

10417/10617
Intermediate Deep Learning:
Fall 2023

Yuanzhi Li / Russ Salakhutdinov
Machine Learning Department



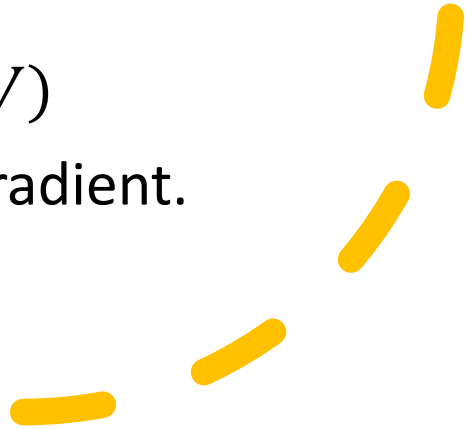
Training of neural networks: Backpropagation

- Give a machine learning model $h(W, x)$, where W is the parameter, x is the input.
- For example: $h(W, x) = \langle W, x \rangle$ for a linear model.
- For MLPs: $h(W, x) = W_{L+1} \sigma(W_L \sigma(W_{L-1} \sigma \circ \dots \circ \sigma(W_1 x + b_1) \dots + b_{L-1}) + b_L) + b_{L+1}$
- For $W = (W_{L+1}, W_L, \dots, W_1, b_{L+1}, b_L, \dots, b_1)$.

Training of neural networks


- Give a machine learning model $h(W, x)$, where W is the parameter, x is the input.
- Given training data $\{x^{(i)}, y^{(i)}\}_{i \in [N]}$, we typically train the models using the Empirical Risk Minimization (ERM) objective.
- $\min_W \frac{1}{N} \sum_{i \in [N]} l(h(W, x^{(i)}), y^{(i)}) + R(W)$
- Here $R(W)$ is the regularizer, typically the norm of W .
- l is a loss function, typically chosen as MSE loss, Cross Entropy Loss, etc.

Training of neural networks

- $\min_W \frac{1}{N} \sum_{i \in [N]} l(h(W, x^{(i)}), y^{(i)}) + R(W)$
 - We want to find the parameter W of the model h , so given the input $x^{(i)}$, the prediction of the model ($h(W, x^{(i)})$) is as close to the actual label $y^{(i)}$ as possible.
 - To find the minimizer, we typically use the gradient descent based approach:
 - Compute the gradient of $L(W) = \frac{1}{N} \sum_{i \in [N]} l(h(W, x^{(i)}), y^{(i)}) + R(W)$
 - Then update W according to the gradient.
- 



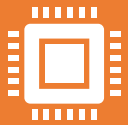
Gradient computation

- Compute the gradient of $L(W) = \frac{1}{N} \sum_{i \in [N]} l(h(W, x^{(i)}), y^{(i)}) + R(W)$
 - Main question: How do we compute the gradient of L w.r.t. W efficiently?
 - This is the main topic we want to learn in the lecture today.
- 

Gradient computation

- When $h(W, x)$ is a linear function ($h(W, x) = \langle W, x \rangle$), we can compute
- $\nabla l(h(W, x^{(i)}), y^{(i)}) = l'(h(W, x^{(i)}), y^{(i)}) \times \nabla h(W, x^{(i)})$
- $\nabla l(h(W, x^{(i)}), y^{(i)}) = l'(h(W, x^{(i)}), y^{(i)}) x^{(i)}$
- But what if $h(W, x)$ is a neural network?

Backpropagation



We will learn how to compute the gradient of L , when $h(W, x)$ is a neural network.



We will learn the most important, and fundamental algorithm for deep learning: Backpropagation.



Backpropagation is very typically used as interview questions in tech companies, for deep learning coding.

Backpropagation

- $\nabla l(h(W, x^{(i)}), y^{(i)}) = l'(h(W, x^{(i)}), y^{(i)}) \times \nabla h(W, x^{(i)})$
- $h(W, x) = W_{L+1} \sigma(W_L \sigma(W_{L-1} \sigma \circ \dots \circ \sigma(W_1 x + b_1) \dots + b_{L-1}) + b_L) + b_{L+1}$
- How do I compute the **gradient of h** w.r.t some W_l ?

