

Assignment 1

Gang Peng # 1002961921

February 24, 2020

The goal of this assignment is to get you familiar with the basics of decision theory and gradient-based model fitting.

1 Decision theory [13pts]

One successful use of probabilistic models is for building spam filters, which take in an email and take different actions depending on the likelihood that it's spam.

Imagine you are running an email service. You have a well-calibrated spam classifier that tells you the probability that a particular email is spam: $p(\text{spam}|\text{email})$. You have three options for what to do with each email: You can show it to the user, put it in the spam folder, or delete it entirely.

Depending on whether or not the email really is spam, the user will suffer a different amount of wasted time for the different actions we can take, $L(\text{action}, \text{spam})$:

Action	Spam	Not spam
Show	10	0
Folder	1	50
Delete	0	200

1. [3pts] Plot the expected wasted user time for each of the three possible actions, as a function of the probability of spam: $p(\text{spam}|\text{email})$

```
losses = [[10, 0],  
          [1, 50],  
          [0, 200]]
```

```
num_actions = length(losses)
```

```
function expected_loss_of_action(prob_spam, action)  
    #TODO: Return expected loss over a Bernoulli random variable  
    #       with mean prob_spam.  
    #       Losses are given by the table above.
```

```
    expected = prob_spam .* losses[action][1] .+ (1.- prob_spam) * losses[action][2]
```

```
    return expected
```

```

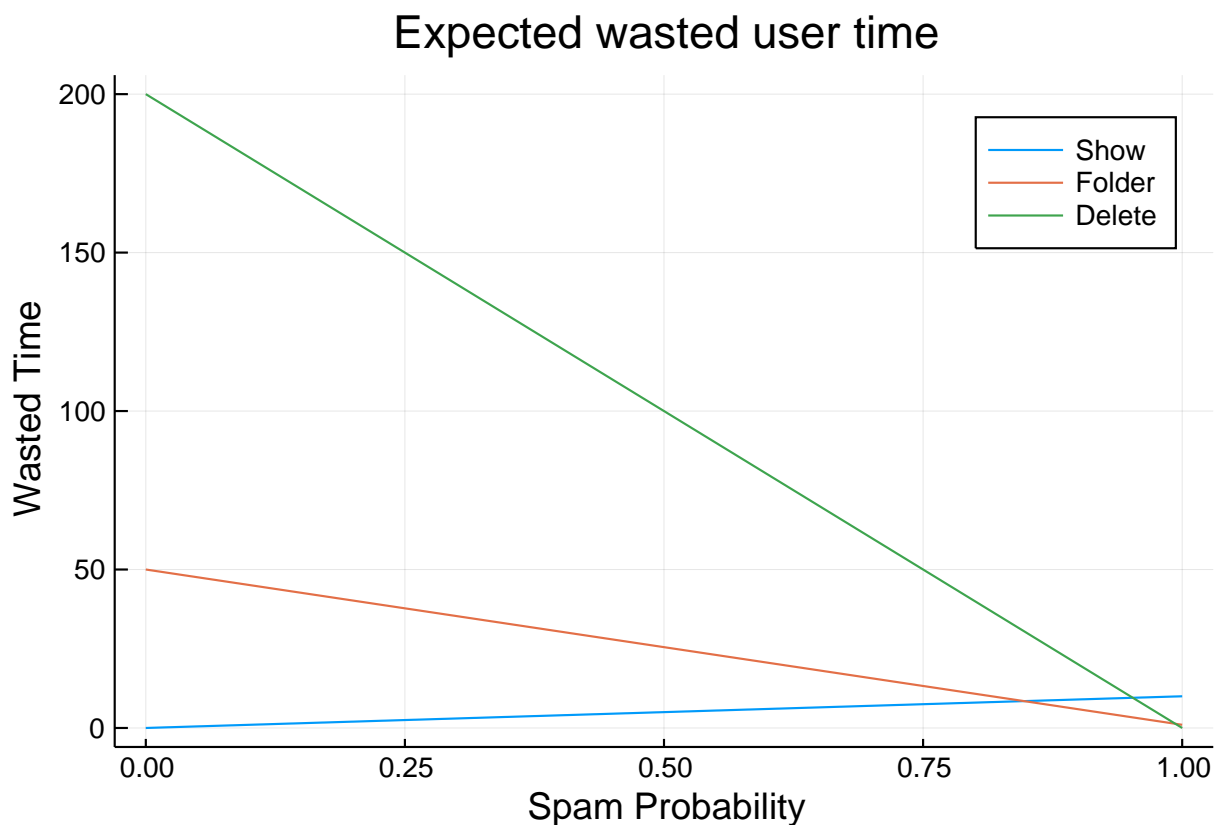
end

prob_range = range(0., stop=1., length=500)

# Make plot
using Plots
labels=["Show" "Folder" "Delete"]

p1=plot(prob_range, expected_loss_of_action(prob_range, 1),title="Expected wasted user
time",
        label=labels[1],xlabel="Spam Probability",ylabel="Wasted Time")
plot!(prob_range,expected_loss_of_action(prob_range, 2),label=labels[2])
display(plot!(prob_range,expected_loss_of_action(prob_range, 3),label=labels[3]))

```



2. [2pts] Write a function that computes the optimal action given the probability of spam.

```

function optimal_action(prob_spam)
    #TODO: return best action given the probability of spam.
    # Hint: Julia's findmin function might be helpful.
    expected_losses=Array{Float64,1}(undef, num_actions)
    for action in 1:num_actions
        expected_losses[action]=expected_loss_of_action(prob_spam,action)
    end
    return findmin(expected_losses)
end

optimal_action (generic function with 1 method)

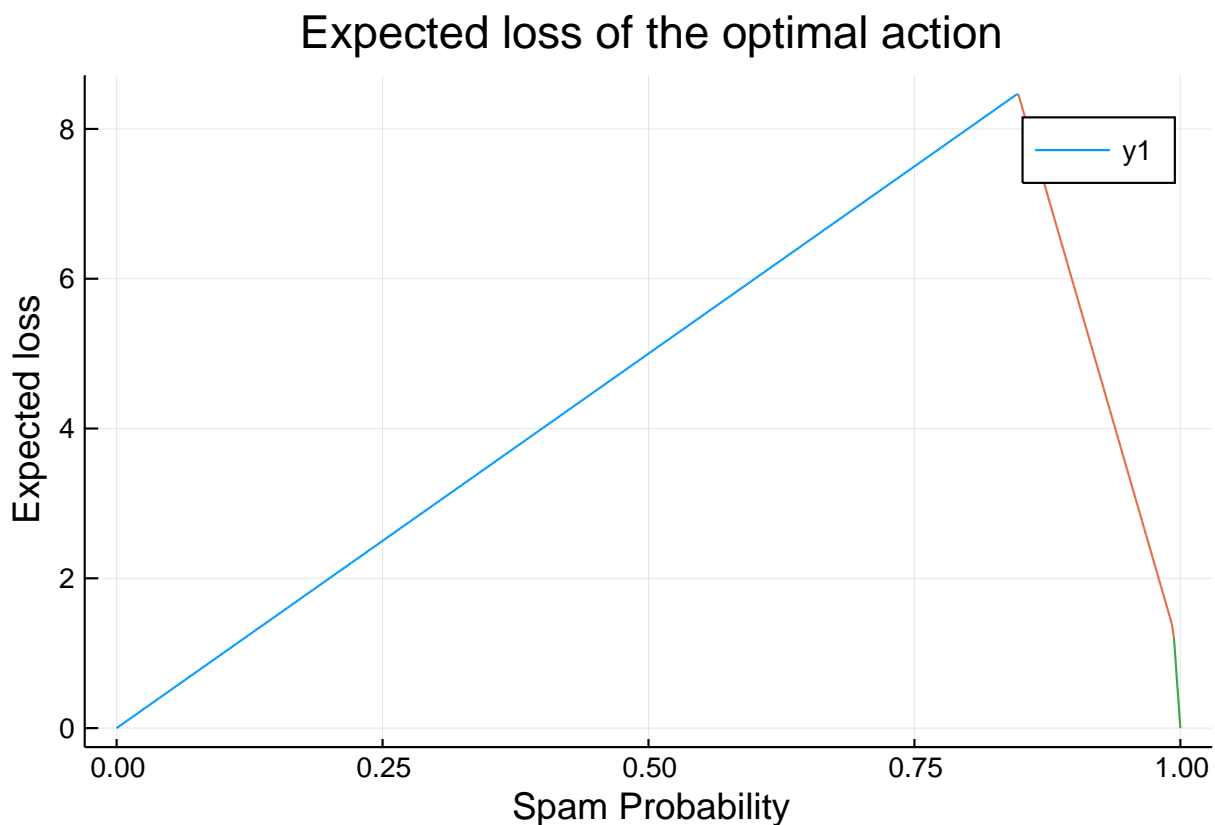
```

3. [4pts] Plot the expected loss of the optimal action as a function of the probability of spam.

Color the line according to the optimal action for that probability of spam.

```
prob_range = range(0, stop=1., length=500)
optimal_losses = []
optimal_actions = []
for p in prob_range
    # TODO: Compute the optimal action and its expected loss for
    # probability of spam given by p.
    append!(optimal_losses, optimal_action(p)[1])
    append!(optimal_actions, optimal_action(p)[2])
end

plot(prob_range, optimal_losses, linecolor=optimal_actions, title="Expected loss of the
optimal action",
xlabel="Spam Probability", ylabel="Expected loss")
```



4. [4pts] For exactly which range of the probabilities of an email being spam should we delete an email?

Find the exact answer by hand using algebra.

We should delete the email when the expected losses is smaller for delete than for folder or show.

That is the probability of spam x satisfy:

$$x \cdot 1 + (1-x) \cdot 50 > (1-x) \cdot 200 \text{ and} \\ x \cdot 10 > (1-x) \cdot 200$$

When $x > \frac{150}{151} \approx 0.9934$, we should delete the email.

2 Regression

2.1 Manually Derived Linear Regression [10pts]

Suppose that $X \in \mathbb{R}^{m \times n}$ with $n \geq m$ and $Y \in \mathbb{R}^n$, and that $Y \sim \mathcal{N}(X^T \beta, \sigma^2 I)$.

In this question you will derive the result that the maximum likelihood estimate $\hat{\beta}$ of β is given by

$$\hat{\beta} = (XX^T)^{-1}XY$$

1. [1pts] What happens if $n < m$?

