0 Instructions

Homework is due Tuesday, March 18, 2024 at 23:59pm Central Time. Please refer to `https://courses.grainger.illinois.edu/cs446/sp2024/homework/hw/index.html` for course policy on homeworks and submission instructions.

1 Neural networks for simple functions

1.1

$$\omega_0 = (1, -1)^T \omega_1 = (1, -1)^T$$

1.2

$$\omega_0 = (m, -m)^T \omega_1 = (1, -1)^T b = (b, -b)^T$$

1.3

$$\omega_0 = 0, \omega_1 = 2b$$

1.4

$$\omega_0 = (3, 6, 3)^T b = (6, 0, -6)^T \omega_1 = (1, -1, 1)^T$$

1.5

No. ReLU makes negative part of the innner linear function to 0, and positive parts to remain the same. So, after applying the outter linear function, the output will also be a combination of a linear function and 0s.

1.6

No. Suppose x_0 satisfies $|w_1^T \sigma(w_0^T x_0 + b) - x_0^2| < \epsilon$, as x_0 goes to infinity, the x_0^2 increases at a second order rate, while the first term increases at a maximum first order rate. So, the difference will go to infinity.

1.7

Yes. From the differential element method, we konw that for any small ϵ, we can always find a large enough amount of line segments to approximate the range-limited quadratic function. So the problem is how to find a suitable set of w_0, w_1 and b to express the line segments.

We give a high-level construction routine for w_0, w_1 and b as follows.

Suppose the line segments are $l_0, l_1, \ldots, l_n$. The objective $w_1^T \sigma(w_0 x + b_0)$ can be viewed as a linear combination of $\sigma(w_0^0 x + b_0^0), \sigma(w_0^1 x + b_0^1), \ldots, \sigma(w_0^m x + b_0^m)$.

Step 1, we can choose $w_0^0 x + b$ to be the line containing the segment l_0. And for every $i \geq 1$ choose w_0^i equal to the differnce between the slope rate of segment $i+1$ and segment i.

Step 2, for every resulting w_0^i, if w_0^i is smaller than 0, we set $w_0^i = -w_0^i$.

Step 3, after getting all w_0^i, b_0^i is equal to the corresponding intersection when the line $w_0^i x + b_0^i = 0$.

The $w_1 = (-1, -1, \ldots, 1, 1)$, where -1 is corresponding to those w_0^i that changed signs at step 2.

2 Backpropagation through time (BPTT)

2.1

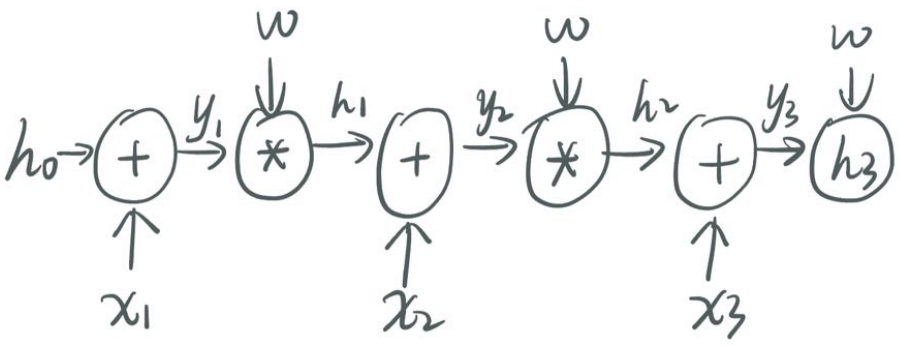

Figure 1: Illustration of 2.1

2.2

$$h_0 = 0 \quad y_1 = h_0 + x_1$$
$$h_1 = w^T(h_0 + x_1) \quad y_2 = w^T(h_0 + x_1) + x_2$$
$$h_2 = w^T(w^T(h_0 + x_1) + x_2) \quad y_3 = w^T(w^T(h_0 + x_1) + x_2) + x_3$$
$$h_3 = w^T(w^T(w^T(h_0 + x_1) + x_2) + x_3)$$

2.3

$$\begin{aligned}
\frac{\partial h_3}{\partial \omega} &= y_3 + \omega \frac{\partial y_3}{\partial \omega} \\
&= y_3 + \omega \frac{\partial h_2}{\partial \omega} \\
&= y_3 + \omega(y_2 + \omega \frac{\partial y_2}{\partial \omega}) \\
&= y_3 + \omega y_2 + \omega^2 \frac{\partial h_1}{\partial \omega} \\
&= y_3 + \omega y_2 + \omega^2(y_1 + \omega \frac{\partial y_1}{\partial \omega}) \\
&= y_3 + \omega y_2 + \omega^2 y_1
\end{aligned}$$

2.4

$$\begin{aligned}\frac{\partial f}{\partial h_1} &= \frac{\partial f}{\partial h_T}\frac{\partial f}{\partial h_{T-1}}\cdots\frac{\partial h_2}{\partial h_1} \\ &= 1\cdot\omega\sigma'(x_t+h_{t-1})\cdot\omega\sigma'(x_{t-1}+h_{t-2})\ldots\omega\sigma'(x_1)\end{aligned}$$

2.5

The RNN back propagation requires to compute the multiplication of gradients of all the previous time steps. When the sequence is long, it is more likely to have more large gradients, which will cause the explosion problem, or more small gradients, which will cause the vanishing problem.

