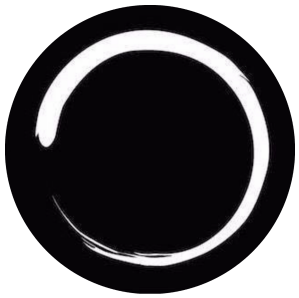
inzva DLSG

Homework 1 Solutions



Open Questions

1. Deep learning is a subset of machine learning that uses deep neural networks to automatically learn patterns from large amounts of data. A key concept of deep learning is the **universal approximation theorem**, which states that a neural network with at least one hidden layer and a sufficient number of neurons can approximate any continuous function to a desired degree of accuracy. This makes neural networks highly flexible and capable of modeling complex, nonlinear relationships in data that cannot be covered by machine learning algorithms.

2. Forward propagation is the neural network process in which inputs are passed through the NN to produce the output. In a feedforward neural network, data starts at the input layer, then moves through hidden layers, and finally reaches the output layer. At each layer, the input is multiplied by weights, added to a bias, and passed through an activation function like ReLU, sigmoid or tanh to introduce non-linearity to the model (additional explanation in Q4).

3. Backpropagation is the neural network process in which the weights of neural network is updated to minimize the designated loss function. After forward propagation, where the network produces an output, backpropagation works by calculating the error/loss between the predicted output and the actual target. This error is propagated backward through the network, layer by layer, starting from the output layer and moving toward the input layer. At each layer, the gradient of the loss function with respect to the weights is computed using the chain rule. These gradients indicate how much each weight contributed to the error, allowing the network to adjust the weights accordingly.

   Backpropagation is essential for neural networks because it enables the model to learn from data by reducing prediction errors. Without backpropagation or in the case of an error in the calculation of partial derivates, the network would not know how to adjust its weights effectively to improve performance and the learning would be impossible.

4. Activation functions are used in the layers of a neural network to introduce non-linearity, which allows the network to model complex patterns and relationships in the data. Without activation functions, the network would simply become a huge linear transformation model, meaning it could only capture linear relationships between the input and output. Non-linear activation functions like ReLU, sigmoid, and tanh enable the network to learn non-linear features and make the model capable of handling tasks such as image recognition, natural language processing, and more, where data relationships are highly non-linear.

   Consider a simple neural network with L layers. Let the input to the network be $\mathbf{x}$, and each layer applies a linear transformation using weights and biases. The output of layer l is represented as:

   $$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{z}^{(l-1)} + \mathbf{b}^{(l)}$$

   where $\mathbf{z}^{(l)}$ is the output, $\mathbf{W}^{(l)}$ is the weight, $\mathbf{b}^{(l)}$ is the bias for layer l and $\mathbf{z}^{(0)} = \mathbf{x}$ is the input to the network. Without activation functions, the final output $\mathbf{z}^{(L)}$ can be expressed as:

$$\mathbf{z}^{(L)} = \mathbf{W}^{(L)}(\mathbf{W}^{(L-1)}(\cdots(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)})\cdots) + \mathbf{b}^{(L)} = \mathbf{W}^{*}\mathbf{x} + \mathbf{b}^{*}$$

which is just a linear transformation.

5. For the neural network in the provided image, we have three layers. Each layer contains 6, 4, and 3 neurons, respectively. Our input vector is a 4×1 vector. Assuming arbitrary activation functions σ, it will be applied element-wise. Therefore, we will have the following forward propagation equations:

$$\underset{6\times1}{a^{[1]}} = \sigma(\underset{6\times4}{\mathbf{W}^{[\mathbf{1}]}} \cdot \underset{4\times1}{a^{[0]}} + \underset{6\times1}{b^{[1]}})$$
$$\underset{4\times1}{a^{[2]}} = \sigma(\underset{4\times6}{\mathbf{W}^{[\mathbf{2}]}} \cdot \underset{6\times1}{a^{[1]}} + \underset{4\times1}{b^{[2]}})$$
$$\underset{3\times1}{a^{[3]}} = \sigma(\underset{3\times4}{\mathbf{W}^{[\mathbf{3}]}} \cdot \underset{4\times1}{a^{[2]}} + \underset{3\times1}{b^{[3]}})$$

