Deep Learning **Summer Term 2024**

http://ml3.leuphana.de/lectures/summer24/DL
Machine Learning Group, Leuphana University of Lüneburg

Soham Majumder (soham.majumder@leuphana.de)

Classwork

1. Sigmoidal Function

Let $Y \in \{0, 1\}$ denote a binary random variable that depends on k other random variables X_i as:

$$P(Y = 1 \mid X_1 = x_1, X_2 = x_2, \ldots, X_k = x_k) = \sigma\left(\sum_{i=1}^{k} w_i x_i\right)$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$. The real-valued parameters w_i here are known as weights. The so-called sigmoid function $\sigma(z)$ arises in many contexts. In neural networks, it models the probability that a neuron Y fires given its input from other neurons X_i; the weights w_i describe the connections between neurons. In statistics, the sigmoid function appears in models of logistic regression. Sketch the function $\sigma(z)$, and verify the following properties:

(a) $\sigma'(z) = \sigma(z)\sigma(-z)$.

(b) $\sigma(-z) + \sigma(z) = 1$.

(c) $\sigma(z) = \frac{\exp(z)}{1+\exp(z)}$

(d) $L(\sigma(z)) = z$, where $L(p) = \log\left(\frac{p}{1-p}\right)$ is the log-odds function.

(e) $w_i = L(p_i)$, where $p_i = P(Y = 1 \mid X_i = 1, X_j = 0 \text{ for all } j \neq i)$.

Solution

(a)

$$\sigma'(z) = \frac{0 + 1 \cdot e^{-z}}{(1 + e^{-z})^2} = \frac{1}{1 + e^{-z}} \cdot \frac{e^{-z}}{1 + e^{-z}} = \sigma(z)\sigma(-z)$$

(b)

$$\sigma(-z) + \sigma(z) = \frac{1}{1 + e^{z}} + \frac{1}{1 + e^{-z}} = 1$$

(c)

$$\sigma(z) = \frac{1}{1 + \exp(-z)} = \frac{\exp(z)}{\exp(z) + \exp(z)\exp(-z)} = \frac{\exp(z)}{\exp(z) + \frac{\exp(z)}{\exp(z)}} = \frac{\exp(z)}{1 + \exp(z)}$$

(d)

$$L(\sigma(z)) = \log\left(\frac{\sigma(z)}{1-\sigma(z)}\right) = \log\left(\frac{1/(1+e^{-z})}{1/(e^{z}+1)}\right) = \log(e^{z}) = z$$

(e)

$$L(p_i) = \log\left(\frac{P(Y=1 \mid X_i=1, X_j=0, \forall j \neq i)}{1-P(Y=1 \mid X_i=1, X_j=0, \forall j \neq i)}\right) = L(\sigma(w_i))$$

According to the conclusion from (d), $L(\sigma(w_i)) = w_i$.

2. Softmax Function

Let $\mathbf{x} \in \mathbb{R}^d$ be a vector. The softmax function $\sigma : \mathbb{R}^d \to [0,1]^d$ is given by

$$\sigma(\mathbf{x})_j = \frac{\exp(x_j)}{\sum_{k=1}^{d} \exp(x_k)} \quad \text{for } j = 1, \ldots, d.$$

(a) Let $\mathbf{x} \in \mathbb{R}^d$ be a vector. What is the main difference of using the softmax $\sigma(\mathbf{x})$ and a sigmoid per dimension $\sigma(x_j)$?

(b) Show for any $\mathbf{x} \in \mathbb{R}^d$ that $\sum_{j=1}^{d} \sigma(\mathbf{x})_j = 1$.

Solution

(a) Consider the problem of classification. The softmax function is normalizing the output such that it sums up to one. Hence, every entry can be interpreted as a probability. This is handy for multi-class classification, where there is a correct class among many wrong ones. On the other hand, the sigmoid function per dimension gives an individual probability per class. That is useful for multi-label classification, where an input instance can belong to multiple classes.

(b)

$$\sum_{j=1}^{d} \sigma(\mathbf{x})_j = \sum_{j=1}^{d} \frac{\exp(x_j)}{\sum_{k=1}^{d} \exp(x_k)} = \frac{1}{\sum_{k=1}^{d} \exp(x_k)} \sum_{j=1}^{d} \exp(x_j) = 1$$

3. Expressive Power of Neural Network

Show that adding layers to a linear deep network, i.e., a network without non-linearity, can never increase the expressive power of the network. Give an example where it actively reduces it.

Solution

Theoretical Explanation

- **Definition of Linear Layers:** A linear layer in a neural network can be represented as a matrix multiplication followed by a bias addition. For layer l, the transformation it applies

to its input $\mathbf{x}$ can be represented as:

$$\mathbf{y} = \mathbf{W}_l\mathbf{x} + \mathbf{b}_l.$$

where $\mathbf{W}_l$ is the weight matrix and $mathbfb_l$ is the bias vector.

