universität freiburg

Assignment 06

Neural Networks – Part 1

Machine Learning WS23/24

Assignment 06
Solution

1. Activation Functions
2. Neural networks

Activation Functions

Activation Functions
Task 1.1: Non-Linear vs. Linear

Why do we use non-linear activation functions?

- Stacking multiple linear functions will again create a linear function (collapse).
- Example:

2 Layer Linear Network

$$\begin{aligned} h^{(1)} &= W^{(1)} \cdot x + b^{(1)} \\ h^{(2)} &= W^{(2)} \cdot h^{(1)} + b^{(2)} \end{aligned}$$

Equivalent single-layer Network

$$\begin{aligned} h^{(2)} &= W^{(2)} \cdot \left(W^{(1)} \cdot x + b^{(1)}\right) + b^{(2)} \\ &= \left(W^{(2)} W^{(1)}\right) \cdot x + \left(W^{(2)} b^{(1)}\right) + b^{(2)} \\ &= W' \cdot x + b' \end{aligned}$$

Activation Functions

Task 1.2: Computing Gradients

Gradient for ReLU

$$\begin{aligned} \mathrm{ReLU}(z) &= \max(z,0) \\ &= \begin{cases} z & z>0 \\ 0 & z\leq 0 \end{cases} \end{aligned}$$

$$\frac{\partial\,\mathrm{ReLU}(z)}{\partial z} = \begin{cases} 1 & z>0 \\ 0 & z<0 \end{cases}$$

Undefined for z = 0!

Activation Functions

Task 1.2: Computing Gradients

Gradient for Sigmoid

- Start with basic derivation:

$$\sigma(z) = \frac{1}{1+e^{-z}} \qquad \frac{\partial \sigma(z)}{\partial z} = \frac{\partial}{\partial z}\left(\frac{1}{1+e^{-z}}\right) = -1\cdot(1+e^{-z})^{-2}\cdot(-e^{-z}) = \frac{e^{-z}}{(1+e^{-z})^2}$$

- Express terms via σ(z):

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

- Then simplify:

$$\frac{\partial \sigma(z)}{\partial z} = \frac{1-\sigma(z)}{\sigma(z)}\cdot(\sigma(z))^2 = (1-\sigma(z))\cdot\sigma(z)$$

Activation Functions

Task 1.2: Computing Gradients

Gradient for TanH

$$\tanh(z) = \frac{e^{2z}-1}{e^{2z}+1}$$

$$\begin{aligned}
\frac{\partial \tanh(z)}{\partial z} &= \frac{\partial}{\partial z}\left(\frac{e^{2z}-1}{e^{2z}+1}\right) \\
&= \frac{2e^{2z}\cdot(e^{2z}+1)-(e^{2z}-1)\cdot 2e^{2z}}{(e^{2z}+1)^2} \\
&= \frac{2e^{4z}+2e^{2z}-2e^{4z}+2e^{2z}}{(e^{2z}+1)^2} \\
&= \frac{e^{4z}+2e^{2z}+1-e^{4z}+2e^{2z}-1}{(e^{2z}+1)^2} \\
&= \frac{(e^{4z}+2e^{2z}+1)-(e^{4z}-2e^{2z}+1)}{(e^{2z}+1)^2} \\
&= \frac{(e^{2z}+1)^2-(e^{2z}-1)^2}{(e^{2z}+1)^2} = 1-\tanh^2(z)
\end{aligned}$$

apply quotient rule

multiply out

subtract e^{4z} and expand by +/-1

separate

apply square rule

Activation Functions
Task 1.3: Advantages and Disadvantages

Advantages and disadvantages of the ReLU activation function

- Fixes vanishing gradient problem
- Computationally inexpensive
- Fragile during training, can “die”
    - Can be overcome with Leaky ReLU, Parametric ReLU

