Deep Learning. Assignment 1

Nico Sharei (nsharei), Artem Bisliouk(abisliou)

April 7, 2024

1 Implement an MLP

1.1 a)

(i) The model parameters consist of the weights w and the bias b. The initialisation is very similar to the Xavier initialisation for neural networks [4], where the parameters are randomly initialised and scaled by the square root of a factor n representing the length of the respective vector or matrix to which it is applied [4]. This is done to avoid generating parameters that are too large or too small, as this can lead to either the exploding or vanishing gradient problem.

After initialisation, the parameters are registered using the *register_parameter* function provided by Pytorch. It serves as an indication to Pytorch that these are the parameters that should be optimised during the training process.

(ii) The model's forward pass is executed by applying the input x, the weights w and the bias b to the following formula.

$$W^T x + b$$

. The values obtained indicate the model's raw prediction for each class. Although these values do not represent probabilities, the class with the highest logit can be considered the most probable. To obtain probabilities, the logits z are transformed into the interval $[0, 1]$ by applying the softmax function to each logit.

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

While this procedure leads to normalized outputs from a valid probability distribution, it simultaneously amplifies the largest logit and suppresses the other ones [1]. This is desirable because it enhances models confidence in its predictions, leading to a faster learning process.

(iii) The LogisticRegression Object is initialised to create an instance of the model. The number of inputs D and number of classes C are passed to the model as parameters and then processed inside its constructor. A prediction can be done by calling the object with $||D||$ one-dimensional datapoints.

2 Experiment with MLPs

2.1 a)

GGiven the universal approximation theorem [2], it can be assumed that as the size of the neurons within a hidden layer increases, the precision of the approximation to the function of the training data will increase congruently. While overfitting may occur above a certain threshold, it is not expected to occur in the small range of 0 to 3 hidden neurons.

0. An MLP without hidden neurons is a linear model. Since the data distribution is sinusoidal, it won't be able to capture the patterns without some element of non-linearity.

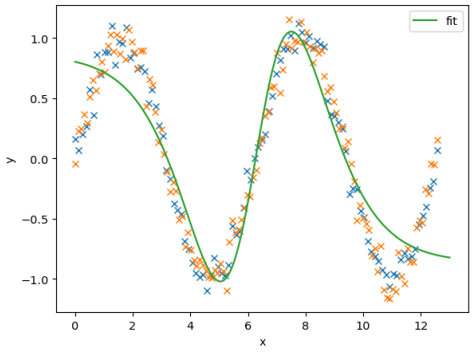
1. One hidden neuron allows for non-linearity and should be able to follow the pattern of the data for a short interval, but will probably need more complexity to accurately capture the variations in amplitude and frequency.

2. A second hidden neuron can lead to a great improvement in capturing the variations in amplitude and frequency. However, the details may not be captured accurately.

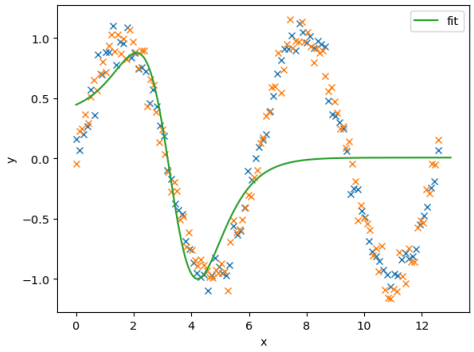
3. Three hidden neurons add even more flexibility to the network and may be able to capture the details as well. According to Hanin and Selke's hypothesis [5], any function can be approximated by an MLP with a sufficient number of hidden layers of size $D + 1$. Although there is only one hidden layer, the constraint of enough hidden neurons is satisfied with two input neurons D and three hidden neurons.

