APL 745
Backpropagation

Dr. Rajdip Nayek

Block V, Room 418D

Department of Applied Mechanics

Indian Institute of Technology Delhi

E-mail: rajdipn@am.iitd.ac.in

Overview

- We've seen that multilayer neural networks are powerful. But how to actually learn (or optimize) the weights and biases?

Update formula

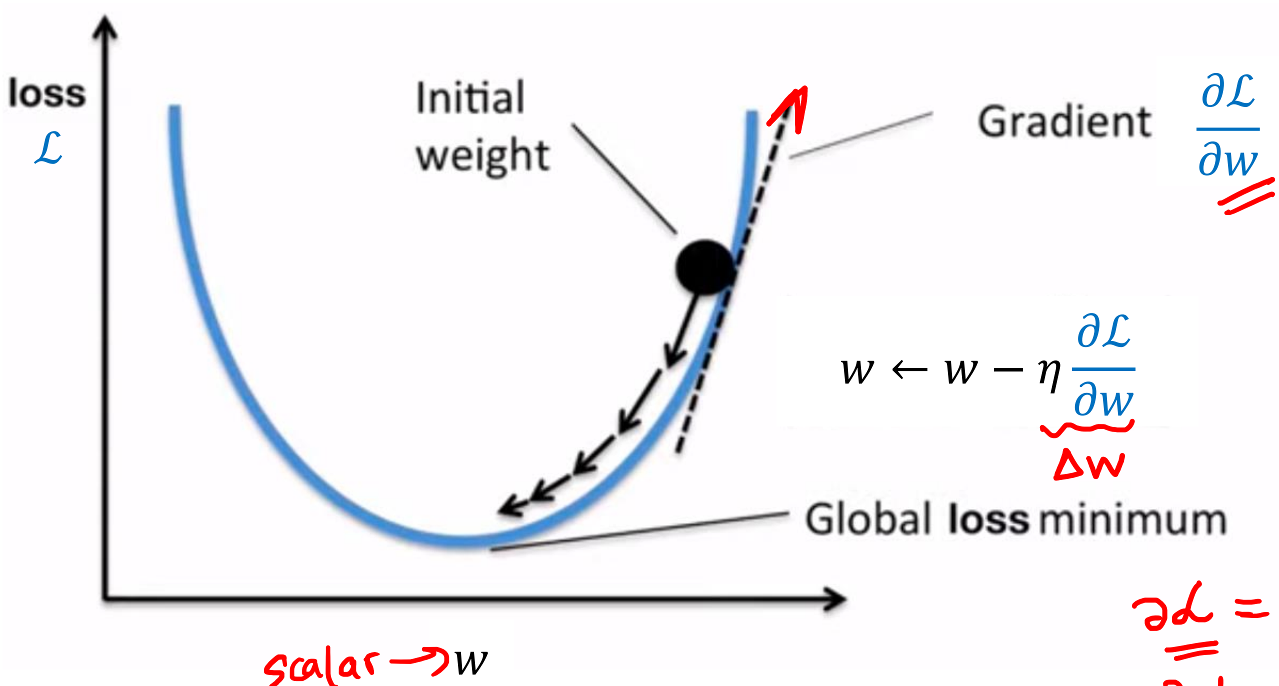
$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}$$

- **Backpropagation** is the central algorithm that is **used for computing gradients**
    - Backpropagation is an efficient use of the **Chain Rule for derivatives**
    - It's an instance of **reverse mode automatic differentiation**

N.B. Lecture slides mostly follow the material of CSC421 by Roger Grosse and Jimmy Ba

Recap: Gradient descent

- Recall that gradient descent moves opposite to the direction of gradient



$$w \leftarrow w - \eta \underbrace{\frac{\partial \mathcal{L}}{\partial w}}_{\Delta w}$$

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} \left(t^{(i)} - y^{(i)}\right)^2$$

- We want to compute the gradient of the loss function $\partial \mathcal{L} / \partial \mathbf{w}$, which is the vector of partial derivatives and is calculated **by averaging over all training examples**
- In this lecture, we will focus on computing the gradients of loss function for a **single training example**

