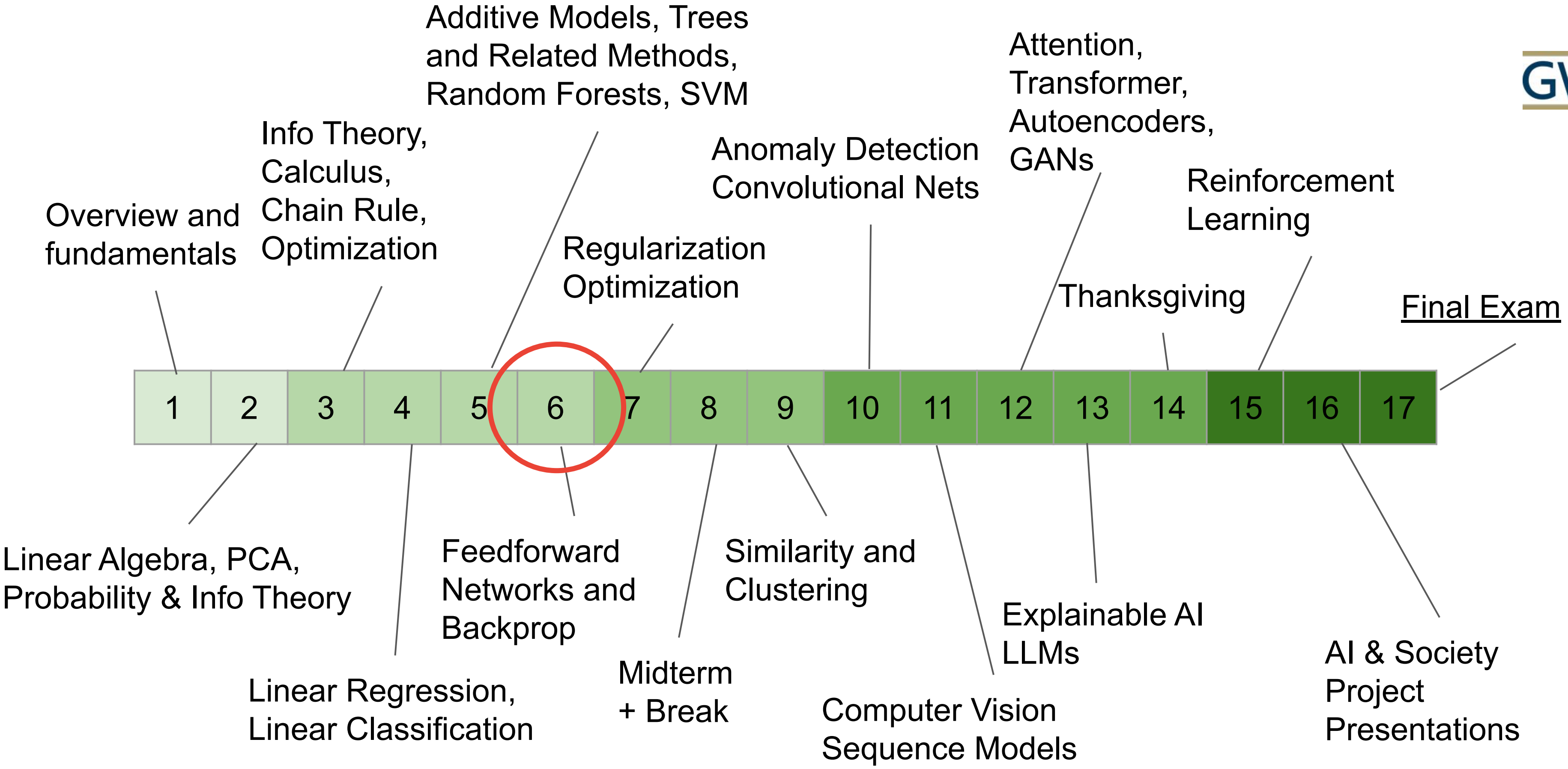
CS 4364/6364

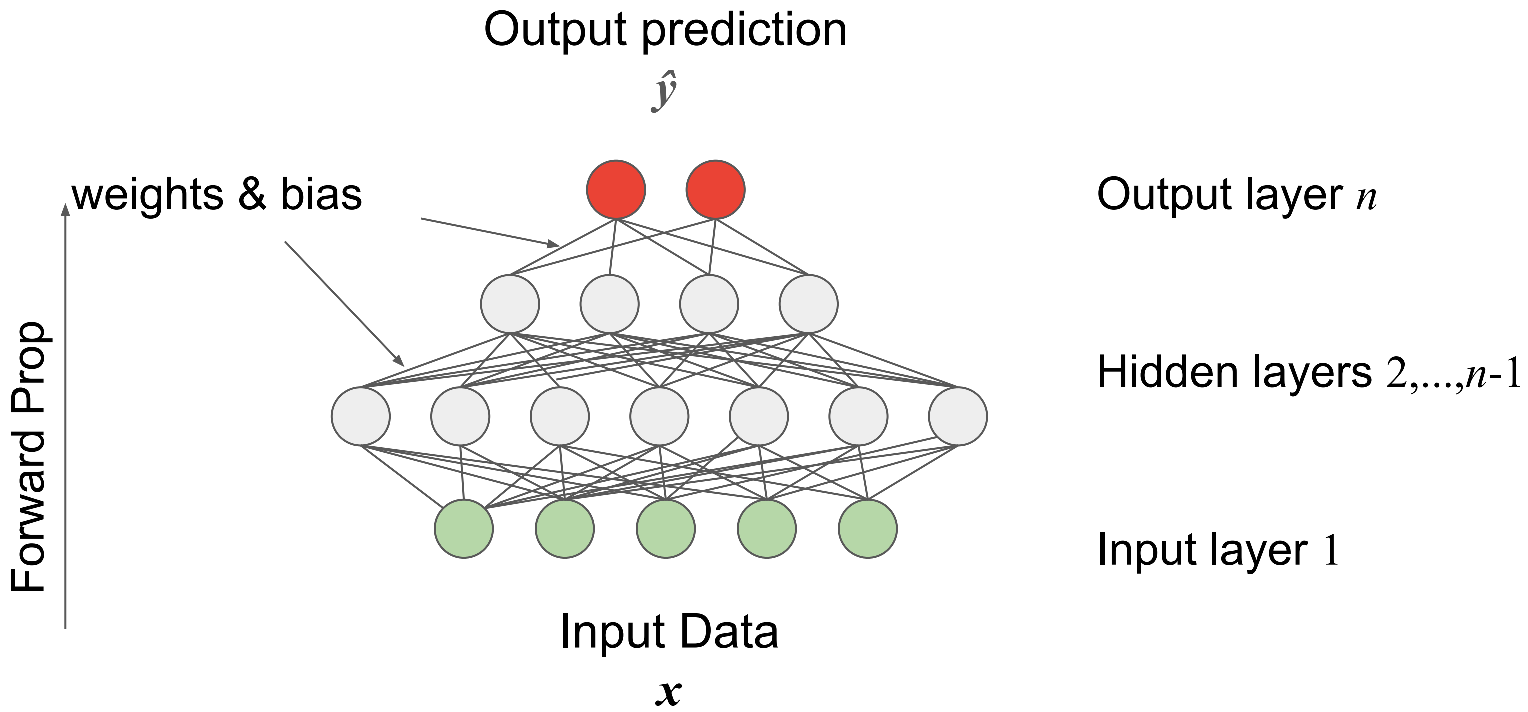
Machine Learning

Fall Semester 9/28/2023
Lecture 11
Back-Propagation 1

John Sipple
jsipple@gwu.edu


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
Overview and fundamentals
Linear Algebra, PCA, Probability & Info Theory
Info Theory, Calculus, Chain Rule, Optimization
Linear Regression, Linear Classification
Additive Models, Trees and Related Methods, Random Forests, SVM
Feedforward Networks and Backprop
Regularization Optimization
Midterm + Break
Similarity and Clustering
Anomaly Detection Convolutional Nets
Computer Vision Sequence Models
Attention, Transformer, Autoencoders, GANs
Explainable AI LLMs
Thanksgiving
Reinforcement Learning
