AOS 2 – Deep learning

Lecture 01: Neural networks: an introduction

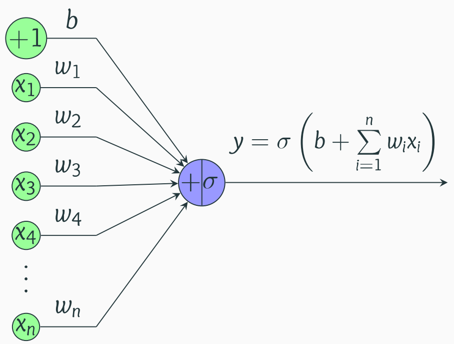
Sylvain Rousseau

Feed forward neural network

What is a neuron?

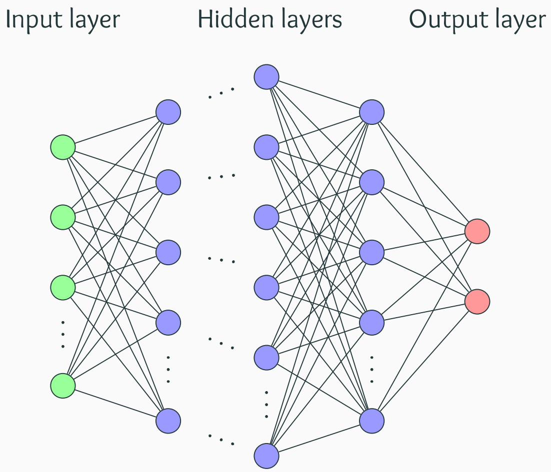
Neuron $=$ linear transform of the input followed by a non-linearity

- Input: $\mathbf{x} = (x_i)_{i=1\dots n} \in \mathbb{R}^n$
- Scalar output: $y \in \mathbb{R}$
- $+1$ denotes the intercept (or bias) b
- The w_i's are called **weights**
- σ is nonlinear function called an **activation function**



Feed forward neural network

- Stacked collection of neurons arranged in layers
- Input layer nodes are not neurons
- Output layer neurons do not have necessarily an activation function



Input layer and first hidden layer

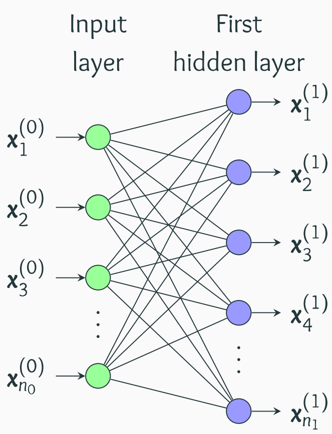
- Input: $\mathbf{x}^{(0)} = \left(\mathbf{x}_i^{(0)}\right)_{i=1\dots n_0} \in \mathbb{R}^{n_0}$
- Output: $\mathbf{x}^{(1)} = \left(\mathbf{x}_i^{(1)}\right)_{i=1\dots n_1} \in \mathbb{R}^{n_1}$

We have

$$\mathbf{x}_i^{(1)} = \sigma\left(\sum_{j=1}^{n_0} \boldsymbol{w}_{ij}^{(1)} \mathbf{x}_j^{(0)} + \boldsymbol{b}_i^{(1)}\right)$$

where

- $\boldsymbol{b}_i^{(1)}$ is the bias of neuron i (not displayed)
- $\boldsymbol{w}_{ij}^{(1)}$ is the j-th coefficient of i-th neuron
- Vector version $\mathbf{x}_i^{(1)} = \sigma\left(\left\langle \boldsymbol{w}_i^{(1)}, \mathbf{x}^{(0)} \right\rangle + \boldsymbol{b}_i^{(1)}\right)$

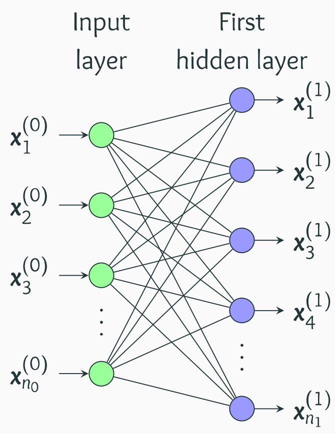


Input layer and first hidden layer

- Matrix version:

$$\mathbf{x}^{(1)} = \sigma\left(\left(W^{(1)}\right)^T \mathbf{x}^{(0)} + \boldsymbol{b}^{(1)}\right)$$

- σ is applied element-wise
- $W^{(1)}$ is of size $n_0 \times n_1$
- $\boldsymbol{w}_i^{(1)} \in \mathbb{R}^{n_0}$ columns of $W^{(1)}$
- $\boldsymbol{b}^{(1)} \in \mathbb{R}^{n_1}$ groups all the biases
- The parameters are $\Theta^{(1)} = \{W^{(1)}, \boldsymbol{b}^{(1)}\}$