Univariate Chain Rule

- We already know of the univariate chain rule

- Recall if $f(x)$ and $x(t)$ are univariate functions, then

$$\frac{d}{dt} f(x(t)) = \frac{df}{dx} \frac{dx}{dt}$$

Chain rule

- Consider a univariate logistic least-squares model

Supervised

(x, t)

input

Preactivation $z = wx + b$

Model pred $y = \sigma(z)$

output

loss $\ell = \frac{1}{2}(t - y)^2$ ← single training example

- Let's compute the loss derivatives

Univariate Chain Rule

- Let's compute the loss derivatives w.r.t. w and b using calculus

t - y

$$\ell = \frac{1}{2}\left(t - \underline{\sigma\left(wx+b\right)}\right)^2$$

$$\begin{aligned}
\frac{\partial \ell}{\partial w} &= \frac{\partial}{\partial w}\left[\frac{1}{2}\left(t-\sigma\left(wx+b\right)\right)^2\right] \\
&= \frac{1}{2}\frac{\partial}{\partial w}\left(t-\sigma\left(wx+b\right)\right)^2 \\
&= \left(t-\sigma\left(wx+b\right)\right)\frac{\partial}{\partial w}\left(t-\sigma\left(wx+b\right)\right) \\
&= -\left(t-\sigma\left(wx+b\right)\right)\,\sigma'\left(wx+b\right)\frac{\partial}{\partial w}\left(wx+b\right) \\
&= -\left(t-\sigma\left(\underline{wx+b}\right)\right)\,\sigma'\left(\underline{wx+b}\right)x
\end{aligned}$$

$$\begin{aligned}
\frac{\partial \ell}{\partial b} &= \frac{\partial}{\partial b}\left[\frac{1}{2}\left(t-\sigma\left(wx+b\right)\right)^2\right] \\
&= \frac{1}{2}\frac{\partial}{\partial b}\left(t-\sigma\left(wx+b\right)\right)^2 \\
&= \left(t-\sigma\left(wx+b\right)\right)\frac{\partial}{\partial b}\left(t-\sigma\left(wx+b\right)\right) \\
&= -\left(t-\sigma\left(wx+b\right)\right)\,\sigma'\left(wx+b\right)\frac{\partial}{\partial b}\left(wx+b\right) \\
&= -\left(t-\sigma\left(wx+b\right)\right)\,\sigma'\left(wx+b\right)
\end{aligned}$$

What are the disadvantages of this approach?

- The calculations are very cumbersome. In this derivation, we had to copy lots of terms from one line to the next
- The final expressions have lots of repeated terms

Univariate Chain Rule

A more structured way of doing it

1) Compute the loss:

$$z = wx + b$$

$$y = \sigma(z)$$

$$\ell = \frac{1}{2}(t - y)^2$$

2) Compute the derivatives:

$$\frac{\partial \ell}{\partial y} = -(t - y)$$

obtained from previous step

$$\frac{\partial \ell}{\partial z} = \frac{\partial \ell}{\partial y}\frac{dy}{dz} = \frac{\partial \ell}{\partial y}\sigma'(z)$$

evaluated at current step

$$\frac{\partial \ell}{\partial w} = \frac{\partial \ell}{\partial z}\frac{\partial z}{\partial w} = \frac{\partial \ell}{\partial z}x$$

$$\frac{\partial \ell}{\partial b} = \frac{\partial \ell}{\partial z}\frac{\partial z}{\partial b} = \frac{\partial \ell}{\partial z}$$