2.2 b)

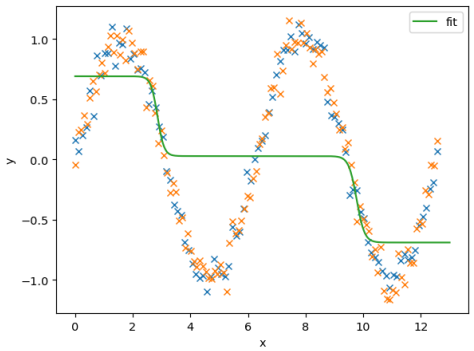
The repeated training of the MLP results in several different function approximations. The most common ones are plotted together with their training error in figure 7. The drastic difference in the accuracy of the approximation is due to the random initialisation of the parameters. This initial state of the parameters is fully responsible for whether the model converges to a good or a bad optimum. The model in figure 7(a) seems to have found the global optimum, while the other models show local optima ranging from passable to very bad. In particular, figure 1(f) shows a function that goes into linearity after crossing the interval $x = [0, 0.1]$. It is only slightly better than a MLP without hidden units, which has an accuracy of 0.50623.



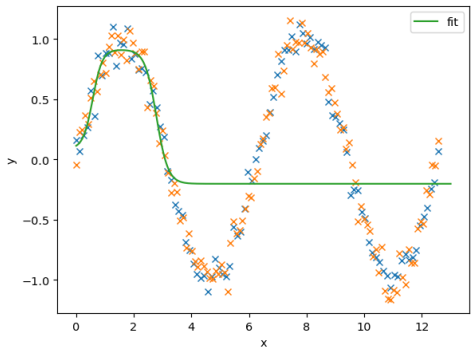
(a) Training error: 0.079572



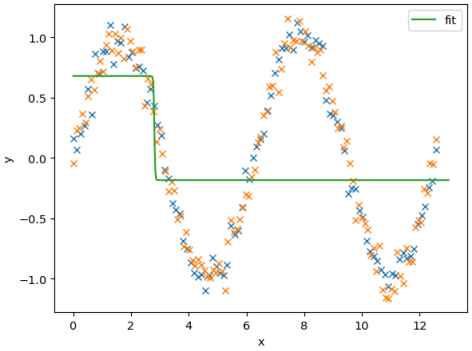
(b) Training error: 0.286909



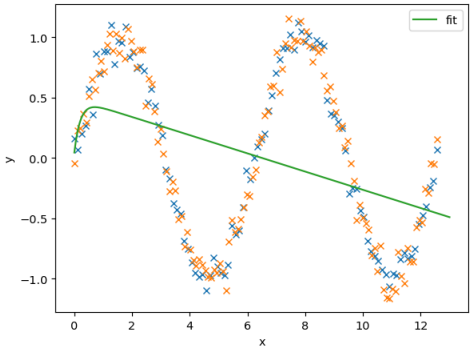
(c) Training error: 0.302086



(d) Training error: 0.357250



(e) Training error: 0.376615



(f) Training error: 0.436065

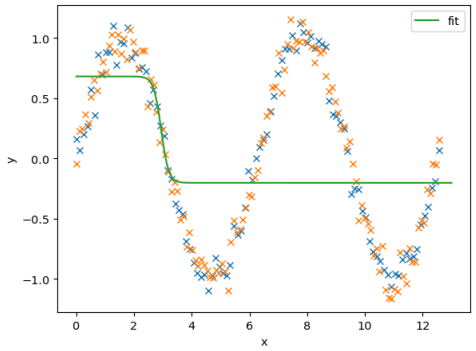
Figure 1: Multiple MLPs with two hidden neurons

2.3 c)

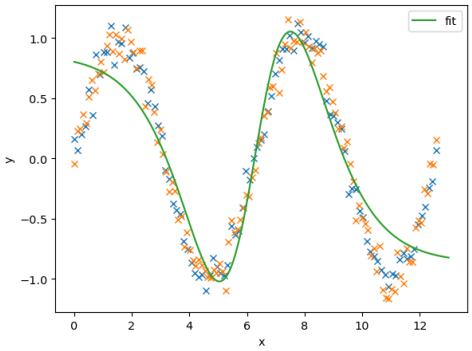
When comparing the models depicted in figures 2(a), 2(b), 2(c) and 2(d), it is evident that an increase in hidden units leads to a more precise fit to the training data. While this is desirable, it is also observed that the function resultingfrom a MLP with 10 Neurons (2(d)) is showing signs of overfitting in the interval $[8, 9]$. Although its training error is slightly better than the training error of the MLP with only 3 neurons (2(c)), it performs worse on the test set.

