From Logistic Regression to Neural Networks

CS114 Lab 5

Kenneth Lai

February 14, 2020

Logistic Regression

- For each feature i:
  - Value x_i
  - Weight w_i

Logistic Regression

- For each feature i:
  - Value x_i
  - Weight w_i
- Bias term b

Logistic Regression

- For each feature i:
  - Value x_i
  - Weight w_i
- Bias term b
- "Score" (log-odds) $z = \left(\sum_{i=1}^{n} w_i x_i \right) + b = \mathbf{w} \cdot \mathbf{x} + b$

Logistic Regression

- For each feature i:
  - Value x_i
  - Weight w_i
- Bias term b
- "Score" (log-odds) $z = \left(\sum_{i=1}^{n} w_i x_i \right) + b = \mathbf{w} \cdot \mathbf{x} + b$
- Logistic function $\sigma(z) = \frac{1}{1 + e^{-z}} = \hat{y}$

Logistic Regression

- For each feature i:
  - Value x_i
  - Weight w_i
- Bias term b
- "Score" (log-odds) $z = \left(\sum_{i=1}^{n} w_i x_i\right) + b = \mathbf{w} \cdot \mathbf{x} + b$
- Logistic function $\sigma(z) = \frac{1}{1 + e^{-z}} = \hat{y}$
- Cross-entropy loss $L = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$

Gradient Descent

- Compute gradient $\nabla L = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \vdots \\ \frac{\partial L}{\partial w_n} \\ \frac{\partial L}{\partial b} \end{bmatrix}$, where

Gradient Descent

- Compute gradient $\nabla L = \begin{bmatrix} \dfrac{\partial L}{\partial w_1} \\ \vdots \\ \dfrac{\partial L}{\partial w_n} \\ \dfrac{\partial L}{\partial b} \end{bmatrix}$, where
  - $\dfrac{\partial L}{\partial w_i} = (\hat{y} - y)x_i$
  - $\dfrac{\partial L}{\partial b} = \hat{y} - y$

Multinomial Logistic Regression

- Separate weights and bias terms for each class c

Multinomial Logistic Regression

- Separate weights and bias terms for each class c

$$z_c = \left(\sum_{i=1}^{n} w_{ic} x_i \right) + b_c = \mathbf{w}_c \cdot \mathbf{x} + b_c$$

Multinomial Logistic Regression

- Separate weights and bias terms for each class c

$$z_c = \left(\sum_{i=1}^{n} w_{ic} x_i \right) + b_c = \mathbf{w}_c \cdot \mathbf{x} + b_c$$

- Softmax function $\sigma(z_c) = \frac{e^{z_c}}{\sum_{k \in C} e^{z_k}} = P(y = c|\mathbf{x}) = \hat{y}_c$

Multinomial Logistic Regression

- Separate weights and bias terms for each class c

$$z_c = \left(\sum_{i=1}^{n} w_{ic} x_i \right) + b_c = \mathbf{w}_c \cdot \mathbf{x} + b_c$$

- Softmax function $\sigma(z_c) = \frac{e^{z_c}}{\sum_{k \in C} e^{z_k}} = P(y = c|\mathbf{x}) = \hat{y}_c$

- Cross-entropy loss $L = -\sum_{k \in C} y_k \log \hat{y}_k$

Multinomial Logistic Regression

- Gradient ∇L becomes a matrix, where

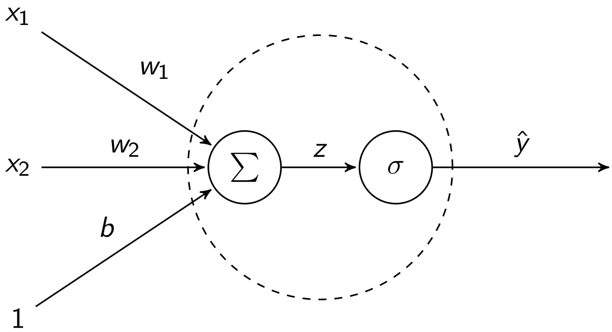
Multinomial Logistic Regression

- Gradient ∇L becomes a matrix, where
  - $\dfrac{\partial L}{\partial w_{ic}} = (\hat{y}_c - y_c)x_i$
  - $\dfrac{\partial L}{\partial b_c} = \hat{y}_c - y_c$

