Deep Learning

Tutorial 4 – Multi-Layer Perceptrons I

Glenn Angrabeit
Autonomous Vision Group
University of Tübingen

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN







ellis
European Laboratory for Learning and Intelligent Systems

Lecture Quiz

Question 1

A network classifies images into $K = 4$ mutually exclusive classes. Which output layer and loss function would you use for this learning problem?

(A) Softmax output, cross-entropy loss

(B) One sigmoid output per class, binary cross-entropy loss

(C) Linear output, squared error loss

(D) Softmax output, squared error loss

Question 1

A network classifies images into $K = 4$ mutually exclusive classes. Which output layer and loss function would you use for this learning problem?

(A) Softmax output, cross-entropy loss

(B) One sigmoid output per class, binary cross-entropy loss

(C) Linear output, squared error loss

(D) Softmax output, squared error loss

Question 2

When a neural network is trained to predict clean images from noisy versions of the same images with pixel-wise L_2 loss, then the network will learn to predict

(A) The denoised image, correct at every pixel

(B) The expected denoised image at the given noise level

(C) The mean image of the corresponding class label

(D) The noise vector

Question 2

When a neural network is trained to predict clean images from noisy versions of the same images with pixel-wise L_2 loss, then the network will learn to predict

(A) The denoised image, correct at every pixel

(B) The expected denoised image at the given noise level

(C) The mean image of the corresponding class label

(D) The noise vector

Question 3

All weights and biases of a network are initialized to the same constant (for example 0). What happens during training?

(A) All units of a layer compute the same output and receive the same gradient, so they stay identical

(B) The gradients explode

(C) The loss diverges

(D) Nothing special: constant and random initialization behave the same

Question 3

All weights and biases of a network are initialized to the same constant (for example 0). What happens during training?

(A) All units of a layer compute the same output and receive the same gradient, so they stay identical

(B) The gradients explode

(C) The loss diverges

(D) Nothing special: constant and random initialization behave the same

Question 4

A layer has $D_{\text{in}} = 400$ inputs with zero mean and unit variance. With which standard deviation σ should we draw the weights so that the pre-activations $h = \sum_i a_i x_i$ also have unit variance?

(A) $\sigma = 1/20 = 0.05$

(B) $\sigma = 1/400$

(C) $\sigma = 1$

(D) $\sigma = 0.01$, independent of D_{in}

Question 4

A layer has $D_{\text{in}} = 400$ inputs with zero mean and unit variance. With which standard deviation σ should we draw the weights so that the pre-activations $h = \sum_i a_i x_i$ also have unit variance?

(A) $\sigma = 1/20 = 0.05$

(B) $\sigma = 1/400$

(C) $\sigma = 1$

(D) $\sigma = 0.01$, independent of D_{in}

Question 4: derivation

Which $\mathrm{Var}(a_i) = \sigma^2$ gives $\mathrm{Var}(h) = 1$ for $h = \sum_{i=1}^{D_{\text{in}}} a_i x_i$ with $D_{\text{in}} = 400$ if:

Question 4: derivation

Which $\mathrm{Var}(a_i) = \sigma^2$ gives $\mathrm{Var}(h) = 1$ for $h = \sum_{i=1}^{D_{\text{in}}} a_i x_i$ with $D_{\text{in}} = 400$ if:

i) weights are initialized independently of inputs

Question 4: derivation

Which $\mathrm{Var}(a_i) = \sigma^2$ gives $\mathrm{Var}(h) = 1$ for $h = \sum_{i=1}^{D_{\text{in}}} a_i x_i$ with $D_{\text{in}} = 400$ if:

i) weights are initialized independently of inputs

ii) inputs: $\mathbb{E}[x_i] = 0,\ \mathrm{Var}(x_i) = 1 \implies \mathbb{E}[x_i^2] = \mathrm{Var}(x_i) + \mathbb{E}[x_i]^2 = 1$

Question 4: derivation

Which $\mathrm{Var}(a_i) = \sigma^2$ gives $\mathrm{Var}(h) = 1$ for $h = \sum_{i=1}^{D_{\text{in}}} a_i x_i$ with $D_{\text{in}} = 400$ if:

i) weights are initialized independently of inputs

ii) inputs: $\mathbb{E}[x_i] = 0,\ \mathrm{Var}(x_i) = 1 \Rightarrow \mathbb{E}[x_i^2] = \mathrm{Var}(x_i) + \mathbb{E}[x_i]^2 = 1$

iii) weights: i.i.d. with $\mathbb{E}[a_i] = 0 \Rightarrow \mathbb{E}[a_i^2] = \mathrm{Var}(a_i) + \mathbb{E}[a_i]^2 = \sigma^2$

Question 4: derivation

Which $\mathrm{Var}(a_i) = \sigma^2$ gives $\mathrm{Var}(h) = 1$ for $h = \sum_{i=1}^{D_{\text{in}}} a_i x_i$ with $D_{\text{in}} = 400$ if:

i) weights are initialized independently of inputs

ii) inputs: $\mathbb{E}[x_i] = 0,\ \mathrm{Var}(x_i) = 1 \ \Rightarrow\ \mathbb{E}[x_i^2] = \mathrm{Var}(x_i) + \mathbb{E}[x_i]^2 = 1$

iii) weights: i.i.d. with $\mathbb{E}[a_i] = 0 \ \Rightarrow\ \mathbb{E}[a_i^2] = \mathrm{Var}(a_i) + \mathbb{E}[a_i]^2 = \sigma^2$

$$\mathrm{Var}(h) = \mathrm{Var}\Big(\sum\nolimits_i a_i x_i\Big) = \sum\nolimits_i \mathrm{Var}(a_i x_i) + \sum\nolimits_{i \neq j} \mathrm{Cov}(a_i x_i,\ a_j x_j)$$

Question 4: derivation

Which $\mathrm{Var}(a_i) = \sigma^2$ gives $\mathrm{Var}(h) = 1$ for $h = \sum_{i=1}^{D_{\text{in}}} a_i x_i$ with $D_{\text{in}} = 400$ if:

i) weights are initialized independently of inputs

ii) inputs: $\mathbb{E}[x_i] = 0,\ \mathrm{Var}(x_i) = 1 \ \Rightarrow\ \mathbb{E}[x_i^2] = \mathrm{Var}(x_i) + \mathbb{E}[x_i]^2 = 1$

iii) weights: i.i.d. with $\mathbb{E}[a_i] = 0 \ \Rightarrow\ \mathbb{E}[a_i^2] = \mathrm{Var}(a_i) + \mathbb{E}[a_i]^2 = \sigma^2$

$$\mathrm{Var}(h) = \mathrm{Var}\Big(\sum\nolimits_i a_i x_i\Big) = \sum\nolimits_i \mathrm{Var}(a_i x_i) + \sum\nolimits_{i \neq j} \mathrm{Cov}(a_i x_i,\ a_j x_j)$$

$$\mathrm{Cov}(a_i x_i,\ a_j x_j) = \mathbb{E}[a_i a_j x_i x_j] - \mathbb{E}[a_i x_i]\,\mathbb{E}[a_j x_j] = \mathbb{E}[a_i]\,\mathbb{E}[a_j]\,\mathbb{E}[x_i x_j] - 0 = 0$$

Question 4: derivation

Which $\mathrm{Var}(a_i) = \sigma^2$ gives $\mathrm{Var}(h) = 1$ for $h = \sum_{i=1}^{D_{\text{in}}} a_i x_i$ with $D_{\text{in}} = 400$ if:

i) weights are initialized independently of inputs

ii) inputs: $\mathbb{E}[x_i] = 0,\ \mathrm{Var}(x_i) = 1 \ \Rightarrow\ \mathbb{E}[x_i^2] = \mathrm{Var}(x_i) + \mathbb{E}[x_i]^2 = 1$

iii) weights: i.i.d. with $\mathbb{E}[a_i] = 0 \ \Rightarrow\ \mathbb{E}[a_i^2] = \mathrm{Var}(a_i) + \mathbb{E}[a_i]^2 = \sigma^2$

$$\mathrm{Var}(h) = \mathrm{Var}\Big(\sum\nolimits_i a_i x_i\Big) = \sum\nolimits_i \mathrm{Var}(a_i x_i) + \sum\nolimits_{i \neq j} \mathrm{Cov}(a_i x_i,\, a_j x_j)$$

$$\mathrm{Cov}(a_i x_i,\, a_j x_j) = \mathbb{E}[a_i a_j x_i x_j] - \mathbb{E}[a_i x_i]\,\mathbb{E}[a_j x_j] = \mathbb{E}[a_i]\,\mathbb{E}[a_j]\,\mathbb{E}[x_i x_j] - 0 = 0$$

$$\mathrm{Var}(a_i x_i) = \mathbb{E}[a_i^2 x_i^2] - \big(\mathbb{E}[a_i x_i]\big)^2$$

