Regression [30pts]

Manually Derived Linear Regression

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. What happens if $n < m$?
2. What are the expectation and covariance matrix of $\hat{\beta}$, for a given true value of β?
3. Show that maximizing the likelihood is equivalent to minimizing the squared error $\sum_{i=1}^n (y_i - x_i\beta)^2$. [Hint: Use $\sum_{i=1}^n a_i^2 = a^T a$]
4. 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]
5. 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.

Toy Data

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
```

```
target_f1 (generic function with 2 methods)
```

```
function target_f1(x, σ_true=0.3)
```

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

target_f2 (generic function with 1 method)

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

target_f3 (generic function with 1 method)

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

Sample data from the target functions

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

```
  md"""
  ### Sample data from the target functions

  Write a function which produces a batch of data $x \sim \text{Uniform}(0,20)$ and `y =
  target_f(x)`
  """
```

sample_batch (generic function with 1 method)

```
  function sample_batch(target_f, batch_size)
    x =#TODO
    y =#TODO
    return (x,y)
  end
```

Test assumptions about your dimensions

```
  md"""
  ### Test assumptions about your dimensions
  """
```

```
  using Test
```

Multiple definitions for n.

Combine all definitions into a single reactive cell using a `begin ... end` block.

```
  @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
```

Plot the target functions

For all three targets, plot a $n = 1000$ sample of the data.

Note: You will use these plots later, in your writeup. Conmsider suppressing display once other questions are complete.

```
using Plots
```

UndefVarError: x1 not defined

1. top-level scope @ Local: 1

```
x1,y1  #TODO
```

UndefVarError: plot_f1 not defined

1. top-level scope @ Local: 1

```
plot_f1  #TODO
```

UndefVarError: x2 not defined

1. top-level scope @ Local: 1

```
x2,y2 #TODO
```

UndefVarError: plot_f2 not defined

1. top-level scope @ Local: 1

```
plot_f2 #TODO
```

UndefVarError: x3 not defined

1. top-level scope @ Local: 1

```
x3,y3  #TODO
```

UndefVarError: plot_f3 not defined

1. top-level scope @ Local: 1

```
plot_f3  #TODO
```

Linear Regression Model with $\hat \beta$ MLE

```
md"""## Linear Regression Model with $\hat \beta$ MLE"""
```

Code the hand-derived MLE

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.

```
md"""
### Code the hand-derived MLE

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.
"""
```

beta_mle (generic function with 1 method)

```
function beta_mle(X,Y)
  beta = #TODO
  return beta
end
```

n =
Multiple definitions for n.

Combine all definitions into a single reactive cell using a `begin ... end` block.

```
n=1000 # batch_size
```

UndefVarError: x_1 not defined

1. top-level scope @ *Local: 1*

```
x_1, y_1  #TODO
```

UndefVarError: β_mle_1 not defined

1. top-level scope @ *Local: 1*

```
β_mle_1  #TODO
```

UndefVarError: x_2 not defined

1. top-level scope @ *Local: 1*

```
x_2, y_2  #TODO
```

UndefVarError: β_mle_2 not defined

1. top-level scope @ *Local: 1*

```
β_mle_2  #TODO
```

UndefVarError: x_3 not defined

1. top-level scope @ *Local: 1*

```
x_3, y_3 #TODO
```

UndefVarError: β_mle_3 not defined

1. top-level scope @ *Local: 1*

```
β_mle_3  #TODO
```

Plot the MLE linear regression model

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**

```
md"""
### Plot the MLE linear regression model

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**
"""
```

UndefVarError: plot_f1 not defined

1. top-level scope @ *Local: 1*

```
plot!(plot_f1,TODO)
```

UndefVarError: plot_f2 not defined

1. top-level scope @ *Local: 1*

```
plot!(plot_f2,TODO)
```

UndefVarError: plot_f3 not defined

1. top-level scope @ *Local: 1*

```
plot!(plot_f3,TODO)
```

Log-likelihood of Data Under Model

Code for Gaussian Log-Likelihood

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., $x_i \sim \mathcal{N}(\mu_i, \sigma_i)$.

```
md"""
```

```
## Log-likelihood of Data Under Model

### Code for Gaussian Log-Likelihood

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

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

```
function gaussian_log_likelihood(μ, σ, x)
  """

  compute log-likelihood of x under N(μ,σ)
  """

  return #TODO: log-likelihood function
end
```

Test Gaussian likelihood against standard implementation

```
md"""
### Test Gaussian likelihood against standard implementation
"""
```

Multiple definitions for μ and x.

