Deep Learning

Gradient-based optimization

Caio Corro

Université Paris-Saclay

Table of contents

Recall: neural networks

Neural network

- $\boldsymbol{x}$: input features
- $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}$: hidden representation
- $\boldsymbol{w}$: output logits or class weights
- $\boldsymbol{p}$: probability distribution over classes
- $\theta = \{\boldsymbol{A}^{(1)}, \boldsymbol{b}^{(1)}, ...\}$: parameters
- σ: non-linear activation function

Neural network

- $\boldsymbol{x}$: input features
- $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}$: hidden representation
- $\boldsymbol{w}$: output logits or class weights
- $\boldsymbol{p}$: probability distribution over classes
- $\theta = \{\boldsymbol{A}^{(1)}, \boldsymbol{b}^{(1)}, ...\}$: parameters
- σ: non-linear activation function

$$\boldsymbol{z}^{(1)} = \sigma\left(\boldsymbol{A}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right)$$

$$\boldsymbol{z}^{(2)} = \sigma\left(\boldsymbol{A}^{(2)}\boldsymbol{z}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\boldsymbol{w} = \sigma\left(\boldsymbol{A}^{(3)}\boldsymbol{z}^{(2)} + \boldsymbol{b}^{(3)}\right)$$

$$\boldsymbol{p} = \text{Softmax}(\boldsymbol{w}) \quad \text{i.e.} \quad p_i = \frac{\exp(\boldsymbol{w}_i)}{\sum_j \exp(\boldsymbol{w}_j)}$$

Neural network

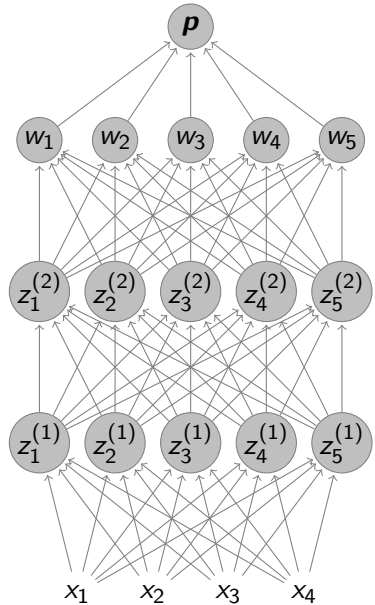
- $\boldsymbol{x}$: input features
- $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}$: hidden representation
- $\boldsymbol{w}$: output logits or class weights
- $\boldsymbol{p}$: probability distribution over classes
- $\theta = \{\boldsymbol{A}^{(1)}, \boldsymbol{b}^{(1)}, ...\}$: parameters
- σ: non-linear activation function

$$\boldsymbol{z}^{(1)} = \sigma\left(\boldsymbol{A}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right)$$

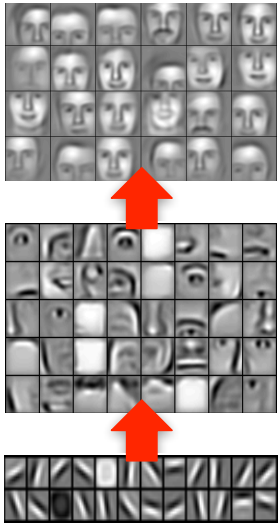
$$\boldsymbol{z}^{(2)} = \sigma\left(\boldsymbol{A}^{(2)}\boldsymbol{z}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\boldsymbol{w} = \sigma\left(\boldsymbol{A}^{(3)}\boldsymbol{z}^{(2)} + \boldsymbol{b}^{(3)}\right)$$

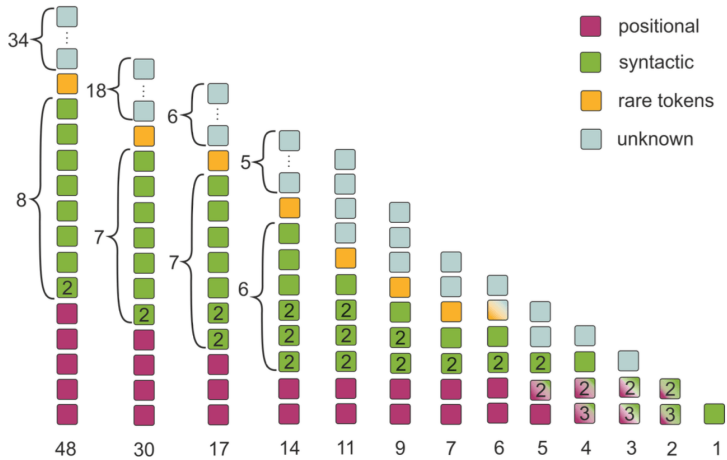
$$\boldsymbol{p} = \text{Softmax}(\boldsymbol{w}) \quad \text{i.e.} \quad p_i = \frac{\exp(\boldsymbol{w}_i)}{\sum_j \exp(\boldsymbol{w}_j)}$$



Representation learning: Computer Vision [Lee et al., 2009]



Representation learning: Natural Language Processing [Voita et al., 2019]



The training loop

The big picture

Data split and usage

- Training set: to learn the parameters of the network
- Development (or dev or validation) set: to monitor the network during training
- Test set: to evaluate our model at the end

Generally you don't have to split the data yourself: there exists standard splits to allow benchmarking.

The big picture

Data split and usage

- Training set: to learn the parameters of the network
- Development (or dev or validation) set: to monitor the network during training
- Test set: to evaluate our model at the end

Generally you don't have to split the data yourself: there exists standard splits to allow benchmarking.

Training loop

1. Update the parameters to minimize the loss on the training set
2. Evaluate the prediction accuracy on the dev set
3. If not satisfied, go back to 1
4. Evaluate the prediction accuracy on the test set with the best parameters on dev

Pseudo-code

function $\textsc{Train}(f, \theta, \mathcal{T}, \mathcal{D})$

Pseudo-code

function $\textsc{Train}(f, \theta, \mathcal{T}, \mathcal{D})$
  $\texttt{bestdev} = -\infty$
  for $\texttt{epoch} = 1$ to E **do**
    Shuffle $\mathcal{T}$
    for $x, y \in \mathcal{T}$ **do**
      $\texttt{loss} = \mathcal{L}(f(x; \theta), y)$
      $\theta = \theta - \epsilon \nabla \texttt{loss}$

Pseudo-code

function $\textsc{Train}(f, \theta, \mathcal{T}, \mathcal{D})$
  $\texttt{bestdev} = -\infty$
  for $\texttt{epoch} = 1$ to E **do**
    Shuffle $\mathcal{T}$
    for $x, y \in \mathcal{T}$ **do**
      $\texttt{loss} = \mathcal{L}(f(x; \theta), y)$
      $\theta = \theta - \epsilon \nabla \texttt{loss}$

    $\texttt{devacc} = \textsc{Evaluate}(f, \mathcal{D})$
    if $\texttt{devacc} > \texttt{bestdev}$ **then**
      $\hat{\theta} = \theta$
      $\texttt{bestdev} = \texttt{devacc}$

  return $\hat{\theta}$

Pseudo-code

function TRAIN($f, \theta, \mathcal{T}, \mathcal{D}$)
  `bestdev` $= -\infty$
  for `epoch` $= 1$ to E **do**
    Shuffle $\mathcal{T}$
    for $x, y \in \mathcal{T}$ **do**
      `loss` $= \mathcal{L}(f(x;\theta), y)$
      $\theta = \theta - \epsilon \nabla$`loss`

    `devacc` $=$EVALUATE($f, \mathcal{D}$)
    if `devacc` $>$ `bestdev` **then**
      $\hat{\theta} = \theta$
      `bestdev` $=$ `devacc`

  return $\hat{\theta}$

function EVALUATE($f, \mathcal{D}$)
  $n = 0$
  for $x, y \in \mathcal{D}$ **do**
    $\hat{y} = \arg\max_y f(x;\theta)_y$
    if $\hat{y} = y$ **then**
      $n = n + 1$
  return $n/|\mathcal{D}|$

Further details

Sampling without replacement

- shuffle the training set
- loop over the new order

Experimentally it works better than "true" sampling and it seems to also have good theoretical properties [Nagaraj et al., 2019]

Verbosity

At each epoch, it is useful to display:

- mean loss
- accuracy on training data
- accuracy on dev data
- timing information
- (sometimes) evaluate on dev several times by epoch

Step-size

$$\theta^{(t+1)} = \theta^{(t)} - \epsilon^{(t)} \nabla \texttt{loss} \qquad \Rightarrow \qquad \text{How to choose the step size } \epsilon^{(t+1)}?$$

Step-size

$$\theta^{(t+1)} = \theta^{(t)} - \epsilon^{(t)} \nabla \texttt{loss} \qquad \Rightarrow \qquad \text{How to choose the step size } \epsilon^{(t+1)}?$$

Convex optimization

- Nonsummable diminishing step size: $\sum_{t=1}^{\infty} \epsilon^{(t)} = \infty$ and $\lim_{t \to \infty} \epsilon^{(t)} = 0$
- Backtracking/exact line search

Step-size

$$\theta^{(t+1)} = \theta^{(t)} - \epsilon^{(t)} \nabla \texttt{loss} \qquad \Rightarrow \qquad \text{How to choose the step size } \epsilon^{(t+1)}?$$

Convex optimization

- Nonsummable diminishing step size: $\sum_{t=1}^{\infty} \epsilon^{(t)} = \infty$ and $\lim_{t \to \infty} \epsilon^{(t)} = 0$
- Backtracking/exact line search

Simple neural network heuristic

1. Start with a small value, e.g. $\epsilon = 0.01$
2. If dev accuracy did not improve during the last N epochs:
   decay the learning rate by a small value α, e.g. $\epsilon = \alpha * \epsilon$ with $\alpha = 0.1$

Step-size

$\theta^{(t+1)} = \theta^{(t)} - \epsilon^{(t)} \nabla \texttt{loss} \qquad \Rightarrow \qquad$ How to choose the step size $\epsilon^{(t+1)}$?

Convex optimization

- Nonsummable diminishing step size: $\sum_{t=1}^{\infty} \epsilon^{(t)} = \infty$ and $\lim_{t \to \infty} \epsilon^{(t)} = 0$
- Backtracking/exact line search

Simple neural network heuristic

1. Start with a small value, e.g. $\epsilon = 0.01$
2. If dev accuracy did not improve during the last N epochs:
   decay the learning rate by a small value α, e.g. $\epsilon = \alpha * \epsilon$ with $\alpha = 0.1$

Step-size annealing

- Step decay: multiple ϵ by $\alpha \in [0, 1]$ every N epochs
- Exponential decay: $\epsilon^{(t)} = \epsilon^{(0)} \exp(-\alpha \cdot t)$
- $1/t$ decay: $\epsilon^{(t)} = \frac{\epsilon^{(0)}}{1 + \alpha \cdot t}$

Backpropagation

Scalar input

Derivative

Let $f : \mathbb{R} \to \mathbb{R}$ be a function and $x, y \in \mathbb{R}$ be variables such that:

$$y = f(x).$$

For a given x, how does an infinitesimal change of x impact y?

Scalar input

Derivative

Let $f : \mathbb{R} \to \mathbb{R}$ be a function and $x, y \in \mathbb{R}$ be variables such that:

$$y = f(x).$$

For a given x, how does an infinitesimal change of x impact y?

$$\frac{dy}{dx} = f'(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

Scalar input

Derivative

Let $f : \mathbb{R} \to \mathbb{R}$ be a function and $x, y \in \mathbb{R}$ be variables such that:

$$y = f(x).$$

For a given x, how does an infinitesimal change of x impact y?

$$\frac{dy}{dx} = f'(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

Linear approximation

Let $\widetilde{f} : \mathcal{R} \to \mathcal{R}$ be function parameterized by $a \in \mathcal{R}$ defined as follows:

$$\widetilde{f}(x; a) = f(a) + f'(a) \cdot (x - a)$$

Then, $\widetilde{f}(x; a)$ is an approximation of f at a.

Scalar input

Example

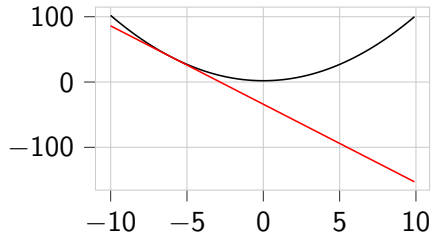
$$
\begin{aligned}
f(x) &= x^2 + 2 \\
f'(x) &= 2x \\
\widetilde{f}(x; a) &= f(a) + f'(a) \cdot (x - a) \\
&= a^2 + 2 + 2a(x - a) \\
&= 2ax + 2 - a^2
\end{aligned}
$$

Scalar input

Example

$$
\begin{aligned}
f(x) &= x^2 + 2 \\
f'(x) &= 2x \\
\widetilde{f}(x; a) &= f(a) + f'(a) \cdot (x - a) \\
&= a^2 + 2 + 2a(x - a) \\
&= 2ax + 2 - a^2
\end{aligned}
$$

Intuition: the sign of $f'(a)$ gives the slope of the approximation, we can use this information to move closer to the minimum of $f(x)$.