Question 4: derivation

Which $\mathrm{Var}(a_i) = \sigma^2$ gives $\mathrm{Var}(h) = 1$ for $h = \sum_{i=1}^{D_{\text{in}}} a_i x_i$ with $D_{\text{in}} = 400$ if:

i) weights are initialized independently of inputs

ii) inputs: $\mathbb{E}[x_i] = 0,\ \mathrm{Var}(x_i) = 1 \ \Rightarrow\ \mathbb{E}[x_i^2] = \mathrm{Var}(x_i) + \mathbb{E}[x_i]^2 = 1$

iii) weights: i.i.d. with $\mathbb{E}[a_i] = 0 \ \Rightarrow\ \mathbb{E}[a_i^2] = \mathrm{Var}(a_i) + \mathbb{E}[a_i]^2 = \sigma^2$

$$\mathrm{Var}(h) = \mathrm{Var}\Big(\sum\nolimits_i a_i x_i\Big) = \sum\nolimits_i \mathrm{Var}(a_i x_i) + \sum\nolimits_{i \neq j} \mathrm{Cov}(a_i x_i,\, a_j x_j)$$

$$\mathrm{Cov}(a_i x_i,\, a_j x_j) = \mathbb{E}[a_i a_j x_i x_j] - \mathbb{E}[a_i x_i]\,\mathbb{E}[a_j x_j] = \mathbb{E}[a_i]\,\mathbb{E}[a_j]\,\mathbb{E}[x_i x_j] - 0 = 0$$

$$\mathrm{Var}(a_i x_i) = \mathbb{E}[a_i^2 x_i^2] - \big(\mathbb{E}[a_i x_i]\big)^2$$

$$= \mathbb{E}[a_i^2]\,\mathbb{E}[x_i^2] - \big(\mathbb{E}[a_i]\,\mathbb{E}[x_i]\big)^2 = \sigma^2 \cdot 1 - 0$$

Question 4: derivation

Which $\mathrm{Var}(a_i) = \sigma^2$ gives $\mathrm{Var}(h) = 1$ for $h = \sum_{i=1}^{D_{\text{in}}} a_i x_i$ with $D_{\text{in}} = 400$ if:

i) weights are initialized independently of inputs

ii) inputs: $\mathbb{E}[x_i] = 0,\ \mathrm{Var}(x_i) = 1 \ \Rightarrow\ \mathbb{E}[x_i^2] = \mathrm{Var}(x_i) + \mathbb{E}[x_i]^2 = 1$

iii) weights: i.i.d. with $\mathbb{E}[a_i] = 0 \ \Rightarrow\ \mathbb{E}[a_i^2] = \mathrm{Var}(a_i) + \mathbb{E}[a_i]^2 = \sigma^2$

$$\mathrm{Var}(h) = \mathrm{Var}\Big(\sum\nolimits_i a_i x_i\Big) = \sum\nolimits_i \mathrm{Var}(a_i x_i) + \sum\nolimits_{i \neq j} \mathrm{Cov}(a_i x_i,\, a_j x_j)$$

$$\mathrm{Cov}(a_i x_i,\, a_j x_j) = \mathbb{E}[a_i a_j x_i x_j] - \mathbb{E}[a_i x_i]\,\mathbb{E}[a_j x_j] = \mathbb{E}[a_i]\,\mathbb{E}[a_j]\,\mathbb{E}[x_i x_j] - 0 = 0$$

$$\mathrm{Var}(a_i x_i) = \mathbb{E}[a_i^2 x_i^2] - \big(\mathbb{E}[a_i x_i]\big)^2$$

$$= \mathbb{E}[a_i^2]\,\mathbb{E}[x_i^2] - \big(\mathbb{E}[a_i]\,\mathbb{E}[x_i]\big)^2 = \sigma^2 \cdot 1 - 0$$

$$\Rightarrow \quad \mathrm{Var}(h) = D_{\text{in}}\,\sigma^2 \overset{!}{=} 1 \quad \Rightarrow \quad \sigma = 1/\sqrt{D_{\text{in}}} = 1/\sqrt{400} = 0.05$$

Question 5

Why do we use data augmentation?

(A) To train for more gradient steps

(B) To achieve a stable initialization

(C) To reduce the training error

(D) To make the network robust to changes in the input

Question 5

Why do we use data augmentation?

(A) To train for more gradient steps

(B) To achieve a stable initialization

(C) To reduce the training error

(D) To make the network robust to changes in the input

Warm-up Questions

True or False?

Statement	**True**	**False**
Logistic regression is used to solve regression problems.		
The backpropagation algorithm computes the output of all compute nodes in the forward pass and all model parameter updates in the backward pass.		
Maximizing a Bernoulli model distribution equals maximizing the BCE loss.		
The output of the softmax function always sums to 1. **Reminder:** $\mathrm{softmax}(\mathbf{s})_i = \exp(s_i) \,/\, \sum_{k=1}^{C} \exp(s_k),\ i = 1, \ldots, C$		
The universal approximation theorem states that neural networks with 1 layer which may be arbitrarily wide are able to approximate any continuous function in the n-dimensional unit cube arbitrarily well.		

True or False?

Statement	True	False
Logistic regression is used to solve regression problems.		X
The backpropagation algorithm computes the output of all compute nodes in the forward pass and all model parameter updates in the backward pass.		
Maximizing a Bernoulli model distribution equals maximizing the BCE loss.		
The output of the softmax function always sums to 1. **Reminder:** $\mathrm{softmax}(\mathbf{s})_i = \exp(s_i) \,/\, \sum_{k=1}^{C} \exp(s_k),\ i = 1, \ldots, C$		
The universal approximation theorem states that neural networks with 1 layer which may be arbitrarily wide are able to approximate any continuous function in the n-dimensional unit cube arbitrarily well.		

True or False?

Statement	True	False
Logistic regression is used to solve regression problems.		X
The backpropagation algorithm computes the output of all compute nodes in the forward pass and all model parameter updates in the backward pass.	X	
Maximizing a Bernoulli model distribution equals maximizing the BCE loss.		
The output of the softmax function always sums to 1. **Reminder:** $\mathrm{softmax}(\mathbf{s})_i = \exp(s_i) \,/\, \sum_{k=1}^{C} \exp(s_k),\ i = 1, \ldots, C$		
The universal approximation theorem states that neural networks with 1 layer which may be arbitrarily wide are able to approximate any continuous function in the n-dimensional unit cube arbitrarily well.		

True or False?

Statement	True	False
Logistic regression is used to solve regression problems.		X
The backpropagation algorithm computes the output of all compute nodes in the forward pass and all model parameter updates in the backward pass.	X	
Maximizing a Bernoulli model distribution equals maximizing the BCE loss.		X
The output of the softmax function always sums to 1. **Reminder:** $\mathrm{softmax}(\mathbf{s})_i = \exp(s_i) \,/\, \sum_{k=1}^{C} \exp(s_k),\ i = 1, \ldots, C$		
The universal approximation theorem states that neural networks with 1 layer which may be arbitrarily wide are able to approximate any continuous function in the n-dimensional unit cube arbitrarily well.		

True or False?

Statement	True	False
Logistic regression is used to solve regression problems.		X
The backpropagation algorithm computes the output of all compute nodes in the forward pass and all model parameter updates in the backward pass.	X	
Maximizing a Bernoulli model distribution equals maximizing the BCE loss.		X
The output of the softmax function always sums to 1. **Reminder:** $\mathrm{softmax}(\mathbf{s})_i = \exp(s_i) \,/\, \sum_{k=1}^{C} \exp(s_k),\ i = 1, \ldots, C$	X	
The universal approximation theorem states that neural networks with 1 layer which may be arbitrarily wide are able to approximate any continuous function in the n-dimensional unit cube arbitrarily well.		

