E0 306: Deep Learning: Theory and Practice 24^{th} January 2019

Scribe Notes for Lecture 6

Lecturer: Navin Goyal *Scribe*: A Saxena

1 Review of neural networks - computational graphs

Continuing from the last lecture, following is a fully connected computational graph for a function $X \to Y$

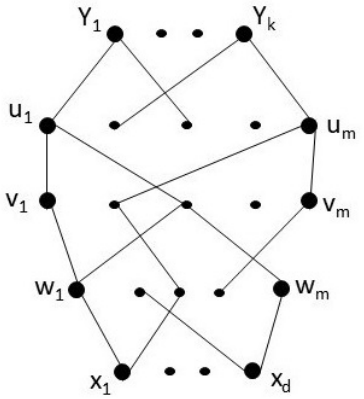


Figure 1: Computational Graph

$$Y_i : \mathbb{R}^m \to \mathbb{R}, i \in [k]$$
$$u_i : \mathbb{R}^m \to \mathbb{R}, i \in [m]$$
$$v_i : \mathbb{R}^m \to \mathbb{R}, i \in [m]$$
$$w_i : \mathbb{R}^m \to \mathbb{R}, i \in [m]$$
$$x_i \in \mathbb{R}, i \in [d]$$
$$Y = y \circ u \circ v \circ w \circ x$$
$$Y_i = Y_i(u_1, u_2, ..., u_m)$$
$$u_j = u_j(w_1, w_2, ..., w_m)$$

Goal: Compute $\frac{\partial Y_j}{\partial x_i}$ for $j \in [k]$ and $i \in [d]$ to compute Jacobian.

Claim: We can compute Jacobian $[\frac{\partial Y_i}{\partial x_j}]$, $i \in [k], j \in [d]$ in $O(\min(k, d) * (V + E))$ time where V and E are the number of vertices and edges in the graph respectively.

Note: *Automatic Differentiation*: it is a set of techniques to numerically evaluate the derivative of a function specified by a computer program. AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations (addition, subtraction, multiplication, division, etc.) and elementary functions (exp, log, sin, cos, etc.). By applying the chain rule repeatedly to these operations, derivatives of arbitrary order can be computed automatically, accurately to working precision, and using at most a small constant factor more arithmetic operations than the original program.

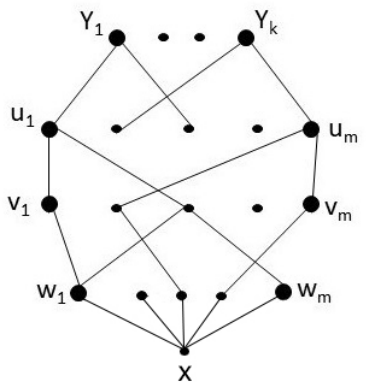


Figure 2: Computational Graph with d = 1

Special cases:

1. If $d = 1$, then we can compute Jacobian in time $O(E)$ for any k

   There are 2 modes for AD: Forward mode AD and Backward mode AD (*Backpropagation*). Both touch each edge of the graph only once.

   Proof for forward mode AD:

   (a) Compute derivatives $\frac{\partial w_i}{\partial x}$, $i \in [m]$ - time taken $O(m)$

   (b) Compute derivatives $\frac{\partial v_i}{\partial w_j}$, $i, j \in [m]$ - time taken $O(m^2)$

   (c) Compute $\frac{\partial v_i}{\partial x} = \frac{\partial v_i}{\partial w_1}.\frac{\partial w_1}{\partial x} + \frac{\partial v_i}{\partial w_2}.\frac{\partial w_2}{\partial x} + ... + \frac{\partial v_i}{\partial w_m}.\frac{\partial w_m}{\partial x}$. m multiplications to get $\frac{\partial v_i}{\partial x}$ so total $O(m^2)$ time to get $\frac{\partial v_i}{\partial x}, i \in [m]$

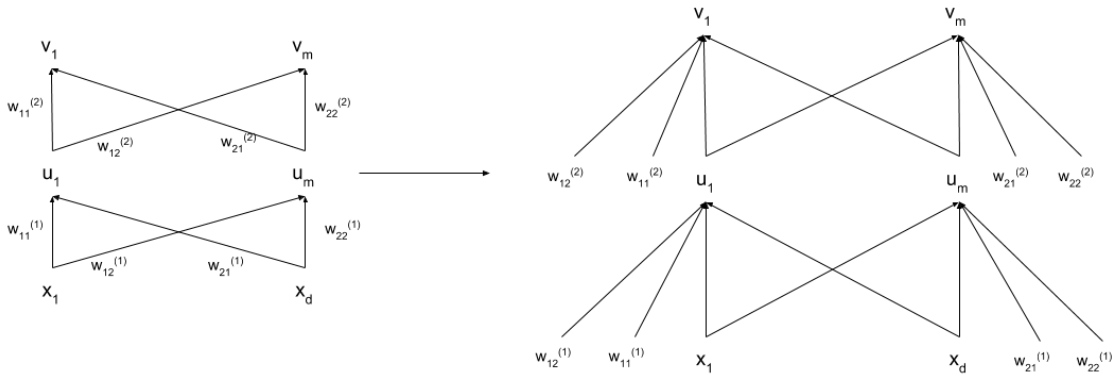
   (d) So total time taken to get $\frac{\partial v_i}{\partial x}, i \in [m]$ is $O(m^2)$

