CSCI 1470/2470
Spring 2023

Ritambhara Singh

February 06, 2023
Monday

Deep Learning

DALL·E 2 prompt “a painting of deep underwater with a yellow submarine in the bottom right corner”
Recap: Optimizer
Recap: Gradient Descent

Basic update rule: $\Delta w_{j,i} = -\alpha \cdot \frac{\partial L}{\partial w_{j,i}}$

• $w_{j,i}$: one network parameter (or “weight”)  
• $\Delta w_{j,i}$: how we change this weight to decrease loss  
• $\alpha$: a constant called the **learning rate**  
• $L$: the loss value
Recap: Our simple regression model

\[ l_j = \sum_k W_{j,k} x_k + b_j \]

\[ l = w \cdot x + b \]

\[ L = (y - l)^2 \]
Recap: Backpropagation

\[
\frac{dL}{dw} = \frac{dL}{dl} \cdot \frac{dl}{dw} = -2(y - 1).x = -2x(y - wx - b) = 2x(wx + b - y)
\]
Today’s goal – continue learning about backpropagation

(1) Building a simple neural network for multi-class classification

(2) Backpropagation of our network (via Chain Rule)

(3) Computation graph for neural networks
Recap: Cross Entropy Loss (for Multi-class classification)

\[-\sum_{j=1}^{m} y_j \log(p_j)\]

\[= -\log(p_a)\]

<table>
<thead>
<tr>
<th>p</th>
<th>Classes (m)</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>&quot;0&quot;</td>
<td>0</td>
</tr>
<tr>
<td>0.2</td>
<td>&quot;1&quot;</td>
<td>0</td>
</tr>
<tr>
<td>0.5</td>
<td>&quot;2&quot;</td>
<td>1</td>
</tr>
</tbody>
</table>

Some examples:

\[\log(0.9) = -0.04\]
\[\log(0.5) = -0.3\]
\[\log(0.001) = -3\]

One hot encoding

We want model to assign high probability to the true class and low to others
A Better Loss: $-\log(p_a)$

Let $p_a$ be the probability.

Operating in “log space” means that near-zero probabilities become large negative numbers, no numerical underflow.
Inverse Probability 1 \(- p_a\) as Loss

- Maximum is 1 (insufficiently strong penalty)
- Gradient at \(p_a=0\) is too flat (doesn’t encourage moving quickly away from wrong answer)

When probabilities get small, floating point numbers often fail to represent differences between them (i.e. numerical ‘underflow’).
Recap: Cross Entropy Loss (for Multi-class classification)

\[- \sum_{j=1}^{m} y_j \log(p_j)\]

\[= -\log(p_\text{true})\]

<table>
<thead>
<tr>
<th>p</th>
<th>Classes (m)</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>“0”</td>
<td>0</td>
</tr>
<tr>
<td>0.2</td>
<td>“1”</td>
<td>0</td>
</tr>
<tr>
<td>0.5</td>
<td>“2”</td>
<td>1</td>
</tr>
</tbody>
</table>

Some examples:

\[\log (0.9) = -0.04\]
\[\log (0.5) = -0.3\]
\[\log (0.001) = -3\]

We want model to assign high probability to the true class and low to others.
Our new probability layer

- What does a probability distribution, \( p \) look like?
  - For any digit \( j \): \( p_j \in [0,1] \)
  - \( \sum_k p_k = 1 \)
- Currently, our outputs \( l \) do not satisfy these properties
  - For any digit \( j \): \( l_j \in \mathbb{R} \)
  - \( \sum_k l_k = \mathbb{R} \)
- How to make our network output satisfy these properties?
The Softmax Function

• The formula:  
  \[ p_j = \frac{e^{l_j}}{\sum_k e^{l_k}} \]

• Using exponents \( e^{l_j} \) means every number is positive

• Dividing by \( \sum_k e^{l_k} \) means every \( p_j \) is between 0 and 1, and that \( \sum_k p_k = 1 \)

We call these numbers **logits** (\( l \) notation)
Recap: Cross Entropy Loss (for Multi-class classification)

\[ -\sum_{j=1}^{m} y_j \log(p_j) \]

We want model to assign high probability to the true class and low to others.