This phenomenon is even more visible in figures 2(e) and 2(f). Both functions strongly overfit on the training set by providing an even more precise training error but a significantly worse test error. This drastic difference is mainly due to the MLP's function estimation in the intervals $[0, 1]$

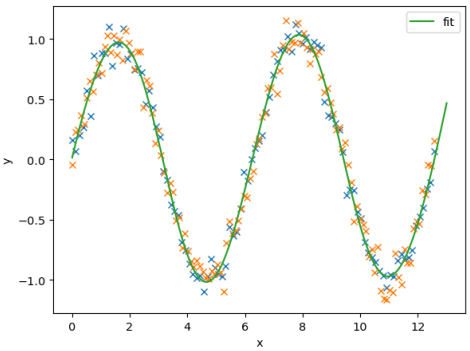
(50 neurons) and $[0, 2]$ (100 neurons). While their predictions perfectly match the data points of the training set, they mispredict the test data points with a high offset.



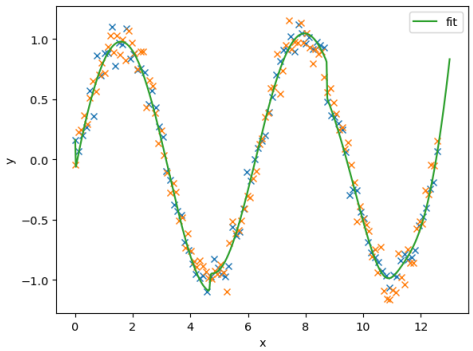
Hidden state with 1 neuron
(a) Training error: 0.372918
Test error: 0.374316



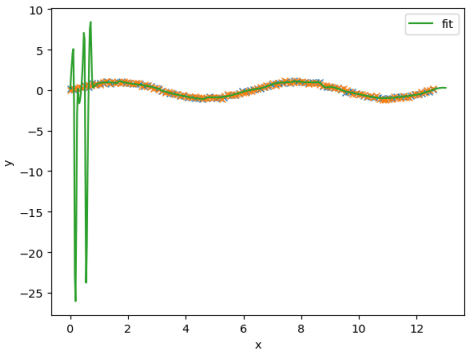
Hidden state with 2 neurons
(b) Training error: 0.079569
Test error: 0.086713



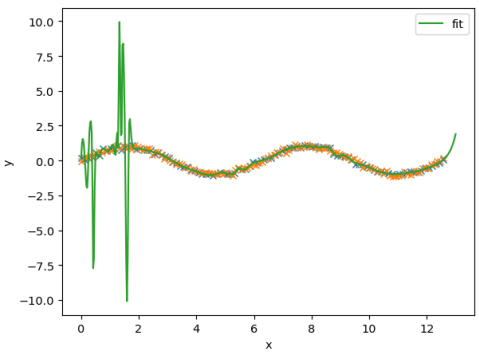
Hidden state with 3 neuron
(c) Training error: 0.0073241
Test error: 0.0103352



