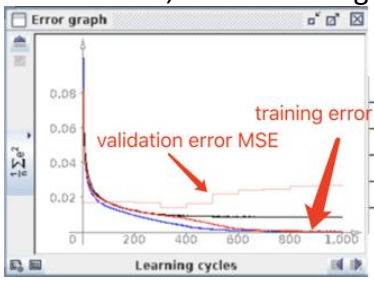
Part1:

1. For the dataset of chronic-kidney-disease-2019.arff, there are 20 attributes and 1 class attribute which is classification problem. In the 20 attributes, there are 10 numeric value and 10 categorical value. In order to fit to the neural network, classification values need to be encoded to 0 1 that will be seen as neural network 'neuron'. Thus, totally, there is 10+10*2 = 30 attributes as input. And 1*2 = 2 attributes are output. Eg. Encode {normal abnormal} to {0 1} {1 0} and all the encoded attributes will follow this rule. (The first attribute appears in chronic-kidney-disease-2019.arff will be seen as 0 1, and the other one will be seen as 1 0)
2. In the dataset, there are 800 attributes and the training, validation and test file will be separated as 50%, 30% and 20% with mixed up (shell coding: shuf). What is more, in order to decrease bias which is influenced by different data level in numeric values. Normalization will be used in weka, which will scale the numeric value in [0 1]. Formula: Y=x-MIN/MAX – MIN. The recommendations from domain expert is obtained, the range can be enlarged, which is suitable for more unseen data prediction in the future. However, this data set will be scaled by existing values (more clear information, please to check the shell code).
   In addition, there are a few missing values in all the provided attributes. For numeric values, the missing values will be processed as 0 after normalization, and for categorical values, the missing value will be processed as 0 0, which can lower influence by the missing values and get benefits from the rest of dataset.
3. Because, for this dataset, the prediction is about predicting the chronic kidney disease, the people who join the test should know whether they got this disease as clear as possible. Thus, the analyse strategy is be set to 304030.
4. This picture gives error graph and recorder for 5 different initializations with training, validation and testing data set.
   Basically, the training error and training MSE will decrease with increasing learning cycles, and with increase of epochs, the overfitting will increase. Although it is clear to see that, with the increasing Epochs, the training error MSE is keep decreasing, which is closing to 0, for the validation error MSE, at first time, the value will decrease with a period of cycles.
   After that, the error MSE will increase and gap between training error line and the validation error line may become bigger, which is overfitting. Furthermore, if it appears a fluctuation in the training error line frequently, this also indicate an overfitting. However, different initialization will have different weight in the net. Thus, this cannot be determined strictly by the number of Epochs.
   In these five times experiment, Run.No 1 can arrive the highest accuracy rate and the lower overfitting (overfitting 4.9211), and compared with train error and test error, the overfitting is not very high (lower than 5%).



Run.NO	Architecture	Parameters	Train MSE	Train Error	Epochs	Test MSE	Test Error
1	30-10-02	lr=.2	0.2873	0	900	5.2084	0.0188
2	30-10-02	lr=.2	0.4526	0	600	5.9183	0.0250
3	30-10-02	lr=.2	0.7466	0.0025	1000	7.0673	0.0312
4	30-10-02	lr=.2	0.2873	0	1000	5.2084	0.0188
5	30-10-02	lr=.2	0.3011	0	1000	5.6983	0.0250

Picture 1

5.

Run.NO	Architecture	Parameters	Train MSE	Train Error	Epochs	Test MSE	Test Error
6	30-02-02	lr=.2	0.4989	0	1900	8.5416	0.0312
7	30-15-02	lr=.2	0.5558	0.25	600	8.8466	0.0375
8	30-20-02	lr=.2	0.3722	0	600	6.6370	0.0312
9	30-25-02	lr=.2	0.3504	0	600	4.5370	0.0250
10	30-30-02	lr=.2	0.3682	0	600	6.65154	0.0312

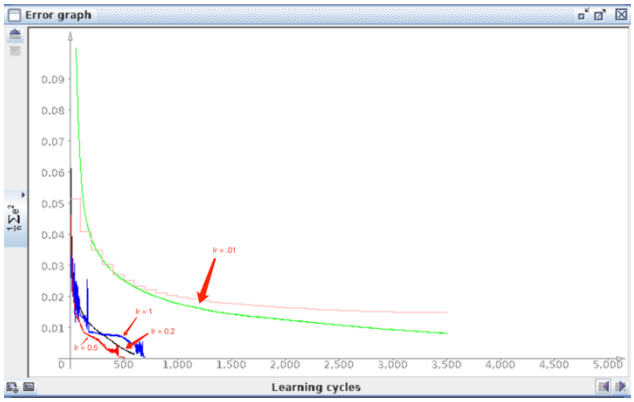
Picture 2

2 hidden nodes are not enough for this data set because, with the increase of Epochs, the error MSE need to learning about 1900 cycles to close to 0, which means the model for training is not complexity enough.

