

# Week 2 Optimisation for Machine Learning

Neimhin Robinson Gunning, 16321701

January 31, 2024

## 1 Part (a)

### 1.1 Part (a) (i)

To represent the expression  $x^4$  using sympy we first create a symbol object for  $x$  (line 1 in Listing 3), and then raise it to the fourth power using standard python arithmetic notation. The `**` operation is overloaded such that `x**4` yields an object representing the expression  $x^4$ . If we assume  $y(x) = x^4$  then the derivative  $dy/dx$  is found with `(x**4).diff()`. Running the code we find this derivative to be  $4x^3$ .

Listing 1: Source code to find the derivative of  $x^4$  using sympy. The result is  $4x^3$ .

```
src/ai.py Thu Jan 11 09:02:38 2024 1
1: import sympy as sp
2: x = sp.symbols('x')
3: y = x**4
4: print(y.diff()) # 4*x**3
5:
```

### 1.2 Part (a) (ii)

Listing 2: Ellided source code to estimate the derivative of  $x^4$  using the finite difference method. The function `diff_with_pert` can be used to estimate the derivative of an arbitrary function of  $x$ . The full script is in the appendix and named `src/a11.py`.

```
src/finite_diff_eg.py Wed Jan 31 14:28:44 2024 1
1: import sympy as sp
2: # finite difference
3: def diff_with_pert(f, xval, pert=0.01):
4:     global x
5:     delta_x = pert
6:     return (f.subs(x, xval + delta_x) - f.subs(x, xval)) / (delta_x)
7: x = sp.symbols('x')
8: y = x**4
9: dydx = y.diff()
10: analytic_ys = [dydx.subs(x, i) for i in my_range()]
11: # ...
12: for pert in np.array([0.01, 0.1, 0.15]):
13:     dydx_finite = diff_with_pert(y, x, pert=pert)
14:     # ...
15:
```

### 1.3 Part (a) (iii)

The accuracy of the finite difference approximation of the derivative is greater for smaller values of  $\delta$ . The reason for this is related to the definition of the derivative. The derivative is defined as

$$\lim_{\delta \rightarrow 0} \frac{f(x + \delta) - f(x)}{x + \delta - x}$$

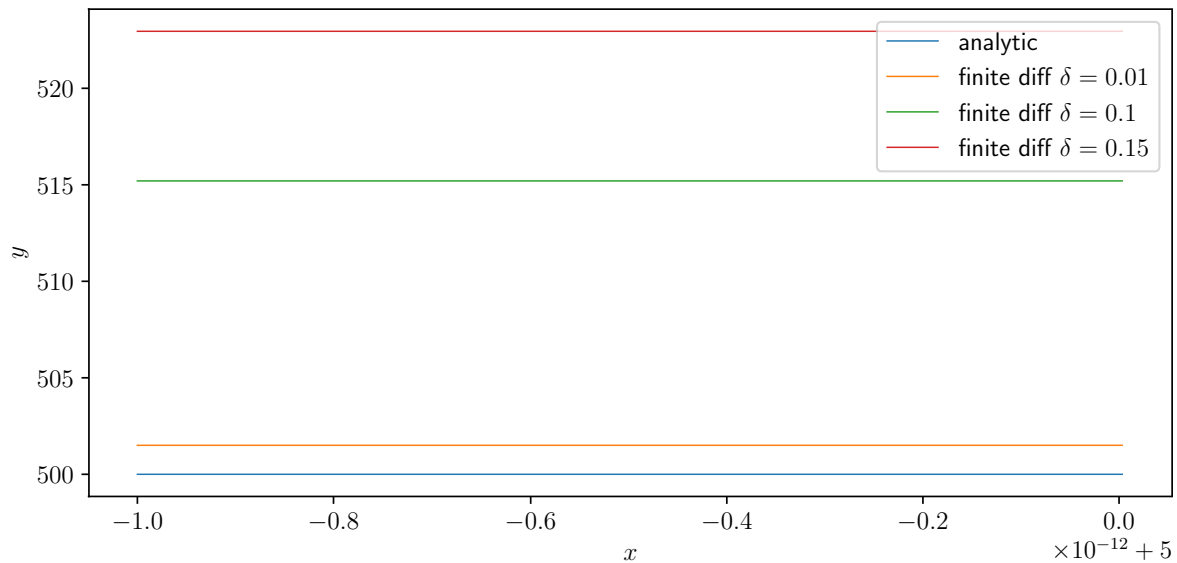


Figure 1: A comparison of the analytic derivation of  $dy/dx = 4x^3$  and some approximations of  $dy/dx$  using the finite difference method. The finite difference approximation was  $\hat{\frac{dy}{dx}}(x) = \frac{(x+\delta)^4 - x^4}{\delta}$ .

