ECE 449/590 – OOP and Machine Learning
Lecture 19 Back-Propagation

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

October 31, 2022

Reading Assignment

- This lecture: Deep Learning 6.5
- Next lecture: Deep Learning 7, 8

Forward and Back-Propagation

- Forward propagation: compute $\hat{\boldsymbol{y}}$ from $\boldsymbol{x}$
  - During training to compute $J(\boldsymbol{\theta})$.
  - During inference to make predictions.
- Back-propagation, a.k.a. backprop: compute $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
  - During training, after $J(\boldsymbol{\theta})$ is computed.
  - To support SGD and its variants.
  - An algorithm to compute gradient/Jacobian for arbitrary function represented as DAG in general.

Computation Graphs

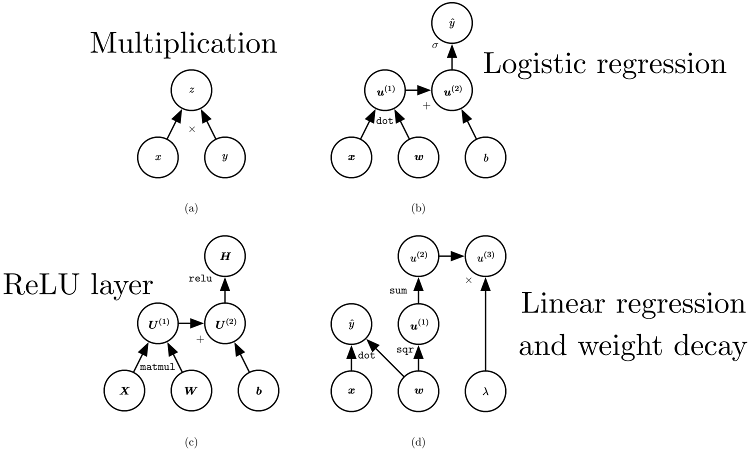


Figure 6.8

(Goodfellow 2017)

- Nodes are named by their outputs.

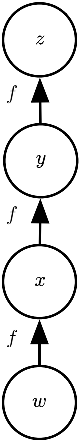
The Chain Rule (1-D)

- Scalar functions: for $y = g(x)$ and $z = f(y) = f(g(x))$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

- Let $z = h(x)$, the chain rule says $h'(x) = f'(g(x))g'(x)$.
- We can calculate the derivative of z to x as long as we can calculate those for g and f.
  - In addition, we would need to know $g(x)$ – this is available from the forward propagation.

Repeated Subexpressions



$$\frac{\partial z}{\partial w} \tag{6.50}$$

$$= \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w} \tag{6.51}$$

$$= f'(y)f'(x)f'(w) \tag{6.52}$$

$$= f'(f(f(w)))f'(f(w))f'(w) \tag{6.53}$$

Back-prop avoids computing this twice

Figure 6.9

(Goodfellow 2017)

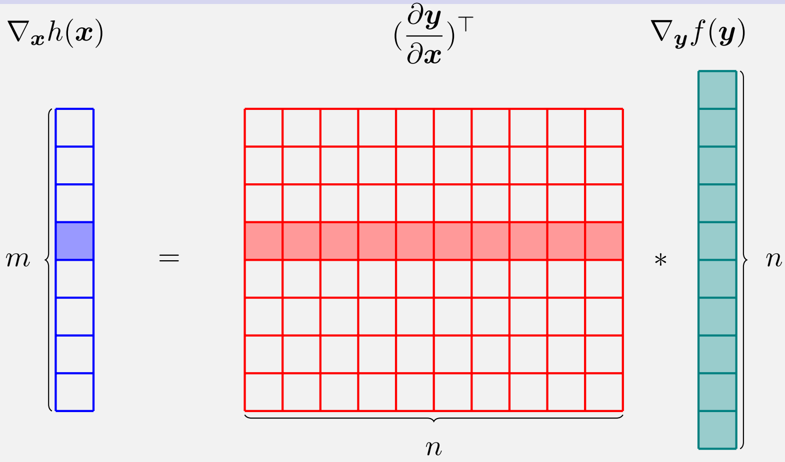
The Chain Rule I

- For $\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x})$ and $z = f(\boldsymbol{y}) = f(\boldsymbol{g}(\boldsymbol{x})) = h(\boldsymbol{x})$,

$$\nabla_{\boldsymbol{x}} z = (\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}})^\top \nabla_{\boldsymbol{y}} z$$