Activation Functions

Task 1.3: Advantages and Disadvantages

Advantages and disadvantages of the Sigmoid activation function

- Computationally more expensive than ReLU
- Not zero-centered
- Has a smooth gradient
- Well-suited for binary classification when used on the output layer (cf. Logistic regression)
- Suffers from vanishing gradient problems

Activation Functions

Task 1.3: Advantages and Disadvantages

Advantages and disadvantages of the TanH activation function

- Computationally more expensive than ReLU
- Is zero-centered
- Has a smooth gradient
- Used more frequently compared to the sigmoid as an activation function in hidden layers.
    - Special use in recurrent networks (LSTM, GRU)
- Suffers from vanishing gradient problems

Activation Functions
Task 1.4: Softmax Predictions

Given a single data point x with:

true label	possible labels	unnormalized log probability
$y = A$	$Y = \{A, B, C\}$	$z_A = 7, \quad z_B = 1, \quad z_C = 2.2$

Compute softmax output:

$$P(y = i \mid x) \approx softmax(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

$$A: \; softmax(z_A) = \frac{e^{z_A}}{e^{z_A} + e^{z_B} + e^{z_C}} \approx \frac{1097}{1097 + 2.72 + 9.03} \approx 98.9\,\%$$

$$B: \; softmax(z_B) = \frac{e^{z_B}}{e^{z_A} + e^{z_B} + e^{z_C}} \approx \frac{2.72}{1097 + 2.72 + 9.03} \approx 0.25\,\%$$

$$C: \; softmax(z_C) = \frac{e^{z_C}}{e^{z_A} + e^{z_B} + e^{z_C}} \approx \frac{9.03}{1097 + 2.72 + 9.03} \approx 0.81\,\%$$

Activation Functions
Task 1.4: Softmax Predictions

Given a single data point x with:

true label	possible labels	unnormalized log probability
$y=A$	$Y=\{A,B,C\}$	$z_A=7,\quad z_B=1,\quad z_C=2.2$

Predicted class:

$A:\ \ softmax(z_A)\ \approx\ 98.9\%$

$B:\ \ softmax(z_B)\ \approx\ 0.25\%$

$C:\ \ softmax(z_C)\ \approx\ 0.81\%$

The predicted class is the class with the highest softmax score
→ predicted class is A

Activation Functions
Task 1.4: Softmax Predictions

Given a single data point x with:

true label	possible labels	unnormalized log probability
$y = A$	$Y = \{A, B, C\}$	$z_A = 7, \quad z_B = 1, \quad z_C = 2.2$

Log-likelihood loss:

$$A: \; P(y = A \mid x) \approx 98.9\%$$

$$B: \; P(y = B \mid x) \approx 0.25\%$$

$$C: \; P(y = C \mid x) \approx 0.81\%$$

$$\begin{aligned} J(w, x) &= -\sum_{i \in Y} 1_{y_i = i} \cdot \log P(y_i = i \mid x) \\ &= -1 \cdot \log P(y_i = A \mid x) \\ &\approx -1 \cdot \log 0.989 \\ &\approx 0.011 \end{aligned}$$

Neural Networks

Neural Networks

Task 2.1: Basis Functions and Neural Networks

What is the difference between using basis functions and neural networks on in put features?

- Basis functions are difficult to design by hand
  - Especially non-linear ones
- Neural networks can be thought of as learning basis functions!
  - Automatic, adapt to the task at hand to achieve optimal performance

Neural Networks

Task 2.2: Compute the Number of Weights in a Neural Network

Given:

- 80 examples
- 6 features
- Labels: {Spring, Summer, Fall, Winter}
- 3 hidden layers, 32 units each

Translate to relevant information:

- 6 input units
- 32 hidden units, 3 hidden layers
- 4 output units

Compute weights per Layer:

$$\#\text{inputs} \cdot \#\text{neurons} + \#\text{neurons}$$
$$\#\text{multiplicative} + \#\text{biases}$$

