Foundations of Deep Learning, Winter Term 2021/22

Week 3: MLPs and Backpropagation

MLPs and Backpropagation

Frank Hutter Abhinav Valada

University of Freiburg

Overview of Week 3

1. Recap of Multilayer Perceptrons
2. Recap of Chain Rule of Calculus
3. Calculating Gradients with Backpropagation
4. Backprop for a Linear Layer in Matrix Form
5. Full Matrix Form of Backpropagation Equations
6. Further Reading, Summary of the Week, References

Lecture Overview

1. Recap of Multilayer Perceptrons
2. Recap of Chain Rule of Calculus
3. Calculating Gradients with Backpropagation
4. Backprop for a Linear Layer in Matrix Form
5. Full Matrix Form of Backpropagation Equations
6. Further Reading, Summary of the Week, References

Foundations of Deep Learning, Winter Term 2021/22

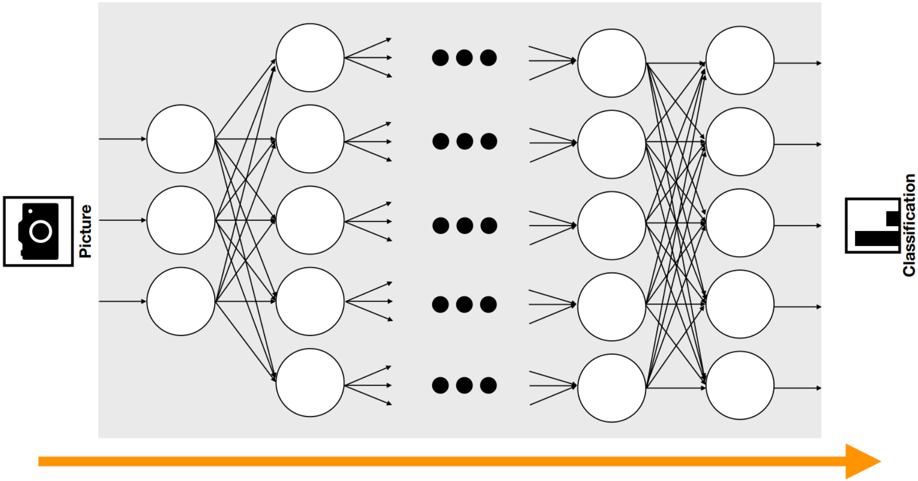
Week 3: MLPs and Backpropagation

Recap of Multilayer Perceptrons

Frank Hutter   Abhinav Valada

University of Freiburg

Computation is Performed Layer-by-Layer



Layer-by-Layer Computations

- Layer 1 pre-activations:
$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)^\mathsf{T}}\mathbf{x} + \mathbf{b}^{(1)}$$
- Layer 1 activations:
$$\mathbf{h}^{(1)} = g^{(1)}(\mathbf{z}^{(1)})$$
- Layer i pre-activations:
$$\mathbf{z}^{(i)} = \mathbf{W}^{(i)^\mathsf{T}}\mathbf{h}^{(i-1)} + \mathbf{b}^{(i)}$$
- Layer i activations:
$$\mathbf{h}^{(i)} = g^{(i)}(\mathbf{z}^{(i)})$$
- Overall network output as one big nested function (network with one hidden layer):
$$\hat{\mathbf{y}} = g^{(2)}(\mathbf{W}^{(2)^\mathsf{T}} g^{(1)}(\mathbf{W}^{(1)^\mathsf{T}}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)})$$

Questions to Answer for Yourself / Discuss with Friends

- Repetition: What are typical examples of activation functions?
- Repetition: What is the purpose of using non-linear activation functions?

