Research Journal

Shad Boswell
University of Utah

2023

June

Jun. 11, 2023 — Sunday

My research today was split between reading d2l.ai (Smola n.d.) and watching Andrej Katpathy's video Karpathy n.d.(b)

Notes on backpropogation and neural networks:

1. Neural networks are just a mathematical expression which can be drawn out using a tree. This tree helps to visualize the expression and understand how backpropogation works
2. Backpropogation involves starting with the result of an expression and working backwards to see how each parameter to the equation influences the total outcome. It's a "recursive application of backpropogation through the computational tree"
3. A loss function is a way to tell how well your model is performing. In machine learning, we are trying to minimize our loss as much as possible.
4. Neurons in a neural network take all the summation of the weights * inputs, add a bias and pass the result to what is known as an "activation function" or "squashing function".
5. Squashing functions are typically a sigmoid or tanh function and map the real numbers to the range [0,1]

Jun. 12, 2023 — Monday

Finishing Andrej Karpathy's video (Karpathy n.d.(b)). Notes:

1. Practice backpropogation. Rules for backpropogation:
    a) Backpropgation through addition disperses the gradient to all of the child nodes

   b) Backpropogation through multiplication takes the the other children's data, multiplies them, and then multiplies by the output gradient
   c) Generally, we take the local derivative of the function and multiply by the output gradient

2. Learn the subtleties of Python. The "__rmul__" function serves as fallback for the "__mul__" function and swaps the order of the operands

3. Practice using things that you have already written. Example: Andrej implemented subtraction as the addition of a negation. Andrej had already written the addition operation so he just need to write the negation operation which is just a multiplication by -1. Andrej had already written multiplication so this was trivial.

4. Pararphrase: "The level at which you implement your operations is up to you... All that matters is that we have some kind of inputs and some kind of outputs and that the output is a function of the input... As long as we can forward pass and backward pass, then the design is up to you." - Karpathy n.d.(b)

5. Using PyTorch and Tensors like Micrograd:
   a) Need to specify that tensors need a gradient with: n.requires_grad = True
   b) Tensors have .data.item() which strips out the value in the Tensor
   c) Tensors also have .grad.item() attribute similar to Value.grad in micrograd
   d) The big deal with PyTorch is the efficiency and our ability to take advantage of parallelism with Tensors

6. Building a Nerual Net. A neural net is a specific class of mathematical expression.
   a) We start by building a Neuron class and then initializing it with some weights and a bias.

7. Loss is a single number that represents the overall performance of our model. In essence it is the difference between the actual output values and the predicted values. Thus, we want to minimize our loss so that our model performs how we expect and accomplishes the task at hand.

8. Common loss function is the mean-squared-error function:

$$\frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2$$

   where Y_i is the target values and $\hat{Y}_i$ is the observed values. Why MSE? The further we are away, the greater the loss would be. This effect is exaggerated by squaring the difference between the values. Any difference less than one will also result in a smaller loss than difference which rewards small loss. In a perfect world, our target and our observed value would be the same which would yield 0 loss.

9. Do not mess with inputs to neural nets only weights and biases. The total number of parameters for our neural net is the total number of weights + the total number of biases. We can find this by taking the sum of products between layers (weights) and adding the number of neurons in the inner layers + the outer layer (biases)

10. Modifying parameters:

    a) If a gradient is negative, you increase the parameter

    b) If a gradient is positive, you decrease the parameter

    c) We can simply just multiply our step size by the -gradient

    d) Andrej states that our gradient vectors point in the direction of increasing loss, so we want to go in the opposite direction.

    e) Then you just iterate this process until loss is really low (low enough for whatever your standards are). This is called **gradient descent**!

    f) Be careful, you may overstep! Similar to Newton's method -> You will bounce back and forth between having close observed values and far away observed values. This is called learning rate. We need to find the step size to be just right. You can either take way too long, or explode your loss.

11. Training loops:

    a) First do forward pass, then backward pass so we have accurate gradients. Then update. This is the fundamental idea of gradient descent!

    b) **MAKE SURE YOU FLUSH THE GRADIENT BEFORE BACKWARD PASS!!**

    c) Learning rate decay: Slowing down the learning rate with more iterations (Scaling the learning rate as a function of the number of iterations) allows you to focus on the finer details as the network learns more and more.

12. Summary:

    a) Neural nets: Mathematical expression for a forward pass, followed by a loss function. We then do a backward pass to get the gradient so we can tune each of the parameters to decrease the loss locally.

    b) When the loss is low, the network is performing how we would like.

    c) ChatGPT essentially takes a series of words and attempts to predict the next word in the sequence. When you train this on the internet, the network has really amazing emergent properties and has billions of parameters.