- 1st layer: $6 \cdot 32 + 32 = 224$
- 2nd layer: $32 \cdot 32 + 32 = 1056$
- 3rd layer: $32 \cdot 32 + 32 = 1056$
- Output layer: $32 \cdot 4 + 4 = 132$

Compute total number of weights:

- Sum up over layers: 2468 weights

Neural Networks

Task 2.3: Manual Optimization

Dataset:

x	y
1	0
2	0.3
10	1

Network:

$$h^{(1)} = \mathrm{ReLU}\left(w^{(1)}_{1,1} \cdot x\right) \qquad w^{(1)}_{1,1} = -0.1$$

$$h^{(2)} = \mathrm{ReLU}\left(w^{(2)}_{1,1} \cdot h^{(1)}\right) \qquad w^{(2)}_{1,1} = 0.1$$

Loss function:

$$J(w) = \frac{1}{N} \sum_{(x,y)\in D} \left(\hat{y}(x) - y\right)^2$$

Should we increase or decrease $w^{(1)}_{1,1}$?

- All values x in our dataset are positive
- $w^{(1)}_{1,1}$ is negative

➡ ReLU for $h^{(1)}$ will always output zero

➡ Network will always output zero

➡ Must change $w^{(1)}_{1,1}$ to be positive, e.g.:

  - Current loss is approx. 0.36
  - Set $w^{(1)}_{1,1} = 1$ and loss is approx. 0.02

Neural Networks
Task 2.3: Manual Optimization

Dataset:

x	y
1	0
2	0.3
10	1

Network:

$$h^{(1)} = \mathrm{ReLU}\left(w_{1,1}^{(1)} \cdot x\right) \qquad w_{1,1}^{(1)} = -0.1$$
$$h^{(2)} = \mathrm{ReLU}\left(w_{1,1}^{(2)} \cdot h^{(1)}\right) \qquad w_{1,1}^{(2)} = 0.1$$

Loss function:

$$J(w) = \frac{1}{N} \sum_{(x,y) \in D} \left(\hat{y}(x) - y\right)^2$$

Bonus: Compute gradient for $w^{(1)}_{1,1}$

- Expand

$$h^{(1)} = \mathrm{ReLU}\left(a^{(1)}\right), \qquad a^{(1)} = w_{1,1}^{(1)} \cdot x$$

- Apply chain-rule

$$\frac{\partial J(w)}{\partial w_{1,1}^{(1)}} = \frac{\partial J(w)}{\partial h^{(2)}} \cdot \frac{\partial h^{(2)}}{\partial h^{(1)}} \cdot \frac{\partial h^{(1)}}{\partial w_{1,1}^{(1)}} = \frac{\partial J(w)}{\partial h^{(2)}} \cdot \frac{\partial h^{(2)}}{\partial h^{(1)}} \cdot \frac{\partial h^{(1)}}{\partial a^{(1)}} \cdot \frac{\partial a^{(1)}}{\partial w_{1,1}^{(1)}}$$

- Note: $a^{(1)}$ will always be negative! Therefore

$$\frac{\partial h^{(1)}}{\partial a^{(1)}} = 0 \quad \text{and} \quad \frac{\partial J(w)}{\partial w_{1,1}^{(1)}} = 0$$

Neural Networks
Task 2.3: Manual Optimization

Dataset:

x	y
1	0
2	0.3
10	1

Network:

$$h^{(1)} = \mathrm{ReLU}\left(w_{1,1}^{(1)} \cdot x\right) \qquad w_{1,1}^{(1)} = -0.1$$
$$h^{(2)} = \mathrm{ReLU}\left(w_{1,1}^{(2)} \cdot h^{(1)}\right) \qquad w_{1,1}^{(2)} = 0.1$$

Loss function:

$$J(w) = \frac{1}{N} \sum_{(x,y)\in D} \left(\hat{y}(x) - y\right)^2$$

Bonus: Compute gradient for $w^{(1)}_{1,1}$

- Gradient is zero for all samples in our dataset

$$\frac{\partial J(w)}{\partial w_{1,1}^{(1)}} = 0$$

- This means:
  - Any gradient based optimization procedure will not change this weight (for our dataset)!
  - „Dead“ ReLU, network is not learning.
- Note: If we set $w^{(2)}_{1,1} < 0$, this would even be independent of our dataset!