This means number of observations smaller than number independent variables. $(XX^T)^{-1}$ does not exist. Then we will not have an estimation of β .

2. [2pts] What are the expectation and covariance matrix of $\hat{\beta}$, for a given true value of β ?

Since $\hat{\beta} = (XX^T)^{-1}XY$, and $Y = X^T \beta + \epsilon$,

$$\begin{aligned}\hat{\beta} &= (XX^T)^{-1}X(X^T \beta + \epsilon) \\ &= \beta + (XX^T)^{-1}X\epsilon \\ E[\hat{\beta}] &= E[\beta] + E[(XX^T)^{-1}X\epsilon] \\ &= \beta + (XX^T)^{-1}XE[\epsilon] \\ &= \beta\end{aligned}$$

As we usually assume $E[\epsilon]=0$.

For the covariance matrix, since $\hat{\beta} - \beta = (XX^T)^{-1}X\epsilon$, we have

$$\begin{aligned}E[(\hat{\beta} - \beta)(\hat{\beta} - \beta)^T] &= E[((XX^T)^{-1}X\epsilon)((XX^T)^{-1}X\epsilon)^T] \\ &= E[(XX^T)^{-1}X\epsilon\epsilon^T((XX^T)^{-1}X)^T] \\ &= E[\epsilon\epsilon^T](XX^T)^{-1}X((XX^T)^{-1}X)^T \\ &= \sigma^2 I \\ &= \sigma^2\end{aligned}$$

If we assume X is non-stochastic.

3. [2pts] Show that maximizing the likelihood is equivalent to minimizing the squared error $\sum_{i=1}^n (y_i - x_i \beta)^2$. [Hint: Use $\sum_{i=1}^n a_i^2 = a^T a$]

The log-likelihood is:

$$\begin{aligned}
l(\theta) &:= \log(p(D|\theta)) \\
&= \log\left(\prod_{i=1}^N p(y_i|X_i, \theta)\right) \\
&= \sum_{i=1}^N \log(p(y_i|X_i, \theta))
\end{aligned}$$

Usually use negative log-likelihood(NLL), and normal distribution to expand:

$$\begin{aligned}
NLL(\theta) &= -\sum_{i=1}^N \log(p(y_i|X_i, \theta)) \\
&= -\sum_{i=1}^N \log\left[\left(\frac{1}{2\pi\sigma^2}\right)^{\frac{1}{2}} \exp\left(-\frac{1}{2\sigma^2}(y_i - X_i\beta)^2\right)\right] \\
&= -\sum_{i=1}^N \left(\frac{1}{2}\log\left(\frac{1}{2\pi\sigma^2}\right) - \frac{1}{2\sigma^2}(y_i - X_i\beta)^2\right) \\
&= -\frac{N}{2}\log\left(\frac{1}{2\pi\sigma^2}\right) - \frac{1}{2\sigma^2} \sum_{i=1}^N (y_i - X_i\beta)^2 \\
&= -\frac{N}{2}\log\left(\frac{1}{2\pi\sigma^2}\right) - \frac{1}{2\sigma^2} RSS(\beta)
\end{aligned}$$

Since maximize likelihood is equivalent to minimize NLL, so it is equivalent to minimizing the squared error $RSS = \sum_{i=1}^n (y_i - x_i\beta)^2$.

4. [2pts] Write the squared error in vector notation, (see above hint), expand the expression, and collect like terms. [Hint: Use $\beta^T x^T y = y^T x \beta$ and $x^T x$ is symmetric]

The squared error in vector notation is,

$$\begin{aligned}
\sum_{i=1}^n (y_i - x_i\beta)^2 &= (Y - X^T\beta)^T (Y - X^T\beta) \\
&= Y^T Y - \beta^T X Y - Y X^T \beta + \beta^T X X^T \beta \\
&= Y^T Y - 2\beta^T X Y + \beta^T X X^T \beta
\end{aligned}$$

5. [3pts] Use the likelihood expression to write the negative log-likelihood. Write the derivative of the negative log-likelihood with respect to β , set equal to zero, and solve to show the maximum likelihood estimate $\hat{\beta}$ as above.

As derived in question 3,

$$\begin{aligned}
NLL(\theta) &= -\sum_{i=1}^N \log(p(y_i|X_i, \theta)) \\
&= -\frac{N}{2}\log\left(\frac{1}{2\pi\sigma^2}\right) - \frac{1}{2\sigma^2} RSS(\beta)
\end{aligned}$$

To minimize NNL, is to minimize:

$$\begin{aligned}RSS(\beta) &= (Y - X^T \beta)^T (Y - X^T \beta) \\ &= Y^T Y - 2\beta^T X Y + \beta^T X X^T \beta\end{aligned}$$

Take derivative w.r.t β is

$$\frac{\partial RSS}{\partial \beta} = -2XY + 2(XX^T)\beta$$

Set the derivative to zero,

$$\begin{aligned}-2XY + 2(XX^T)\beta &= 0 \\ \hat{\beta} &= (XX^T)^{-1}XY\end{aligned}$$

2.2 Toy Data [2pts]

For visualization purposes and to minimize computational resources we will work with 1-dimensional toy data.

That is $X \in \mathbb{R}^{m \times n}$ where $m = 1$.

We will learn models for 3 target functions

- `target_f1`, linear trend with constant noise.
- `target_f2`, linear trend with heteroskedastic noise.
- `target_f3`, non-linear trend with heteroskedastic noise.

```
using LinearAlgebra
```

```
function target_f1(x, σ_true=0.3)
    noise = randn(size(x))
    y = 2x .+ σ_true.*noise
    return vec(y)
end
```

```
function target_f2(x)
    noise = randn(size(x))
    y = 2x + norm.(x)*0.3.*noise
    return vec(y)
end
```

```
function target_f3(x)
    noise = randn(size(x))
    y = 2x + 5sin.(0.5*x) + norm.(x)*0.3.*noise
    return vec(y)
end
```

```
target_f3 (generic function with 1 method)
```

1. [1pts] Write a function which produces a batch of data $x \sim \text{Uniform}(0, 20)$ and $y = \text{target_f}(x)$

```
using Random
using Distributions
function sample_batch(target_f, batch_size)
    x = rand(Uniform{Float64, Real}(0, 20), 1, batch_size)
    y = target_f(x)
    return (x,y)
end

sample_batch (generic function with 1 method)

using Test
@testset "sample dimensions are correct" begin
    m = 1 # dimensionality
    n = 200 # batch-size
    for target_f in (target_f1, target_f2, target_f3)
        x,y = sample_batch(target_f,n)
        @test size(x) == (m,n)
        @test size(y) == (n,)
    end
end

Test Summary:                               | Pass  Total
sample dimensions are correct |      6      6
Test.DefaultTestSet("sample dimensions are correct", Any[], 6, false)
```

2. [1pts] For all three targets, plot a $n = 1000$ sample of the data. **Note: You will use these plots later, in your writeup display once other questions are complete.**

```
using Plots
s1=sample_batch(target_f1,1000)
x1=s1[1]
y1=s1[2]
plot_f1 = plot(x1',y1,seriestype=:scatter,title="target_f1 Plot",xlabel="x1",
ylabel="y1")

s2=sample_batch(target_f2,1000)
x2=s2[1]
y2=s2[2]
plot_f2 = plot(x2',y2,seriestype=:scatter,title="target_f2 Plot",xlabel="x2",
ylabel="y2")

s3=sample_batch(target_f3,1000)
x3=s3[1]
y3=s3[2]
plot_f3 = plot(x3',y3,seriestype=:scatter,title="target_f3 Plot",xlabel="x3",
ylabel="y3");
```

2.3 Linear Regression Model with $\hat{\beta}$ MLE [4pts]

1. [2pts] Program the function that computes the the maximum likelihood estimate given X and Y . Use it to compute the estimate $\hat{\beta}$ for a $n = 1000$ sample from each target function.

Since in the question we assume $m=1$, the formula for regression is: $Y_i = \beta_1 x_i + \beta_0$ then every β parameter is a array of two elements, denoting slope and intercept of the regression line.

```
using LinearAlgebra
function beta_mle(X,Y)
    beta = inv(X*X')*X*Y
    return beta
end

n=1000 # batch_size

#Use the same sample data that used to create plot_f1 above
x1=s1[1]
y1=s1[2]
x0=ones(1,n) #add a column of "1"s to multiply with  $\beta_0$ , the intercept of regression line
X1=vcat(x0,x1)
 $\beta_{mle\_1}$  = beta_mle(X1,y1)

#Use the same sample data that used to create plot_f2 above
x2=s2[1]
y2=s2[2]
x0=ones(1,n)
X2=vcat(x0,x2)
 $\beta_{mle\_2}$  = beta_mle(X2,y2)

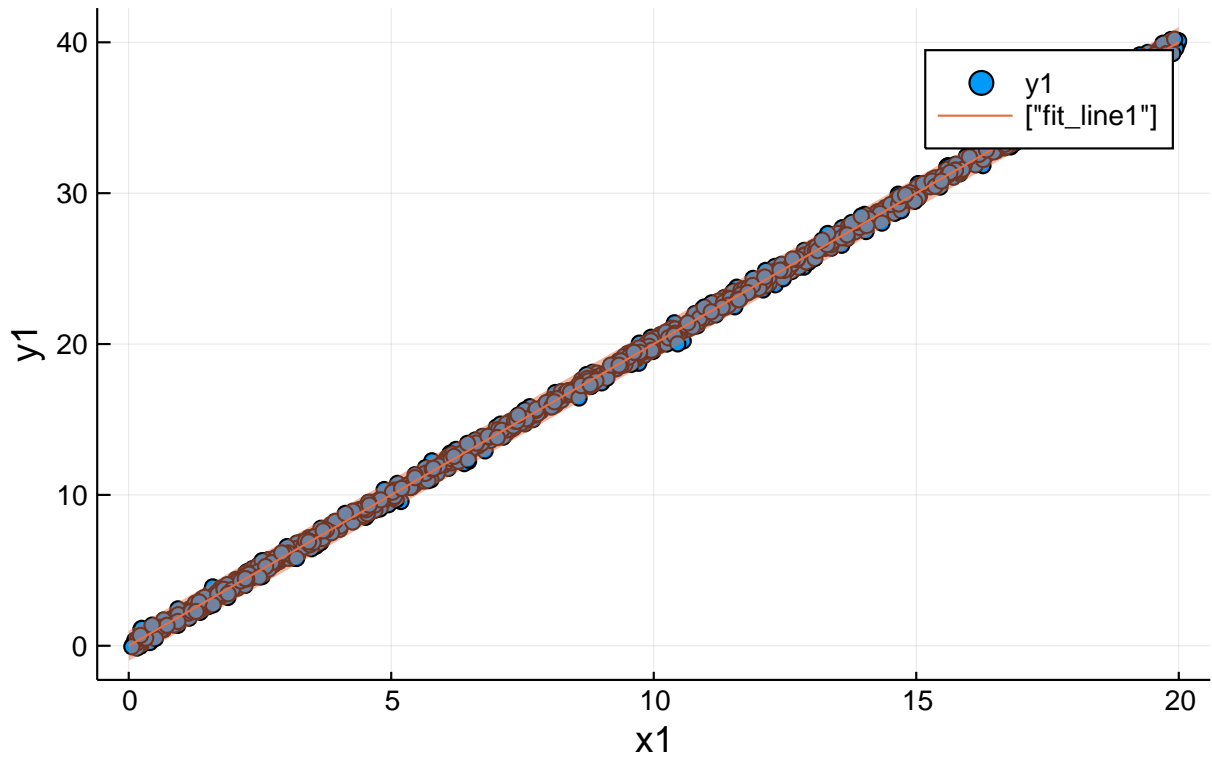
#Use the same sample data that used to create plot_f3 above
x3=s3[1]
y3=s3[2]
x0=ones(1,n)
X3=vcat(x0,x3)
 $\beta_{mle\_3}$  = beta_mle(X3,y3)

2-element Array{Float64,1}:
 0.9906697149123878
 1.9942681718013897
```