6. A model experiences underfitting when it is too simple to capture the underlying patterns in the data. This typically happens when the model has insufficient complexity (too few parameters or layers in a neural network), leading to poor performance on both the training and testing datasets. The most important indicator for underfitting is high bias, and high bias occurs when both train loss and test loss are significantly higher than the Bayes error. Overfitting, on the other hand, happens when the model becomes too complex and learns not only the underlying patterns but also the noise or irrelevant details (outliers) in the training data. This leads to very good performance on the training set but poor generalization to new, unseen data. The most important indicator for overfitting is high variance, where the test loss is significantly higher than the train loss compared to the Bayes error. To evaluate these situations, one can look at the training and validation loss curves during training. In underfitting, both training and validation losses will remain high. In overfitting, training loss will continue to decrease while validation loss increases after a certain point. Regularization techniques such as early stopping or dropout are used to prevent overfitting.

7. **Learning Rate** controls how much the weights of the model are changed per step. Too high a learning rate means parameters will vary greatly and it might make the model wander around and not converge to a minima. Too low learning rate means that the model will update slowly and might get stuck in a local minima, not learning even after long training processes.

8. **Dropout** is a regularization method that will provide robustness to the model by turning some neurons off in training time. Prevents overfitting via making the model not rely on a specific set of neurons that will learn a set of features. Mostly used as a random dropout that will take $\boldsymbol{p}$ as a float parameter to decide which percentage of the last layer to exclude from the current training step.

9. **Weights** are internal to the model and are learned by training processes. **Hyperparameters** are external settings that get defined before the runtime of the model starts. Learning rate, batch size and loss function are some general hyperparameters.

10. Before each update on parameters:
    - using the whole dataset is **Batch Gradient Descent**
    - using one sample of the dataset is **Stochastic Gradient Descent**
    - using a pre-determined number of samples is **Mini-Batch Gradient Descent**

11. In basic gradient descent, parameter update is $\omega := \omega - \eta \cdot \nabla_\omega J(\omega)$
    - ω is the parameter of the model.
    - η is the learning rate — step size.
    - $\nabla_\omega J(\omega)$ is the gradient of the loss function w.r.t. the parameter.

    Different to this mechanism, **RMSProp** uses something called the *adaptive optimization* technique via adjusting the learning rate in consideration of recent gradients. This allows dealing with exploding and vanishing gradients a little bit.
    Update rule definition for RMSProp:
    - $g_t = \nabla_\omega J(\omega)_t$
    - $E[g^2]_t = \gamma \cdot E[g^2]_{t-1} + (1-\gamma) \cdot g_t^2$
    - $\omega_{t+1} := \omega_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t$

    typical values for γ is 0.9 and ϵ is 1.e-8

    In the case with **Adam**, it also uses an *adaptive optimization* technique but compines what is defined in RMSProp and SGD with momentum.
    Update rule definition for Adam:
    - $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$
    - $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$
    - $\omega_t = w_{t-1} - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$
    - $g_t = \nabla_\omega J(\omega)_t$
    - $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$
    - $v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$

    where β_1 is the first-moment estimate with a typical value of 0.9,
    β_2 is the second-moment estimate with a typical value of 0.999
    ϵ is exactly the same in RMSProp avoiding zero division.

Coding Assignment

Click for the coding assignment

Bonus Questions

1. Assume we have a linear regression model:

$$\hat{y} = f_w(x) = w_1 x + w_0$$

for a dataset of pairs (x_i, y_i) where $1 \leq i \leq N$.

The residuals of the model for each pair are given by:

$$e_i = y_i - \hat{y}_i = y_i - (w_1 x_i + w_0)$$

The residual sum of squares is defined as:

$$RSS = \sum_{i=1}^{N} e_i^2 = \sum_{i=1}^{N} (y_i - (w_1 x_i + w_0))^2$$

Our goal is to minimize the RSS by finding the best w_1 and w_0.

To minimize the RSS, we take partial derivatives with respect to w_1 and w_0 and set them to zero.

$$\frac{\partial RSS}{\partial w_1} = -2 \sum_{i=1}^{N} x_i \left(y_i - (w_1 x_i + w_0)\right) = 0$$

Simplifying:

$$\sum_{i=1}^{N} x_i y_i = w_1 \sum_{i=1}^{N} x_i^2 + w_0 \sum_{i=1}^{N} x_i$$

$$\frac{\partial RSS}{\partial w_0} = -2 \sum_{i=1}^{N} \left(y_i - (w_1 x_i + w_0)\right) = 0$$

