Lecture 5c - Softmax in depth + ReLU and Subgradients

Contents

10 Softmax in depth + ReLU and Subgradients

10.1 Assumptions used in Lecture 5b

In Lecture 5b, we assumed:

1. The computation of the output of 1 neuron does not directly affect computation of other neurons in the same or previous layers
2. Input neurons only combine through weighted activation
3. Activations are differentiable

All of the above assumptions are (frequently) practically not application. (Some) Cases in which above assumptions are void:

1. Vector Activations
2. Multiplicative Networks (Skipping)
3. Non-Differentiable Activations

10.2 Scalar Activations

Scalar Activations refer to situations in neural networks where each output is influenced primarily by a single input rather than being a function of all inputs. In scalar activations, each output can be expressed as:

$$y_j = f(z_j) \tag{1}$$

y_j is the j-th output. z_j is the j-th input or the weighted sum of the inputs feeding into the j-th neuron. f is an activation function applied to z_j.
Each output can be calculated independently based on its corresponding input, meaning that modifying one input does not directly affect other outputs. The derivatives with respect to the inputs are simpler since each output depends solely on its specific input. **Sigmoid is a scalar activation.**

10.2.1 Derivatives of Scalar Activations

For Scalar Activations, the derivative of the loss w.r.t the input to the unit is a simple product of the derivatives. i.e.:

$$\frac{\partial \mathcal{D}}{\partial z_i^{(k)}} = \frac{\partial \mathcal{D}}{\partial y_i^{(k)}} \frac{\partial y_i^{(k)}}{\partial z_i^{(k)}} \tag{2}$$

10.3 Vector Activations

Vector Activations

In vector activations, all outputs are functions of all inputs. This interconnectedness means that any change in an input will affect all outputs.

Any perturbation in the input vector $\mathbf{z}$ will modify all components of the output vector $\mathbf{y}$. Consequently, any perturbation of an output vector component can be attributed to all input components. Modifying a single component of the input vector leads to collective changes in the output vector. In vector activations, each output can be expressed as:

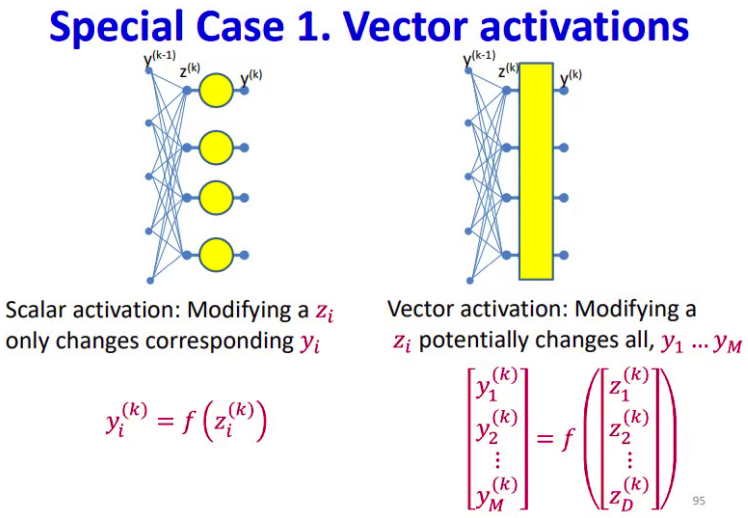
$$\mathbf{y} = f(\mathbf{z}) \tag{3}$$

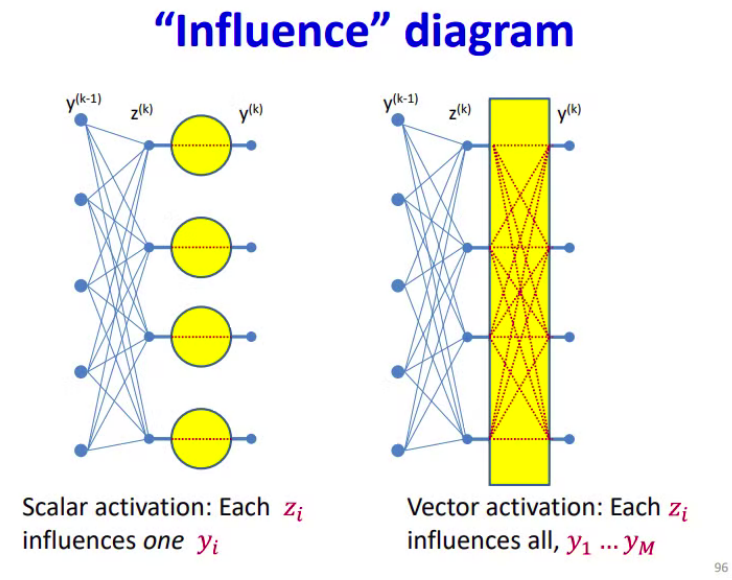
10.3.1 Derivatives in Vector Activations

For vector activations, the relationship between the derivatives is more complex because each output is influenced by all inputs. The derivative of the loss with respect to the input to a unit must account for all outgoing paths from that input. The equation can be expressed as follows:

$$\frac{\partial \mathcal{D}}{\partial z_j^{(k)}} = \sum_i \frac{\partial \mathcal{D}}{\partial y_i^{(k)}} \frac{\partial y_i^{(k)}}{\partial z_j^{(k)}} \tag{4}$$

The summation indicates that you are considering contributions from all output components $y_i^{(k)}$ that depend on the input $z_j^{(k)}$.