Why does the softmax always sum to 1?

$$\text{softmax}(\mathbf{s})_i = \frac{\exp(s_i)}{\sum_{k=1}^{C} \exp(s_k)}, \qquad i = 1, \ldots, C$$

$$\Rightarrow \quad \sum_{i=1}^{C} \text{softmax}(\mathbf{s})_i = \sum_{i=1}^{C} \frac{\exp(s_i)}{\sum_{k=1}^{C} \exp(s_k)}$$

$$= \frac{1}{\sum_{k=1}^{C} \exp(s_k)} \cdot \sum_{i=1}^{C} \exp(s_i) = 1$$

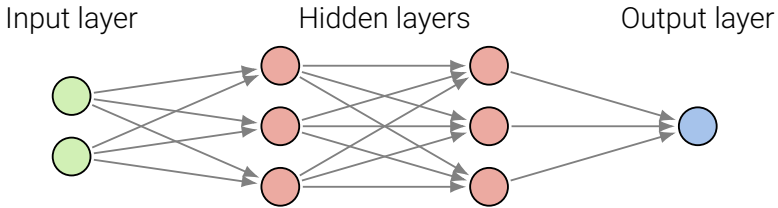
True or False?

Statement	True	False
Logistic regression is used to solve regression problems.		X
The backpropagation algorithm computes the output of all compute nodes in the forward pass and all model parameter updates in the backward pass.	X	
Maximizing a Bernoulli model distribution equals maximizing the BCE loss.		X
The output of the softmax function always sums to 1. **Reminder:** $\mathrm{softmax}(\mathbf{s})_i = \exp(s_i) \,/\, \sum_{k=1}^{C} \exp(s_k),\ i = 1, \ldots, C$	X	
The universal approximation theorem states that neural networks with 1 layer which may be arbitrarily wide are able to approximate any continuous function in the n-dimensional unit cube arbitrarily well.		

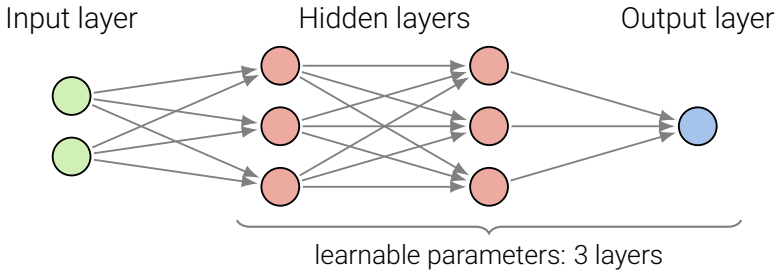
True or False?

Statement	True	False
Logistic regression is used to solve regression problems.		X
The backpropagation algorithm computes the output of all compute nodes in the forward pass and all model parameter updates in the backward pass.	X	
Maximizing a Bernoulli model distribution equals maximizing the BCE loss.		X
The output of the softmax function always sums to 1. **Reminder:** $\mathrm{softmax}(\mathbf{s})_i = \exp(s_i) \,/\, \sum_{k=1}^{C} \exp(s_k),\ i = 1, \ldots, C$	X	
The universal approximation theorem states that neural networks with 1 layer which may be arbitrarily wide are able to approximate any continuous function in the n-dimensional unit cube arbitrarily well.		X

Why is the universal approximation statement false?



Why is the universal approximation statement false?

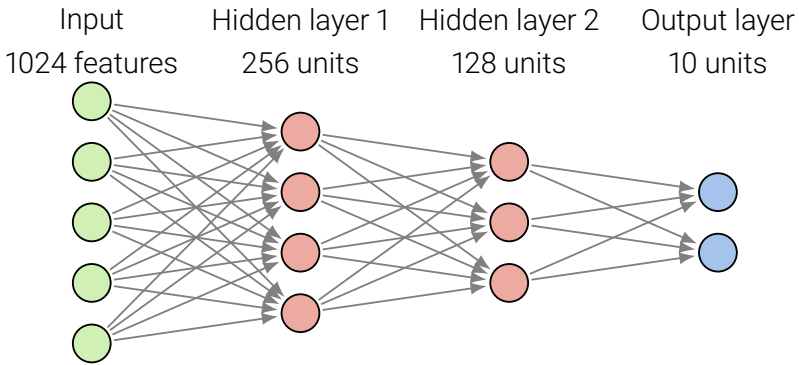


The **depth / number of layers** usually refers to the number of layers with **learnable parameters**: the hidden layers + one output layer.

$\Rightarrow$ With just 1 layer (i.e., no hidden layers), the only learnable part of the MLP is an **affine transformation** $\mathbf{W}\mathbf{x} + \mathbf{b}$.

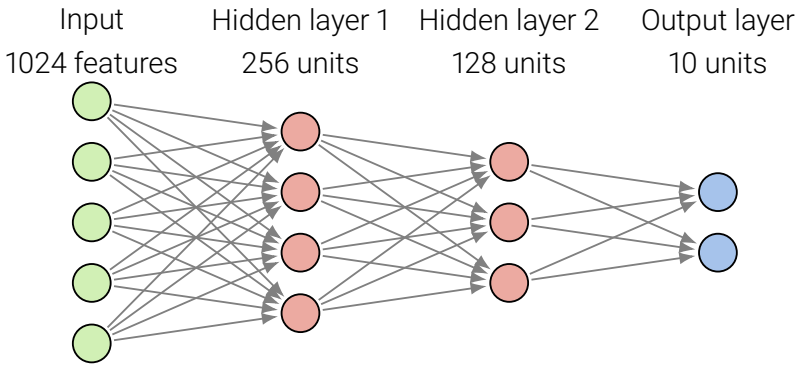
MLPs: Counting parameters

How many learnable parameters are in this MLP (ignoring normalization)?



MLPs: Counting parameters

How many learnable parameters are in this MLP (ignoring normalization)?



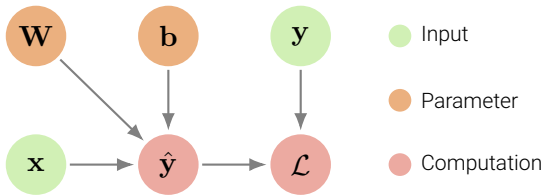
$$(256 \cdot 1024 + 256) + (128 \cdot 256 + 128) + (10 \cdot 128 + 10) = 296{,}586$$

Recap: Backpropagation

Recap: Computation graphs

Consider linear regression with an L_2 loss:

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{x} + \mathbf{b}, \qquad \mathcal{L} = \|\hat{\mathbf{y}} - \mathbf{y}\|^2$$



The gradients we are interested in belong to the **parameter nodes**: $\nabla_{\mathbf{b}}\mathcal{L}$, $\nabla_{\mathbf{W}}\mathcal{L}$

Recap: Gradient descent and the Jacobian

Gradient descent updates each entry using its derivative at the current values:

$$b_i \leftarrow b_i - \eta \, \partial \mathcal{L} / \partial b_i$$

Recap: Gradient descent and the Jacobian

Gradient descent updates each entry using its derivative at the current values:

$$b_i \leftarrow b_i - \eta \, \partial \mathcal{L} / \partial b_i$$

The **gradient** collects these partial derivatives to simplify notation:

$$\nabla_{\mathbf{b}} \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial b_1}, \ldots, \frac{\partial \mathcal{L}}{\partial b_n} \right)^\top \quad \Rightarrow \quad \mathbf{b} \leftarrow \mathbf{b} - \eta \, \nabla_{\mathbf{b}} \mathcal{L}$$

Recap: Gradient descent and the Jacobian

Gradient descent updates each entry using its derivative at the current values:

$$b_i \leftarrow b_i - \eta \, \partial \mathcal{L} / \partial b_i$$

The **gradient** collects these partial derivatives to simplify notation:

$$\nabla_{\mathbf{b}} \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial b_1}, \ldots, \frac{\partial \mathcal{L}}{\partial b_n} \right)^\top \quad \Rightarrow \quad \mathbf{b} \leftarrow \mathbf{b} - \eta \, \nabla_{\mathbf{b}} \mathcal{L}$$

The **Jacobian** of $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ is the matrix of all partial derivatives:

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix} \in \mathbb{R}^{m \times n}$$

Recap: Gradient descent and the Jacobian

Gradient descent updates each entry using its derivative at the current values:

$$b_i \leftarrow b_i - \eta \, \partial \mathcal{L} / \partial b_i$$

The **gradient** collects these partial derivatives to simplify notation:

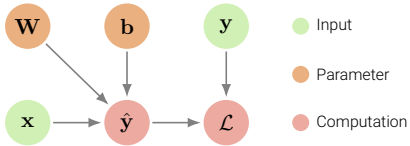
$$\nabla_{\mathbf{b}} \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial b_1}, \ldots, \frac{\partial \mathcal{L}}{\partial b_n} \right)^\top \quad \Rightarrow \quad \mathbf{b} \leftarrow \mathbf{b} - \eta \, \nabla_{\mathbf{b}} \mathcal{L}$$

The **Jacobian** of $\mathbf{f} \colon \mathbb{R}^n \to \mathbb{R}^m$ is the matrix of all partial derivatives:

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix} \in \mathbb{R}^{m \times n}$$

Since $\mathcal{L}$ is a scalar-valued function ($m = 1$), its Jacobian equals the row vector $\left(\frac{\partial \mathcal{L}}{\partial b_1}, \ldots, \frac{\partial \mathcal{L}}{\partial b_n} \right)$ and thus the transpose of the gradient: $\frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \left(\nabla_{\mathbf{b}} \mathcal{L} \right)^\top$

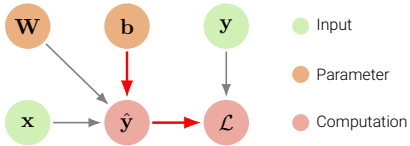
Recap: Backpropagation



Backpropagation provides a mathematical framework to efficiently compute the Jacobian $\partial\mathcal{L}/\partial\mathbf{b}$:

- Trace the path backwards from $\mathcal{L}$ to $\mathbf{b}$ and multiply all neighboring derivatives along the way.
- If there are multiple paths, sum over all of them.

Recap: Backpropagation

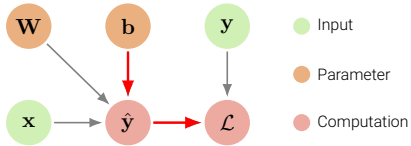


Backpropagation provides a mathematical framework to efficiently compute the Jacobian $\partial\mathcal{L}/\partial\mathbf{b}$:

- Trace the path backwards from $\mathcal{L}$ to $\mathbf{b}$ and multiply all neighboring derivatives along the way.
- If there are multiple paths, sum over all of them.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \cdot \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{b}}$$

Recap: Backpropagation



Backpropagation provides a mathematical framework to efficiently compute the Jacobian $\partial \mathcal{L} / \partial \mathbf{b}$:

- Trace the path backwards from $\mathcal{L}$ to $\mathbf{b}$ and multiply all neighboring derivatives along the way.
- If there are multiple paths, sum over all of them.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \cdot \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{b}}$$

Use this algorithm to find $\partial \mathcal{L} / \partial \mathbf{b}$ and take its transpose to get the gradient $\nabla_{\mathbf{b}} \mathcal{L}$.

Gradient descent for matrices

Analogously to vectors, we want to update each entry of a matrix using its derivative:

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial \mathcal{L}}{\partial w_{ij}}$$

Gradient descent for matrices

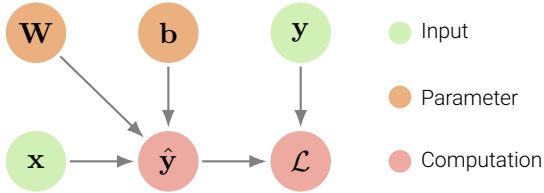
Analogously to vectors, we want to update each entry of a matrix using its derivative:

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial \mathcal{L}}{\partial w_{ij}}$$

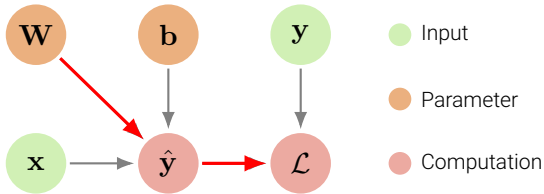
The **gradient** again collects these partial derivatives and preserves the original shape:

$$\nabla_{\mathbf{W}} \mathcal{L} = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial w_{11}} & \cdots & \frac{\partial \mathcal{L}}{\partial w_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{L}}{\partial w_{m1}} & \cdots & \frac{\partial \mathcal{L}}{\partial w_{mn}} \end{pmatrix} \quad \Rightarrow \quad \mathbf{W} \leftarrow \mathbf{W} - \eta \, \nabla_{\mathbf{W}} \mathcal{L}$$

Backpropagation for matrices

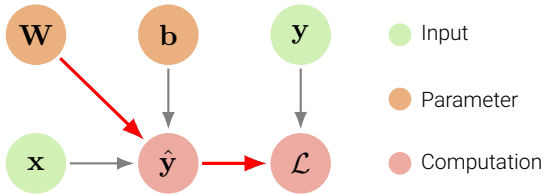


Backpropagation for matrices



$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \cdot \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{W}}$$

Backpropagation for matrices



$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \cdot \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{W}}$$

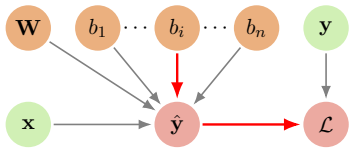
This is not defined for matrices:

$\partial \hat{\mathbf{y}} / \partial \mathbf{W}$ has three indices, so it does not fit into a matrix product.

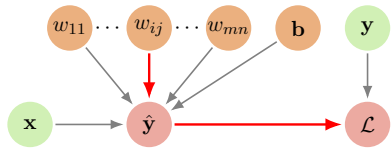
Solution: break down the parameters

As in the loops of lecture 3, take one entry at a time. The chain rule then works.

Bias $\mathbf{b}$



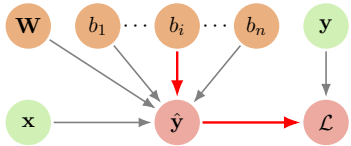
Matrix $\mathbf{W}$



Solution: break down the parameters

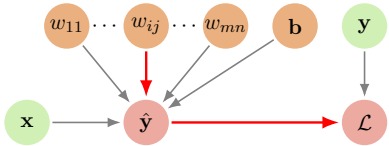
As in the loops of lecture 3, take one entry at a time. The chain rule then works.

Bias $\mathbf{b}$



$$\frac{\partial \mathcal{L}}{\partial b_i} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \cdot \frac{\partial \hat{\mathbf{y}}}{\partial b_i}$$

Matrix $\mathbf{W}$

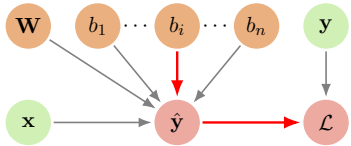


$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \cdot \frac{\partial \hat{\mathbf{y}}}{\partial w_{ij}}$$

Solution: break down the parameters

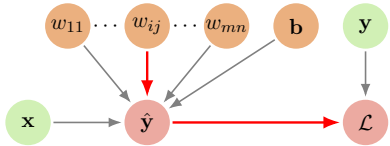
As in the loops of lecture 3, take one entry at a time. The chain rule then works.