Some examples:

<table>
<thead>
<tr>
<th>( p )</th>
<th>Classes (m)</th>
<th>( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>&quot;0&quot;</td>
<td>0</td>
</tr>
<tr>
<td>0.2</td>
<td>&quot;1&quot;</td>
<td>0</td>
</tr>
<tr>
<td>0.5</td>
<td>&quot;2&quot;</td>
<td>1</td>
</tr>
</tbody>
</table>

We can get these probabilities by using a Softmax function.

We want model to assign high probability to the true class and low to others.
What changes do we make for this task?

\[ l_j = \sum_{i} W_{j,i} x_i + b_j \]

\[ l = w \cdot x + b \]

\[ L = (y - l)^2 \]

- Remove the MSE loss layer
- Increase the number of outputs to m classes
- Add a softmax layer
- Add a cross entropy loss layer
Our model before

- This is a simplified view of our model with an input and a linear layer

\[ l_j = \sum_i w_{j,i} x_i + b_j \]
Our model after

- This is our model with the new probability layer

\[ l_j = \sum_i W_{j,i} x_i + b_j \]

\[ p_j = \frac{e^{l_j}}{\sum_k e^{l_k}} \]

\[ p_j = P(y = j | I) \]
Adding Cross Entropy Loss to Our Network

\[ l_j = \sum_i W_{j,i} x_i + b_j \]

\[ p_j = \frac{e^{l_j}}{\sum_k e^{l_k}} \]

\[ L = -\log(p_a) \]
What is the Chain Rule in Our Network?

- Here’s our function: $L \left( p(l(w)) \right) \Rightarrow$

$$l_j = \sum_i W_{j,i} x_i + b_j$$

$$p_j = \frac{e^{l_j}}{\sum_k e^{l_k}}$$

$$L = - \log(p_a)$$
The Chain Rule in Our Network

Here’s our function: \( L \left( p(l(w)) \right) \Rightarrow \frac{dL}{dw} = \frac{dL}{dp} \cdot \frac{dp}{dl} \cdot \frac{dl}{dw} \)
Derivative for Cross Entropy Loss Layer

\[ l_j = \sum_l W_{j,l} x_l + b_j \]

\[ p_a = \frac{e^{l_a}}{\sum_k e^{l_k}} \]

\[ L = -\log(p_a) \]

\[ \frac{dL}{dp_a} \]
Derivative for Cross Entropy Loss Layer

\[
\frac{dL}{dp_a} = \frac{d}{dp_a} (\log(p_a))
\]
Derivative for Cross Entropy Loss Layer

\[
\frac{\partial L}{\partial p_a} = \frac{\partial (-\log(p_a))}{\partial p_a} = -\frac{1}{p_a}
\]

\[l_j = \sum_i w_{j,i} x_i + b_j\]

\[p_a = \frac{e^{l_a}}{\sum_k e^{l_k}}\]

\[\frac{dL}{dp_j}\]
Chain Rule for Softmax Layer

\[
l_j = \sum_i w_{j,i} x_i + b_j
\]

\[
p_a = \frac{e^{l_a}}{\sum_k e^{l_k}}
\]

\[
L = -\log(p_a)
\]
Chain Rule for Softmax Layer

\[
\frac{\partial p_a}{\partial l_j} =
\]

\[
p_a = \frac{e^{l_a}}{\sum_k e^{l_k}}
\]

\[
L = -\log(p_a)
\]
Chain Rule for Softmax Layer

\[
\frac{\partial p_a}{\partial l_j} = \frac{\partial \left( \frac{e^{l_a}}{\sum_k e^{l_k}} \right)}{\partial l_j} = 
\]

\[
x \xrightarrow{\text{linear layer}} \Sigma \xrightarrow{\text{softmax}} \sigma \xrightarrow{\text{loss}} L = -\log(p_a)
\]
Chain Rule for Softmax Layer

\[
\frac{\partial p_a}{\partial l_j} = \partial \left( \frac{e^{l_a}}{\sum_k e^{l_k}} \right) = ???
\]

Because of multiple inputs and outputs

Which component (output element) of softmax we're seeking to find the derivative of?

