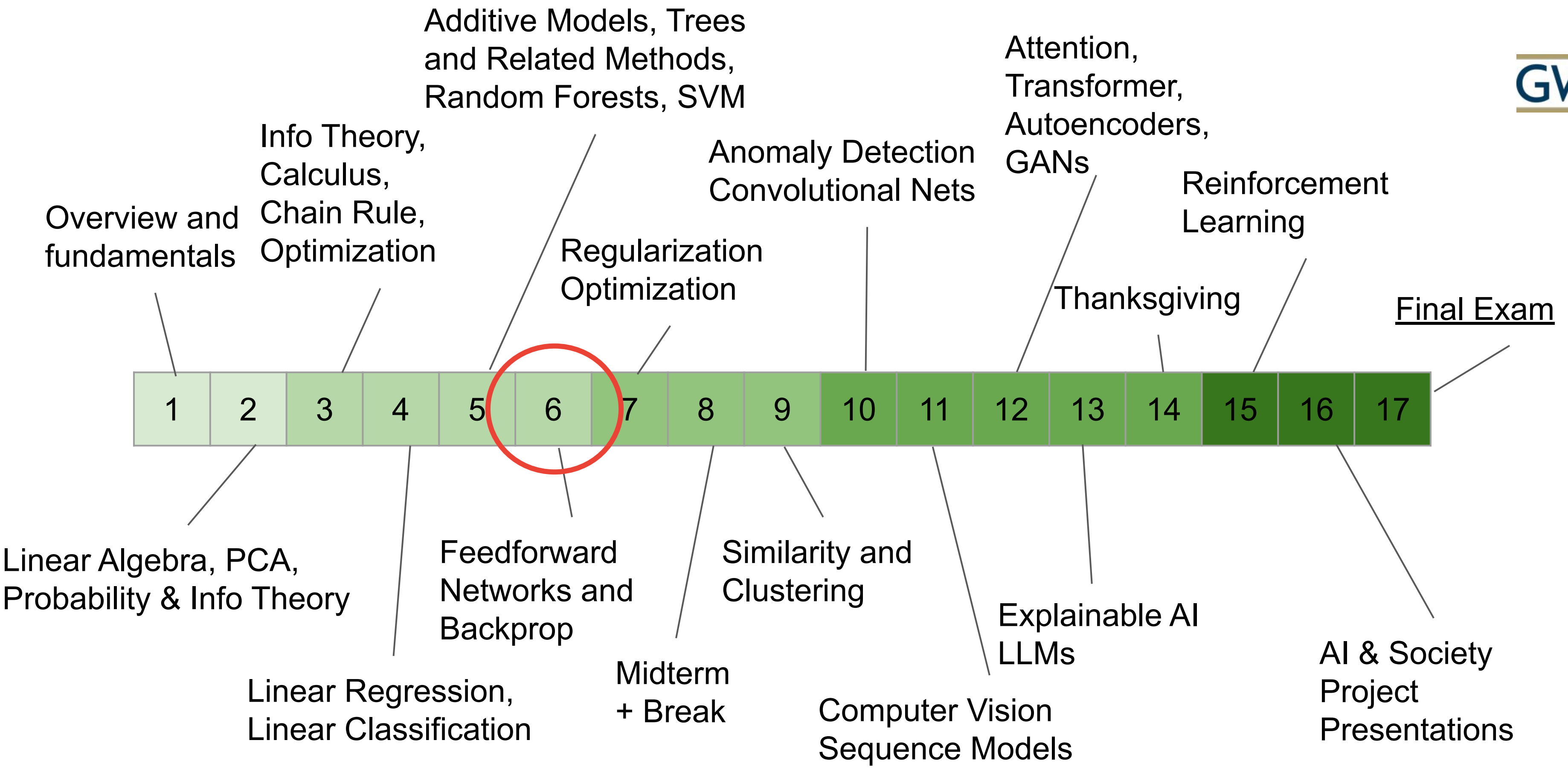
THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

