Neuronale Netze

Eine Neuronales Netz kann im Gegensatz zur Linearen Regression auch nicht-lineare Zusammenhänge abbilden. Dafür ist es jedoch deutlich komplexer und stellt bezüglich des Zustandekommens seiner Ausgabe eine Blackbox dar. Ihre Stärken haben Neuronale Netze bei komplexen Aufgabenstellungen wie beispielsweise der Bildanalyse oder dem Verständnis natürlicher Sprache. Hier soll ein Neuronales Netz zur Prognose von Heizenergieverbräuchen eingesetzt werden.

Aufgabe 1: Daten laden

Zuerst wird die Datei `features.csv` mit den Daten mit Hilfe der Funktion `readtable` geladen. Dann werden die Timestamps und die Features separat gespeichert und mit der Funktion `table2array` zu Arrays konvertiert.

```
clearvars;
clc;

data = readtable('features.csv', 'Delimiter', ',', 'TreatAsEmpty', 'true');
timestamps = table2array(data(:, 1));
features = table2array(data(:, 2 : end));
```

Aufgabe 2: Aufteilung Trainings- und Test-Daten

In diesem Schritt werden die Daten in Trainingsdaten zum Erstellen und in Testdaten zum Testen des Neuronalen Netzes unterteilt. Als Trainingsdaten werden die ersten 729 Zeilen verwendet da diese einen Zeitraum von zwei Jahren abdecken. Außerdem werden die Zielwerte (1. Spalte) von den Features getrennt. Es sollen also die Arrays `training_targets`, `training_features`, `test_targets` und `test_features` vorliegen. Zusätzlich sollen als Vorarbeit für folgende Schritte auch die Timestamps in `training_timestamps` und `test_timestamps` aufgeteilt werden.

```
training_targets = features(1 : 729, 9);
training_features = features(1 : 729, [1 : 8, 10 : end]);
training_timestamps = timestamps(1 : 729, :);
test_targets = features(730 : end, 9);
test_features = features(730 : end, [1 : 8, 10 : end]);
test_timestamps = timestamps(730 : end, :);
```

Aufgabe 3: Neuronales Netz mit Matlab Toolbox erstellen und trainieren

Die Matlab Toolbox stellt eine vollständige Implementierung einer Vielzahl von Neuronalen Netzen zur Verfügung. Mit der Funktion `feedforwardnet` kann ein Feed Forward Netz mit beliebiger Layer Struktur erzeugt werden. Für das Training kann die Funktion `train` genutzt werden. Außerdem kann durch Zuweisung entsprechender Werte zwischen `0` und `1` der Anteil der Validierungsdaten `net.divide.valRatio` und Trainingsdaten `net.divide.trainRatio` festgelegt werden. Hier soll ein Netz mit 14 Units im ersten und 5 Units im zweiten Hidden Layer verwendet werden. Als Validierungsdaten sollen 10% der Trainingsdaten verwendet werden.

Hinweis:

```
net = feedforwardnet([14 5]);
net.divideParam.trainRatio = 90/100;
net.divideParam.valRatio = 10/100;
[net, tr] = train(net, training_features.', training_targets.');
```

Aufgabe 4: Netz testen

Zum Testen des soeben trainierten Netzes werden die zu Anfang von den Trainingsdaten getrennten Testdaten verwendet.

```
output = net(test_features.');
```

Um die Güte der Prognose zu bestimmen werden die durch das Netz prognostizierten Zielwerte mit den tatsächlichen Zielwerten verglichen. Hierfür bieten sich verschiedene Metriken an. An dieser Stelle soll der RMSE (Rooted Mean Square Error) verwendet werden: $RMSE = \sqrt{\frac{\sum_{t=1}^{n}(y(t) - \hat{y}(t))^2}{n}}$

Hinweise:

- zur Subtraktion zweier Matrizen kann die Funktion `gsubtract` verwendet werden
- für das Quadrat zweier Matrizen kann der Operator `.^` verwendet werden

```
RMSE = sqrt(mean(gsubtract(output, test_targets.').^2))
```