Two hidden layers

- n_k is the number of neurons at layer k

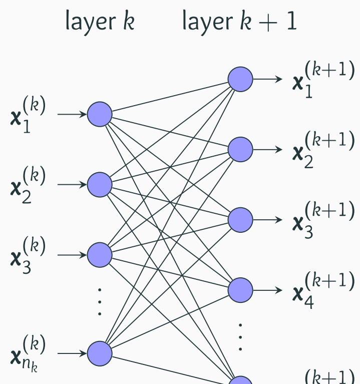
$$\mathbf{x}_i^{(k+1)} = \sigma\left(\sum_{j=1}^{n_k} \boldsymbol{w}_{ij}^{(k+1)} \mathbf{x}_j^{(k)} + \boldsymbol{b}_i^{(k+1)}\right)$$

- Condensed version

$$\mathbf{x}^{(k+1)} = \sigma\left(\left(W^{(k+1)}\right)^T \mathbf{x}^{(k)} + \boldsymbol{b}^{(k+1)}\right)$$

- The parameters at layer $k+1$ are

$$\Theta^{(k+1)} = \{W^{(k+1)}, \boldsymbol{b}^{(k+1)}\}$$



All layers

- Transformation at layer k:

$$F_{\Theta_k}(\mathbf{x}) = \sigma\left(\left(W^{(k)}\right)^T \mathbf{x} + \boldsymbol{b}^{(k)}\right)$$

- Suppose the neural network has K layers (input excluded, output included)

$$\mathbf{x}^{(K)} = F_{\Theta_K}\left(\mathbf{x}^{(K-1)}\right)$$

$$\vdots$$

$$\mathbf{x}^{(K)} = F_{\Theta_K}\left(F_{\Theta_{K-1}}\left(\dots F_{\Theta_1}\left(\mathbf{x}^{(0)}\right)\right)\right) = F_\Theta\left(\mathbf{x}^{(0)}\right)$$

- Parameters are

$$\Theta = \left\{\boldsymbol{b}^{(1)}, W^{(1)}, \boldsymbol{b}^{(2)}, W^{(2)}, \dots, \boldsymbol{b}^{(K)}, W^{(K)}\right\}$$

Loss function

- Measure the discrepancy between
  - the neural network prediction $\widehat{\boldsymbol{y}} = F_\Theta(\mathbf{x})$ with $\widehat{\boldsymbol{y}} = (\widehat{y}_1, \ldots, \widehat{y}_{n_K})$
  - the expected output $\boldsymbol{y} = (y_1, \ldots, y_{n_K})$
- Denoted $\ell(\widehat{\boldsymbol{y}}, \boldsymbol{y})$
- Mean squared error (MSE) loss:

$$\ell(\widehat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{n_K} \|\widehat{\boldsymbol{y}} - \boldsymbol{y}\|^2 = \frac{1}{n_K} \sum_{i=1}^{n_K} (\widehat{y}_i - y_i)^2$$

Cross-entropy

- Adapted to probabilistic output and probabilistic observation (non-negative, sum to 1)
- Cross-entropy error

$$\mathrm{CE}(\widehat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_{i=1}^{n_K} y_i \log \widehat{y}_i$$

- For classification task, $\boldsymbol{y}$ is one-hot encoded (all the y_i's are zero except one), in that case

$$\mathrm{CE}(\widehat{\boldsymbol{y}}, \boldsymbol{y}) = -\log \widehat{y}_k \quad \text{with } y_k = 1$$

Softmax

- Parameter-free normalizing transform

$$\mathrm{softmax} : \mathbb{R}^{n_K} \to \{(p_1, \ldots, p_{n_K}) \in \mathbb{R}^{n_K},\ p_i \geqslant 0, p_1 + \cdots + p_{n_K} = 1\}$$

- $\mathbf{z} = \mathrm{softmax}\,(\mathbf{x})$ defined by

$$z_i = \frac{\exp x_i}{\sum_{j=1}^{n_K} \exp x_j}$$

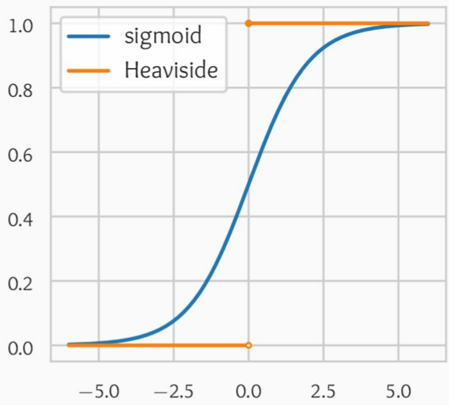
- $\mathrm{softmax}\,(\mathbf{x} + c\mathbb{1}) = \mathrm{softmax}\,(\mathbf{x})$, with $c \in \mathbb{R}$
- If $\mathbf{x}_i \geqslant \mathbf{x}_j$ then $\mathrm{softmax}\,(\mathbf{x})_i \geqslant \mathrm{softmax}\,(\mathbf{x})_j$
- Really an "arg softmax" rather than a "softmax"