2. [2pts] For each function, plot the linear regression model given by $Y \sim \mathcal{N}(X^T \hat{\beta}, \sigma^2 I)$ for $\sigma = 1$. This plot should have the line of best fit given by the maximum likelihood estimate, as well as a shaded region around the line corresponding to plus/minus one standard deviation (i.e. the fixed uncertainty $\sigma = 1.0$). Using `Plots.jl` this shaded uncertainty region can be achieved with the `ribbon` keyword argument. **Display 3 plots, one for each target function, showing samples of data and maximum likelihood estimate linear regression model**

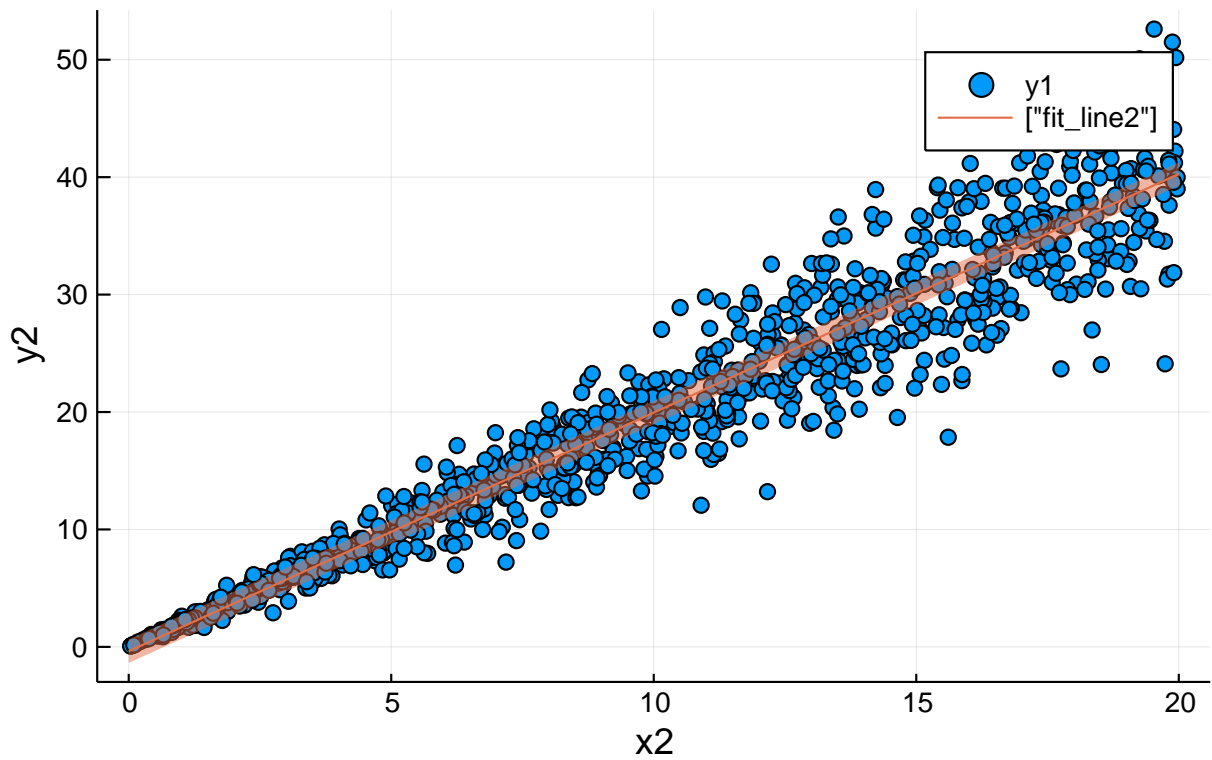
```
x_range = range(0., stop=20., length=1000)
fit_line1= $\beta_{mle\_1}$ [2] .*x_range .+  $\beta_{mle\_1}$ [1]
plot!(plot_f1,x_range,fit_line1,ribbon=1.0,label=["fit_line1"],title="target f1 Plot with regression line",xlabel="x1",ylabel="y1")
```


target f1 Plot with regression line



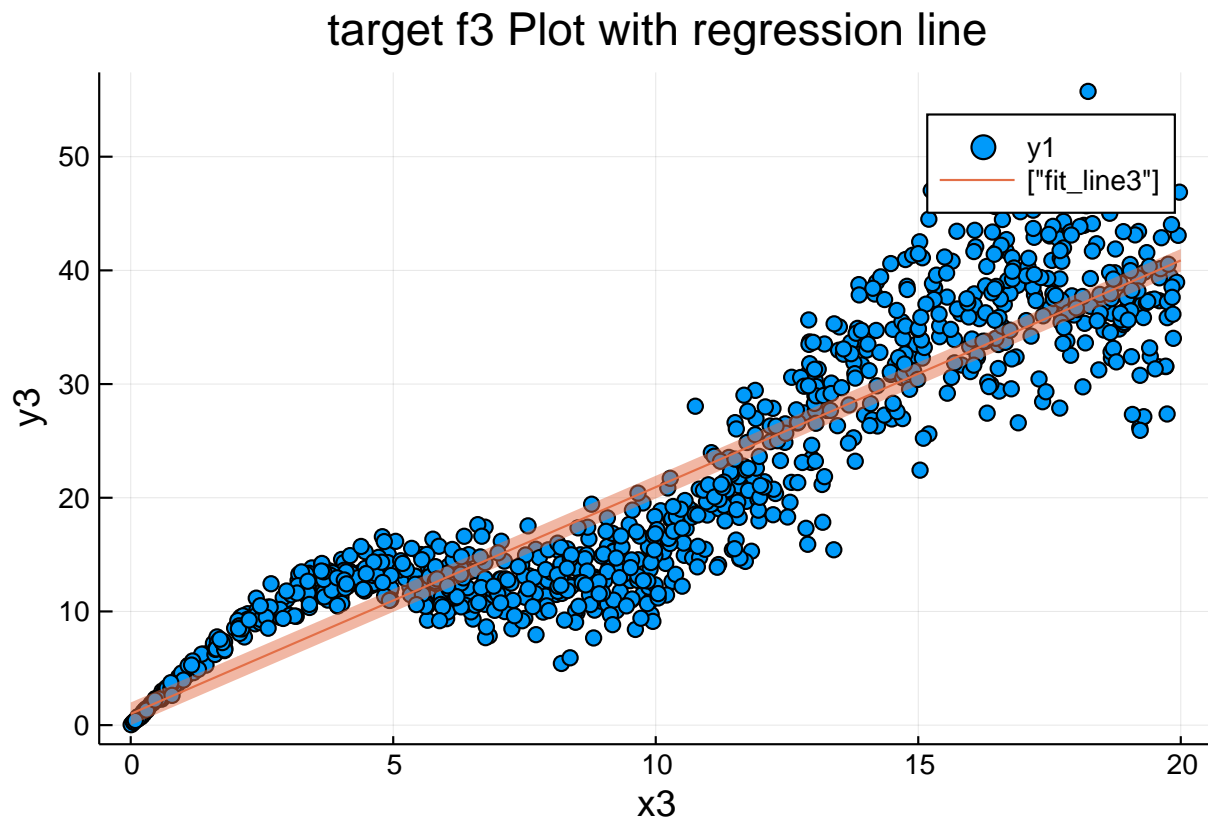
```
fit_line2= $\beta_{mle\_2}[2]$  .*x_range .+  $\beta_{mle\_2}[1]$ 
plot!(plot_f2,x_range,fit_line2,ribbon=1,label=["fit_line2"],title="target f2 Plot with
regression line",xlabel="x2"
,ylabel="y2")
```

target f2 Plot with regression line



```
fit_line3= $\beta_{mle\_3}[2]$  .*x_range .+  $\beta_{mle\_3}[1]$ 
```

```
plot!(plot_f3,x_range,fit_line3,ribbon=1.0,label=["fit_line3"],title="target f3 Plot with
regression line",xlabel="x3",
ylabel="y3")
```



