Question 1

a

A neural network that implements Standard Linear Regression:
Input layer: $\tilde{x} = (1, x) \in \mathbb{R}^{p+1}$
Weights: $w_i = \beta_i \quad , \quad i = 0, ..., p$
No hidden layers.
Output layer: $\hat{y} = w^T \tilde{x}$
Minimize loss function with Gradient Descent (we'll use LS): $||y - \hat{y}||^2$
This implements linear regression since the solutio will be a linear combination that tries to minmized the least square loss function

A neural network that implements Logistic Regression:
Input layer: $\tilde{x} = (1, x) \in \mathbb{R}^{p+1}$
Weights: $w_i = \beta_i \quad , \quad i = 0, ..., p$
No hidden layers.\ Non-linear function between layers: $g(\tilde{x}, w) = log(\frac{1}{1+e^{-w^T \tilde{x}}})$
Output layer: $\hat{p} = g(w^T \tilde{x})$
Minimize loss function with Gradient Descent (we'll use Binary Entropy):
$\tilde{p} \qquad , \quad y = 1$
$1 - \tilde{p} \quad , \quad y = 0$
This implements logistic regression by preforming a linear comination threw the logit function that tries to minmized the least logit loss function

Becuase both problems are convex we expect that the model will be the same if the neural network manges to converge. Because the step size of the gradient descent isn't optimaly choosen we have have no garuntee of converges.

```
set.seed(6)
heart = read.csv("https://[redacted].data",row.names=1)
heart$famhist= as.numeric(heart$famhist) # need only numbers
heart=array(unlist(heart),dim=c(462,10)) # move from data frame to array for keras

n = dim(heart)[1]
p = dim(heart)[2]
test.id = sample(n,n/3)

x_train = heart[-test.id,-p]
y_train = heart[-test.id,p]
x_test = heart[test.id,-p]
y_test = heart[test.id,p]


batch_size <- 32
epochs <- 1000
rm(model)
model <- keras_model_sequential()
model %>%
  layer_dense(units = 1, activation = 'sigmoid', input_shape = c(dim(x_train)[2]))

# example with two layers:
#  model %>%
#  layer_dense(units = 3, activation = 'sigmoid', input_shape = c(dim(x_train)[2])) %>%
#  layer_dense(units = 1, activation = 'sigmoid')
```

```r
summary(model)
```

```
## ___________________________________________________________________________
## Layer (type)                     Output Shape                  Param #
## ===========================================================================
## dense_7 (Dense)                  (None, 1)                     10
## ===========================================================================
## Total params: 10
## Trainable params: 10
## Non-trainable params: 0
## ___________________________________________________________________________
```

```r
model %>% compile(
  loss = 'binary_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)


model %>% fit(
  x_train, y_train,
  batch_size = batch_size,
  epochs = epochs,
  validation_data = list(x_test, y_test)
)

#######
model_lin <- keras_model_sequential()
model_lin %>%
  layer_dense(units = 1, input_shape = c(dim(x_train)[2]))

# example with two layers:
#  model %>%
#  layer_dense(units = 3, activation = 'sigmoid', input_shape = c(dim(x_train)[2])) %>%
#  layer_dense(units = 1, activation = 'sigmoid')


summary(model_lin)
```

```
## ___________________________________________________________________________
## Layer (type)                     Output Shape                  Param #
## ===========================================================================
## dense_8 (Dense)                  (None, 1)                     10
## ===========================================================================
## Total params: 10
## Trainable params: 10
## Non-trainable params: 0
## ___________________________________________________________________________
```

```r
model_lin %>% compile(
  loss = 'mean_squared_error',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)


model_lin %>% fit(
  x_train, y_train,
  batch_size = batch_size,
  epochs = epochs,
  validation_data = list(x_test, y_test)
)


model_hidden <- keras_model_sequential()
model_hidden %>%
  layer_dense(units = 10, activation = 'sigmoid', input_shape = c(dim(x_train)[2])) %>%
  layer_dense(units = 1, activation = 'sigmoid')



summary(model_hidden)
```

```