Activation functions

Logistic function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

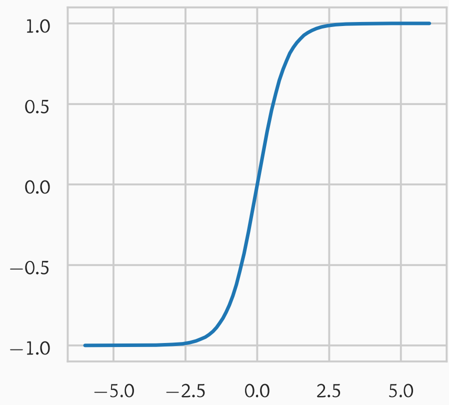
- Saturates at 0 and 1 when $x \to \pm\infty$
- Smooth version of Heaviside function
- $\sigma'(x) = \sigma(x)(1 - \sigma(x))$
- Killing gradients



Hyperbolic tangent

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

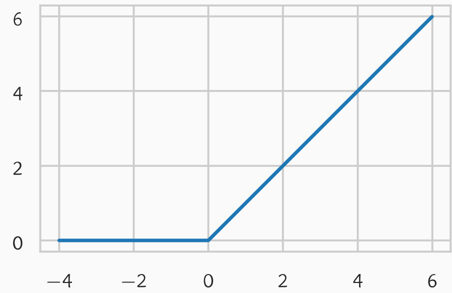
- Symmetric sigmoid
- Saturates at ± 1 when $x \to \pm\infty$
- $\tanh'(x) = 1 + \tanh^2(x)$
- Linearly related to the sigmoid function by
  $$\tanh(x) = 2\sigma(2x) - 1$$
- Killing gradients



Rectified linear unit (ReLU)

$$\mathrm{ReLU}(x) = \max(x, 0)$$

- Learns faster that sigmoid-like activation function
- Simpler to compute
- Provide sparsity of activations
- Dead ReLU: never activated across whole training set.
- Non negative activation function: zig-zag learning

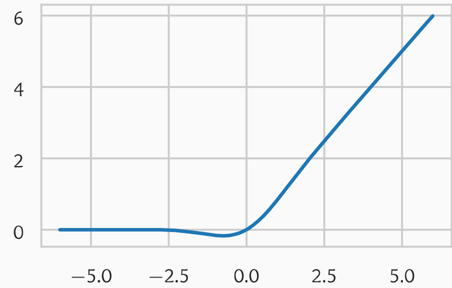


Gaussian Error Linear Unit (GELU), see Hendrycks and Gimpel 2023

$$\mathrm{GELU}(x) = x\Phi(x)$$

where Φ is the standard Gaussian cumulative distribution function

- Smoother version of ReLU
- Mostly used in transformers



Summary

- Use ReLU
- Sigmoid not used anymore
- Prefer hyperbolic tangent to sigmoid

Gradient descent algorithms

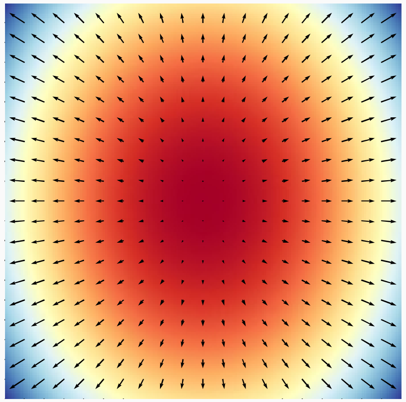
Gradient of a (scalar) function

- Differentiable function

$$f(x,y) = 1.5x^2 + y^2 + 2$$

- The gradient

$$\nabla_{(x,y)} f = \begin{pmatrix} 3x \\ 2y \end{pmatrix}$$



Gradient descent: motivation I

Gradient descent step improves current solution:

- Suppose $\mathcal{L}$ is a differentiable function we want to minimize

$$\arg\min_{\boldsymbol{\theta} \in \Theta} \mathcal{L}(\boldsymbol{\theta})$$

- Starting from the first order Taylor expansion. For a small $\|\mathbf{h}\|$ we have

$$\mathcal{L}(\boldsymbol{\theta} + \mathbf{h}) \approx \mathcal{L}(\boldsymbol{\theta}) + \langle \nabla_{\boldsymbol{\theta}} \mathcal{L}, \mathbf{h} \rangle$$