   (e) Continue this procedure till last layer. No. of operations = 1 for each edge since there are m^2 edges between 2 hidden layers (fully connected graph)

2. If $k = d = m$ and ℓ is the number of layers in the graph, then time complexity of computing Jacobian is $O(\ell * m^3)$. Using fast matrix multiplication, it becomes $O(\ell * m^{2.39})$ approx.

3. If we can solve this problem in $O(\ell * m^2)$ time then we can multiply matrics in $O(m^2)$ (non-trivial proof omitted)

1.1 Neural networks as computational graphs



This figure shows how a neural network can be unfolded into a computational graph as described earlier.

2 Initialization of weights for gradient descent

2.1 Exploding and vanishing gradients

Problem: Let $\ell(w; S)$ be the loss function.$\nabla_w \ell(w; S)$ can become very large or very small depending on the value of w.

Example: Let there be a network with 1 scalar input x, L hidden layers with 1 neuron each and identity activation function for each neuron and 1 scalar output y.

$$y = w_L * w_{L-1} * ... * w_2 * w_1 * x$$

The i^{th} component of $\nabla_w y$ will be

$$\nabla_w y^i = w_L * w_{L-1} * ... * w_{i-1} * w_{i+1} * ... * w_2 * w_1 * x$$

So if, for example, w_j are initialized as 2, this value explodes to become 2^{L-1} . On the other hand, if w_j are initialized as $\frac{1}{2}$, it becomes $2^{-(L-1)}$. Since $\ell(w; S)$ is a function of y, $\nabla_w \ell(w; S)$ depends on $\nabla_w y$ and hence it can explode or vanish depending on the size of these weights which causes problems when doing gradient descent.
However, if w_j are close to 1, the gradient neither explodes nor vanishes.

2.2 He initialization for ReLU networks [1]

We assume that data is normalized. To do this set

$$x_i \leftarrow x_i - \frac{1}{m} \sum_j x_j$$

He initialization: Given a fully connected network of L layers where the size of each layer l is d_l with weight matrix $w^{(l)}$, we initialize each weight in $w^{(l)}$ as iid

$$w_{ij}^{(l)} \sim N(0, \sqrt{\frac{2}{d_l}})$$

Note: This is similar to what was done for *tanh* networks in Xavier initialization by Glorot and Bengio, 2010 [2] where weights were initialized as:

$$w_{ij}^{(l)} \sim N(0, \sqrt{\frac{1}{d_l}})$$

2.2.1 Proof of heuristic used in He initialization

Notation:

$$y^{(l)} = w^{(l)}x^{(l)} + b^{(l)}, 0 \leq l \leq L$$
$$x^{(l+1)} = \text{ReLU}(y^{(l)})$$
$$x^{(0)} = x$$

Assumptions:

- $x^{(l)}$ is a random variable with iid components
- $x^{(l)}$ and $w^{(l)}$ are independent
- $b^{(l)} = 0$

We will show the theoretical basis for He initialization.

$$\begin{aligned}\text{Var}(y_1^{(l)}) &= \text{Var}(\sum_i w_{1i}^{(l)} x_1^{(l)}) \\ &= d_l \text{Var}(w_{1i}^{(l)} x_1^{(l)}) \end{aligned} \tag{1}$$

We know that if X and Y are independent and $\mathbb{E}(Y) = 0$, then

$$\begin{aligned}\text{Var}(XY) &= \mathbb{E}(X^2)\mathbb{E}(Y^2) - \mathbb{E}(X)^2\mathbb{E}(Y)^2 \\ &= \text{Var}(Y^2)\mathbb{E}(X^2) \end{aligned} \tag{2}$$

So if we let $w_{1i}^{(l)}$ have 0 mean, then from equations (1) and (2) we get

$$\text{Var}(y_1^{(l)}) = d_l \text{Var}(w_{1i}^{(l)}) * \mathbb{E}[(x_i^{(l)})^2]$$

If we also let $w_{1i}^{(l)}$ have a symmetric distribution around 0 then y also has zero mean and symmetric distribution around zero. So

$$\begin{aligned}\mathbb{E}(x_1^{(l)})^2 &= \mathbb{E}[\text{ReLU}(y_1^{(l-1)})]^2 \\ &= \frac{1}{2}\mathbb{E}(y_1^{(l-1)})^2 \\ &= \frac{1}{2}\text{Var}(y_1^{(l-1)}) \end{aligned} \tag{3}$$