, i.e. the derivative is the function approached by this expression as  $\delta$  approaches 0. As we decrease  $\delta$  and apply the finite difference method we are approaching the true derivative.

## 2 (b)

### 2.1 (b) (i)

Listing 3 shows ellided source code for a python class that can be used to run gradient descent. Ellided functions allow for specification of the function to be optimized, the convergence condition, the max number of iterations etc. Line 13 calculates the gradient at the current value, and line 14 updates the value based on the gradient and step size. The convergence condition is a function of  $x_n$  and  $x_{n-1}$ . Examples of this class in use are in the appendix, e.g. in `src/bi.py`, in which the gradient function is computed with `sympy`.

Listing 3: Ellided source code a python class that implements gradient descent with a fixed step size.

```
src/gradient_descent_listing.py    Wed Jan 31 15:38:46 2024    1
1: class GradientDescent():
2:     # ...
3:     def iterate(self):
4:         import math
5:         x_value = self._start
6:         old_x_value = None
7:         iteration = 0
8:         while True:
9:             yield [iteration, float(x_value), float(self._function(x_value))]
10:            iteration += 1
11:            if self._max_iter > 0 and iteration > self._max_iter:
12:                break
13:            grad_value = self._gradient(x_value)
14:            x_value -= self._step_size * grad_value # Update step
15:            if old_x_value is not None and self._converged(x_value, old_x_value):
16:                yield [iteration, float(x_value), float(self._function(old_x_value))]
17:                print("converged")
18:                break
19:            old_x_value = x_value
```

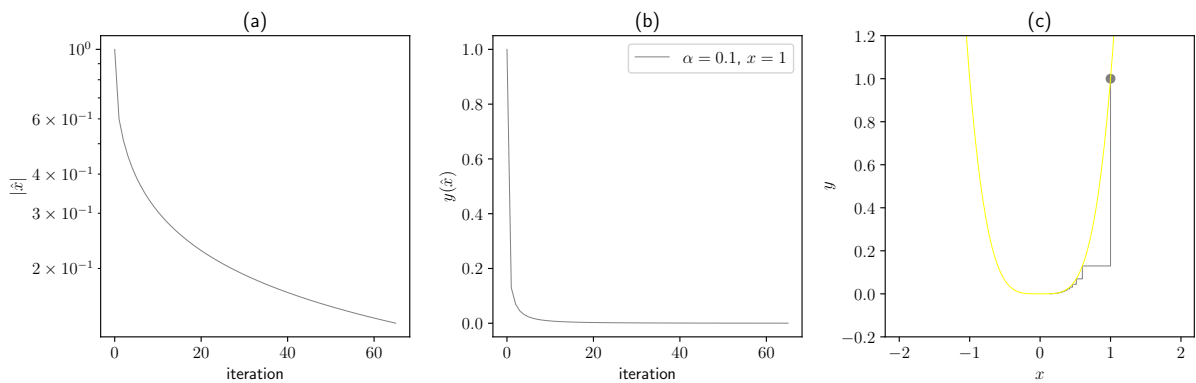


Figure 2: Running gradient descent on  $x^4$  with initial value  $x_0 = 1$  and step size  $\alpha = 0.1$ . The yellow line in subplot (c) is  $x^4$ .

## 2.2 (b) (ii)

In Figure 2 is a plot of gradient descent on  $x^4$  with  $x_0 = 1$  and  $\alpha = 0.1$ . We see that updates to  $|\hat{x}|$  and  $y(\hat{x})$  are initially large but quickly become tiny. Because  $x^4$  is very flat around the minimum (see subplot (c)) we have a very small gradient, and thus very small step sizes, so the algorithm updates extremely slowly towards  $\hat{x} = 0$ , but cannot reach it in a finite number of iterations.

