EECS 189 Introduction to Machine Learning
Fall 2020

DIS13

This discussion was released **Friday, December 4**.

In this discussion, we will be taking a deep dive into convolutional layers of neural networks. There will be a lot of visualizations for building our understanding on the convolution operator and benefits of convolutional layers. Then, we will come back to the worksheet to work on Neural Tangent Kernel (NTK), an approach to theoretically analyze neural networks based on its linearization.

1 Convolutional Neural Networks

We will work with a jupyter notebook in this repo. Part of the jupyter notebook will require PyTorch but not GPUs. PyTorch installation with CPU is sufficient. You may run it on DataHub, but it can be a little slow when a lot of people run it at the same time. If you can, consider running it locally or copying it to Colab. If you run it on your local machine, make sure you have some space left on your hard disk and memory because later part of the code will download neural network's weight which is roughly 250MB.

2 Neural Tangent Kernels

This problem shows how we can understand neural network behavior better if we leverage a very standard tool that you have seen in EECS16B (or wherever you learned similar ideas) — linearization of nonlinear functions. This staple tool from control theory and analog transistor circuits (where it is called "small signal analysis" and is developed in courses like EE105) turns out to be vital to give us insight into modern neural networks. In hindsight, this is perhaps not surprising given that a neural network is basically an analog circuit with nonlinear elements in it. The key is to view the network as a function not of its supposed inputs, but instead of its parameter weights. Since the learning dynamics evolve on the weights, a "small signal" view of the response to the weights sheds some important insight into what is going on. Consider the two-layer neural network which computes the following function:

$$f(\mathbf{x}|\mathbf{a}, \mathbf{W}) = \frac{1}{\sqrt{m}} \sum_{r=1}^{m} a_r \sigma(\mathbf{w}_r^\top \mathbf{x}).$$

We impose the following assumptions:

- Initial values of final layer weights $\{a_i\}$ are centered i.i.d. random variables from the segment $[-A, A]$ (not necessarily from the uniform distribution).
- Initial values of weight vectors $\{w_r\}$ are i.i.d. random vectors independent of the $\{a_r\}$,

- The nonlinear activation function σ and its first and second derivatives σ' and σ'' are all bounded (for example you can think of sigmoid activation function).

For understanding convenience, the model that we will use to make predictions from the learned weights is the following:

$$\hat{f}(\mathbf{x}|\mathbf{a}, \mathbf{W}) := f(\mathbf{x}|\mathbf{a}, \mathbf{W}) - f(\mathbf{x}|\mathbf{a}_{\text{initial}}, \mathbf{W}_{\text{initial}})$$

i.e. we subtract out the guess that our initial random weights make. (This corresponds to the concept of bias point in analog circuits and the idea of operating point in control theory.) Why? Because the spirit of this problem is to take the Taylor expansion of the neural network with respect to the weights that we're learning and just keep the first two terms — the constant term (corresponding to the random initial state's predictions) and the linear term (that captures how the network's behavior will change locally as we learn). The prediction model above lets us focus entirely on the linear term where all the action is — the jupyter parts will help you see this more clearly. Our further goal is to understand what happens for large neural networks. Here, we'll use another standard approach — we'll take limits in two ways. We'll let $m \to \infty$ and we will also use a continuous-time perspective since that lets us leverage the core differential-equation thinking that you've developed in other courses (like EECS16B). Consequently, this problem studies training of such a neural net with gradient descent in the "infinite width" regime, which means that we will set parameters of σ, distributions of a_r and $\mathbf{w}_r$, the number of data points n and the data $\mathbf{X}$ itself to be **constant**, and will only have m going to infinity. What happens when we take m to infinity? As it turns out, in this regime training a neural net is no different from just doing a kernel regression with kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \nabla_{\mathbf{a},\mathbf{W}} f(\mathbf{x}_i|\mathbf{a}, \mathbf{W}), \nabla_{\mathbf{a},\mathbf{W}} f(\mathbf{x}_j|\mathbf{a}, \mathbf{W}) \rangle$, which in turn itself converges to a fixed kernel in the infinite-width limit. Sure, there are an infinite number of "features," but that is not an obstacle since the actual (delta-) weights being applied to them are also infinitesimal. This limiting kernel is often called the "Neural Tangent Kernel" (NTK) for obvious reasons. This is what "small signal" analysis as applied to a neural net is about. The rough outline of our argument in this problem is the following: we spend most of our time showing that we don't go very far in the weight space during gradient descent, and then in the last part we show why it means that we are doing kernel regression. We start with computing the gradient of our prediction:

$$\frac{\partial}{\partial a_r} f(\mathbf{x}|\mathbf{a}, \mathbf{W}) = \frac{1}{\sqrt{m}} \sigma(\mathbf{w}_r^\top \mathbf{x}),$$
$$\nabla_{\mathbf{w}_r} f(\mathbf{x}|\mathbf{a}, \mathbf{W}) = \frac{1}{\sqrt{m}} a_r \sigma'(\mathbf{w}_r^\top \mathbf{x}) \mathbf{x},$$