More hidden nodes have a more complex model, which can help training MSE become lower in lower Epochs. What is more, although for 25 hidden nodes, it has the smallest overfitting (4.1866), the classification error rate is not the lowest one (compared with picture1).

Thus, 10 hidden nodes (Test error 0.0188, overfitting 0.49211) seem enough for this data set.

6. With the increase of learning rate, the Epochs decrease sharply, which means the training error line will close to 0 in less Epochs. However, if the learning rate is too high, it will lead to an extremely fluctuation, which influence the error line to close 0. In this dataset, 0.2 is the sweet point.



Picture 3

7.

Run.NO	Architecture	Parameters	Train MSE	Train Error	Epochs	Test MSE	Test Error
1	30-10-02	lr=0.1 momentum =0.1	0.41065	0.00%	600	5.891864	2.50%
2	30-10-02	lr=0.8 momentum =0.8	3.94	0.25%	400	4.3067	1.25%
3	30-10-02	lr=0.2 momentum =0.6	0.128715	0.00%	300	5.388117	2.50%
4	30-10-02	lr=0.4 momentum =0.4	0.721331	0.00%	300	7.692579	2.50%
5	30-10-02	lr=0.6 momentum =0.2	0.11391	0.00%	300	6.63066	2.50%

Picture 4

Learning rate and momentum will increase the learning speed for data set. Athough lr = 0.8, momentum = 0.8 can get the lowest test error and the smallest overfitting (0.4067), it can see an extremely high fluctuation rate in the error graph during learning, which means the study rate is too fast.

8.

Run.NO	Architecture	Parameters	Train MSE	Train Error	Epochs	Test MSE	Test Error
12	30-10-02	lr=.2	0.2378	0	1400	6.2216	0.0250

Picture 5

The gap between the training MSE and Test MSE is very high, which is about 6. Although the test error rate is not very high, it may influence by the type of data set or validation strategy (30 40 30). In this dataset, it did not influence so much, but it appears overfitting.

9. The accuracy rate for both of algorithm, J48 and Multiplayer Perceptron in default parameter is very high prediction in 10 folds cross validation, which is 99.13% and 99.75%, which is higher than all the situations of neural network.
   It is obvious to see that multiplayer perceptron has the highest accuracy rate and the lowest overfitting.

	Parameters	Train Error	Cross validation	Overfitting
J48	C = 0.25, M = 2	0.25%	0.88%	0.63
MultiplayerPerceptron	L=0.3 M=0.2 N=500 V=0 S=0 E=20 H=a	0	0.25%	0.25

Part2:

1. Totally, there are 20 attributes. 9 numeric values, 11 categorical values and 1 class attribute which is age. Most of encoding is similar with part1. However, the attribute age will become to a class and the original attribute will be a training attribute(categorical value). For all the categorical value, it will be encoded to 1 0, 0 1 or 0 0 for missing value. Thus, there are 9+11*2=31 attributes input and 1 output.
2. There are 800 attributes and the training, validation and test file will be separated as 50%, 30% and 20% with mixed up. For numeric values except the attribute of age, those will be normalized to [0 1]. The missing value will be set as 0. The strategy for normalizing the age is to get the minimal and maximum value by sort function, and then formula Y=x-MIN/MAX – MIN will be used in a for loop method to calculate results one by one. Vice versa for reverse scale. Formula: Y*(MAX -MIN) + MIN = X (please to check appendix)
3. Because the reverse scale has been coded in reverse.sh, it will generate real age gap automatically. It is obviously to see that 100 Epochs will have the lowest overfitting, which is about 0.93. However, that means this model did not fully train the model as well because the test MAE is very high. According to above experimental data, run. No 2 will be selected because it got the lowest test MAE and do not have very high overfitting.

Run.NO	Architecture	Parameters	Train MAE	Epochs	Test MAE	Overfitting
1	31-05-01	lr=.2	10.80	800	16.08	5.28
2	31-05-01	lr=.2	11.68	600	14.32	2.64
3	31-05-01	lr=.2	12.56	400	15.20	2.64
4	31-05-01	lr=.2	13.44	200	15.20	1.76
5	31-05-01	lr=.2	16.96	100	17.89	0.93

4. Here we can observe that the overfitting will stably increase with increase of hidden nodes. In the hidden nodes 2, it will get the lowest overfitting (0.77) and a not bad test MAE.

