Neural Networks: I. Initialization (Part 5)

Jan Bauer

jan.bauer@dhbw-mannheim.de

03.06.19

Sigmoid Function

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

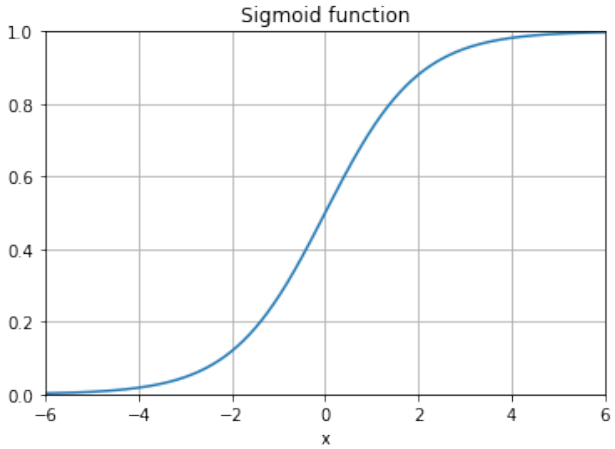

Figure: Sigmoid Function

Sigmoid Function

Problems

- saturated Neurons kill the Gradient flow

Sigmoid Function

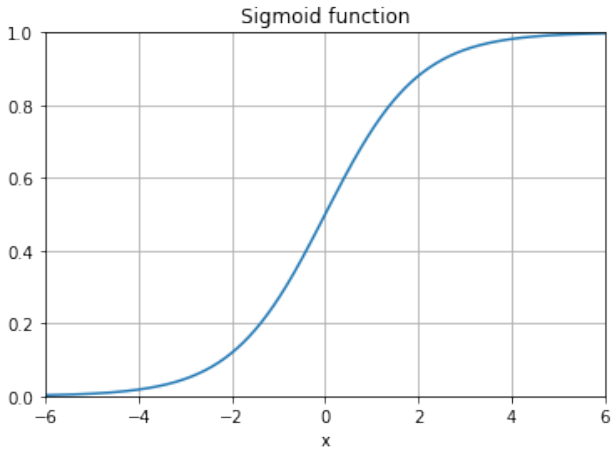

Figure: Sigmoid Function

What happens when...

- x is (fairly) large? (e.g. $x > 6$)
- $x = 0$?
- x is (fairly) small? (e.g. $x < -6$)

Sigmoid Function

Problems

- saturated Neurons kill the Gradient flow
- outputs are all positive (i.e. **not zero-centered**)

Sigmoid Function

- the input of the subsequent Neuron (say Neuron j) is the output of the current one.
- the sign of

$$\frac{\partial}{\partial w}\phi_j\left(xW\right) = \frac{\partial}{\partial w}\phi_j\left(\sum_i x_i w_i\right)$$

  depends only on x

$\Rightarrow$ the gradient is all positive or all negative

$\Rightarrow$ we move only in a strictly positive or strictly negative direction

$\Rightarrow$ inefficient

tanh Function

$$\phi(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} = \frac{\exp(2x) - 1}{\exp(2x) + 1}$$