CS 4364/6364

Machine Learning

Fall Semester 10/3/2023
Lecture 12.
Back-Propagation 2

John Sipple
jsipple@gwu.edu


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

“ The back-propagation algorithm is very simple“

General Back-Propagation

- Start at the top of the graph, where $\frac{\partial z}{\partial z} = 1$
- Then compute the gradiant wrt each parent of z by multiplying the current gradient by the Jacobian of the operation that produced z
- Keep passing the gradients down and multiplying by the Jacobians until you reach input x
- If there are more than two paths to a variable, just add them at the node!

$$\sum_i (\nabla_{\mathsf{X}} \mathrm{op.f}(\mathrm{inputs})_i)\, \mathsf{G}_i$$

General Back-Propagation

For each tensor variable **V** in graph G we need the following operations:

- `get_operation(`**V**`)`: a pointer to the operation that computes **V**
- `get_consumers(`**V,** G`)`: children downstream (from the BP perspective) of **V**
- `get_inputs(`**V,** G`)`: parents upstream (from the BP perspective) of **V**

`op`: The operation, **Z**= f (**Y**)

`bprop`: The Jacobian ✖ Gradient $\nabla_{\mathbf{X}} z = \sum_j (\nabla_{\mathbf{X}} \mathbf{Y}_j) \frac{\partial z}{\partial \mathbf{Y}_j}$

Outside skeleton of Back-Propagation

Algorithm 6.5 Outside Skeleton of the back-propagation algorithm

---

Require: $\mathbb{T}$, the target set of variables whose gradients must be computed.
Require: $\mathcal{G}$, the computational graph
Require: z, the variable to be differentiated

Let $\mathcal{G}'$ be $\mathcal{G}$ pruned to contain only nodes that are ancestors of z and descendents of nodes in $\mathbb{T}$.
Initialize `grad_table`, a data structure associating tensors to their gradients
`grad_table`$[z] \leftarrow 1$
for $\mathbf{V}$ in $\mathbb{T}$ **do**
    `build_grad`$(\mathbf{V}, \mathcal{G}, \mathcal{G}',$ `grad_table`$)$
end for
Return `grad_table` restricted to $\mathbb{T}$

Inner Loop of Back-Propagation

Algorithm 6.6 Inner loop subroutine **build_grad**($\mathbf{V}$, G, G', grad_table)

Require: V, the variable whose gradient should be added to $\mathcal{G}$ and **grad_table**
Require: $\mathcal{G}$, the graph to modify
Require: $\mathcal{G}'$, the restriction of $\mathcal{G}$ to nodes that participate in the gradient
Require: grad_table, a data structure mapping nodes to their gradients

```
if V is in grad_table then
    Return grad_table[V]
end if
i ← 1
for C in get_consumers(V, G') do
    op ← get_operation(C)
    D ← build_grad(C, G, G', grad_table)
    G^(i) ← op.bprop(get_inputs(C, G'), V, D)
    i ← i + 1
end for
G ← ∑_i G^(i)
grad_table[V] = G
Insert G and the operations creating it into G
Return G
```