Bias $\mathbf{b}$



$$\frac{\partial \mathcal{L}}{\partial b_i} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \cdot \frac{\partial \hat{\mathbf{y}}}{\partial b_i}$$

Matrix $\mathbf{W}$



$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \cdot \frac{\partial \hat{\mathbf{y}}}{\partial w_{ij}}$$

Stack the entries:

$$\nabla_{\mathbf{b}} \mathcal{L} = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial b_1} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial b_n} \end{pmatrix}$$

$$\nabla_{\mathbf{W}} \mathcal{L} = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial w_{11}} & \cdots & \frac{\partial \mathcal{L}}{\partial w_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{L}}{\partial w_{m1}} & \cdots & \frac{\partial \mathcal{L}}{\partial w_{mn}} \end{pmatrix}$$

Backpropagation on MLPs

Old exam question

Consider a simple MLP with one hidden layer:

$$\mathbf{h} = \text{ReLU}(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1), \qquad y = \mathbf{W}_2\mathbf{h} + b_2$$

Compute the value of the output y and the gradient $\nabla_{\mathbf{W}_1} y$ for the following values:

$$\mathbf{W}_1 = \begin{pmatrix} 1 & 2 \\ -1 & 1 \end{pmatrix}, \quad \mathbf{b}_1 = \begin{pmatrix} -1 \\ 0 \end{pmatrix}, \quad \mathbf{W}_2 = \begin{pmatrix} 1 & 2 \end{pmatrix}, \quad b_2 = -1, \quad \mathbf{x} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Reminder: $\text{ReLU}(x) = \max(x, 0)$

Old exam question

Consider a simple MLP with one hidden layer:

$$\mathbf{h} = \text{ReLU}(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1), \qquad y = \mathbf{W}_2\mathbf{h} + b_2$$

Compute the value of the output y and the gradient $\nabla_{\mathbf{W}_1} y$ for the following values:

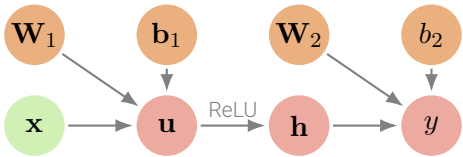
$$\mathbf{W}_1 = \begin{pmatrix} 1 & 2 \\ -1 & 1 \end{pmatrix}, \quad \mathbf{b}_1 = \begin{pmatrix} -1 \\ 0 \end{pmatrix}, \quad \mathbf{W}_2 = \begin{pmatrix} 1 & 2 \end{pmatrix}, \quad b_2 = -1, \quad \mathbf{x} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

1. Draw the computation graph
2. Compute y in the forward pass
3. Write out the equation for $\partial y / \partial [\mathbf{W}_1]_{ij}$ using the chain rule

Reminder: $\text{ReLU}(x) = \max(x, 0)$

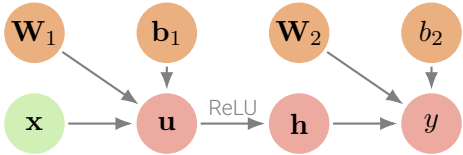
Steps 1–3: computation graph, forward pass and chain rule

(1)



Steps 1–3: computation graph, forward pass and chain rule

(1)



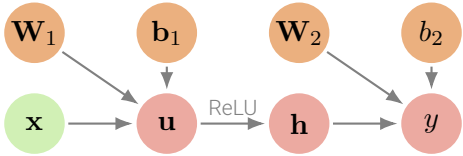
(2)

$$\mathbf{u} = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 = \begin{pmatrix} 1 & 2 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \begin{pmatrix} -1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$$

$$\mathbf{h} = \mathrm{ReLU}(\mathbf{u}) = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \qquad y = \mathbf{W}_2\mathbf{h} + b_2 = \begin{pmatrix} 1 & 2 \end{pmatrix} \begin{pmatrix} 2 \\ 0 \end{pmatrix} - 1 = 1$$

Steps 1–3: computation graph, forward pass and chain rule

(1)



(2)

$$\mathbf{u} = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 = \begin{pmatrix} 1 & 2 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \begin{pmatrix} -1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$$

$$\mathbf{h} = \mathrm{ReLU}(\mathbf{u}) = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \qquad y = \mathbf{W}_2\mathbf{h} + b_2 = \begin{pmatrix} 1 & 2 \end{pmatrix} \begin{pmatrix} 2 \\ 0 \end{pmatrix} - 1 = 1$$

(3)

$$\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} = \frac{\partial y}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial \mathbf{u}} \cdot \frac{\partial \mathbf{u}}{\partial [\mathbf{W}_1]_{ij}}$$

Steps 4–5: your turn

$$\mathbf{W}_1 = \begin{pmatrix} 1 & 2 \\ -1 & 1 \end{pmatrix}, \quad \mathbf{b}_1 = \begin{pmatrix} -1 \\ 0 \end{pmatrix}, \quad \mathbf{W}_2 = \begin{pmatrix} 1 & 2 \end{pmatrix}, \quad b_2 = -1, \quad \mathbf{x} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

(2) $$\mathbf{u} = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \quad \mathbf{h} = \mathrm{ReLU}(\mathbf{u}) = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \quad y = \mathbf{W}_2\mathbf{h} + b_2 = 1$$

(3) $$\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} = \frac{\partial y}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial \mathbf{u}} \cdot \frac{\partial \mathbf{u}}{\partial [\mathbf{W}_1]_{ij}}$$

4. Derive $\partial y / \partial [\mathbf{W}_1]_{ij}$
5. Plug in the values and arrange the gradient

Reminder: $\mathrm{ReLU}'(u) = \begin{cases} 1 & u > 0 \\ 0 & \text{else} \end{cases}$

Step 4: $\partial y / \partial \mathbf{h}$

$$\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} = \frac{\partial y}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial \mathbf{u}} \cdot \frac{\partial \mathbf{u}}{\partial [\mathbf{W}_1]_{ij}}$$

$$\frac{\partial y}{\partial \mathbf{h}} = \frac{\partial}{\partial \mathbf{h}} \left(\mathbf{W}_2 \mathbf{h} + b_2 \right)$$

Step 4: $\partial y / \partial \mathbf{h}$

$$\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} = \frac{\partial y}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial \mathbf{u}} \cdot \frac{\partial \mathbf{u}}{\partial [\mathbf{W}_1]_{ij}}$$

$$\frac{\partial y}{\partial \mathbf{h}} = \frac{\partial}{\partial \mathbf{h}} (\mathbf{W}_2 \mathbf{h} + b_2)$$

Reminder: Jacobian of an affine transformation $\mathbf{f}(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$:

$$\left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right]_{ij} = \frac{\partial f_i}{\partial x_j} = \frac{\partial}{\partial x_j}\Big(\sum_k W_{ik} x_k + b_i\Big) = W_{ij} \quad \Rightarrow \quad \frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \mathbf{W}$$

Step 4: $\partial y / \partial \mathbf{h}$

$$\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} = \frac{\partial y}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial \mathbf{u}} \cdot \frac{\partial \mathbf{u}}{\partial [\mathbf{W}_1]_{ij}}$$

$$\frac{\partial y}{\partial \mathbf{h}} = \frac{\partial}{\partial \mathbf{h}} \left(\mathbf{W}_2 \mathbf{h} + b_2 \right) = \mathbf{W}_2$$

Reminder: Jacobian of an affine transformation $\mathbf{f}(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$:

$$\left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}} \right]_{ij} = \frac{\partial f_i}{\partial x_j} = \frac{\partial}{\partial x_j} \Big(\sum_k W_{ik} x_k + b_i \Big) = W_{ij} \quad \Rightarrow \quad \frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \mathbf{W}$$

Step 4: $\partial \mathbf{h} / \partial \mathbf{u}$

$$\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} = \frac{\partial y}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial \mathbf{u}} \cdot \frac{\partial \mathbf{u}}{\partial [\mathbf{W}_1]_{ij}}$$

$$\frac{\partial \mathbf{h}}{\partial \mathbf{u}} = \frac{\partial}{\partial \mathbf{u}} \text{ReLU}(\mathbf{u})$$

Step 4: $\partial \mathbf{h} / \partial \mathbf{u}$

$$\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} = \frac{\partial y}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial \mathbf{u}} \cdot \frac{\partial \mathbf{u}}{\partial [\mathbf{W}_1]_{ij}}$$

$$\frac{\partial \mathbf{h}}{\partial \mathbf{u}} = \frac{\partial}{\partial \mathbf{u}} \mathrm{ReLU}(\mathbf{u})$$

Reminder: Jacobian of an elementwise activation $\mathbf{f}(\mathbf{x}) = \big(g(x_1), \ldots, g(x_n)\big)^\top$:

$$\left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right]_{ij} = \frac{\partial f_i}{\partial x_j} = \frac{\partial}{\partial x_j} g(x_i) = \begin{cases} g'(x_i) & i = j \\ 0 & i \neq j \end{cases} \quad \Rightarrow \quad \frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \mathrm{diag}\left(g'(x_1), \ldots, g'(x_n)\right)$$