AI & Society Project Presentations
Final Exam

General Architecture of Feedforward Nets



Essential Parts of Training a Neural Net

- Loss function
- Optimization
- Gradient Computation

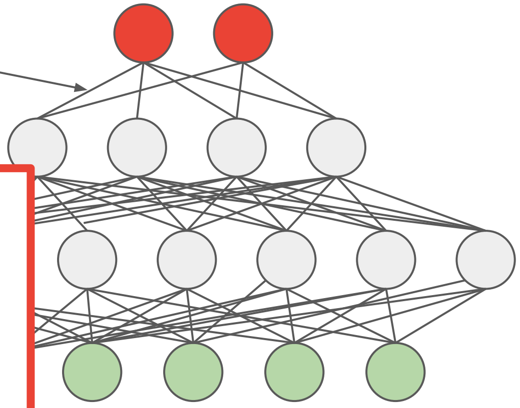
Back-propagation

Outputs of Back-Propagation

Output prediction

$\hat{y}$

weights & bias



Output layer n

Hidden layers $2,\ldots,n$-1

Input layer 1

Back Prop

$$\nabla_{\mathbf{W}^{(i)}} J = \frac{\partial J}{\partial \mathbf{W}^{(i)}}$$

$$\nabla_{\mathbf{b}^{(i)}} J = \frac{\partial J}{\partial \mathbf{b}^{(i)}}$$

$$J = L(\hat{y}, y) + \lambda \Omega(\mathbf{W}^{(i)})$$

Input Data

$\boldsymbol{x}$

The Gradient

General gradient of f with respect to $\boldsymbol{x}$, but ignoring $\boldsymbol{y}$

