Reading Notes for Week 5

Learning XOR Function

- The key observation is that XOR is not linearly separable. Thus if we use generalized linear model, like logistic regression (based on bernoulli distribution), the decision boundary for output 0 and 1 is linear, but the original x space cannot be separated by this linear decision boundary, so introducing non-linearity is the key motivation to use neural network.
- The reason why we prefer RELU is that it produces a very consistent and strong gradient instead of diminishing during backpropagation. By applying non-linearity using rectified-RELU activation function, in the new feature space (hidden states), the model acts as linearly separable.

Gradient-Based Learning

There exists largest difference between linear models and neural networks:

- **Nonlinearity** of a neural network causes most interesting loss functions under linear models to become **non-convex. (w.r.t some matrix parameters)**. Because of this, neural networks are usually trained by using iterative, gradient-based method like gradient descent or SGD covered in previous lectures.
- Convex optimization converges starting from any initial parameters. In contrast, stochastic gradient descent applied to non-convex loss functions has no such convergence guarantee, and is sensitive to the values of the initial parameters
    - For feedforward neural networks, it is important to initialize **all weights to small random values**
    - **Biases may be initialized to zero** or to small positive values
- The discontinuity of RELU activation at x=0 inspires the use of subgradient descent instead of gradient descent. But in reality, by chain rule, it is very unlikely that we will evaluate the gradient of RELU at 0. After all, common practice is that we pretend that the gradient of RELU is 0 at 0.

Learning Conditional Statistics

- In most cases, parametric model defines a conditional distribution $p(x \mid x; \theta)$. Using the principle of MLE, we get the cross-entropy between the training data and the model predictions as the cost function.
- Sometimes, we take a simpler approach: rather than predicting a complete probability distribution over y, merely predict some **conditional statistics of y** conditioned on x.

Cost Function

- We can view the cost function as being a **functional** rather than just a function
    - **functional**: a mapping from functions to real numbers
    - Thus, we can think of learning as choosing a function rather than merely choosing a set of parameters
    - Thus we can design the cost functional to have its minimum occur at some specific function we desire.

- Mean squared error and mean absolute error often lead to poor results when used with gradient-based optimization since some output units that saturate produce very small gradients when combined with these cost functions.
- This is one reason that the cross-entropy cost function is more popular, even when it is not necessary to estimate an entire distribution p(y | x)

Output Units

- The choice of cost function is strongly connected to the output types/distribution. For example for gaussian likelihood, we choose linear output layer with MSE cost function. For bernoulli likelihood, we choose sigmoid output layer and cross-entropy cost function. For multinoulli likelihood, we choose softmax output layer and multi-class discrete cross entropy function.
- Any kind of neural network unit that may be used as an output can also be used as a hidden unit
- Suppose that the feedforward network provides a set of hidden features defined by h = f(x; θ), the role of the output layer is then to provide some additional transformation from the features to complete the task that the network must perform.

Back Propagation Algorithms

- To discuss backprop, it's useful to first develop computational graph language
    - Let each node in the graph indicate a variable (a scalar, vector, matrix, tensor, or other)
    - Introduce the idea of an operation- a simple function of one or more variables
- Chain Rule of Calculus(a review)
- Several back propagation algorithms

Symbol-to-symbol Derivatives

- Algebraic expressions and computational graphs both operate on symbols- variables that do not have specific values
    - These algebraic and graph-based representations are called symbolic representations
    - When using or training a neural network, symbolic inputs are replaced with a specific numeric value
- Some approaches to backprop use a **"symbol-to-number"** approach to differentiation:
    - Take a computational graph and a set of numerical values for the inputs to the graph.
    - Return a set of numerical values describing the gradient at those input values,
- Another approach is to use a **"symbol-to-symbol"** approach to differentiation:
    - Take a computational graph and add additional nodes to the graph that provide a symbolic description of the desired derivatives.

- **Primary advantage**: derivatives are described in the same language as the original expression
    - Because the derivatives are just another computational graph, it is possible to run backprop again, differentiating the derivatives in order to obtain higher derivatives
    - Every node can be evaluated as soon as its parents' values are available, like a DAG, allowing us to avoid specifying exactly when each operation should be computed symbol-to-symbol subsumes the symbol-to-number approach
    - Symbol-to-number performs the exact same computations as are done in the graph built by the symbol-to-symbol approach
- **The key difference**: symbol-to-number does not expose the graph