- Let $\boldsymbol{x} \in \mathbb{R}^m$, $\boldsymbol{y} \in \mathbb{R}^n$.
  - $\nabla_{\boldsymbol{x}} z \in \mathbb{R}^m$ is $\nabla_{\boldsymbol{x}} h(\boldsymbol{x}) = (\frac{\partial h}{\partial x_1}(\boldsymbol{x}), \ldots, \frac{\partial h}{\partial x_m}(\boldsymbol{x}))$.
  - $\nabla_{\boldsymbol{y}} z \in \mathbb{R}^n$ is $\nabla_{\boldsymbol{y}} f(\boldsymbol{y}) = (\frac{\partial f}{\partial y_1}(\boldsymbol{y}), \ldots, \frac{\partial f}{\partial y_n}(\boldsymbol{y}))$.
- Write $\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x})$ as $y_1 = g_1(\boldsymbol{x}), \ldots, y_n = g_n(\boldsymbol{x})$.
  - The Jacobian $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \in \mathbb{R}^{n \times m}$ has $\frac{\partial g_j}{\partial x_i}(\boldsymbol{x})$ on its jth row for $j = 1, 2, \ldots, n$ and ith column for $i = 1, 2, \ldots, m$.

The Chain Rule II



$$\frac{\partial h}{\partial x_i}(\boldsymbol{x}) = \sum_{j=1}^{n} \frac{\partial f}{\partial y_j}(\boldsymbol{y}) \frac{\partial g_j}{\partial x_i}(\boldsymbol{x})$$

- Need to reuse $\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x})$ calculated from forward propagation.

The Backprop Algorithm

- Idea: apply chain rules recursively to compute gradients.
  - Utilize values calculated in forward propagation to save computations.
- Input: $h(\boldsymbol{x}) = f(\boldsymbol{g}(\boldsymbol{x}))$
- Output: $\nabla_{\boldsymbol{x}} h(\boldsymbol{x})$ at a given $\boldsymbol{x} = \boldsymbol{a}$.
- Details
  - Evaluate $\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x})$ by forward propagation for $\boldsymbol{x} = \boldsymbol{a}$ to obtain $\boldsymbol{y} = \boldsymbol{b} = \boldsymbol{g}(\boldsymbol{a})$.
  - Compute Jacobian of $\boldsymbol{g}(\boldsymbol{x})$ directly at $\boldsymbol{x} = \boldsymbol{a}$.
  - Compute $\nabla_{\boldsymbol{y}} f(\boldsymbol{y})$ for $\boldsymbol{y} = \boldsymbol{b}$ directly if possible or apply Backprop (recursively) to compute it.
  - Use the chain rule to compute $\nabla_{\boldsymbol{x}} h(\boldsymbol{x})$ for $\boldsymbol{x} = \boldsymbol{a}$ using Jacobian of $\boldsymbol{g}(\boldsymbol{x})$ at $\boldsymbol{x} = \boldsymbol{a}$ and $\nabla_{\boldsymbol{y}} f(\boldsymbol{y})$ at $\boldsymbol{y} = \boldsymbol{b}$.
- The recursive step computes gradients from outputs toward inputs layer by layer – that's why the algorithm is called Backprop.

Symbol-to-Symbol Differentiation

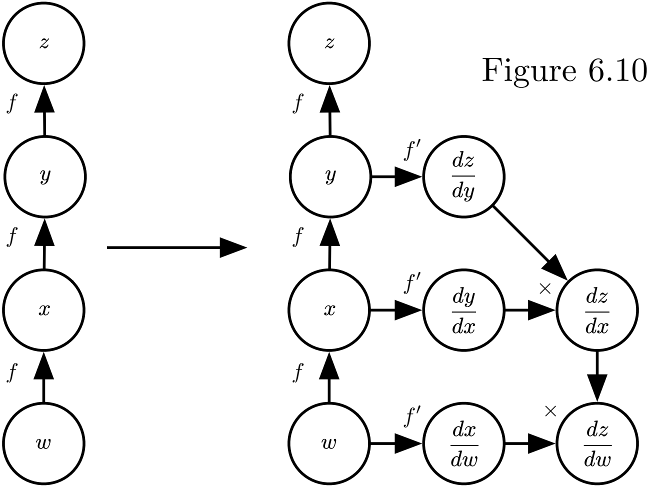


Figure 6.10

(Goodfellow 2017)

Some Simple Cases

- $\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x})$, $z = f(\boldsymbol{y}) = f(\boldsymbol{g}(\boldsymbol{x})) = h(\boldsymbol{x})$. Assume $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$.
- For $g_i(\boldsymbol{x}) = a_i x_i + b_i$, the Jacobian is simply $\mathrm{diag}(a_1, \ldots, a_n)$.