- Choose $\mathbf{h} = -\eta \nabla_{\boldsymbol{\theta}} \mathcal{L}$ (the gradient descent step), we have

$$\mathcal{L}(\boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}) \approx \mathcal{L}(\boldsymbol{\theta}) - \eta \left\|\nabla_{\boldsymbol{\theta}} \mathcal{L}\right\|^2 < \mathcal{L}(\boldsymbol{\theta})$$

- $\boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}$ is better than $\boldsymbol{\theta}$

Gradient descent: motivation II

Gradient descent step yields best update of linearized and regularized objective function

- Looking for the best $\boldsymbol{\theta}' = \boldsymbol{\theta} + \mathbf{h}$ around fixed $\boldsymbol{\theta}$
- Instead of minimizing $\mathcal{L}(\boldsymbol{\theta}')$, we minimize

$$\mathcal{L}(\boldsymbol{\theta}') \approx \mathcal{L}(\boldsymbol{\theta}) + \langle \mathbf{h}, \nabla_{\boldsymbol{\theta}} \mathcal{L} \rangle \tag{1}$$

- $\boldsymbol{\theta}'$ should stay close to $\boldsymbol{\theta}$ for (1) to hold, we penalize by $\|\boldsymbol{\theta} - \boldsymbol{\theta}'\| = \mathbf{h}$

$$\mathcal{L}(\boldsymbol{\theta}) + \langle \mathbf{h}, \nabla_{\boldsymbol{\theta}} \mathcal{L} \rangle + \frac{1}{2\eta} \|\mathbf{h}\|^2$$

- Minimizing w.r.t $\mathbf{h}$ gives

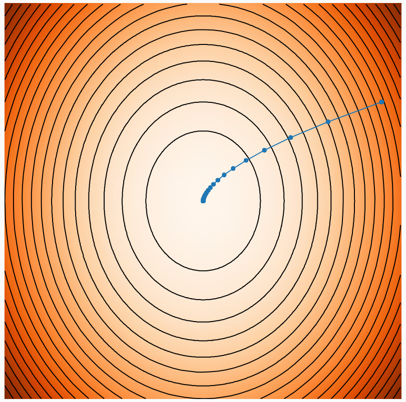
$$\boldsymbol{\theta}' = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}$$

Gradient descent algorithm

- Starting point $\boldsymbol{\theta}_0$
- Gradient descent step

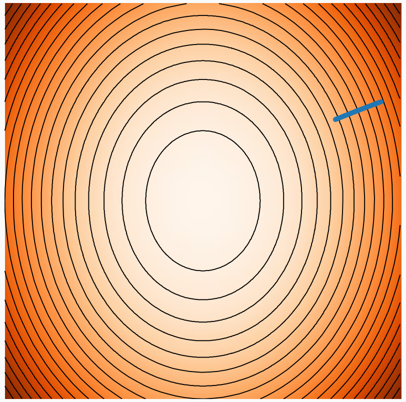
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}_t} \mathcal{L}$$

- η is the *learning rate*
- Learning rate too small: slow convergence
- Learning rate too high: fluctuate around minimum or even diverge

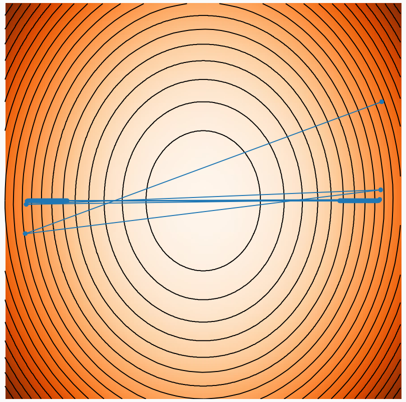


(a) $\eta = 0.1$

Gradient descent: illustration



(a) $\eta = 0.001$



(b) $\eta = 0.665757$

Backpropagation algorithm

Backpropagation

- Chain rule

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

- Generalization to any number of functions ($F = f_3 \circ f_2 \circ f1$)

$$\begin{aligned} F'(\theta) = (f_3 \circ f_2 \circ f_1)'(\theta) &= (f_3' \circ f_2 \circ f_1)(\theta) \cdot (f_2' \circ f_1)(\theta) \cdot f_1'(\theta) \\ &= f_3'((f_2 \circ f_1)(\theta)) \cdot f_2'(f_1(\theta)) \cdot f_1'(\theta) \end{aligned}$$