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}, \boldsymbol{y})$$

Gradient of the loss J with respect to parameters $\boldsymbol{\theta}$:

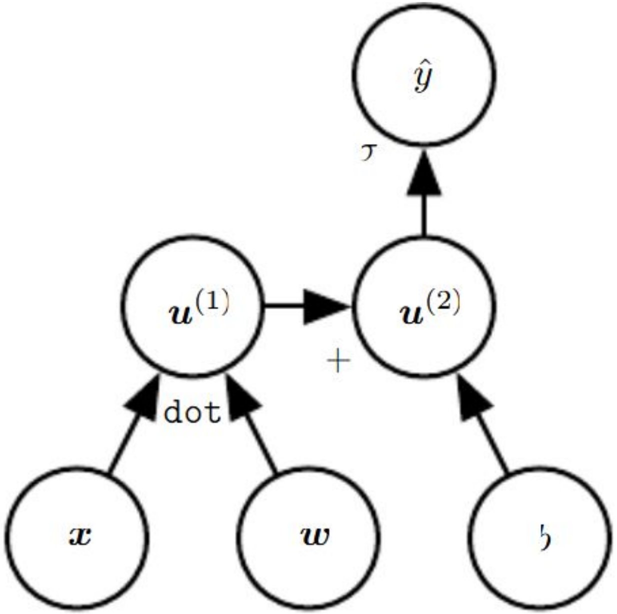
$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

Computational Graphs

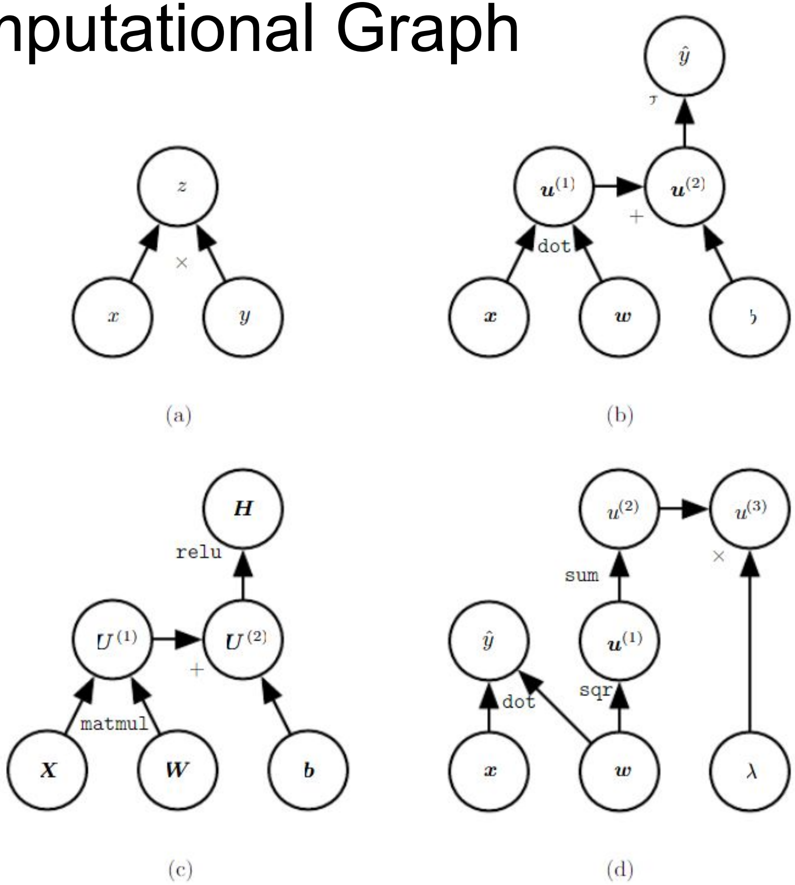
Computational Graph

- Efficient method of decomposing forward/backward propagations
- Graph of nodes and edges, *G(V,E)*
- Variables are nodes
- Operations are edges

$$\hat{y} = \sigma(\boldsymbol{w}^{\top}\boldsymbol{x} + b)$$



Computational Graph



(a) $z = xy$

(b) $\hat{y} = \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b)$

(c) $\boldsymbol{H} = \max\{0, \boldsymbol{XW} + \boldsymbol{b}\}$

(d) Two outputs:

- $\hat{y} = \boldsymbol{x}^\top \boldsymbol{w}$
- $\lambda \sum_i w_i^2$

Chain Rule, Jacobians, and Gradients

Chain Rule of Calculus

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

Knowing the instantaneous rate of change of z relative to y and that of y relative to x allows one to calculate the instantaneous rate of change of z relative to x as the product of the two rates of change.