2.4 Log-likelihood of Data Under Model [6pts]

1. [2pts] Write code for the function that computes the likelihood of x under the Gaussian distribution $\mathcal{N}(\mu, \sigma)$. For reasons that will be clear later, this function should be able to broadcast to the case where x, μ, σ are all vector valued and return a vector of likelihoods with equivalent length, i.e., $x_i \sim \mathcal{N}(\mu_i, \sigma_i)$.

```
function gaussian_log_likelihood( $\mu$ ,  $\sigma$ , x)
    """
    compute log-likelihood of x under  $\mathcal{N}(\mu, \sigma)$ 
    """
    ll= log.(1 ./ (sqrt(2*pi) .*  $\sigma$ )) .+ (-1 ./ (2 .*  $\sigma$ .^2) .* (x .-  $\mu$ ).^2)
    return ll
end
```

```
gaussian_log_likelihood (generic function with 1 method)
```

```
# Test Gaussian likelihood against standard implementation
@testset "Gaussian log likelihood" begin
    using Distributions: pdf, Normal
    # Scalar mean and variance
    x = randn()
     $\mu$  = randn()
     $\sigma$  = rand()
```

```

@test size(gaussian_log_likelihood( $\mu$ , $\sigma$ , $x$ )) == () # Scalar log-likelihood
@test gaussian_log_likelihood( $\mu$ , $\sigma$ , $x$ )  $\approx$  log.(pdf.(Normal( $\mu$ , $\sigma$ ), $x$ )) # Correct Value

# Vector valued x under constant mean and variance
x = randn(100)
 $\mu$  = randn()
 $\sigma$  = rand()
@test size(gaussian_log_likelihood( $\mu$ , $\sigma$ , $x$ )) == (100,) # Vector of log-likelihoods
@test gaussian_log_likelihood( $\mu$ , $\sigma$ , $x$ )  $\approx$  log.(pdf.(Normal( $\mu$ , $\sigma$ ), $x$ )) # Correct Values

# Vector valued x under vector valued mean and variance
x = randn(10)
 $\mu$  = randn(10)
 $\sigma$  = rand(10)
@test size(gaussian_log_likelihood( $\mu$ , $\sigma$ , $x$ )) == (10,) # Vector of log-likelihoods
@test gaussian_log_likelihood( $\mu$ , $\sigma$ , $x$ )  $\approx$  log.(pdf.(Normal( $\mu$ , $\sigma$ ), $x$ )) # Correct Values
end

Test Summary:          | Pass  Total
Gaussian log likelihood |    6     6
Test.DefaultTestSet("Gaussian log likelihood", Any[], 6, false)

```

2. [2pts] Use your gaussian log-likelihood function to write the code which computes the negative log-likelihood of the target value Y under the model $Y \sim \mathcal{N}(X^T \beta, \sigma^2 * I)$ for a given value of β .

Use the previous gaussian log-likelihood function, the negative log-likelihood function is written as below,

```

function lr_model_nll( $\beta$ , $x$ , $y$ ;  $\sigma=1$ .)
    NLL= -sum(gaussian_log_likelihood( $x'$ * $\beta$ , $\sigma$ , $y$ ))
end

lr_model_nll (generic function with 1 method)

```

3. [1pts] Use this function to compute and report the negative-log-likelihood of a $n \in \{10, 100, 1000\}$ batch of data under the model with the maximum-likelihood estimate $\hat{\beta}$ and $\sigma \in \{0.1, 0.3, 1., 2.\}$ for each target function.

```

for n in (10,100,1000)
    println("----- $n -----")
    for target_f in (target_f1,target_f2, target_f3)
        println("----- $target_f -----")
        for  $\sigma_{\text{model}}$  in (0.1,0.3,1.,2.)
            println("-----  $\sigma_{\text{model}}$  -----")
            x,y = sample_batch(target_f,n)
             $\beta_{\text{mle}}$  = beta_mle(x,y)
            nll = lr_model_nll( $\beta_{\text{mle}}$ , $x$ , $y$ ;  $\sigma=\sigma_{\text{model}}$ )
            println("Negative Log-Likelihood: $nll")
        end
    end
end
end

```

```

----- 10 -----
----- target_f1 -----
----- 0.1 -----
Negative Log-Likelihood: 32.82047139405386
----- 0.3 -----
Negative Log-Likelihood: 3.7616647657692517
----- 1.0 -----
Negative Log-Likelihood: 9.315589196639227
----- 2.0 -----
Negative Log-Likelihood: 16.192954695816802
----- target_f2 -----
----- 0.1 -----
Negative Log-Likelihood: 1398.4204765577051
----- 0.3 -----
Negative Log-Likelihood: 427.44127362663835
----- 1.0 -----
Negative Log-Likelihood: 43.86858564343372
----- 2.0 -----
Negative Log-Likelihood: 33.35182084432413
----- target_f3 -----
----- 0.1 -----
Negative Log-Likelihood: 11095.796501543544
----- 0.3 -----
Negative Log-Likelihood: 715.4447037080137
----- 1.0 -----
Negative Log-Likelihood: 105.06446839188533
----- 2.0 -----
Negative Log-Likelihood: 34.27108117665104
----- 100 -----
----- target_f1 -----
----- 0.1 -----
Negative Log-Likelihood: 362.90418614355934
----- 0.3 -----
Negative Log-Likelihood: 18.284348428149045
----- 1.0 -----
Negative Log-Likelihood: 96.6049936810737
----- 2.0 -----
Negative Log-Likelihood: 162.27973757667678
----- target_f2 -----
----- 0.1 -----
Negative Log-Likelihood: 71658.61927116942
----- 0.3 -----
Negative Log-Likelihood: 6585.207647459448
----- 1.0 -----
Negative Log-Likelihood: 565.6850276195437
----- 2.0 -----
Negative Log-Likelihood: 419.9244886718769
----- target_f3 -----
----- 0.1 -----
Negative Log-Likelihood: 127745.28070269336
----- 0.3 -----
Negative Log-Likelihood: 13392.697413328717
----- 1.0 -----
Negative Log-Likelihood: 1432.7329216104722
----- 2.0 -----
Negative Log-Likelihood: 384.79039711360144
----- 1000 -----
----- target_f1 -----
----- 0.1 -----

```

```

Negative Log-Likelihood: 3195.10462891413
----- 0.3 -----
Negative Log-Likelihood: 244.34998996863402
----- 1.0 -----
Negative Log-Likelihood: 963.1971545492812
----- 2.0 -----
Negative Log-Likelihood: 1623.8381984619457
----- target_f2 -----
----- 0.1 -----
Negative Log-Likelihood: 611233.8720929804
----- 0.3 -----
Negative Log-Likelihood: 60255.311766616425
----- 1.0 -----
Negative Log-Likelihood: 7224.891192916572
----- 2.0 -----
Negative Log-Likelihood: 3056.9787746659044
----- target_f3 -----
----- 0.1 -----
Negative Log-Likelihood: 1.102868118245692e6
----- 0.3 -----
Negative Log-Likelihood: 136181.4662935136
----- 1.0 -----
Negative Log-Likelihood: 11939.78878353474
----- 2.0 -----
Negative Log-Likelihood: 4246.136373196782

```

4. [1pts] For each target function, what is the best choice of σ ?

From the printed info, we can see in most cases, NLL numebrs when $\sigma=0.1$ are not stable, for example there is a huge increase of NLL for target f3 when $\sigma=0.1$ changed to $\sigma=0.3$ and $n=1000$. This huge increase does not happen when $n=10$ and $n=100$. As expected too small values of σ have precision issues. Especially for target f3, which has bigger variance, small σ choice will not reflect the true variance of the distribution.

To choose the right σ , maybe better to look at data with bigger batch size.

For target f3, $\sigma=0.1$ data not precise, $\sigma=0.3$ and $\sigma=1$ NLL number are close, $\sigma=2$ NLL also not very stable for example values drop too much as $n=1000$ and $n=100$. So choose $\sigma=1$ for better precision.

For target f2, situation more or less similar to target.f3. For example big drop of NLL from $\sigma=0.1$ to 0.3 when $n=10$, $n=100$ and $n=1000$, this may results from both values not stable. Also choose $\sigma=1$.

For target f1, the ture distribution known to be have small variance, and this reflected in the result, values across all n values relatively stable. Choose $\sigma=1$ or 0.3 not very different. We can choose either one.

2.5 Automatic Differentiation and Maximizing Likelihood [3pts]

In a previous question you derived the expression for the derivative of the negative log-likelihood with respect to β . We will use that to test the gradients produced by automatic differentiation.

1. [3pts] For a random value of β , σ , and $n = 100$ sample from a target function, use automatic differentiation to compute the derivative of the negative log-likelihood of the sampled data with respect β . Test that this is equivalent to the hand-derived value.

```
using Zygote: gradient

@testset "Gradients wrt parameter" begin
   $\beta_{\text{test}}$  = randn()
   $\sigma_{\text{test}}$  = rand()
  x,y = sample_batch(target_f1,100)
  f( $\beta$ )=lr_model_nll( $\beta$ ,x,y; $\sigma$ = $\sigma_{\text{test}}$ )[1]
  ad_grad = gradient(f, $\beta_{\text{test}}$ )
  hand_derivative = 1/(2* $\sigma_{\text{test}}$ ^2) .*(-2 .*x*y .+2 .*(x*x')* $\beta_{\text{test}}$ )[1]
  @test ad_grad[1]  $\approx$  hand_derivative
end

Test Summary:          | Pass  Total
Gradients wrt parameter |      1      1
Test.DefaultTestSet("Gradients wrt parameter", Any[], 1, false)
```

2.5.1 Train Linear Regression Model with Gradient Descent [5pts]

In this question we will compute gradients of of negative log-likelihood with respect to β . We will use gradient descent to find β that maximizes the likelihood.

1. [3pts] Write a function `train_lin_reg` that accepts a target function and an initial estimate for β and some hyperparameters for batch-size, model variance, learning rate, and number of iterations. Then, for each iteration:
 - sample data from the target function
 - compute gradients of negative log-likelihood with respect to β
 - update the estimate of β with gradient descent with specified learning rate

and, after all iterations, returns the final estimate of β .