Backpropagation

- Chain rule

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

- Generalization to any number of functions ($F = f_3 \circ f_2 \circ f1$)

$$\begin{aligned} F'(\theta) = (f_3 \circ f_2 \circ f_1)'(\theta) &= (f_3' \circ f_2 \circ f_1)(\theta) \cdot (f_2' \circ f_1)(\theta) \cdot f_1'(\theta) \\ &= f_3'((f_2 \circ f_1)(\theta)) \cdot f_2'(f_1(\theta)) \cdot f_1'(\theta) \end{aligned}$$

- Forward and backward pass

$$\theta \xrightarrow{\quad f_1 \quad} f_1(\theta)$$

Backpropagation

- Chain rule

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

- Generalization to any number of functions ($F = f_3 \circ f_2 \circ f1$)

$$\begin{aligned} F'(\theta) = (f_3 \circ f_2 \circ f_1)'(\theta) &= (f_3' \circ f_2 \circ f_1)(\theta) \cdot (f_2' \circ f_1)(\theta) \cdot f_1'(\theta) \\ &= f_3'((f_2 \circ f_1)(\theta)) \cdot f_2'(f_1(\theta)) \cdot f_1'(\theta) \end{aligned}$$

- Forward and backward pass

$$\theta \xrightarrow{\quad f_1 \quad} f_1(\theta) \xrightarrow{\quad f_2 \quad} f_2(f_1(\theta))$$

Backpropagation

- Chain rule

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

- Generalization to any number of functions ($F = f_3 \circ f_2 \circ f1$)

$$\begin{aligned} F'(\theta) = (f_3 \circ f_2 \circ f_1)'(\theta) &= (f_3' \circ f_2 \circ f_1)(\theta) \cdot (f_2' \circ f_1)(\theta) \cdot f_1'(\theta) \\ &= f_3'((f_2 \circ f_1)(\theta)) \cdot f_2'(f_1(\theta)) \cdot f_1'(\theta) \end{aligned}$$

- Forward and backward pass

$$\theta \xrightarrow{\quad f_1 \quad} f_1(\theta) \xrightarrow{\quad f_2 \quad} f_2(f_1(\theta)) \xrightarrow{f_3} f_3(f_2(f_1(\theta))) = F(\theta)$$

Backpropagation

- Chain rule

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

- Generalization to any number of functions $(F = f_3 \circ f_2 \circ f1)$

$$\begin{aligned} F'(\theta) = (f_3 \circ f_2 \circ f_1)'(\theta) &= (f_3' \circ f_2 \circ f_1)(\theta) \cdot (f_2' \circ f_1)(\theta) \cdot f_1'(\theta) \\ &= f_3'((f_2 \circ f_1)(\theta)) \cdot f_2'(f_1(\theta)) \cdot f_1'(\theta) \end{aligned}$$

- Forward and backward pass

$$\begin{tikzcd} \theta \arrow[r, "f_1"] & f_1(\theta) \arrow[r, "f_2"] & f_2(f_1(\theta)) \arrow[r, "f_3"] \arrow[d] & f_3(f_2(f_1(\theta))) = F(\theta) \\ & & f_3'(f_2(f_1(\theta))) & \end{tikzcd}$$

Backpropagation

- Chain rule

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

- Generalization to any number of functions $(F = f_3 \circ f_2 \circ f1)$

$$\begin{aligned} F'(\theta) = (f_3 \circ f_2 \circ f_1)'(\theta) &= (f_3' \circ f_2 \circ f_1)(\theta) \cdot (f_2' \circ f_1)(\theta) \cdot f_1'(\theta) \\ &= f_3'((f_2 \circ f_1)(\theta)) \cdot f_2'(f_1(\theta)) \cdot f_1'(\theta) \end{aligned}$$

- Forward and backward pass

$$\begin{tikzcd} \theta \arrow[r, "f_1"] & f_1(\theta) \arrow[r, "f_2"] \arrow[d] & f_2(f_1(\theta)) \arrow[r, "f_3"] \arrow[d] & f_3(f_2(f_1(\theta))) = F(\theta) \\ & f_2'(f_1(\theta)) & f_3'(f_2(f_1(\theta))) \arrow[l, "\times"'] & \end{tikzcd}$$

Backpropagation

- Chain rule

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

- Generalization to any number of functions $(F = f_3 \circ f_2 \circ f1)$

$$\begin{aligned} F'(\theta) = (f_3 \circ f_2 \circ f_1)'(\theta) &= (f_3' \circ f_2 \circ f_1)(\theta) \cdot (f_2' \circ f_1)(\theta) \cdot f_1'(\theta) \\ &= f_3'((f_2 \circ f_1)(\theta)) \cdot f_2'(f_1(\theta)) \cdot f_1'(\theta) \end{aligned}$$

