Deep Learning Assignment 12 in FS 2022

Radial Basis Function Network

Microsoft Forms Document:
https://forms.office.com/r/Zyre1SxphD

Manuel Günther

Distributed: Friday, May 27, 2022
Discussed: Friday, June 3, 2022

In this assignment, we show an alternative learning strategy, which has been forgotten in the last years but might get more popularity in near future. A radial basis function layer has some similarities with a fully-connected layer, but the outputs of the layer are not determined by a dot product between the weights and the input, but rather it computes a distance. Unfortunately, `PyTorch` does not provide us with an implementation of an RBF layer, so we need to implement our own. Additionally, the activation function is replaced by a function that creates higher outputs for small absolute inputs, such as a Gaussian distribution:

$$a_r = \|\vec{w}_r - \vec{x}\|^2 \tag{1}$$

$$h_r = e^{-\frac{a_r^2}{2\sigma_r^2}} \tag{2}$$

Finally, we want to combine the convolutional network from Assignment 8 with an RBF layer and a final fully-connected layer to compute the 10 outputs. For simplicity, let us define K to be the input dimension of our RBF layer, R to be the number of basis functions (the number of outputs) of our RBF layer, and $O = 10$ the number of outputs (logits) of our network. The complete network topology is given in Table 1. We are both interested in the deep feature representation that is output of the first fully-connected layer, and in the logits that are output from the second fully-connected layer.

1. 2D convolutional layer with Q_1 channels, kernel size 5×5, stride 1 and padding 2
2. 2D maximum pooling layer with kernel size 2×2 and stride 2
3. activation function ReLU
4. 2D convolutional layer with Q_2 channels, kernel size 5×5, stride 1 and padding 2
5. 2D maximum pooling layer with kernel size 2×2 and stride 2
6. activation function ReLU
7. flatten layer to convert the convolution output into a vector
8. fully-connected layer with the correct number of inputs and K outputs
9. RBF layer with K inputs and R outputs (implemented in Task 2)
10. RBF activation function (implemented in Task 3)
11. fully-connected layer with R inputs and O outputs

Table 1: Network configuration of the radial basis function network

1 Dataset

We will train and test our methods on the MNIST dataset.

Task 1: Dataset Instantiate the training and validation set data loaders for the MNIST dataset. Select appropriate batch sizes and parameters.

2 Radial Basis Function

We will split our implementations into three different parts. First, we implement the output a_r as provided in (1). Second, we implement the activation function according to (2). Finally, we build our network by inserting an RBF layer and in between the fully-connected layers of our default network.

In order to implement the output a_k of the RBF layer, we will use a weight matrix $\mathbf{W} \in \mathbb{R}^{R \times K} = [\vec{w}_1, \vec{w}_2, \ldots, \vec{w}_R]$ where each vector is of dimension K. When handling batches $\mathbf{X} \in \mathbb{R}^{B \times K}$, we require to compute our activations as:

$$\mathbf{A} \in \mathbb{R}^{B \times R} = (a_{n,r}) \qquad \text{with} \qquad a_{n,r} = \left\| \vec{w}_r - \vec{x}^{[n]} \right\|^2 = \sum_{k=1}^{K} \left\| w_{r,k} - \vec{x}_r^{[n]} \right\|^2 \tag{3}$$

In order to speed up processing and enabling the use of `tensor` operations, we need to bring $\mathbf{W}$ and $\mathbf{X}$ to the same size $\mathbb{R}^{B \times R \times K}$ by logically (not physically!) copying the data and the weights to $\mathcal{W}$ and $\mathcal{X}$. Then, the resulting $\mathcal{A} = \|\mathcal{W} - \mathcal{X}\|$ needs to be summed up over dimension K to arrive at $\mathbf{A}$ as given in (3).