Lecture Overview

1. Recap of Multilayer Perceptrons
2. Recap of Chain Rule of Calculus
3. Calculating Gradients with Backpropagation
4. Backprop for a Linear Layer in Matrix Form
5. Full Matrix Form of Backpropagation Equations
6. Further Reading, Summary of the Week, References

Foundations of Deep Learning, Winter Term 2021/22

Week 3: MLPs and Backpropagation

Recap of Chain Rule of Calculus

Frank Hutter Abhinav Valada

University of Freiburg

Chain Rule

The chain rule computes derivatives for compositions of functions by using their individual derivatives and the product of their functions as below.

For two functions $g(x)$ and $f(y) = f(g(x))$, the chain rule states:

$$(f \circ g)'(x) = (f(g(x)))' = f'(g(x)) \cdot g'(x)$$

For $y = g(x)$ and $z = f(g(x)) = f(y)$:

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

Let $y = g(x) = \sin(x)$ and $z = f(\sin(x)) = \ln(y)$. Then:

$$\frac{\partial z}{\partial x} = \frac{\partial \ln(\sin(x))}{\partial \sin(x)}\frac{\partial \sin(x)}{\partial x} = \frac{1}{\sin(x)} \cdot \cos(x)$$

Chain Rule

As a generalization of the scalar case, consider $\mathbf{x} \in \mathbb{R}^m$, $\mathbf{y} \in \mathbb{R}^n$, $g : \mathbb{R}^m \to \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$. If $\mathbf{y} = g(\mathbf{x})$ and $z = f(\mathbf{y})$, then

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

We can write this in a more compact way using vector notation as:

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

where $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ is the $n \times m$ Jacobian matrix of g, and with $\nabla_{\mathbf{y}} z$ being the gradient of z with respect to the vector $\mathbf{y}$. This can also be generalized to tensors, see chapter $6.5.2$ in the book.

Partial Derivative Example

Consider the vector $\mathbf{y} \in \mathbb{R}^n$, a function $f : \mathbb{R}^n \to \mathbb{R}$, and $z = f(\mathbf{y})$, then

$$\nabla_{\mathbf{y}} z = \begin{bmatrix} \frac{\partial z}{\partial y_1} \\ \frac{\partial z}{\partial y_2} \\ \vdots \\ \frac{\partial z}{\partial y_n} \end{bmatrix}$$

is the gradient of z with respect to the vector $\mathbf{y}$.
For $f(\mathbf{y}) = \dfrac{1}{2}\|\mathbf{y}\|_2^2$, the derivative of z with respect to y would be:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Partial Derivative Example 2

Consider the vectors $\mathbf{x} \in \mathbb{R}^m$, $\mathbf{y} \in \mathbb{R}^n$ a function $g : \mathbb{R}^m \to \mathbb{R}^n$ and $\mathbf{y} = g(\mathbf{x})$, then

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_m} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \frac{\partial y_n}{\partial x_2} & \cdots & \frac{\partial y_n}{\partial x_m} \end{bmatrix}$$

is the $n \times m$ Jacobian matrix of g.

Questions to Answer for Yourself / Discuss with Friends

- Application of what you just learned:
  What is the derivative of the ReLU activation function?

- Application of what you just learned:
  What is the derivative of the logistic sigmoid activation function?

Lecture Overview

1. Recap of Multilayer Perceptrons
2. Recap of Chain Rule of Calculus
3. Calculating Gradients with Backpropagation
4. Backprop for a Linear Layer in Matrix Form
5. Full Matrix Form of Backpropagation Equations
6. Further Reading, Summary of the Week, References