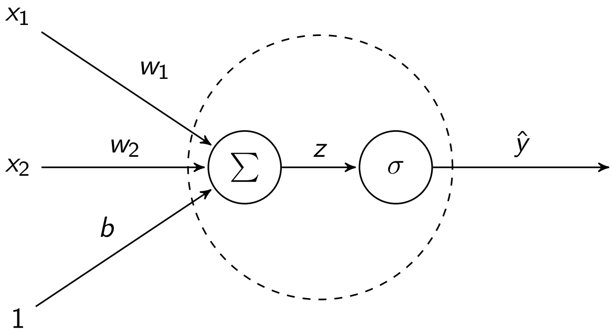
Example

- Python time!

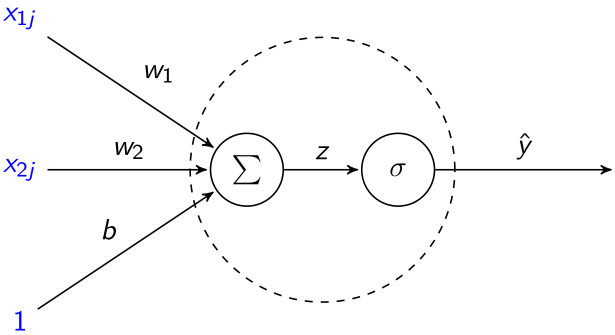
Graphical Representation of Logistic Regression



Graphical Representation of a Neuron

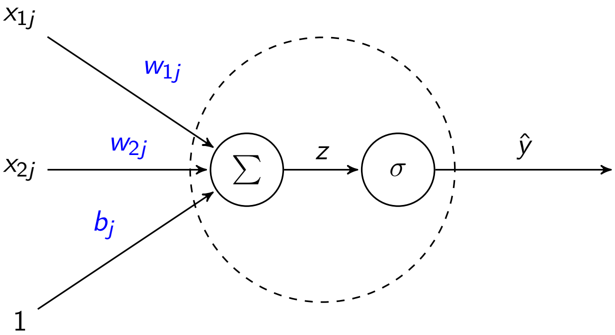


Graphical Representation of a Neuron



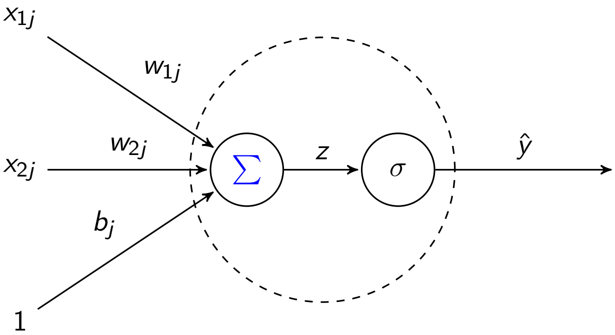
- Inputs (to neuron j)

Graphical Representation of a Neuron



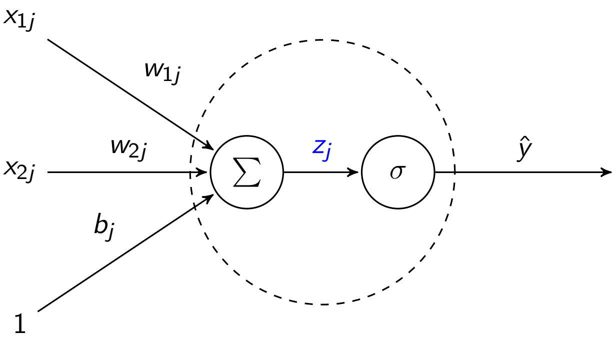
- Weights (and bias term)

Graphical Representation of a Neuron



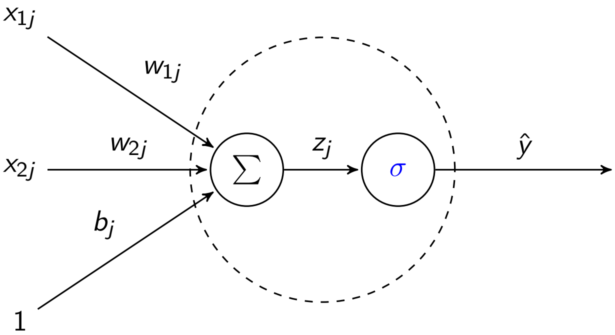
- Input function (almost always $\sum$)