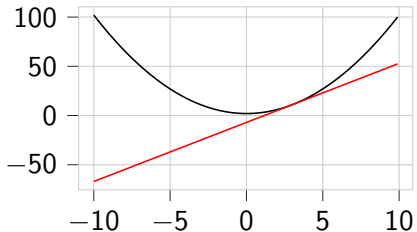
- $a = -6$
- Black: $f(x)$
- Red: $\widetilde{f}(x; a = -6)$

Scalar input

Example

$$
\begin{aligned}
f(x) &= x^2 + 2 \\
f'(x) &= 2x \\
\widetilde{f}(x; a) &= f(a) + f'(a) \cdot (x - a) \\
&= a^2 + 2 + 2a(x - a) \\
&= 2ax + 2 - a^2
\end{aligned}
$$

Intuition: the sign of $f'(a)$ gives the slope of the approximation, we can use this information to move closer to the minimum of $f(x)$.



- $a = 3$
- Black: $f(a)$
- Red: $\widetilde{f}(x; a = 3)$

Scalar input

Chain rule

Let $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$ be two functions and x, y, z be variables such that:

$$\begin{aligned} z &= f(x), \\ y &= g(z) \qquad\qquad \text{i.e. } y = g(f(x)) = g \circ f(x). \end{aligned}$$

For a given x, how does an infinitesimal change of x impact y?

Scalar input

Chain rule

Let $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$ be two functions and x, y, z be variables such that:

$$\begin{aligned} z &= f(x), \\ y &= g(z) \qquad\qquad \text{i.e. } y = g(f(x)) = g \circ f(x). \end{aligned}$$

For a given x, how does an infinitesimal change of x impact y?

$$\frac{dy}{dx} = \frac{dy}{dz} \cdot \frac{dz}{dx}$$

Scalar input

Example: explicit differentiation

$$f(x) = (2x + 1)^2$$

Scalar input

Example: explicit differentiation

$$f(x) = (2x + 1)^2 = 4x^2 + 4x + 1$$

Scalar input

Example: explicit differentiation

$$f(x) = (2x+1)^2 = 4x^2 + 4x + 1$$
$$f'(x) = 8x + 4$$

Scalar input

Example: explicit differentiation

$$f(x) = (2x+1)^2 = 4x^2 + 4x + 1$$
$$f'(x) = 8x + 4$$

Example: differentiation using the chain rule

$$z = 2x + 1$$

$$y = z^2 = f(x)$$

Scalar input

Example: explicit differentiation

$$f(x) = (2x+1)^2 = 4x^2 + 4x + 1$$
$$f'(x) = 8x + 4$$

Example: differentiation using the chain rule

$$z = 2x + 1 \qquad \frac{dz}{dx} = 2$$
$$y = z^2 = f(x) \qquad \frac{dy}{dz} = 2z$$

Scalar input

Example: explicit differentiation

$$f(x) = (2x+1)^2 = 4x^2 + 4x + 1$$
$$f'(x) = 8x + 4$$

Example: differentiation using the chain rule

$$z = 2x + 1 \qquad \frac{dz}{dx} = 2$$
$$y = z^2 = f(x) \qquad \frac{dy}{dz} = 2z$$
$$\frac{dy}{dx} = \frac{dy}{dz} \cdot \frac{dz}{dx}$$

Scalar input

Example: explicit differentiation

$$f(x) = (2x+1)^2 = 4x^2 + 4x + 1$$
$$f'(x) = 8x + 4$$

Example: differentiation using the chain rule

$$z = 2x + 1 \qquad \frac{dz}{dx} = 2$$
$$y = z^2 = f(x) \qquad \frac{dy}{dz} = 2z$$
$$\frac{dy}{dx} = \frac{dy}{dz} \cdot \frac{dz}{dx} = 2z * 2$$

Scalar input

Example: explicit differentiation

$$f(x) = (2x+1)^2 = 4x^2 + 4x + 1$$
$$f'(x) = 8x + 4$$

Example: differentiation using the chain rule

$$z = 2x + 1 \qquad \frac{dz}{dx} = 2$$
$$y = z^2 = f(x) \qquad \frac{dy}{dz} = 2z$$
$$\frac{dy}{dx} = \frac{dy}{dz} \cdot \frac{dz}{dx} = 2z * 2 = 4(2x+1)$$

Scalar input

Example: explicit differentiation

$$f(x) = (2x+1)^2 = 4x^2 + 4x + 1$$
$$f'(x) = 8x + 4$$

Example: differentiation using the chain rule

$$z = 2x + 1 \qquad \frac{dz}{dx} = 2$$
$$y = z^2 = f(x) \qquad \frac{dy}{dz} = 2z$$
$$\frac{dy}{dx} = \frac{dy}{dz} \cdot \frac{dz}{dx} = 2z * 2 = 4(2x+1) = 8x + 4 = f'(x)$$

Vector input

Let $f : \mathbb{R}^m \to \mathbb{R}$ be a function and $\boldsymbol{x} \in \mathbb{R}^m, y \in \mathbb{R}$ be variables such that:

$$y = f(\boldsymbol{x}).$$

Vector input

Let $f : \mathbb{R}^m \to \mathbb{R}$ be a function and $\boldsymbol{x} \in \mathbb{R}^m, y \in \mathbb{R}$ be variables such that:

$$y = f(\boldsymbol{x}).$$

Partial derivative

For a given $\boldsymbol{x}$, how does an infinitesimal change of x_i impact y?

$$\frac{\partial y}{\partial x_i}$$

i.e. each input $x_j, j \neq i$ is considered as a constant.

Vector input

Let $f : \mathbb{R}^m \to \mathbb{R}$ be a function and $\boldsymbol{x} \in \mathbb{R}^m, y \in \mathbb{R}$ be variables such that:

$$y = f(\boldsymbol{x}).$$

Partial derivative

For a given $\boldsymbol{x}$, how does an infinitesimal change of x_i impact y?

$$\frac{\partial y}{\partial x_i}$$

i.e. each input $x_j, j \neq i$ is considered as a constant.

Gradient

For a given $\boldsymbol{x}$, how does an infinitesimal change of $\boldsymbol{x}$ impact y?

$$\nabla_{\boldsymbol{x}} y = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \cdots \end{bmatrix}$$

Vector input

Chain rule

Let $f : \mathbb{R}^m \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}$ be two functions and $\boldsymbol{x}^m, \boldsymbol{z}^n, y$ be variables such that:

$$\begin{aligned} \boldsymbol{z} &= f(\boldsymbol{x}), \\ y &= g(\boldsymbol{z}) \end{aligned}$$

For a given x_i, how does an infinitesimal change of x_i impact y?

Vector input

Chain rule

Let $f : \mathbb{R}^m \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}$ be two functions and $\boldsymbol{x}^m, \boldsymbol{z}^n, y$ be variables such that:

$$\begin{aligned} \boldsymbol{z} &= f(\boldsymbol{x}), \\ y &= g(\boldsymbol{z}) \end{aligned}$$

For a given x_i, how does an infinitesimal change of x_i impact y?

$$\frac{\partial y}{\partial x_i} = \sum_j \frac{\partial y}{\partial z_j} \cdot \frac{\partial z_j}{\partial x_i}$$

Vector example

$$\boldsymbol{z} = \boldsymbol{W}x + b \quad \text{or} \quad z_j = \sum_i W_{j,i} x_i + b_j$$

$$y = \sum_j z_j$$

Vector example

$$\boldsymbol{z} = \boldsymbol{W}x + b \quad \text{or} \quad z_j = \sum_i W_{j,i} x_i + b_j \qquad \frac{\partial z_j}{x_i} = W_{j,i}$$

$$y = \sum_j z_j \qquad \frac{\partial y}{z_j} = 1$$

Vector example

$$\boldsymbol{z} = \boldsymbol{W}x + b \quad \text{or} \quad z_j = \sum_i W_{j,i} x_i + b_j \qquad \frac{\partial z_j}{x_i} = W_{j,i}$$

$$y = \sum_j z_j \qquad \frac{\partial y}{z_j} = 1$$

$$\frac{\partial y}{\partial x_i} = \sum_j \frac{\partial y}{\partial z_j} \cdot \frac{\partial z_j}{\partial x_i} = \sum_j 1 * W_{j,i}$$

Vector example

$$\boldsymbol{z}^{(1)} = ...\boldsymbol{x}...$$
$$\boldsymbol{z}^{(2)} = ...\boldsymbol{z}^{(1)}...$$
$$y = ...\boldsymbol{z}^{(2)}...$$

$$\frac{\partial y}{\partial x_i}$$

Vector example

$$\begin{aligned} \boldsymbol{z}^{(1)} &= ...\boldsymbol{x}... \\ \boldsymbol{z}^{(2)} &= ...\boldsymbol{z}^{(1)}... \\ y &= ...\boldsymbol{z}^{(2)}... \end{aligned}$$

$$\frac{\partial y}{\partial x_i} = \sum_k \frac{\partial y}{\partial z_k^{(2)}} \cdot \frac{\partial z_k^{(2)}}{\partial x_i}$$

Vector example

$$\begin{aligned} \boldsymbol{z}^{(1)} &= ...\boldsymbol{x}... \\ \boldsymbol{z}^{(2)} &= ...\boldsymbol{z}^{(1)}... \\ y &= ...\boldsymbol{z}^{(2)}... \end{aligned}$$

$$\frac{\partial y}{\partial x_i} = \sum_k \frac{\partial y}{\partial z_k^{(2)}} \cdot \frac{\partial z_k^{(2)}}{\partial x_i} = \sum_k \frac{\partial y}{\partial z_k^{(2)}} \cdot \sum_j \frac{\partial z_k^{(2)}}{\partial z_j^{(1)}} \cdot \frac{\partial z_j^{(1)}}{\partial x_i}$$

$\Rightarrow$ **It is starting to get annoying!**

Jacobian

Let $f : \mathbb{R}^m \to \mathbb{R}^n$ be a function and $\boldsymbol{x} \in \mathbb{R}^m, \boldsymbol{y} \in \mathbb{R}^n$ be variables such that:

$$\boldsymbol{y} = f(\boldsymbol{x}).$$

Jacobian

Let $f : \mathbb{R}^m \to \mathbb{R}^n$ be a function and $\boldsymbol{x} \in \mathbb{R}^m, \boldsymbol{y} \in \mathbb{R}^n$ be variables such that:

$$\boldsymbol{y} = f(\boldsymbol{x}).$$

Gradient

For a given $\boldsymbol{x}$, how does an infinitesimal change of $\boldsymbol{x}$ impact y_j?

$$\nabla_{\boldsymbol{x}} y_j = \begin{bmatrix} \frac{\partial y_j}{\partial x_1} \\ \frac{\partial y_j}{\partial x_2} \\ \dots \end{bmatrix}$$

Jacobian

Let $f : \mathbb{R}^m \to \mathbb{R}^n$ be a function and $\boldsymbol{x} \in \mathbb{R}^m, \boldsymbol{y} \in \mathbb{R}^n$ be variables such that:

$$\boldsymbol{y} = f(\boldsymbol{x}).$$

Gradient

For a given $\boldsymbol{x}$, how does an infinitesimal change of $\boldsymbol{x}$ impact y_j?

$$\nabla_{\boldsymbol{x}} y_j = \begin{bmatrix} \frac{\partial y_j}{\partial x_1} \\ \frac{\partial y_j}{\partial x_2} \\ \dots \end{bmatrix}$$

Jacobian

For a given $\boldsymbol{x}$, how does an infinitesimal change of $\boldsymbol{x}$ impact $\boldsymbol{y}$?

$$\mathbf{J}_{\boldsymbol{x}} \boldsymbol{y} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \dots \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \dots \\ \dots & \dots & \dots \end{bmatrix}$$

Chain rule using the Jacobian notation

Let $f : \mathbb{R}^m \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}$ be two functions and $\boldsymbol{x}^m, \boldsymbol{z}^n, y$ be variables such that:

$$\boldsymbol{z} = f(\boldsymbol{x}), \quad y = g(\boldsymbol{z})$$

Chain rule using the Jacobian notation

Let $f : \mathbb{R}^m \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}$ be two functions and $\boldsymbol{x}^m, \boldsymbol{z}^n, y$ be variables such that:

$$\boldsymbol{z} = f(\boldsymbol{x}), \quad y = g(\boldsymbol{z})$$

Partial notation

$$\frac{\partial y}{\partial x_i} = \sum_j \frac{\partial y}{\partial z_j} \cdot \frac{\partial z_j}{\partial x_i}$$

Chain rule using the Jacobian notation

Let $f : \mathbb{R}^m \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}$ be two functions and $\boldsymbol{x}^m, \boldsymbol{z}^n, y$ be variables such that:

$$\boldsymbol{z} = f(\boldsymbol{x}), \quad y = g(\boldsymbol{z})$$

Partial notation

$$\frac{\partial y}{\partial x_i} = \sum_j \frac{\partial y}{\partial z_j} \cdot \frac{\partial z_j}{\partial x_i}$$

Gradient+Jacobian notation

Let $\langle \cdot, \cdot \rangle$ be the dot product operation:

$$\nabla_{\boldsymbol{x}} y = \langle \mathbf{J}_{\boldsymbol{x}} \boldsymbol{z}, \nabla_{\boldsymbol{z}} y \rangle$$

$$\nabla_{\boldsymbol{x}} y = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \dots \end{bmatrix} \in \mathbb{R}^m \qquad \mathbf{J}_{\boldsymbol{x}} \boldsymbol{z} = \begin{bmatrix} \frac{\partial z_1}{\partial x_1} & \frac{\partial z_1}{\partial x_2} & \dots \\ \frac{\partial z_2}{\partial x_1} & \frac{\partial z_2}{\partial x_2} & \dots \\ \dots & \dots & \dots \end{bmatrix} \in \mathbb{R}^{n \times m} \qquad \nabla_{\boldsymbol{z}} y = \begin{bmatrix} \frac{\partial y}{\partial z_1} \\ \frac{\partial y}{\partial z_2} \\ \dots \end{bmatrix} \in \mathbb{R}^n$$