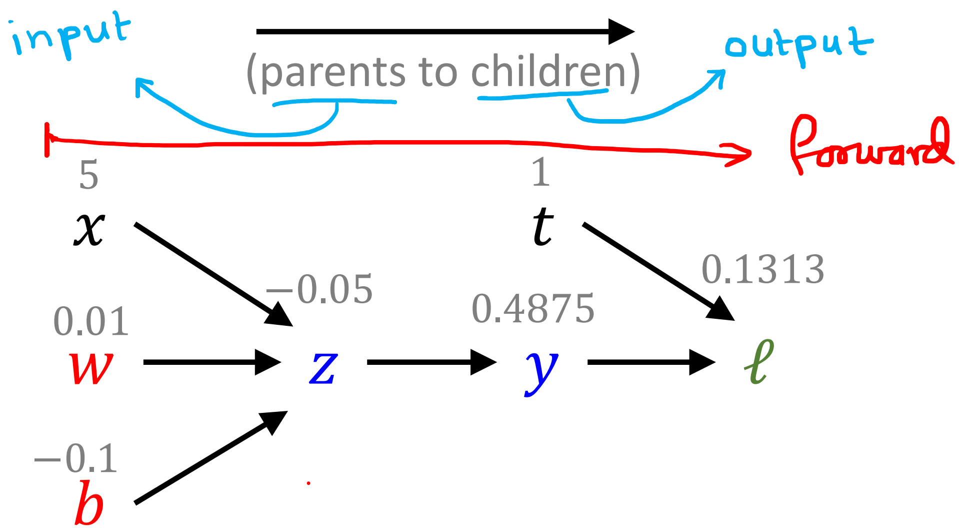
This form of computation is clean!

- No repeated expressions

Univariate Chain Rule

- We can plot these computations using a **computation graph**
- The **nodes** represent all the inputs and computed quantities
- The **edges** represent which nodes are computed directly as a function of other nodes

1) Compute loss



1) Compute loss

$$z = wx + b$$

$$y = \sigma(z)$$

$$\ell = \frac{1}{2}(t - y)^2$$

Univariate Chain Rule

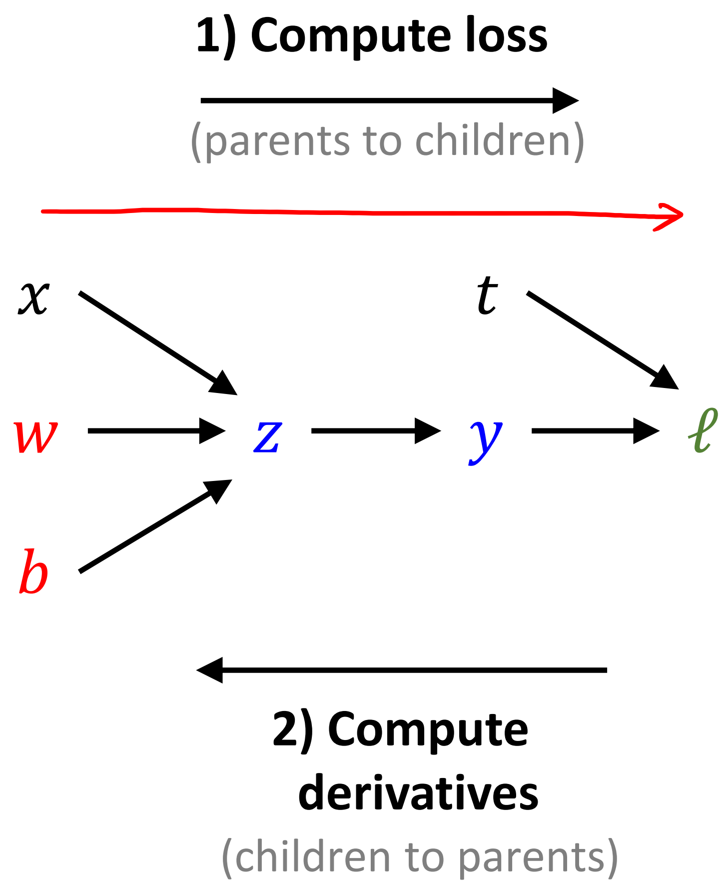
- We can plot these computations using a **computation graph**
- The **nodes** represent all the inputs and computed quantities
- The **edges** represent which nodes are computed directly as a function of other nodes