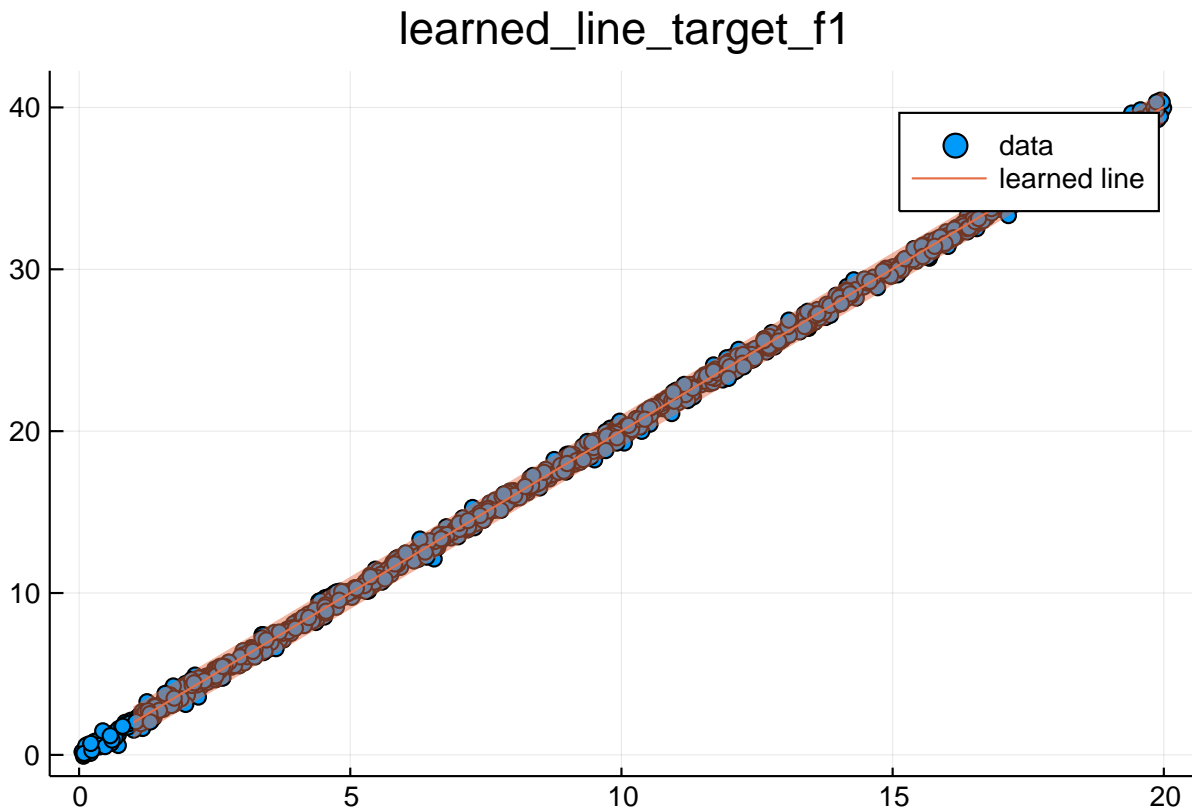
```
using Logging # Print training progress to REPL, not pdf

function train_lin_reg(target_f,  $\beta_{\text{init}}$ ; bs= 100, lr = 1e-6, iters=1000,  $\sigma_{\text{model}}$  = 1. )
   $\beta_{\text{curr}}$  =  $\beta_{\text{init}}$ 
  for i in 1:iters
    x,y = sample_batch(target_f,bs)
    NLL= lr_model_nll( $\beta_{\text{curr}}$ ,x,y; $\sigma$ = $\sigma_{\text{model}}$ )
    f( $\beta$ )=lr_model_nll( $\beta$ ,x,y; $\sigma$ = $\sigma_{\text{model}}$ )[1]
    grad_ $\beta$  = gradient(f, $\beta_{\text{curr}}$ )[1]
    #@info "loss: $(NLL)   $\beta$ :  $\beta_{\text{curr}}$  grad:$grad_ $\beta$  " #TODO: log loss, if you want
    to monitor training progress
     $\beta_{\text{curr}}$  =  $\beta_{\text{curr}}$  - grad_ $\beta$  * lr
  end
  return  $\beta_{\text{curr}}$ 
end

train_lin_reg (generic function with 1 method)
```

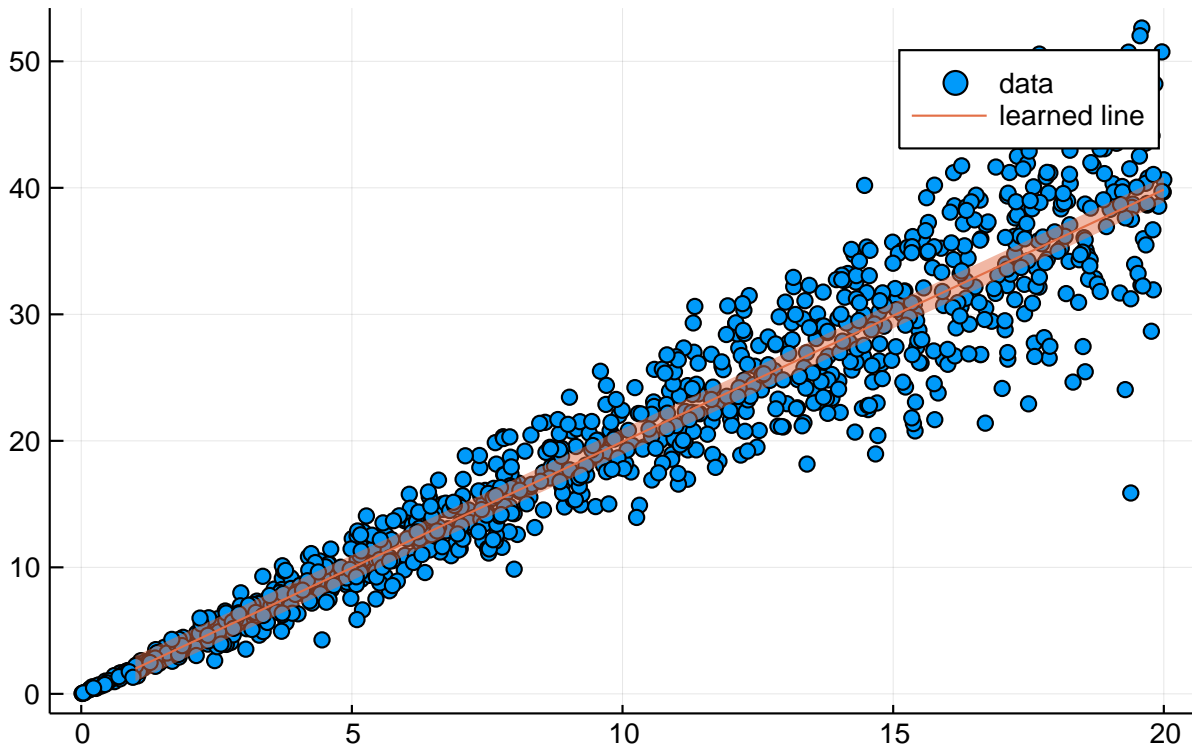
2. [2pts] For each target function, start with an initial parameter β , learn an estimate for β_{learned} by gradient descent. Then plot a $n = 1000$ sample of the data and the learned linear regression model with shaded region for uncertainty corresponding to plus/minus one standard deviation.

```
# Plot for learned regression for target_f1
β_init = 1000*randn() # Initial parameter
β1_learned= train_lin_reg(target_f1, β_init; bs= 100, lr = 1e-6, iters=1000, σ_model = 1.
)
x_range=1:20
learned_line1 = β1_learned .* x_range
x,y = sample_batch(target_f1,1000)
plot11 = plot(x',y,title="learned_line_target_f1",seriestype=:scatter,label="data")
display(plot!(plot11,learned_line1,label="learned line",ribbon=1))
```

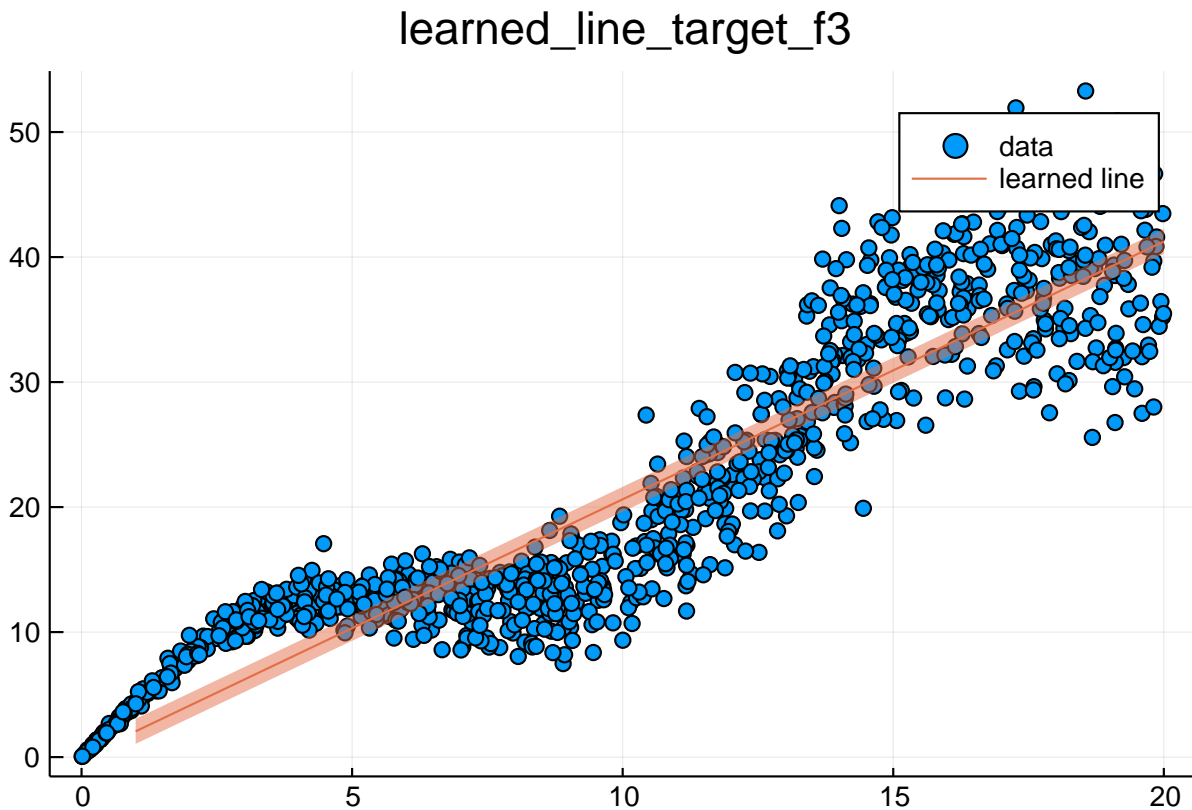


```
#Plot for learned regression for target_f2
β_init2 = 1000*randn()
β2_learned = train_lin_reg(target_f2, β_init2; bs= 100, lr = 1e-6, iters=1000, σ_model =
1. )
learned_line2 = β2_learned .* x_range
x,y = sample_batch(target_f2,1000)
plot12 = plot(x',y,title="learned_line2_target_f2",seriestype=:scatter,label="data")
display(plot!(plot12,learned_line2,ribbon=1,label="learned line"))
```

learned_line2_target_f2



```
#Plot for learned regression for target_f3
 $\beta_{init3}$  = 1000*randn()
 $\beta_{3\_learned}$  = train_lin_reg(target_f3,  $\beta_{init3}$ ; bs= 100, lr = 1e-6, iters=1000,  $\sigma_{model}$  =
1. )
learned_line3 =  $\beta_{3\_learned}$  .* x_range
x,y = sample_batch(target_f3,1000)
plotl3 = plot(x',y,title="learned_line_target_f3",seriestype=:scatter,label="data")
display(plot!(plotl3,learned_line3,ribbon=1,label="learned line"))
```

2.5.2 Non-linear Regression with a Neural Network [9pts]

In the previous questions we have considered a linear regression model

$$Y \sim \mathcal{N}(X^T \beta, \sigma^2)$$

This model specified the mean of the predictive distribution for each datapoint by the product of that datapoint with our parameter.