Forward and backward passes

Forward pass

$$\boldsymbol{z}^{(1)} = f^{(1)}(\boldsymbol{x} \quad ; \theta^{(1)})$$

Backward pass

Forward and backward passes

Forward pass

$$\boldsymbol{z}^{(1)} = f^{(1)}(\boldsymbol{x} \quad ; \theta^{(1)})$$

$$\downarrow$$

$$\boldsymbol{z}^{(2)} = f^{(2)}(\boldsymbol{z}^{(1)}; \theta^{(2)})$$

Backward pass

Forward and backward passes

Forward pass

$$\boldsymbol{z}^{(1)} = f^{(1)}(\boldsymbol{x} \quad ; \theta^{(1)})$$

$$\downarrow$$

$$\boldsymbol{z}^{(2)} = f^{(2)}(\boldsymbol{z}^{(1)}; \theta^{(2)})$$

$$\downarrow$$

$$\boldsymbol{z}^{(3)} = f^{(3)}(\boldsymbol{z}^{(2)}; \theta^{(3)})$$

Backward pass

Forward and backward passes

Forward pass

$$\boldsymbol{z}^{(1)} = f^{(1)}(\boldsymbol{x} \quad ; \theta^{(1)})$$

$$\downarrow$$

$$\boldsymbol{z}^{(2)} = f^{(2)}(\boldsymbol{z}^{(1)}; \theta^{(2)})$$

$$\downarrow$$

$$\boldsymbol{z}^{(3)} = f^{(3)}(\boldsymbol{z}^{(2)}; \theta^{(3)})$$

$$\downarrow$$

$$\boldsymbol{z}^{(4)} = f^{(4)}(\boldsymbol{z}^{(3)}; \theta^{(4)})$$

Backward pass

Forward and backward passes

Forward pass

$$
\begin{aligned}
\boldsymbol{z}^{(1)} &= f^{(1)}(\boldsymbol{x} \quad ; \theta^{(1)}) \\
&\downarrow \\
\boldsymbol{z}^{(2)} &= f^{(2)}(\boldsymbol{z}^{(1)}; \theta^{(2)}) \\
&\downarrow \\
\boldsymbol{z}^{(3)} &= f^{(3)}(\boldsymbol{z}^{(2)}; \theta^{(3)}) \\
&\downarrow \\
\boldsymbol{z}^{(4)} &= f^{(4)}(\boldsymbol{z}^{(3)}; \theta^{(4)}) \\
&\downarrow \\
y \quad &= f^{(5)}(\boldsymbol{z}^{(4)}; \theta^{(5)})
\end{aligned}
$$

Backward pass

Forward and backward passes

Forward pass	**Backward pass**
$\boldsymbol{z}^{(1)} = f^{(1)}(\boldsymbol{x} \quad ; \theta^{(1)})$	$\nabla_{\theta^{(1)}} y$
$\downarrow$	
$\boldsymbol{z}^{(2)} = f^{(2)}(\boldsymbol{z}^{(1)}; \theta^{(2)})$	$\nabla_{\theta^{(2)}} y$
$\downarrow$	
$\boldsymbol{z}^{(3)} = f^{(3)}(\boldsymbol{z}^{(2)}; \theta^{(3)})$	$\nabla_{\theta^{(3)}} y$
$\downarrow$	
$\boldsymbol{z}^{(4)} = f^{(4)}(\boldsymbol{z}^{(3)}; \theta^{(4)})$	$\nabla_{\theta^{(4)}} y$
$\downarrow$	
$y \quad = f^{(5)}(\boldsymbol{z}^{(4)}; \theta^{(5)})$	$\nabla_{\theta^{(5)}} y$

Forward and backward passes

Forward pass	**Backward pass**
$\boldsymbol{z}^{(1)} = f^{(1)}(\boldsymbol{x} \quad ; \theta^{(1)})$	$\nabla_{\theta^{(1)}} y$
$\downarrow$	
$\boldsymbol{z}^{(2)} = f^{(2)}(\boldsymbol{z}^{(1)}; \theta^{(2)})$	$\nabla_{\theta^{(2)}} y$
$\downarrow$	
$\boldsymbol{z}^{(3)} = f^{(3)}(\boldsymbol{z}^{(2)}; \theta^{(3)})$	$\nabla_{\theta^{(3)}} y$
$\downarrow$	
$\boldsymbol{z}^{(4)} = f^{(4)}(\boldsymbol{z}^{(3)}; \theta^{(4)})$	$\nabla_{\theta^{(4)}} y = \langle \mathbf{J}_{\theta^{(4)}} \boldsymbol{z}^{(4)}, \nabla_{\boldsymbol{z}^{(4)}} y \rangle$
$\downarrow$	
$y \quad = f^{(5)}(\boldsymbol{z}^{(4)}; \theta^{(5)})$	$\nabla_{\theta^{(5)}} y$

Forward and backward passes

Forward pass		**Backward pass**
$z^{(1)} = f^{(1)}(x \quad ; \theta^{(1)})$		$\nabla_{\theta^{(1)}} y$
$\downarrow$		
$z^{(2)} = f^{(2)}(z^{(1)}; \theta^{(2)})$		$\nabla_{\theta^{(2)}} y$
$\downarrow$		
$z^{(3)} = f^{(3)}(z^{(2)}; \theta^{(3)})$		$\nabla_{\theta^{(3)}} y$
$\downarrow$		
$z^{(4)} = f^{(4)}(z^{(3)}; \theta^{(4)})$		$\nabla_{\theta^{(4)}} y = \langle \mathbf{J}_{\theta^{(4)}} z^{(4)}, \nabla_{z^{(4)}} y \rangle$
$\downarrow$		$\uparrow$
$y \quad = f^{(5)}(z^{(4)}; \theta^{(5)})$	$\nabla_{z^{(4)}} y$	$\nabla_{\theta^{(5)}} y$

Forward and backward passes

Forward pass

$$z^{(1)} = f^{(1)}(x \quad ; \theta^{(1)})$$
$$\downarrow$$
$$z^{(2)} = f^{(2)}(z^{(1)}; \theta^{(2)})$$
$$\downarrow$$
$$z^{(3)} = f^{(3)}(z^{(2)}; \theta^{(3)})$$
$$\downarrow$$
$$z^{(4)} = f^{(4)}(z^{(3)}; \theta^{(4)})$$
$$\downarrow$$
$$y \quad = f^{(5)}(z^{(4)}; \theta^{(5)})$$

Backward pass

$$\nabla_{\theta^{(1)}} y$$
$$\nabla_{\theta^{(2)}} y$$
$$\nabla_{\theta^{(3)}} y = \langle \mathbf{J}_{\theta^{(3)}} z^{(3)}, \nabla_{z^{(3)}} y \rangle$$
$$\nabla_{\theta^{(4)}} y = \langle \mathbf{J}_{\theta^{(4)}} z^{(4)}, \nabla_{z^{(4)}} y \rangle$$
$$\uparrow$$
$$\nabla_{z^{(4)}} y \qquad \nabla_{\theta^{(5)}} y$$

Forward and backward passes

Forward pass	**Backward pass**
$\boldsymbol{z}^{(1)} = f^{(1)}(\boldsymbol{x} \quad ; \theta^{(1)})$	$\nabla_{\theta^{(1)}} y$
$\downarrow$	
$\boldsymbol{z}^{(2)} = f^{(2)}(\boldsymbol{z}^{(1)}; \theta^{(2)})$	$\nabla_{\theta^{(2)}} y$
$\downarrow$	
$\boldsymbol{z}^{(3)} = f^{(3)}(\boldsymbol{z}^{(2)}; \theta^{(3)})$	$\nabla_{\theta^{(3)}} y = \langle \mathbf{J}_{\theta^{(3)}} \boldsymbol{z}^{(3)}, \nabla_{\boldsymbol{z}^{(3)}} y \rangle$
$\downarrow$	$\uparrow$
$\boldsymbol{z}^{(4)} = f^{(4)}(\boldsymbol{z}^{(3)}; \theta^{(4)})$	$\nabla_{\boldsymbol{z}^{(3)}} y = \langle \mathbf{J}_{\boldsymbol{z}^{(3)}} \boldsymbol{z}^{(4)}, \nabla_{\boldsymbol{z}^{(4)}} y \rangle$ $\quad \nabla_{\theta^{(4)}} y = \langle \mathbf{J}_{\theta^{(4)}} \boldsymbol{z}^{(4)}, \nabla_{\boldsymbol{z}^{(4)}} y \rangle$
$\downarrow$	$\uparrow$
$y \quad = f^{(5)}(\boldsymbol{z}^{(4)}; \theta^{(5)})$	$\nabla_{\boldsymbol{z}^{(4)}} y \quad \nabla_{\theta^{(5)}} y$

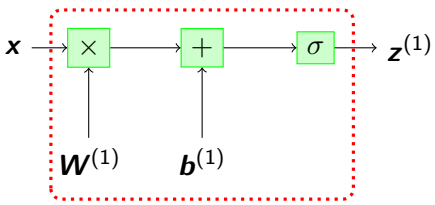
Forward and backward passes

Forward pass	**Backward pass**	
$z^{(1)} = f^{(1)}(x \quad ; \theta^{(1)})$		$\nabla_{\theta^{(1)}} y = \langle \mathbf{J}_{\theta^{(1)}} z^{(1)}, \nabla_{z^{(1)}} y \rangle$
$\downarrow$		$\uparrow$
$z^{(2)} = f^{(2)}(z^{(1)}; \theta^{(2)})$	$\nabla_{z^{(1)}} y = \langle \mathbf{J}_{z^{(1)}} z^{(2)}, \nabla_{z^{(2)}} y \rangle$	$\nabla_{\theta^{(2)}} y = \langle \mathbf{J}_{\theta^{(2)}} z^{(2)}, \nabla_{z^{(2)}} y \rangle$
$\downarrow$		$\uparrow$
$z^{(3)} = f^{(3)}(z^{(2)}; \theta^{(3)})$	$\nabla_{z^{(2)}} y = \langle \mathbf{J}_{z^{(2)}} z^{(3)}, \nabla_{z^{(3)}} y \rangle$	$\nabla_{\theta^{(3)}} y = \langle \mathbf{J}_{\theta^{(3)}} z^{(3)}, \nabla_{z^{(3)}} y \rangle$
$\downarrow$		$\uparrow$
$z^{(4)} = f^{(4)}(z^{(3)}; \theta^{(4)})$	$\nabla_{z^{(3)}} y = \langle \mathbf{J}_{z^{(3)}} z^{(4)}, \nabla_{z^{(4)}} y \rangle$	$\nabla_{\theta^{(4)}} y = \langle \mathbf{J}_{\theta^{(4)}} z^{(4)}, \nabla_{z^{(4)}} y \rangle$
$\downarrow$		$\uparrow$
$y \quad = f^{(5)}(z^{(4)}; \theta^{(5)})$	$\nabla_{z^{(4)}} y$	$\nabla_{\theta^{(5)}} y$

Computation Graph (CG) 1/2

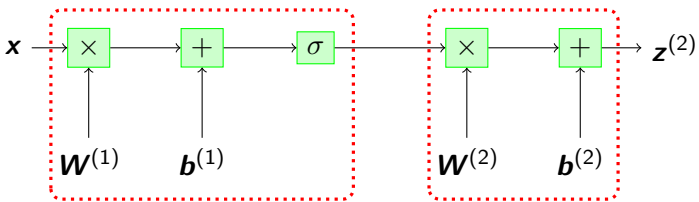
$\boldsymbol{x}$

Computation Graph (CG) 1/2



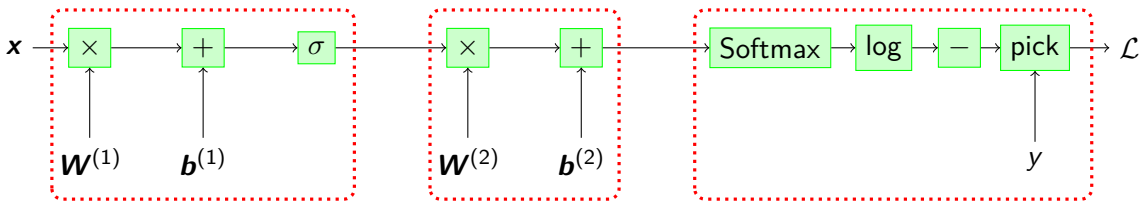
$$\boldsymbol{z}^{(1)} = \sigma\left(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right)$$

Computation Graph (CG) 1/2



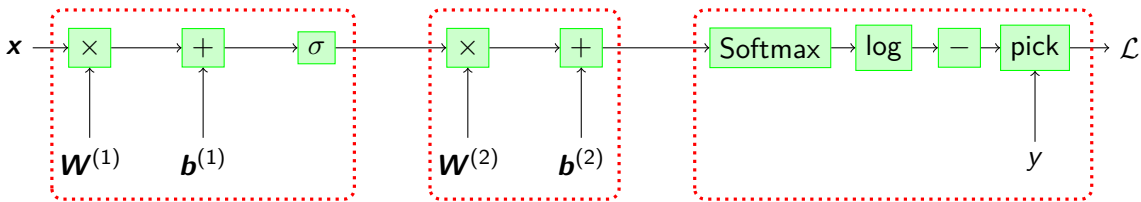
$$\boldsymbol{z}^{(1)} = \sigma\left(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right) \qquad \boldsymbol{z}^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{x} + \boldsymbol{b}^{(2)}$$