Backpropagation

- $h(W, x) = W_{L+1} \sigma(W_L \sigma(W_{L-1} \sigma \circ \dots \circ \sigma(W_1 x + b_1) \dots + b_{L-1}) + b_L) + b_{L+1}$
- Key observation: We can write (for some function F_l)
- $h(W, x) = F_l(W_l h_l(x) + b_l)$
 - Let's w.l.o.g consider the output of h is in \mathbb{R} (otherwise, we can look at each dimension separately)
- Then by chain rule, $\nabla_{W_l} h(W, x) = \nabla F_l(W_l h_l(x) + b_l) h_l(x)^T$

Backpropagation

- $\nabla_{W_l} h(W, x) = \nabla F_l(W_l h_l(x) + b_l) h_l(x)^T$
- So we need two things:
 - $h_l(x)$: This is easy to compute using the forward pass.
 - $\nabla F_l(W_l h_l(x) + b_l)$: How do we compute this?
- How do we compute $\nabla F_l(z_l)$? For $z_l := W_l h_l(x) + b_l$

Backpropagation

- How do we compute $\nabla F(z_l)$?
- Observation: if we define
 - $F_l(z_l) = W_{L+1} \sigma(W_L \sigma(W_{L-1} \sigma \circ \dots \circ \sigma(W_{l+1} \sigma(z_l) + b_{l+1}) \dots + b_{L-1}) + b_L) + b_{L+1}$
- Then $F_l(z_l) = F_{l+1}(W_{l+1} \sigma(z_l) + b_{l+1})$

Backpropagation

- $F_l(z_l) = F_{l+1}(W_{l+1}\sigma(z_l) + b_{l+1})$
- Again, by chain rule, we have that:
- $\nabla F_l(z_l) = W_{l+1}^T \nabla F_{l+1}(W_{l+1}\sigma(z_l) + b_{l+1}) \otimes \sigma'(z_l)$
 - Here \otimes means element-wise multiplication.
- $\nabla F_l(z_l) = W_{l+1}^T \nabla F_{l+1}(z_{l+1}) \otimes \sigma'(z_l)$



Backpropagation

- $\nabla F_l(z_l) = W_{l+1}^T \nabla F_{l+1}(z_{l+1}) \otimes \sigma'(z_l)$
 - If we denote $\nabla F_l(z_l) = g_l$
 - Then we have: $g_l = W_{l+1}^T g_{l+1} \otimes \sigma'(z_l)$
 - Where $g_{L+1} = 1$
 - This is a backward induction formula, so the algorithm is called backpropagation.
-

Backpropagation

- In summary, to compute $\nabla_{W_l} h(W, x)$
- We know that $\nabla_{W_l} h(W, x) = g_l h_l(x)^T$
- Where $g_l = W_{l+1}^T g_{l+1} \otimes \sigma'(z_l)$
- To compute $h_l(x)$, we need to do a forward pass.
- To compute g_l , we need to do a backward pass.