With respect to which input element the partial derivative is computed?

https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/
Chain Rule for Softmax Layer

\[
\frac{\partial p_a}{\partial l_j} = \frac{\partial}{\partial l_j} \left( \frac{e^{l_a}}{\sum_k e^{l_k}} \right) = ???
\]

Because of multiple inputs and outputs:
Two cases to consider:
1. \( j = a \) (i.e. the logit of the correct answer)
2. \( j \neq a \)
\[ \frac{\partial p_a}{\partial l_j} = \partial \left( \frac{e^{l_a}}{\sum_k e^{l_k}} \right) \]
\[
\frac{\partial p_a}{\partial l_j} = \partial \left( \frac{e^{l_a}}{\sum_k e^{l_k}} \right) \frac{\partial l_j}{\partial l_j}
\]
\frac{\partial p_a}{\partial l_j} \equiv \partial \left( \frac{e^{l_a}}{\sum_k e^{l_k}} \right) / \partial l_j
\[
\frac{\partial p_a}{\partial l_j} = \partial \left( \frac{e^{l_a}}{\sum_k e^{l_k}} \right)
\]
\frac{\partial p_a}{\partial l_j} = \frac{\partial \left( \frac{e^{l_a}}{\sum_k e^{l_k}} \right)}{\partial l_j}
\[
\frac{\partial p_a}{\partial l_j} = \partial \left( \frac{e^{l_a}}{\sum_k e^{l_k}} \right)
\]
Chain Rule for Softmax Layer

\[
\frac{\partial p_a}{\partial l_j} = \frac{\partial}{\partial l_j} \left( \frac{e^{l_a}}{\sum_k e^{l_k}} \right)
= \begin{cases} 
(1 - p_j)p_a & \text{if } a = j \\
-p_jp_a & \text{if } a \neq j 
\end{cases}
\]

Derivative is positive (increasing the \(a\)th logit will boost the probability of predicting the correct answer)

Derivative is negative (decreasing the probability of every other logit will boost the probability of predicting the correct answer)
Chain Rule for Softmax Layer

\[
\frac{\partial p_a}{\partial l_j} = \begin{cases} 
(1 - p_j)p_a & a = j \\
-p_jp_a & a \neq j 
\end{cases}
\]

A simpler way to write it:

\[
\nabla_l p_a = (y - p)p_a
\]

The vector of all predicted probabilities

\textbf{The gradient} of \( p_a \) with respect to the logit vector \( l \)

A \textbf{one-hot} vector
Chain Rule for Softmax Layer

\[

l_j = \sum_i w_{j,i} x_i + b_j
\]

\[
p_a = \frac{e^{l_a}}{\sum_k e^{l_k}}
\]

\[
L = -\log(p_a)
\]
Chain Rule for Linear Layer

\[
\frac{\partial l_j}{\partial w_{j,i}} =
\]

\[
l_j = \sum_i w_{j,i} x_i + b_j
\]

\[
p_a = \frac{e^{l_a}}{\sum_k e^{l_k}}
\]

\[
L = -\log(p_a)
\]
Chain Rule for Linear Layer

\[
\frac{\partial l_j}{\partial w_{j,i}} = \frac{\partial (\sum_i w_{j,i} x_i)}{\partial w_{j,i}}
\]

\[l_j = \sum_i w_{j,i} x_i + b_j\]

\[p_a = \frac{e^{l_a}}{\sum_k e^{l_k}}\]

\[L = -\log(p_a)\]
Chain Rule for Linear Layer

\[
\frac{\partial l_j}{\partial w_{j,i}} = \frac{\partial (\sum_i w_{j,i} x_i)}{\partial w_{j,i}} = \frac{\partial (\cdots + w_{j,i} x_i + \cdots)}{\partial w_{j,i}}
\]