Computation Graph (CG) 1/2



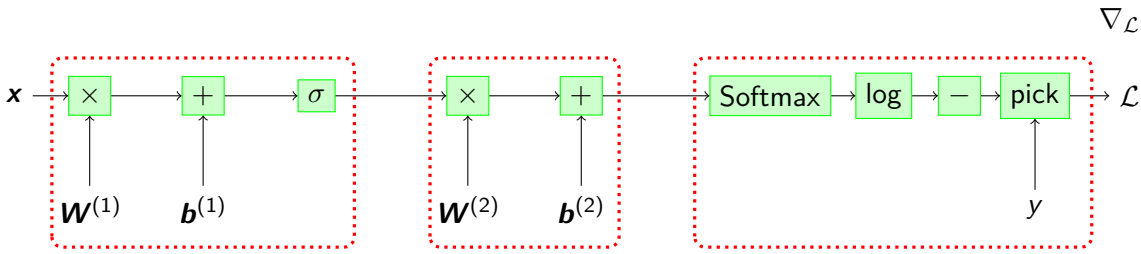
$$\boldsymbol{z}^{(1)} = \sigma\left(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right) \qquad \boldsymbol{z}^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{x} + \boldsymbol{b}^{(2)} \qquad \mathcal{L} = -\log \frac{\exp(\boldsymbol{z}_y^{(2)})}{\sum_{y'} \exp(\boldsymbol{z}_{y'}^{(2)})}$$

Computation Graph (CG) 1/2



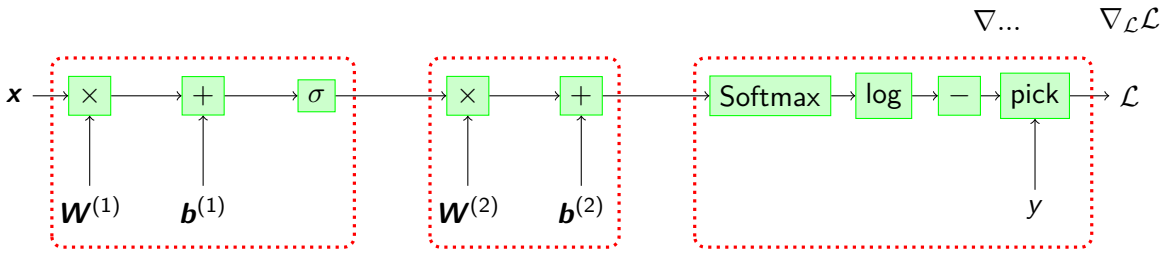
$$\boldsymbol{z}^{(1)} = \sigma\left(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right) \qquad \boldsymbol{z}^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{x} + \boldsymbol{b}^{(2)} \qquad \mathcal{L} = -\log \frac{\exp(\boldsymbol{z}_y^{(2)})}{\sum_{y'} \exp(\boldsymbol{z}_{y'}^{(2)})}$$

Computation Graph (CG) 1/2



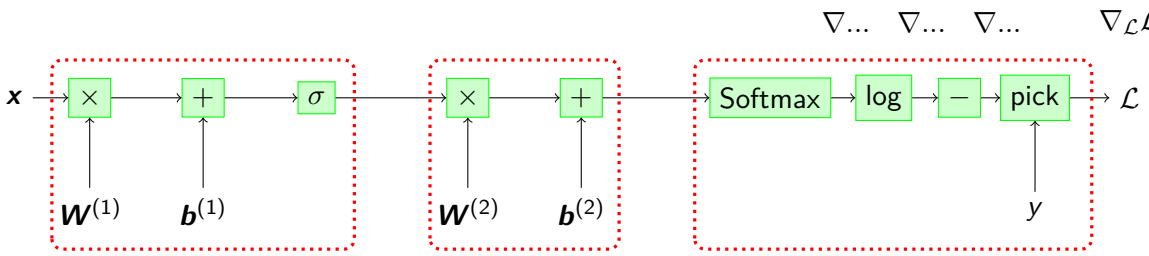
$$\boldsymbol{z}^{(1)} = \sigma\left(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right) \qquad \boldsymbol{z}^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{x} + \boldsymbol{b}^{(2)} \qquad \mathcal{L} = -\log \frac{\exp(\boldsymbol{z}_y^{(2)})}{\sum_{y'} \exp(\boldsymbol{z}_{y'}^{(2)})}$$

Computation Graph (CG) 1/2



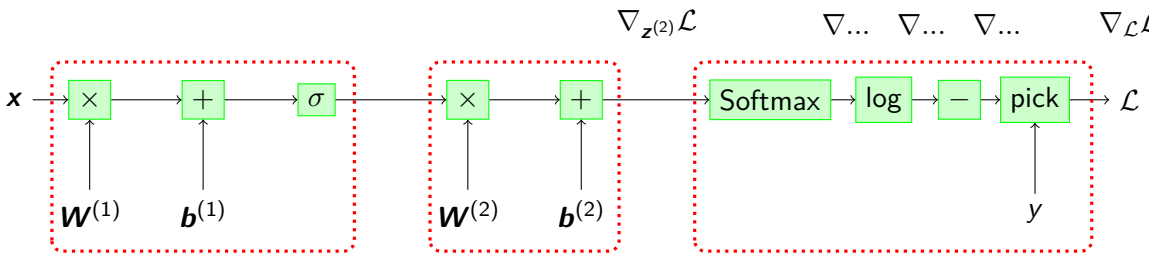
$$\boldsymbol{z}^{(1)} = \sigma\left(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right) \qquad \boldsymbol{z}^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{x} + \boldsymbol{b}^{(2)} \qquad \mathcal{L} = -\log \frac{\exp(\boldsymbol{z}_y^{(2)})}{\sum_{y'} \exp(\boldsymbol{z}_{y'}^{(2)})}$$

Computation Graph (CG) 1/2



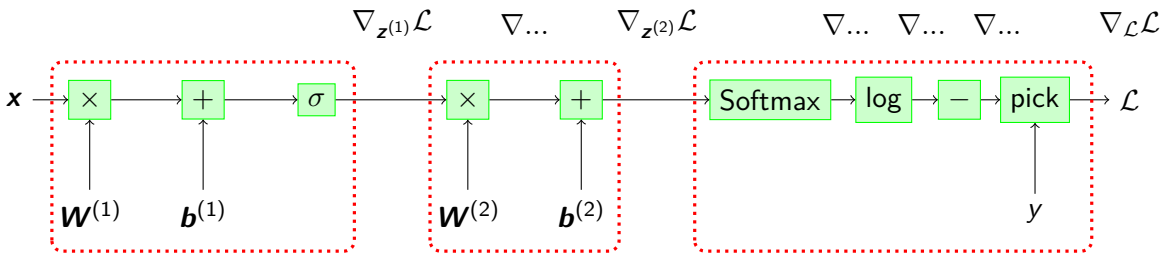
$$\boldsymbol{z}^{(1)} = \sigma\left(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right) \qquad \boldsymbol{z}^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{x} + \boldsymbol{b}^{(2)} \qquad \mathcal{L} = -\log \frac{\exp(\boldsymbol{z}_y^{(2)})}{\sum_{y'} \exp(\boldsymbol{z}_{y'}^{(2)})}$$

Computation Graph (CG) 1/2



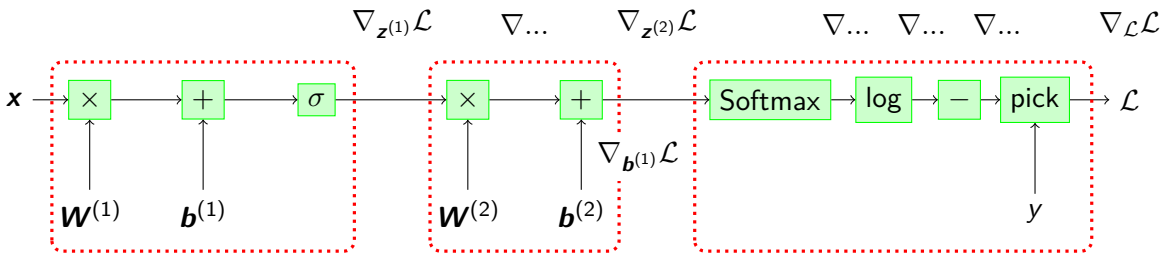
$$z^{(1)} = \sigma\left(W^{(1)}x + b^{(1)}\right) \qquad z^{(2)} = W^{(2)}x + b^{(2)} \qquad \mathcal{L} = -\log \frac{\exp(z_y^{(2)})}{\sum_{y'} \exp(z_{y'}^{(2)})}$$

Computation Graph (CG) 1/2



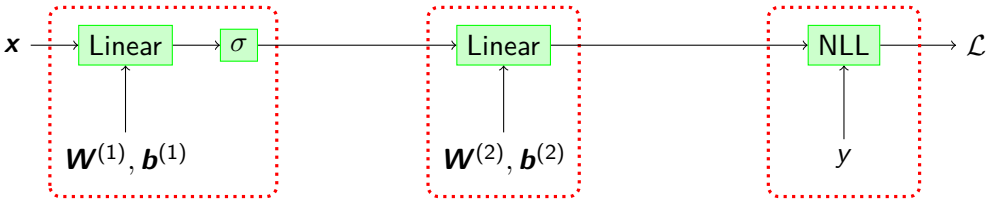
$$z^{(1)} = \sigma\left(W^{(1)}x + b^{(1)}\right) \qquad z^{(2)} = W^{(2)}x + b^{(2)} \qquad \mathcal{L} = -\log \frac{\exp(z_y^{(2)})}{\sum_{y'} \exp(z_{y'}^{(2)})}$$

Computation Graph (CG) 1/2



$$z^{(1)} = \sigma\left(W^{(1)}x + b^{(1)}\right) \qquad z^{(2)} = W^{(2)}x + b^{(2)} \qquad \mathcal{L} = -\log \frac{\exp(z_y^{(2)})}{\sum_{y'} \exp(z_{y'}^{(2)})}$$

Computation Graph (CG) 2/2



$$\boldsymbol{z}^{(1)} = \sigma\left(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right) \qquad \boldsymbol{z}^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{x} + \boldsymbol{b}^{(2)} \qquad \mathcal{L} = -\log \frac{\exp(\boldsymbol{z}_y^{(2)})}{\sum_{y'} \exp(\boldsymbol{z}_{y'}^{(2)})}$$

Computation Graph (CG) implementation

CG construction / Eager forward pass

The computation graph is built in **topological order** ($\sim$order execution of operations):

- $\boldsymbol{x}, \boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, ..., \mathcal{L}$: Expression nodes
- $\boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)}, ...$: Parameter nodes

Expression node

- Values
- Gradient
- Backward operation
- Backpointer(s) to antecedents

The backward operation and backpointer(s) are null for input operations

Parameter node

- **Persistent values**
- Gradient

Eager forward pass example

Non-linear activation function:

$$\boldsymbol{z}' = \text{relu}(\boldsymbol{z})$$

Eager forward pass example

Non-linear activation function:

$$z' = \text{relu}(z)$$

function RELU(z)

▷ Create node

$z' = \text{ExpressionNode}()$

▷ Compute forward value

$$z'.value = \begin{bmatrix} \max(0, z_1) \\ \max(0, z_2) \\ \dots \end{bmatrix}$$

▷ Set backward operation

$z'.d = \text{d\_relu}$

▷ Set backpointers

$z'.backptrs = [z]$

return z'

Eager forward pass example

Non-linear activation function:

$$\boldsymbol{z}' = \text{relu}(\boldsymbol{z})$$

function RELU($\boldsymbol{z}$)

▷ Create node

$\boldsymbol{z}' = \text{ExpressionNode}()$

▷ Compute forward value

$$\boldsymbol{z}'.\textit{value} = \begin{bmatrix} \max(0, \boldsymbol{z}_1) \\ \max(0, \boldsymbol{z}_2) \\ \dots \end{bmatrix}$$

▷ Set backward operation

$\boldsymbol{z}'.d = \text{d\_relu}$

▷ Set backpointers

$\boldsymbol{z}'.\textit{backptrs} = [\boldsymbol{z}]$

return $\boldsymbol{z}'$

Projection operation $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$:

$$\boldsymbol{z} = \text{Linear}(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b})$$

Eager forward pass example

Non-linear activation function:

$$\boldsymbol{z}' = \text{relu}(\boldsymbol{z})$$

function RELU($\boldsymbol{z}$)

▷ Create node

$\boldsymbol{z}' = \text{ExpressionNode}()$

▷ Compute forward value

$$\boldsymbol{z}'.value = \begin{bmatrix} \max(0, \boldsymbol{z}_1) \\ \max(0, \boldsymbol{z}_2) \\ \dots \end{bmatrix}$$

▷ Set backward operation

$\boldsymbol{z}'.d = \text{d\_relu}$

▷ Set backpointers

$\boldsymbol{z}'.backptrs = [\boldsymbol{z}]$

return $\boldsymbol{z}'$

Projection operation $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$:

$$\boldsymbol{z} = \text{Linear}(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b})$$

function LINEAR($\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}$)