Graphical Representation of a Neuron



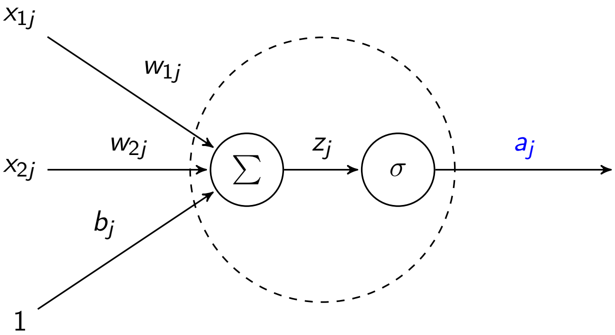
- “Score”

Graphical Representation of a Neuron



- Activation function (logistic, softmax, tanh, ReLU, etc.)

Graphical Representation of a Neuron



- Activation (output of neuron j)

Gradients in Logistic Regression

- We want to compute $\dfrac{\partial L}{\partial w_i}$

Gradients in Logistic Regression

- We want to compute $\dfrac{\partial L}{\partial w_i}$
- Chain Rule of calculus: $\dfrac{dy}{dx} = \dfrac{dy}{dz}\dfrac{dz}{dx}$

Gradients in Logistic Regression

- We want to compute $\frac{\partial L}{\partial w_i}$
- Chain Rule of calculus: $\frac{dy}{dx} = \frac{dy}{dz}\frac{dz}{dx}$
- Looking at the graph: $\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z}\frac{\partial z}{\partial w_i}$

Gradients in Logistic Regression

- $$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_i}$$

Gradients in Logistic Regression

- $\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} x_i$
  - $\frac{\partial z}{\partial w_i} = x_i$

Gradients in Logistic Regression

- $\dfrac{\partial L}{\partial w_i} = \dfrac{\partial L}{\partial \hat{y}} \hat{y}(1 - \hat{y}) x_i$
  - $\dfrac{\partial z}{\partial w_i} = x_i$
  - For the logistic function: $\dfrac{\partial \hat{y}}{\partial z} = \hat{y}(1 - \hat{y})$

Gradients in Logistic Regression

- $$\frac{\partial L}{\partial w_i} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})} \hat{y}(1 - \hat{y}) x_i$$
  - $$\frac{\partial z}{\partial w_i} = x_i$$
  - For the logistic function: $\frac{\partial \hat{y}}{\partial z} = \hat{y}(1 - \hat{y})$
  - For cross-entropy loss: $\frac{\partial L}{\partial \hat{y}} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})}$

Gradients in Logistic Regression

- $\frac{\partial L}{\partial w_i} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})}\hat{y}(1-\hat{y})x_i = (\hat{y} - y)x_i$
  - $\frac{\partial z}{\partial w_i} = x_i$
  - For the logistic function: $\frac{\partial \hat{y}}{\partial z} = \hat{y}(1-\hat{y})$
  - For cross-entropy loss: $\frac{\partial L}{\partial \hat{y}} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})}$

Gradients in Feedforward Neural Networks

- For an output neuron:

Gradients in Feedforward Neural Networks

- For an output neuron:
  - $\dfrac{\partial L}{\partial w_{ij}} = \dfrac{a_j - y}{a_j(1-a_j)} a_j(1-a_j)x_{ij} = (a_j - y)x_{ij}$
    - $\dfrac{\partial z_j}{\partial w_{ij}} = x_{ij}$
    - For the logistic function: $\dfrac{\partial a_j}{\partial z_j} = a_j(1-a_j)$
    - For cross-entropy loss: $\dfrac{\partial L}{\partial a_j} = \dfrac{a_j - y}{a_j(1-a_j)}$

Gradients in Feedforward Neural Networks

- For an output neuron:
  - $\frac{\partial L}{\partial w_{ij}} = \frac{a_j - y}{a_j(1-a_j)} a_j(1-a_j)x_{ij} = (a_j - y)x_{ij}$
    - $\frac{\partial z_j}{\partial w_{ij}} = x_{ij}$
    - For the logistic function: $\frac{\partial a_j}{\partial z_j} = a_j(1-a_j)$
    - For cross-entropy loss: $\frac{\partial L}{\partial a_j} = \frac{a_j - y}{a_j(1-a_j)}$
  - Note that for an output neuron, $a_j = \hat{y}$