Example of computing gradients for a small network

True label

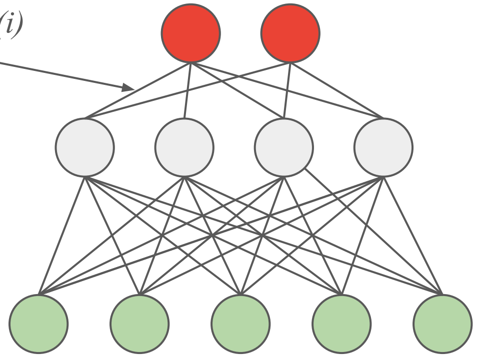
y

Output prediction

$\hat{y}$

Weights $W^{(i)}$

Output layer 2



Hidden layer 1

Input layer 0

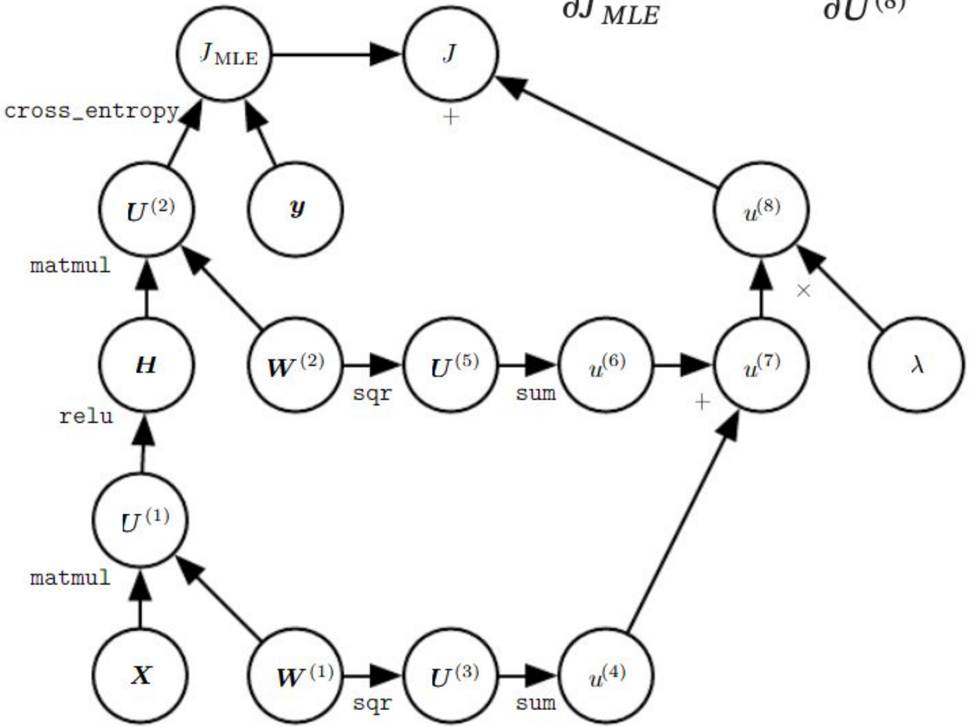
Input Data $X \in \mathbb{R}^{N \times d}$

$$J = J_{\text{MLE}} + \lambda \left(\sum_{i,j} (W_{i,j}^{(1)})^2 + \sum_{i,j} (W_{i,j}^{(2)})^2 \right)$$

Example of computing gradients for a small network

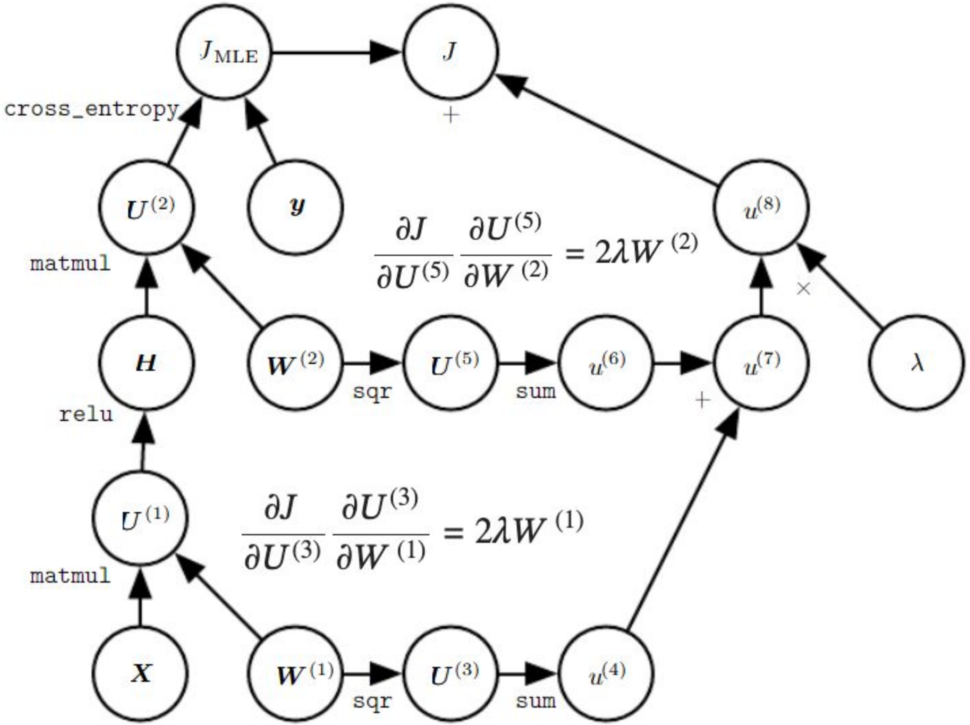
The first step is to calculate the gradients of the objective function with respect to the loss term and the regularization term:

$$\frac{\partial J}{\partial J_{MLE}} = 1 \text{ and } \frac{\partial J}{\partial U^{(8)}} = 1$$



Example of computing gradients for a small network

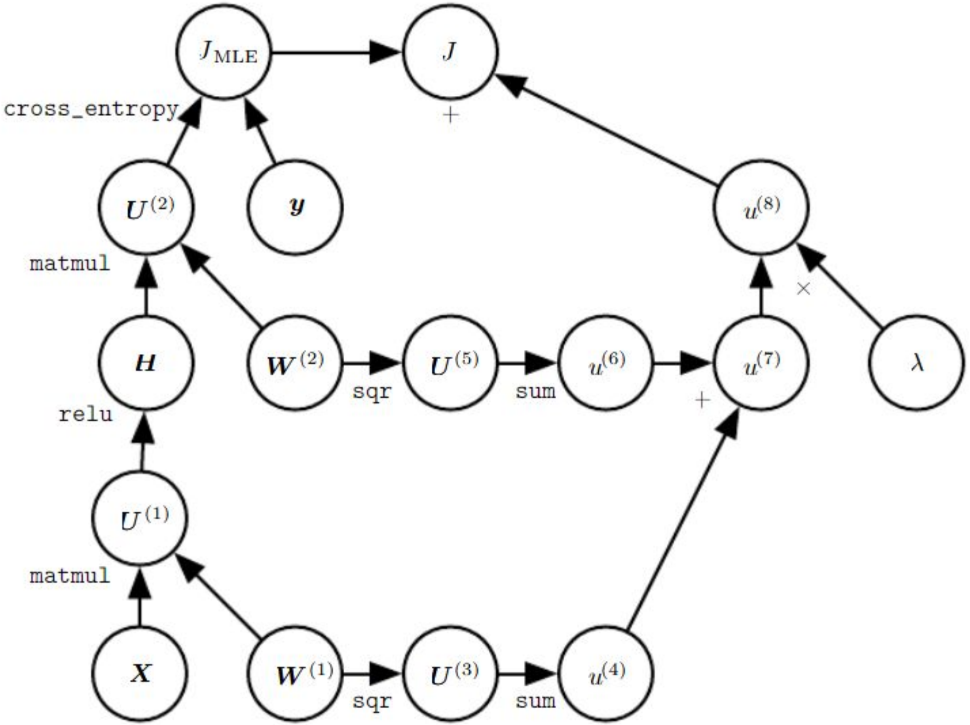
Next, we calculate the gradients of the regularization term with respect to both parameters:



Example of computing gradients for a small network

Next, we compute the gradient of the objective function with respect to variable of the output layer $U^{(2)}$ according to the chain rule:

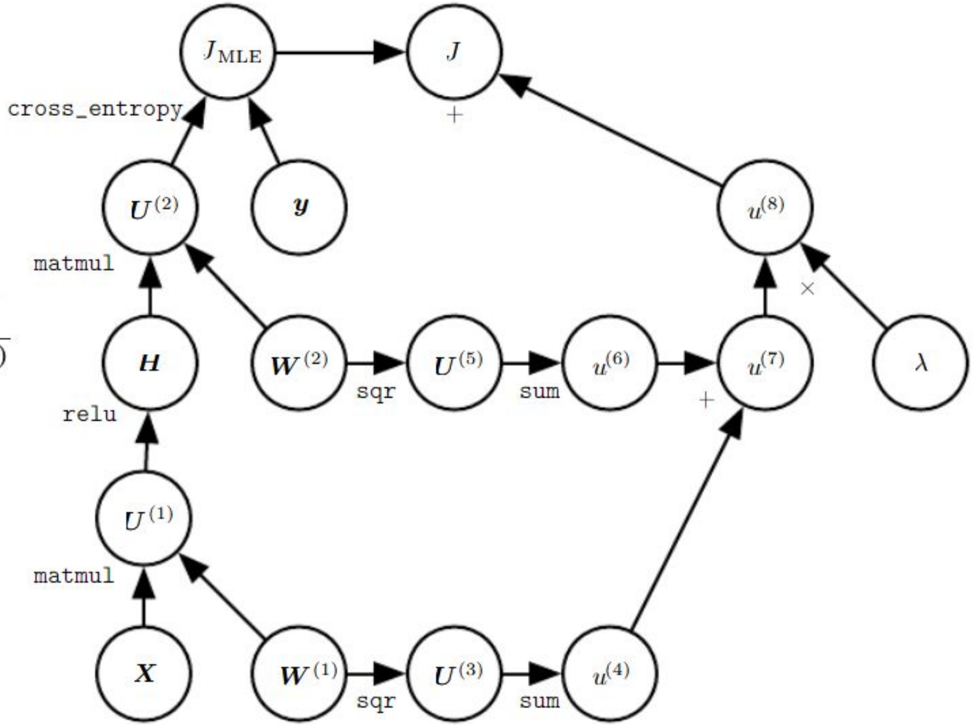
$$G = \frac{\partial J}{\partial U^{(2)}} = \frac{\partial J}{\partial J_{MLE}} \frac{\partial J_{MLE}}{\partial U^{(2)}}$$
$$= \frac{\partial J_{MLE}}{\partial U^{(2)}}$$



Example of computing gradients for a small network

Now we are able to calculate the gradient $\frac{\partial J}{\partial W^{(2)}} \in \mathbb{R}^{q \times h}$ of the model parameters closest to the output layer.

$$\frac{\partial J}{\partial W^{(2)}}$$
$$= \frac{\partial J}{\partial U^{(2)}} \frac{\partial U^{(2)}}{\partial W^{(2)}} + \frac{\partial J}{\partial U^{(5)}} \frac{\partial U^{(5)}}{\partial W^{(2)}}$$
$$= \frac{\partial J}{\partial U^{(2)}} H^{\top} + 2\lambda W^{(2)}$$
$$= G H^{\top} + 2\lambda W^{(2)}$$



Example of computing gradients for a small network

To obtain the gradiant WRT $W^{(1)}$ need to continue backprop along the output layer to the hidden layer. The gradient WRT the hidden layer output $\frac{\partial J}{\partial H} \in \mathbb{R}^h$ is given by:

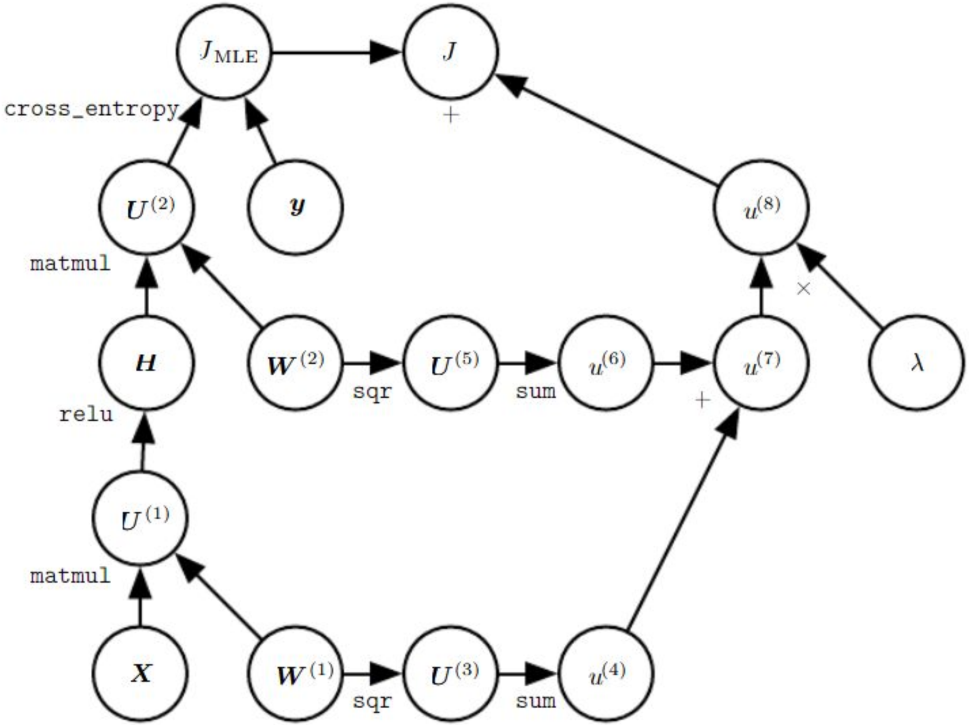
Using the property:

$$\frac{\partial AB}{\partial A} = B^\intercal$$

$$\frac{\partial J}{\partial H} = \frac{\partial J}{\partial U^{(2)}} \frac{\partial U^{(2)}}{\partial H}$$

$$= \frac{\partial J}{\partial U^{(2)}} W^{(2)\intercal}$$

$$= GW^{(2)\intercal}$$



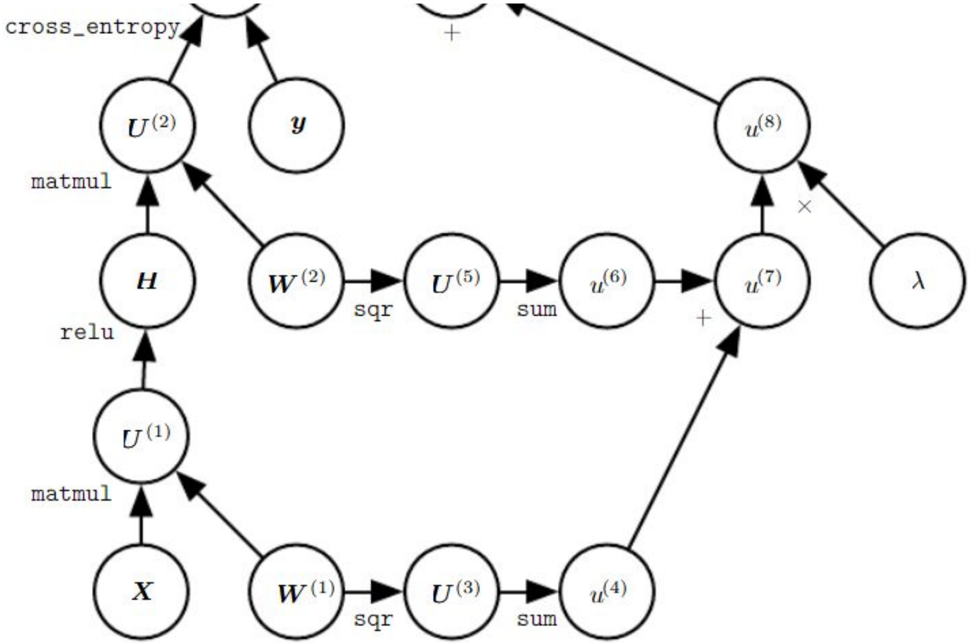
Example of computing gradients for a small network