Step 4: $\partial \mathbf{h}/\partial \mathbf{u}$

$$\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} = \frac{\partial y}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial \mathbf{u}} \cdot \frac{\partial \mathbf{u}}{\partial [\mathbf{W}_1]_{ij}}$$

$$\frac{\partial \mathbf{h}}{\partial \mathbf{u}} = \frac{\partial}{\partial \mathbf{u}} \mathsf{ReLU}(\mathbf{u}) = \operatorname{diag}\left(\mathsf{ReLU}'(u_1), \ldots, \mathsf{ReLU}'(u_m)\right)$$

Reminder: Jacobian of an elementwise activation $\mathbf{f}(\mathbf{x}) = \big(g(x_1), \ldots, g(x_n)\big)^\top$:

$$\left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right]_{ij} = \frac{\partial f_i}{\partial x_j} = \frac{\partial}{\partial x_j} g(x_i) = \begin{cases} g'(x_i) & i = j \\ 0 & i \neq j \end{cases} \quad \Rightarrow \quad \frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \operatorname{diag}\left(g'(x_1), \ldots, g'(x_n)\right)$$

Step 4: $\partial \mathbf{u} / \partial [\mathbf{W}_1]_{ij}$

$$\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} = \frac{\partial y}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial \mathbf{u}} \cdot \frac{\partial \mathbf{u}}{\partial [\mathbf{W}_1]_{ij}}$$

$$\frac{\partial \mathbf{u}}{\partial [\mathbf{W}_1]_{ij}} = \frac{\partial}{\partial [\mathbf{W}_1]_{ij}} (\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

Step 4: $\partial \mathbf{u} / \partial [\mathbf{W}_1]_{ij}$

$$\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} = \frac{\partial y}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial \mathbf{u}} \cdot \frac{\partial \mathbf{u}}{\partial [\mathbf{W}_1]_{ij}}$$

$$\frac{\partial \mathbf{u}}{\partial [\mathbf{W}_1]_{ij}} = \frac{\partial}{\partial [\mathbf{W}_1]_{ij}} (\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

Reminder: Derivative of $\mathbf{f}(\mathbf{W}) = \mathbf{W}\mathbf{x} + \mathbf{b}$ w.r.t. a single entry W_{ij}:

$$\Big[\frac{\partial \mathbf{f}}{\partial W_{ij}}\Big]_k = \frac{\partial f_k}{\partial W_{ij}} = \frac{\partial}{\partial W_{ij}} \Big(\sum_l W_{kl} x_l + b_k \Big) = \begin{cases} x_j & k = i \\ 0 & k \neq i \end{cases} \quad \Rightarrow \quad \frac{\partial \mathbf{f}}{\partial W_{ij}} = x_j \, \mathbf{e}_i$$

where $\mathbf{e}_i$ is the i-th unit vector.

Step 4: $\partial \mathbf{u} / \partial [\mathbf{W}_1]_{ij}$

$$\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} = \frac{\partial y}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial \mathbf{u}} \cdot \frac{\partial \mathbf{u}}{\partial [\mathbf{W}_1]_{ij}}$$

$$\frac{\partial \mathbf{u}}{\partial [\mathbf{W}_1]_{ij}} = \frac{\partial}{\partial [\mathbf{W}_1]_{ij}} \big(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1 \big) = x_j \, \mathbf{e}_i$$

Reminder: Derivative of $\mathbf{f}(\mathbf{W}) = \mathbf{W}\mathbf{x} + \mathbf{b}$ w.r.t. a single entry W_{ij}:

$$\Big[\frac{\partial \mathbf{f}}{\partial W_{ij}} \Big]_k = \frac{\partial f_k}{\partial W_{ij}} = \frac{\partial}{\partial W_{ij}} \Big(\sum_l W_{kl} x_l + b_k \Big) = \begin{cases} x_j & k = i \\ 0 & k \neq i \end{cases} \quad \Rightarrow \quad \frac{\partial \mathbf{f}}{\partial W_{ij}} = x_j \, \mathbf{e}_i$$

where $\mathbf{e}_i$ is the i-th unit vector.

Step 4: stitching the results together

$$\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} = \frac{\partial y}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial \mathbf{u}} \cdot \frac{\partial \mathbf{u}}{\partial [\mathbf{W}_1]_{ij}}$$

$$\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} = \mathbf{W}_2 \ \mathrm{diag}\left(\mathsf{ReLU}'(u_1), \ldots, \mathsf{ReLU}'(u_m)\right) x_j \, \mathbf{e}_i$$

Step 4: stitching the results together

$$\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} = \frac{\partial y}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial \mathbf{u}} \cdot \frac{\partial \mathbf{u}}{\partial [\mathbf{W}_1]_{ij}}$$

$$\begin{aligned}
\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} &= \mathbf{W}_2 \ \mathrm{diag}\left(\mathsf{ReLU}'(u_1), \ldots, \mathsf{ReLU}'(u_m)\right) x_j \, \mathbf{e}_i \\
&= x_j \, \mathbf{W}_2 \ \mathrm{diag}\left(\mathsf{ReLU}'(u_1), \ldots, \mathsf{ReLU}'(u_m)\right) \mathbf{e}_i
\end{aligned}$$

Step 4: stitching the results together

$$\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} = \frac{\partial y}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial \mathbf{u}} \cdot \frac{\partial \mathbf{u}}{\partial [\mathbf{W}_1]_{ij}}$$

$$\begin{aligned}
\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} &= \mathbf{W}_2 \ \mathrm{diag}\left(\mathsf{ReLU}'(u_1), \ldots, \mathsf{ReLU}'(u_m)\right) x_j \, \mathbf{e}_i \\
&= x_j \, \mathbf{W}_2 \ \mathrm{diag}\left(\mathsf{ReLU}'(u_1), \ldots, \mathsf{ReLU}'(u_m)\right) \mathbf{e}_i \\
&= x_j \ \begin{pmatrix} [\mathbf{W}_2]_{1,1} \, \mathsf{ReLU}'(u_1) & \cdots & [\mathbf{W}_2]_{1,m} \, \mathsf{ReLU}'(u_m) \end{pmatrix} \mathbf{e}_i
\end{aligned}$$

Step 4: stitching the results together

$$\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} = \frac{\partial y}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial \mathbf{u}} \cdot \frac{\partial \mathbf{u}}{\partial [\mathbf{W}_1]_{ij}}$$

$$\begin{aligned}
\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} &= \mathbf{W}_2 \ \mathrm{diag}\left(\mathsf{ReLU}'(u_1), \ldots, \mathsf{ReLU}'(u_m)\right) x_j \, \mathbf{e}_i \\
&= x_j \, \mathbf{W}_2 \ \mathrm{diag}\left(\mathsf{ReLU}'(u_1), \ldots, \mathsf{ReLU}'(u_m)\right) \mathbf{e}_i \\
&= x_j \begin{pmatrix} [\mathbf{W}_2]_{1,1} \, \mathsf{ReLU}'(u_1) & \cdots & [\mathbf{W}_2]_{1,m} \, \mathsf{ReLU}'(u_m) \end{pmatrix} \mathbf{e}_i
\end{aligned}$$

Reminder: Multiplying a row vector with the unit vector $\mathbf{e}_i$ picks its i-th entry:

$$\begin{pmatrix} a_1 & \cdots & a_m \end{pmatrix} \mathbf{e}_i = a_i$$

Step 4: stitching the results together

$$\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} = \frac{\partial y}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial \mathbf{u}} \cdot \frac{\partial \mathbf{u}}{\partial [\mathbf{W}_1]_{ij}}$$

$$\begin{aligned}
\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} &= \mathbf{W}_2 \ \mathrm{diag}\left(\mathsf{ReLU}'(u_1), \ldots, \mathsf{ReLU}'(u_m)\right) x_j \, \mathbf{e}_i \\
&= x_j \, \mathbf{W}_2 \ \mathrm{diag}\left(\mathsf{ReLU}'(u_1), \ldots, \mathsf{ReLU}'(u_m)\right) \mathbf{e}_i \\
&= x_j \begin{pmatrix} [\mathbf{W}_2]_{1,1} \, \mathsf{ReLU}'(u_1) & \cdots & [\mathbf{W}_2]_{1,m} \, \mathsf{ReLU}'(u_m) \end{pmatrix} \mathbf{e}_i \\
&= x_j \, [\mathbf{W}_2]_{1,i} \, \mathsf{ReLU}'(u_i)
\end{aligned}$$