1) Compute loss

$$z = wx + b$$

$$y = \sigma(z)$$

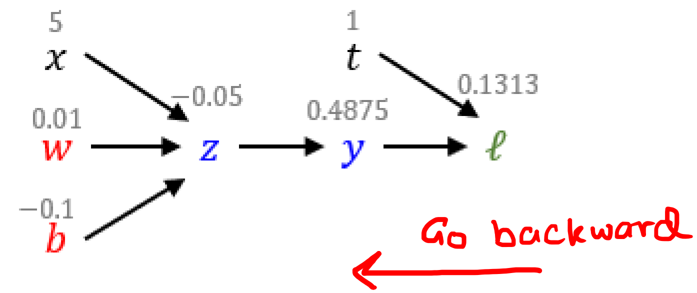
$$\ell = \frac{1}{2}(t - y)^2$$



Univariate Chain Rule

A weird but slightly convenient notation

- Usual notation: $\nabla_v \ell = \frac{\partial \ell}{\partial v}$
- Instead, use $\bar{v} = \nabla_v \ \ell = \frac{\partial \ell}{\partial v}$



2) Compute the derivatives:

1) Compute the loss:

$$z = wx + b$$

$$y = \sigma(z)$$

$$\ell = \frac{1}{2}(t - y)^2$$

$$\frac{\partial \ell}{\partial y} = -(t - y)$$

$$\frac{\partial \ell}{\partial z} = \frac{\partial \ell}{\partial y}\frac{dy}{dz} = \frac{\partial \ell}{\partial y}\sigma'(z)$$

$$\frac{\partial \ell}{\partial w} = \frac{\partial \ell}{\partial z}\frac{\partial z}{\partial w} = \frac{\partial \ell}{\partial z}x$$

$$\frac{\partial \ell}{\partial b} = \frac{\partial \ell}{\partial z}\frac{\partial z}{\partial b} = \frac{\partial \ell}{\partial z}$$

$$\bar{y} = -(t - y)$$

$$\bar{z} = \bar{y}\,\sigma'(z)$$

$$\bar{w} = \bar{z}\,x$$

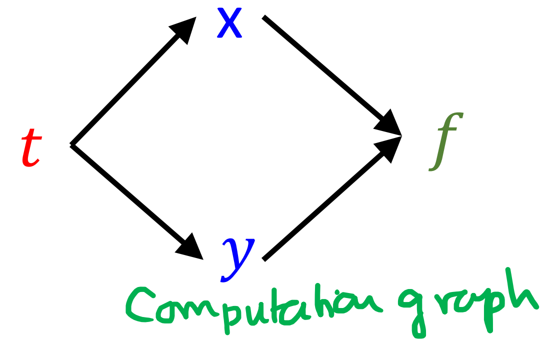
$$\bar{b} = \bar{z}$$

Note: $\bar{w}$ used here should not be confused with augmented $\bar{w}$ used in linear regression/classification

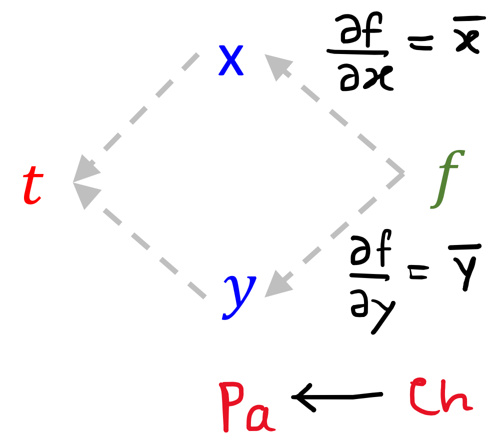
Multivariate Chain Rule

- Suppose we have a function $f(x(t), y(t))$. Then

$$\frac{d}{dt} f\left(x(t), y(t)\right) = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$



- In the context of backpropagation (backward pass)



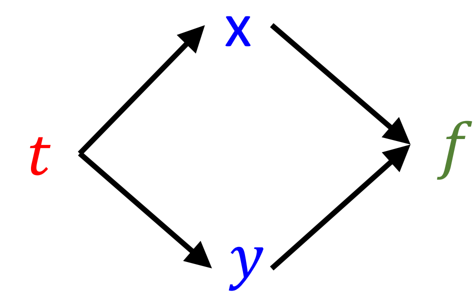
$$\frac{d}{dt} f\left(x(t), y(t)\right) = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$