In Figure 3 several runs of gradient descent on  $x^4$  are visualized. Subplot (a) shows  $|\hat{x}|$  for each iteration, which is the distance of the estimate  $\hat{x}$  from the optimal value,  $x = 0$ . With initial value  $x = 1$  and step size  $\alpha = 0.1$  we see that the steps are initially large, i.e. moving quickly to the optimum, but on successive iterations the steps slow down dramatically. In subplot (b) we see the value of  $y(\hat{x})$  on each iteration.

The convergence criterion for gradient descent adopted here is when  $|\alpha\Delta x| < 0.001$  or  $x$  is  $\pm\infty$ <sup>1</sup>. For the function  $x^4$  we can see from Table 1 that the final guess, i.e. the estimate of  $\arg \min x^4$  is most accurate and converges fastest with larger values of  $\alpha$ , **unless**  $\alpha$  is too large and the iterations of gradient descent degenerate, i.e. subsequent iterations give worse or unchanged estimates. The reason for this is that the gradient of  $x^4$  is tiny as  $x$  approaches the minimum. So, a smaller value of  $\alpha$  means the convergence condition,  $|\alpha\Delta x| < 0.001$ , can be met with a higher value of  $\Delta x$ , but in the case of  $x^4$ ,  $\Delta x$  is higher when  $x$  is further from the optimal value 0. This is why the final guess is worse for smaller values of  $\alpha$ , but why does a smaller  $\alpha$  lead to slower convergence time?

Since  $\alpha$  is constant for a single run of gradient descent, the convergence condition depends on  $\Delta x$ , namely we need  $|\Delta x| < 0.001/\alpha$ . A larger  $\alpha$  entails a larger step, but there are two factors at play here:

1. A larger step size may entail a smaller  $\Delta x$  on the subsequent iteration.
2. A larger step size moves us away from meeting the convergence condition.

It turns out for  $x^4$  that the first of these factors is more important, in cases where gradient descent is converging. Thus larger  $\alpha$  results in faster convergence (for  $x^4$ ).

<sup>1</sup>Meaning the floating point values `math.inf` or `-math.inf`

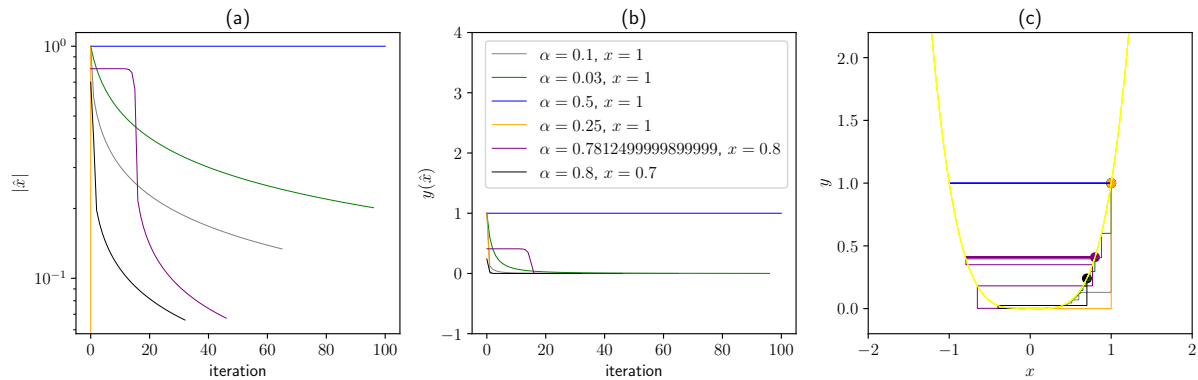


Figure 3: Various runs of gradient descent on  $x^4$ . One run, where  $\alpha = 0.5$  and  $x_0 = 1$ , is stuck in a 2-step loop, flipping  $\hat{x}$  back and forth between 1 and -1. The yellow line in subplot (c) is  $x^4$ .

Table 1: Various runs of gradient descent on  $x^4$  with different outcomes. The maximum number of iterations was set to 100, and the convergence threshold to 0.001. Some of the  $\alpha$  values were purposefully selected to lie near the margin of non-convergence, which can be calculated with  $\alpha_m = \frac{2x_0}{4x_0^3}$ . So rows 8 and 9 use  $\alpha_m \pm \epsilon$ .