Since the activation function f = relu applies elementwise, calculating the gradient

$$\frac{\partial J}{\partial U^{(1)}} \in \mathbb{R}^{h}$$

of the intermediate variable $U^{(1)}$ of the intermediate variable requires that we use the elementwise multiplication operator, which we denote by $\odot$:

$$G' = \frac{\partial J}{\partial U^{(1)}} = \frac{\partial J}{\partial H}\frac{\partial H}{\partial U^{(1)}}$$
$$= \frac{\partial J}{\partial H} \odot f'(U^{(1)})$$

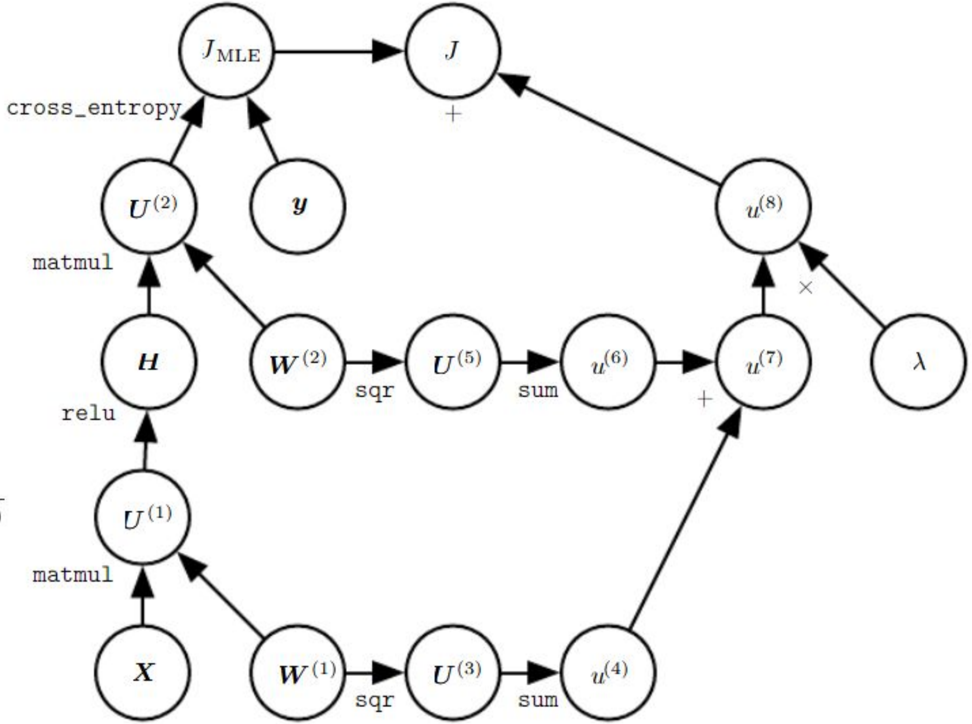


Example of computing gradients for a small network

Finally, we have the gradient $\frac{\partial J}{\partial W^{(1)}} \in \mathbb{R}^{h \times d}$ of the model parameters closest to the input layer. According to the chain rule, we get:

Using the property:

$$\frac{\partial AB}{\partial A} = B^\top$$



$$\begin{aligned}
\frac{\partial J}{\partial W^{(1)}} &= \frac{\partial J}{\partial U^{(1)}} \frac{\partial U^{(1)}}{\partial W^{(1)}} + \frac{\partial J}{\partial U^{(3)}} \frac{\partial U^{(3)}}{\partial W^{(1)}} \\
&= \frac{\partial J}{\partial U^{(1)}} X^\top + 2\lambda W^{(1)} \\
&= G' X^\top + 2\lambda W^{(1)}
\end{aligned}$$

Readings

- Goodfellow - Chapter 7
- Goodfellow - Chapter 8