Gradients in Feedforward Neural Networks

- For a non-output neuron:

Gradients in Feedforward Neural Networks

- For a non-output neuron:
  - $$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial a_j}\frac{\partial a_j}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}}$$

Gradients in Feedforward Neural Networks

- For a non-output neuron:
  - $\dfrac{\partial L}{\partial w_{ij}} = \dfrac{\partial L}{\partial a_j} a_j(1 - a_j) x_{ij}$
    - $\dfrac{\partial z_j}{\partial w_{ij}} = x_{ij}$
    - For the logistic function: $\dfrac{\partial a_j}{\partial z_j} = a_j(1 - a_j)$

Gradients in Feedforward Neural Networks

- For a non-output neuron:
  - $\dfrac{\partial L}{\partial w_{ij}} = \dfrac{\partial L}{\partial a_j} a_j(1 - a_j)x_{ij}$
    - $\dfrac{\partial z_j}{\partial w_{ij}} = x_{ij}$
    - For the logistic function: $\dfrac{\partial a_j}{\partial z_j} = a_j(1 - a_j)$
    - $\dfrac{\partial L}{\partial a_j} = ?$
  - Note that for a non-output neuron, $a_j \neq \hat{y}$

Gradients in Feedforward Neural Networks

- Note that for a non-output neuron, $a_j = x_{j\ell}$ for some other neuron(s) ℓ

Gradients in Feedforward Neural Networks

- Note that for a non-output neuron, $a_j = x_{j\ell}$ for some other neuron(s) ℓ
  - Let $\mathcal{L}$ be the set of all such ℓ

Gradients in Feedforward Neural Networks

- Note that for a non-output neuron, $a_j = x_{j\ell}$ for some other neuron(s) ℓ
  - Let $\mathcal{L}$ be the set of all such ℓ
- Chain Rule of multivariable calculus:

$$\frac{df(g_1(x), \ldots, g_n(x))}{dx} = \sum_{i=1}^{n} \frac{\partial f}{\partial g_i(x)} \frac{dg_i(x)}{dx}$$

Gradients in Feedforward Neural Networks

- Note that for a non-output neuron, $a_j = x_{j\ell}$ for some other neuron(s) ℓ
  - Let $\mathcal{L}$ be the set of all such ℓ
- Chain Rule of multivariable calculus:

$$\frac{df(g_1(x), \ldots, g_n(x))}{dx} = \sum_{i=1}^{n} \frac{\partial f}{\partial g_i(x)} \frac{dg_i(x)}{dx}$$

- Express L as a function of z_ℓ: $\dfrac{\partial L}{\partial a_j} = \sum_{\ell \in \mathcal{L}} \dfrac{\partial L}{\partial z_\ell} \dfrac{\partial z_\ell}{\partial x_{j\ell}}$

Gradients in Feedforward Neural Networks

- For a non-output neuron:
  - $\dfrac{\partial L}{\partial w_{ij}} = \dfrac{\partial L}{\partial a_j} a_j(1 - a_j)x_{ij}$
    - $\dfrac{\partial z_j}{\partial w_{ij}} = x_{ij}$
    - For the logistic function: $\dfrac{\partial a_j}{\partial z_j} = a_j(1 - a_j)$

Gradients in Feedforward Neural Networks

- For a non-output neuron:
  - $$\frac{\partial L}{\partial w_{ij}} = \left(\sum_{\ell \in \mathcal{L}} \frac{\partial L}{\partial z_\ell} \frac{\partial z_\ell}{\partial x_{j\ell}} \right) a_j(1 - a_j) x_{ij}$$
    - $$\frac{\partial z_j}{\partial w_{ij}} = x_{ij}$$
    - For the logistic function: $\frac{\partial a_j}{\partial z_j} = a_j(1 - a_j)$
    - $$\frac{\partial L}{\partial a_j} = \sum_{\ell \in \mathcal{L}} \frac{\partial L}{\partial z_\ell} \frac{\partial z_\ell}{\partial x_{j\ell}}$$