▷ Create node

$\boldsymbol{z} = \text{ExpressionNode}()$

▷ Compute forward value

$\boldsymbol{z}.value = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$

▷ Set backward operation

$\boldsymbol{z}.d = \text{d\_linear}$

▷ Set backpointers

$\boldsymbol{z}.backptrs = [\boldsymbol{W}, \boldsymbol{b}]$

return $\boldsymbol{z}$

Backward pass

Execution of the backward pass

Nodes are visited in **reverse topological order** (reverse order of creation):

- The gradient of the loss (last created node) is set to 1
- For each node, we call it's derivative function
- The derivative functions will backpropagate gradient to antecedents
  Gradient must be **accumulated** (expressions can be used several times)

Backward pass

Execution of the backward pass

Nodes are visited in **reverse topological order** (reverse order of creation):

- The gradient of the loss (last created node) is set to 1
- For each node, we call it's derivative function
- The derivative functions will backpropagate gradient to antecedents
  Gradient must be **accumulated** (expressions can be used several times)

function BACKWARD(`nodes`, $\mathcal{L}$)
  $\mathcal{L}.grad = 1$
  for `n` $\in$ reversed(`nodes`) **do**
    ▷ Call the derivative functions
    `n`.d(n.backptrs)

Backward pass example: relu 1/2

$$\text{relu}(x) = \begin{cases} 0 \text{ if } \leq 0 \\ x \text{ otherwise} \end{cases} \qquad \frac{\partial}{\partial x} \text{relu}(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x > 0 \\ \text{undefined otherwise, } =0 \end{cases}$$

Backward pass example: relu 1/2

$$\text{relu}(x) = \begin{cases} 0 \text{ if } \leq 0 \\ x \text{ otherwise} \end{cases} \qquad \frac{\partial}{\partial x}\text{relu}(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x > 0 \\ \text{undefined otherwise, } =0 \end{cases}$$

$$\nabla_{\boldsymbol{z}}\mathcal{L} = \langle \mathbf{J}_{\boldsymbol{z}}\boldsymbol{z}', \nabla_{\boldsymbol{z}'}\mathcal{L} \rangle$$

$$\mathbf{J}_{\boldsymbol{z}}\boldsymbol{z}' = \begin{bmatrix} \frac{\partial z_1'}{\partial z_1} & \frac{\partial z_1'}{\partial z_2} & \dots \\ \frac{\partial z_2'}{\partial z_1} & \frac{\partial z_2'}{\partial z_2} & \dots \\ \dots & \dots & \dots \end{bmatrix}$$

Backward pass example: relu 1/2

$$\text{relu}(x) = \begin{cases} 0 \text{ if } \leq 0 \\ x \text{ otherwise} \end{cases} \qquad \frac{\partial}{\partial x}\text{relu}(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x > 0 \\ \text{undefined otherwise, } =0 \end{cases}$$

$$\nabla_{\boldsymbol{z}}\mathcal{L} = \langle \mathbf{J}_{\boldsymbol{z}}\boldsymbol{z}', \nabla_{\boldsymbol{z}'}\mathcal{L} \rangle$$

$$\mathbf{J}_{\boldsymbol{z}}\boldsymbol{z}' = \begin{bmatrix} \frac{\partial z_1'}{\partial z_1} & \frac{\partial z_1'}{\partial z_2} & \dots \\ \frac{\partial z_2'}{\partial z_1} & \frac{\partial z_2'}{\partial z_2} & \dots \\ \dots & \dots & \dots \end{bmatrix}$$

$$\frac{\partial z_i'}{\partial z_j} = \begin{cases} 0, \text{ if } i \neq j \quad \text{(piecewise function!)} \\ \frac{\partial}{\partial z_i} f(z_i), \text{ if } i = j \end{cases}$$

Backward pass example: relu 1/2

$$\text{relu}(x) = \begin{cases} 0 \text{ if } \leq 0 \\ x \text{ otherwise} \end{cases} \qquad \frac{\partial}{\partial x}\text{relu}(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x > 0 \\ \text{undefined otherwise, } =0 \end{cases}$$

$$\nabla_{\boldsymbol{z}}\mathcal{L} = \langle \mathbf{J}_{\boldsymbol{z}}\boldsymbol{z}', \nabla_{\boldsymbol{z}'}\mathcal{L}\rangle$$

$$\nabla_{\boldsymbol{z}}\mathcal{L} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial z_1} \\ \frac{\partial \mathcal{L}}{\partial z_2} \\ \cdots \end{bmatrix}$$

$$\mathbf{J}_{\boldsymbol{z}}\boldsymbol{z}' = \begin{bmatrix} \frac{\partial z_1'}{\partial z_1} & \frac{\partial z_1'}{\partial z_2} & \cdots \\ \frac{\partial z_2'}{\partial z_1} & \frac{\partial z_2'}{\partial z_2} & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix}$$

$$\frac{\partial z_i'}{\partial z_j} = \begin{cases} 0, \text{ if } i \neq j \quad \text{(piecewise function!)} \\ \frac{\partial}{\partial z_i} f(z_i), \text{ if } i = j \end{cases}$$

Backward pass example: relu 1/2

$$\text{relu}(x) = \begin{cases} 0 \text{ if } \leq 0 \\ x \text{ otherwise} \end{cases} \qquad \frac{\partial}{\partial x}\text{relu}(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x > 0 \\ \text{undefined otherwise, } = 0 \end{cases}$$

$$\nabla_{\boldsymbol{z}}\mathcal{L} = \langle \mathbf{J}_{\boldsymbol{z}}\boldsymbol{z}', \nabla_{\boldsymbol{z}'}\mathcal{L} \rangle$$

$$\nabla_{\boldsymbol{z}}\mathcal{L} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial z_1} \\ \frac{\partial \mathcal{L}}{\partial z_2} \\ \dots \end{bmatrix}$$

$$\mathbf{J}_{\boldsymbol{z}}\boldsymbol{z}' = \begin{bmatrix} \frac{\partial z_1'}{\partial z_1} & \frac{\partial z_1'}{\partial z_2} & \dots \\ \frac{\partial z_2'}{\partial z_1} & \frac{\partial z_2'}{\partial z_2} & \dots \\ \dots & \dots & \dots \end{bmatrix}$$

$$\frac{\partial \mathcal{L}}{\partial z_i}$$

$$\frac{\partial z_i'}{\partial z_j} = \begin{cases} 0, \text{ if } i \neq j \quad \text{(piecewise function!)} \\ \frac{\partial}{\partial z_i} f(z_i), \text{ if } i = j \end{cases}$$

Backward pass example: relu 1/2

$$\text{relu}(x) = \begin{cases} 0 \text{ if } \leq 0 \\ x \text{ otherwise} \end{cases} \qquad \frac{\partial}{\partial x}\text{relu}(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x > 0 \\ \text{undefined otherwise, } = 0 \end{cases}$$

$$\nabla_{\boldsymbol{z}}\mathcal{L} = \langle \mathbf{J}_{\boldsymbol{z}}\boldsymbol{z}', \nabla_{\boldsymbol{z}'}\mathcal{L} \rangle$$

$$\nabla_{\boldsymbol{z}}\mathcal{L} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial z_1} \\ \frac{\partial \mathcal{L}}{\partial z_2} \\ \cdots \end{bmatrix}$$

$$\mathbf{J}_{\boldsymbol{z}}\boldsymbol{z}' = \begin{bmatrix} \frac{\partial z_1'}{\partial z_1} & \frac{\partial z_1'}{\partial z_2} & \cdots \\ \frac{\partial z_2'}{\partial z_1} & \frac{\partial z_2'}{\partial z_2} & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix}$$

$$\frac{\partial \mathcal{L}}{\partial z_i} = \sum_j \frac{\partial \mathcal{L}}{\partial z_j'} \cdot \frac{\partial z_j'}{\partial z_i}$$

$$\frac{\partial z_i'}{\partial z_j} = \begin{cases} 0, \text{ if } i \neq j \quad \text{(piecewise function!)} \\ \frac{\partial}{\partial z_i} f(z_i), \text{ if } i = j \end{cases}$$

Backward pass example: relu 1/2

$$\text{relu}(x) = \begin{cases} 0 \text{ if } \leq 0 \\ x \text{ otherwise} \end{cases} \qquad \frac{\partial}{\partial x}\text{relu}(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x > 0 \\ \text{undefined otherwise, } =0 \end{cases}$$

$$\nabla_{\boldsymbol{z}}\mathcal{L} = \langle \mathbf{J}_{\boldsymbol{z}}\boldsymbol{z}', \nabla_{\boldsymbol{z}'}\mathcal{L} \rangle$$

$$\nabla_{\boldsymbol{z}}\mathcal{L} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial z_1} \\ \frac{\partial \mathcal{L}}{\partial z_2} \\ \cdots \end{bmatrix}$$

$$\mathbf{J}_{\boldsymbol{z}}\boldsymbol{z}' = \begin{bmatrix} \frac{\partial z_1'}{\partial z_1} & \frac{\partial z_1'}{\partial z_2} & \cdots \\ \frac{\partial z_2'}{\partial z_1} & \frac{\partial z_2'}{\partial z_2} & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix}$$

$$\frac{\partial \mathcal{L}}{\partial z_i} = \sum_j \frac{\partial \mathcal{L}}{\partial z_j'} \cdot \frac{\partial z_j'}{\partial z_i}$$

$$= \frac{\partial \mathcal{L}}{\partial z_i'} \cdot \frac{\partial z_i'}{\partial z_i} \quad \text{(piecewise function!)}$$

$$\frac{\partial z_i'}{\partial z_j} = \begin{cases} 0, \text{ if } i \neq j \quad \text{(piecewise function!)} \\ \frac{\partial}{\partial z_i} f(z_i), \text{ if } i = j \end{cases}$$

Backward pass example: relu 1/2

$$\text{relu}(x) = \begin{cases} 0 \text{ if } \leq 0 \\ x \text{ otherwise} \end{cases} \qquad \frac{\partial}{\partial x}\text{relu}(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x > 0 \\ \text{undefined otherwise, } = 0 \end{cases}$$

$$\nabla_{\boldsymbol{z}}\mathcal{L} = \langle \mathbf{J}_{\boldsymbol{z}}\boldsymbol{z}', \nabla_{\boldsymbol{z}'}\mathcal{L} \rangle$$

$$\nabla_{\boldsymbol{z}}\mathcal{L} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial z_1} \\ \frac{\partial \mathcal{L}}{\partial z_2} \\ \cdots \end{bmatrix}$$

$$\mathbf{J}_{\boldsymbol{z}}\boldsymbol{z}' = \begin{bmatrix} \frac{\partial z_1'}{\partial z_1} & \frac{\partial z_1'}{\partial z_2} & \cdots \\ \frac{\partial z_2'}{\partial z_1} & \frac{\partial z_2'}{\partial z_2} & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix}$$

$$\frac{\partial \mathcal{L}}{\partial z_i} = \sum_j \frac{\partial \mathcal{L}}{\partial z_j'} \cdot \frac{\partial z_j'}{\partial z_i}$$

$$= \frac{\partial \mathcal{L}}{\partial z_i'} \cdot \frac{\partial z_i'}{\partial z_i} \quad \text{(piecewise function!)}$$

$$= \frac{\partial \mathcal{L}}{\partial z_i'} \cdot \mathbb{1}[z_i > 0]$$

$$\frac{\partial z_i'}{\partial z_j} = \begin{cases} 0, \text{ if } i \neq j \quad \text{(piecewise function!)} \\ \frac{\partial}{\partial z_i} f(z_i), \text{ if } i = j \end{cases}$$

Backward pass example: relu 2/2

$$\text{relu}(x) = \begin{cases} 0 \text{ if } \leq 0 \\ x \text{ otherwise} \end{cases} \qquad \text{relu}'(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x > 0 \\ \text{undefined otherwise} \end{cases}$$

function RELU(z)
  $z' = \text{ExpressionNode}()$
  $z'.value = \begin{bmatrix} \max(0, z_1) \\ \max(0, z_2) \\ \dots \end{bmatrix}$
  $z'.d = \text{d\_relu}$
  $z'.backptrs = [z]$

  return z'

$$\frac{\partial \mathcal{L}}{\partial z_i} = \frac{\partial \mathcal{L}}{\partial z'_i} \cdot \mathbb{1}[z_i > 0]$$

function D_RELU($z', [z]$)
  for $i \in \{1...n\}$ **do**
    ▷ If the value is positive,
    ▷ we copy the gradient
    if $z_i > 0$ **then**
      $z.grad_i = z.grad_i + z'.grad_i$

Backward pass example: linear projection 1/2

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \qquad \Leftrightarrow \qquad z_j = \sum_k W_{j,k} x_k + b_j$$

Backward pass example: linear projection 1/2

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \qquad \Leftrightarrow \qquad z_j = \sum_k W_{j,k} x_k + b_j$$

$$\frac{\partial \mathcal{L}}{\partial b_i} = \sum_j \frac{\partial \mathcal{L}}{\partial z_j} \cdot \frac{\partial z_j}{\partial b_i}$$

Backward pass example: linear projection 1/2

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \qquad \Leftrightarrow \qquad z_j = \sum_k W_{j,k} x_k + b_j$$

$$\frac{\partial \mathcal{L}}{\partial b_i} = \sum_j \frac{\partial \mathcal{L}}{\partial z_j} \cdot \frac{\partial z_j}{\partial b_i}$$

$$\frac{\partial z_j}{\partial b_i} = \frac{\partial}{\partial b_i} \sum_k W_{j,k} x_k + b_j$$