- Forward and backward pass

$$\begin{tikzcd} \theta \arrow[r, "f_1"] \arrow[d] & f_1(\theta) \arrow[r, "f_2"] \arrow[d] & f_2(f_1(\theta)) \arrow[r, "f_3"] \arrow[d] & f_3(f_2(f_1(\theta))) = F(\theta) \\ F'(\theta) = f_1'(\theta) & f_2'(f_1(\theta)) \arrow[l, "\times"'] & f_3'(f_2(f_1(\theta))) \arrow[l, "\times"'] & \end{tikzcd}$$

Backpropagation

- Generalizable to $\mathbb{R}^n$ to $\mathbb{R}^p$ functions using jacobians
- Generalizable to functions having their own set of parameters
- Generalizable to a computational graph (DAG)

Backpropagating the loss

Total loss on the minibatch $\mathcal{B}$ (whole training set for now)

$$\mathcal{L}_{\mathcal{B}} = \frac{1}{|\mathcal{B}|} \sum_{(\mathbf{x},\boldsymbol{y}) \in \mathcal{B}} \ell(F_\Theta(\mathbf{x}), \boldsymbol{y})$$

Differentiating w.r.t any scalar parameter θ (any $\boldsymbol{w}_{ij}^{(k)}$ or $\boldsymbol{b}_j^{(k)}$)

$$\frac{\partial \mathcal{L}_{\mathcal{B}}}{\partial \theta} = \frac{1}{|\mathcal{B}|} \sum_{(\mathbf{x},\boldsymbol{y}) \in \mathcal{B}} \frac{\partial \ell(\mathbf{x}^{(K)}, \boldsymbol{y})}{\partial \theta} \quad \text{with } \mathbf{x}^{(K)} = \widehat{\boldsymbol{y}} = F_\Theta(\mathbf{x})$$

It suffices to compute

$$\frac{\partial \ell(\mathbf{x}^{(K)}, \boldsymbol{y})}{\partial \theta}$$

MSE loss

- MSE loss is defined by

$$\ell\left(\mathbf{x}^{(K)}, \boldsymbol{y}\right) = \frac{1}{n_K} \sum_{i=1}^{n_K} \left(\mathbf{x}_i^{(K)} - \boldsymbol{y}_i\right)^2$$

- Differentiating with respect to some scalar parameter $\theta = \boldsymbol{b}_q^{(k)}$ or $\theta = \boldsymbol{w}_{pq}^{(k)}$
- ($\mathbf{x}_i^{(K)}$ and $\boldsymbol{y}_i$ are scalars)

$$\begin{aligned} \frac{\partial \ell(\mathbf{x}^{(K)}, \boldsymbol{y})}{\partial \theta} &= \frac{1}{n_K} \sum_{i=1}^{n_K} \frac{\partial \left(\mathbf{x}_i^{(K)} - \boldsymbol{y}_i\right)^2}{\partial \mathbf{x}_i^{(K)}} \cdot \frac{\partial \mathbf{x}_i^{(K)}}{\partial \theta} \\ &= \frac{2}{n_K} \sum_{i=1}^{n_K} \left(\mathbf{x}_i^{(K)} - \boldsymbol{y}_i\right) \cdot \frac{\partial \mathbf{x}_i^{(K)}}{\partial \theta} \end{aligned}$$

- We need to know $\dfrac{\partial \mathbf{x}_i^{(K)}}{\partial \theta}$ now!

Cross entropy loss

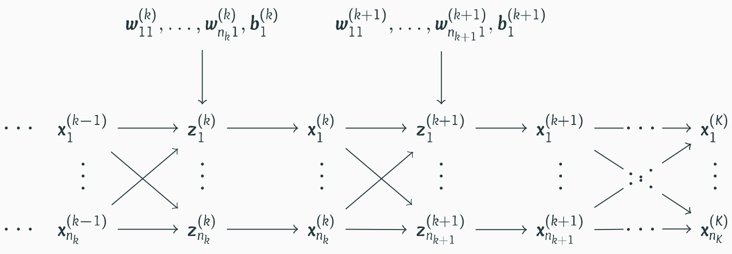
- Cross entropy loss: $\ell\left(\mathbf{x}^{(K)}, \boldsymbol{y}\right) = -\log \sigma_y\left(\mathbf{x}^{(K)}\right)$
- Differentiating with respect to some scalar parameter $\theta = \boldsymbol{b}_q^{(k)}$ or $\theta = \boldsymbol{w}_{pq}^{(k)}$

$$\begin{aligned} \frac{\partial \ell(\mathbf{x}^{(K)}, \boldsymbol{y})}{\partial \theta} &= -\sum_{i=1}^{n_K} \frac{\partial \log \sigma_y\left(\mathbf{x}^{(K)}\right)}{\partial \mathbf{x}_i^{(K)}} \cdot \frac{\partial \mathbf{x}_i^{(K)}}{\partial \theta} \\ &= -\sum_{i=1}^{n_K} (\boldsymbol{y}_i - \sigma_i(\mathbf{x})) \frac{\partial \mathbf{x}_i^{(K)}}{\partial \theta} \end{aligned}$$

- We need to know $\dfrac{\partial \mathbf{x}_i^{(K)}}{\partial \theta}$ now!