Neural Networks

Task 2.4: Setting up PyTorch

In short:

- Set up a python environment (or use your standard python installation)
  - E.g. conda
- Then run `pip install torch`

Any problems?

Neural Networks

Task 2.5: Implementing a Network (a)

```python
def create_2unit_net() -> Module:
    """
    Create a two-layer MLP (1 hidden layer, 1 output layer) with 2 hidden units
    as described in the exercise.

    Returns:
        2-layer MLP module with 2 hidden units.
    """
    # START TODO ##################
    # Define the model here
    linear_units = 2
    model = Sequential(
        Linear(2, linear_units),
        ReLU(),
        Linear(linear_units, 2),
    )
    # END TODO ###################
    params = model.state_dict()

    # Now we assign the model weights since we still did not learn backpropagation
    params['0.weight'] = torch.Tensor(np.array([[3.21, 3.21], [-2.34, -2.34]]))
    params['0.bias'] = torch.Tensor(np.array([-3.21, 2.34]))
    params['2.weight'] = torch.Tensor(np.array([[3.19, 4.64], [-2.68, -3.44]]))
    params['2.bias'] = torch.Tensor(np.array([-4.08, 4.42]))
    model.load_state_dict(params)

    return model
```

```
$ python run_2unit_model.py

Raw prediction logits:
[[ 6.7775993 -3.6295996]
 [-4.08       4.42     ]
 [-4.08       4.42     ]
 [ 6.1599007 -4.1828003]]

Prediction after softmax:
[[9.9996984e-01 3.0213278e-05]
 [2.0342699e-04 9.9979657e-01]
 [2.0342699e-04 9.9979657e-01]
 [9.9996781e-01 3.2226122e-05]]

True labels, one-hot encoded:
[[1. 0.]
 [0. 1.]
 [0. 1.]
 [1. 0.]]

Loss: 0.00011732114
```

Neural Networks

Task 2.5: Implementing a Network (b)

```python
def create_3unit_net() -> Module:
    """
    Create a two-layer MLP (1 hidden layer, 1 output layer) with 3 hidden units
    as described in the exercise.

    Returns:
        2-layer MLP module with 3 hidden units.
    """
    # START TODO ##################
    # Define the model here
    linear_units = 3
    model = Sequential(Linear(2, linear_units), ReLU(), Linear(linear_units, 2))
    # END TODO ##################

    params = model.state_dict()
    # START TODO ##################
    # change the model weights

    params['0.weight'] = torch.Tensor(np.array([[3.21, 3.21], [-2.34, -2.34], [0, 0]]))
    params['0.bias'] = torch.Tensor(np.array([-3.21, 2.34, 0]))
    params['2.weight'] = torch.Tensor(np.array([[3.19, 4.64, 0], [-2.68, -3.44, 0]]))
    params['2.bias'] = torch.Tensor(np.array([-4.08, 4.42]))

    # END TODO ##################
    model.load_state_dict(params)

    return model
```

```
$ python run_3unit_model.py

Raw prediction logits:
[[ 6.7775993 -3.6295996]
 [-4.08       4.42     ]
 [-4.08       4.42     ]
 [ 6.1599007 -4.1828003]]

Prediction after softmax:
[[9.9996984e-01 3.0213278e-05]
 [2.0342699e-04 9.9979657e-01]
 [2.0342699e-04 9.9979657e-01]
 [9.9996781e-01 3.2226122e-05]]

True labels, one-hot encoded:
[[1. 0.]
 [0. 1.]
 [0. 1.]
 [1. 0.]]

Loss: 0.00011732114
```