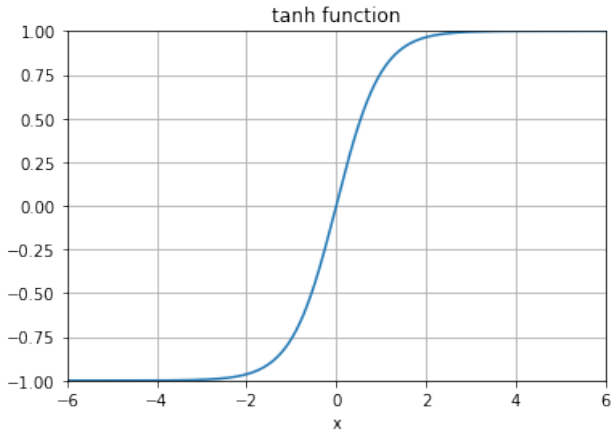

Figure: tanh Function

tanh Function

No problems

- outputs are zero-centered

Problems

- saturated Neurons kill the Gradient flow

ReLU

$$\phi(x) = \max(0, x)$$

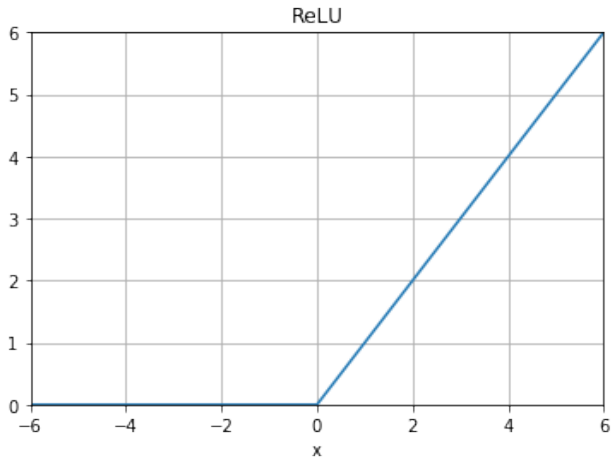

Figure: ReLU Function

ReLU

No problems

- does not saturate for $x > 0$

Problems

- outputs are not zero-centered
- saturates for $x \leq 0$

Cherry on the cake:

- computationally efficient

Parametric Rectifier (PReLU)

$$\phi(x) = \max(\alpha x, x)$$

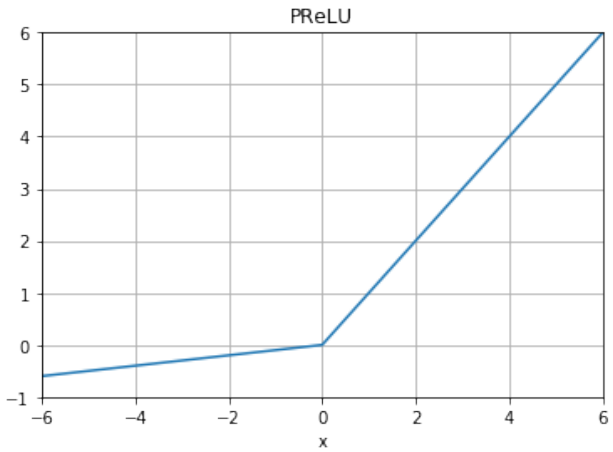

Figure: PReLU with $\alpha = 0.1$

PReLU

No problems

- does not saturate
- outputs can be positive and negative (although not zero-centered)

Cherry on the cake:

