Mathematical Foundations of Machine Learning
Assignment

David Felipe Alvear Goyes

1. Step length for gradient descent.
When the loss function f has an L-smooth gradient with a known Lipschitz constant L, the learning rate α can be chosen to ensure a quantifiable reduction in loss, which we derive in this exercise.

- combine a Taylor expansion with the L-smoothness of the gradient to derive the following:

$$f(x+\alpha d) \leq f(x) + \alpha \nabla f(x)^T d + \alpha^2 \frac{L}{2} \|d\|^2$$

- show that if we move in the negative gradient direction $d = -\nabla f(x^k)$ the step length $\alpha = 1/L$ minimizes the expression on the right
- show that this learning rate guarantees a reduction in the loss that is at least $\frac{1}{2L} \|\nabla f(x^k)\|^2$

Solution

→ $f(x)$: loss function

→ Taylor Expansion: $f(x+p) = f(x) + \nabla f(x)^T p + p^T \nabla^2 f(x+\gamma p) p$

→ Lipschitz Constant L: $\| \nabla f(x) - \nabla f(y) \| \leq L \|x - y\|$

$$f(x+\alpha d) = f(x) + \alpha \nabla f(x) d + \frac{1}{2} \alpha^2 d^T \nabla^2 f(x+\alpha d) d$$

$$\| \nabla f(x+\alpha d) - \nabla f(x) \| \leq L \| \cancel{x} + \alpha d - \cancel{x} \| = L \| \alpha d \|$$

$$\| \nabla f(x+\alpha d) - \nabla f(x) \| \leq L \alpha \|d\|$$

Using the mean value theorem → $\nabla f(x+\alpha d) = \nabla f(x) + \alpha \nabla^2 f(x') d$

$\nabla f(x+\alpha d) - \nabla f(x) = \alpha \nabla^2 f(x') d$

where $x' \in \{x, x+\alpha d\}$

→ $\| \alpha \nabla^2 f(x+\alpha d) d \| \leq L \alpha \|d\|$

→ $\alpha^2 \nabla^2 f(x+\alpha d) d \leq L \alpha^2 \|d\|$

$$\boxed{f(x+\alpha d) \leq f(x) + \alpha \nabla f^T(x) d + \frac{L}{2} \alpha^2 \|d\|}$$

Show $\rightarrow$ if $d = -\nabla f(x^k)$, the expression $\alpha = 1/L$ minimize $f(x+\alpha d)$

$$f(x^k+\alpha d) \leq f(x^k) + \alpha \nabla f^T(x^k)\left(-\nabla f(x^k)\right) + \frac{L}{2}\alpha^2 \|\nabla f(x^k)\|$$

$$\leq f(x^k) - \alpha \nabla f^T(x^k) \nabla f(x^k) + \frac{L}{2}\alpha^2 \|\nabla f(x^k)\|$$

$\rightarrow$ to minimize $f(x^k+\alpha d) \rightarrow f(x^k+\alpha d) \leq f(x^k)$

then $$\frac{L}{2}\alpha^2 \|\nabla f(x^k)\| - \alpha \nabla f^T(x^k) \nabla f(x^k) = 0$$

$\rightarrow$ Find α that minimize the expression $\frac{\partial f()}{\partial \alpha}$

$$\alpha L \cancel{\|\nabla f(x^k)\|^2} - \cancel{\|\nabla f(x^k)\|^2} = 0$$

$$\boxed{\alpha = \frac{1}{L}}$$

$\rightarrow$ Reduction in the loss: $f(x+\alpha d) - f(x)$ = Reduction in the loss

$$f(x+\alpha d) - f(x) \leq -\frac{1}{L}\|\nabla f(x)\|^2 + \frac{1}{2L}\|\nabla f(x)\|^2 = \frac{1}{2L}\|\nabla f(x)\|^2$$

$$\boxed{f(x+\alpha d) - f(x) \leq \frac{1}{2L}\|\nabla f(x)\|^2}$$ $\rightarrow$ Maximum Possible loss reduction

3. Computational complexity of backpropagation

Consider an MLP (aka feedforward) deep neural network, with k dense hidden layers each consisting of n neurons with ReLU activations, and a final dense layer with a single neuron whose value is used in a binary classification task. For simplicity, we assume the input is a vector of size n. We train the network with batches of size b input samples.

- in terms of k, n, and b, how many floating point operations (FLOPS) are needed to evaluate the network for a batch? (write down the dominant term only, in $O()$ notation)
- in terms of k, n, and b, how many FLOPS are performed during the backward phase of backpropagation for computing the gradient of the loss from a batch?
- in terms of k, n, and b, how much memory is needed during the forward/backward passes of backpropagation?

