STAT154 HW7
Neural Network and Back Propagation

Seth Metcalf

1 May 2024

1 Theoretical Exercises

Q1 (Forward and Back propagation algorithm)

- Calculate the derivatives $\partial_{w_i} E((x,y); w_1, w_2, w_3)$ using the chain rule. (You can use the intermediate quantities in the final formula, and the final result can contain expressions like $\partial_{\hat{y}}\ell(...)$ and $\sigma'(...)$.)
- Given specific numeric values of (x, y, w_1, w_2, w_3), describe the back-propagation algorithm to get the specific numeric values of $\partial_{w_i} E((x,y); w_1, w_2, w_3)$. (You may find it helpful to first use the forward-propagation algorithm to get the intermediate quantities.)
- Take $\sigma(x) = \max\{x, 0\}$, $\ell(y, \hat{y}) = (y - \hat{y})^2$, $x = 1$, $y = 10$, $w_3 = 3$, $w_2 = 2$, $w_1 = 2$. (Note that σ is not actually differentiable at 0, but you can simply assume the expression $\sigma'(x) = 1\{x > 0\}$ for all $x \in \mathbb{R}$.) Please calculate the numeric values of $\partial_{w_i} E((x,y); w_1, w_2, w_3)$ using the back-propagation algorithm. (This will not be too complicated if you use the correct algorithm.)

$$\frac{\partial E}{\partial w_3} = \frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_3} = \frac{\partial l}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_3} = \frac{\partial l}{\partial \hat{y}} \cdot x_2 \cdot x_2 = \partial_{\hat{y}} l \cdot x_2^2$$

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2} = \frac{\partial l}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_3} \cdot \frac{\partial z_3}{\partial x_2} \cdot \frac{\partial x_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2} = \frac{\partial l}{\partial \hat{y}} \cdot w_3 \cdot \sigma' z_2 \cdot x_1 \cdot x_1 \cdot x_1 = \partial_{\hat{y}} l \cdot w_3 \cdot \sigma'(x_2) \cdot x_1$$

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_1} = \frac{\partial l}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_3} \cdot \frac{\partial z_3}{\partial x_2} \cdot \frac{\partial x_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial x_1} \cdot \frac{\partial x_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1} = \partial_{\hat{y}} l \cdot w_3 \cdot \sigma'(z_2) \cdot w_2 \cdot \sigma'(z_1) \cdot x$$

Back-propagation efficiently computes the gradient of the loss function by recursively applying the chain rule.

First use forward propagation in order to get the intermediate qualities:

- $z_1 = w_1 x = 2 \times 1 = 2.$
- $x_1 = \sigma(z_1) = \max\{z_1, 0\} = \max\{2, 0\} = 2.$
- $z_2 = w_2 x_1 = 2 \times 2 = 4.$
- $x_2 = \sigma(z_2) = \max\{z_2, 0\} = \max\{4, 0\} = 4.$
- $z_3 = w_3 x_2 = 3 \times 4 = 12.$
- $\hat{y} = x_3 = z_3 = 12$

Then, use back propagation, start by computing the derivative of the loss function:

$$\frac{\partial E}{\partial \hat{y}} = -2(y - \hat{y}) = -2(10 - 12) = 4$$

Once deriving the loss function, we can recursively apply the chain rule (w/ respect to weights):

$$\frac{\partial E}{\partial w_3} = \frac{\partial E}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_3} \times \frac{\partial z_3}{\partial w_3} = 16$$

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_3} \times \frac{\partial z_3}{\partial x_2} \times \frac{\partial x_2}{\partial z_2} \times \frac{\partial z_2}{\partial w_2} = 24$$

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_3} \times \frac{\partial z_3}{\partial x_2} \times \frac{\partial z_2}{\partial x_1} \times \frac{\partial x_1}{\partial z_1} \times \frac{\partial z_1}{\partial w_1} = 24$$

Q2 (Matrix Calculus for Neural Networks)

In order to express $\nabla_W \phi(x; W)$ in terms of $(v, \nabla f, \nabla g)$, first we need to find the derivative of ϕ:

$$\frac{\partial \phi}{\partial W_{ij}}(x; W) = \frac{\partial}{\partial W_{ij}}(v^T f(Wg(x)))$$