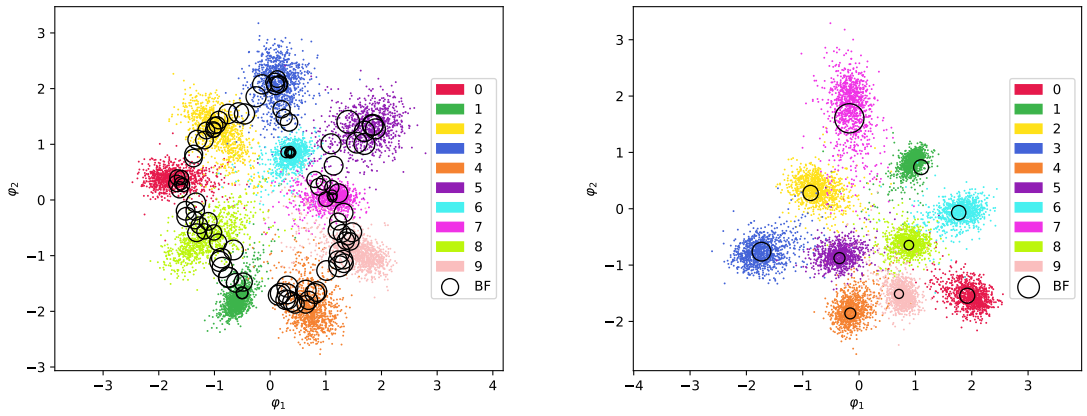
Task 2: Radial Basis Function Layer Implement a layer in PyTorch that computes the activation $\mathbf{A}$ of a radial basis function. Derive your layer from `torch.nn.Module`. In the `__init__` function, instantiate the weight matrix $\mathbf{W}$ as a `torch.nn.Parameter`, and initialize the weight values randomly from values in range $[-2, 2]$. In `forward`, compute and return the activation from the stored weight matrix and the given input batch as indicated above.

Task 3: Radial Basis Function Activation Implement a layer in PyTorch that computes the RBF activation as given in (2). Derive your layer from `torch.nn.Module`. In the `__init__` function, instantiate the `sigma2` parameter as a `torch.nn.Parameter`, and initialize them with 1. For simplicity, we can define `sigma2 = 2*`σ^2, in order to avoid computing the square of the σ each time.

Test 1: RBF Layer and Activation Instantiate an RBF layer from task 2 and an activation function from task 3, for $K = 4$ and $R = 10$. Generate a batch $\mathcal{X}$ with batch size of $B = 12$. Forward this batch through the layer and the activation function. Make sure that the output is of the desired shape.

Task 4: Radial Basis Function Network Implement a network in PyTorch that implements the radial basis function network according to Table 1. In `forward`, return both the logits from the second fully-connected layer, and the deep feature representation after the first fully-connected layer, which we will use later for visualization purposes.

Task 5: Training and Validation Loop Instantiate the RBF network from Task 4 with $Q_1 = 32$, $Q_2 = 64$, $K = 2$, $R = 100$ and $O = 10$. Instantiate an optimizer of your choice with a suitable learning rate, as well as an appropriate loss function. Run the training for 20 epochs on the training set. After each epoch, compute and report the validation set accuracy.



(a) $R = 100$ (b) $R = 10$

Figure 1: Exemplary plots of learned deep feature representations and basis functions. Note that both plots were generated without bias neurons in the final fully-connected layer.

3 Visualization

We have selected the dimensionality of the deep features to be $K = 2$ in order to be able to visualize them. The goal is to extract the deep feature representations of all validation samples. For each sample, we regard the 2D representation as a point in a 2D space, which we can mark with a color that is dependent on the label of that sample. After plotting a dot for each of the validation samples, we can observe if we find some structure in the feature space.

Finally, also the learned basis functions $\vec{w}_r$ can be plotted, including their according size σ_r. This allows us to see whether the RBF network has adapted itself to the data. Examples can be found in Figure 1.

Task 6: Deep Feature Extraction Iterate through the validation set samples and extract the 2D deep feature representations for the images. Store the results in 10 different lists, one for each target class.

Task 7: Deep Feature Visualization Obtain a list of 10 different colors, one for each digit class. For each of the 10 digits, plot all the samples in a 2D plot in the according color as a dot.

Task 8: Basis Function Visualization Obtain the basis functions that were learned during network training from the layer implemented in Task 2. Obtain the scaling factor σ_r for each of the basis functions that were learned by the activation function layer from Task 3. For each of the basis functions $\vec{w}_r$, draw a black circle with a radius corresponding to the corresponding σ_r. Overlay the plot from Task 7 with these circles. Examples for $R = 10$ and $R = 100$ can be found in Figure 1 – obtaining the shown legend is more difficult and does not need to be replicated.