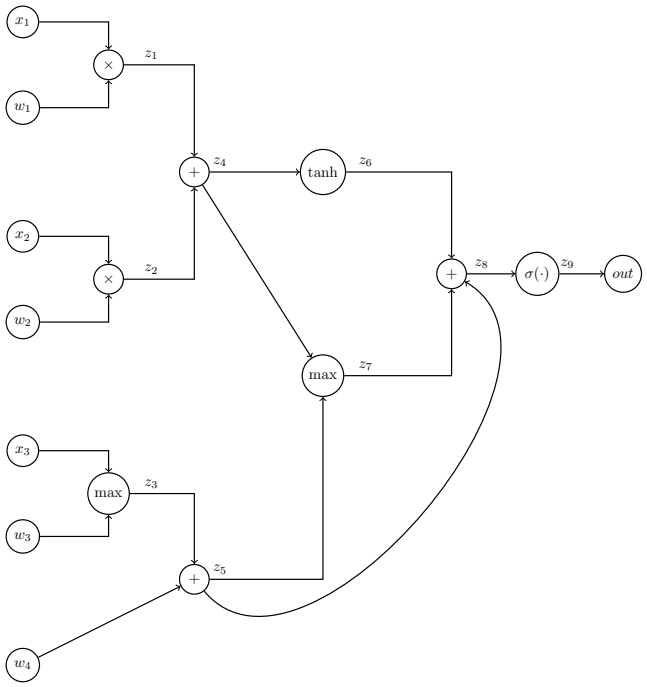
Exercise 2: Deep Learning

Ehsan Mousavi, Kasra Alishahi

December 16, 2023

Problem 1: Computational Graph

Let $f(x_1, x_2, x_3, w_1, w_2, w_4)$ be a model represented as the following computation graph.



Here. $\sigma(x) = \frac{1}{1+e^{-x}}$ is the Sigmoid function.

i.) For each node, write the output as an expression of inputs. For example, $z_5 = w_4 + z_3$.

ii.) Given the inputs $x_1 = 1, x_2 = 3, x_3 = 2,\ w_1 = 2, w_2 = 1, w_3 = 5, w_4 = 2$, use the forward pass algorithm to compute function output f.

iii.) Use backward-pass to compute $\frac{\partial f}{\partial x_i}$, $\frac{\partial f}{\partial w_i}$ and $\frac{\partial f}{\partial z_i}$ for $i = 1, 2, 3$ at the given point $x_1 = 1, x_2 = 3, x_3 = 2$, $w_1 = 2, w_2 = 1, w_3 = 5, w_4 = 2$.

Problem 2: Backpropation

In this exercise, we implement the gradient descent and backpropagation algorithms for a multi-layer DNN. Consider a neural network architecture with L hidden layers, where all hidden layers have d nodes. The input layer has $d_{\text{in}} = 5$ nodes, and the output layer has $K = 3$ nodes. The final layer utilizes the softmax activation, and the loss function is cross-entropy. The activation function applied to all layers is ELU with $\alpha = 1$.

i.) Implement the feedforward algorithm. Report the output of the network for $L = 5$, $d = 10$, $x_i = i$ for $i = 1, \ldots, 5$. Initialized the weights by following function:

```
def initialize_weights(layer_dimensions):
    np.random.seed(42)  # For reproducibility

    parameters = {}
    L = len(layer_dimensions) - 1  # Number of layers excluding input layer

    for l in range(1, L + 1):
        # He initialization for weights
        parameters[f"W{l}"] = np.random.randn(
            layer_dimensions[l], layer_dimensions[l - 1]
            ) * np.sqrt(2 / layer_dimensions[l - 1])
        # Initializing biases to zeros
        parameters[f"b{l}"] = np.zeros((layer_dimensions[l], 1))
    return parameters
```

Here's the corrected version of the text:

ii.) Implement the Backpropagation algorithm. Set the number of layers to $L = 30$. Compute the norm of the derivative for layer l, denoted as $\delta_l = |\nabla_{W_l} \ell(f(x), y)|$, where $y = (1, 0, 0)$. Plot δ_l versus l and provide an explanation for your observations.

iii.) Implement SGD, SGD with momentum, and Nesterov momentum. Set the number of layers to $L = 5$. For the provided dataset, plot $\|W_t\|_2$, representing the norm of the updated weights in iteration t. Additionally, plot the evolution of loss and accuracy on the train and test sets using W_t as the updated weights in iteration t. Use an appropriate learning rate and set $\rho = 0.9$ for momentum.

