Building ML Models

Week 5 – Autodiff and Gradient Descent Review

Autodiff Review

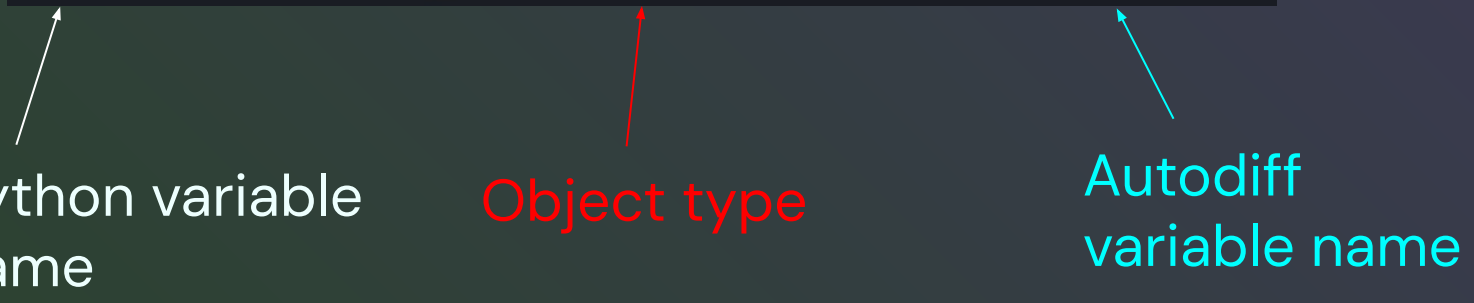
Autodiff Review

- We created this autodiff library, what does it actually do?

Autodiff Review: Expression

- `expression` is the base class. As the purest form, it represents a single variable.
- You create an instance of it by doing:

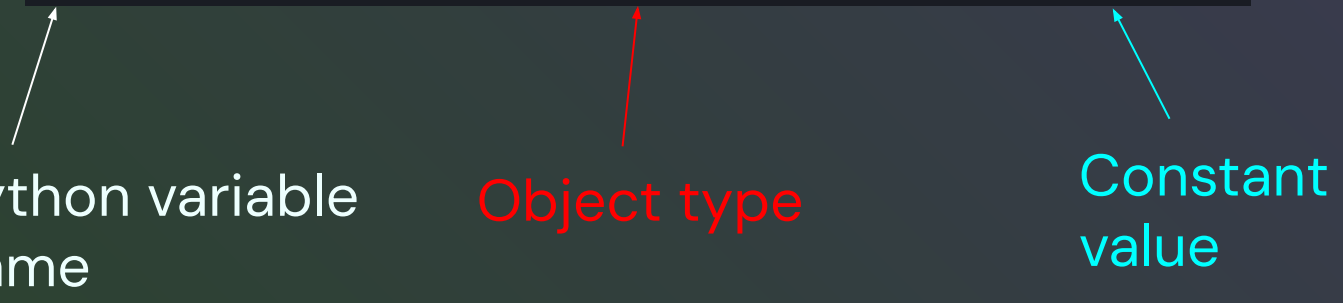
```
x = expression("x")
```



Autodiff Review: Constant

- `constant` is another fundamental class. It represents a constant value, like 10.
- You create an instance of it by doing:

```
one = constant(1)
```



Autodiff Review: Combining Expressions

- We can combine multiple of these expressions together with basic math operations:

```
two = constant(2)
x = expression("x")
two_x = two * x
```

This will implicitly create a `multiplication` object, which will be storing the `expression` 2x

Autodiff Review: Evaluating Expressions

- We can evaluate expressions with the `eval` function:

```
result = two_x.eval({"x": 10})
```

Note: we need to pass in a dictionary telling the autodiff library what the values of our `expressions` are.

Autodiff Review: Differentiating

- We can differentiate expressions with the `diff` function:

```
result = two_x.diff({"x": 10}, "x")
```

Note: we need to pass in a dictionary telling the autodiff library what the values of our expressions are AND what variable we need to differentiate with respect to.

Think: why do we need to pass in the values?

Gradient Descent Review

Gradient Descent Review

- We've talked about the formulas a lot for gradient descent, but what does that look like when training an actual model?

Gradient Descent Review: Training

- When trying to train a regression model, we pass in a set of data to fit on, and the expected outputs
- For each set of that data, we calculate what our model would predict with this formula:

$$\hat{Y} = m_1 x_1 + m_2 x_2 + \ldots + b$$

Gradient Descent Review: Updating Params

- At first, our model is probably very far off from the right answer!
- We need to somehow tell the model how far off it is, and update its parameters

Gradient Descent Review: Updating Params

- At first, our model is probably very far off from the right answer!
- We need to somehow tell the model how far off it is, and update its parameters
- We do this via mean-squared error

Gradient Descent Review: Updating Params

- We calculate how “far off” the model is via this formula:

$$MSE(\hat{Y}, Y) = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y} - Y)^2$$

Y is what the model outputs, Ŷ is the true value

Gradient Descent Review: Updating Slopes

- We can use MSE to update the slopes:

$$\Delta m_i = -\frac{\partial}{\partial m_i} MSE(\hat{Y}, Y)$$

Note how this is only for a single slope value, for a single sample of our training data. We need to accumulate this loss across all samples separately, for *each* slope value.