George F. Simmons: "*if a car travels twice as fast as a bicycle and the bicycle is four times as fast as a walking man, then the car travels 2 × 4 = 8 times as fast as the man.*"

The Jacobian (from before…)

If the function has m inputs and n outputs:

$$\boldsymbol{f} : \mathbb{R}^m \to \mathbb{R}^n$$

The derivative m-by-n matrix is called the Jacobian $\boldsymbol{J} \in \mathbb{R}^{n \times m}$ of $\boldsymbol{f}$:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial}{\partial x_1} f(\boldsymbol{x})_1 & \frac{\partial}{\partial x_2} f(\boldsymbol{x})_1 & \cdots & \frac{\partial}{\partial x_m} f(\boldsymbol{x})_1 \\ \frac{\partial}{\partial x_1} f(\boldsymbol{x})_2 & \frac{\partial}{\partial x_2} f(\boldsymbol{x})_2 & \cdots & \frac{\partial}{\partial x_m} f(\boldsymbol{x})_2 \\ \cdots & & & \\ \frac{\partial}{\partial x_1} f(\boldsymbol{x})_n & \frac{\partial}{\partial x_2} f(\boldsymbol{x})_n & \cdots & \frac{\partial}{\partial x_m} f(\boldsymbol{x})_n \end{bmatrix}$$

Jacobian with matrix input output

Let's take a 2 x 2 matrix $\mathbf{X}$:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}$$

On which an elementwise operation is perfomed in the forward pass: $a_{ij} = \sigma(x_{ij})$

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

Jacobian Example

For the backward pass, we need the Jacobian $\frac{\partial A}{\partial x_{ij}}$:

$$\frac{\partial A}{\partial X} = \begin{bmatrix} \frac{\partial a_{11}}{\partial x_{11}} & \frac{\partial a_{12}}{\partial x_{11}} & \frac{\partial a_{21}}{\partial x_{11}} & \frac{\partial a_{22}}{\partial x_{11}} \\ \frac{\partial a_{11}}{\partial x_{12}} & \frac{\partial a_{12}}{\partial x_{12}} & \frac{\partial a_{21}}{\partial x_{12}} & \frac{\partial a_{22}}{\partial x_{12}} \\ \frac{\partial a_{11}}{\partial x_{21}} & \frac{\partial a_{12}}{\partial x_{21}} & \frac{\partial a_{21}}{\partial x_{21}} & \frac{\partial a_{22}}{\partial x_{21}} \\ \frac{\partial a_{11}}{\partial x_{22}} & \frac{\partial a_{12}}{\partial x_{22}} & \frac{\partial a_{21}}{\partial x_{22}} & \frac{\partial a_{22}}{\partial x_{22}} \end{bmatrix}$$

Jacobian Example

For most operations, the non-diagonal terms reduce to 0.

$$\frac{\partial A}{\partial X} = \begin{bmatrix} \frac{\partial a_{11}}{\partial x_{11}} & 0 & 0 & 0 \\ 0 & \frac{\partial a_{12}}{\partial x_{12}} & 0 & 0 \\ 0 & 0 & \frac{\partial a_{21}}{\partial x_{21}} & 0 \\ 0 & 0 & 0 & \frac{\partial a_{22}}{\partial x_{22}} \end{bmatrix}$$

Hence, the Jacobian can be written as:

$$\frac{\partial A}{\partial X} = \mathrm{diag}(f'(X))$$

where:

$$A = f(X)$$

Jacobian

For the nonlinear activation function:

$$f(\mathbf{X}) = \tanh(\mathbf{X})$$

And its derivative:

$$f'(\mathbf{X}) = 1 - \tanh^2(\mathbf{X})$$

$$\frac{\partial \mathbf{A}}{\partial \mathbf{X}} = \begin{bmatrix} 1 - \tanh^2(x_{11}) & 0 & 0 & 0 \\ 0 & 1 - \tanh^2(x_{12}) & 0 & 0 \\ 0 & 0 & 1 - \tanh^2(x_{21}) & 0 \\ 0 & 0 & 0 & 1 - \tanh^2(x_{22}) \end{bmatrix}$$

