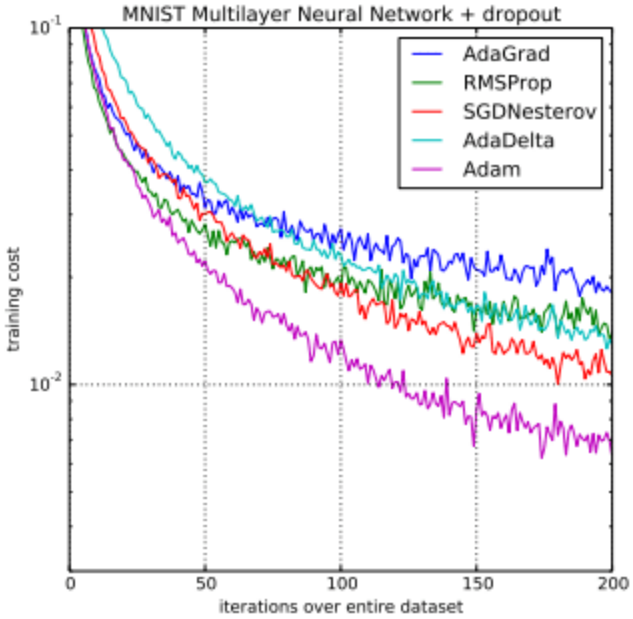
$V_{dw\_corrected} = V_{dw} / (1-beta1**t)$ $S_{dw\_corrected} = S_{dw} / (1-beta2**t)$

$V_{db\_corrected} = V_{db} / (1-beta1**t)$ $S_{db\_corrected} = S_{db} + (1-beta2\ **t)$

$W := W - alpha\ (V_{dw\_corrected})/sqrt\ (S_{dw\_corrected})+epsilon$

$b := b - alpha\ (V_{db\_corrected})/sqrt\ (S_{db\_corrected})+epsilon$

Beta1 is the decay rate for the first moment, sum of gradient (aka momentum), commonly set at 0.9. Beta 2 is the decay rate for the second moment, sum of gradient squared, and it is commonly set at 0.999.



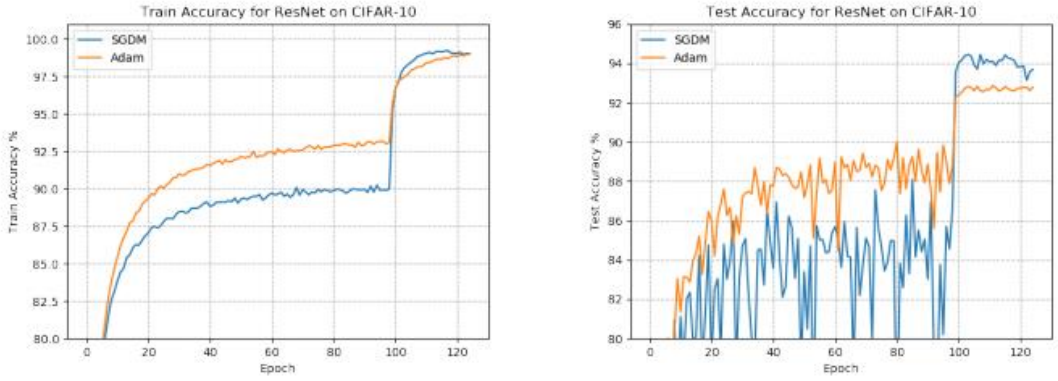
در تصویر بالا عملکرد Adam بر روی دیتاست MNIST مشخص است که بهتر از بقیه optimizer ها مثل RMSprop استفاده میکند.

دلیل برتری Adam نسبت به تنزل گرادیان و ممنتوم از این جهت است که سرعت بالاتری در همگرا شدن دارد و همچنین از نقاط قوت روش ممنتوم نیز استفاده میکند که یکی از علت های سریع تر بودن Adam است .

Adam به خوبی مناسب دیتاست هایی با حجم بالا میباشد و همچنین از لحاظ محاسباتی نیز بسیار به صرفه است .

اگرچه Adam به سرعت همگرا میشود ولی میتواند به یک راه حل بهینه همگرا نشود و عملکرد Gradient descent در همگرایی بهتر از Adam باشد زیرا Adam نمیتواند در Generalization مانند SGD عمل کند.

نرخ یادگیری شده توسط Adam در بعضی از شرایط میتواند خیلی کوچک باشد که سبب میشود که نتواند مسیر مناسب برای همگرا شدن پیدا کند. در بعضی از شرایط Adam نرخ یادگیری را بسیار بزرگ میکند که برای یادگیری مناسب نیست.



(a) train (b) test

https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning

https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-c١٠٢٣٢٥b٨٩٨momentum-adagrad-rmsprop-adam-f

VCZ/adam-optimization-٩https://www.coursera.org/learn/deep-neural-network/lecture/walgorithm

/