Foundations of Deep Learning, Winter Term 2021/22

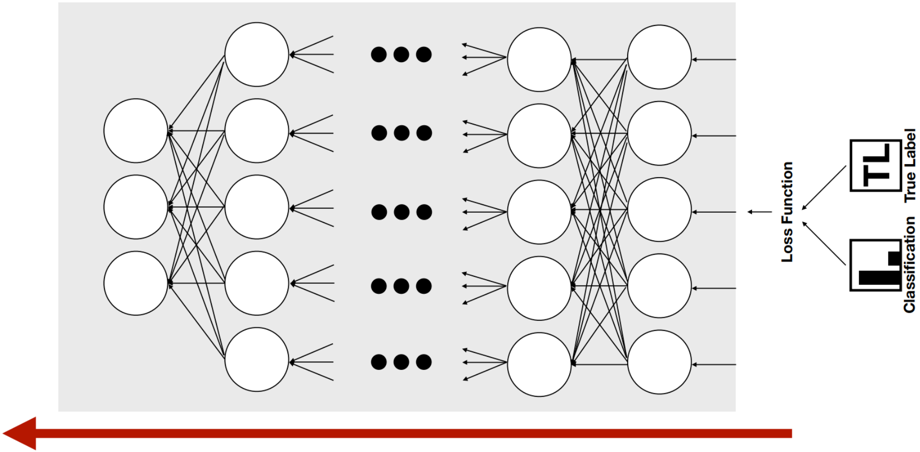
Week 3: MLPs and Backpropagation

Calculating Gradients with Backpropagation

Frank Hutter   Abhinav Valada

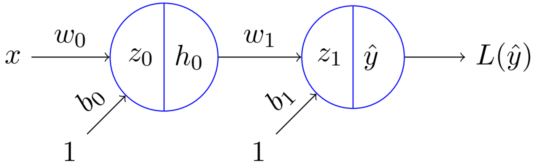
University of Freiburg

Backpropagation: Information Flow Illustration



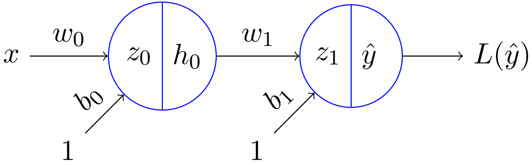
Calculating Partial Derivatives (cont.)

- We will look at how to derive the gradients in the context of a simple example network:



- We would like to know how the change in any of the weights and biases influences the loss, so we calculate $\frac{\partial L}{\partial w}$ and $\frac{\partial L}{\partial b}$ for all weights and biases in the network.

Calculating Partial Derivatives (cont.)



$$
\begin{aligned}
\hat{y} &= g_1(z_1) \\
z_1 &= w_1 h_0 + b_1 \\
h_0 &= g_0(z_0) \\
z_0 &= w_0 x + b_0
\end{aligned}
$$

$$
\begin{aligned}
&\frac{\partial L}{\partial \hat{y}} \\
\frac{\partial L}{\partial z_1} &= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_1} \\
\frac{\partial L}{\partial h_0} &= \frac{\partial L}{\partial z_1} \frac{\partial z_1}{\partial h_0} \\
\frac{\partial L}{\partial z_0} &= \frac{\partial L}{\partial h_0} \frac{\partial h_0}{\partial z_0}
\end{aligned}
$$

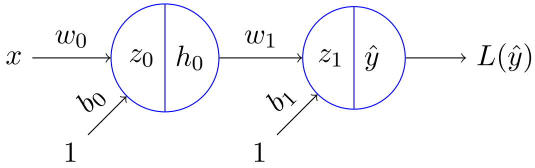
$$
\begin{aligned}
\frac{\partial L}{\partial w_1} &= \frac{\partial L}{\partial z_1} \frac{\partial z_1}{\partial w_1} \\
\frac{\partial L}{\partial b_1} &= \frac{\partial L}{\partial z_1} \frac{\partial z_1}{\partial b_1} \\
\frac{\partial L}{\partial w_0} &= \frac{\partial L}{\partial z_0} \frac{\partial z_0}{\partial w_0} \\
\frac{\partial L}{\partial b_0} &= \frac{\partial L}{\partial z_0} \frac{\partial z_0}{\partial b_0}
\end{aligned}
$$

Calculating Partial Derivatives (cont.)