Values computed first

Multivariate Chain Rule

- Suppose we have a function $f(x(t), y(t))$. Then

$$\frac{d}{dt} f\left(x(t), y(t)\right) = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$

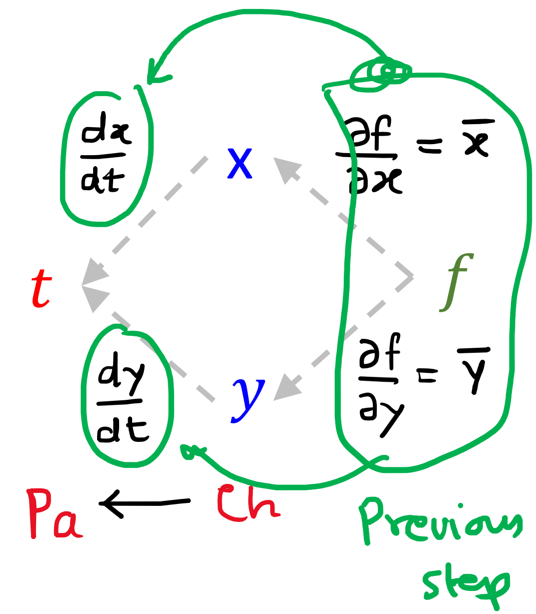


- In the context of backpropagation (backward pass)

Mathematical expressions to be evaluated

$$\frac{d}{dt} f\left(x(t), y(t)\right) = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$

Values already computed



- In our notation:

$$\frac{\partial f}{\partial t} \rightarrow \bar{t} = \overset{\frac{\partial f}{\partial x}}{\bar{x}}\frac{dx}{dt} + \overset{\frac{\partial f}{\partial y}}{\bar{y}}\frac{dy}{dt}$$

Backpropagation

Full backpropagation algorithm:

Let $\boxed{v_1, \cdots, v_N}$ be a topological ordering of the computation graph (i.e. where parents come before children)

v_N denotes the variable we are trying to compute derivatives of (e.g. loss function)

Forward pass
Compute values

For $i = 1, \ldots, N$

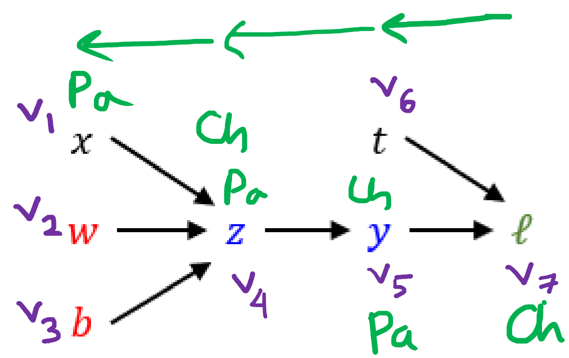
Compute v_i as a function of Pa(v_i)

Backward pass
Compute derivatives

$$\bar{v}_N = \frac{dv_N}{dv_N} = 1$$

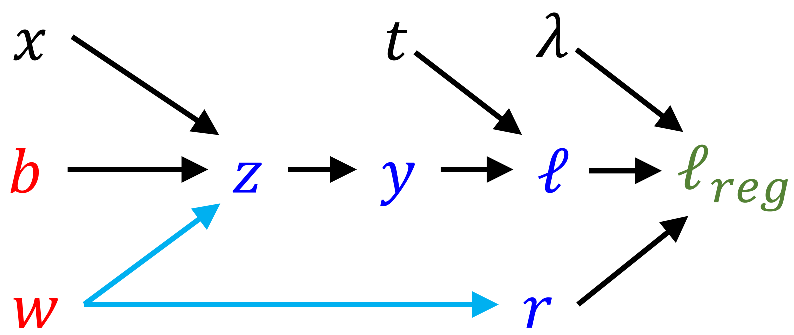
For $i = N-1, \ldots, 1$

$$\bar{v}_i = \sum_{j \in \mathrm{Ch}(\bar{v}_i)} \bar{v}_j \frac{\partial v_j}{\partial v_i}$$