\[
\frac{dl_j}{dw_i} = \frac{dp_a}{dl_j} = \frac{dL}{dp_a}
\]

\[
l_j = \sum_i w_{j,i} x_i + b_j
\]

\[
p_a = \frac{e^{l_a}}{\sum_k e^{l_k}}
\]

\[
L = -\log(p_a)
\]
Chain Rule for Linear Layer

\[
\frac{\partial l_j}{\partial w_{j,i}} = \frac{\partial (\sum_i w_{j,i} x_i)}{\partial w_{j,i}} = \frac{\partial (\ldots + w_{j,i} x_i + \ldots)}{\partial w_{j,i}} = x_i
\]
Chain Rule Put Together

\[ \Delta w_{j,i} = \]

\[ l_j = \sum_i w_{j,i} x_i + b_j \]

\[ p_a = \frac{e^{l_a}}{\sum_k e^{l_k}} \]

\[ L = -\log(p_a) \]
Chain Rule Put Together

$$\Delta w_{j,i} = -\alpha \frac{\partial L}{\partial w_{j,i}} =$$

\[ l_j = \sum_i w_{j,i} x_i + b_j \]

\[ p_a = \frac{e^{l_a}}{\sum_k e^{l_k}} \]

\[ L = -\log(p_a) \]
Chain Rule Put Together

\[ \Delta_{w_{j,i}} = -\alpha \frac{\partial L}{\partial w_{j,i}} = -\alpha \cdot \frac{\partial L}{\partial p_a} \cdot \frac{\partial p_a}{\partial l_j} \cdot \frac{\partial l_j}{\partial w_{j,i}} = \]

\[ l_j = \sum_i w_{j,i} x_i + b_j \]

\[ p_a = \frac{e^{l_a}}{\Sigma_k e^{l_k}} \]

\[ L = -\log(p_a) \]
Chain Rule Put Together

$$\Delta w_{j,i} = -\alpha \frac{\partial L}{\partial w_{j,i}} = -\alpha \cdot \frac{\partial L}{\partial p_a} \cdot \frac{\partial p_a}{\partial l_j} \cdot \frac{\partial l_j}{\partial w_{j,i}} = -\alpha \cdot \left( \frac{-1}{p_a} \right) \cdot (p_a(y_j - p_j)) \cdot (x_i) =$$
Chain Rule Put Together

\[ \Delta w_{j,i} = -\alpha \frac{\partial L}{\partial w_{j,i}} = -\alpha \cdot \frac{\partial L}{\partial p_a} \cdot \frac{\partial p_a}{\partial l_j} \cdot \frac{\partial l_j}{\partial w_{j,i}} = -\alpha \cdot \left( \frac{-1}{p_a} \right) \cdot \left( p_a(y_j - p_j) \right) \cdot (x_i) = -\alpha \cdot (p_j - y_j) \cdot x_i \]
Gradient Descent: Conclusion

• Update rule: $\Delta w_{j,i} = -\alpha \cdot (p_j - y_j) \cdot x_i = \alpha \cdot (y_j - p_j) \cdot x_i$

• We use this to descend along the gradient toward the minimum loss value

• We used chain rule to propagate backwards through the entire network while doing the derivative - backpropagation

Any questions?
Backpropagation for Deeper Networks

- The function computed by the network is a composition of the functions computed by individual layers (e.g., linear layers and nonlinearities):

  ![Diagram of a neural network with layers](image)

- More precisely:
Computation Graph

- A directed acyclic graph (DAG) that is used to specify mathematical computations:
  - Each edge represents a data dependency (i.e. feed a variable as input to the function)
  - Each node represents a function, or a variable (scalars, vectors, matrices, tensors)
- Recall that neural networks are compositions of functions
- A computation graph can be used to specify a general neural network
Computation Graph