Derivative of the activation function w.r.t its activation $\frac{\partial h}{\partial z} = h'(z)$ depends on which activation we use:

- linear activation: $h(z) = z \rightarrow h'(z) = 1$
- logistic sigmoid activation: $h(z) = 1/(1 + \exp(-z)) \rightarrow h'(z) = h(z)(1 - h(z))$
- hyperbolic tangent sigmoid activation: $h(z) = \text{tanh}(z) \rightarrow h'(z) = 1 - h(z)^2$
- ReLU activation: $h(z) = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases} \rightarrow h'(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$

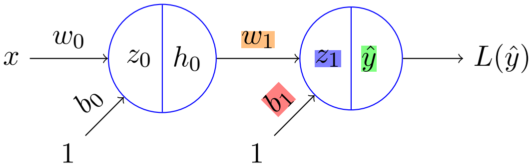
Reconsider 2-Layer MLP as an Example



For each pattern $\mathbf{x}_n$ in training set, perform forward pass:

- hidden layer:
  - $z_0 = xw_0 + b_0$
  - $h_0 = g_0(z_0) = \text{ReLU}(z_0)$
- output layer:
  - $z_1 = h_0 w_1 + b_1$
  - $\hat{y} = g_1(z_1) = z_1$
- g_0 being a ReLU, and g_1 being a linear activation function
- Consider squared error loss: $L = \frac{1}{2}(\hat{y} - y)^2$

Reconsider 2-Layer Example (cont.)



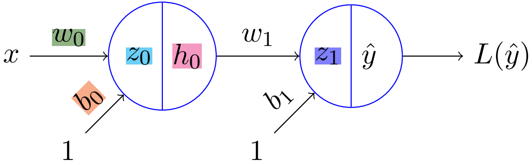
Forward pass:

$$
\begin{aligned}
L &= \frac{1}{2}(\hat{y} - y)^2 \\
\hat{y} &= g_1(z_1) && = z_1 \\
z_1 &= w_1 h_0 + b_1
\end{aligned}
$$

Backward pass:

$$
\begin{aligned}
\frac{\partial L}{\partial \hat{y}} &= \hat{y} - y \\
\frac{\partial L}{\partial z_1} &= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_1} && = (\hat{y} - y) \cdot g_1'(z_1) && = (\hat{y} - y) \cdot 1 \\
\frac{\partial L}{\partial w_1} &= \frac{\partial L}{\partial z_1} \frac{\partial z_1}{\partial w_1} && = \frac{\partial L}{\partial z_1} h_0 \\
\frac{\partial L}{\partial b_1} &= \frac{\partial L}{\partial z_1} \frac{\partial z_1}{\partial b_1} && = \frac{\partial L}{\partial z_1} \cdot 1
\end{aligned}
$$

Reconsider 2-Layer Example (cont.)



Forward pass:

$$h_0 = g_0(z_0) = \begin{cases} z_0 & \text{if } z_0 > 0 \\ 0 & \text{if } z_0 \leq 0 \end{cases}$$

$$z_0 = w_0 x + b_0$$

Backward pass:

$$\frac{\partial L}{\partial h_0} = \frac{\partial L}{\partial z_1} \frac{\partial z_1}{\partial h_0} = \frac{\partial L}{\partial z_1} w_1$$

$$\frac{\partial L}{\partial z_0} = \frac{\partial L}{\partial h_0} \frac{\partial h_0}{\partial z_0} = \begin{cases} \frac{\partial L}{\partial h_0} & \text{if } z_0 > 0 \\ 0 & \text{if } z_0 \leq 0 \end{cases}$$

$$\frac{\partial L}{\partial w_0} = \frac{\partial L}{\partial z_0} \frac{\partial z_0}{\partial w_0} = \frac{\partial L}{\partial z_0} x$$

$$\frac{\partial L}{\partial b_0} = \frac{\partial L}{\partial z_0} \frac{\partial z_0}{\partial b_0} = \frac{\partial L}{\partial z_0} \cdot 1$$