Generalizing the chain rule to vectors

Suppose

- $\boldsymbol{x} \in \mathbb{R}^m$ and $\boldsymbol{y} \in \mathbb{R}^n$ and
- $y = g(\boldsymbol{x}) : \mathbb{R}^m \to \mathbb{R}^n$
- $z = f(\boldsymbol{y}) : \mathbb{R}^n \to \mathbb{R}$

Loss function

Then

$$\frac{\partial z}{\partial x_i} = \sum_j^n \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

In vector notation:

$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \right)^{\mathsf{T}} \nabla_{\boldsymbol{y}} z$$

Gradient of loss

$n \times m$ Jacobian of g

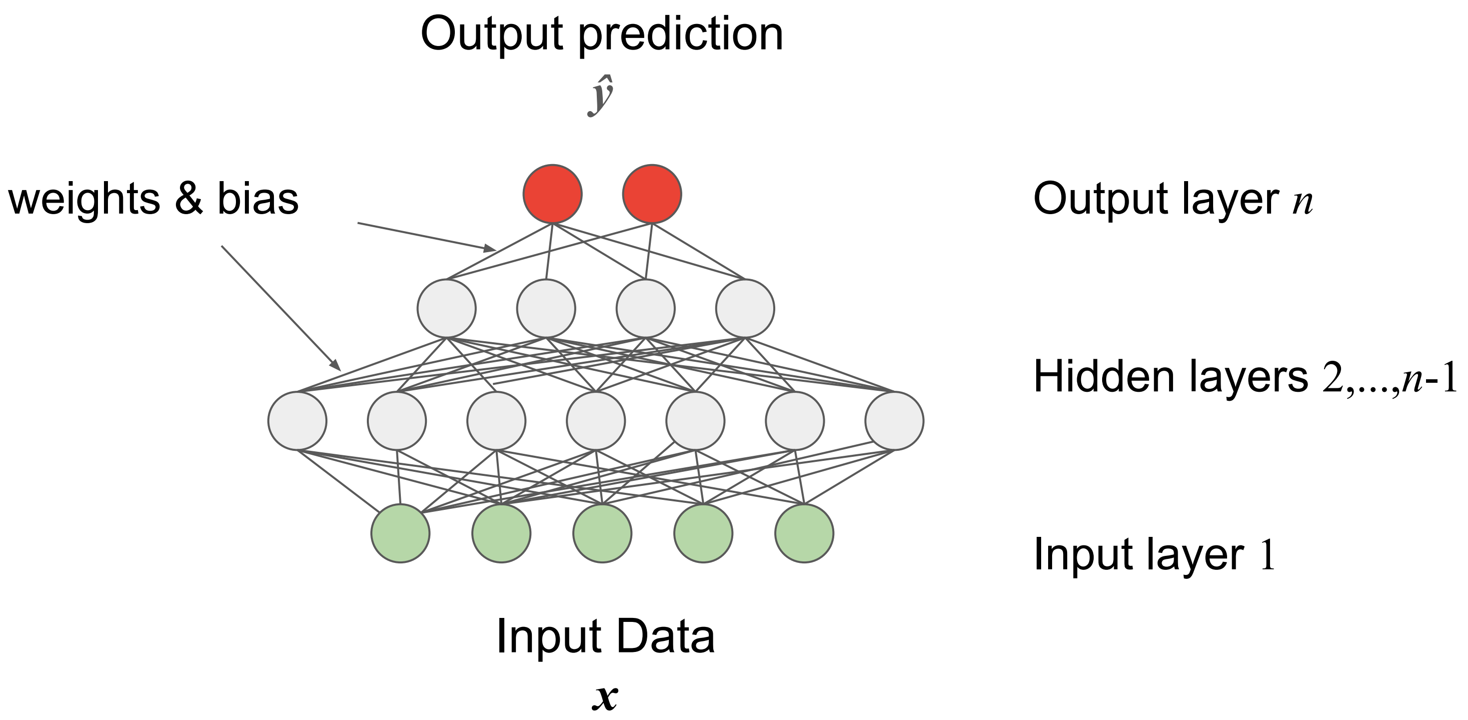
Generalizing the chain rule to tensors

- Conceptually the same idea as vectors, but arbitrary dimensionality: **multiply the Jacobian by the Gradient**.
- Replace scalar indices with tuples e.g., $i = \{(0,0,0), (0,0,1), \ldots\}$
- For all possible index tuples, i, $(\nabla_{\mathsf{X}} z)_i$ gives $\frac{\partial z}{\partial \mathsf{X}_i}$, is equivalent to integer index i, $(\nabla_{\boldsymbol{x}} z)_i$ gives $\frac{\partial z}{\partial x_i}$
- Given $\mathsf{Y} = g(\mathsf{X})$, and $z = f(\mathsf{Y})$, then

$$\nabla_{\mathsf{X}} z = \sum_j \underbrace{(\nabla_{\mathsf{X}} \mathsf{Y}_j)}_{\text{Jacobian}} \underbrace{\frac{\partial z}{\partial \mathsf{Y}_j}}_{\text{Gradient}}$$