3 Transformers

3.1

No. Since the denominator is the sum of all the nominators, so α can not be infinite. Since $e^{k^T q}$ is always positive, so α can not be zero.

3.2

In order to garantee $a_i \gg a_j$, we have to make $k_i^T q \gg k_j^T q$. In this case, $c = v_i$. Intuitively, in this case, the attention mechanism will only focus on the i-th feature.

3.3

We can choose $q = C(k_i + k_j)$, where C is a large constant.

3.4

$$\begin{aligned}
\alpha_i &= \frac{e^{-\frac{1}{2\sigma}||q-k_i||^2}}{\sum e^{-\frac{1}{2\sigma}||q-k_i||^2}} \\
&= \frac{1}{\sum e^{-\frac{1}{2\sigma}(||q-k_j||^2-||q-k_i||^2)}} \\
&= \frac{1}{\sum e^{-\frac{1}{2\sigma}(q^Tq-2q^Tk_j+k_j^Tk_j-q^Tq+2q^Tk_i-k_i^Tk_i)}} \\
&= \frac{1}{\sum e^{-\frac{1}{2\sigma}(2q^T(k_i-k_j)-||k_i||^2+||k_j||^2)}} \\
&= \frac{e^{\frac{1}{\sigma}q^Tk_i}}{\sum e^{-\frac{1}{\sigma}q^Tk_j}}
\end{aligned} \tag{1}$$

We can see the result is the same as the original dot product softmax similarity.

3.5

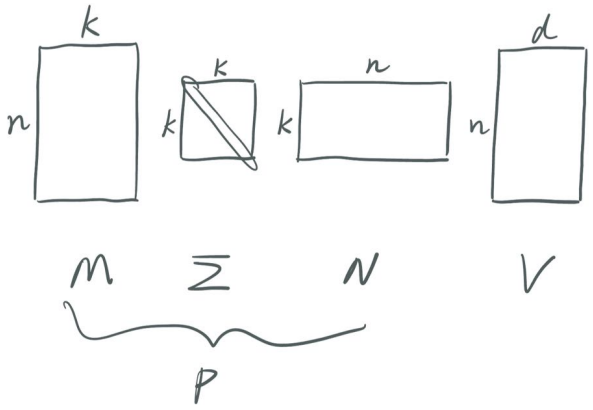

Figure 2: Illustration of the Attention Computing using SVD

As shown in 2, we first compute NV in $O(knd)$ time and set result as $X \in \mathbb{R}^{k\times d}$. Because the matrix Σ is a diagonal matrix, we can compute ΣX in $O(kd)$ time. Suppose the result is $Y \in \mathbb{R}^{k\times d}$. Finally, we compute MY in $O(knd)$ time. The total time complexity is $O(knd + kd + knd) = O(knd)$.

4 Resnet

4.2

Batch normalization will compute the average and variance of the same channel over the minibatch. If we use bias in Conv2D, every number of the output in the same channel will be added by a same constant. So, if followed by a batch normalization, the constant will be canceled out when the mean is subtracted. So the bias term won't have any effect.

5 Coding: Image overfitting

5.5

I found the experiments are hard to converge (especially for ReLU and tanh), so I added learning rate decay to all training processes.

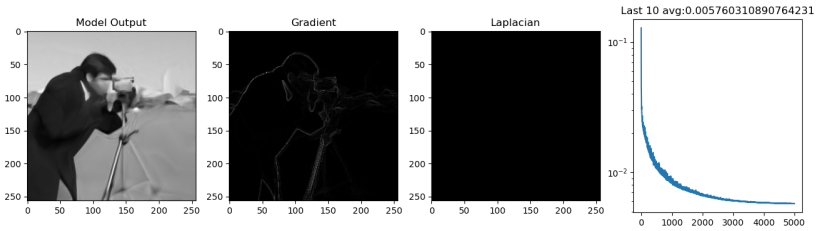

Figure 3: Activation: ReLU, Epochs: 5000, Batch Size: 1024, Learning Rate: 10^{-3}, with Learning Rate Decay