Simplifying:

$$\sum_{i=1}^{N} y_i = w_1 \sum_{i=1}^{N} x_i + N w_0$$

We now have the following system of equations:
1. $\sum_{i=1}^{N} x_i y_i = w_1 \sum_{i=1}^{N} x_i^2 + w_0 \sum_{i=1}^{N} x_i$ 2. $\sum_{i=1}^{N} y_i = w_1 \sum_{i=1}^{N} x_i + N w_0$
First, solve for w_0 using the second equation:

$$w_0 = \frac{1}{N} \sum_{i=1}^{N} y_i - w_1 \frac{1}{N} \sum_{i=1}^{N} x_i = \bar{y} - w_1 \bar{x}$$

where $\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$ and $\bar{y} = \frac{1}{N} \sum_{i=1}^{N} y_i$ are the means of x and y.

Next, substitute this expression for w_0 into the first equation to solve for w_1:

$$\sum_{i=1}^{N} x_i y_i = w_1 \sum_{i=1}^{N} x_i^2 + (\bar{y} - w_1 \bar{x}) \sum_{i=1}^{N} x_i$$

After simplification, this yields the following expression for w_1:

$$w_1 = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

Finally, substitute w_1 back into the equation for w_0:

$$w_0 = \bar{y} - w_1 \bar{x}$$

Thus, the estimated weights are given by:

$$w_1 = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

$$w_0 = \bar{y} - w_1 \bar{x}$$

where $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$ and $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$ are the means of the samples.

2. Additional notation: $z^{[1]}$: Linear combination before the activation function at the first hidden layer (size 4x1), $z^{[2]}$: Linear combination before the activation function at the output layer (size 1x1)

1. Forward Propagation

$$z^{[1]} = W^{[1]} X + b^{[1]} = W^{[1]} A^{[0]} + b^{[1]}$$

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

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$

$$\hat{y} = a^{[2]} = \sigma(z^{[2]}) = \frac{1}{1 + e^{-z^{[2]}}}$$

$$L(y, \hat{y}) = -\left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right]$$

2. Backward Propagation

$$\frac{\partial L}{\partial \hat{y}} = -\left(\frac{y}{\hat{y}} - \frac{1 - y}{1 - \hat{y}}\right) = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})}$$

Since $\frac{\partial \hat{y}}{\partial z^{[2]}} = \hat{y}(1 - \hat{y})$, the gradient with respect to $z^{[2]}$ (backpropagating through the sigmoid function) is:

$$\frac{\partial L}{\partial z^{[2]}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z^{[2]}} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})} \cdot \hat{y}(1 - \hat{y})$$

So we get:

$$\frac{\partial L}{\partial z^{[2]}} = \hat{y} - y$$

Therefore, when computing the gradient with respect to $W^{[2]}$ and $b^{[2]}$:

$$\frac{\partial L}{\partial W^{[2]}} = \frac{\partial L}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial W^{[2]}} = (\hat{y} - y) \cdot a^{[1]}$$

$$\frac{\partial L}{\partial b^{[2]}} = \frac{\partial L}{\partial z^{[2]}} = \hat{y} - y$$

Similarly, backpropagating to the hidden layer by computing the gradient with respect to $a^{[1]}$ gives us:

$$\frac{\partial L}{\partial a^{[1]}} = \frac{\partial L}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a^{[1]}} = (\hat{y} - y) \cdot W^{[2]}$$

Computing the gradient with respect to $z^{[1]}$ (backpropagating through the sigmoid function at the hidden layer):

$$\frac{\partial L}{\partial z^{[1]}} = \frac{\partial L}{\partial a^{[1]}} \cdot \frac{\partial a^{[1]}}{\partial z^{[1]}} = \left[(\hat{y} - y) \cdot W^{[2]}\right] \cdot a^{[1]}(1 - a^{[1]})$$

Finally computing the gradient with respect to $W^{[1]}$ and $b^{[1]}$:

$$\frac{\partial L}{\partial W^{[1]}} = \frac{\partial L}{\partial z^{[1]}} \cdot \frac{\partial z^{[1]}}{\partial W^{[1]}} = \left[(\hat{y} - y) \cdot W^{[2]}\right] \cdot a^{[1]}(1 - a^{[1]}) \cdot X$$

$$\frac{\partial L}{\partial b^{[1]}} = \frac{\partial L}{\partial z^{[1]}} = \left[(\hat{y} - y) \cdot W^{[2]}\right] \cdot a^{[1]}(1 - a^{[1]})$$