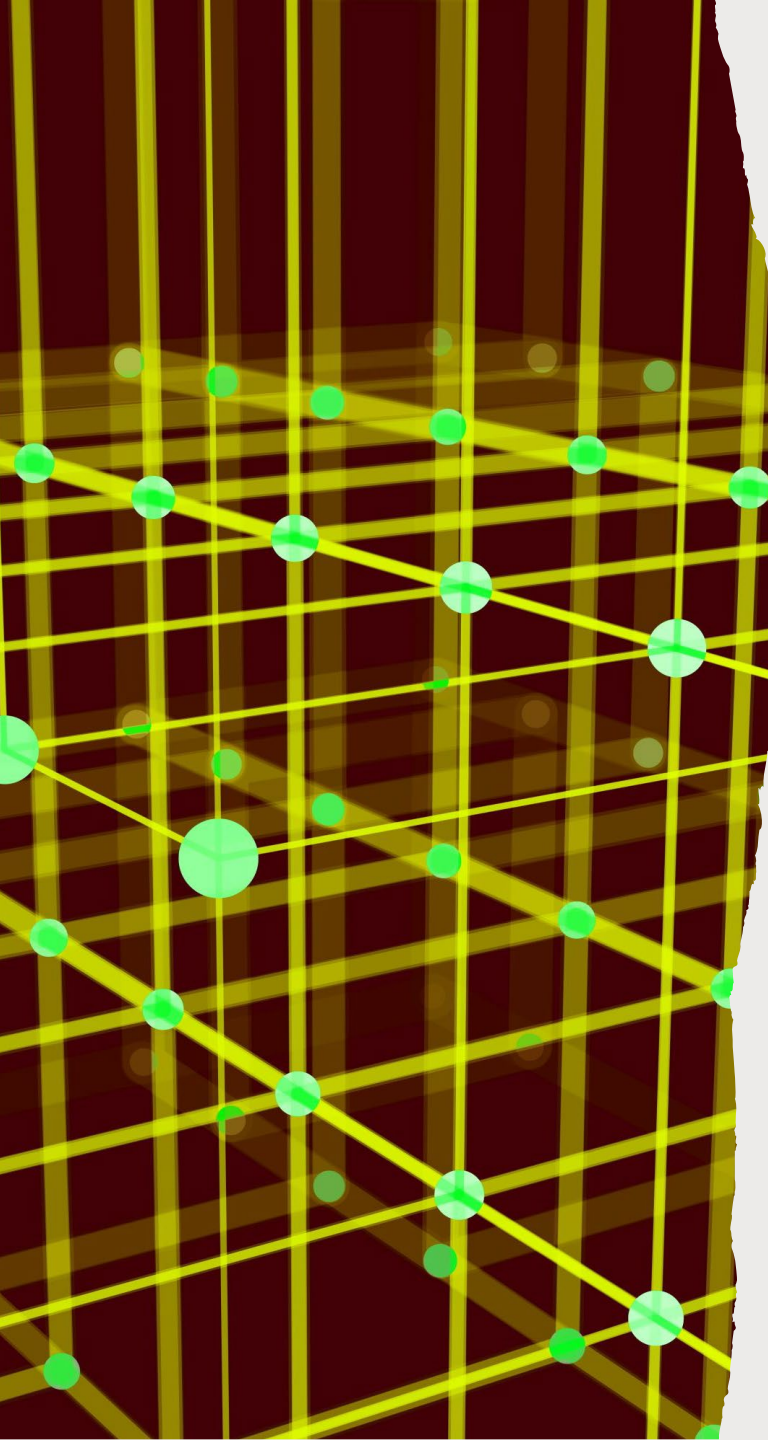
Computation time and memory usage.

- So what is the computation time and memory usage of backpropagation?
 - For simplicity, let's consider the case where all $d_l = d$ for $l \leq L + 1$
 - Let's first think about computing $h(W, x)$ (forward pass)
 - Computation time: $O(d^2L)$ (there are L many $d \times d$ matrix times $d \times 1$ vector operations).
 - Memory usage: $O(d)$ (only need to store $h_l(x)$ when computing $h_{l+1}(x)$)
-



Computation time and memory usage.

- Backpropagation:
- We know that $\nabla_{W_l} h(W, x) = g_l h_l(x)^T$
- Where $g_l = W_{l+1}^T g_{l+1} \otimes \sigma'(z_l)$
- Computation time:
 - $O(d^2 L)$ to compute every $h_l(x)$ and $\sigma'(z_l)$
 - $O(d^2 L)$ to compute every g_l
 - So total time is still $O(d^2 L)$ – **As fast as the forward pass.**
- Memory usage:
 - $O(d \cdot L)$ to memorize every $h_l(x)$ and $\sigma'(z_l)$
 - $O(d \cdot L)$ to memorize every g_l
 - This is much higher than the $O(d)$ memory usage for forward pass.
 - For example, for a 96 layers model, it uses $96 \times 3 = 288$ times more memory!



Model parallel in backpropagation

- Key challenge: What if $O(dL)$ is too large for my GPU's memory?
- A100 has 80G memory, it can maximally hold the backpropagation of a ~7B transformer model with context length 2048.
 - What if I want to train a 70B parameter model?
- Solution: Model parallel/Pipeline parallel

Pipeline parallel in backpropagation

- Key idea: Suppose we have M gpus, then we divide the weights into M groups:
 - $(W_1, W_2, \dots, W_{\frac{L+1}{M}}, b_1, b_2, \dots, b_{\frac{L+1}{M}})$
 - $(W_{\frac{L+1}{M}+1}, W_{\frac{L+1}{M}+2}, \dots, W_{2\frac{L+1}{M}}, b_{\frac{L+1}{M}+1}, b_{\frac{L+1}{M}+2}, \dots, b_{2\frac{L+1}{M}})$
 - ...
- Each GPU only operates on one group of weights.