## ___________________________________________________________________________
## Layer (type)                     Output Shape                  Param #
## ===========================================================================
## dense_9 (Dense)                  (None, 10)                    100
## ___________________________________________________________________________
## dense_10 (Dense)                 (None, 1)                     11
## ===========================================================================
## Total params: 111
## Trainable params: 111
## Non-trainable params: 0
## ___________________________________________________________________________
```

```r
model_hidden %>% compile(
  loss = 'mean_squared_error',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)


model_hidden %>% fit(
  x_train, y_train,
  batch_size = batch_size,
  epochs = epochs,
  validation_data = list(x_test, y_test)
)


model_hidden_linear <- keras_model_sequential()
model_hidden_linear %>%
```

```r
  layer_dense(units = 3 , input_shape = c(dim(x_train)[2])) %>%
  layer_dense(units = 1)



summary(model_hidden_linear)
```

```
## ___________________________________________________________________________
## Layer (type)                     Output Shape                  Param #
## ===========================================================================
## dense_11 (Dense)                 (None, 3)                     30
## ___________________________________________________________________________
## dense_12 (Dense)                 (None, 1)                     4
## ===========================================================================
## Total params: 34
## Trainable params: 34
## Non-trainable params: 0
## ___________________________________________________________________________
```

```r
model_hidden_linear %>% compile(
  loss = 'mean_squared_error',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)


model_hidden_linear %>% fit(
  x_train, y_train,
  batch_size = batch_size,
  epochs = epochs,
  validation_data = list(x_test, y_test)
)

print("Neural logistic")
```

```
## [1] "Neural logistic"
```

```r
phat_NN_1_sig = predict_proba(model,x_test, batch_size = NULL, verbose = 0, steps = NULL)
#summary(phat_NN_1_sig)
table(phat_NN_1_sig>0.5, y_test) # 2*2 table
```

```
##       y_test
##          0   1
##   TRUE 100  54
```

```r
print("Neural linear")
```

```
## [1] "Neural linear"
```

```r
phat_NN_1_sig = predict(model_lin,x_test, batch_size = NULL, verbose = 0, steps = NULL)
#summary(phat_NN_1_sig)
table(phat_NN_1_sig>0.5, y_test) # 2*2 table
```

```
##        y_test
##           0   1
##   FALSE 100  46
##   TRUE    0   8
```

```r
print("logistic")
```

```
## [1] "logistic"
```

```r
mod.lr = glm(y~., data=data.frame(x=x_train,y=y_train), family=binomial)
phat_logit = predict.glm(mod.lr, newdata = data.frame(x=x_test,y=y_test),type="response")
table(phat_logit>0.5, y_test) # 2*2 table
```

```
##        y_test
##          0  1
##   FALSE 86 20
##   TRUE  14 34
```

```r
print("linear")
```

```
## [1] "linear"
```

```r
mod.lr2 = lm(y~., data=data.frame(x=x_train,y=y_train), family=binomial)
lin_logit = predict.glm(mod.lr2, newdata = data.frame(x=x_test,y=y_test))
table(lin_logit>0.5, y_test) # 2*2 table
```

```
##        y_test
##          0  1
##   FALSE 88 24
##   TRUE  12 30
```

```r
print("Neural hidden")
```

```
## [1] "Neural hidden"
```

```r
phat_NN_1_sig = predict_proba(model_hidden,x_test, batch_size = NULL, verbose = 0, steps = NULL)
#summary(phat_NN_1_sig)
table(phat_NN_1_sig>0.5, y_test) # 2*2 table
```

```
##        y_test
##          0  1
##   FALSE 67 28
##   TRUE  33 26
```

```r
print("Neural hidden_linear")
```

```
## [1] "Neural hidden_linear"
```

```r
phat_NN_1_sig = predict(model_hidden_linear,x_test, batch_size = NULL, verbose = 0, steps = NULL)
table(phat_NN_1_sig>0.5, y_test) # 2*2 table
```

```
##        y_test
##          0  1
##   FALSE 77 15
##   TRUE  23 39
```

b

We see that the logistic neural network didnt mange to converge well on a good solution. The linear regression did and preforfmed close to the real model. A reason why there might be a diffreance between the linear neural network and the actual model is that the neural network approximate the minimum numerically while the linear regression model uses an analytical solution.

c

It did mange to find a reasonable solutoion but it is not much better than the linear and logistic models. It did get a very low loss on the train set that might indicate overfiting.

d

Obviously since the input to all nodes is linear and the activation is linear we have a linear combination of linear combination which in itself is a linear combination. This model basically tries to find a linear regression solution for the problem.