Backward pass example: linear projection 1/2

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \qquad \Leftrightarrow \qquad z_j = \sum_k W_{j,k} x_k + b_j$$

$$\frac{\partial \mathcal{L}}{\partial b_i} = \sum_j \frac{\partial \mathcal{L}}{\partial z_j} \cdot \frac{\partial z_j}{\partial b_i}$$

$$\frac{\partial z_j}{\partial b_i} = \frac{\partial}{\partial b_i} \sum_k W_{j,k} x_k + b_j = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

Backward pass example: linear projection 1/2

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \qquad \Leftrightarrow \qquad z_j = \sum_k W_{j,k} x_k + b_j$$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial b_i} &= \sum_j \frac{\partial \mathcal{L}}{\partial z_j} \cdot \frac{\partial z_j}{\partial b_i} \\
&= \sum_j \frac{\partial \mathcal{L}}{\partial z_j} \cdot \mathbb{1}[j = i] \\
&= \frac{\partial \mathcal{L}}{\partial z_i} \quad \text{(copy incoming gradient!)}
\end{aligned}$$

$$\frac{\partial z_j}{\partial b_i} = \frac{\partial}{\partial b_i} \sum_k W_{j,k} x_k + b_j = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

Backward pass example: linear projection 1/2

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \qquad \Leftrightarrow \qquad z_j = \sum_k W_{j,k} x_k + b_j$$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial b_i} &= \sum_j \frac{\partial \mathcal{L}}{\partial z_j} \cdot \frac{\partial z_j}{\partial b_i} \\
&= \sum_j \frac{\partial \mathcal{L}}{\partial z_j} \cdot \mathbb{1}[j = i] \\
&= \frac{\partial \mathcal{L}}{\partial z_i} \quad \text{(copy incoming gradient!)}
\end{aligned}$$

$$\frac{\partial z_j}{\partial b_i} = \frac{\partial}{\partial b_i} \sum_k W_{j,k} x_k + b_j = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

$$\frac{\partial \mathcal{L}}{\partial W_{i,l}} = \sum_j \frac{\partial \mathcal{L}}{\partial z_j} \cdot \frac{\partial z_j}{W_{i,l}}$$

Backward pass example: linear projection 1/2

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \qquad \Leftrightarrow \qquad z_j = \sum_k W_{j,k} x_k + b_j$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial b_i} &= \sum_j \frac{\partial \mathcal{L}}{\partial z_j} \cdot \frac{\partial z_j}{\partial b_i} \\ &= \sum_j \frac{\partial \mathcal{L}}{\partial z_j} \cdot \mathbb{1}[j = i] \\ &= \frac{\partial \mathcal{L}}{\partial z_i} \quad \text{(copy incoming gradient!)}\end{aligned}$$

$$\frac{\partial z_j}{\partial b_i} = \frac{\partial}{\partial b_i} \sum_k W_{j,k} x_k + b_j = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

$$\frac{\partial \mathcal{L}}{\partial W_{i,l}} = \sum_j \frac{\partial \mathcal{L}}{\partial z_j} \cdot \frac{\partial z_j}{W_{i,l}}$$

$$\begin{aligned}\frac{\partial z_j}{\partial W_{i,l}} &= \frac{\partial}{\partial W_{i,l}} \sum_k W_{j,k} x_k + b_j \\ &= \begin{cases} x_l, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}\end{aligned}$$

Backward pass example: linear projection 1/2

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \qquad \Leftrightarrow \qquad z_j = \sum_k W_{j,k} x_k + b_j$$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial b_i} &= \sum_j \frac{\partial \mathcal{L}}{\partial z_j} \cdot \frac{\partial z_j}{\partial b_i} \\
&= \sum_j \frac{\partial \mathcal{L}}{\partial z_j} \cdot \mathbb{1}[j = i] \\
&= \frac{\partial \mathcal{L}}{\partial z_i} \quad \text{(copy incoming gradient!)}
\end{aligned}$$

$$\frac{\partial z_j}{\partial b_i} = \frac{\partial}{\partial b_i} \sum_k W_{j,k} x_k + b_j = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial W_{i,l}} &= \sum_j \frac{\partial \mathcal{L}}{\partial z_j} \cdot \frac{\partial z_j}{W_{i,l}} \\
&= \sum_j \frac{\partial \mathcal{L}}{\partial z_j} \cdot x_l \cdot \mathbb{1}[i = j]
\end{aligned}$$

$$\begin{aligned}
\frac{\partial z_j}{\partial W_{i,l}} &= \frac{\partial}{\partial W_{i,l}} \sum_k W_{j,k} x_k + b_j \\
&= \begin{cases} x_l, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}
\end{aligned}$$

Backward pass example: linear projection 1/2

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \qquad \Leftrightarrow \qquad z_j = \sum_k W_{j,k} x_k + b_j$$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial b_i} &= \sum_j \frac{\partial \mathcal{L}}{\partial z_j} \cdot \frac{\partial z_j}{\partial b_i} \\
&= \sum_j \frac{\partial \mathcal{L}}{\partial z_j} \cdot \mathbb{1}[j = i] \\
&= \frac{\partial \mathcal{L}}{\partial z_i} \quad \text{(copy incoming gradient!)}
\end{aligned}$$

$$\frac{\partial z_j}{\partial b_i} = \frac{\partial}{\partial b_i} \sum_k W_{j,k} x_k + b_j = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}$$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial W_{i,l}} &= \sum_j \frac{\partial \mathcal{L}}{\partial z_j} \cdot \frac{\partial z_j}{W_{i,l}} \\
&= \sum_j \frac{\partial \mathcal{L}}{\partial z_j} \cdot x_l \cdot \mathbb{1}[i = j] \\
\nabla_{\boldsymbol{W}} \mathcal{L} &= (\nabla_{\boldsymbol{z}} \mathcal{L})(\boldsymbol{x}^\top) \quad \text{(outer product)}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial z_j}{\partial W_{i,l}} &= \frac{\partial}{\partial W_{i,l}} \sum_k W_{j,k} x_k + b_j \\
&= \begin{cases} x_l, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}
\end{aligned}$$

Backward pass example: linear projection 2/2

function LINEAR($\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}$)
  $\boldsymbol{z}$ = ExpressionNode()
  $\boldsymbol{z}.value = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$
  $\boldsymbol{z}.d$ = d_linear
  $\boldsymbol{z}.backptrs = [\boldsymbol{W}, \boldsymbol{b}]$

  return $\boldsymbol{z}$

$$\frac{\partial \mathcal{L}}{\partial b_i} = \frac{\partial \mathcal{L}}{\partial z_i} \qquad \nabla_{\boldsymbol{W}} \mathcal{L} = (\nabla_{\boldsymbol{z}} \mathcal{L})(\boldsymbol{x}^\top)$$

function D_LINEAR($\boldsymbol{z}, [\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}]$)
  $\boldsymbol{b}.grad = \boldsymbol{b}.grad + \boldsymbol{z}.grad$
  $\boldsymbol{W}.grad = \boldsymbol{W}.grad + \boldsymbol{z}.grad$ @ $\boldsymbol{x}^\top$

Backward pass example: linear projection 2/2

function LINEAR($\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}$)
  $\boldsymbol{z}$ = ExpressionNode()
  $\boldsymbol{z}.value = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$
  $\boldsymbol{z}.d$ = d_linear
  $\boldsymbol{z}.backptrs = [\boldsymbol{W}, \boldsymbol{b}]$

  return $\boldsymbol{z}$

$$\frac{\partial \mathcal{L}}{\partial b_i} = \frac{\partial \mathcal{L}}{\partial z_i} \qquad \nabla_{\boldsymbol{W}} \mathcal{L} = (\nabla_{\boldsymbol{z}} \mathcal{L})(\boldsymbol{x}^\top)$$

function D_LINEAR($\boldsymbol{z}, [\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}]$)
  $\boldsymbol{b}.grad = \boldsymbol{b}.grad + \boldsymbol{z}.grad$
  $\boldsymbol{W}.grad = \boldsymbol{W}.grad + \boldsymbol{z}.grad$ @ $\boldsymbol{x}^\top$
  $\boldsymbol{x}.grad = \boldsymbol{x}.grad + \ ...$

Missing gradient?

Why don't we backpropagate to $\boldsymbol{x}$?!
We do not need it for today's lab exercises, you will see how to do that next week.

Summary

Computation graph

- Forward pass: compute values
- Backward pass: compute gradient for each parameter
- Gradient initialization: you should be careful with that because gradient is accumulated

First lab exercises

- Simple linear model: don't build a computation graph,
  explicitly apply forward and backward operations
- `d_Linear`: return a tuple with gradient of $\boldsymbol{W}$ and $\boldsymbol{b}$ instead of writing into a node
- Do not need to worry about gradient initialization / accumulation :-)

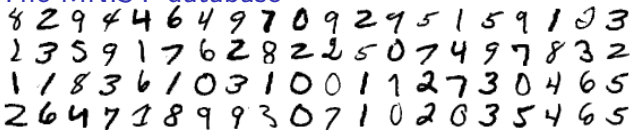
Pytorch

In Pytorch, expression nodes used to be of type `Variable`.
Nowadays, autodiff is directly implemented in the `Tensor` class.

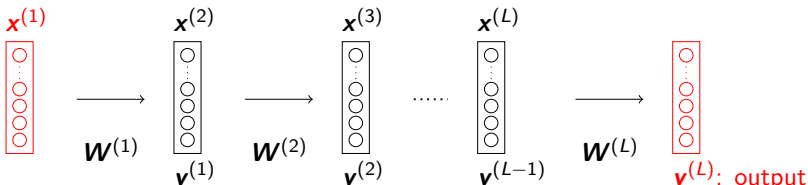
Vanishing gradient, activation functions and initialization

Experimental observations

The MNIST database



Comparison of different depth for feed-forward architecture



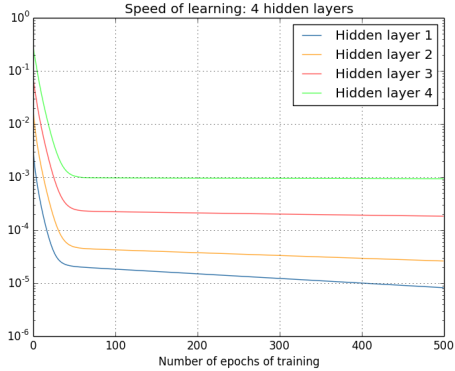
- Hidden layers have a sigmoid activation function.
- The output layer is softmax.

Experimental observations: http://neuralnetworksanddeeplearning.com/chap5.html

- Without hidden layer: $\approx 88\%$ accuracy
- 1 hidden layer (30): $\approx 96.5\%$ accuracy
- 2 hidden layer (30): $\approx 96.9\%$ accuracy
- 3 hidden layer (30): $\approx 96.5\%$ accuracy
- 4 hidden layer (30): $\approx 96.5\%$ accuracy

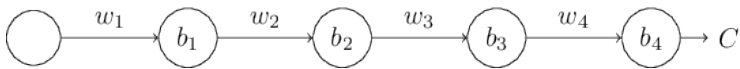
Experimental observations: http://neuralnetworksanddeeplearning.com/chap5.html

- Without hidden layer: $\approx 88\%$ accuracy
- 1 hidden layer (30): $\approx 96.5\%$ accuracy
- 2 hidden layer (30): $\approx 96.9\%$ accuracy
- 3 hidden layer (30): $\approx 96.5\%$ accuracy
- 4 hidden layer (30): $\approx 96.5\%$ accuracy



Intuitive explanation 1/2

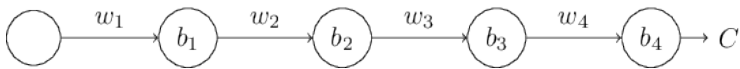
Let consider the simplest deep neural network, with just a single neuron in each layer.



w_i, b_i are resp. the weight and bias of neuron i and C some loss function.

Intuitive explanation 1/2

Let consider the simplest deep neural network, with just a single neuron in each layer.



w_i, b_i are resp. the weight and bias of neuron i and C some loss function.

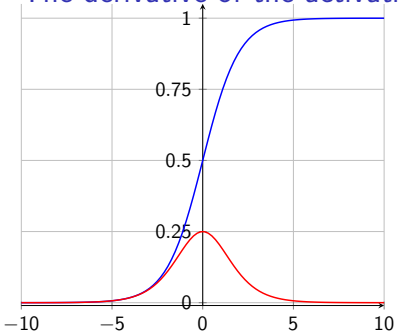
Compute the gradient of C *w.r.t* the bias b_1

$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial y_4} \times \frac{\partial y_4}{\partial a_4} \times \frac{\partial a_4}{\partial y_3} \times \frac{\partial y_3}{\partial a_3} \times \frac{\partial a_3}{\partial y_2} \times \frac{\partial y_2}{\partial a_2} \times \frac{\partial a_2}{\partial y_1} \times \frac{\partial y_1}{\partial a_1} \times \frac{\partial a_1}{\partial b_1} \tag{1}$$

$$= \frac{\partial C}{\partial y_4} \times \sigma'(a_4) \times w_4 \times \sigma'(a_3) \times w_3 \times \sigma'(a_2) \times w_2 \times \sigma'(a_1) \tag{2}$$

Intuitive explanation 2/2

The derivative of the activation function: σ'



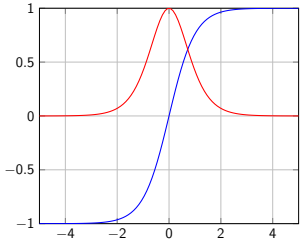
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$
$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Vanishing gradient

- if the last layer are well trained (and outputs "strong values" close to 0 or 1),
- early layers receive a really small incoming gradient.