- computationally efficient

Standardization

$$X_s = \frac{X - \hat{\mathbb{E}}X}{\sqrt{\hat{\mathbb{V}}X}}$$

- helpful when comparing variables with different scores

Standardization

Consider $x_1 \sim \mathcal{N}(2, 3)$ and $x_2 \sim \mathcal{N}(3, 1)$

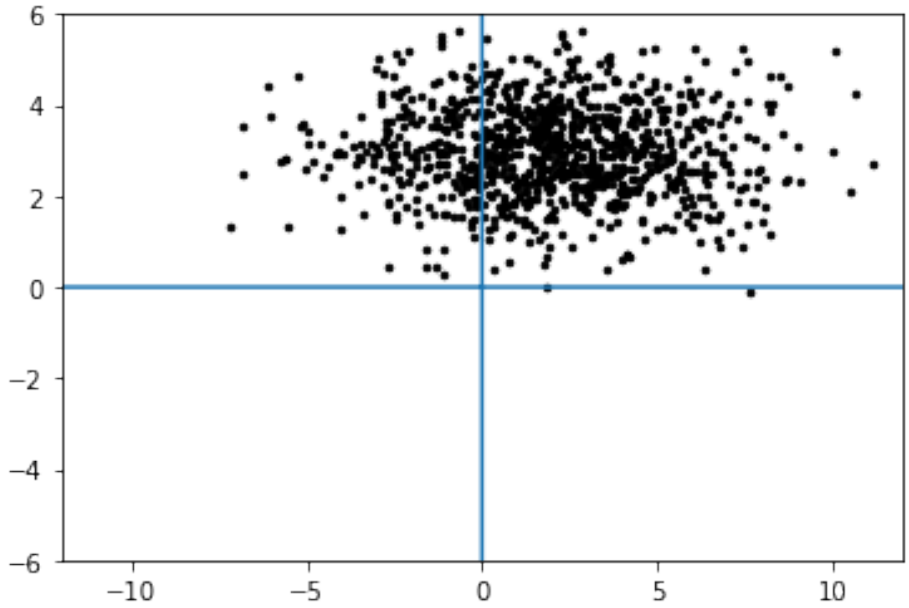

Figure: Random Gaussian Data

Standardization

The mean centered data is then given by

$$X_c = X - \hat{\mathrm{E}}X$$

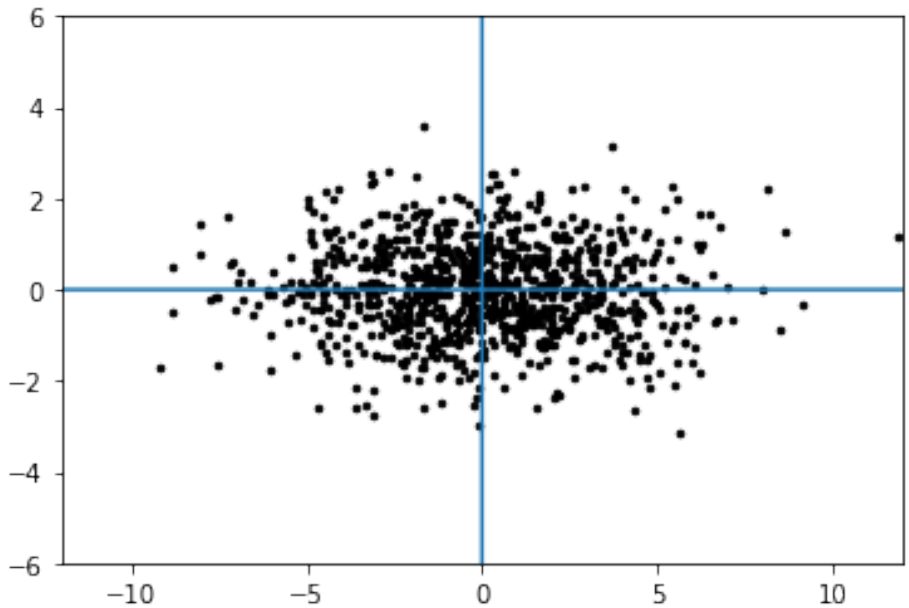

Figure: Mean Centered Random Gaussian Data

Standardization

The standardized data is then given by

$$X_s = \frac{X_c}{\sqrt{\hat{\mathbb{V}}X}} = \frac{X - \hat{\mathbb{E}}X}{\sqrt{\hat{\mathbb{V}}X}}$$