Reminder: Multiplying a row vector with the unit vector $\mathbf{e}_i$ picks its i-th entry:

$$\begin{pmatrix} a_1 & \cdots & a_m \end{pmatrix} \mathbf{e}_i = a_i$$

Step 5: plug in the values

$$\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} = x_j \, [\mathbf{W}_2]_{1,i} \, \mathrm{ReLU}'(u_i)$$

Step 5: plug in the values

$$\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} = x_j \, [\mathbf{W}_2]_{1,i} \, \mathrm{ReLU}'(u_i)$$

Our specific values:

$$\mathbf{x} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \mathbf{W}_2 = \begin{pmatrix} 1 & 2 \end{pmatrix}, \quad \mathbf{u} = \begin{pmatrix} 2 \\ 0 \end{pmatrix} \quad \Rightarrow \quad \mathrm{ReLU}'(u_1) = 1, \quad \mathrm{ReLU}'(u_2) = 0$$

Step 5: plug in the values

$$\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} = x_j \, [\mathbf{W}_2]_{1,i} \, \mathrm{ReLU}'(u_i)$$

Our specific values:

$$\mathbf{x} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \mathbf{W}_2 = \begin{pmatrix} 1 & 2 \end{pmatrix}, \quad \mathbf{u} = \begin{pmatrix} 2 \\ 0 \end{pmatrix} \quad \Rightarrow \quad \mathrm{ReLU}'(u_1) = 1, \quad \mathrm{ReLU}'(u_2) = 0$$

Arrange the entries into the gradient:

$$\nabla_{\mathbf{W}_1} y = \begin{pmatrix} x_1 [\mathbf{W}_2]_{1,1} \mathrm{ReLU}'(u_1) & x_2 [\mathbf{W}_2]_{1,1} \mathrm{ReLU}'(u_1) \\ x_1 [\mathbf{W}_2]_{1,2} \mathrm{ReLU}'(u_2) & x_2 [\mathbf{W}_2]_{1,2} \mathrm{ReLU}'(u_2) \end{pmatrix}$$

Step 5: plug in the values

$$\frac{\partial y}{\partial [\mathbf{W}_1]_{ij}} = x_j \, [\mathbf{W}_2]_{1,i} \, \mathrm{ReLU}'(u_i)$$

Our specific values:

$$\mathbf{x} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \mathbf{W}_2 = \begin{pmatrix} 1 & 2 \end{pmatrix}, \quad \mathbf{u} = \begin{pmatrix} 2 \\ 0 \end{pmatrix} \quad \Rightarrow \quad \mathrm{ReLU}'(u_1) = 1, \quad \mathrm{ReLU}'(u_2) = 0$$

Arrange the entries into the gradient:

$$\nabla_{\mathbf{W}_1} y = \begin{pmatrix} x_1 [\mathbf{W}_2]_{1,1} \mathrm{ReLU}'(u_1) & x_2 [\mathbf{W}_2]_{1,1} \mathrm{ReLU}'(u_1) \\ x_1 [\mathbf{W}_2]_{1,2} \mathrm{ReLU}'(u_2) & x_2 [\mathbf{W}_2]_{1,2} \mathrm{ReLU}'(u_2) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$$

Practice for home

Show that

$$\nabla_{\mathbf{W}_1} y = \left(\mathbf{W}_2^\top \odot \mathrm{ReLU}'(\mathbf{u})\right) \mathbf{x}^\top,$$

where $\odot$ is the Hadamard product (elementwise multiplication).

Backpropagation with Tensors

Old exam question (2024)

Assume that we are given as input a matrix $\mathbf{X} \in \mathbb{R}^{B \times C}$ and we want to apply the softmax function across the second dimension. Assume that our output is called $\mathbf{Y}$.

a) What shape does $\mathbf{Y}$ have? Define the batchwise softmax function in terms of its elements y_{ij}.

c) Given a loss $\mathcal{L}(\mathbf{Y})$, specify $\partial \mathcal{L} / \partial x_{ij}$ as well as $\partial y_{ic} / \partial x_{ij}$. You can consider the partial derivative $\partial \mathcal{L} / \partial y_{ic}$ as given.

Task a)

What shape does $\mathbf{Y}$ have for $\mathbf{X} \in \mathbb{R}^{B \times C}$? Define the batchwise softmax via y_{ij}.

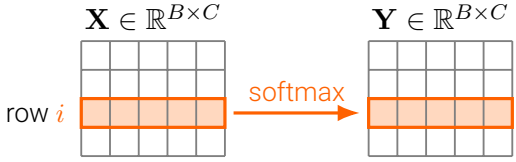
One sample: $\mathbf{x} \in \mathbb{R}^C \Rightarrow \mathbf{y} \in \mathbb{R}^C, \quad y_j = \frac{\exp(x_j)}{\sum_{c=1}^{C} \exp(x_c)}$

Task a)

What shape does $\mathbf{Y}$ have for $\mathbf{X} \in \mathbb{R}^{B \times C}$? Define the batchwise softmax via y_{ij}.

One sample: $\mathbf{x} \in \mathbb{R}^C \Rightarrow \mathbf{y} \in \mathbb{R}^C, \quad y_j = \dfrac{\exp(x_j)}{\sum_{c=1}^{C} \exp(x_c)}$

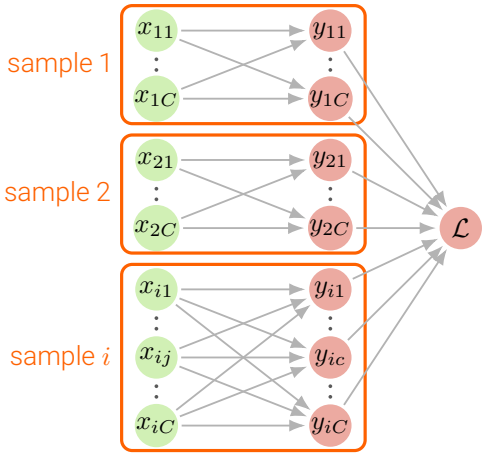
Trick: row i of $\mathbf{Y}$ only depends on row i of $\mathbf{X}$, **the samples never mix**.



One batch: $\mathbf{X} \in \mathbb{R}^{B \times C} \Rightarrow \mathbf{Y} \in \mathbb{R}^{B \times C}, \quad y_{ij} = \dfrac{\exp(x_{ij})}{\sum_{c=1}^{C} \exp(x_{ic})}$

Task c): the chain rule

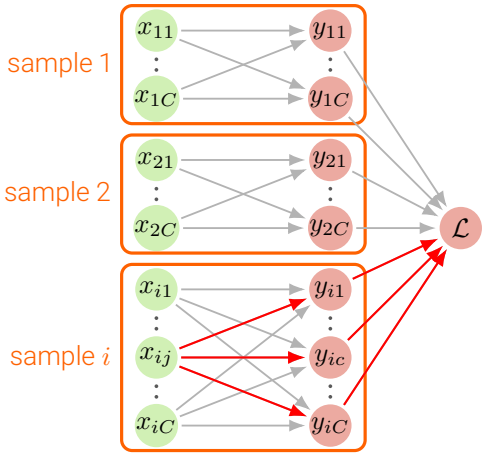
For $\mathcal{L}(\mathbf{Y})$, specify $\partial\mathcal{L}/\partial x_{ij}$ and $\partial y_{ic}/\partial x_{ij}$. $\partial\mathcal{L}/\partial y_{ic}$ is given.



Chain rule: sum over all paths from $\mathcal{L}$ to x_{ij}

Task c): the chain rule

For $\mathcal{L}(\mathbf{Y})$, specify $\partial\mathcal{L}/\partial x_{ij}$ and $\partial y_{ic}/\partial x_{ij}$. $\partial\mathcal{L}/\partial y_{ic}$ is given.