Reading d2l.ai staring at section 3.1: Regression
Meeting with Ganesh

1. You need to share Tensors. That is the name of the game with machine learning.

2. groq inc paper - Software-defined tensor streaming multiprocessor for large-scale machine learning. They have their own chips and they essentially want to get rid of the idea of kernels.

3. NaN in the GPU: Do not have a great status. In the CPU you can use a specific bit pattern to quiet the NaN or make it signal. Read "The Secret life of NaN"

4. Attacks to floating point computation

    a) Injection exceptions in the input to see how far it goes
    b) Lowering precision for some stages
    c) Alterting the inputs to consume a larger dynamic range
    d) Changing hyper parameters
    e) Changing how the libraries are sparsified

    How do we find and locate these vulnerabilities? What is the least number of monitoring layers to ensure security of the network?

5. Look at FPTuner

Watching Andrej Karpathy's video Karpathy n.d.(a)

1. Makemore is designed to makemore of whatever you feed it.

2. Makemore is a character level language model. It treats every single line of input as an example and then each as sequences of characters. This is the level on which we are building out makemore.

3. With character level language models, we need to be careful of all of the examples packed into a single word. For example, consider the word "isabella". The examples embedded in this word are:

    a) "i" will come first
    b) "s" is likely to follow an "i"
    c) "a" is likely to follow "is"
    d) and so on...
    e) until we get to the end where the word where "a" is likely to follow "isabell"
    f) There is one last example and that is that the word is likely to end after "isabella"
    g) With long words, we have a **TON** of information

4. A bigram is where we try to predict the next letter by only looking at the current letter.

5. The zip function in python is great! It takes two iterators together and pairs the iterations together, truncating the longer iterator to meet the shorter one.

6. What are the statistics involved in a bigram? The simplest way is just through counting.

    a) Let's keep a dictionary to keep track of the counts of tuples. How? Make a tuple of the characters, get its value from the dictionary (default to 0) and add 1 to it. Then restore it.

    b) It's actually going to be more convenient to store these counts in 2d arrays where the rows are the first character in the bigram and the columns are the second character in the bigram. We will use PyTorch's Tensor

    c) Tip: Use setbuilder notation to build lookup tables!

    d) Tip 2: plt.imshow() is pretty cool for plotting 2d arrays

7. Now that we have the counts, we can just sample our data repeatedly to create a new word. We can do this by creating probabilities for the starting characters. This is done by normalizing the array of starting characters.

8. We then use Python's multinomial function to feed it a probability distribution which it will use to pick a number. The numbers than it can select from are the same as the indices of the distribution. Andrej also used a Python generator object to provide a dterministic way of sampling the probability distribution. (I think was for teaching purposes only and maybe not advised? I will have to look into generators even more, but I think they always pick the same if you give them a specific seed.)

9. All in all, bigram language models are pretty terrible. However, they still do something reasonable. If we compare the results to a model that is completely untrained, it performs far better. The words are actually word like where the untrained model just spits out garbage.

10. Once you are finished writing a model, be sure to look for inefficiencies that can be solved easily. (Remember Kopta: "The #1 rule of optimization is: Don't"). The optimization that Andrej did was to factor out the probability calculation from the loop. Rather, we would like to create a 27x27 matrix of probabilities that we can read from. Ideally, we would like each rows probability to be calculated in parallel.

11. We can also sum along dimensions with Tensors. For example, with a 2d Tensor, summing along dimension, or axis, 0 will sum the values in the columns to create a 1 x N Tensor. On the other hand, summing along dimension, or axis, 1 will sum the values in a row to create an N X 1 Tensor. This is really helpful because we can then take advantage of Broadcasting! Wooo!

12. Andrej's tips: READ THROUGH BROADCASTING SEMANTICS! Treat it with respect and take your time.

13. In place operations have the potential to be faster. Use them when you can.

References

Karpathy, Andrej (n.d.[a]). *The spelled-out intro to language modeling: building make-more*. URL: https://www.youtube.com/watch?v=PaCmpygFfXo&list=PLAqhIrjkxbuWI23v9cThsA9GvCAUhRvKZ&index=2&t=2s&ab_channel=AndrejKarpathy.

——— (n.d.[b]). *The spelled-out intro to neural networks and backpropagation: building micrograd*. URL: https://www.youtube.com/watch?v=VMj-3S1tku0&t=5246s&ab_channel=AndrejKarpathy.

Smola, Aston Zhang Zachary C. Lipton Mu Li Alexander J. (n.d.). *Dive into Deep Learning*. URL: https://d2l.ai.