Combine all definitions into a single reactive cell using a `begin ... end` block.

```
@testset "Gaussian log likelihood" begin
using Distributions: pdf, Normal
# Scalar mean and variance
x = randn()
μ = randn()
σ = rand()
@test size(gaussian_log_likelihood(μ,σ,x)) == () # Scalar log-likelihood
@test gaussian_log_likelihood.(μ,σ,x) ≈ log.(pdf.(Normal(μ,σ),x)) # Correct Value
# Vector valued x under constant mean and variance
x = randn(100)
μ = randn()
σ = rand()
@test size(gaussian_log_likelihood.(μ,σ,x)) == (100,) # Vector of log-likelihoods
@test gaussian_log_likelihood.(μ,σ,x) ≈ log.(pdf.(Normal(μ,σ),x)) # Correct Values
# Vector valued x under vector valued mean and variance
x = randn(10)
μ = randn(10)
σ = rand(10)
@test size(gaussian_log_likelihood.(μ,σ,x)) == (10,) # Vector of log-likelihoods
@test gaussian_log_likelihood.(μ,σ,x) ≈ log.(pdf.(Normal.(μ,σ),x)) # Correct Values
end
```

Model Negative Log-Likelihood

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.

```
md"""
### Model Negative Log-Likelihood
```

```
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 $\beta$.
"""
```

lr_model_nll (generic function with 1 method)

```
function lr_model_nll(β,x,y;σ=1.)
  return #TODO: Negative Log Likelihood
end
```

Compute Negative-Log-Likelihood on data

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.

```
md"""
### Compute Negative-Log-Likelihood on data

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.
"""
```

UndefVarError: nll not defined

1. top-level scope @ *Local: 7* [inlined]
2. top-level scope @ *none:0*

```
for n in (10,100,1000)
    println("--------  $n  ------------")
    for target_f in (target_f1,target_f2, target_f3)
      println("--------  $target_f  ------------")
      for σ_model in (0.1,0.3,1.,2.)
        println("--------  $σ_model  ------------")
        x,y = #TODO
        β_mle = #TODO
        nll = #TODO
        println("Negative Log-Likelihood: $nll")
      end
    end
end
```

Effect of model variance

For each target function, what is the best choice of σ?

Please note that σ and batch-size n are modelling hyperparameters. In the expression of maximum likelihood estimate, σ or n do not appear, and in principle shouldn't affect the final answer. However, in practice these can have significant effect on the numerical stability of the model. Too small values of σ will make data away from the mean very unlikely, which can cause issues with precision. Also, the negative log-likelihood objective involves a sum over the log-likelihoods of each datapoint. This means that with a larger batch-size n, there are more datapoints to sum over, so a larger negative log-likelihood is not necessarily worse. The take-home is that you cannot directly compare the negative log-likelihoods achieved by these models with different hyperparameter settings.

```
md"""
### Effect of model variance

For each target function, what is the best choice of $\sigma$?


Please note that $\sigma$ and batch-size $n$ are modelling hyperparameters.
In the expression of maximum likelihood estimate, $\sigma$ or $n$ do not appear, and
in principle shouldn't affect the final answer.
However, in practice these can have significant effect on the numerical stability of
the model.
Too small values of $\sigma$ will make data away from the mean very unlikely, which
can cause issues with precision.
Also, the negative log-likelihood objective involves a sum over the log-likelihoods of
each datapoint. This means that with a larger batch-size $n$, there are more
datapoints to sum over, so a larger negative log-likelihood is not necessarily worse.
The take-home is that you cannot directly compare the negative log-likelihoods
achieved by these models with different hyperparameter settings.
"""
```

Automatic Differentiation and Maximizing Likelihood

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.

```
md"""
## Automatic Differentiation and Maximizing Likelihood

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

Compute Gradients with AD, Test against hand-derived

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.

```
md"""
### Compute Gradients with AD, Test against hand-derived
For a random value of $\beta$, $\sigma$, 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 $\beta$.
Test that this is equivalent to the hand-derived value.
"""
```

ArgumentError: Package Zygote not found in current path:

```
- Run `import Pkg; Pkg.add("Zygote")` to install the Zygote package.