Now, let us generalize this to consider a model where the mean of the predictive distribution is a non-linear function of each datapoint. We will have our non-linear model be a simple function called `neural_net` with parameters θ (collection of weights and biases).

$$Y \sim \mathcal{N}(\text{neural_net}(X, \theta), \sigma^2)$$

1. [3pts] Write the code for a fully-connected neural network (multi-layer perceptron) with one 10-dimensional hidden layer and a `tanh` nonlinearity. You must write this yourself using only basic operations like matrix multiply and `tanh`, you may not use layers provided by a library.

This network will output the mean vector, test that it outputs the correct shape for some random parameters.

```
function neural_net(x,θ)
    h=tanh.(θ.w1'*x .+ θ.b1)
    z=θ.w2'*h .+ θ.b2
    return vec(z)
end
```

```

# Random initial Parameters
w1=30*randn(1,10)
b1=10*randn(10,1)
w2=30*randn(10,1)
b2=10*randn()
 $\theta$  = (w1=w1, b1=b1, w2=w2, b2=b2)

@testset "neural net mean vector output" begin
n = 100
x,y = sample_batch(target_f1,n)
 $\mu$  = neural_net(x, $\theta$ )
@test size( $\mu$ ) == (n,)
end

Test Summary:                               | Pass  Total
neural net mean vector output |      1      1
Test.DefaultTestSet("neural net mean vector output", Any[], 1, false)

```

2. [2pts] Write the code that computes the negative log-likelihood for this model where the mean is given by the output of the neural network and $\sigma = 1.0$

```

function nn_model_nll( $\theta$ ,x,y; $\sigma$ =1)
    return sum(-gaussian_log_likelihood( $\theta$ , $\sigma$ ,y))
end

```

nn_model_nll (generic function with 1 method)

3. [2pts] Write a function `train_nn_reg` that accepts a target function and an initial estimate for θ and some hyperparameters for batch-size, model variance, learning rate, and number of iterations. Then, for each iteration:

- sample data from the target function
- compute gradients of negative log-likelihood with respect to θ
- update the estimate of θ with gradient descent with specified learning rate

and, after all iterations, returns the final estimate of θ .

```

using Logging # Print training progress to REPL, not pdf

function train_nn_reg(target_f,  $\theta$ _init; bs= 100, lr = 1e-4, iters=1000,  $\sigma$ _model = 1. )
     $\theta$ _curr =  $\theta$ _init
    w1=1/100*randn(1,10)
    b1=2/100*randn(10,1)
    w2=1/100*randn(10,1)
    b2=2/100*randn()
    theta= (w1=w1, b1=b1, w2=w2, b2=b2)
    for i in 1:iters
        x,y = sample_batch(target_f, bs)
         $\mu$ =neural_net(x,theta)
        NLL= nn_model_nll( $\mu$ ,x,y; $\sigma$ = $\sigma$ _model)
        f( $\theta$ )=nn_model_nll( $\theta$ ,x,y; $\sigma$ = $\sigma$ _model)
        grad_theta = gradient(f, $\theta$ _curr)[1]
    end
end

```

```

    #@info "loss: $(NLL),  $\theta$ :  $\theta_{curr}$  grad:  $\text{grad}_\theta$ " #TODO: log loss, if you want
    to monitor training
     $\theta_{curr} = \theta_{curr} - \text{grad}_\theta * \text{lr}$ 
end
return  $\theta_{curr}$ 
end

train_nn_reg (generic function with 1 method)

```

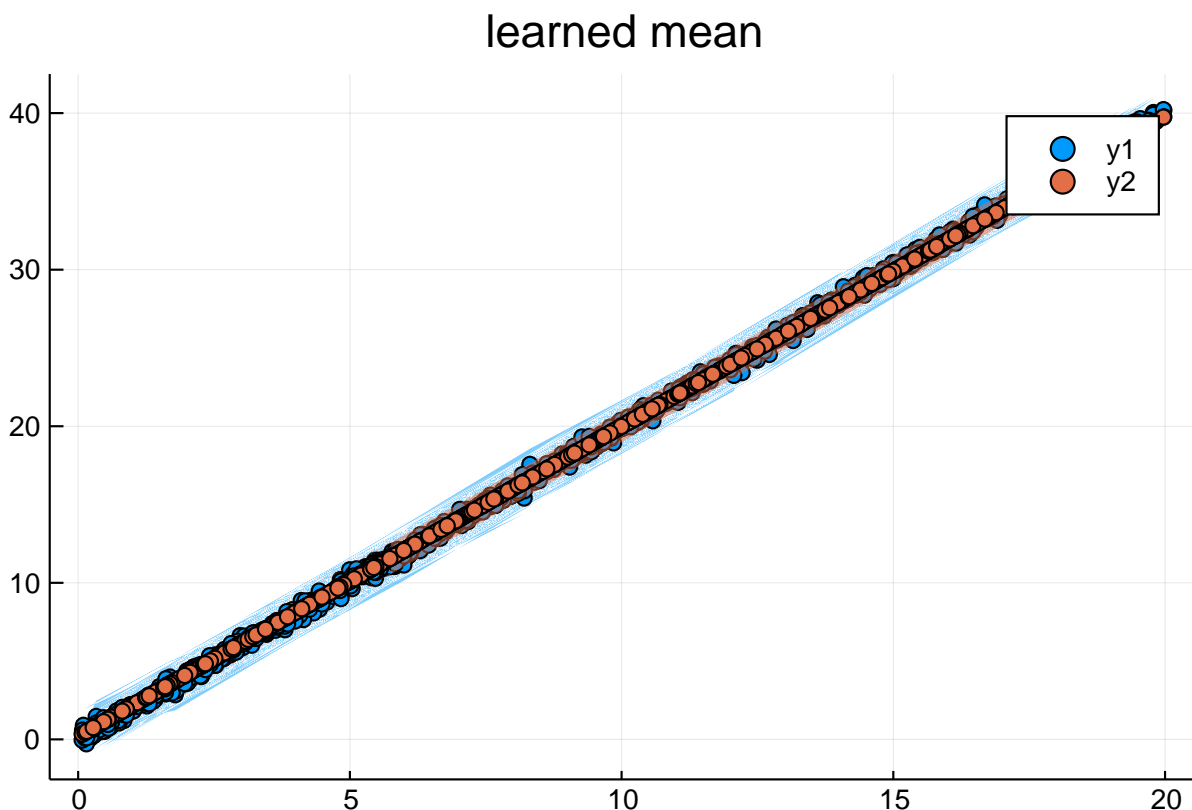
4. [2pts] For each target function, start with an initialization of the network parameters, θ , use your train function to minimize the negative log-likelihood and find an estimate for θ_{learned} by gradient descent. Then plot a $n = 1000$ sample of the data and the learned regression model with shaded uncertainty bounds given by $\sigma = 1.0$

The correct way to set θ_{init} is use neural network to generate initial mean, but as the initial value unpredictable, a lot of NaN values produced for NLL, some times, the process will not generate any result at all.

```

x,y = sample_batch(target_f1,1000)
 $\theta_{\text{init}} = 2 \cdot \text{vec}(x)$ 
 $\theta_{1\_learned} = \text{train\_nn\_reg}(\text{target\_f1}, \theta_{\text{init}}; \text{bs} = 1000, \text{lr} = 1\text{e-}5, \text{iters} = 1000, \sigma_{\text{model}} = 1.)$ 
)
display(plot(x',hcat(y, $\theta_{1\_learned}$ ),ribbon=1,title="learned mean",seriestype=:scatter))