Example Computation Graph for a Neural Network with a Loss Layer:
Chain Rule on a Deeper Neural Network

\[ \frac{\partial e}{\partial w_k} = \frac{\partial e}{\partial h_K} \frac{\partial h_K}{\partial h_{K-1}} \]

- General case:
Chain Rule on a Deeper Neural Network

\[
\frac{\partial e}{\partial w_k} = \frac{\partial e}{\partial h_k} \frac{\partial h_K}{\partial h_{K-1}} \cdots \frac{\partial h_{K+1}}{\partial h_K} \frac{\partial h_k}{\partial w_k}
\]

- General case:
  - Local gradient
  - Upstream gradient

Lazebnik
Backpropagation: Summary

Parameter update:

\[ \frac{\partial e}{\partial w_k} = \frac{\partial e}{\partial h_k} \frac{\partial h_k}{\partial w_k} \]

Local gradient:

\[ f_k(h_{k-1}, w_k) \]

Local gradient:

\[ \frac{\partial h_k}{\partial h_{k-1}} \]

Local gradient:

\[ \frac{\partial h_k}{\partial h_{k-1}} \]

Upstream gradient:

\[ \frac{\partial e}{\partial h_k} \]

Downstream gradient:

\[ \frac{\partial e}{\partial h_{k-1}} = \frac{\partial e}{\partial h_k} \frac{\partial h_k}{\partial h_{k-1}} \]

Forward pass

Backward pass

Lazebnik
Backpropagation: Layer Abstraction

- Layer is an abstraction of a function (linear layer, softmax layer, ReLU layer)

- Forward pass: Just need to implement the function itself $f_k(h_{k-1}, w_k)$

- Backward pass requires two functions to compute local gradients: $\frac{\partial h_k}{\partial h_{k-1}}$ and also $\frac{\partial h_k}{\partial w_k}$ (if the function has parameters)
Recap: Our Weight Matrix

- We have an input vector of size $N$ and an output vector of size $M$, so our weights matrix $W$ is of dimensionality $M \times N$
Dealing with Vectors

\[ \frac{\partial z}{\partial x} = \begin{pmatrix} \frac{\partial z^{(1)}}{\partial x^{(1)}} & \cdots & \frac{\partial z^{(1)}}{\partial x^{(N)}} \\ \vdots & \ddots & \vdots \\ \frac{\partial z^{(M)}}{\partial x^{(1)}} & \cdots & \frac{\partial z^{(M)}}{\partial x^{(N)}} \end{pmatrix} \]

_Jacobian:_ rows correspond to outputs, columns correspond to inputs.

The \( j, i \) th element of the Jacobian is the partial derivative of the \( j \)th output w.r.t. \( i \)th input.
Dealing with Vectors

\[
\frac{\partial e}{\partial x^{(1)}} = \sum_{i=1}^{M} \frac{\partial e}{\partial z^{(i)}} \frac{\partial z^{(i)}}{\partial x^{(1)}}
\]

\[
\frac{\partial e}{\partial x^{(N)}} = \sum_{i=1}^{M} \frac{\partial e}{\partial z^{(i)}} \frac{\partial z^{(i)}}{\partial x^{(N)}}
\]

\[
\frac{\partial z}{\partial x} = \begin{pmatrix}
\frac{\partial z^{(1)}}{\partial x^{(1)}} & \cdots & \frac{\partial z^{(1)}}{\partial x^{(N)}} \\
\vdots & \ddots & \vdots \\
\frac{\partial z^{(M)}}{\partial x^{(1)}} & \cdots & \frac{\partial z^{(M)}}{\partial x^{(N)}}
\end{pmatrix}
\]

Jacobian: rows correspond to outputs, columns correspond to inputs.

The \( j, i \) th element of the Jacobian is the partial derivative of the \( j \)th output w.r.t. \( i \)th input.
Recap

- Cross entropy loss revisited
- Softmax function
- Building a simple model with new layers
- Chain rule for multi-class classification
- Computation graph
- Dealing with vectors

Multi-class classification neural network

Backpropagation and computation