	$\alpha$	start	convergence time	final guess
0	0.01	1.0	100	0.3323366923926806
1	0.02	1.0	100	0.24119415115009837
2	0.03	1.0	97	0.20146846304616808
3	0.04	1.0	89	0.18252923619780276
4	0.05	1.0	83	0.1691984331910481
5	0.1	1.0	66	0.13379629115760108
6	0.4999999999999999	1.0	55	-0.07827643923095963
7	0.5	1.0	100	-1.0
8	0.7812500000000009	0.8	29	inf
9	0.7812499999999989	0.8	24	0.02474194758833489
10	0.8	0.7	33	-0.06609857197723801

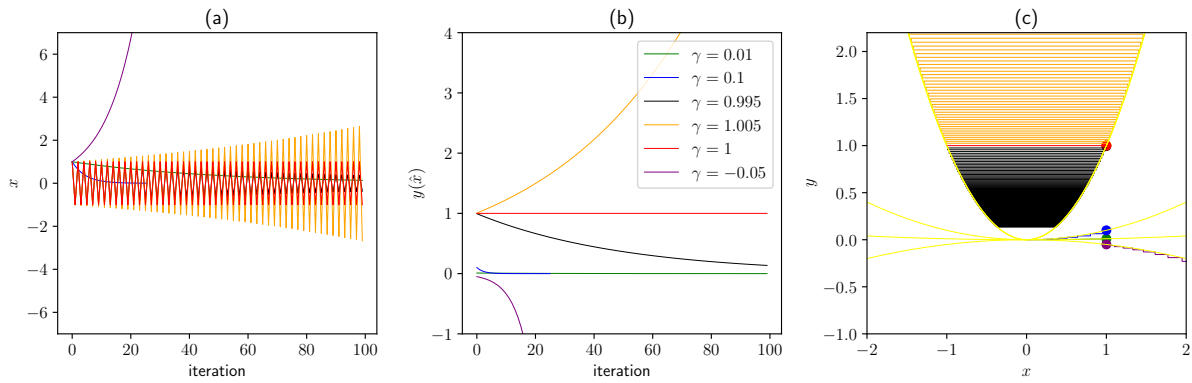


Figure 4: Various runs of gradient descent on  $\gamma x^2$ . Subplot (a) shows the estimate of  $x$  for each iteration. The yellow lines in subplot (c) are plots of the various  $\gamma x^2$  functions.

### 3 (c)

#### 3.1 (c) (i) Optimising $\gamma x^2$

Firstly, note that if  $\gamma$  is negative, then  $\gamma x^2$  and  $\gamma|x|$  have no minimum, and thus gradient descent will behave weirdly. For instance, in my implementation, with enough iterations it will actually finish with `math.inf`, which is reasonable. If  $\gamma = 0$  then gradient descent terminates immediately, and any value for  $x$  minimizes  $\gamma x^2 = 0x^2$ . Increasing  $\gamma$  increases the magnitude of the gradient, resulting in larger steps, but the convergence time and accuracy will then depend on the initial value.

Let  $d_0 = |x_0 - \arg \min_x \gamma x^2|$  be the initial distance to the optimal value of  $x$ . Let's assume  $x_0$  is 'far' from

$$\arg \min_x \gamma x^2$$

, i.e.  $d_0$  is large, meaning gradient descent will at first take several steps in the same direction. Increasing  $\gamma$  to  $\gamma'$ , i.e.  $\gamma' > \gamma$ , results in a larger gradient  $2\gamma'x'_0 > 2\gamma x_0$  meaning the first step is larger. If after the first iteration the distance has improved,  $d_1 < d_0$  then gradient descent will converge, if not it will diverge. Let's assume we are in a converging condition. While the second gradient  $2\gamma'x'_1$  is not necessarily larger than  $2\gamma x_1$ , the updated  $x'_1$  is now closer to the minimum than  $x_1$ . The result of these dynamics is that with a larger  $\gamma'$  there is a fewer number iterations until the first 'overshoot'.

For certain values this means that convergence time is smaller for the larger  $\gamma'$  than for  $\gamma$ .

#### 3.2 (c) (ii) Optimising $\gamma|x|$