Chain rule: sum over all paths from $\mathcal{L}$ to x_{ij}

x_{ij} only reaches $\mathcal{L}$ through $y_{i1}, \ldots, y_{iC}$:

$$\frac{\partial \mathcal{L}}{\partial x_{ij}} = \sum_{c=1}^{C} \frac{\partial \mathcal{L}}{\partial y_{ic}} \frac{\partial y_{ic}}{\partial x_{ij}}$$

Task c): the softmax derivative

For $\mathcal{L}(\mathbf{Y})$, specify $\partial\mathcal{L}/\partial x_{ij}$ and $\partial y_{ic}/\partial x_{ij}$. $\partial\mathcal{L}/\partial y_{ic}$ is given.

One sample: the softmax derivative from exercise sheet 2

$$\frac{\partial y_c}{\partial x_j} = \begin{cases} y_j(1 - y_j) & c = j \\ -y_c\, y_j & c \neq j \end{cases}$$

Task c): the softmax derivative

For $\mathcal{L}(\mathbf{Y})$, specify $\partial\mathcal{L}/\partial x_{ij}$ and $\partial y_{ic}/\partial x_{ij}$. $\partial\mathcal{L}/\partial y_{ic}$ is given.

One sample: the softmax derivative from exercise sheet 2

$$\frac{\partial y_c}{\partial x_j} = \begin{cases} y_j(1-y_j) & c = j \\ -y_c\, y_j & c \neq j \end{cases}$$

One batch: add the batch index

$$\frac{\partial y_{ic}}{\partial x_{ij}} = \begin{cases} y_{ij}(1-y_{ij}) & c = j \\ -y_{ic}\, y_{ij} & c \neq j \end{cases}$$

Old exam question (2024): implementation

Softmax across the second dimension of $\mathbf{X} \in \mathbb{R}^{B \times C}$, output $\mathbf{Y}$.

a) $\mathbf{Y} \in \mathbb{R}^{B \times C}, \quad y_{ij} = \dfrac{\exp(x_{ij})}{\sum_{c=1}^{C} \exp(x_{ic})}$

b) Implement the forward pass of the softmax function from task a) in EDF. You only need to write the body of the forward function.

c) $\dfrac{\partial \mathcal{L}}{\partial x_{ij}} = \sum_{c=1}^{C} \dfrac{\partial \mathcal{L}}{\partial y_{ic}} \dfrac{\partial y_{ic}}{\partial x_{ij}}, \quad \dfrac{\partial y_{ic}}{\partial x_{ij}} = \begin{cases} y_{ij}(1 - y_{ij}) & c = j \\ -y_{ic}\, y_{ij} & c \neq j \end{cases}$

d) Implement the backward pass of the softmax function in EDF. You only need to write the body of the backward function. It is **not** required to use vectorized numpy code. An implementation with nested for-loops suffices.

Recap: EDF, the forward pass

b) Implement the forward pass of the softmax function from task a) in EDF.



Recap: EDF, the forward pass

b) Implement the forward pass of the softmax function from task a) in EDF.



$$\texttt{self.value[i, j]} = y_{ij} = \frac{\exp(x_{ij})}{\sum_{c=1}^{C} \exp(x_{ic})} = \frac{\texttt{exp(self.x.value[i, j])}}{\sum_{c} \texttt{exp(self.x.value[i, c])}}$$

Recap: EDF, the forward pass

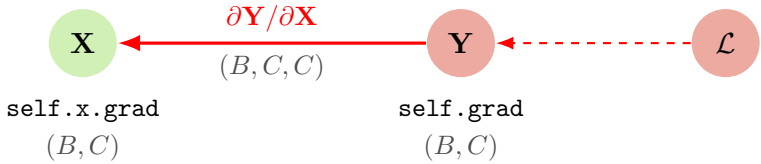
b) Implement the forward pass of the softmax function from task a) in EDF.



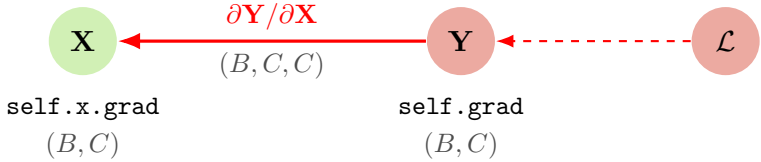
$$\texttt{self.value[i, j]} = y_{ij} = \frac{\exp(x_{ij})}{\sum_{c=1}^{C} \exp(x_{ic})} = \frac{\texttt{exp(self.x.value[i, j])}}{\sum_c \texttt{exp(self.x.value[i, c])}}$$

```
def forward(self):
    exp_x = np.exp(self.x.value)                                  # (B, C)
    self.value = exp_x / exp_x.sum(axis=1, keepdims=True)   # sum over c, keep (B, 1)
```

Recap: EDF, the backward pass



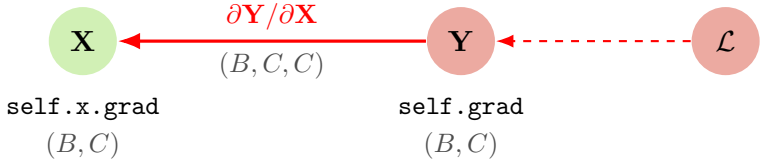
Recap: EDF, the backward pass



`self.grad[i, c]` $= \partial \mathcal{L}/\partial y_{ic}$ given, filled in by the nodes after $\mathbf{Y}$

`self.x.grad[i, j]` $= \partial \mathcal{L}/\partial x_{ij}$ our job, via `self.x.addgrad(...)`

Recap: EDF, the backward pass



`self.grad[i, c]` $= \partial \mathcal{L}/\partial y_{ic}$ given, filled in by the nodes after $\mathbf{Y}$

`self.x.grad[i, j]` $= \partial \mathcal{L}/\partial x_{ij}$ our job, via `self.x.addgrad(...)`

Reminder chain rule:

$$\frac{\partial \mathcal{L}}{\partial x_{ij}} = \sum_{c=1}^{C} \frac{\partial \mathcal{L}}{\partial y_{ic}} \frac{\partial y_{ic}}{\partial x_{ij}}$$

$$\Rightarrow \quad \texttt{self.x.grad[i, j]} = \sum_{c=1}^{C} \texttt{self.grad[i, c]} \cdot \frac{\partial y_{ic}}{\partial x_{ij}}$$

Task d): the backward pass

d) Implement the backward pass of the softmax in EDF (for-loops suffice).

Task d): the backward pass

d) Implement the backward pass of the softmax in EDF (for-loops suffice).

$$\texttt{self.x.grad[i, j]} = \sum_{c=1}^{C} \texttt{self.grad[i, c]} \cdot \frac{\partial y_{ic}}{\partial x_{ij}}, \qquad \frac{\partial y_{ic}}{\partial x_{ij}} = \begin{cases} y_{ij}(1 - y_{ij}) & c = j \\ -y_{ic}\, y_{ij} & c \neq j \end{cases}$$

Task d): the backward pass

d) Implement the backward pass of the softmax in EDF (for-loops suffice).

$$\texttt{self.x.grad[i, j]} = \sum_{c=1}^{C} \texttt{self.grad[i, c]} \cdot \frac{\partial y_{ic}}{\partial x_{ij}}, \qquad \frac{\partial y_{ic}}{\partial x_{ij}} = \begin{cases} y_{ij}(1 - y_{ij}) & c = j \\ -y_{ic}\, y_{ij} & c \neq j \end{cases}$$

Hint: create `delta = np.zeros((B, C))`, fill every entry `delta[i, j]` with the formula above in nested for-loops, then call `self.x.addgrad(delta)`.

Task d): the backward pass

d) Implement the backward pass of the softmax in EDF (for-loops suffice).

$$\texttt{self.x.grad[i, j]} = \sum_{c=1}^{C} \texttt{self.grad[i, c]} \cdot \frac{\partial y_{ic}}{\partial x_{ij}}, \qquad \frac{\partial y_{ic}}{\partial x_{ij}} = \begin{cases} y_{ij}(1 - y_{ij}) & c = j \\ -y_{ic}\, y_{ij} & c \neq j \end{cases}$$

```
def backward(self):
    B, C = self.value.shape
    delta = np.zeros((B, C))
    for i in range(B):
        for j in range(C):
            for c in range(C):
                dy_dx = self.value[i, c] * ((c == j) - self.value[i, j])
                delta[i, j] += self.grad[i, c] * dy_dx
    self.x.addgrad(delta)        # self.x.grad += delta
```