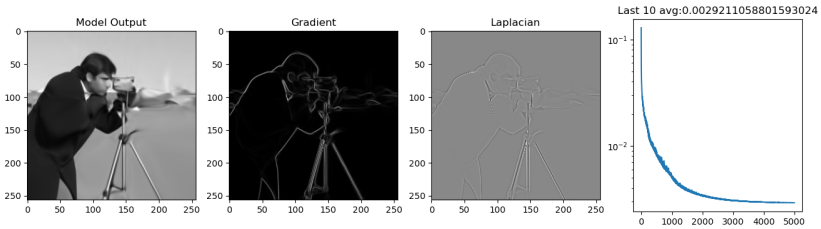

Figure 4: Activation: tanh, Epochs: 5000, Batch Size: 1024, Learning Rate: 10^{-3}, with Learning Rate Decay

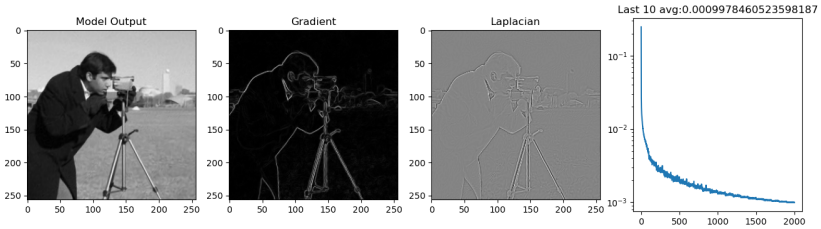

Figure 5: Activation: sin, Epochs: 2000, Batch Size: 1024, Learning Rate: 10^{-4}, with Learning Rate Decay

5.6

The Laplacian is computed by

$$\begin{aligned}
\Delta f &= \frac{\partial^2 f_n}{\partial x \partial y} + \frac{\partial^2 f_n}{\partial y \partial x} \\
f_n &= \sigma(w_n^T f_{n-1} + b_n) \\
f_{n-1} &= \sigma(w_{n-1}^T f_{n-2} + b_{n-1}) \\
&\cdots \\
f_1 &= \sigma(w_1^T f_0 + b_1) \\
f_0 &= \sigma(w_0^T \vec{x} + b_0)
\end{aligned}$$

The first order derivative is (for simplicity, we only show the derivative with respect to x)

$$\frac{\partial f}{\partial x} = \frac{\partial f_n}{\partial f_{n-1}} \frac{\partial f_{n-1}}{\partial f_{n-2}} \cdots \frac{\partial f_1}{\partial f_0} \frac{\partial f_0}{\partial x}$$

Because the activation function is ReLu, each term above is whether 0 or w_i. So, the first order derivative is just a combination of w_i or 0. To compute the Laplacian, we need to compute derivatives with respect to y. We can see the result of the first order derivative is independent of y, so the Laplacian is always 0, which results in the all black image.

5.7

We can decrease the learning rate, which is called learning rate decay.

5.8

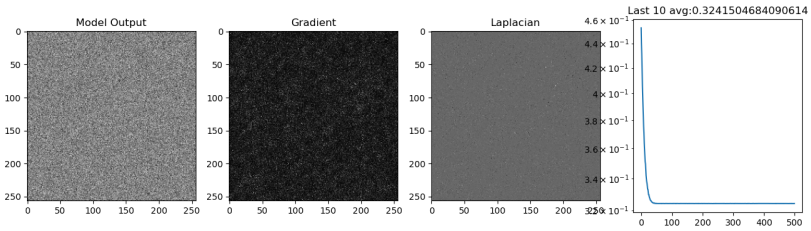

Figure 6: Activation: sin, Epochs: 500, Batch Size: 1024, Learning Rate: 10^{-4}, without Learning Rate Decay, without Initialization

As shown in 6, when training without initialization, the model seems to get stuck in a local minimum. The loss is indeed decreasing, but the output image are just random noise.

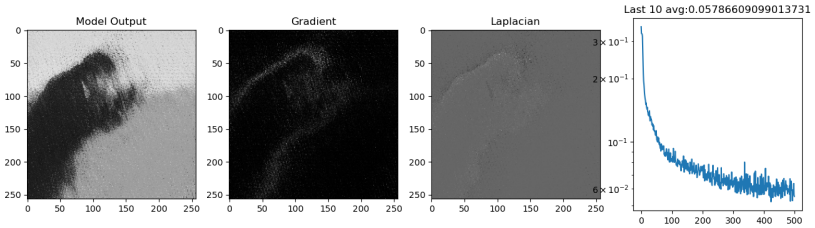

Figure 7: Activation: sin, Epochs: 500, Batch Size: 1024, Learning Rate: 10^{-4}, without Learning Rate Decay, without Normalization

When training without normalization (input coordinates are original integer tuples), the output shows a large scale of variance. I think the reason is that the inputs are too discrete.