  1. require(::Module, ::Symbol) @ loading.jl:893
  2. top-level scope @ Local: 1
```

```julia
using Zygote: gradient
```

Multiple definitions for y and x.

```
Combine all definitions into a single reactive cell using a `begin ... end` block.
```

```julia
@testset "Gradients wrt parameter" begin
β_test = randn()
σ_test = rand()
x,y = sample_batch(target_f1,100)
ad_grad = #TODO
hand_derivative = #TODO
@test ad_grad[1] ≈ hand_derivative
end
```

Train Linear Regression Model with Gradient Descent

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.

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.

```julia
md"""
### Train Linear Regression Model with Gradient Descent

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

Write a function `train_lin_reg` that accepts a target function and an initial estimate for $\beta$ 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 $\beta$
* update the estimate of $\beta$ with gradient descent with specified learning rate
and, after all iterations, returns the final estimate of $\beta$.
"""
```

syntax: unexpected ")"

1. top-level scope @ *none:1*

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

function train_lin_reg(target_f, β_init; bs= 100, lr = 1e-6, iters=1000, σ_model = 1.)
    β_curr = β_init
    for i in 1:iters
      x,y = #TODO
      @info "loss: $()  β: $β_curr" #TODO: log loss, if you want to monitor training progress
      grad_β = #TODO: compute gradients
      β_curr = #TODO: gradient descent
    end
    return β_curr
end
```

Parameter estimate by gradient descent

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.

```
md"""
### Parameter estimate by gradient descent

For each target function, start with an initial parameter $\beta$,
    learn an estimate for $\beta_\text{learned}$ 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.
"""
```

β_init = -1045.4328263647565

```
β_init = 1000*randn() # Initial parameter
```

UndefVarError: β_learned not defined

1. top-level scope @ *Local: 1*

```
β_learned  #TODO: call training function
```

Plot learned models

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.

```
md"""
### Plot learned models

For each target function, start with an initial parameter $\beta$,
learn an estimate for $\beta_\text{learned}$ 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.
"""
```

```
#TODO: For each target function, plot data samples and learned regression
```

Non-linear Regression with a Neural Network

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).

$$Y \sim \mathcal{N}(\texttt{neural\_net}(X,\theta), \sigma^2)$$

```
md"""
Non-linear Regression with a Neural Network

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 $\theta$
(collection of weights and biases).

$$Y \sim \mathcal{N}(\texttt{neural\_net}(X,\theta), \sigma^2)$$
"""
```

Fully-connected Neural Network

Write the code for a fully-connected neural network (multi-layer perceptron) with one 10-dimensional hidden layer and a `tanh` nonlinearirty. 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.

`neural_net (generic function with 1 method)`

```
# Neural Network Function
function neural_net(x,θ)
  return #TODO
end
```

UndefVarError: θ not defined

1. top-level scope @ *Local: 2*

```
# Random initial Parameters
θ #TODO
```

Test assumptions about model output

Test, at least, the dimension assumptions.

```
md"""
### Test assumptions about model output

Test, at least, the dimension assumptions.
"""
```

Multiple definitions for μ, n, y and x.

Combine all definitions into a single reactive cell using a `begin ... end` block.

```
@testset "neural net mean vector output" begin
n = 100
x,y = sample_batch(target_f1,n)
μ = neural_net(x,θ)
@test size(μ) == (n,)
end
```

Negative Log-likelihood of NN model

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 $\sigma = 1.0$

```
md"""
### Negative Log-likelihood of NN model
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 $\sigma = 1.0$
"""
```

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

```
function nn_model_nll(θ,x,y;σ=1)
  return #TODO
end
```

Training model to maximize likelihood

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.

```
md"""
### Training model to maximize likelihood

```

```
Write a function `train_nn_reg` that accepts a target function and an initial estimate
for $\theta$ 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 $\theta$
* update the estimate of $\theta$ with gradient descent with specified learning rate
and, after all iterations, returns the final estimate of $\theta$.
"""
```

```
syntax: unexpected ")"

  1. top-level scope @ none:1
```

```julia
function train_nn_reg(target_f, θ_init; bs= 100, lr = 1e-5, iters=1000, σ_model = 1. )
    θ_curr = θ_init
    for i in 1:iters
      x,y = #TODO
      @info "loss: $()" #TODO: log loss, if you want to montior training
      grad_θ = #TODO: compute gradients
      θ_curr = #TODO: gradient descent
    end
    return θ_curr
end
```