- **Composition of Linear Layers:** Consider a network with two consecutive layers. The output $\mathbf{y}$ from these layers, given input $\mathbf{x}$ can be written as:

  $$\mathbf{y} = \mathbf{W}_2(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2.$$

  Simplifying, we get

  $$\mathbf{y} = \mathbf{W}_2\mathbf{W}_1\mathbf{x} + \mathbf{W}_2\mathbf{b}_1 + \mathbf{b}_2.$$

  We can denote a new weight matrix $\mathbf{W}' = \mathbf{W}_2\mathbf{W}_1$ and a new bias vector $\mathbf{b}' = \mathbf{W}_2\mathbf{b}_1 + \mathbf{b}_2$. This shows that the composition of two linear transformations is still a linear transformation.

- **Generalization to Multiple Layers:** The final output of a deep linear network can always be represented as a single matrix multiplication and a bias addition, $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$, where $\mathbf{W}$ and b are the effective cumulative weight matrix and the bias vector of the entire network.

  Thus, since any number of linear layers just results in a single linear transformation, the network's expressive power, defined as its ability to represent different functions, remains the same no matter how many layers are added. It can only represent linear mappings from inputs to outputs.

Example Demonstrating Reduction in Expressive Power

- **Initialization to Zero:** Suppose each layer l initializes weights $\mathbf{W}_l$ to a zero matrix. The output will always be the bias of the last layer, disregarding any input, that is, $\mathbf{y} = \mathbf{b}_n$, where n is the number of layers.

- **Rank Deficiency:** If $\mathbf{W}_1$ has a lower rank, the space of representable functions is limited to a subspace of the output space. Adding more layers like $\mathbf{W}_2, \mathbf{W}_3, \ldots$ cannot expand this space and might further restrict it if these matrices also have rank deficiencies.

  Example: $\mathbf{W}_1 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \mathbf{W}_2 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$, then $\mathbf{W}_2\mathbf{W}_1 = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}$. The product $\mathbf{W}_2\mathbf{W}_1$ has a rank of 1, regardless of the individual configurations of $\mathbf{W}_1$ and $\mathbf{W}_2$, limiting the expressive capability of the network to just projecting inputs onto a line, thus actively reducing its ability to capture more complex patterns that a single full-rank layer might capture.

  This example illustrates how, in specific cases, adding more layers to a linear network can not only fail to increase but may actually decrease the expressive power of the network.

4. Activation Function Similarities

(a) Consider the following activation function: $h(x) = x\sigma(\beta x)$ where σ denotes the sigmoid function $\sigma(z) = \frac{1}{1+\exp(-z)}$. Show that, for appropriately chosen values of β, this activation function can approximate (1) the linear activation function and (2) the ReLU activation function.

(b) Show that $\tanh(x) + 1 = 2 \cdot \text{sigmoid}(2x)$

(c) Prove that the function classes parameterized by *tanh* and *sigmoid* non-linearities are identical. Hint: affine layers have bias terms, too.

Solution

(a) Given $h(x) = x\sigma(\beta x)$ where σ is the usual sigmoid activation function.

$$h(x) = \frac{x}{1 + \exp(-\beta x)}$$

Case 1: Linear approximation

Take $\beta = 0$, we will have

$$h(x) = \frac{x}{2} \quad \text{[Linear]}$$

Case 2: Approximating ReLU

Take $\beta \to \infty$, we will have $\exp(-\beta x) \to \infty$ for all $x < 0$ and $\exp(-\beta x) \to 0$ for $x \geq 0$. Thus, we get

$$h(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

Hence, we can approximate ReLU using the given activation function $h(x)$, a.k.a Swish activation function.

(b)

$$\begin{aligned} \tanh(x) &= \frac{e^x - e^{-x}}{e^x + e^{-x}}, \\ \tanh(x) + 1 &= \frac{e^x - e^{-x} + e^x + e^{-x}}{e^x + e^{-x}} = \frac{2e^x}{e^x + e^{-x}}, \\ \tanh(x) + 1 &= 2 \cdot \frac{e^x}{e^x + e^{-x}} = 2 \cdot \frac{e^{2x}}{e^{2x} + 1} = 2 \cdot \text{sigmoid}(2x). \end{aligned}$$

(c) We can consider the general form of a neural network layer with either activation function. For both, a layer takes the form: $\mathbf{y} = \mathbf{f}(\mathbf{W}\mathbf{x} + \mathbf{b})$, where $\mathbf{f}$ is the nonlinearity(tanh or σ) $\mathbf{W}$ is the weight matrix, $\mathbf{x}$ is the input and b is the bias vector.

Given the transformation $\tanh(x) + 1 = 2 \cdot \sigma(2x)$ and the ability to adjust $\mathbf{W}$ and $\mathbf{b}$ in neural networks, any function computable by a tanh network can also be computed by a σ network and vice versa. Specifically, using the transformation from 4.(b), the addition of 1 and multiplication by 2 can be absorbed into the bias and weight scaling of the network layers. This manipulation is possible due to the presence of the bias term b in each layer, which allows for affine transformations necessary to convert between scaled and shifted versions of *tanh* and σ. Through this example, we can see the interconnections between some commonly used activation functions.