In the "best case", we successive multiplications by 0.25!

Other activation functions

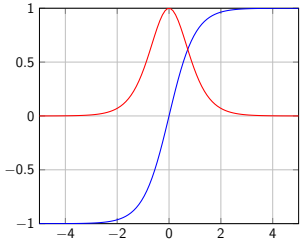


Hyperbolic tangent

$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)} \qquad \tanh'(x) = 1 - \tanh(x)^2$$

- Better gradient than sigmoid around 0
- Popular in Natural Language Processing

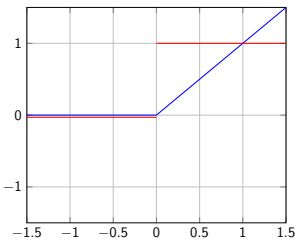
Other activation functions



Hyperbolic tangent

$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)} \qquad \tanh'(x) = 1 - \tanh(x)^2$$

- Better gradient than sigmoid around 0
- Popular in Natural Language Processing



Rectified Linear Unit

$$\text{relu}(x) = \begin{cases} 0 \text{ if } \leq 0 \\ x \text{ otherwise} \end{cases} \qquad \text{relu}'(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x > 0 \\ \text{undefined otherwise} \end{cases}$$

- No vanishing gradient issue
- "Dead units" problem (i.e. $b_i << 0$)
- Popular in Computer Vision (very deep networks)

Parameters initialization

What do we want?

- Values close to 0 prevent gradient vanishing
  (or gradient exploding/disappearing in the case of relu)
- Gradient magnitude approximately similar for all layers
  (to prevent that a subset of layers do all the works while others are useless)

Parameters initialization

What do we want?

- Values close to 0 prevent gradient vanishing
  (or gradient exploding/disappearing in the case of relu)
- Gradient magnitude approximately similar for all layers
  (to prevent that a subset of layers do all the works while others are useless)

Hyperbolic tangent

Let $\boldsymbol{W} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{R}^{m}$:

- $\boldsymbol{W} \sim \mathcal{U}\left[-\frac{\sqrt{6}}{\sqrt{m+n}}, +\frac{\sqrt{6}}{\sqrt{m+n}}\right]$
- $\boldsymbol{b} = 0$

Usually called Xavier or Glorot initialization
[Glorot and Bengio, 2010]

Rectified Linear Unit

Let $\boldsymbol{W} \in \mathbb{R}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{R}^{m}$:

- $\boldsymbol{W} \sim \mathcal{U}\left[-\frac{\sqrt{6}}{\sqrt{n}}, +\frac{\sqrt{6}}{\sqrt{n}}\right]$
- $\boldsymbol{b} = 0$
  (or $\boldsymbol{b} = 0.01$ to prevent dying units)

Usually called Kaiming or He initialization
[He et al., 2015]

Regularization

Generalization

Overparameterized neural networks

Networks where the number of parameters exceed the training dataset size.

- Can learn by heart the dataset,
  i.e. **overfit the data** $\rightarrow$ does not generalize well to unseen data
- Are easier to optimize in practice

Generalization

Overparameterized neural networks

Networks where the number of parameters exceed the training dataset size.

- Can learn by heart the dataset,
  i.e. **overfit the data** $\rightarrow$ does not generalize well to unseen data
- Are easier to optimize in practice

Monitoring the training process

- Loss should go down $\Rightarrow$ otherwise your step-size is probably too big!
- Training accuracy should go up
- Dev accuracy should go up $\Rightarrow$ otherwise the network is overfitting!

Generalization

Overparameterized neural networks

Networks where the number of parameters exceed the training dataset size.

- Can learn by heart the dataset,
  i.e. **overfit the data** $\rightarrow$ does not generalize well to unseen data
- Are easier to optimize in practice

Monitoring the training process

- Loss should go down $\Rightarrow$ otherwise your step-size is probably too big!
- Training accuracy should go up
- Dev accuracy should go up $\Rightarrow$ otherwise the network is overfitting!

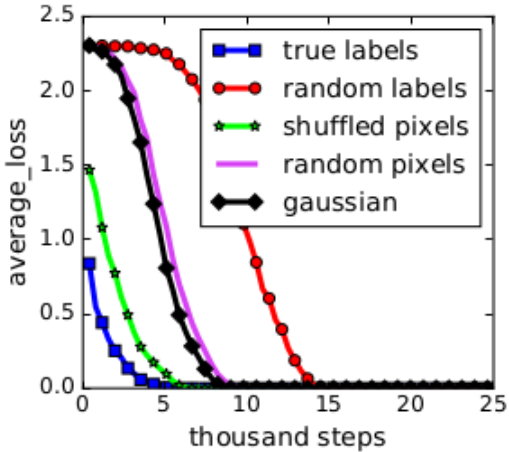
Regularization

Techniques to control parameters during learning and prevent overfitting

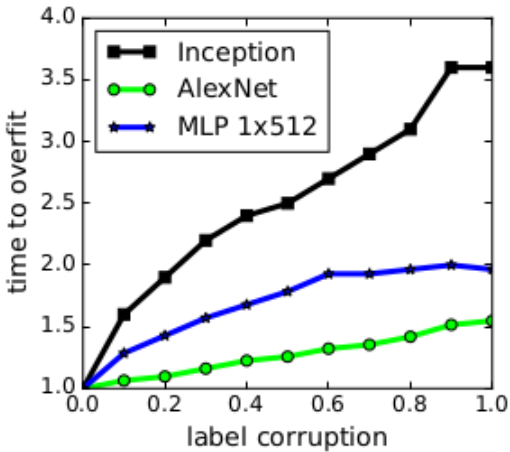
Learning with random inputs and labels 1/2 [Zhang et al., 2017]

model	# params	random crop	weight decay	train accuracy	test accuracy
Inception	1,649,402	yes	yes	100.0	89.05
		yes	no	100.0	89.31
		no	yes	100.0	86.03
		no	no	100.0	85.75
(fitting random labels)		no	no	100.0	9.78
Inception w/o BatchNorm	1,649,402	no	yes	100.0	83.00
		no	no	100.0	82.00
(fitting random labels)		no	no	100.0	10.12
Alexnet	1,387,786	yes	yes	99.90	81.22
		yes	no	99.82	79.66
		no	yes	100.0	77.36
		no	no	100.0	76.07
(fitting random labels)		no	no	99.82	9.86
MLP 3x512	1,735,178	no	yes	100.0	53.35
		no	no	100.0	52.39
(fitting random labels)		no	no	100.0	10.48
MLP 1x512	1,209,866	no	yes	99.80	50.39
		no	no	100.0	50.51
(fitting random labels)		no	no	99.34	10.61

Learning with random inputs and labels 2/2 [Zhang et al., 2017]



(a) learning curves



(b) convergence slowdown

Regularization $L2$ or Gaussian prior or weight decay 1/3

$$\begin{aligned}\hat{\theta} &= \arg\min_{\theta} \mathcal{L}(f(x;\theta), y) \quad + \quad \frac{\lambda}{2}||\boldsymbol{\theta}||^2 \\ &= \arg\min_{\theta} \mathcal{L}(f(x;\theta), y) \quad + \quad \mathcal{R}(\theta;\lambda)\end{aligned}$$

Regularization term

The second term $\mathcal{R}(\theta;\lambda)$ is a $L2$ regularization term which can be equivalently interpreted as:

- a soft constraint on the magnitude of parameters
- a Gaussian prior on parameters: $\mathcal{N}(0, 1/\lambda)$
- re-scaling the parameters after each update (weight decay)

Regularization $L2$ or Gaussian prior or weight decay 2/3

$$\begin{aligned}\hat{\theta} &= \arg\min_{\theta} \mathcal{L}(f(x;\theta), y) \quad + \quad \frac{\lambda}{2}||\boldsymbol{\theta}||^2 \\ &= \arg\min_{\theta} \mathcal{L}(f(x;\theta), y) \quad + \quad \mathcal{R}(\theta;\lambda)\end{aligned}$$

Gradient update

$$\begin{aligned}\boldsymbol{\theta} &= \boldsymbol{\theta} - \epsilon\nabla_\theta\mathcal{L} - \epsilon\nabla_\theta\mathcal{R} \\ &= \boldsymbol{\theta} - \epsilon(\nabla_\theta\mathcal{L} - \nabla_\theta\mathcal{R})\end{aligned}$$

Regularization $L2$ or Gaussian prior or weight decay 2/3

$$\begin{aligned}\hat{\theta} &= \arg\min_{\theta} \mathcal{L}(f(x;\theta), y) \quad + \quad \frac{\lambda}{2}||\boldsymbol{\theta}||^2 \\ &= \arg\min_{\theta} \mathcal{L}(f(x;\theta), y) \quad + \quad \mathcal{R}(\theta;\lambda)\end{aligned}$$

Gradient update

$$\begin{aligned}\boldsymbol{\theta} &= \boldsymbol{\theta} - \epsilon\nabla_\theta\mathcal{L} - \epsilon\nabla_\theta\mathcal{R} \\ &= \boldsymbol{\theta} - \epsilon(\nabla_\theta\mathcal{L} - \nabla_\theta\mathcal{R})\end{aligned}$$

What does the gradient of the regularizer look like?

Let b be a a parameter of the network:

$$\frac{\partial}{\partial b}\mathcal{R} = \frac{\partial}{\partial b}\frac{\lambda}{2}||\theta||^2 = \frac{\lambda}{2}2b = \lambda b$$

Regularization $L2$ or Gaussian prior or weight decay 3/3

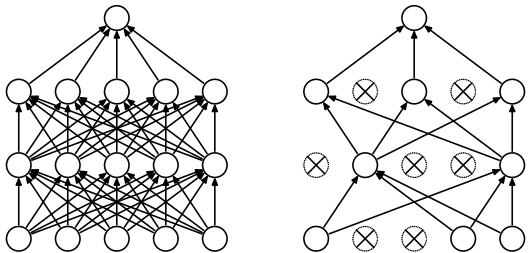
Implementation from Pytorch (slightly modified):

```
class SGD(Optimizer):
  def step(self, closure=None):
    """Performs a single optimization step."""
    for group in self.param_groups:
      for p in group['params']:
        if p.grad is None:
          continue

        d_p = p.grad.data  # get gradient
        weight_decay = group['weight_decay']
        if weight_decay != 0:
          d_p.add_(weight_decay, p.data)  # add weight decay to the gradient

        p.data.add_(-group['lr'], d_p)  # update parameters
```

Dropout 1/4 [Hinton et al., 2012, Srivastava et al., 2014]

How does dropout work?

- During training, we randomly "turn off" neurons,
  i.e. we randomly set elements of hidden layers $\boldsymbol{z}$ to 0
- During test, we do use the full network

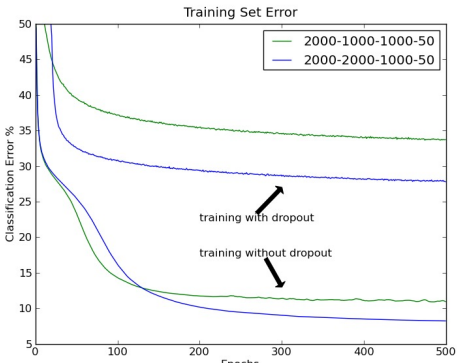


(a) Standard Neural Net (b) After applying dropout.

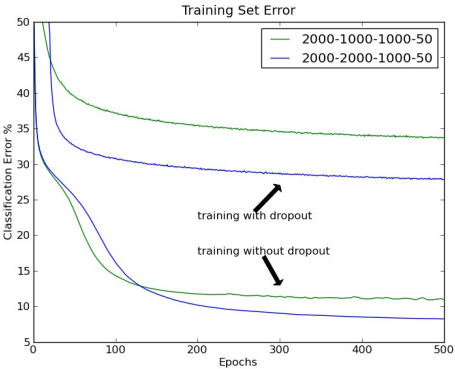
Intuition

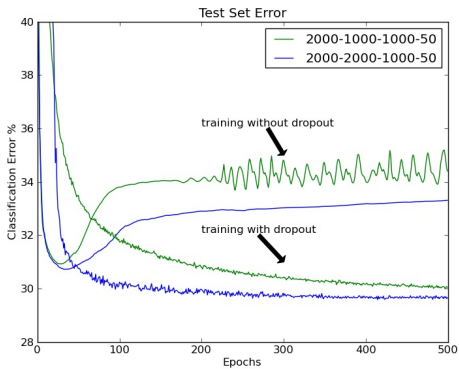
- prevents co-adaptation between units
- equivalent to averaging different models that have different structure but share parameters

Dropout 2/4 [Hinton et al., 2012]



Dropout 2/4 [Hinton et al., 2012]





Dropout 3/4

Dropout layer

A dropout layer is parameterized by the probability of "turning off" a neuron $p \in [0, 1]$:

$$\boldsymbol{z}' = \text{Dropout}(\boldsymbol{z}; p = 0.5)$$

Dropout 3/4

Dropout layer

A dropout layer is parameterized by the probability of "turning off" a neuron $p \in [0, 1]$:

$$\boldsymbol{z}' = \text{Dropout}(\boldsymbol{z}; p = 0.5)$$

Implementation

- $\boldsymbol{z} \in \mathbb{R}^n$: output of a hidden layer
- $p \in [0, 1]$: dropout probability
- $\boldsymbol{m} \in \{0, 1\}^n$: mask vector
- $\boldsymbol{z}'$: hidden values after dropout application

The mask $\boldsymbol{m}$ is a vector of booleans stating if neurons z_i is kept ($m_i = 1$) or "turned off" ($m_i = 0$).