For the last part of the exercise, use : "diabetes health indicators dataset"

Problem 3: Stochastic Gradient Descent

The goal of this exercise is to analyze the rate of convergence of stochastic gradient descent. Let $f(w) = E[g(X, w)]$, a differentiable function. The stochastic gradient descent sequence is defined by

$$w_{k+1} = w_k - \alpha_k \nabla_w g(X_k, w)$$

where α_k is the learning rate, and $X_1, X_2, \ldots$ is a sequence of i.i.d variables. Consider the following assumptions:

Assumption 1 *Let f be a differentiable function for which ∇f is L-Lipschitz. Equivalently*

$$f(\tilde{w}) \leq f(w) + \nabla f(w)^T(\tilde{w} - w) + \frac{L}{2}||\tilde{w} - w||_2^2 \quad \text{for all } \tilde{w}, w \in \mathbb{R}^d$$

Assumption 2 *There exists a constant σ^2 such that*

$$E[\|\nabla_w g(w, X)\|^2] \leq \sigma^2$$

for all $w \in \mathbb{R}^d$

i.) Assume that (1) holds, show that

$$E[f(w_{k+1})] \leq f(w_k) - \alpha_k \|\nabla f(w_k)\|^2 + \frac{\alpha_k^2 L}{2} E[\|\nabla_w g\|^2]$$

ii.) Assume that (1, 2) holds then prove that

$$\sum_{k=0}^{t-1} \alpha_k E[\|\nabla f(w_k)\|^2] \leq f(w_0) - E[f(w_t)] + \frac{\sigma^2 L}{2} \sum_{k=0}^{t-1} \alpha_k^2$$

iii.) Assume that (1, 2) holds and $f(w)$ is bounded below. Let $f^* = \min_w f(w)$, show that

$$\min_{0 \leq k \leq t} E[\|\nabla f(w_k)\|^2] \leq \frac{f(w_0) - f^*}{\sum_{k=0}^{t-1} \alpha_k} + \frac{\sigma^2 L}{2} \frac{\sum_{k=0}^{t-1} \alpha_k^2}{\sum_{k=0}^{t-1} \alpha_k}$$

iv.) What is R.H.S if the learning rate $\alpha_k = \alpha$ is constant? What if $\alpha_k = \frac{1}{k^{3/4}}$? How about if $\alpha_k = \frac{1}{k^{1/2}}$?

Problem 4: Dropout

In this exercise, we will examine the additive regularization effects of dropout. For simplicity, we will only consider node dropout is applied to only layer i of the network. Let h_i denote the i-th hidden layer of the network, and let F_i denote

the composition of the layers after h_i. That is, F_i represents the function that takes h_i as input and outputs the model prediction. Thus, the full network is expressed as $F(x) = F_i(h_i(x))$.Let's $\vec{\eta} \in \mathbb{R}^d$ be random vector with independent coordinates.

$$\eta_k = \begin{cases} -1 & \text{with prob } q \\ \frac{q}{1-q} & \text{with prob } 1-q \end{cases}$$

i.) Show that $E[\vec{\eta}] = 0$ and argue that we can apply dropout by computing $h_i^{drop} = (1 + \vec{\eta}) \odot h_i(x)$. Here, $v_1 \odot v_2$ refers to the entry-wise product for vectors v_1, v_2. *Hint: The correct answer is not longer than two lines.!!!!*

ii.) For small enough q, show that

$$E_\eta[\ell(F(x,\eta), y)] - \ell(F(x), y) \approx \frac{q}{2(1-q)} \left\langle \mathcal{D}^2_{h_i}(\ell \circ F_i)[h_i], diag(h_i(x) \odot h_i(x)) \right\rangle. \tag{1}$$

Here, $\langle A, B \rangle = tr(A^T B)$ is the inner product of matrices $A, B \in \mathbb{R}^{d \times d}$. Also, $\mathcal{D}^2_u(g)[b]$ represents second derivative of g with respect to u evaluated at point $u = b$. Finally, $H = diag([r_1, \ldots, r_m])$ is a diagonal matrix such that $H[i,i] = r_i$ and $H[i,j[= 0$ if $i \neq j$.

Note: It can be argue that this approximation holds even for non-small q

iii.) For linear case $F(x) = Wx$ and square error loss $\ell(\hat{y}, y) = (\hat{y} - y)^2$, compute the right hand side of (1). Interprets your observation.

Problem 5: Effect of Transfer learning

In this exercise, we are interested to build model to detect faces with and without masks by taking advantage of Transfer Learning. Transfer learning is a machine learning technique where a model trained on one task is reused or adapted as a starting point for a related task. In this task, we compare the performance of a simple CNN model and transfer learning model.

i.) Follow the instruction in colab, to download face-mask-detection dataset in colab. Use the provided code to explore data set, resize the image and creating training and validation Sets.

ii.) Build a CNN model. The model includes 4 convolution layers with Relu function follows by two dense layers. Train the model and plot the evolution of loss and accuracy over train and validation set.

iii.) Add Batch normalization to each convolution layers and dropout to the first dense layer. Train the model (by correct initialization) and plot the evolution of loss and accuracy over train and validation set.

iv.) Utilize the pre-trained ResNet34 model both with and without pre-trained weights on the ImageNet dataset. Add a linear layer to the output of the ResNet34. Proceed to train the model in both scenarios (with and without pre-trained weights) and compare the accuracy and loss values between the training and validation sets with the previous case.