Gradient Descent Review: Updating Y-int

- Updating the y-intercept is mostly the same:

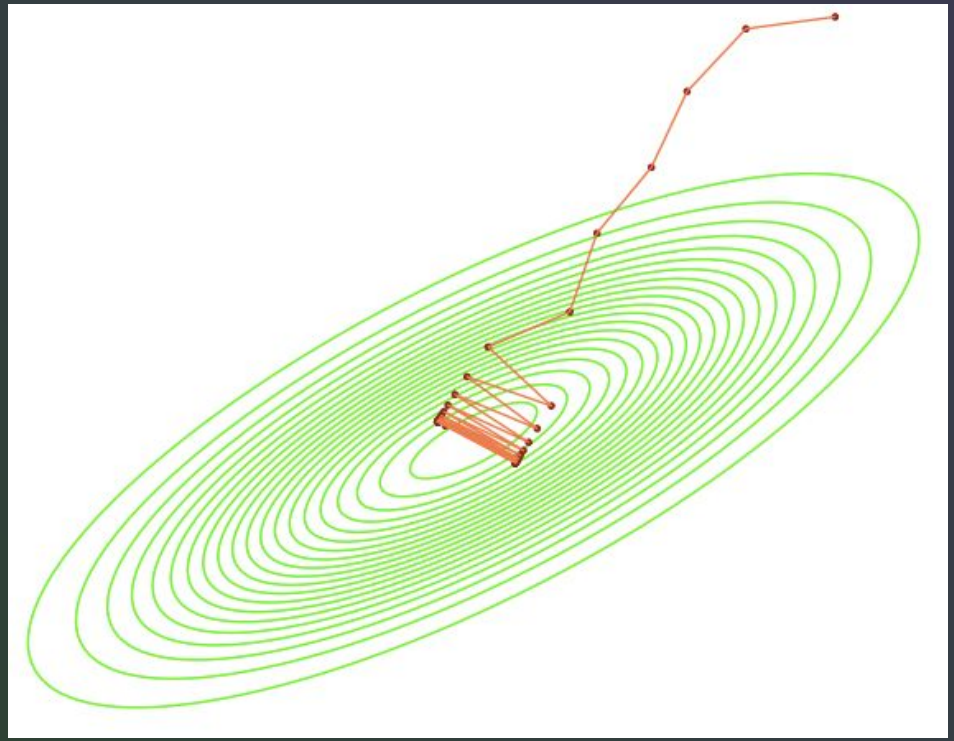
$$\Delta b = -\frac{\partial}{\partial b} MSE(\hat{Y}, Y)$$

Recall from last week: we only need to keep track of a *singular* y-intercept value.

Gradient Descent Review: Oscillation

- When we're updating the weights in gradient descent, we don't want to keep overshooting our target
- This is why a learning rate is introduced, to prevent oscillation
- Otherwise, our weights would never converge
- Multiply the final updated delta values by this learning rate

Gradient Descent Review: Oscillation



Overview of Linear Regression Code

Linear Regression Code: Member Variables

- `self.mx`: a dictionary with string keys “m0”, “m1”, … to “mN”, N is the number of slope variables
- `self.yint`: the y-intercept value

! Actual ***values***, not just autodiff expressions !

Linear Regression Code: fit()

- `fit()` takes in 4 parameters, `X_fit`, `y_fit`, `n_epochs`, and `learning_rate`
- `X_fit`: a dataframe to train the model on
- `y_fit`: the expected y-values for each row of `X_fit`
- `n_epochs` the number of iterations you'll iterate for in gradient descent
- `learning_rate`: the factor to multiply the losses by

Linear Regression Code: fit() TODOs

- Create `prediction_expr`, which is m0*x0 + m1*x1 + ... + yint
  - Hint: you can get the specific row/column element in a dataframe by doing `X_fit.iloc[row, column]`
- Gradient descent: store the current loss values for each slope variable and the y-intercept
  - Hint: we need to iterate through each of the keys in `self.mx`, what do we need to differentiate with respect to?

Linear Regression Code: fit() TODOs

- Update `self.mx` and `self.yint`
  - Use the learning rate and the gradients you've calculated for that epoch to update the slope variables you stored previously

Linear Regression Code: predict()

- `predict()` takes in one parameter
- `X_pred` is a dataframe of the data we want to predict the y-values for

Linear Regression Code: predict() TODOs

- Prediction calculation. Use the formula y = m0*x0 + m1*x1 + ... + yint to predict a y-value and add it to the `predictions` list.
  - Hint: did we use this formula anywhere else?

Autodiff Exercises

Autodiff Exercises

- GitHub
- Jupyter Notebook