Hidden state with 10 neuron
(d) Training error: 0.0061324
Test error: 0.0150370



Hidden state with 50 neuron
(e) Training error: 0.0033172
Test error: 3.559721

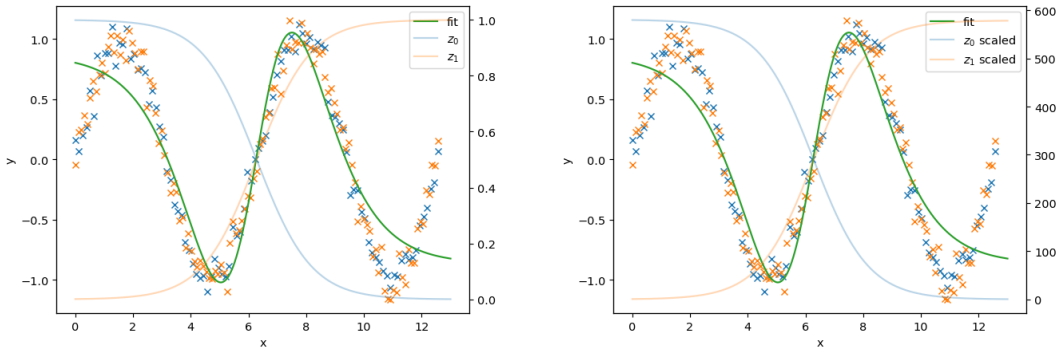


Hidden state with 100 neuron
(f) Training error: 0.0021686
Test error: 1.7257210

Figure 2: MLPs with a varying quantity in hidden neurons

2.4 d)

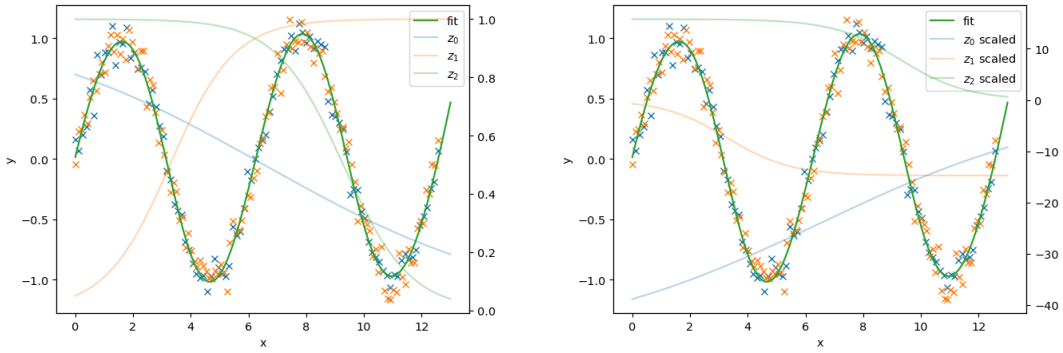
The function approximations along a visualisation of their hidden states is provided in figures 3, 4 and 5. Each hidden neuron aims on learning a certain feature from the data distribution. Collectively, these neurons result into an embedding which aims on describing the data well. When comparing figure 3 and 4 with figure 5, it becomes coherent that a higher quantity of hidden neurons results in a more flexible function approximation, allowing the model to capture more and more detailed patterns. However, the extremely large number of neurons that fulfill the role of detecting such patterns inevitably leads to offerfitting as can be observed in the case of 10 neurons (figure 5).



(a) Two hidden neurons (not scaled)

(b) Two hidden neurons (scaled)

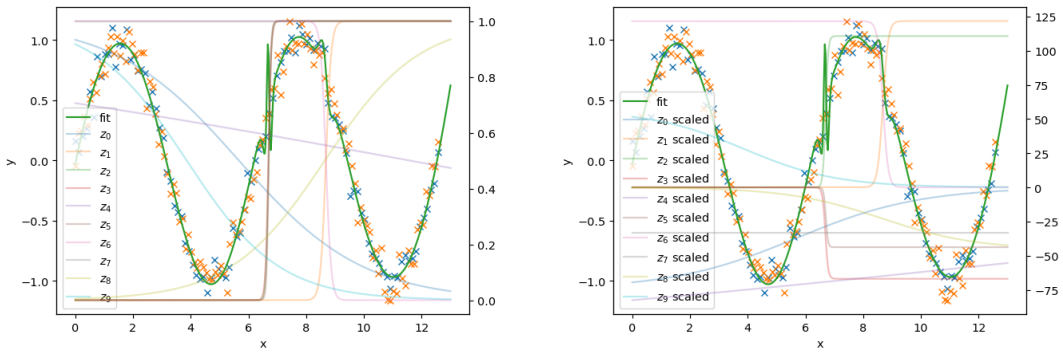
Figure 3: MLP hidden states with two neurons