Dropout 3/4

Dropout layer

A dropout layer is parameterized by the probability of "turning off" a neuron $p \in [0, 1]$:

$$\boldsymbol{z}' = \text{Dropout}(\boldsymbol{z}; p = 0.5)$$

Implementation

- $\boldsymbol{z} \in \mathbb{R}^n$: output of a hidden layer
- $p \in [0, 1]$: dropout probability
- $\boldsymbol{m} \in \{0, 1\}^n$: mask vector
- $\boldsymbol{z}'$: hidden values after dropout application

Forward pass:

$$\boldsymbol{m} \sim \text{Bernoulli}(1 - p)$$
$$z'_i = \frac{z_i * m_i}{1 - p}$$

The mask $\boldsymbol{m}$ is a vector of booleans stating if neurons z_i is kept ($m_i = 1$) or "turned off" ($m_i = 0$).

Dropout 3/4

Dropout layer

A dropout layer is parameterized by the probability of "turning off" a neuron $p \in [0, 1]$:

$$\boldsymbol{z}' = \text{Dropout}(\boldsymbol{z}; p = 0.5)$$

Implementation

- $\boldsymbol{z} \in \mathbb{R}^n$: output of a hidden layer
- $p \in [0, 1]$: dropout probability
- $\boldsymbol{m} \in \{0, 1\}^n$: mask vector
- $\boldsymbol{z}'$: hidden values after dropout application

Forward pass:

$$\boldsymbol{m} \sim \text{Bernoulli}(1 - p)$$
$$z'_i = \frac{z_i * m_i}{1 - p}$$

Backward pass:

$$\frac{\partial z'_i}{z_i} = \frac{m}{1 - p}$$

$\Rightarrow$ no gradient for "turned off" neurons.

The mask $\boldsymbol{m}$ is a vector of booleans stating if neurons z_i is kept ($m_i = 1$) or "turned off" ($m_i = 0$).

Dropout 4/4

Where do you apply dropout?

- On the input of the neural network $\boldsymbol{x}$
- **After** activation functions ($\sigma(0) \neq 0$)
- **Do not** apply dropout on the output logits

Dropout 4/4

Where do you apply dropout?

- On the input of the neural network $\boldsymbol{x}$
- **After** activation functions ($\sigma(0) \neq 0$)
- **Do not** apply dropout on the output logits

Which dropout probability should you use?

- Empirical question: you have to test!
- Dropout probability at different layers can be different (especially input vs. hidden layers)
- Usually $0.1 \leq p \leq 0.5$

Dropout variants

Dropout can be applied differently for special neural network architectures (e.g. convolutions, recurrent neural networks)

Better optimizers

Stochastic Gradient Descent (SGD)

$$\theta^{(t+1)} = \theta^{(t)} - \epsilon^{(t)} \nabla_\theta \mathcal{L}$$

Advantages

- Simple
- Single hyper-parameter: the step-size ϵ

Stochastic Gradient Descent (SGD)

$$\theta^{(t+1)} = \theta^{(t)} - \epsilon^{(t)} \nabla_\theta \mathcal{L}$$

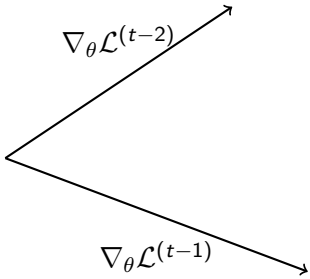
Advantages

- Simple
- Single hyper-parameter: the step-size ϵ

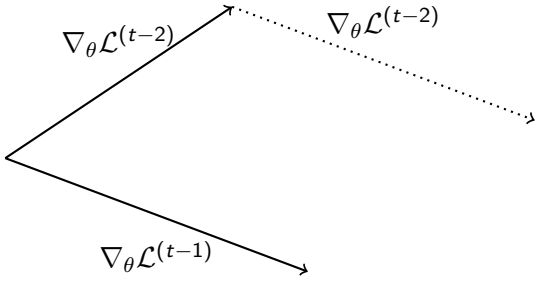
Downsides

- Forget information about previous updates
- Require to search for the best step-size strategy
- Require step-size annealing in practice: how? what scaling factor?
- Based on first-order information only
  (i.e. the curvature of the optimized function is ignored)

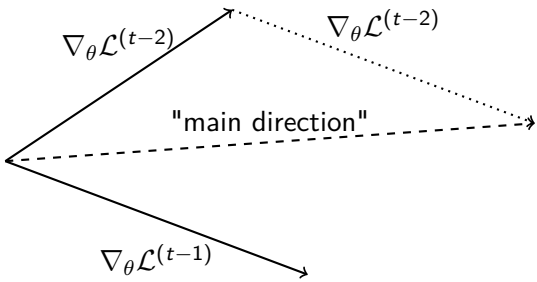
Momentum 1/3



Momentum 1/3


$\nabla_\theta \mathcal{L}^{(t-2)}$
$\nabla_\theta \mathcal{L}^{(t-2)}$
$\nabla_\theta \mathcal{L}^{(t-1)}$

Momentum 1/3


$\nabla_\theta \mathcal{L}^{(t-2)}$
$\nabla_\theta \mathcal{L}^{(t-2)}$
"main direction"
$\nabla_\theta \mathcal{L}^{(t-1)}$

Momentum 2/3

[Polyak, 1964]

- γ: velocity of parameters, i.e. cumulative information about past gradients
- $\mu \in [0, 1]$: momentum, i.e. how much information must be preserved?

$$\gamma^{(t+1)} = \mu\gamma^{(t)} + \nabla_\theta \mathcal{L}$$
$$\theta^{(t+1)} = \theta^{(t)} - \epsilon\gamma^{(t+1)}$$

Variants

- Gradient dampening, i.e. diminish the contribution of the current gradient
- Nesterov's Accelerated Gradient [Sutskever et al., 2013]

Momentum 3/3

Implementation from Pytorch (slightly modified):

```
for group in self.param_groups:
  for p in group['params']:
    if p.grad is None:
      continue

  d_p = p.grad.data  # get the gradient
  if momentum != 0:
    param_state = self.state[p]
    if 'momentum_buffer' not in param_state:  # initialize velocity vector
      buf = param_state['momentum_buffer'] = torch.clone(d_p).detach()
    else:
      buf = param_state['momentum_buffer']  # retrieve velocity vector
      buf.mul_(momentum).add_(d_p)  # update velocity vector
  d_p = buf

  p.data.add_(-group['lr'], d_p)  # update parameters
```

Adaptive learning rates 1/2

Adagrad [Duchi et al., 2011]

- Replace global step-size with dynamic per parameter step-size + global learning rate
- The dynamic per parameter step-size is computed w.r.t. previous gradient l2-norm
  $\Rightarrow$ parameters with small (resp. large) gradient will have a large (resp. small) step-size

Adaptive learning rates 1/2

Adagrad [Duchi et al., 2011]

- Replace global step-size with dynamic per parameter step-size + global learning rate
- The dynamic per parameter step-size is computed w.r.t. previous gradient l2-norm
  $\Rightarrow$ parameters with small (resp. large) gradient will have a large (resp. small) step-size

Adadelta [Zeiler, 2012]

- Dynamic per parameter rate is computed with a fixed window of past gradients
- Approximate second-order information to incorporate curvature information
  $\Rightarrow$ less sensitive to the learning rate hyper-parameter!

Adaptive learning rates 1/2

Adagrad [Duchi et al., 2011]

- Replace global step-size with dynamic per parameter step-size + global learning rate
- The dynamic per parameter step-size is computed w.r.t. previous gradient l2-norm
  $\Rightarrow$ parameters with small (resp. large) gradient will have a large (resp. small) step-size

Adadelta [Zeiler, 2012]

- Dynamic per parameter rate is computed with a fixed window of past gradients
- Approximate second-order information to incorporate curvature information
  $\Rightarrow$ less sensitive to the learning rate hyper-parameter!

	SGD	MOMENTUM	ADAGRAD
$\epsilon = 1e^{0}$	**2.26%**	89.68%	43.76%
$\epsilon = 1e^{-1}$	2.51%	**2.03%**	2.82%
$\epsilon = 1e^{-2}$	7.02%	2.68%	**1.79%**
$\epsilon = 1e^{-3}$	17.01%	6.98%	5.21%
$\epsilon = 1e^{-4}$	58.10%	16.98%	12.59%

Table 1. MNIST test error rates after 6 epochs of training for various hyperparameter settings using SGD, MOMENTUM, and ADAGRAD.

Adaptive learning rates 1/2

Adagrad [Duchi et al., 2011]

- Replace global step-size with dynamic per parameter step-size + global learning rate
- The dynamic per parameter step-size is computed w.r.t. previous gradient l2-norm
  $\Rightarrow$ parameters with small (resp. large) gradient will have a large (resp. small) step-size

Adadelta [Zeiler, 2012]

- Dynamic per parameter rate is computed with a fixed window of past gradients
- Approximate second-order information to incorporate curvature information
  $\Rightarrow$ less sensitive to the learning rate hyper-parameter!

	SGD	MOMENTUM	ADAGRAD
$\epsilon = 1e^{0}$	**2.26%**	89.68%	43.76%
$\epsilon = 1e^{-1}$	2.51%	**2.03%**	2.82%
$\epsilon = 1e^{-2}$	7.02%	2.68%	**1.79%**
$\epsilon = 1e^{-3}$	17.01%	6.98%	5.21%
$\epsilon = 1e^{-4}$	58.10%	16.98%	12.59%

Table 1. MNIST test error rates after 6 epochs of training for various hyperparameter settings using SGD, MOMENTUM, and ADAGRAD.

	$\rho = 0.9$	$\rho = 0.95$	$\rho = 0.99$
$\epsilon = 1e^{-2}$	2.59%	2.58%	2.32%
$\epsilon = 1e^{-4}$	2.05%	1.99%	2.28%
$\epsilon = 1e^{-6}$	1.90%	**1.83%**	2.05%
$\epsilon = 1e^{-8}$	2.29%	2.13%	2.00%

Table 2. MNIST test error rate after 6 epochs for various hyperparameter settings using ADADELTA.

Adaptive learning rate 2/2

Adam [Kingma and Ba, 2015]

- Combine dynamic per parameter learning rate and momentum
- Initialization bias correction

Convergence issue but works very well in practice [Reddi et al., 2018]
Variants: AdaMax, Nadam [Dozat, 2016], Radam [Liu et al., 2019], AMSGrad

Adaptive learning rate 2/2

Adam [Kingma and Ba, 2015]

- Combine dynamic per parameter learning rate and momentum
- Initialization bias correction

Convergence issue but works very well in practice [Reddi et al., 2018]
Variants: AdaMax, Nadam [Dozat, 2016], Radam [Liu et al., 2019], AMSGrad

Rule of thumb

- Optimizers based on adaptive learning rates usually work out of the box
  e.g. Adam is really popular in Natural Language Processing
- Fine-tuned SGD with step-size annealing can provide better results at the cost of expensive hyper-parameter tuning

Regularization issue

Weight decay is not equivalent to l2-norm when using adaptive learning rates!

References I


Dozat, T. (2016).
Incorporating nesterov momentum into adam.
ICLR Workshop.


Duchi, J., Hazan, E., and Singer, Y. (2011).
Adaptive subgradient methods for online learning and stochastic optimization.
Journal of Machine Learning Research, 12(Jul):2121–2159.


Glorot, X. and Bengio, Y. (2010).
Understanding the difficulty of training deep feedforward neural networks.
In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR.

References II


He, K., Zhang, X., Ren, S., and Sun, J. (2015).
Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.
In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 1026–1034, Washington, DC, USA. IEEE Computer Society.


Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012).
Improving neural networks by preventing co-adaptation of feature detectors.
CoRR, abs/1207.0580.


Kingma, D. P. and Ba, J. (2015).
Adam: A method for stochastic optimization.
ICLR.

References III


Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. (2009).
Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations.
pages 609–616.


Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., and Han, J. (2019).
On the variance of the adaptive learning rate and beyond.
arXiv preprint arXiv:1908.03265.


Nagaraj, D., Jain, P., and Netrapalli, P. (2019).
SGD without replacement: Sharper rates for general smooth convex functions.
In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4703–4711, Long Beach, California, USA. PMLR.

References IV


Polyak, B. T. (1964).
Some methods of speeding up the convergence of iteration methods.
USSR Computational Mathematics and Mathematical Physics, 4(5):1–17.


Reddi, S. J., Kale, S., and Kumar, S. (2018).
On the convergence of adam and beyond.
ICLR.


Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014).
Dropout: A simple way to prevent neural networks from overfitting.
Journal of Machine Learning Research, 15:1929–1958.

References V


Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013).
On the importance of initialization and momentum in deep learning.
In Dasgupta, S. and McAllester, D., editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA. PMLR.


Voita, E., Talbot, D., Moiseev, F., Sennrich, R., and Titov, I. (2019).
Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned.
In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, Florence, Italy. Association for Computational Linguistics.


Zeiler, M. D. (2012).
Adadelta: an adaptive learning rate method.
arXiv preprint arXiv:1212.5701.

References VI


Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2017).
Understanding deep learning requires rethinking generalization.
ICLR 2017.