Learn model parameters

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.

```
md"""
### Learn model parameters

For each target function, start with an initialization of the network parameters,
$\theta$,
    use your train function to minimize the negative log-likelihood and find an
estimate for $\theta_\text{learned}$ by gradient descent.

"""
```

```
syntax: invalid syntax (incomplete #<julia: "incomplete: premature end of input">)

  1. top-level scope @ none:1
```

```julia
θ_init = #TODO
θ_learned = #TODO
```

Plot neural network regression

Then plot a $n = 1000$ sample of the data and the learned regression model with shaded uncertainty bounds given by $\sigma = 1.0$

```
md"""
### Plot neural network regression

Then plot a $n=1000$ sample of the data and the learned regression model with shaded
uncertainty bounds given by $\sigma = 1.0$
"""
```

Input-dependent Variance

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)

$$Y \sim \mathcal{N}(\texttt{neural\_net}(X,\theta), \sigma^2)$$

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.

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

```
md"""
## Input-dependent Variance

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)

$$Y \sim \mathcal{N}(\texttt{neural\_net}(X,\theta), \sigma^2)$$

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.

$$\begin{align*}
\mu, \log \sigma &= \texttt{neural\_net}(X,\theta)\\
Y &\sim \mathcal{N}(\mu, \exp(\log \sigma)^2)
\end{align*}$$
"""
```

Neural Network that outputs log-variance

Write the code for a fully-connected neural network (multi-layer perceptron) with one 10-dimensional hidden layer and a `tanh` nonlinearirty, and outputs both a vector for mean and $\log \sigma$.

```
md"""
### Neural Network that outputs log-variance

```

```
Write the code for a fully-connected neural network (multi-layer perceptron) with one
10-dimensional hidden layer and a `tanh` nonlinearirty, and outputs both a vector for
mean and $\log \sigma$.
"""
```

neural_net_w_var (generic function with 1 method)

```
# Neural Network with variance
function neural_net_w_var(x,θ)
  return #TODO
end
```

UndefVarError: θ not defined

1. top-level scope @ *Local: 2*

```
# Random initial Parameters
θ  #TODO
```

Test model assumptions

Test the output shape is as expected.

```
md"""
### Test model assumptions

Test the output shape is as expected.
"""
```

Multiple definitions for μ, n, y and x.

Combine all definitions into a single reactive cell using a `begin ... end` block.

```

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

Negative log-likelihood with modelled variance

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 $\exp \log \sigma$)

```
md"""
### Negative log-likelihood with modelled variance

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 $\exp \log \sigma$)
"""
```

nn_with_var_model_nll (generic function with 1 method)

```
function nn_with_var_model_nll(θ,x,y)
```

```
    return #TODO
end
```

Write training loop

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.

```
md"""
### Write training loop

Write a function `train_nn_w_var_reg` that accepts a target function and an initial
estimate for $\theta$ 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 $\theta$
* update the estimate of $\theta$ with gradient descent with specified learning rate

and, after all iterations, returns the final estimate of $\theta$.
"""
```

syntax: unexpected ")"

1. **top-level scope** @ *none:1*

```
function train_nn_w_var_reg(target_f, θ_init; bs= 100, lr = 1e-4, iters=10000)
    θ_curr = θ_init
    for i in 1:iters
      x,y = #TODO
      @info "loss: $()" #TODO: log loss
      grad_θ = #TODO compute gradients
      θ_curr = #TODO gradient descent
    end
    return θ_curr
end
```

Learn model with input-dependent variance

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.

```
md"""
### Learn model with input-dependent variance

 For each target function, start with an initialization of the network parameters, $\theta$,
    learn an estimate for $\theta_\text{learned}$ 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 $\sigma$)

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.

"""
```

UndefVarError: θ_init not defined

1. top-level scope @ *Local: 2*

```
#TODO: For each target function
θ_init  #TODO
```

UndefVarError: θ_learned not defined

1. top-level scope @ *Local: 1*

```
θ_learned #TODO
```

Plot model

```
md"""
### Plot model
"""
```

```
#TODO: plot data samples and learned regression
```

Spend time making it better (optional)

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.

```
md"""
### Spend time making it better (optional)

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.
"""
```