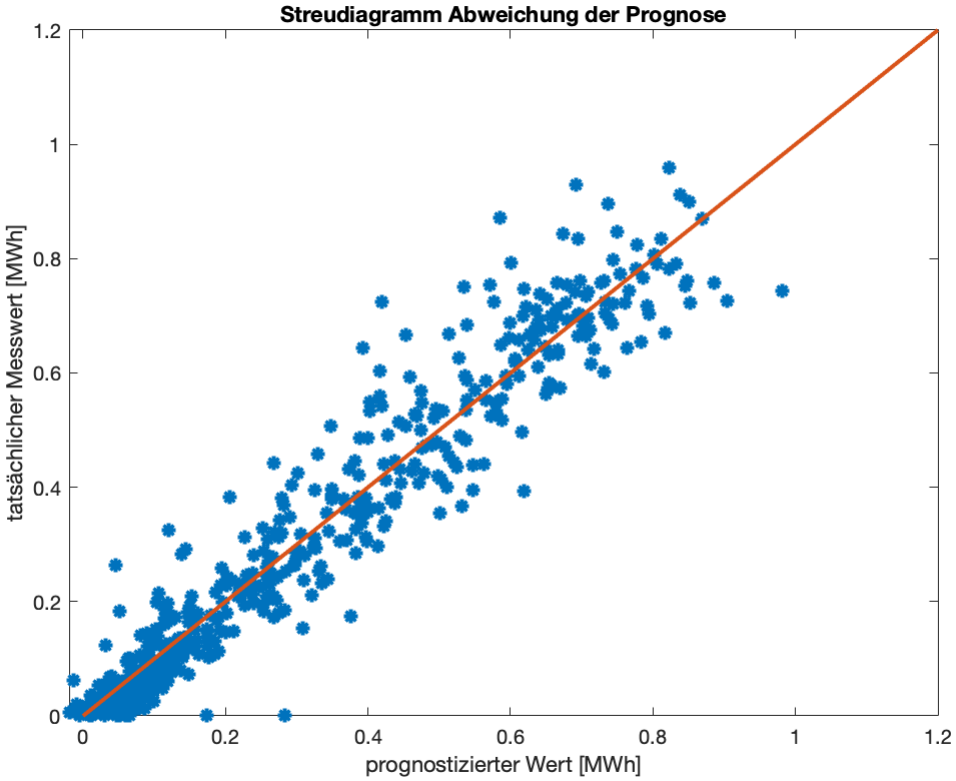
```
RMSE = 0.0678
```

Aufgabe 5: Ergebnisse visualisieren

Für einen visuellen Vergleich der Testergebnisse können verschiedene Plots erstellt werden.

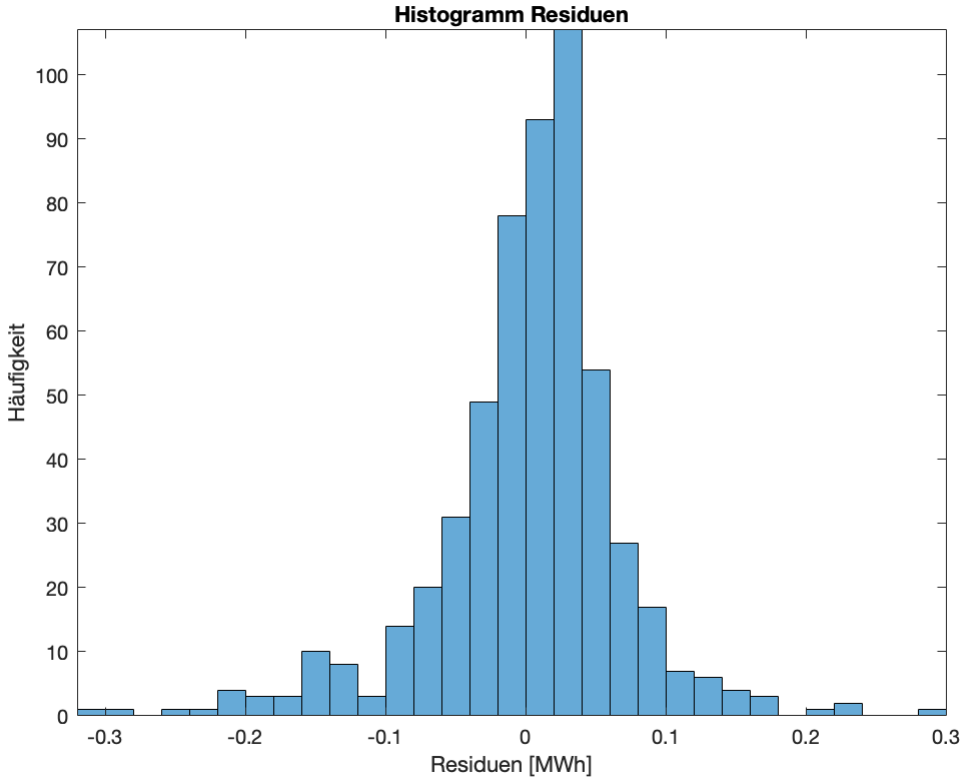
- Ein Streudiagramm mit den prognostizierten Verbräuchen auf der x-Achse und den tatsächlichen Verbräuchen auf der y-Achse:

```
figure
plot(output, test_targets, '*')
hold on
plot(0 : 0.001 : 1.2, 0 : 0.001 : 1.2)
title('Streudiagramm Abweichung der Prognose')
xlabel('prognostizierter Wert [MWh]')%, 'FontSize', 32)
ylabel('tatsächlicher Messwert [MWh]')%, 'FontSize', 32)
%set(gca, 'FontSize', 32)
set(findall(gca, 'Type', 'Line'), 'LineWidth', 2);
axis tight
hold off
```