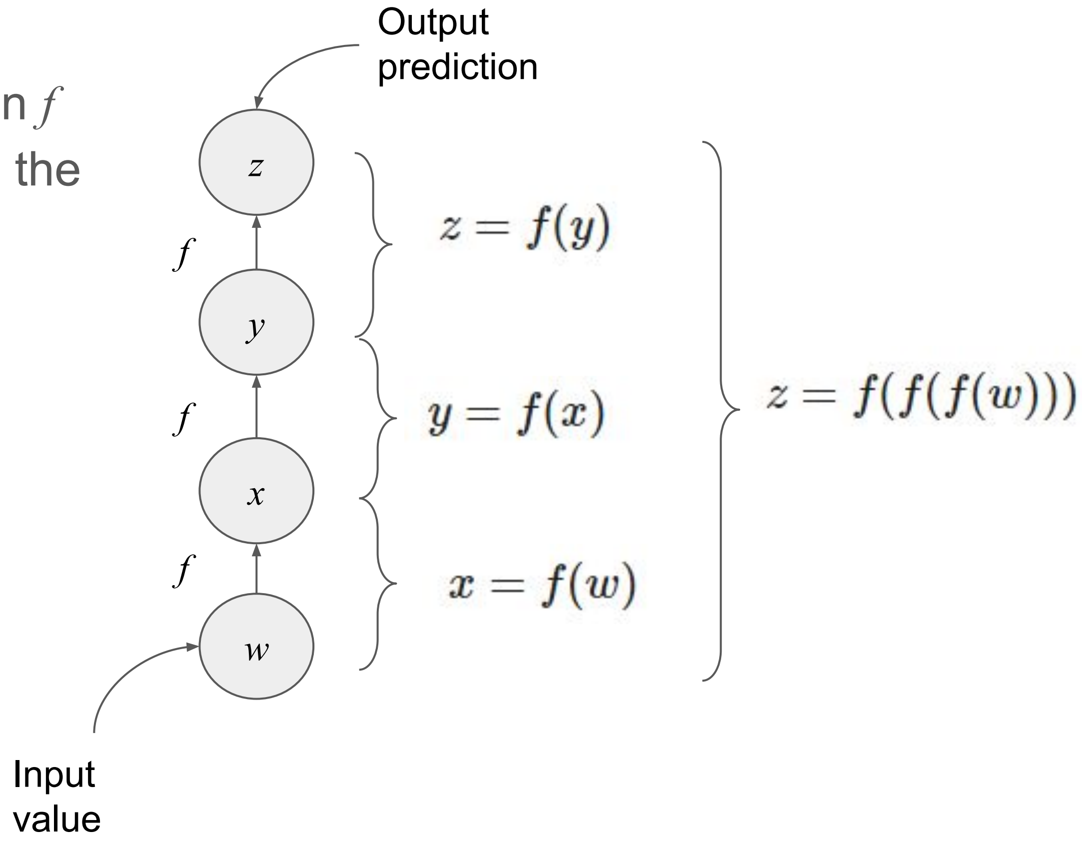
Backprop: Recursive Chain Rule

General Architecture of Feedforward Nets



Recursive Chain Rule

Forward prop: Apply the function f progressively from the input w to the output z