(a) Three hidden neurons (not scaled)

(b) Three hidden neurons (scaled)

Figure 4: MLP hidden states with three neurons



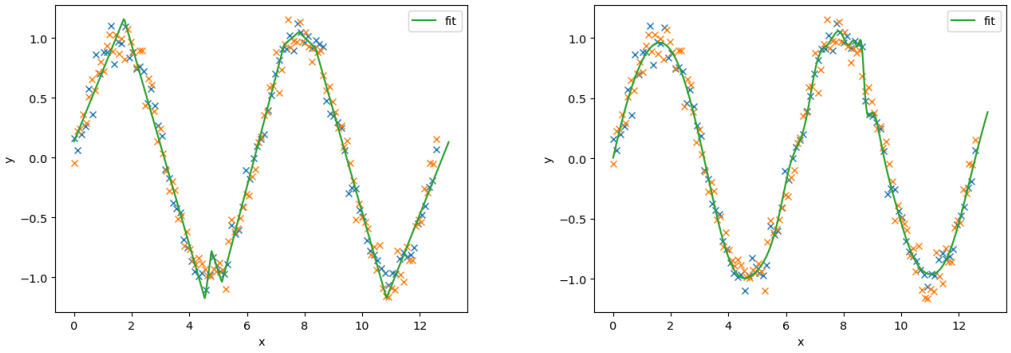
(a) Ten hidden neurons (not scaled)

(b) Ten hidden neurons (scaled)

Figure 5: MLP hidden states with ten neurons

2.5 e)

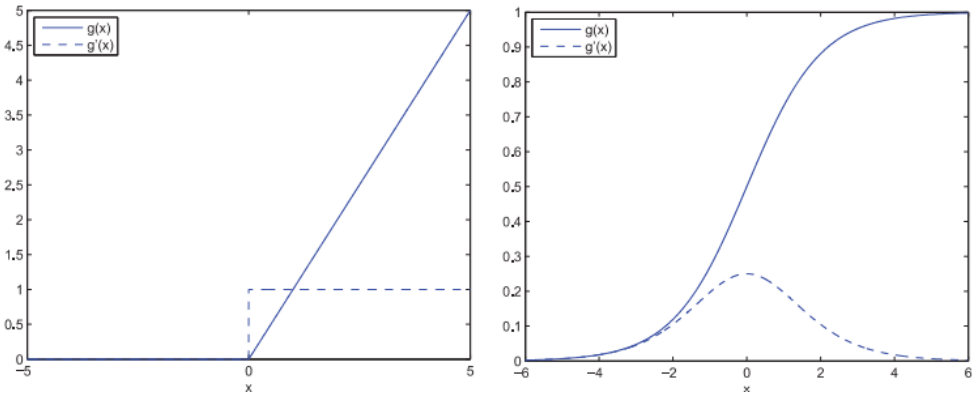
Figure 6(a) shows an MLP with three hidden layers consisting of 3, 5 and 4 ReLU units. Compared to the same architecture with sigmoid units (figure 6(b)), the function with ReLU units approximates the function in a much more linear way. This can be understood by looking at the two activation functions shown in figure 6(a) and 6(b). While the ReLU activation function is completely linear for positive values, the sigmoid activation function transitions smoothly between 0 and 1, leading to more non-linear approximations. Although both activation functions have a fairly similar test error, it appears that the sigmoid activation function finds a better fit to the shape of the data as its difference in training and test error is not as drastic as the ReLU activation function. The sigmoid activation function appears to generalise better to sinusoidal data because it can adapt much more easily to gradual changes in the function being approximated.