Model →

layer_1 : n neurons → $2n \times n \times b$
ReLU → $n \times b$
layer_2 : n neurons → $2n \times n \times b$
⋮
layer_K : n neurons → $2n \times n \times b$
ReLU ⋮
Dense (1) → $2n \times b$

Forward Pass: $K(2n^2)b + Knb + 2nb$

: $2Kn^2b + Knb + 2nb$

→ Evaluating a batch we will have $O(Kn^2b)$

Backward Pass

→ For computing the gradients for each layer :

layer_1 : n neurons → $2n \times n \times b$
ReLU → $n \times b$
layer_2 : n neurons → $2n \times n \times b$) $2n \times n$ → Derivatives
⋮
layer_K : n neurons → $2n \times n \times b$) $2n \times n$ → Derivative
ReLU ⋮
Dense (1) → $2n \times b$

Flops: $Kn^2 + Knb$

→ Backward Pass: $O(K(n^2 + nb))$

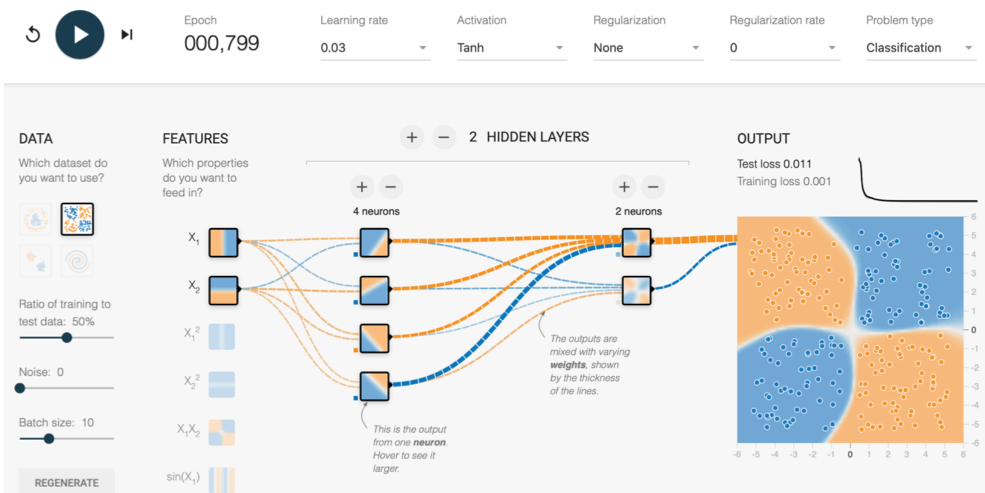
Memory usage

x_0 layer_1 : n neurons → $n \times n$
ReLU → n
x_1 layer_2 : n neurons → $n \times n$
⋮
x_K layer_K : n neurons → $n \times n$
ReLU → n
Dense (1) → n

1 → $x_0, x_1 \cdots x_K$ → then → Knb
2 → $w_0, w_1, \cdots w_K$ → then → Kn^2
3 → gradient $w_0 \cdots w_K$ → Kn^2

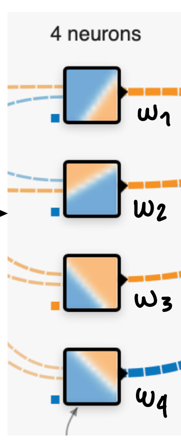
memory to save $= Knb + Kn^2 + Kn^2 = Kn(b+n)$

4. Interpretation of neurons response.
`playground.tensorflow.org` has a visual interface to an MLP network for binary classification of datasets whose elements consist of two features. There are a few sample datasets illustrated. Consider the one that consists of a class of points in the first and third quadrants, and a second class in the second and fourth quadrants (see screenshot below). Use a trained network similar to the one below to answer the following questions.



- give a very brief description of the features that each of the four neurons of the first hidden layer appear to be detecting? The description should be in terms of the geometry of the classification. (recall that the output of a neuron in this layer is $z_j = \varphi(w_j^T x)$, i.e., it "lights up" when the input x matches the template defined by its w_j weights—we are basically trying to give a geometric interpretation of each of these w_j templates)
- consider the neuron of the second hidden layer that most strongly affects the final output. Can you briefly explain how the feature that it is detecting is obtained from a combination of the features of the neurons from the first layer? Could this feature have been generated by a direct combination of the input features x, or is the first layer necessary here?

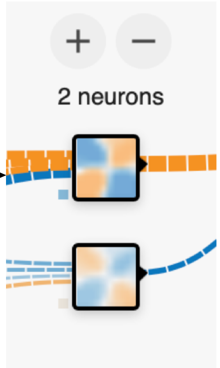
First hidden layer →



→ neuron 1 takes the interaction of x_1, x_2 and lightsup for the data that is avobe the white slopped line.
→ neuron 2 take also the interaction of x_1, x_2 and lights up for the data that is below the white line.
→ neuron 3 detects the data that belong to the upper Right Part and assign a negative value for the other side divided by the white boundary.
→ Similar to neuron 3, this assign a negative value for data in the left bottom corner.

→ the first hidden layer detect specific portions where the data belong. the four neurons lights up when the data belong to the zone that they learned to detect.

Second hidden layer



→ we can see that the first neuron is that affects strongly the output. Combining the decision of the detection of the different zones in previous layers Creating more complex boundaries to separate the data. The first layer detect some zones that are no horizontal neither vertical to then pass that decision to create a more suitable boundary.

For this reason, this generated feature in the last layer can not be implemented only using the original data x_1, x_2. This last neuron learn to conclude from the zones or feature classifications in a more complex geometry boundaries.