$$\frac{\partial h}{\partial x_i}(\boldsymbol{x}) = a_i \frac{\partial f}{\partial y_i}(\boldsymbol{g}(\boldsymbol{x}))$$

- For $g_i(\boldsymbol{x}) = \mathrm{ReLU}(x_i)$, the Jacobian is $\mathrm{diag}(u(x_1), \ldots, u(x_n))$.

$$\frac{\partial h}{\partial x_i}(\boldsymbol{x}) = u(x_i) \frac{\partial f}{\partial y_i}(\boldsymbol{g}(\boldsymbol{x}))$$

  - $u(x)$ is the step function: 0 if $x \leq 0$ and 1 otherwise.
- Similar results when $\boldsymbol{g}(\boldsymbol{x})$ is element-wise.

Cross-Entropy Loss Function I

- Assume there are C classes and each minibatch has N examples.
  - Let $\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(N)} \in \mathbb{R}^C$ be generated from the output units.
  - Let $y_1, \ldots, y_N$ be the training labels. They are constants.
- Cross-entropy loss with softmax,

$$J(\boldsymbol{\theta}) = -\frac{1}{N} \sum_{k=1}^{N} \log \frac{exp(z_{y_k}^{(k)})}{\sum_{j=1}^{C} exp(z_j^{(k)})} = \frac{1}{N} \sum_{k=1}^{N} L_{y_k}(\boldsymbol{z}^{(k)})$$

  - Loss per example: $L_y(\boldsymbol{z}) = -z_y + \log \sum_{j=1}^{C} exp(z_j)$
- To compute $\nabla_{\boldsymbol{\theta}} J$ as a whole would require to apply backprop to all examples, e.g. to obtain $\nabla_{\boldsymbol{z}^{(1)},\ldots,\boldsymbol{z}^{(N)}} J$ first.
- Alternatively, we may apply Backprop one example at a time.
  - Compute $\nabla_{\boldsymbol{\theta}} L_{y_k}(\boldsymbol{z}^{(k)})$ via Backprop for each $k = 1, 2, \ldots, N$.
  - Then calculate the average.
  - Easier to understand conceptually.

Cross-Entropy Loss Function II

$$L_y(\boldsymbol{z}) = -z_y + \log \sum_{j=1}^{C} exp(z_j)$$

- As discussed in the previous slide, let's focus on $\boldsymbol{z}$ from a single training example $\boldsymbol{x}$ and the training label y.
- For $j \neq y$,

$$\frac{\partial L_y}{\partial z_j} = \frac{\frac{\partial}{\partial z_j} \sum_{j'=1}^{C} exp(z_{j'})}{\sum_{j'=1}^{C} exp(z_{j'})} = \frac{exp(z_j)}{\sum_{j'=1}^{C} exp(z_{j'})}$$

- For $j = y$,

$$\frac{\partial L_y}{\partial z_y} = \frac{exp(z_y)}{\sum_{j'=1}^{C} exp(z_{j'})} - 1$$

- Note that the index j' is used for the summation in softmax.

Linear Layers I

- Consider a minibatch of N examples.
  - Inputs with I features: $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N)} \in \mathbb{R}^I$
  - Outputs with O features: $\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(N)} \in \mathbb{R}^O$
  - Computed by forward propagation.
- Parameters: $\boldsymbol{W} \in \mathbb{R}^{I \times O}, \boldsymbol{b} \in \mathbb{R}^O$
  - The function $\boldsymbol{g}$: $\boldsymbol{v}^{(k)} = \boldsymbol{g}(\boldsymbol{x}^{(k)}, \boldsymbol{W}, \boldsymbol{b}) = \boldsymbol{W}^\top \boldsymbol{x}^{(k)} + \boldsymbol{b}$.
- The loss function $J(\theta) = \frac{1}{N} \sum_{k=1}^{N} L_{y_k}(\boldsymbol{z}^{(k)})$
  - For each example k, $\boldsymbol{z}^{(k)}$ are computed from $\boldsymbol{u}^{(k)}$.
  - Loss per example:
    $L_y(\boldsymbol{z}) = f_y(\boldsymbol{v}) = f_y(\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b})) = h_y(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b})$.
- Similar to previous slides, we would like to apply Backprop per example to L_y instead of J.