(a) MLP with ReLU hidden units
Training error: 0.005603
Test error: 0.012087

(b) MLP with Sigmoid hidden units
Training error: 0.010578
Test error: 0.013780

Figure 6: MLP with 3, 5 and 4 ReLU units in hidden layers



(a) ReLU activation function (b) Sigmoid activation function

Figure 7: The ReLU and Sigmoid activation functions [3]

3 Backpropagation

(i) Derivation of the derivative of the loss function with respect to itself:

$$\frac{\partial l}{\partial l} = 1 \quad \text{[The derivative of a function with respect to itself is always 1]}$$

(ii) Derivation of the derivative of the loss function with respect to y:

$$l = (y - \hat{y})^2 \quad \text{[The loss function]}$$
$$\frac{\partial l}{\partial y} = 2 * (y - \hat{y})^{2-1} \quad \text{[Applying the power rule]}$$
$$\frac{\partial l}{\partial y} = 2 * (y - \hat{y}) * \frac{\partial (y - \hat{y})}{\partial y} \quad \text{[Applying the chain rule]}$$
$$\frac{\partial l}{\partial y} = 2 * (y - \hat{y}) \quad \text{[The derivative of } (y - \hat{y}) \text{ with respect to } y \text{ is 1]}$$

(iii) Derivation of the derivative of the loss function with respect to $\hat{y}$:

$$l = (y - \hat{y})^2 \quad \text{[The loss function]}$$
$$\frac{\partial l}{\partial \hat{y}} = 2 * (y - \hat{y})^{2-1} \quad \text{[Applying the power rule]}$$
$$\frac{\partial l}{\partial \hat{y}} = 2 * (y - \hat{y}) * \frac{\partial (y - \hat{y})}{\partial \hat{y}} \quad \text{[Applying the chain rule]}$$
$$\frac{\partial l}{\partial \hat{y}} = -2 * (y - \hat{y}) \quad \text{[The derivative of } (y - \hat{y}) \text{ with respect to } \hat{y} \text{ is -1]}$$

(iv) Derivation of the derivative of the loss function with respect to b2:

$$\frac{\partial l}{\partial b2} = \frac{\partial l}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial b2} \quad \text{[Applying the chain rule]}$$
$$\frac{\partial \hat{y}}{\partial b2} = 1 \quad \text{[Because } \hat{y} = z4 + b2\text{]}$$
$$\frac{\partial l}{\partial b2} = \frac{\partial l}{\partial \hat{y}} \quad \text{[Substituting } \frac{\partial \hat{y}}{\partial b2} \text{ into the equation]}$$

(v) Derivation of the derivative of the loss function with respect to z4:

$$\frac{\partial l}{\partial z4} = \frac{\partial l}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z4} \quad \text{[Applying the chain rule]}$$
$$\frac{\partial \hat{y}}{\partial z4} = 1 \quad \text{[Because } \hat{y} = z4 + b2\text{]}$$
$$\frac{\partial l}{\partial z4} = \frac{\partial l}{\partial \hat{y}} \quad \text{[Substituting } \frac{\partial \hat{y}}{\partial z4} \text{ into the equation]}$$

(vi) Derivation of the derivative of the loss function with respect to W2:

$$\frac{\partial l}{\partial W2} = \frac{\partial l}{\partial z4} * \frac{\partial z4}{\partial W2} \quad \text{[Applying the chain rule]}$$
$$\frac{\partial z4}{\partial W2} = z3 \quad \text{[Because } z4 = z3 \cdot W2\text{]}$$
$$\frac{\partial l}{\partial W2} = \frac{\partial l}{\partial z4} * z3 \quad \text{[Substituting } \frac{\partial z4}{\partial W2} \text{ into the equation]}$$