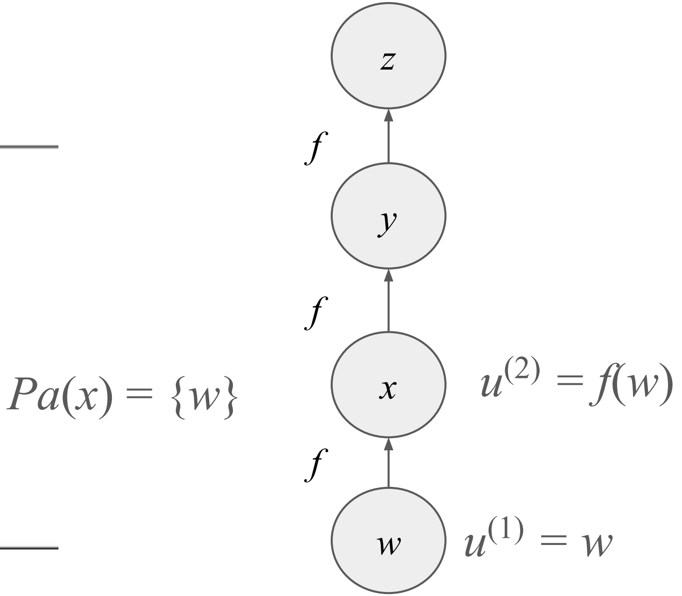


Forward Propagation Algorithm

Algorithm 6.2 Basic Forward-propagation

---

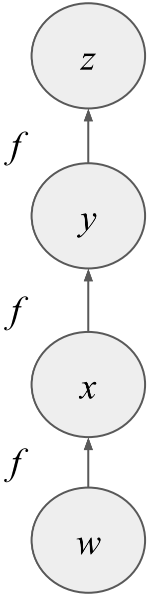
for $i = 1, \ldots, n_i$ **do**
    $u^{(i)} \leftarrow x_i$
end for
for $i = n_i + 1, \ldots, n$ **do**
    $\mathbb{A}^{(i)} \leftarrow \{u^{(j)} \mid j \in Pa(u^{(i)})\}$
    $u^{(i)} \leftarrow f^{(i)}(\mathbb{A}^{(i)})$
end for
return $u^{(n)}$



Recursive Chain Rule

Backprop: Obtain the gradient of the weights and biases by recursively applying chain rule from the Loss function down to the inputs

$$\frac{\partial z}{\partial w} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w} = f'(y)f'(x)f'(w) = f'(f(f(w)))f'(f(w))f'(w)$$



$$\frac{\partial z}{\partial y} = f'(y) = f'(f(f(w)))$$

$$\frac{\partial y}{\partial x} = f'(x) = f'(f(w))$$

$$\frac{\partial x}{\partial w} = f'(w)$$

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

Simplified Backprop Algo

Algorithm 6.2 Simplified Back-propagation

Run forward propagation (algorithm 6.1 for this example) to obtain the activations of the network.

Initialize grad_table, a data structure that will store the derivatives that have been computed. The entry grad_table$[u^{(i)}]$ will store the computed value of $\frac{\partial u^{(n)}}{\partial u^{(i)}}$.

grad_table$[u^{(n)}] \leftarrow 1$

for $j = n - 1$ down to 1 **do**

  The next line computes $\frac{\partial u^{(n)}}{\partial u^{(j)}} = \sum_{i:j \in Pa(u^{(i)})} \frac{\partial u^{(n)}}{\partial u^{(i)}} \frac{\partial u^{(i)}}{\partial u^{(j)}}$ using stored values:

  grad_table$[u^{(j)}] \leftarrow \sum_{i:j \in Pa(u^{(i)})}$ grad_table$[u^{(i)}] \frac{\partial u^{(i)}}{\partial u^{(j)}}$

end for

return $\{$grad_table$[u^{(i)}] \mid i = 1, \ldots, n_i\}$

Improved FP/BP on a fully connected network

Apply the following enhancements to the previous algorithms:

- Add in the Loss function $L(\boldsymbol{\hat{y}}, \boldsymbol{y})$
- Output prediction $\boldsymbol{\hat{y}}$
- Include regularization $\lambda\Omega(\boldsymbol{\theta})$
- Compute weights and biases at each layer $\boldsymbol{W}^{(i)}$, $\boldsymbol{b}^{(i)}$

Regularized Loss function

$J = L(\boldsymbol{\hat{y}}, \boldsymbol{y}) - \lambda\Omega(\boldsymbol{\theta})$

General Architecture of Feedforward Nets

True label

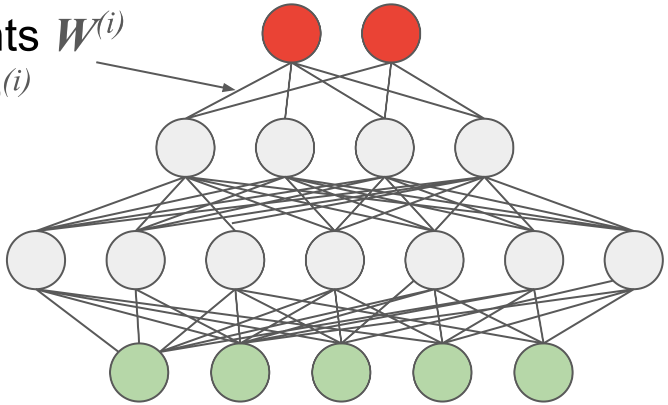
$\boldsymbol{y}$

Output prediction

$\hat{\boldsymbol{y}}$

Weights $\boldsymbol{W}^{(i)}$

Bias $\boldsymbol{b}^{(i)}$



Output layer n

Hidden layers $2,...,n$-1

Input layer 1

Input Data

$\boldsymbol{x}$

Regularized Loss function

$J = L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ - $\lambda\Omega(\boldsymbol{\theta})$