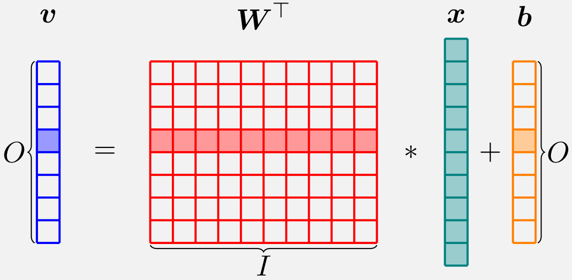
Linear Layers II

$$L_y(\boldsymbol{z}) = f_y(\boldsymbol{v}) = f_y(\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b})) = h_y(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b})$$

- Known from Backprop recursively: $(\frac{\partial f_y}{\partial v_1}, \ldots, \frac{\partial f_y}{\partial v_O})^\top$.
- Need gradients of h_y with respect to $\boldsymbol{x}$, $\boldsymbol{W}$, and $\boldsymbol{b}$.
  - To continue Backprop recursively: $(\frac{\partial h_y}{\partial x_1}, \ldots, \frac{\partial h_y}{\partial x_I})^\top$.
  - To be averaged over all examples for gradients of J:

$$\frac{\partial h_y}{\partial w_{i,j}} \text{ and } \frac{\partial h_y}{\partial b_j} \text{ for } i = 1, \ldots, I, \text{ and } j = 1, \ldots, O.$$

Linear Layers III



Since $v_j = \sum_{i=1}^{I} w_{i,j} x_i + b_j$, we compute $\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{x}}, \frac{\partial \boldsymbol{v}}{\partial \boldsymbol{W}}, \frac{\partial \boldsymbol{v}}{\partial \boldsymbol{b}}$ as

$$\frac{\partial v_j}{\partial x_i} = w_{i,j}, \qquad \frac{\partial v_j}{\partial w_{i,j}} = x_i, \qquad \frac{\partial v_j}{\partial b_j} = 1.$$

$$\frac{\partial v_j}{\partial w_{i,j'}} = 0 \text{ and } \frac{\partial v_j}{\partial b_{j'}} = 0 \text{ for } j' \neq j.$$

Linear Layers IV

- Recall $L_y(\boldsymbol{z}) = f_y(\boldsymbol{v}) = h_y(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b})$, the chain rule is

$$\nabla_{\boldsymbol{x},\boldsymbol{W},\boldsymbol{b}} h_y(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}) = (\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{x}}, \frac{\partial \boldsymbol{v}}{\partial \boldsymbol{W}}, \frac{\partial \boldsymbol{v}}{\partial \boldsymbol{b}})^\top \nabla_{\boldsymbol{v}} f_y(\boldsymbol{v}).$$

- Therefore,

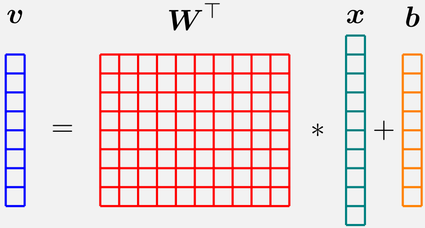
$$\frac{\partial h_y}{\partial x_i} = \sum_{j=1}^{O} \frac{\partial f_y}{\partial v_j} \frac{\partial v_j}{\partial x_i} = \sum_{j=1}^{O} w_{i,j} \frac{\partial f_y}{\partial v_j},$$

$$\frac{\partial h_y}{\partial w_{i,j}} = \sum_{j'=1}^{O} \frac{\partial f_y}{\partial v_{j'}} \frac{\partial v_{j'}}{\partial w_{i,j}} = x_i \frac{\partial f_y}{\partial v_j},$$

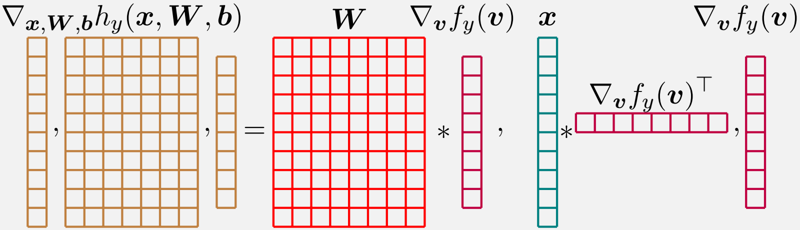
$$\frac{\partial h_y}{\partial b_j} = \sum_{j'=1}^{O} \frac{\partial f_y}{\partial v_{j'}} \frac{\partial v_{j'}}{\partial b_j} = \frac{\partial f_y}{\partial v_j}.$$

Linear Layers V

- Forward propagation



- Backprop



Summary

- The Backprop algorithm applies the chain rule recursively to compute gradients for functions represented as DAGs.