So we get

$$\text{Var}(y_1^{(l)}) = \frac{d_l}{2}\text{Var}(w_{1i}^{(l)}) * \text{Var}(y_1^{(l-1)})$$

By recursively substituting for $\text{Var}(y^{(l)})$ we get

$$\text{Var}(y_1^{(L)}) = \text{Var}(y_1^{(1)}) * [\frac{d_2}{2}\text{Var}(w_1 i^{(1)})] * ... * [\frac{d_L}{2}\text{Var}(w_1 i^{(L)})]$$

We want each of these terms to be close to 1 so that the overall variance doesn't blow up or vanish. This can be achieved by setting

$$\text{Var}(w_{1i}^{(l)}) = \frac{2}{d_l}$$

3 Limitations of gradient based methods for deep learning

We work in the realizable setting of PAC learning. $\mathcal{X}$ is domain set, $\mathcal{Y}$ is label set and $\mathcal{H}$ is the set of hypotheses.

$$\mathcal{D}_x \sim \textit{distribution on } \mathcal{X}$$

$$(x, y) = (x, h(x))$$

for some unknown target $h \in \mathcal{H}$

3.1 Gradient based methods

We define a loss function

$$L(w; S) = \sum_{(x,y)\in S} L(w; (x, y))$$

We need to minimize $L(w; S)$ wrt w. For gradient descent, this is done by applying the following update.

$$w^{(t+1)} = w^{(t)} - \eta^{(t)} \nabla_w L(W; S)$$

From paper *Failures of gradient based deep learning (Shalev-Schwartz et al., 2017)*[3]

Claim 1 : *If we choose hypothesis $h \in \mathcal{H}$ uniformly, then with high probability, $\nabla_w L(w; S)$ is essentially independent of h*

Say

$$L(w) = \mathbb{E}_{(x,y)\sim D} L(w; (x, y))$$

and $h \in \mathcal{H}$ is the target function. $w \in \mathbb{R}^n$
Define $F_h(w)$ as

$$F_h(w) := L(w)$$

$$F_h(w) = \mathbb{E}_{x\sim D} l(p_w(x), h(x))$$

Where p is the predictor parameterized by w and l is a loss function of type

$$l = \frac{1}{2}(y - \hat{y})^2$$

or

$$l = \gamma(y.\hat{y})$$

where γ is a 1-Lipschitz function.
Assumption: $F_h(W)$ is differentiable wrt W.
Define

$$\text{Var}(F, \mathcal{H}, w) := \mathbb{E}_{h \epsilon_U \mathcal{H}} ||\nabla_w F_h(w) - \mathbb{E}_{h' \epsilon_U \mathcal{H}} \nabla_w F_{h'}(w)||^2$$

If variance is small, then Claim-1 is true.

Theorem 1: Suppose that

1. Functions in $\mathcal{H}$ are of type $\mathcal{X} \to \{-1, 1\}$
2. $\mathbb{E}_{x \sim D_x} h(x)h'(x) = 0$ for distinct $h, h' \in \mathcal{H}$
3. $p_w(x)$ is differentiable wrt w and satisfies

   $$\mathbb{E}_x ||\nabla_w p_w(x)||^2 \leq G(w)^2$$

   for some function $G(w)$
4. The loss ℓ is either square loss or $\gamma(y.\hat{y})$ where γ is 1-Lipschitz

Then

$$\text{Var}(\mathcal{H}, F, w) \leq \frac{G(w)^2}{|\mathcal{H}|}$$

[proof for this theorem for parity functions is done in next lecture]

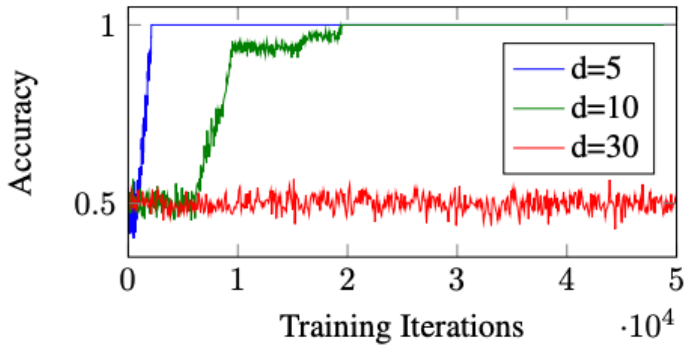


Figure 3: Parity Experiment: Accuracy as a function of the number of training iterations, for various input dimensions [3]

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 1026–1034, Washington, DC, USA, 2015. IEEE Computer Society.

[2] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[3] Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. Failures of deep learning. *CoRR*, abs/1703.07950, 2017.