10.4 Softmax Function

Softmax Function is a Vector Activation Function. The softmax function is commonly used in the output layer of neural networks for multi-class classification problems. The softmax function transforms the vector $\mathbf{z}$ into a probability distribution $\mathbf{y}$ defined as:

$$y_j = \frac{e^{z_j}}{\sum_k e^{z_k}} \quad \text{for } j = 1, 2, \ldots, m$$

Where y_j is the probability associated with class j.

10.4.1 Properties of Softmax

- **Probability Interpretation:** Each y_j represents the probability that the input belongs to class j, with the constraint that the sum of all probabilities equals 1:

$$\sum_{j=1}^{m} y_j = 1$$

- **Effect of Incrementing z:** If the z_j for a particular class is increased, y_j will increase, while the other probabilities y_k (for $k \neq j$) will decrease due to normalization.

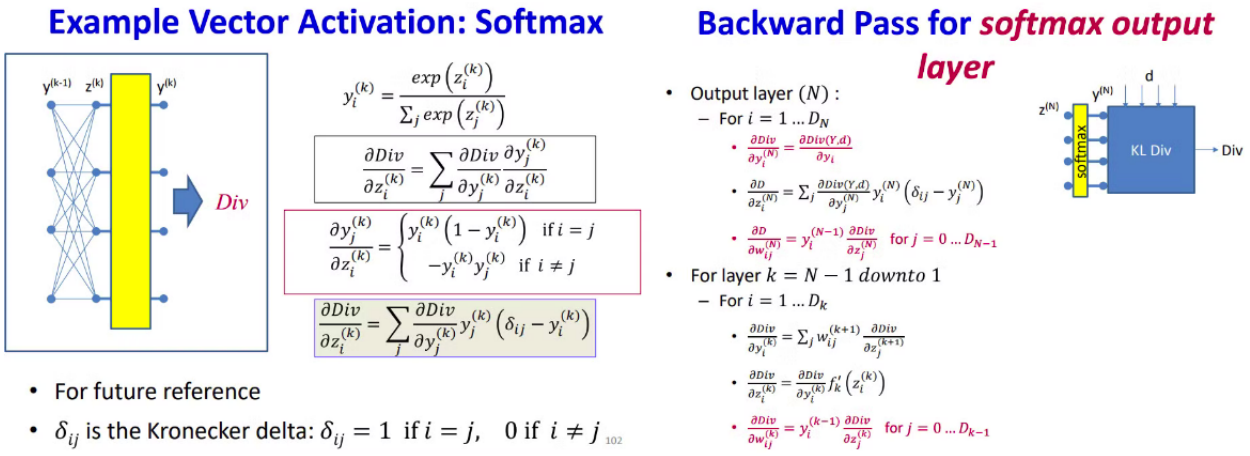
10.4.2 Derivative Analysis

The derivative of y_j with respect to z_i can be expressed as:

$$\frac{dy_j}{dz_i} = y_j(\delta_{ij} - y_i)$$

Where δ_{ij} is the Kronecker delta, which is 1 if $i = j$ and 0 otherwise. This shows that:

- For the class corresponding to the incremented z, the derivative is positive.
- For other classes, the derivative is negative.



10.5 ReLU Activation Function

The ReLU (Rectified Linear Unit) activation function is commonly used in neural networks due to its simplicity and effectiveness. It is defined as follows:

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \tag{5}$$

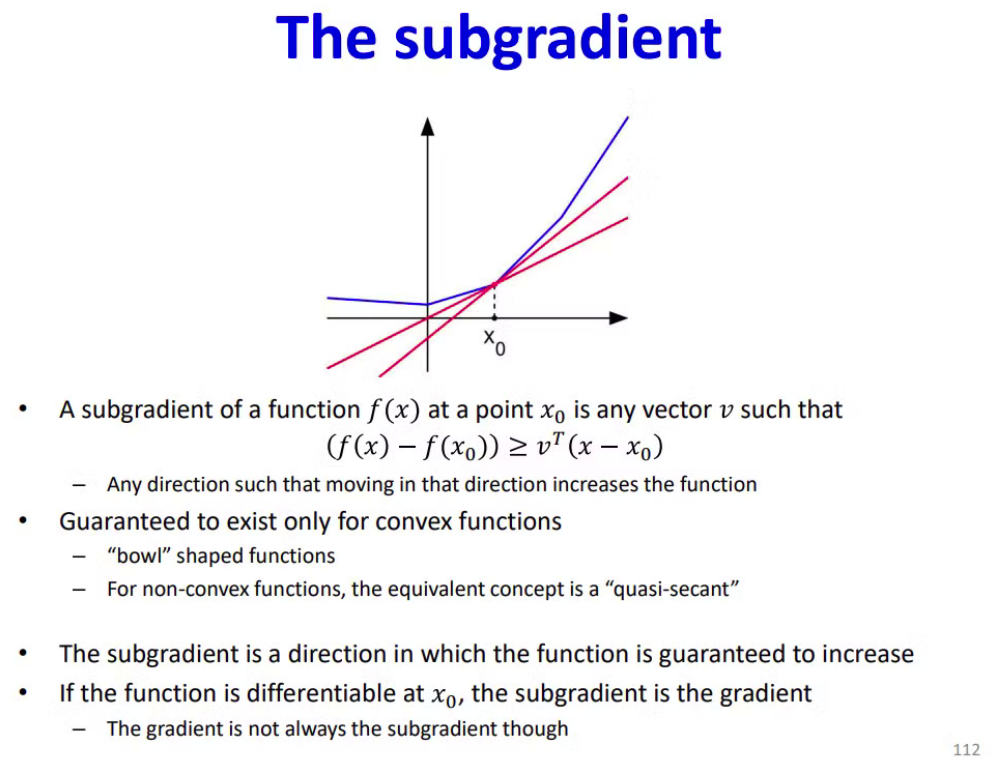
10.5.1 Continuity and Non-Differentiability

- **Continuity:** The ReLU function is continuous because it does not have jumps or holes. Approaching 0 from either direction yields the same output:

$$\lim_{x \to 0^-} f(x) = 0 \quad \text{and} \quad \lim_{x \to 0^+} f(x) = 0$$

- **Non-Differentiability at Zero:** The function is not differentiable at $x = 0$ because the slope is undefined at that point. Thus, we cannot directly compute the derivative at $x = 0$.

- **Sub-gradient:** At non-differentiable points like $x = 0$, we use the concept of a sub-gradient. The sub-gradient at this point can be any value between 0 and 1 because moving in either direction may decrease the loss.



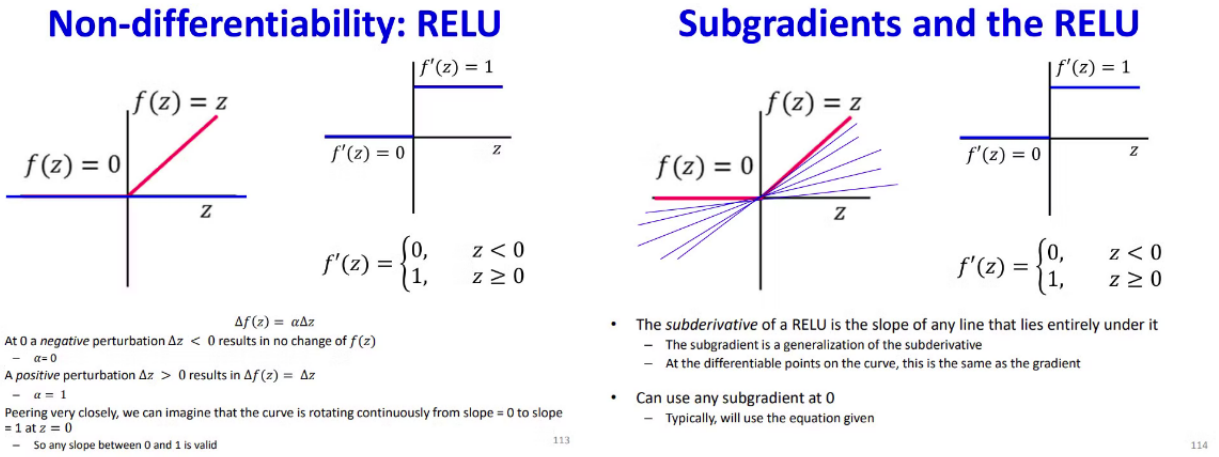
10.6 Implications for Gradient Descent

When adjusting parameters to minimize the loss, we use the derivative (or sub-gradient) to determine the direction of modification. The derivative indicates how to adjust the input values to decrease the loss effectively.