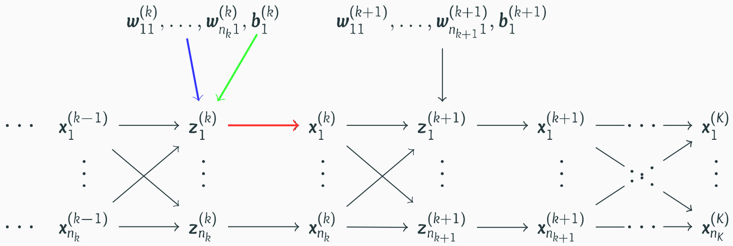
Computational graph

- Define $\mathbf{z}_i^{(k)} = \left\langle \boldsymbol{w}_i^{(k)}, \mathbf{x}^{(k-1)} \right\rangle + \boldsymbol{b}_i^{(k)}$ so that we have $\mathbf{x}_i^{(k)} = \sigma\left(\mathbf{z}_i^{(k)}\right)$
- Computational graph for $\mathbf{x}_1^{(k-1)}$, $\mathbf{x}_1^{(k)}$, $\mathbf{x}_1^{(k+1)}$ only!



Gradient of last layer w.r.t parameters I

Computational graph for neuron 1 at layer k ($p = 1$):



$$\frac{\partial \mathbf{x}_i^{(K)}}{\partial \boldsymbol{w}_{qp}^{(k)}} = \frac{\partial \mathbf{x}_i^{(K)}}{\partial \mathbf{x}_p^{(k)}} \cdot \frac{\partial \mathbf{x}_p^{(k)}}{\partial \mathbf{z}_p^{(k)}} \cdot \frac{\partial \mathbf{z}_p^{(k)}}{\partial \boldsymbol{w}_{qp}^{(k)}} \qquad \frac{\partial \mathbf{x}_i^{(K)}}{\partial \boldsymbol{b}_p^{(k)}} = \frac{\partial \mathbf{x}_i^{(K)}}{\partial \mathbf{x}_p^{(k)}} \cdot \frac{\partial \mathbf{x}_p^{(k)}}{\partial \mathbf{z}_p^{(k)}} \cdot \frac{\partial \mathbf{z}_p^{(k)}}{\partial \boldsymbol{b}_p^{(k)}}$$

Gradient of last layer w.r.t parameters II

Given that $\mathbf{x}_p^{(k)} = \sigma\left(\mathbf{z}_p^{(k)}\right)$, we have $\dfrac{\partial \mathbf{x}_p^{(k)}}{\partial \mathbf{z}_p^{(k)}} = \sigma'\left(\mathbf{z}_p^{(k)}\right)$

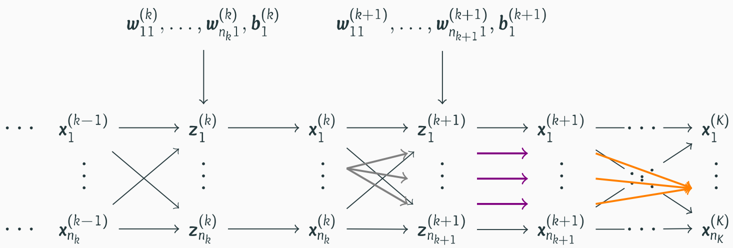
And $\mathbf{z}_p^{(k)} = \left\langle \boldsymbol{w}_p^{(k)}, \mathbf{x}^{(k-1)} \right\rangle + \boldsymbol{b}_p^{(k)}$, so $\dfrac{\partial \mathbf{z}_p^{(k)}}{\partial \boldsymbol{w}_{qp}^{(k)}} = \mathbf{x}_q^{(k-1)}$ and $\dfrac{\partial \mathbf{z}_p^{(k)}}{\partial \boldsymbol{b}_p^{(k)}} = 1$

Finally

$$\frac{\partial \mathbf{x}_i^{(K)}}{\partial \boldsymbol{w}_{qp}^{(k)}} = \frac{\partial \mathbf{x}_i^{(K)}}{\partial \mathbf{x}_p^{(k)}} \cdot \sigma'\left(\mathbf{z}_p^{(k)}\right) \cdot \mathbf{x}_q^{(k-1)} \qquad \frac{\partial \mathbf{x}_i^{(K)}}{\partial \boldsymbol{b}_p^{(k)}} = \frac{\partial \mathbf{x}_i^{(K)}}{\partial \mathbf{x}_p^{(k)}} \cdot \sigma'\left(\mathbf{z}_p^{(k)}\right)$$

Need to compute $\dfrac{\partial \mathbf{x}_i^{(K)}}{\partial \mathbf{x}_p^{(k)}}$ now!