Run.NO	Architecture	Parameters	Train MAE	Epochs	Test MAE	Overfitting
6	31-02-01	lr=.2	13.55	300	14.32	0.77
7	31-08-01	lr=.2	12.56	300	14.32	1.76
8	31-15-01	lr=.2	11.68	300	14.30	2.62
9	31-25-01	lr=.2	10.80	300	14.32	3.52
10	31-40-01	lr=.2	10.80	300	14.32	3.52

5.

8. The below table shows that there is an higher overfitting, which is about 4.4.

Run.NO	Architecture	Parameters	Train MAE	Epochs	Test MAE	Overfitting
1	31-05-01	lr=.2	9.92	50000	14.32	4.40

9. The MultiplayerPerceptron can have a little bit high overfitting (reverse convert: 5.4) and a lower error value in cross validation(reverse convert: 13.6). Compared with all the neural network experimental statistics, this is the best prediction. For overfitting, it needs to adjust parameters to decrease overfitting.

	Parameters	Train Error(MAE)	Cross validation(MAE)	Overfitting
J48	C = 0.25, M = 2	0.0807	0.142	0.0613
MultiplayerPerceptron	L=0.3 M=0.2 N=500 V=0 S=0 E=20 H=a	0.0932	0.1319	0.0387

Appendix:

Part1 preprocess.sh:

```
FILE='chronic-kidney-disease-2019-normalized.arff'
#head -n +25 $FILE
#exit

# separate the title, finally, use cat to connect it
head -n +25 $FILE > title.arff

# extract raw data
tail -n +26 $FILE | shuf  > raw-data.arff
#head -25 title.arff

# seperate all data to 20 lines
#!/bin/bash
for ((i=1; i<22; i=i+1))
do
   cut -d, -f$i raw-data.arff >temp$i.arff
done

# processed the first two columns
for ((i=1; i<3; i=i+1))
do
#       if [[ $i == 1 ]] || [[ $i == 2 ]] ;then
        cat temp$i.arff|
        sed -e 's/?/0/' temp$i.arff >processed$i.arff;
#     fi
done
# processed the third to sixth columns, abnormal as 1 0, normal as 0 1
# not present 1 0, present 0 1
for ((i=3; i<7; i=i+1))
do
        cat temp$i.arff|\
        sed -e "s/abnormal/1 0/g"\
            -e "s/normal/0 1/g"\
            -e "s/notpresent/1 0/g"\
            -e "s/present/0 1/g"\
            -e "s/?/0 0/g">processed$i.arff;
done

# processed the 7th to 14th, ? to 0
for ((i=7; i<15; i=i+1))
do
        cat temp$i.arff|\
        sed -e "s/?/0/g">processed$i.arff;
done
```

```
# processed the 15th to 21th, yes as 0 1, no as 1 0
# poor as 1 0, good as 0 1, notskd as 1 0, ckd as 0 1
for ((i=15; i<22; i=i+1))
do
        cat temp$i.arff|\
        sed -e "s/poor/1 0/g"\
            -e "s/good/0 1/g"\
            -e "s/notckd/1 0/g"\
            -e "s/ckd/0 1/g"\
            -e "s/no/1 0/g"\
            -e "s/yes/0 1/g"\
            -e "s/?/0 0/g" > processed$i.arff;
done
# create an output file for javanns

COMBINE=''
str1=" "
str2="processed"
for ((i=1; i<22; i=i+1))
do
COMBINE=$COMBINE$str2$i.arff$str1
done
paste -d " " $COMBINE > output-javanns.arff

# create an output file for weka
cat title.arff output-javanns.arff > output-weka.arff

# separate fill to different pattern, which will be inserted into javanns
TRAIN=400
VALID=240
TEST=160

# load the iris.arff from the system to the memory, which begins from the 1 line
# fgrep -v, show results from the file except "%", and then shuf it
tail -n +1 output-javanns.arff | fgrep -v "%" > output-javanns.txt

# The training file
# initial the disease-train.pat file
/bin/echo "SNNS pattern definition file V3.2"  >disease-train.pat
/bin/echo "generated at Mon Apr 25 15:58:23 1994"  >>disease-train.pat
/bin/echo ""  >>disease-train.pat
/bin/echo ""  >>disease-train.pat
/bin/echo "No. of patterns : $TRAIN"  >>disease-train.pat
/bin/echo "No. of input units : 30"  >>disease-train.pat
/bin/echo "No. of output units : 2"  >>disease-train.pat
```

```
# split it to train file
        head -$TRAIN output-javanns.txt >> disease-train.pat
# The validation file
/bin/echo "SNNS pattern definition file V3.2"  >disease-valid.pat
/bin/echo "generated at Mon Apr 25 15:58:23 1994"  >>disease-valid.pat
/bin/echo ""  >>disease-valid.pat
/bin/echo ""  >>disease-valid.pat
/bin/echo "No. of patterns : $VALID"  >>disease-valid.pat
/bin/echo "No. of input units : 30"  >>disease-valid.pat
/bin/echo "No. of output units : 2"  >>disease-valid.pat

# for the valid pattern, it should begin from the length of train test + 1
# from is a new variable which is used to save the current line
FROM=`expr $TRAIN + 1`
tail -n +$FROM output-javanns.txt | head -$VALID >> disease-valid.pat



# The test file
/bin/echo "SNNS pattern definition file V3.2"  >disease-test.pat
/bin/echo "generated at Mon Apr 25 15:58:23 1994"  >>disease-test.pat
/bin/echo ""  >>disease-test.pat
/bin/echo ""  >>disease-test.pat
/bin/echo "No. of patterns : $TEST"  >>disease-test.pat
/bin/echo "No. of input units : 30 "  >>disease-test.pat
/bin/echo "No. of output units : 2"  >>disease-test.pat
FROM=`expr $FROM + $VALID`
tail -n +$FROM output-javanns.txt >> disease-test.pat

exit
```