```

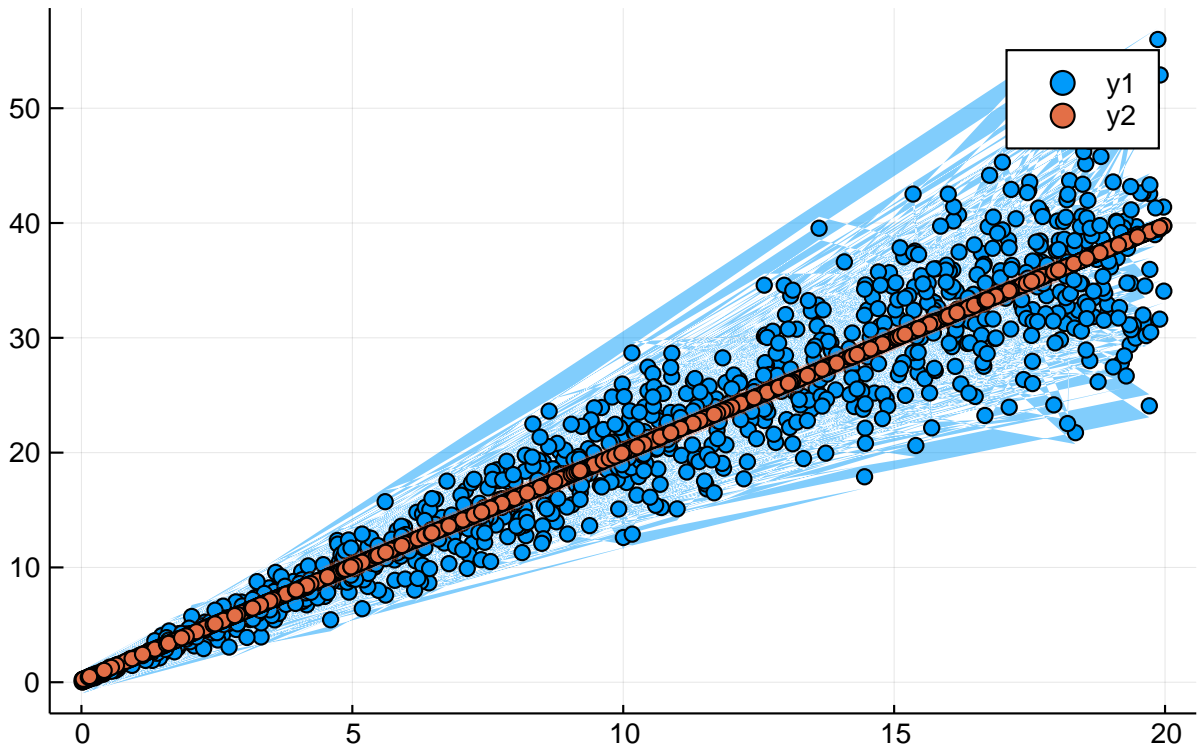


```

x,y = sample_batch(target_f2,1000)
 $\theta_{\text{init}} = 2 \cdot \text{vec}(x)$ 
 $\theta_{2\_learned} = \text{train\_nn\_reg}(\text{target\_f2}, \theta_{\text{init}}; \text{bs} = 1000, \text{lr} = 1\text{e-}5, \text{iters} = 1000, \sigma_{\text{model}} = 1.)$ 
)
display(plot(x',hcat(y, $\theta_{2\_learned}$ ),ribbon=1,title="learned mean2",seriestype=:scatter))

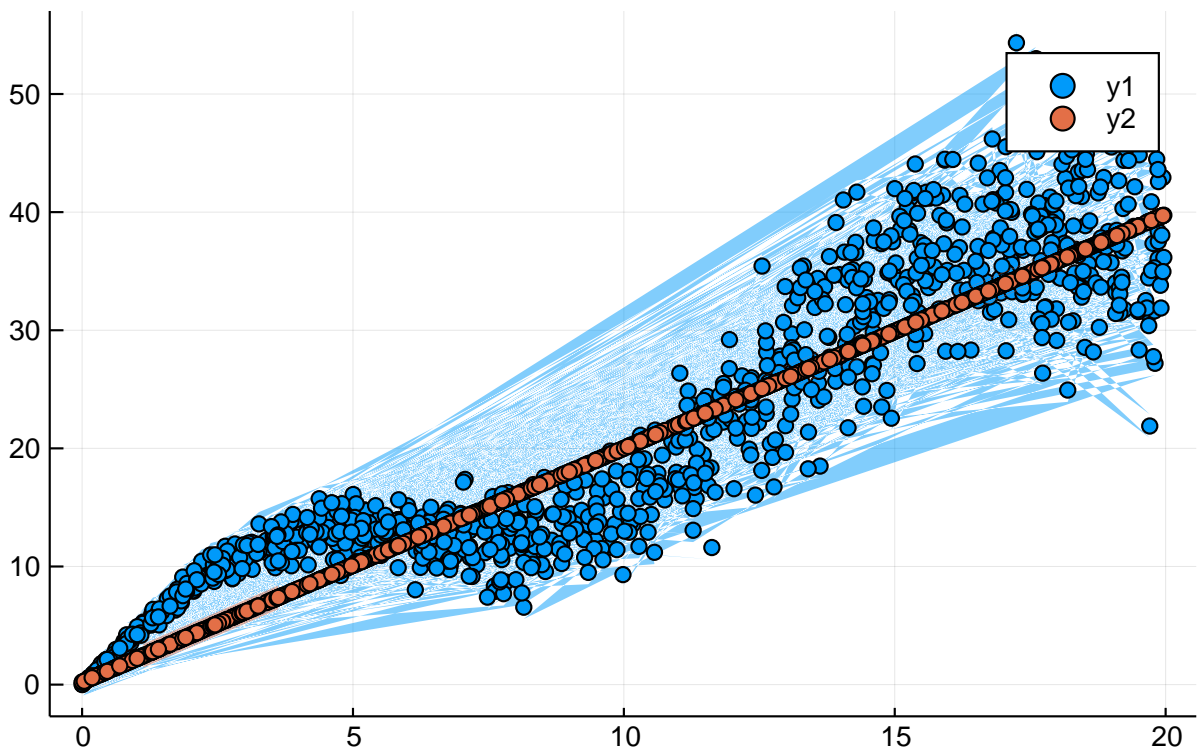
```

learned mean2



```
x,y = sample_batch(target_f3,1000)
theta_init=2.*vec(x)
theta3_learned = train_nn_reg(target_f3, theta_init; bs= 1000, lr = 1e-5, iters=1000, sigma_model = 1.
)
plott3 = plot(x',hcat(y,theta3_learned),ribbon=1,title="learned mean3",seriestype=:scatter)
```

learned mean3



2.5.3 Non-linear Regression and Input-dependent Variance with a Neural Network [8pts]

In the previous questions we've gone from a gaussian model with mean given by linear combination

$$Y \sim \mathcal{N}(X^T \beta, \sigma^2)$$

to gaussian model with mean given by non-linear function of the data (neural network)

$$Y \sim \mathcal{N}(\text{neural_net}(X, \theta), \sigma^2)$$

However, in all cases we have considered so far, we specify a fixed variance for our model distribution. We know that two of our target datasets have heteroscedastic noise, meaning any fixed choice of variance will poorly model the data.

In this question we will use a neural network to learn both the mean and log-variance of our gaussian model.

$$\begin{aligned} \mu, \log \sigma &= \text{neural_net}(X, \theta) \\ Y &\sim \mathcal{N}(\mu, \exp(\log \sigma)^2) \end{aligned}$$

1. [1pts] Write the code for a fully-connected neural network (multi-layer perceptron) with one 10-dimensional hidden layer and a `tanh` nonlinearity, and outputs both a vector for mean and $\log \sigma$. Test the output shape is as expected.

```
# Neural Network Function
function neural_net_w_var(x,theta)
    h = tanh.(theta.w1'*x .+ theta.b1)
    z = theta.w2'*h .+ theta.b2
    mu = vec(z ./size(x)[2])
    logsigma = log.(1 ./ (size(x)[2]-1) .* (vec(x)-mu).^2)
    return mu, logsigma
end

# Random initial Parameters
w1=4*randn(1,10)
b1=1*randn(10,1)
w2=4*randn(10,1)
b2=1*randn()
theta= (w1=w1, b1=b1, w2=w2, b2=b2)

@testset "neural net mean and logsigma vector output" begin
    n = 100
    x,y = sample_batch(target_f1,n)
    mu, logsigma = neural_net_w_var(x,theta)
    @test size(mu) == (n,)
    @test size(logsigma) == (n,)
end
```

```
Test Summary: | Pass Total
neural net mean and logsigma vector output |    2    2
Test.DefaultTestSet("neural net mean and logsigma vector output", Any[], 2,
false)
```

2. [2pts] Write the code that computes the negative log-likelihood for this model where the mean and $\log \sigma$ is given by the output of the neural network. (Hint: Don't forget to take $\exp \log \sigma$)

```
function nn_with_var_model_nll(theta,x,y)
    mu=theta[1]
    sigma=exp.(theta[2])
    return sum(-gaussian_log_likelihood(mu,sigma,y))
end

nn_with_var_model_nll (generic function with 1 method)
```

3. [1pts] Write a function `train_nn_w_var_reg` that accepts a target function and an initial estimate for θ and some hyperparameters for batch-size, learning rate, and number of iterations. Then, for each iteration:

- sample data from the target function
- compute gradients of negative log-likelihood with respect to θ
- update the estimate of θ with gradient descent with specified learning rate

and, after all iterations, returns the final estimate of θ .

```
function train_nn_w_var_reg(target_f, theta_init; bs= 100, lr = 1e-4, iters=10000)
    theta_curr = theta_init
    w1=4*randn(1,10)
    b1=1*randn(10,1)
    w2=4*randn(10,1)
    b2=1*randn()
    theta= (w1=w1, b1=b1, w2=w2, b2=b2)
    for i in 1:iters
        x,y = sample_batch(target_f,bs)
        theta = neural_net_w_var(x,theta)
        NLL1= nn_with_var_model_nll(theta,x,y)
        f(theta)=nn_with_var_model_nll(theta,x,y)
        #@info "loss: $(NLL), theta: $theta_curr, grad: $grad_theta "
        grad_theta = gradient(f, theta_curr)[1]
        theta_curr = theta_curr .- grad_theta .*lr
    end
    return theta_curr
end

train_nn_w_var_reg (generic function with 1 method)
```

4. [4pts] For each target function, start with an initialization of the network parameters, θ , learn an estimate for θ_{learned} by gradient descent. Then plot a $n = 1000$ sample of the dataset and the learned regression model with shaded uncertainty bounds corresponding to plus/minus one standard deviation given by the variance of the predictive distribution at each input location (output by the neural network). (Hint: `ribbon` argument for shaded uncertainty bounds can accept a vector of σ)