Gradient of last layer w.r.t other layer



$$\frac{\partial \mathbf{x}_i^{(K)}}{\partial \mathbf{x}_p^{(k)}} = \sum_{j=1}^{n_{k+1}} \frac{\partial \mathbf{x}_i^{(K)}}{\partial \mathbf{x}_j^{(k+1)}} \cdot \frac{\partial \mathbf{x}_j^{(k+1)}}{\partial \mathbf{z}_j^{(k+1)}} \cdot \frac{\partial \mathbf{z}_j^{(k+1)}}{\partial \mathbf{x}_p^{(k)}}$$

Backpropagating from next layer

Given that $\mathbf{x}_j^{(k+1)} = \sigma\left(\mathbf{z}_j^{(k+1)}\right)$, we have

$$\frac{\partial \mathbf{x}_j^{(k+1)}}{\partial \mathbf{z}_j^{(k+1)}} = \sigma'\left(\mathbf{z}_j^{(k+1)}\right)$$

And $\mathbf{z}_j^{(k+1)} = \left\langle \boldsymbol{w}_j^{(k+1)}, \mathbf{x}^{(k)} \right\rangle + \boldsymbol{b}_j^{(k+1)}$, so

$$\frac{\partial \mathbf{z}_j^{(k+1)}}{\partial \mathbf{x}_p^{(k)}} = \boldsymbol{w}_{pj}^{(k+1)}$$

Replacing in last equation, we have

$$\frac{\partial \mathbf{x}_i^{(K)}}{\partial \mathbf{x}_p^{(k)}} = \sum_{j=1}^{n_{k+1}} \frac{\partial \mathbf{x}_i^{(K)}}{\partial \mathbf{x}_j^{(k+1)}} \cdot \sigma'\left(\mathbf{z}_j^{(k+1)}\right) \cdot \boldsymbol{w}_{pj}^{(k+1)}$$

Backpropagation equations

- Backpropagation equations for parameters $\boldsymbol{w}_{qp}^{(k)}$ and $\boldsymbol{b}_{p}^{(k)}$

$$\frac{\partial \mathbf{x}_i^{(K)}}{\partial \boldsymbol{w}_{qp}^{(k)}} = \frac{\partial \mathbf{x}_i^{(K)}}{\partial \mathbf{x}_p^{(k)}} \cdot \sigma'\left(\mathbf{z}_p^{(k)}\right) \cdot \mathbf{x}_q^{(k-1)} \quad (2)$$

$$\frac{\partial \mathbf{x}_i^{(K)}}{\partial \boldsymbol{b}_{p}^{(k)}} = \frac{\partial \mathbf{x}_i^{(K)}}{\partial \mathbf{x}_p^{(k)}} \cdot \sigma'\left(\mathbf{z}_p^{(k)}\right) \quad (3)$$

- Backpropagation equation

$$\frac{\partial \mathbf{x}_i^{(K)}}{\partial \mathbf{x}_p^{(k)}} = \sum_{j=1}^{n_{k+1}} \frac{\partial \mathbf{x}_i^{(K)}}{\partial \mathbf{x}_j^{(k+1)}} \cdot \sigma'\left(\mathbf{z}_j^{(k+1)}\right) \cdot \boldsymbol{w}_{pj}^{(k+1)} \quad (4)$$

- Only one sweep backward is necessary to compute all gradients

References i

[1] Sergey Ioffe and Christian Szegedy. **"Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift".** Feb. 10, 2015. arXiv: `1502.03167 [cs]`. URL: `http://arxiv.org/abs/1502.03167` (visited on 11/06/2017).

[2] Ian Goodfellow et al. ***Deep Learning.*** Vol. 1. MIT press Cambridge, 2016.

[3] Aston Zhang et al. ***Dive into Deep Learning.*** 2020.

[4] Dan Hendrycks and Kevin Gimpel. ***Gaussian Error Linear Units (GELUs).*** June 5, 2023. DOI: `10.48550/arXiv.1606.08415`. arXiv: `1606.08415 [cs]`. URL: `http://arxiv.org/abs/1606.08415` (visited on 11/07/2023). Pre-published.