Generic MLP Learning Algorithm Using Backpropagation

- Generic MLP learning algorithm:
  - choose an initial weight vector $\boldsymbol{w}$
  - initialize minimization approach
  - **while** error did not converge **do**
    - **for all** $(\mathbf{x}, y) \in \mathcal{D}$ **do**
      - apply $\mathbf{x}$ to network and calculate the network output (forward pass)
      - calculate $\frac{\partial L_n}{\partial w}$ and $\frac{\partial L_n}{\partial b}$ for all weights and biases (backward pass)
    - **end for**
    - calculate total gradients $\frac{\partial L}{\partial w}$ and $\frac{\partial L}{\partial b}$ for all weights and biases, summing over all training patterns
    - perform one update step of the minimization approach
  - **end while**
- Learning by epoch: All training patterns are considered for each update step of function minimization

Questions to Answer for Yourself / Discuss with Friends

- Repetition: What do the computed gradients represent?
- Repetition: Why do we have to compute the gradients?

Lecture Overview

1. Recap of Multilayer Perceptrons
2. Recap of Chain Rule of Calculus
3. Calculating Gradients with Backpropagation
4. Backprop for a Linear Layer in Matrix Form
5. Full Matrix Form of Backpropagation Equations
6. Further Reading, Summary of the Week, References

Foundations of Deep Learning, Winter Term 2021/22

Week 3: MLPs and Backpropagation

Backprop for a Linear Layer in Matrix Form

Frank Hutter Abhinav Valada

University of Freiburg

Acknowledgment

Example adopted from notes [Johnson, 2017] by Justin Johnson.

Setup and Forward Pass

We will derive the equations for backpropagating through a linear layer using minibatches.

- Input $\mathbf{X}$ $(N \times D)$, weight matrix $\mathbf{W}$ $(D \times M)$, output $\mathbf{Y}$ $(N \times M)$
- Note that data points are stored in the rows of $\mathbf{X}$, just as in PyTorch
- Consider $N = 2, D = 2, M = 3 \rightarrow$ forward pass:

$$\mathbf{X} = \begin{pmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \end{pmatrix} \qquad \mathbf{W} = \begin{pmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \end{pmatrix}$$

$$\begin{aligned} \mathbf{Y} &= \mathbf{XW} \\ &= \begin{pmatrix} x_{1,1}w_{1,1} + x_{1,2}w_{2,1} & x_{1,1}w_{1,2} + x_{1,2}w_{2,2} & x_{1,1}w_{1,3} + x_{2,2}w_{2,3} \\ x_{2,1}w_{1,1} + x_{2,2}w_{2,1} & x_{2,1}w_{1,2} + x_{2,2}w_{2,2} & x_{2,1}w_{1,3} + x_{2,2}w_{2,3} \end{pmatrix} \end{aligned}$$

- Assume derivative of loss L w.r.t. output Y, $\frac{\partial L}{\partial Y}$ $(N \times M)$, has already been computed:

$$\frac{\partial L}{\partial \mathbf{Y}} = \begin{pmatrix} \frac{\partial L}{\partial y_{1,1}} & \frac{\partial L}{\partial y_{1,2}} & \frac{\partial L}{\partial y_{1,3}} \\ \frac{\partial L}{\partial y_{2,1}} & \frac{\partial L}{\partial y_{2,2}} & \frac{\partial L}{\partial y_{2,3}} \end{pmatrix}$$

Setup and Forward Pass

- Goal during backward pass: use $\frac{\partial L}{\partial \mathbf{Y}}$ to compute $\frac{\partial L}{\partial \mathbf{X}}$ $(N \times D)$ and $\frac{\partial L}{\partial \mathbf{W}}$ $(D \times M)$
- Using the chain rule:

$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \frac{\partial \mathbf{Y}}{\partial \mathbf{X}} \qquad \frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \mathbf{Y}} \frac{\partial \mathbf{Y}}{\partial \mathbf{W}}$$

- The terms $\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$ and $\frac{\partial \mathbf{Y}}{\partial \mathbf{W}}$ are called *Jacobian matrices* (they are actually 4d-tensors here). What's their shape, e.g. for $\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$? $N \times M \times N \times D$.
- For a network with $N = 64$ and $M = D = 4096$, how much memory would we need if we want to store $\frac{\partial Y}{\partial X}$ in 32-bit floating point precision?
  $64 \cdot 4096 \cdot 64 \cdot 4096 \cdot 4 = 274,877,906,944$ bytes, which is roughly 256 GB $\rightarrow$ want to avoid having to store this!

