Tutorial 4

Autograd

- *Automatic differentiation* is a method to get exact gradients (derivatives) efficiently, by storing the computational graph as we perform a forward computation, which we can reuse when going backwards
  - This is different from symbolic differentiation, which manipulates symbolic expressions to get an exact algebraic expression for the derivative
    * This is expensive and impractical for very complex computations like a large neural network
  - This is also different from numeric differentiation, which approximates derivatives by finite differences
    * This is can also be expensive and unstable
  - Takes code that computes a function and returns code that computes the derivative
- Any function can be broken down into a computational graph of basic operations which we know how to differentiate, then we can apply the chain rule
- `autograd` is a Python package for automatic differentiation
  - It can auto-differentiate Python and numpy code
  - `import autograd.numpy as np` gives a thin wrapper around regular numpy functions
    * This replaces normal numpy functions with ones that also track the computational graph
  - The `autograd.grad` takes a function, and gives a function that computes its gradient
  - Can handle most common Python structures
  - Can calculate higher order derivatives as well, by simply calling `grad()` multiple times
- `autograd` performs backpropagation to calculate the gradients
- For functions with multiple parameters (note the return value should be a single scalar):
  - `grad(f, argnum)` computes the gradient with respect to the argument at position `argnum`
    * By default the gradient is taken with respect only to the first variable
    * The resulting function is still a function of all the original variables
  - `grad_named(f, argname)` computes the gradient with respect to the argument with name `argname`
  - `multigrad(f, argnums)` computes gradients with respect to multiple arguments simultaneously (`argnums` is a list)
  - `multigrad_dict(f)` computes gradients with respect to all arguments simultaneously, returning a dict mapping argument names to gradient values
- Custom gradients can be registered, if we want to manually specify the gradient of a function, for purposes such as speed, numerical stability, etc