Applying chain rule:

$$\Rightarrow \sum_{k=1}^{d_3} \sum_{l=1}^{d_4} \frac{\partial}{\partial W_{ij}}(v_l f_l(Wg(x))) \Rightarrow \sum_{k=1}^{d_3} \sum_{l=1}^{d_4} v_l \frac{\partial f_l}{\partial z_k}(Wg(x)) \cdot \frac{\partial}{\partial W_{ij}}(W_{kl} g_l(x))$$

Now since $\frac{\partial}{\partial W_{ij}}(W_{kl} g_l(x))$, we get:

$$\Rightarrow \sum_{l=1}^{d_4} v_l \frac{\partial f_l}{\partial z_i}(Wg(x)) \cdot g_j(x) = (\nabla f(Wg(x)))^T v \cdot g(x)$$

Therefore:

$$\nabla_W \phi(x; W) = (\nabla f(Wg(x)))^T v \cdot g(x)$$

In order to express $\nabla_{W_j} E$, we also need to compute the derivative:

$$\frac{\partial E}{\partial W_j} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial H_2} \frac{\partial H_2}{\partial W_j}$$

Computing each of the partials:

$$\frac{\partial E}{\partial \hat{y}} = -2(y - \hat{y})$$

$$\frac{\partial \hat{y}}{\partial H_2} = W_3 \text{diag}(\sigma'(H_2))$$

$$\frac{\partial H_2}{\partial W_1} = (W_2^T \text{diag}(\sigma'(H_1)) \cdot x^T$$

$$\frac{\partial H_2}{\partial W_2} = \sigma'(H_1) \cdot x^T$$

Therefore, to express $\nabla_{W_j} E$:

$$\nabla_{w_3} E = -2(y - \hat{y}) \cdot W_3^T \text{diag}(\sigma'(H_2))$$

$$\nabla_{w_2} E = -2(y - \hat{y}) \cdot W_3^T \text{diag}(\sigma'(H_2)) \cdot \sigma'(H_1) \cdot x^T$$

$$\nabla_{w_1} E = -2(y - \hat{y}) \cdot W_3^T \text{diag}(\sigma'(H_2)) \cdot (W_2^T \text{diag}(\sigma'(H_1)) \cdot x^T$$

2 Computational Exercises

Q1 (Finite difference method 1)

- $\sigma(x) = \max\{x, 0\}$
- $\hat{y}(x; w_1, w_2, w_3) = w_3\sigma(w_2\sigma(w_1 x))$
- $\Delta = 10^{-4}$