One Element at a Time

- We can get around having to form the Jacobians by proceeding one element at a time. Let's work out the case for $\frac{\partial L}{\partial X}$:

$$\frac{\partial L}{\partial \mathbf{X}} = \begin{pmatrix} \frac{\partial L}{\partial x_{1,1}} & \frac{\partial L}{\partial x_{1,2}} \\ \frac{\partial L}{\partial x_{2,1}} & \frac{\partial L}{\partial x_{2,2}} \end{pmatrix}$$

- One element at a time, let's first look at $\frac{\partial L}{\partial x_{1,1}}$. Using the chain rule, we compute it as:

$$\begin{aligned} \frac{\partial L}{\partial x_{1,1}} &= \mathrm{Tr}\left(\left[\frac{\partial L}{\partial \mathbf{Y}}\right]^{\intercal} \frac{\partial \mathbf{Y}}{\partial x_{1,1}}\right) = \frac{\partial L}{\partial y_{1,1}} \frac{\partial y_{1,1}}{\partial x_{1,1}} + \frac{\partial L}{\partial y_{1,2}} \frac{\partial y_{1,2}}{\partial x_{1,1}} + \frac{\partial L}{\partial y_{1,3}} \frac{\partial y_{1,3}}{\partial x_{1,1}} \\ &= \frac{\partial L}{\partial y_{1,1}} w_{1,1} + \frac{\partial L}{\partial y_{1,2}} w_{1,2} + \frac{\partial L}{\partial y_{1,3}} w_{1,3} \end{aligned}$$

One Element at a Time

$$\frac{\partial L}{\partial x_{1,1}} = \frac{\partial L}{\partial y_{1,1}} w_{1,1} + \frac{\partial L}{\partial y_{1,2}} w_{1,2} + \frac{\partial L}{\partial y_{1,3}} w_{1,3}$$
$$= \operatorname{Tr}\left(\begin{pmatrix} \frac{\partial L}{\partial y_{1,1}} & \frac{\partial L}{\partial y_{2,1}} \\ \frac{\partial L}{\partial y_{1,2}} & \frac{\partial L}{\partial y_{2,2}} \\ \frac{\partial L}{\partial y_{1,3}} & \frac{\partial L}{\partial y_{2,3}} \end{pmatrix} \begin{pmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ 0 & 0 & 0 \end{pmatrix} \right)$$

- Doing this for the other elements of $\frac{\partial L}{\partial \mathbf{X}}$, the final matrix is:

$$\frac{\partial L}{\partial \mathbf{X}} = \begin{pmatrix} \frac{\partial L}{\partial y_{1,1}} w_{1,1} + \frac{\partial L}{\partial y_{1,2}} w_{1,2} + \frac{\partial L}{\partial y_{1,3}} w_{1,3} & \frac{\partial L}{\partial y_{1,1}} w_{2,1} + \frac{\partial L}{\partial y_{1,2}} w_{2,2} + \frac{\partial L}{\partial y_{1,3}} w_{2,3} \\ \frac{\partial L}{\partial y_{2,1}} w_{1,1} + \frac{\partial L}{\partial y_{2,2}} w_{1,2} + \frac{\partial L}{\partial y_{2,3}} w_{1,3} & \frac{\partial L}{\partial y_{2,1}} w_{2,1} + \frac{\partial L}{\partial y_{2,2}} w_{2,2} + \frac{\partial L}{\partial y_{2,3}} w_{2,3} \end{pmatrix}$$
$$= \begin{pmatrix} \frac{\partial L}{\partial y_{1,1}} & \frac{\partial L}{\partial y_{1,2}} & \frac{\partial L}{\partial y_{1,3}} \\ \frac{\partial L}{\partial y_{2,1}} & \frac{\partial L}{\partial y_{2,2}} & \frac{\partial L}{\partial y_{2,3}} \end{pmatrix} \begin{pmatrix} w_{1,1} & w_{2,1} \\ w_{1,2} & w_{2,2} \\ w_{1,3} & w_{2,3} \end{pmatrix} = \boxed{\frac{\partial L}{\partial \mathbf{Y}} \mathbf{W}^{\intercal}}$$