Improved forward propagation

Algorithm 6.3 Improved Forward-propagation

Require: Network depth, l
Require: $\boldsymbol{W}^{(i)}, i \in \{1, \dots, l\}$, the weight matrices of the model
Require: $\boldsymbol{b}^{(i)}, i \in \{1, \dots, l\}$, the bias parameters of the model
Require: $\boldsymbol{x}$, the input to process
Require: $\boldsymbol{y}$, the target output

$\boldsymbol{h}^{(0)} = \boldsymbol{x}$
for $k = 1, \dots, l$ **do**
  $\boldsymbol{a}^{(k)} = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)} \boldsymbol{h}^{(k-1)}$
  $\boldsymbol{h}^{(k)} = f(\boldsymbol{a}^{(k)})$
end for
$\hat{\boldsymbol{y}} = \boldsymbol{h}^{(l)}$
$J = L(\hat{\boldsymbol{y}}, \boldsymbol{y}) + \lambda \Omega(\theta)$

Improved Back-propagation

Algorithm 6.4 Improved Back-propagation

After the forward computation, compute the gradient on the output layer:
$\boldsymbol{g} \leftarrow \nabla_{\hat{\boldsymbol{y}}} J = \nabla_{\hat{\boldsymbol{y}}} L(\hat{\boldsymbol{y}}, \boldsymbol{y})$
for $k = l, l-1, \ldots, 1$ **do**
Convert the gradient on the layer's output into a gradient on the pre-nonlinearity activation (element-wise multiplication if f is element-wise):

$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{a}^{(k)}} J = \boldsymbol{g} \odot f'(\boldsymbol{a}^{(k)})$
Compute gradients on weights and biases (including the regularization term, where needed):
$\nabla_{\boldsymbol{b}^{(k)}} J = \boldsymbol{g} + \lambda \nabla_{\boldsymbol{b}^{(k)}} \Omega(\theta)$
$\nabla_{\boldsymbol{W}^{(k)}} J = \boldsymbol{g}\, \boldsymbol{h}^{(k-1)\top} + \lambda \nabla_{\boldsymbol{W}^{(k)}} \Omega(\theta)$
Propagate the gradients w.r.t. the next lower-level hidden layer's activations:
$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{h}^{(k-1)}} J = \boldsymbol{W}^{(k)\top} \boldsymbol{g}$
end for

Linear Activation:

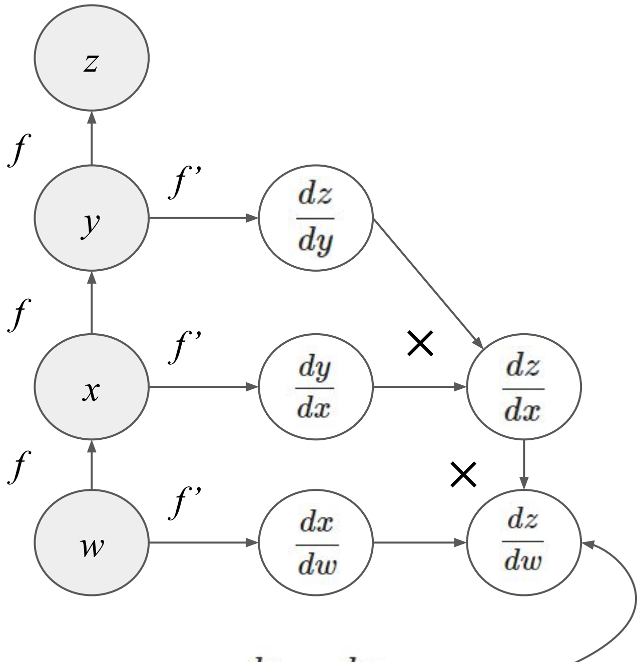
$$\boldsymbol{a}^{(k)} = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)} \boldsymbol{h}^{(k-1)}$$

Layer output with non-linear activation:

$$\boldsymbol{h}^{(k)} = f(\boldsymbol{a}^{(k)})$$

Structuring Backprop with a Computational Graph

- To reduce the runtime complexity from exponential to linear time, expand the computational graph with additional nodes for back propagation
- Note the use of the chain rule
- Graph is populated with values as soon as the parent nodes are available



$$\frac{dz}{dw} = \text{Jacobian} \times \text{Gradient} = \frac{dz}{dx} \times \frac{dx}{dw}$$

Readings

- Goodfellow - Chapter 7
- Goodfellow - Chapter 8