Part2 preprocess.sh:

```
FILE='chronic-kidney-disease-2019-normalized-except-age.arff'
#head -n +24 $FILE
#exit

# separate the title, finally, use cat to connect it
head -n +24 $FILE > title.arff

# extract raw data
tail -n +25 $FILE > raw-data.arff
#head -30 title.arff

# seperate all data to 20 lines
#!/bin/bash
for ((i=1; i<21; i=i+1))
do
   cut -d, -f$i raw-data.arff >temp$i.arff
done

# processed the first two columns, change the missing value to 0
for ((i=1; i<2; i=i+1))
do
#       if [[ $i == 1 ]] || [[ $i == 2 ]] ;then
        cat temp$i.arff|
        sed -e 's/?/0/' temp$i.arff >processed$i.arff;
#     fi
done
# processed the third to sixth columns, abnormal as 1 0, normal as 0 1
# not present 1 0, present 0 1
for ((i=2; i<6; i=i+1))
do
        cat temp$i.arff|\
        sed -e "s/abnormal/1 0/g"\
            -e "s/normal/0 1/g"\
```

```
        -e "s/notpresent/1 0/g"\
        -e "s/present/0 1/g"\
        -e "s/?/0 0/g">processed$i.arff;
done

# processed the 7th to 14th, ? to 0
for ((i=6; i<14; i=i+1))
do
      cat temp$i.arff|\
      sed -e "s/?/0/g">processed$i.arff;
done

# processed the 15th to 21th, yes as 0 1, no as 1 0
# poor as 1 0, good as 0 1, notskd as 1 0, ckd as 0 1
for ((i=14; i<21; i=i+1))
do
      cat temp$i.arff|\
      sed -e "s/poor/1 0/g"\
         -e "s/good/0 1/g"\
         -e "s/notckd/1 0/g"\
         -e "s/ckd/0 1/g"\
         -e "s/no/1 0/g"\
         -e "s/yes/0 1/g"\
         -e "s/?/0 0/g" > processed$i.arff;
done

# normalized the age value
# get the MIN and MAX value
AGEFILE='age-class.arff'
> processed21.arff


MIN=$(tail -n +6 $AGEFILE |sed -e "s/?/2/g" | sort -n | head -1)
MAX=$(tail -n +6 $AGEFILE |sed -e "s/?/2/g" |sort -n -r |head -1)

tail -n +6 $AGEFILE|
sed -e "s/?/2/g" > temp-age.arff


# scale it to [0 1]
for NUM in $(tail -n +1 temp-age.arff)
do
 value=$(echo "scale =2; ($NUM -$MIN)/($MAX-$MIN)" | bc)
 echo "$(printf "%.2f" $value)"  >> processed21.arff
done
#exit
```

```
# create an output file for javanns

COMBINE=''
str1=" "
str2="processed"
for ((i=1; i<22; i=i+1))
do
COMBINE=$COMBINE$str2$i.arff$str1
done
paste -d " " $COMBINE > output-javanns.arff


# separate fill to different pattern, which will be inserted into javanns
TRAIN=400
VALID=240
TEST=160

# load the iris.arff from the system to the memory, which begins from the 1 line
# fgrep -v, show results from the file except "%", and then shuf it
tail -n +1 output-javanns.arff | fgrep -v "%" | shuf > output-javanns.txt

# The training file
# initial the disease-train.pat file
/bin/echo "SNNS pattern definition file V3.2"  >disease-train.pat
/bin/echo "generated at Mon Apr 25 15:58:23 1994"  >>disease-train.pat
/bin/echo ""  >>disease-train.pat
/bin/echo ""  >>disease-train.pat
/bin/echo "No. of patterns : $TRAIN"  >>disease-train.pat
/bin/echo "No. of input units : 31"  >>disease-train.pat
/bin/echo "No. of output units : 1"  >>disease-train.pat

# split it to train file
head -$TRAIN output-javanns.txt >> disease-train.pat



# The validation file
/bin/echo "SNNS pattern definition file V3.2"  >disease-valid.pat
/bin/echo "generated at Mon Apr 25 15:58:23 1994"  >>disease-valid.pat
/bin/echo ""  >>disease-valid.pat
/bin/echo ""  >>disease-valid.pat
/bin/echo "No. of patterns : $VALID"  >>disease-valid.pat
/bin/echo "No. of input units : 31"  >>disease-valid.pat
/bin/echo "No. of output units : 1"  >>disease-valid.pat

# for the valid pattern, it should begin from the length of train test + 1
# from is a new variable which is used to save the current line
```

```
FROM=`expr $TRAIN + 1`
tail -n +$FROM output-javanns.txt | head -$VALID >> disease-valid.pat



# The test file
/bin/echo "SNNS pattern definition file V3.2"  >disease-test.pat
/bin/echo "generated at Mon Apr 25 15:58:23 1994"  >>disease-test.pat
/bin/echo ""  >>disease-test.pat
/bin/echo ""  >>disease-test.pat
/bin/echo "No. of patterns : $TEST"  >>disease-test.pat
/bin/echo "No. of input units : 31 "  >>disease-test.pat
/bin/echo "No. of output units : 1"  >>disease-test.pat
FROM=`expr $FROM + $VALID`
tail -n +$FROM output-javanns.txt >> disease-test.pat

exit
```