For given training data $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^\top$ and $\mathbf{y} = [y_1, \ldots, y_n]^\top$ consider the MSE loss:

$$L(\mathbf{X}, \mathbf{W}, \mathbf{a}) = \frac{1}{2} \sum_{i=1}^{n} \left(y_i - \hat{f}(\mathbf{x}_i|\mathbf{a}, \mathbf{W}) \right)^2$$

To simplify the math, instead of considering (usual) discrete-time gradient descent, we will assume that during the training the parameters of our NN evolve according to the continuous-time gradient descent defined by the system of differential equations:

$$\frac{d\mathbf{w}_r(t)}{dt} = -\nabla_{w_i} L(\mathbf{X}, \mathbf{W}(t), \mathbf{a}(t)) = \sum_{i=1}^{n} \left(y_i - \hat{f}(\mathbf{x}_i|\mathbf{a}(t), \mathbf{W}(t)) \right) \nabla_{\mathbf{w}_r} f(\mathbf{x}_i|\mathbf{a}(t), \mathbf{W}(t)),$$

$$\frac{d\mathbf{a}(t)}{dt} = -\nabla_{\mathbf{a}} L(\mathbf{X}, \mathbf{W}(t), \mathbf{a}(t)) = \sum_{i=1}^{n} \Big(y_i - \hat{f}(\mathbf{x}_i|\mathbf{a}(t), \mathbf{W}(t))\Big) \nabla_{\mathbf{a}} f(\mathbf{x}_i|\mathbf{a}(t), \mathbf{W}(t)).$$

You can think of this process as of your usual gradient descent with infinitesimally small step size and a rescaling of time. It turns out that looking at the evolution of weights directly is not very convenient. The main idea is to look at the dynamics of the predictions on the training data: denote

$$u_i(t) := \hat{f}(\mathbf{x}_i|\mathbf{a}(t), \mathbf{W}(t)), \quad \mathbf{u}(t) := [u_1(t), \dots, u_n(t)]^\top.$$

Note that $\mathbf{u}(0) = \mathbf{0}$ because of how we simplified things. Then during the GD we have

$$\begin{aligned}
\frac{du_i(t)}{dt} &= \left\langle \nabla_{\mathbf{a}} f(\mathbf{x}_i|\mathbf{a}, \mathbf{W}), \frac{d\mathbf{a}(t)}{dt} \right\rangle + \sum_{r=1}^{m} \left\langle \nabla_{\mathbf{w}_i} f(\mathbf{x}_i|\mathbf{a}, \mathbf{W}), \frac{d\mathbf{w}_r(t)}{dt} \right\rangle \\
&= \sum_{j=1}^{n} \Big(y_j - u_j(t)\Big) \Big\langle \nabla_{\mathbf{a}} f(\mathbf{x}_i|\mathbf{a}, \mathbf{W}), \nabla_{\mathbf{a}} f(\mathbf{x}_j|\mathbf{a}, \mathbf{W}) \Big\rangle \\
&+ \sum_{j=1}^{n} \sum_{r=1}^{m} \Big(y_j - u_j(t)\Big) \Big\langle \nabla_{\mathbf{w}_r} f(\mathbf{x}_i|\mathbf{a}, \mathbf{W}), \nabla_{\mathbf{w}_r} f(\mathbf{x}_j|\mathbf{a}, \mathbf{W}) \Big\rangle
\end{aligned}$$