- Ein Histogramm der Residuen:

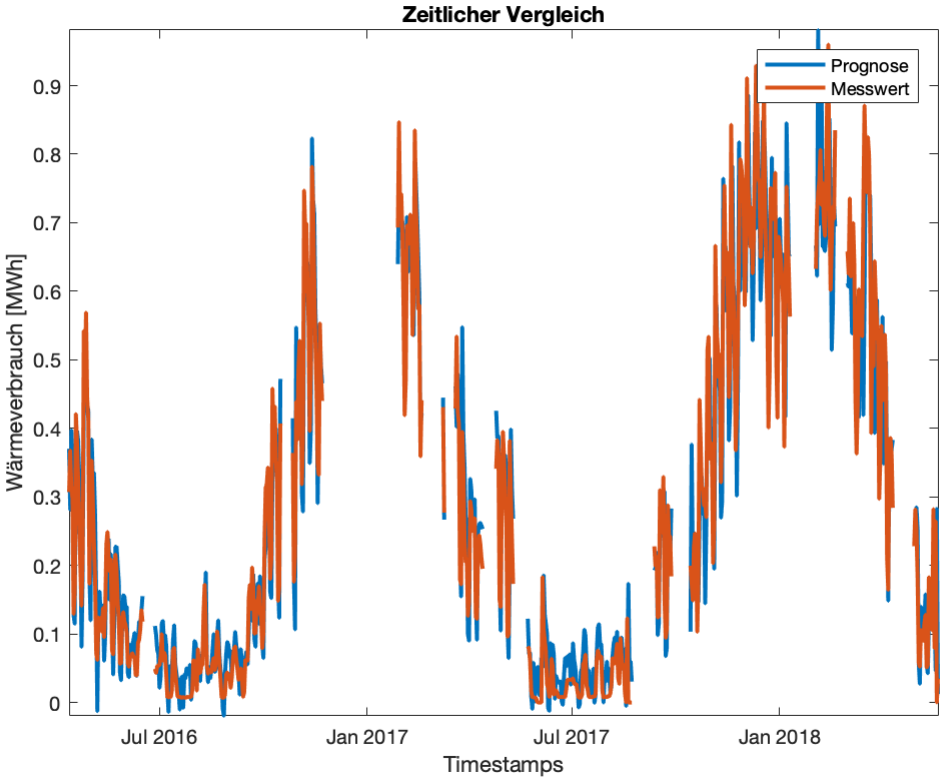
```
figure
histogram(output.' - test_targets, 'BinWidth', 0.02)
hold on
title('Histogramm Residuen')
xlabel('Residuen [MWh]')%, 'FontSize', 32)
ylabel('Häufigkeit')%, 'FontSize', 32)
%set(gca,'FontSize', 32)
set(findall(gca, 'Type', 'Line'), 'LineWidth', 2);
axis tight
hold off
```



- Ein Liniendiagramm welches den Verlauf der prognostizierten Werte sowie der tatsächlichen Werte über den gesamten Zeitraum zeigt:

Hinweis: um einen schönen Plot zu erhalten müssen die Datenlücken in `output` und in `test_targets` zuvor mit `NaN` gefüllt werden. Dazu müssen diese zuerst zu einer `timetable` konvertiert werden und diese dann an die Funktion `retime(tt, 'daily')` übergeben werden.

```
output_gaps = retime(timetable(test_timestamps, output.'), 'daily');
test_targets_gaps = retime(timetable(test_timestamps, test_targets), 'daily');
figure
plot(output_gaps.Properties.RowTimes, output_gaps{:, 1})
hold on
plot(test_targets_gaps.Properties.RowTimes, test_targets_gaps{:, 1})
title('Zeitlicher Vergleich')
xlabel('Timestamps')%, 'FontSize', 32)
ylabel('Wärmeverbrauch [MWh]')%, 'FontSize', 32)
lgd = legend('Prognose', 'Messwert');
%lgd.FontSize = 32;
%set(gca, 'FontSize', 32)
set(findall(gca, 'Type', 'Line'), 'LineWidth', 2);
axis tight
hold off
```



Aufgabe 6: Overfitting

Jetzt soll ein neues Netz ohne Validierungsdaten trainiert und getestet werden. Was fällt beim Vergleich der Ergebnisse auf und wie lässt sich dieser Effekt erklären?

Antwort: Ohne Validierungsset werden die Trainingsdaten "auswendig gelernt" wodurch die Prognose auf dem Testset erheblich schlechter wird. (Overfitting!)

```
net = feedforwardnet([14 5]);
net.divideParam.trainRatio = 100/100;
net.divideParam.valRatio = 0/100;
[net, tr] = train(net, training_features.', training_targets.');

output = net(test_features.');

RMSE = sqrt(mean(gsubtract(output, test_targets.').^2))
```

```
RMSE = 0.1448
```