Part2 Reverse.sh:

```
# temp-age.arff includs all the age rows, which is cut from origial file
AGEFILE=temp-age.arff
MIN=$(tail -n +6 $AGEFILE | sort -n | head -1)
MAX=$(tail -n +6 $AGEFILE |sort -n -r |head -1)

# test1test.res is generated by javanns in test file (just predicted results)
# change different file names to match and analyze the files to get reverse results
RESULT=test10.res
TRAIN=disease-test.pat
# got all the even values which are pure predicted values in all ages
tail -n +9 $RESULT | awk 'NR%2==0' > real-result.txt
# separate age column from disease-test.pat which is expect results in age
tail -n +8 $TRAIN | cut -d" " -f32 > expect-result.txt
paste -d " " real-result.txt expect-result.txt > raw-output-age.txt
# get gap between two values (some values are positive, some values are negative)
var= cat raw-output-age.txt | awk -F ' ' '{print ($1-$2)}' > age-gap.txt

#function abs()
#{
#  if (( $1 > 0));then
#       return $1
#  else
```

```
#       return $((0-$1));
#  fi
#
#}

abs() {
   [[ $[ $@ ] -lt 0 ]] && echo "$[ ($@) * -1 ]" || echo "$[ $@ ]"
}
# save all the value in absolute type
> age-gap2.txt
for NUM in $(tail -n +1 age-gap.txt)
do
echo $NUM | sed 's/-//' >> age-gap2.txt
#echo $(($NUM))
done

# reverse function, scale normalized values to real age
count=0;
total=0;
# get sumup/count --> average age gap
for i in $(tail -n +1 age-gap2.txt)
  do
    total=$(echo $total+$i | bc)
    ((count++))
  done
value=$(echo "scale=2; $total/$count" | bc)
echo "$(printf "%.2f" $value)" > output-age-gap.txt
#echo $value
#echo $MAX
#echo $MIN

# use the function to convert age. expect function: $result*($MAX-$MIN))+$MIN
# cuz above formula doesn't work, the variable is separated one by one
result=`head -1 output-age-gap.txt`
# result= $((head -1 output-age-gap.txt))
result1=$(($MAX-$MIN))
# got results for real age gap
echo "results for age from [0 1] to real age MAE: "
echo "scale=2; $result1*$result+$MIN" | bc

exit
```