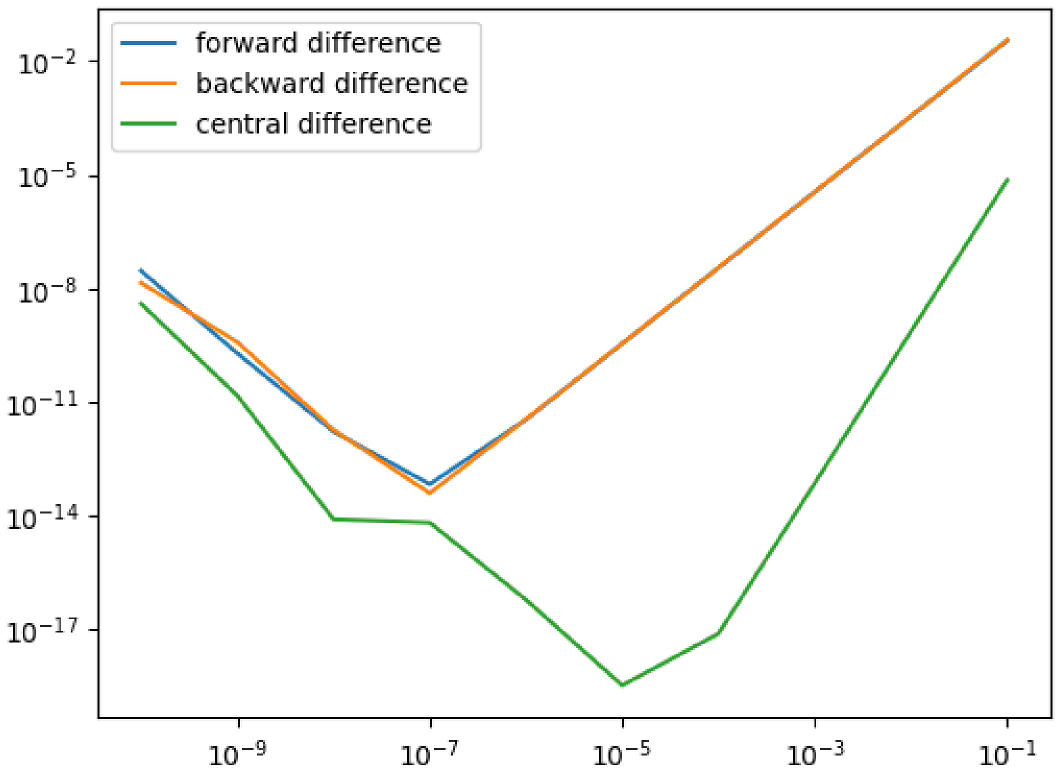
```
import numpy as np

# sigmoid function
def sigmoid(x):
    return np.maximum(x, 0.)

# function of \hat y
def hat_y(w):
    return (w[2]*sigmoid(w[1]*sigmoid(w[0])) - 10)**2


e1 = np.array([1, 0, 0])
e2 = np.array([0, 1, 0])
e3 = np.array([0, 0, 1])
w = np.array([2., 2., 3.])
delta = 1e-4

fwd_diff = ((np.array([hat_y(w + e1*delta), hat_y(w + e2*delta), hat_y(w + e3*delta)])) - hat_y(w))/del
bck_diff = (hat_y(w) - (np.array([hat_y(w - e1*delta), hat_y(w - e2*delta), hat_y(w - e3*delta)])))/del
cen_diff = (np.array([hat_y(w + e1*delta), hat_y(w + e2*delta), hat_y(w + e3*delta)]) - np.array([hat_y
print('forward diff:', fwd_diff, '\nbackward diff:', bck_diff, '\ncentral diff:', cen_diff)
```

forward diff: [24.0036 24.0036 16.0016] backward diff: [23.9964 23.9964 15.9984] central diff: [24. 24. 16.]

Q2 (Finite difference method 2)

```
import numpy as np
import mygrad as mg
import matplotlib.pyplot as plt


def compute_gradient(input_data, function):
    tensor_input = mg.tensor(input_data)
    function_output = function(tensor_input)
    function_output.backward()
    return tensor_input.grad

def sigmoid(x):
    return (1.)/(1 + np.exp(-x))

def hat_y(w):
    return (w[2]*sigmoid(w[1]*sigmoid(w[0])) - 10)**2
```

```
e1 = np.array([1, 0, 0])
e2 = np.array([0, 1, 0])
e3 = np.array([0, 0, 1])
w = np.array([2., 2., 3.])
grad = compute_gradient(w, hat_y)
deltas = np.logspace(-10, -1, num=10)

fwd_error = np.zeros_like(deltas)
bck_error = np.zeros_like(deltas)
cen_error = np.zeros_like(deltas)

for i in range(deltas.shape[0]):
    delta = deltas[i]

    fwd_diff = ((np.array([hat_y(w + e1*delta), hat_y(w + e2*delta), hat_y(w + e3*delta)])) - hat_y(w))/
    bck_diff = (hat_y(w) - (np.array([hat_y(w - e1*delta), hat_y(w - e2*delta), hat_y(w - e3*delta)])))/
    cen_diff = ((np.array([hat_y(w + e1*delta), hat_y(w + e2*delta), hat_y(w + e3*delta)])) - (np.array

    fwd_error[i] = np.linalg.norm(grad - fwd_diff)**2
    bck_error[i] = np.linalg.norm(grad - bck_diff)**2
    cen_error[i] = np.linalg.norm(grad - cen_diff)**2


plt.loglog(deltas, fwd_error, label='forward difference')
plt.loglog(deltas, bck_error, label='backward difference')
plt.loglog(deltas, cen_error, label='central difference')
plt.legend()
plt.show()
```



The plot is above:

We can see in the plot above that all of them decrease as the stepsize decreases (delta decreases) until a certain point in which it increases due to the rounding error.