(vii) Derivation of the derivative of the loss function with respect to z3:

$$\frac{\partial l}{\partial z3} = \frac{\partial l}{\partial z4} * \frac{\partial z4}{\partial z3} \quad \text{[Applying the chain rule]}$$
$$\frac{\partial z4}{\partial z3} = W2^T \quad \text{[Because } z4 = W2^T \cdot z3\text{]}$$
$$\frac{\partial l}{\partial z3} = \frac{\partial l}{\partial z4} * W2^T \quad \text{[Substituting } \frac{\partial z4}{\partial z3} \text{ into the equation]}$$

(viii) Derivation of the derivative of the loss function with respect to z2:

$$\frac{\partial l}{\partial z2} = \frac{\partial l}{\partial z3} * \frac{\partial z3}{\partial z2} \quad \text{[Applying the chain rule]}$$
$$\frac{\partial z3}{\partial z2} = \sigma'(z2) \quad \text{[Because } z3 = \sigma(z2)\text{]}$$
$$\frac{\partial l}{\partial z2} = \frac{\partial l}{\partial z3} * \sigma'(z2) \quad \text{[Substituting } \frac{\partial z3}{\partial z2} \text{ into the equation]}$$

(ix) Derivation of the derivative of the loss function with respect to b1:

$$\frac{\partial l}{\partial b1} = \frac{\partial l}{\partial z2} * \frac{\partial z2}{\partial b1} \quad \text{[Applying the chain rule]}$$
$$\frac{\partial z2}{\partial b1} = 1 \quad \text{[Because } z2 = z1 + b1\text{]}$$
$$\frac{\partial l}{\partial b1} = \frac{\partial l}{\partial z2} \quad \text{[Substituting } \frac{\partial z2}{\partial b1} \text{ into the equation]}$$

(x) Derivation of the derivative of the loss function with respect to z1:

$$\frac{\partial l}{\partial z1} = \frac{\partial l}{\partial z2} * \frac{\partial z2}{\partial z1} \quad \text{[Applying the chain rule]}$$
$$\frac{\partial z2}{\partial z1} = 1 \quad \text{[Because } z2 = z1 + b1\text{]}$$
$$\frac{\partial l}{\partial z1} = \frac{\partial l}{\partial z2} \quad \text{[Substituting } \frac{\partial z2}{\partial z1} \text{ into the equation]}$$

(xi) Derivation of the derivative of the loss function with respect to W1:

$$\frac{\partial l}{\partial W1} = \frac{\partial l}{\partial z1} * \frac{\partial z1}{\partial W1} \quad \text{[Applying the chain rule]}$$
$$\frac{\partial z1}{\partial W1} = x \quad \text{[Because } z1 = x \cdot W1\text{]}$$
$$\frac{\partial l}{\partial W1} = \frac{\partial l}{\partial z1} * x \quad \text{[Substituting } \frac{\partial z1}{\partial W1} \text{ into the equation]}$$

(xii) Derivation of the derivative of the loss function with respect to x:

$$\frac{\partial l}{\partial x} = \frac{\partial l}{\partial z1} * \frac{\partial z1}{\partial x} \quad \text{[Applying the chain rule]}$$
$$\frac{\partial z1}{\partial x} = W1^T \quad \text{[Because } z1 = W1^T \cdot x\text{]}$$
$$\frac{\partial l}{\partial x} = \frac{\partial l}{\partial z1} * W1^T \quad \text{[Substituting } \frac{\partial z1}{\partial x} \text{ into the equation]}$$

References

[1] John Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. *Advances in neural information processing systems*, 2, 1989.

[2] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[3] Bin Ding, Huimin Qian, and Jun Zhou. Activation functions and their characteristics in deep neural networks. In *2018 Chinese control and decision conference (CCDC)*, pages 1836–1841. IEEE, 2018.

[4] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[5] Boris Hanin and Mark Sellke. Approximating continuous functions by relu nets of minimal width (2018). *arXiv preprint arXiv:1710.11278*.