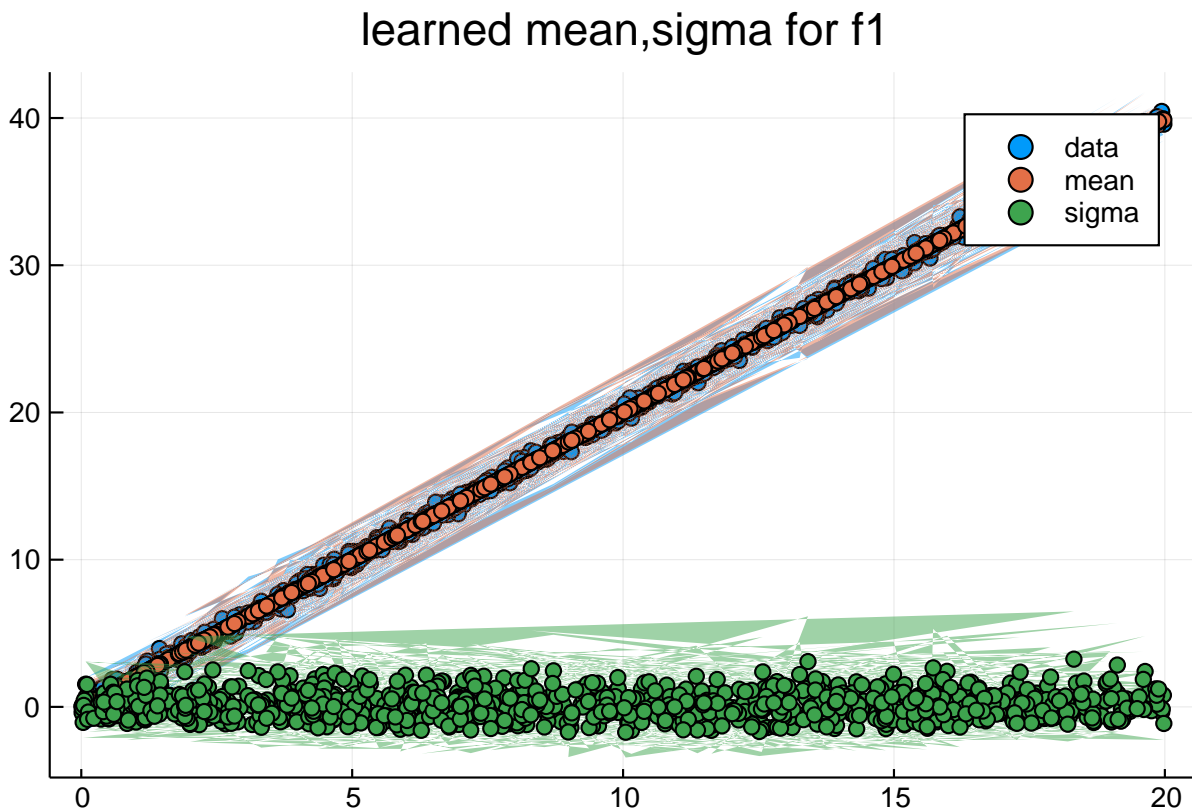
Note: Learning the variance is tricky, and this may be unstable during training. There are some things you can try:

- Adjusting the hyperparameters like learning rate and batch size
- Train for more iterations
- Try a different random initialization, like sample random weights and bias matrices with lower variance.

For this question **you will not be assessed on the final quality of your model**. Specifically, if you fails to train an optimal model for the data that is okay. You are expected to learn something that is somewhat reasonable, and **demonstrates that this model is training and learning variance**.

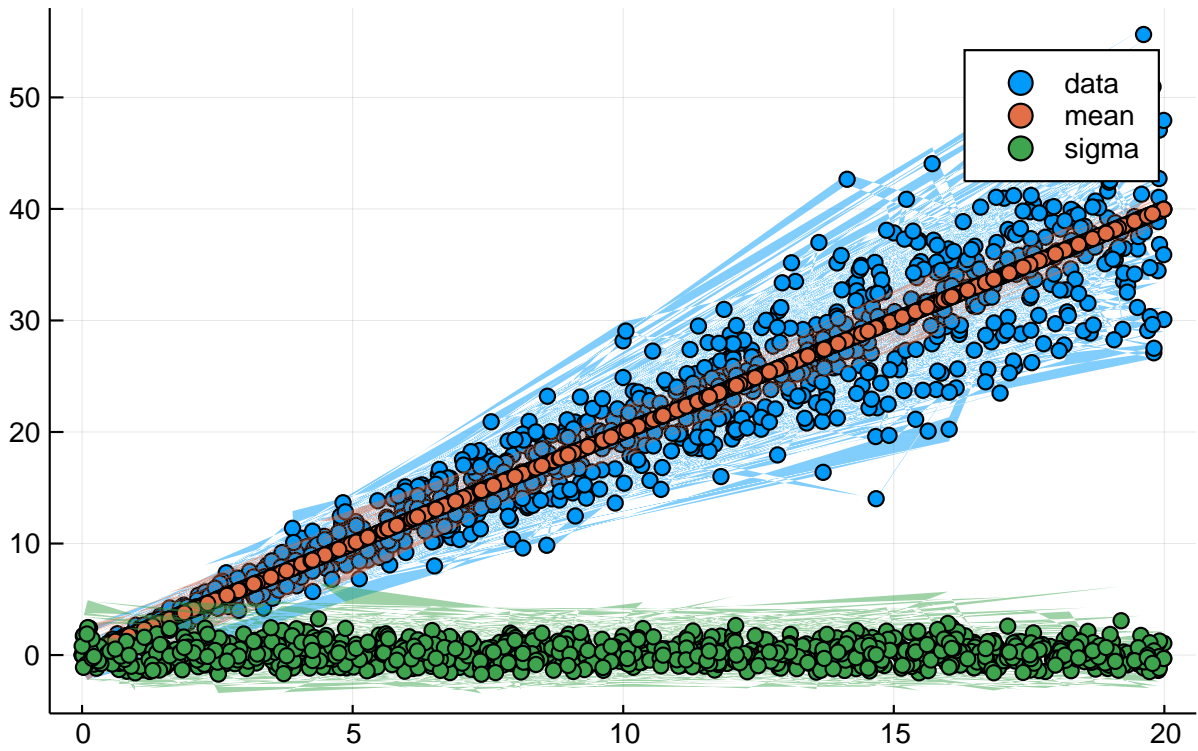
If your implementation is correct, it is possible to learn a reasonable model with fewer than 10 minutes of training on a laptop CPU. The default hyperparameters should help, but may need some tuning.

```
x,y = sample_batch(target_f1,1000)
theta_init=[2 .*vec(x),randn(1000)]
theta1_learned = train_nn_w_var_reg(target_f1, theta_init; bs= 1000, lr = 1e-7, iters=1000)
sigma1= theta1_learned[2]
display(plot(x',hcat(y,theta1_learned[1],sigma1),ribbon=sigma1,title="learned mean,sigma for f1",
seriestype=:scatter,label=["data" "mean" "sigma"])))
```



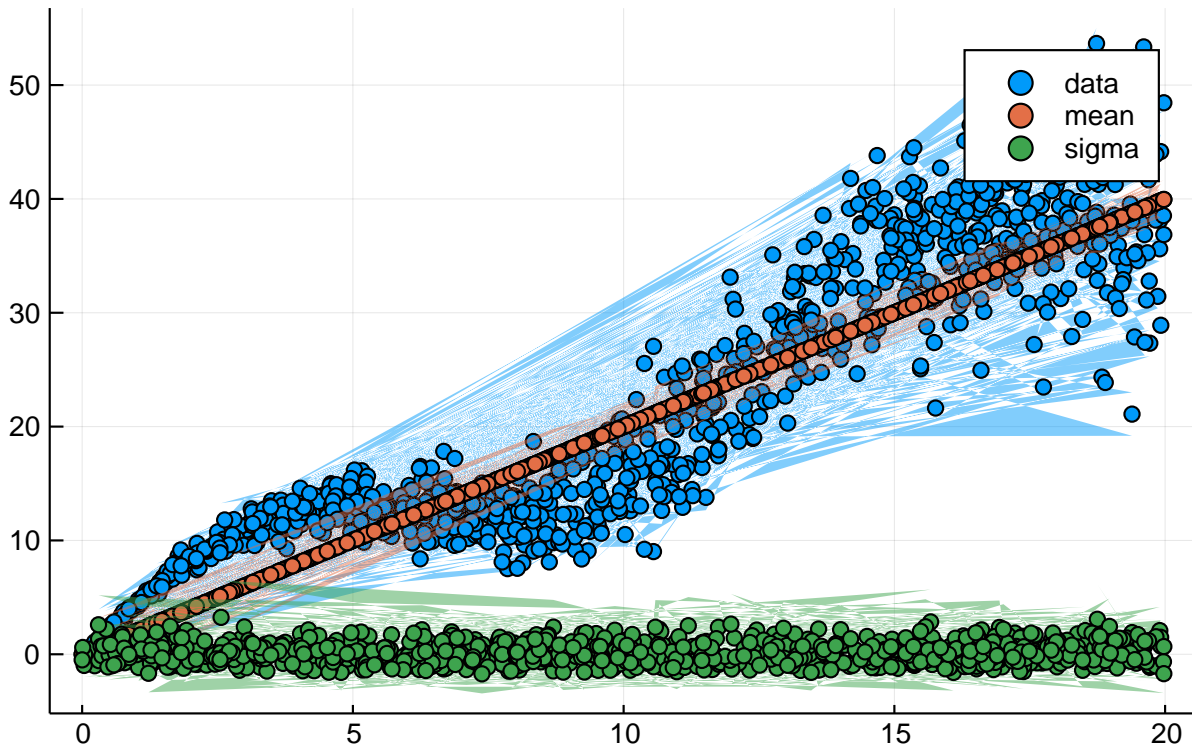
```
x,y = sample_batch(target_f2,1000)
theta_init=[2 .*vec(x),randn(1000)]
theta2_learned = train_nn_w_var_reg(target_f2, theta_init; bs= 1000, lr = 1e-7, iters=1000)
sigma2= theta2_learned[2]
display(plot(x',hcat(y,theta2_learned[1],sigma2),ribbon=sigma2,title="learned mean,sigma for f2",
seriestype=:scatter,label=["data" "mean" "sigma"])))
```


learned mean,sigma for f2



```
x,y = sample_batch(target_f3,1000)
theta_init=[2 .*vec(x),randn(1000)]
theta3_learned = train_nn_w_var_reg(target_f3, theta_init; bs= 1000, lr = 1e-7, iters=1000)
sigma3= theta1_learned[2]
display(plot(x',hcat(y,theta3_learned[1],sigma3),ribbon=sigma3,title="learned mean,sigma for
f3",
seriestype=:scatter,label=["data" "mean" "sigma"])))
```


learned mean,sigma for f3



If you would like to take the time to train a very good model of the data (specifically for target functions 2 and 3) with a neural network that outputs both mean and $\log \sigma$ you can do this, but it is not necessary to achieve full marks. You can try

- Using a more stable optimizer, like Adam. You may import this from a library.
- Increasing the expressivity of the neural network, increase the number of layers or the dimensionality of the hidden layer.
- Careful tuning of hyperparameters, like learning rate and batchsize.