One Element at a Time

- This holds for any values of N, D, and M
- Using the same strategy, one component at a time, for $\frac{\partial L}{\partial \mathbf{W}}$, we get:

$$\frac{\partial L}{\partial \mathbf{W}} = \boxed{\mathbf{X}^{\intercal} \frac{\partial L}{\partial \mathbf{Y}}}$$

Lecture Overview

1. Recap of Multilayer Perceptrons
2. Recap of Chain Rule of Calculus
3. Calculating Gradients with Backpropagation
4. Backprop for a Linear Layer in Matrix Form
5. **Full Matrix Form of Backpropagation Equations**
6. Further Reading, Summary of the Week, References

Foundations of Deep Learning, Winter Term 2021/22

Week 3: MLPs and Backpropagation

Full Matrix Form of Backpropagation Equations

Frank Hutter Abhinav Valada

University of Freiburg

2-Layer MLP in Matrix Notation

Consider a two-layer MLP with D inputs, M hidden units, and K output units, where we calculate the forward pass for n data points in parallel, using matrix notation for everything.

Forward pass:

- hidden layer:
  - $\mathbf{Z}_0 = \mathbf{X}\mathbf{W}_0 + \mathbf{B}_0$
  - $\mathbf{H}_0 = g_0(\mathbf{Z}_0)$
- output layer:
  - $\mathbf{Z}_1 = \mathbf{H}_0\mathbf{W}_1 + \mathbf{B}_1$
  - $\hat{\mathbf{Y}} = g_1(\mathbf{Z}_1)$
- g_0 being a ReLU, and g_1 being a linear activation function, both applied element-wise
- dimensions: $\mathbf{X} : n \times D, \mathbf{W}_0 : D \times M, \mathbf{B}_0 : n \times M, \mathbf{Z}_0 : n \times M, \mathbf{H}_0 : n \times M, \mathbf{W}_1 : M \times K, \mathbf{B}_1 : n \times K, \mathbf{Z}_1 : n \times K, \hat{\mathbf{Y}} : n \times K$

Mean squared error loss: $L = \frac{1}{2n} \sum_{i,j} \left(\hat{\mathbf{Y}}_{ij} - \mathbf{Y}_{ij} \right)^2$

2-Layer MLP in Matrix Notation (cont.)

Backward pass starts with calculating the error gradient:

$$\frac{\partial L}{\partial \hat{\mathbf{Y}}} = \frac{1}{n}(\hat{\mathbf{Y}} - \mathbf{Y})$$

Gradients for the output layer:

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{Z}_1} &= \frac{\partial L}{\partial \hat{\mathbf{Y}}} \frac{\partial \hat{\mathbf{Y}}}{\partial \mathbf{Z}_1} = \frac{\partial L}{\partial \hat{\mathbf{Y}}} \odot g_1'(\mathbf{Z}_1) = \frac{1}{n}(\hat{\mathbf{Y}} - \mathbf{Y}) \odot \mathbf{1}_{n\times K} = \frac{1}{n}(\hat{\mathbf{Y}} - \mathbf{Y}) \\
\frac{\partial L}{\partial \mathbf{W}_1} &= \frac{\partial L}{\partial \mathbf{Z}_1} \frac{\partial \mathbf{Z}_1}{\partial \mathbf{W}_1} = \mathbf{H}_0^\intercal \frac{\partial L}{\partial \mathbf{Z}_1} \\
\frac{\partial L}{\partial \mathbf{B}_1} &= \frac{\partial L}{\partial \mathbf{Z}_1} \frac{\partial \mathbf{Z}_1}{\partial \mathbf{B}_1} = \mathbf{1}_{n\times n} \frac{\partial L}{\partial \mathbf{Z}_1}
\end{aligned}$$