Gradients in Feedforward Neural Networks

- For a non-output neuron:
  - $$\frac{\partial L}{\partial w_{ij}} = \left(\sum_{\ell \in \mathcal{L}} \frac{\partial L}{\partial z_\ell} w_{j\ell} \right) a_j(1 - a_j)x_{ij}$$
    - $\frac{\partial z_j}{\partial w_{ij}} = x_{ij}$
    - For the logistic function: $\frac{\partial a_j}{\partial z_j} = a_j(1 - a_j)$
    - $\frac{\partial L}{\partial a_j} = \sum_{\ell \in \mathcal{L}} \frac{\partial L}{\partial z_\ell} \frac{\partial z_\ell}{\partial x_{j\ell}}$
    - $\frac{\partial z_\ell}{\partial x_{j\ell}} = w_{j\ell}$

Gradients in Feedforward Neural Networks

- For a non-output neuron:
  - $\dfrac{\partial L}{\partial w_{ij}} = \left(\sum_{\ell \in \mathcal{L}} \dfrac{\partial L}{\partial z_\ell} w_{j\ell} \right) a_j (1 - a_j) x_{ij}$
    - $\dfrac{\partial z_j}{\partial w_{ij}} = x_{ij}$
    - For the logistic function: $\dfrac{\partial a_j}{\partial z_j} = a_j(1 - a_j)$
    - $\dfrac{\partial L}{\partial a_j} = \sum_{\ell \in \mathcal{L}} \dfrac{\partial L}{\partial z_\ell} \dfrac{\partial z_\ell}{\partial x_{j\ell}}$
    - $\dfrac{\partial z_\ell}{\partial x_{j\ell}} = w_{j\ell}$
  - What about $\dfrac{\partial L}{\partial z_\ell}$?

Backpropagation

- We can compute $\dfrac{\partial L}{\partial w_{j\ell}} = \dfrac{\partial L}{\partial a_\ell}\dfrac{\partial a_\ell}{\partial z_\ell}\dfrac{\partial z_\ell}{\partial w_{j\ell}}$ for an output neuron ℓ

Backpropagation

- We can compute $\frac{\partial L}{\partial w_{j\ell}} = \frac{\partial L}{\partial a_\ell} \frac{\partial a_\ell}{\partial z_\ell} \frac{\partial z_\ell}{\partial w_{j\ell}}$ for an output neuron ℓ
- If we have already computed $\frac{\partial L}{\partial w_{j\ell}}$ for some neuron ℓ, then we have also computed $\frac{\partial L}{\partial z_\ell} \left(= \frac{\partial L}{\partial a_\ell} \frac{\partial a_\ell}{\partial z_\ell} \right)$

Backpropagation

- We can compute $\dfrac{\partial L}{\partial w_{j\ell}} = \dfrac{\partial L}{\partial a_\ell}\dfrac{\partial a_\ell}{\partial z_\ell}\dfrac{\partial z_\ell}{\partial w_{j\ell}}$ for an output neuron ℓ
- If we have already computed $\dfrac{\partial L}{\partial w_{j\ell}}$ for some neuron ℓ, then we have also computed $\dfrac{\partial L}{\partial z_\ell}\left(= \dfrac{\partial L}{\partial a_\ell}\dfrac{\partial a_\ell}{\partial z_\ell}\right)$
- We can then use $\dfrac{\partial L}{\partial z_\ell}$ to calculate $\dfrac{\partial L}{\partial w_{ij}} = \left(\sum_{\ell \in \mathcal{L}} \dfrac{\partial L}{\partial z_\ell} w_{j\ell}\right)\dfrac{\partial a_j}{\partial z_j}\dfrac{\partial z_j}{\partial w_{ij}}$ for the previous neuron(s) j

Backpropagation

- Simplifying assumptions:

Backpropagation

- Simplifying assumptions:
  - Suppose that our neurons are grouped into a sequence of layers
  - Also suppose that these layers are fully connected (every neuron in one layer is connected to every neuron in the next layer, and no others)

Backpropagation

- Simplifying assumptions:
  - Suppose that our neurons are grouped into a sequence of layers
  - Also suppose that these layers are fully connected (every neuron in one layer is connected to every neuron in the next layer, and no others)
- Then we can use matrix multiplication

Backpropagation

- Notation:

Backpropagation

- Notation:
  - Let δ_l be the vector of $\dfrac{\partial L}{\partial z_i}$ for all neurons i in layer l
  - Let $\sigma'(\mathbf{z}_l)$ be the vector of $\dfrac{\partial a_i}{\partial z_i}$ for $i \in l$
  - Let $\odot$ denote the element-wise (Hadamard) product

Backpropagation

- Notation:
  - Let δ_l be the vector of $\dfrac{\partial L}{\partial z_i}$ for all neurons i in layer l
  - Let $\sigma'(\mathbf{z}_l)$ be the vector of $\dfrac{\partial a_i}{\partial z_i}$ for $i \in l$
  - Let $\odot$ denote the element-wise (Hadamard) product
- For all layers l:
  - $\nabla_l L = \delta_l \odot \mathbf{x}_l$

Backpropagation

- Notation:
  - Let δ_l be the vector of $\dfrac{\partial L}{\partial z_i}$ for all neurons i in layer l
  - Let $\sigma'(\mathbf{z}_l)$ be the vector of $\dfrac{\partial a_i}{\partial z_i}$ for $i \in l$
  - Let $\odot$ denote the element-wise (Hadamard) product
- For all layers l:
  - $\nabla_l L = \delta_l \odot \mathbf{x}_l$
- For an output layer $\mathcal{L}$:
  - $\delta_{\mathcal{L}} = \hat{\mathbf{y}} - \mathbf{y}$

Backpropagation

- Notation:
  - Let δ_l be the vector of $\dfrac{\partial L}{\partial z_i}$ for all neurons i in layer l
  - Let $\sigma'(\mathbf{z}_l)$ be the vector of $\dfrac{\partial a_i}{\partial z_i}$ for $i \in l$
  - Let $\odot$ denote the element-wise (Hadamard) product
- For all layers l:
  - $\nabla_l L = \delta_l \odot \mathbf{x}_l$
- For an output layer $\mathcal{L}$:
  - $\delta_{\mathcal{L}} = \hat{\mathbf{y}} - \mathbf{y}$
- For a non-output layer J (with next layer K):
  - $\delta_J = (\mathbf{W}_K^T \cdot \delta_K) \odot \sigma'(\mathbf{z}_J)$

Backpropagation

- For each (layer of) neuron(s) j:
  - Initialize parameters $\theta_j = \mathbf{w}_j, b_j$ randomly (note: not $\mathbf{0}$)

Backpropagation

- For each (layer of) neuron(s) j:
  - Initialize parameters $\theta_j = \mathbf{w}_j, b_j$ randomly (note: not $\mathbf{0}$)
- At each time step t:

Backpropagation

- For each (layer of) neuron(s) j:
  - Initialize parameters $\theta_j = \mathbf{w}_j, b_j$ randomly (note: not $\mathbf{0}$)
- At each time step t:
  - For each (layer of) neuron(s) j, starting from the output and working backwards:
    - Compute gradient $\nabla_j L$

Backpropagation

- For each (layer of) neuron(s) j:
  - Initialize parameters $\theta_j = \mathbf{w}_j, b_j$ randomly (note: not $\mathbf{0}$)
- At each time step t:
  - For each (layer of) neuron(s) j, starting from the output and working backwards:
    - Compute gradient $\nabla_j L$
  - For each (layer of) neuron(s) j:
    - Move in direction of negative gradient

Backpropagation

- For each (layer of) neuron(s) j:
  - Initialize parameters $\theta_j = \mathbf{w}_j, b_j$ randomly (note: not $\mathbf{0}$)
- At each time step t:
  - For each (layer of) neuron(s) j, starting from the output and working backwards:
    - Compute gradient $\nabla_j L$
  - For each (layer of) neuron(s) j:
    - Move in direction of negative gradient
- Because L is not necessarily convex anymore, we are not guaranteed to reach a global minimum

Backpropagation

- For each (layer of) neuron(s) j:
  - Initialize parameters $\theta_j = \mathbf{w}_j, b_j$ randomly (note: not $\mathbf{0}$)
- At each time step t:
  - For each (layer of) neuron(s) j, starting from the output and working backwards:
    - Compute gradient $\nabla_j L$
  - For each (layer of) neuron(s) j:
    - Move in direction of negative gradient
- Because L is not necessarily convex anymore, we are not guaranteed to reach a global minimum
  - But it works well enough in practice

Example

- Python time (again)!