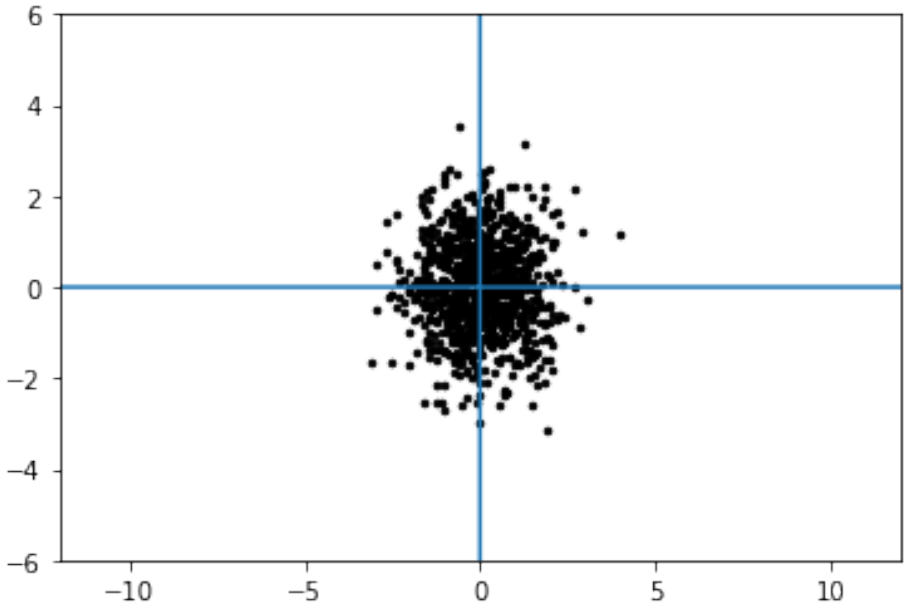

Figure: Mean Centered Random Gaussian Data

Weights initialization

How to initialize weights?

Weights initialization

- zero initialization
- random initialization
  - how to generate them?
  - large values?
  - small values?

zero initialization

Idea:

- presumably, some weights will eventually be positive, some will be negative

$\Rightarrow$ why not setting every weight equal to the mean of zero?

Problem:

- data becomes irrelevant (since $x \cdot 0 = 0$)
- all weights change in the same direction in Gradient Descent, since all Neurons produce the same output
  $\rightarrow$ no asymmetry

random initialization (large values)

Idea:

- draw large weights from a multivariate Gaussian distribution, e.g. $W^{(lm)} \sim 3 \cdot \mathcal{N}(0, 1)$
- weights point towards different directions (nice)

Problem:

- considering deep Neural Networks, the Neurons tend to receive tiny inputs (multiplied by small weights)

$\Rightarrow$ saturation!

random initialization (small values)

Idea:

- draw small weights from a multivariate Gaussian distribution, i.e. $W^{(lm)} \sim 0.01 \cdot \mathcal{N}(0, 1)$
- weights point into different directions (nice)

Concern:

- Neurons receive large input values

$\Rightarrow$ saturation is a concern

Xavier initialization

Idea:

- draw weights from a scaled multivariate Gaussian distribution
- all weights are scaled in a way to contain the variance within the data

$$W^{(lm)} \sim \frac{1}{\sqrt{D}} \cdot \mathcal{N}(0, 1)$$

- weights point towards different directions (nice)

Beyond Initialization: Training Data & Test Data

Worst approach:

1. train the NN using all data (training & test data) ✓

Beyond Initialization: Training Data & Test Data

What I have done:

1. train the NN using the training data
2. run the NN on the test data - **this is your final result**

Awesome? Bad? Okay-ish?

Beyond Initialization: Training Data & Test Data

It is a poor approach, since

- improving the NN is not possible - we stick to our first try

Beyond Initialization: Training Data, Validation Data & Test Data

What about this?

1. split your data into a training, validation and test data set
2. train the NN using the training data
3. check performance using the validation data
4. repeat 2-3 until you are satisfied
5. run the NN on the test data - **this is your final result**

Awesome? Bad? Okay-ish?

Beyond Initialization: Training Data, Validation Data & Test Data

Good approach, but one possible pitfall:

- we keep on improving our NN for the validation data

$\Rightarrow$ we might run into overfitting issues

$\Rightarrow$ generalization is a concern!

Beyond Initialization: Training Data, Validation Data & Test Data

Best approach:

1. split your training data into several, say 7, folds
2. train the NN once on each training data (7) assigning one fold to be the validation data
3. check performance for each training data using its validation data fold
4. average the results
5. repeat 1-4 (or 2-4) until you are satisfied
6. run the NN on the test data - **this is your final result**

Beyond Initialization: Training Data, Validation Data & Test Data

machine intense!

What about Standardization?

Let X be the training data and X^t be the test data (or validation data). Then

$$X_s = \frac{X - \hat{\mathrm{E}}X}{\sqrt{\hat{\mathrm{V}}X}}$$

and

$$X_s^t = \frac{X^t - \hat{\mathrm{E}}X}{\sqrt{\hat{\mathrm{V}}X}}$$

will be fed into the NN.