To calculate $\frac{\partial L}{\partial \mathbf{W}_1}$ we need the activations $\mathbf{H}_0$ we calculated in the forward pass. The Hadamard product $a \odot b$ denotes element-wise multiplication of two matrices. $\mathbf{1}_{n\times n}$ is a $n \times n$ matrix where every element is equal to 1.

2-Layer MLP in Matrix Notation (cont.)

Gradients for the hidden layer:

$$
\begin{aligned}
\frac{\partial L}{\partial \mathbf{H}_0} &= \frac{\partial L}{\partial \mathbf{Z}_1}\frac{\partial \mathbf{Z}_1}{\partial \mathbf{H}_0} = \frac{\partial L}{\partial \mathbf{Z}_1}\mathbf{W}_1^\intercal \\
\frac{\partial L}{\partial \mathbf{Z}_0} &= \frac{\partial L}{\partial \mathbf{H}_0}\frac{\partial \mathbf{H}_0}{\partial \mathbf{Z}_0} \\
&= \frac{\partial L}{\partial \mathbf{H}_0} \odot g_0'(\mathbf{Z}_0) = \left(z_{ij}'\right) \text{ with } z_{ij}' = \begin{cases} \left(\frac{\partial L}{\partial \mathbf{H}_0}\right)_{ij} & \text{if } z_{ij} > 0 \\ 0 & \text{if } z_{ij} \leq 0 \end{cases} \\
\frac{\partial L}{\partial \mathbf{W}_0} &= \frac{\partial L}{\partial \mathbf{Z}_0}\frac{\partial \mathbf{Z}_0}{\partial \mathbf{W}_0} = X^\intercal \frac{\partial L}{\partial \mathbf{Z}_0} \\
\frac{\partial L}{\partial \mathbf{B}_0} &= \frac{\partial L}{\partial \mathbf{Z}_0}\frac{\partial \mathbf{Z}_0}{\partial \mathbf{B}_0} = \mathbf{1}_{n\times n}\frac{\partial L}{\partial \mathbf{Z}_0}
\end{aligned}
$$

Questions to Answer for Yourself / Discuss with Friends

- Transfer: Can you write the matrix notation for a 2-layer MLP with logistic sigmoid activation functions for both g_0 and g_1?

Lecture Overview

1. Recap of Multilayer Perceptrons
2. Recap of Chain Rule of Calculus
3. Calculating Gradients with Backpropagation
4. Backprop for a Linear Layer in Matrix Form
5. Full Matrix Form of Backpropagation Equations
6. Further Reading, Summary of the Week, References

Foundations of Deep Learning, Winter Term 2021/22

Week 3: MLPs and Backpropagation

Further Reading, Summary of the Week, References

Frank Hutter Abhinav Valada

University of Freiburg

Summary by Learning Goals

Having heard this week's lectures, you can now ...

- explain how to calculate partial derivatives for MLP weights using backpropagation
- use matrix form of backpropagation to calculate gradients for mini-batches

Further Reading

- Resources for this lecture: Deep Learning Book [Goodfellow et al., 2016, chapter 6]
- Stanford Course on Deep Learning [Johnson, 2017]

References

Goodfellow, I., Bengio, Y., Courville, A. (2016)
Deep Learning
MIT Press.
https://www.deeplearningbook.org/

Johnson, J. (2017)
Backpropagation for a Linear Layer
Stanford University CS231n
PDF 2017: cs231n.stanford.edu/handouts/linear-backprop.pdf
Course website: http://cs231n.stanford.edu/
Course notes: https://cs231n.github.io/
Lecture 2: https://cs231n.github.io/optimization-2