Pipeline parallel in backpropagation

- We know that $\nabla_{W_l} h(W, x) = g_l h_l(x)^T$
- Where $g_l = W_{l+1}^T g_{l+1} \otimes \sigma'(z_l)$
- Model parallel
 - Gpu-1 computes $h_{\frac{L+1}{M}}(x)$ and sends to gpu2.
 - Gpu-2 computes $h_{2\frac{L+1}{M}}(x)$ and sends to gpu3.
 - ...
 - Gpu-M computes $h_{L+1}(x)$, then computes $g_{(M-1)\frac{L+1}{M}+1}$ and sends to gpu-(M - 1)
 - Gpu-(M-1) computes $g_{(M-2)\frac{L+1}{M}+1}$ and send to gpu-(M - 2)
 - ...
 - Gpu-(2) computes $g_{\frac{L+1}{M}+1}$ and sends to gpu-1
- Memory usage per GPU: $O(\frac{dL}{M})$

Model parallel in backpropagation

- Also known as Tensor Parallel.
- To compute $\nabla_{W_l} h(W, x)$
 - We know that $\nabla_{W_l} h(W, x) = g_l h_l(x)^T$
 - Where $g_l = W_{l+1}^T g_{l+1} \otimes \sigma'(z_l)$
- Key idea: each GPU only stores $\frac{d}{M}$ coordinates of each g_l , h_l and $\sigma'(z_l)$
- Communication cost is higher than pipeline parallel, but computation speed is faster.

Backpropagation for general computation graph



We learned backpropagation to compute the gradient in MLP.



But in later lectures, we will learn neural networks that are not simply MLP (normalization layers, attention layers, etc.)



We want to have a generic algorithm to compute the gradient w.r.t. any computation graph.

Backpropagation for general computation graph

- A computation graph is defined by a **directed acyclic graph G**.
- Each vertex v is associated with a function F_{W_v} , the function takes input the outputs of all the in-neighbors of v , then outputs a vector.
- The output of the vertex v is this vector.
- The node without any in-neighbors is the input node (x). The node without any out-neighbors is the output node ($h(W, x)$)
- $W = (W_v)_{v \in V(G)}$

Backpropagation for general computation graph

- How do we compute the gradient of $h(W, x)$ with respect to the weight W_v ?
- We can write $h(W, x) = G \left(\left\{ F_{W_{v'}} \left(F_{W_v}(z_v), * \right) \right\}_{v' \text{ are the out neighbors of } v} \right)$
- Here $*$ denotes the other inputs to $F_{W_{v'}}$.
- z_v is the input to F_{W_v} .

Backpropagation for general computation graph

- $h(W, x) = G \left(\left\{ F_{W_{v'}} (F_{W_v}(z_v), *) \right\}_{v' \text{ are the out neighbors of } v} \right)$
- Again, let's consider the case that the output dimension of h is 1.
- Then we know that for $r_v = F_{W_v}(z_v)$, $**$ is the input to G :
- $\nabla_{W_v} h(W, x) = \sum_{v' \text{ are the out neighbors of } v} \sum_i \nabla_{W_v} [F_{W_v}(z_v)_i] \nabla_{[r_v]_i} F_{W_{v'}}(r_v, *) \nabla_{v'} G(**)$

Backpropagation for general computation graph

- $\nabla_{W_v} h(W, x) = \sum_{v' \text{ are the out neighbors of } v} \sum_i \nabla_{W_v} F_{W_v}(z_v)_i \nabla_{[r_v]_i} F_{W_{v'}}(r_v, *) \nabla_{v'} G(**)$
- The induction relationship:
 - $\nabla_v G = \sum_{v' \text{ are the out neighbors of } v} \nabla_{r_v} F_{W_{v'}}(r_v, *) \nabla_{v'} G$
- So, we still need
 - Forward pass, in order to compute z_v , $r_v = F_{W_v}(z_v)$.
 - Backward pass, in order to compute ∇G_v