So, looking at the differential equation satisfied by the residuals (this should remind you of what you say in the review problem in HW0 when we looked at discrete-time gradient-descent for least-squares), we see:

$$\frac{d}{dt}(\mathbf{y} - \mathbf{u}(t)) = -(\mathbf{G}(t) + \mathbf{H}(t))(\mathbf{y} - \mathbf{u}(t)),$$

where $\mathbf{G}(t)$ and $\mathbf{H}(t)$ are empirical kernel matrices:

$$\begin{aligned}
\mathbf{G}_{i,j}(t) :=& \frac{1}{m} \sum_{r=1}^{m} \sigma(\mathbf{w}_r(t)^\top \mathbf{x}_i) \sigma(\mathbf{w}_r(t)^\top \mathbf{x}_j) \\
\mathbf{H}_{i,j}(t) :=& \mathbf{x}_i^\top \mathbf{x}_j \frac{1}{m} \sum_{r=1}^{m} a_r(t)^2 \sigma'(\mathbf{w}_r(t)^\top \mathbf{x}_i) \sigma'(\mathbf{w}_r(t)^\top \mathbf{x}_j)
\end{aligned}$$

As m goes to infinity, $\mathbf{G}(0)$ and $\mathbf{H}(0)$ converge to kernel Gram matrices with kernels $\mathbf{G}^\infty$ and $\mathbf{H}^\infty$ correspondingly:

$$\begin{aligned}
\mathbf{G}^\infty_{i,j} :=& \mathbb{E}_{\mathbf{w}}[\sigma(\mathbf{w}^\top \mathbf{x}_i) \sigma(\mathbf{w}^\top \mathbf{x}_j)], \\
\mathbf{H}^\infty_{i,j} :=& \mathbf{x}_i^\top \mathbf{x}_j \mathbb{E}[a^2] \mathbb{E}_{\mathbf{w}}[\sigma'(\mathbf{w}^\top \mathbf{x}_i) \sigma'(\mathbf{w}^\top \mathbf{x}_j)].
\end{aligned}$$

Denote

$$\mathbf{K}(t) := \mathbf{G}(t) + \mathbf{H}(t), \quad \mathbf{K}^\infty := \mathbf{G}^\infty + \mathbf{H}^\infty.$$

Note that $\mathbf{K}^\infty$ is exactly the kernel Gram matrix that corresponds to NTK. In this problem we assume that the matrix $\mathbf{K}^\infty$ is non-degenerate, i.e. $\lambda_{\min}(\mathbf{K}^\infty) > 0$. (This is a reasonable assumption since there are more features than data points.) Our goal is to show that under this condition training with gradient descent leads to the same prediction as kernel regression with the neural tangent kernel.

(a) The first observation that we are going to make is that if the lowest eigenvalue of our matrix $\mathbf{K}(t)$ remains separated from zero, then the residuals converge rapidly to zero loss. This rapid convergence will help us prove that our weights don't go very far during the training. Suppose that for some $T > 0$ for any $t \in [0, T]$ the lowest eigenvalue $\lambda_{\min}(\mathbf{K})$ of $\mathbf{K}(t)$ is at least $\bar{\lambda} > 0$. **Show that for any** $t \in [0, T]$

$$\|\mathbf{y} - \mathbf{u}(t)\|^2 \leq \|\mathbf{y}\|^2 \exp\left(-2t\bar{\lambda}\right)$$

Hint: consider $\frac{d}{dt}\|\mathbf{y} - \mathbf{u}(t)\|^2$. *Don't forget that* $\mathbf{u}(0) = \mathbf{0}$.

(b) As promised, now we want to show that our weights don't go very far and we will not get out of the local regime where the NTK is a good approximation. Suppose for some $T > 0$ there exist $\gamma > 0$ and $\bar{\lambda} > 0$ s.t. for any $t < T$ the following holds:

- for any r, $|a_r(t)| < \gamma$ (i.e. there is some bound on how big the final-layer weights become),
- $\lambda_{\min}(\mathbf{K}(t)) \geq \bar{\lambda}$

Show that for any $t < T$ **and any** r

$$\left\| \frac{d\mathbf{w}_r(t)}{dt} \right\| \leq \frac{1}{\sqrt{m}} \|\mathbf{y}\| \gamma C(\sigma, \mathbf{X}, n) \exp\left(-2t\bar{\lambda}\right),$$
$$\left| \frac{da_r(t)}{dt} \right| \leq \frac{1}{\sqrt{m}} \|\mathbf{y}\| C(\sigma, \mathbf{X}, n) \exp\left(-2t\bar{\lambda}\right)$$

where $C(\sigma, \mathbf{X}, n)$ **is some constant that only depends on** σ**,** $\mathbf{X}$ **and** n**.**

Contributors:

- Alexander Tsigler
- Anant Sahai
- Chawin Sitawarin
- Joshua Sanz