10.6.1 Max Activation Function

The max activation function outputs the maximum value from a collection of inputs, which can be viewed as a special case of ReLU. It can be defined mathematically as:

$$f(x_1, x_2, \ldots, x_n) = \max(x_1, x_2, \ldots, x_n) \tag{6}$$



10.7 Conclusion and Introduction to Vector Formulation

10.7.1 Recap of Gradient Descent Process

1. **Initialization:** We start by initializing weights and parameters for our neural network.
2. **Forward Pass:**
   - For each training instance, pass the input through the network.
   - Aggregate the loss using a loss function, typically the mean squared error or cross-entropy loss.
3. **Backward Pass:**
   - Compute the derivatives of the loss with respect to each parameter.
   - Aggregate these derivatives across all training instances.
4. **Parameter Update:**
   - Update the parameters using the average derivative to minimize the loss.

The loss function can be expressed as the average divergence over all training instances:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{D}(y_i, \hat{y}_i) \tag{7}$$

Where N is the number of training instances, y_i is the true label, and $\hat{y}_i$ is the predicted output.

10.8 Vector and Matrix Representation

To enhance computational efficiency, we can represent operations using vectors and matrices:

- Let $\mathbf{z}_k$ be a vector of all affine terms in the k-th layer, calculated as:

$$\mathbf{z}_k = \mathbf{W}_{k-1}\mathbf{y}_{k-1} + \mathbf{b}_k \tag{8}$$

  Where:

  - $\mathbf{W}_{k-1}$ is the weight matrix connecting layer $k-1$ to layer k.
  - $\mathbf{y}_{k-1}$ is the vector of outputs from the previous layer.
  - $\mathbf{b}_k$ is the bias vector for layer k.

- The output of the activation function in the k-th layer can be expressed as:

$$\mathbf{y}_k = f_k(\mathbf{z}_k) \tag{9}$$

  Where f_k denotes the activation function applied to the affine term vector $\mathbf{z}_k$.

10.9 Summary

Utilizing vector and matrix operations allows for more efficient computations in neural networks, facilitating faster training and inference processes. This approach leverages optimized libraries, which significantly reduce the complexity of calculations.

Backward Pass: Recap

- Output layer (N) :
  - For $i = 1 \ldots D_N$
    - $\frac{\partial Div}{\partial y_i^{(N)}} = \frac{\partial Div(Y,d)}{\partial y_i}$
    - $\frac{\partial Div}{\partial z_i^{(N)}} = \frac{\partial Div}{\partial y_i^{(N)}} \frac{\partial y_i^{(N)}}{\partial z_i^{(N)}}$ OR $\sum_j \frac{\partial Div}{\partial y_j^{(N)}} \frac{\partial y_j^{(N)}}{\partial z_i^{(N)}}$ (vector activation)
    - $\frac{\partial Div}{\partial w_{ji}^{(N)}} = y_j^{(N-1)} \frac{\partial Di}{\partial z_i^{(N)}}$ for $j = 0 \ldots D_k$
- For layer $k = N-1$ *downto* 1
  - For $i = 1 \ldots D_k$
    - $\frac{\partial Div}{\partial y_i^{(k)}} = \sum_j w_{ij}^{(k+1)} \frac{\partial Div}{\partial z_j^{(k+1)}}$
    - $\frac{\partial Di}{\partial z_i^{(k)}} = \frac{\partial Div}{\partial y_i^{(k)}} \frac{\partial y_i^{(k)}}{\partial z_i^{(k)}}$ OR $\sum_j \frac{\partial Div}{\partial y_j^{(k)}} \frac{\partial y_j^{(k)}}{\partial z_i^{(k)}}$ (vector activation)
    - $\frac{\partial Div}{\partial w_{ji}^{(k)}} = y_j^{(k-1)} \frac{\partial Div}{\partial z_i^{(k)}}$ for $j = 0 \ldots D_k$

These may be subgradients

119

Training by BackProp

- Initialize weights $\boldsymbol{W^{(k)}}$ for all layers $k = 1 \ldots K$
- Do: *(Gradient descent iterations)*
  - **Initialize** $Loss = 0$; For all i, j, k, initialize $\frac{dLoss}{dw_{i,j}^{(k)}} = 0$
  - For all $t = 1{:}T$ *(Iterate over training instances)*
    - **Forward pass:** Compute
      - Output $\boldsymbol{Y_t}$
      - $Loss \mathrel{+}= \boldsymbol{Div(Y_t, d_t)}$
    - **Backward pass:** For all i, j, k:
      - Compute $\frac{\boldsymbol{dDiv(Y_t, d_t)}}{dw_{i,j}^{(k)}}$
      - $\frac{dLoss}{dw_{i,j}^{(k)}} \mathrel{+}= \frac{\boldsymbol{dDiv(Y_t, d_t)}}{dw_{i,j}^{(k)}}$
  - For all i, j, k, update:

$$w_{i,j}^{(k)} = w_{i,j}^{(k)} - \frac{\eta}{T} \frac{dLoss}{dw_{i,j}^{(k)}}$$

- Until $Loss$ has converged

121