Backpropagation

Example: Logistic least-squares regression



Forward pass

Compute values

$$z = wx + b$$
$$y = \sigma(z)$$
$$\ell = \frac{1}{2}(t-y)^2$$
$$r = \frac{1}{2}w^2$$
$$\ell_{reg} = \ell + \lambda r$$

$$\boxed{\bar{v}_i = \frac{\partial \ell_{reg}}{\partial v_i}}$$

Backward pass

Compute derivatives

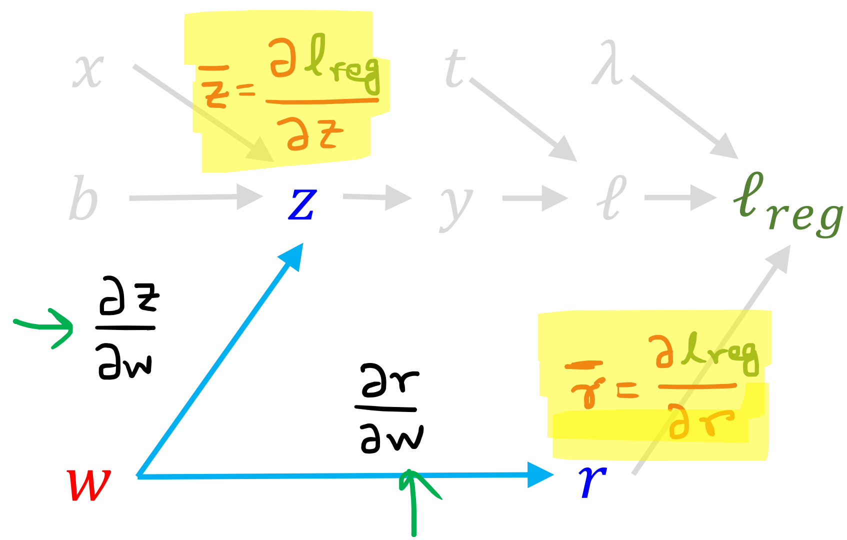
$$\bar{\ell}_{reg} = 1$$
$$\bar{r} = \bar{\ell}_{reg}\frac{\partial \ell_{reg}}{\partial r} = \bar{\ell}_{reg}\lambda$$
$$\frac{\partial \ell_{reg}}{\partial \ell} = \bar{\ell} = \bar{\ell}_{reg}\frac{\partial \ell_{reg}}{\partial \ell} = \bar{\ell}_{reg}$$
$$\frac{\partial \ell_{reg}}{\partial y} = \bar{y} = \bar{\ell}\frac{\partial \ell}{\partial y} = -\bar{\ell}\,(t-y)$$

$$\bar{z} = \bar{y}\frac{dy}{dz} = \bar{y}\sigma'(z)$$
$$\bar{b} = \bar{z}\frac{\partial z}{\partial b} = \bar{z}$$
$$\bar{w} = \bar{z}\frac{\partial z}{\partial w} + \bar{r}\frac{\partial r}{\partial w} = \bar{z}x + \bar{r}w$$

Backpropagation

Example: Logistic least-squares regression

$$\overline{v_i} = \frac{\partial \ell_{reg}}{\partial v_i}$$



$$\overline{v_i} = \frac{\partial \ell_{reg}}{\partial v_i}$$

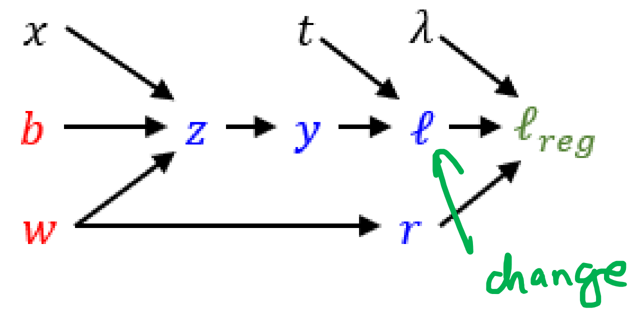
$$\begin{aligned}
z &= wx + b \\
y &= \sigma(z) \\
\ell &= \frac{1}{2}(t - y)^2 \\
r &= \frac{1}{2}w^2 \\
\ell_{reg} &= \ell + \lambda r
\end{aligned}$$

$$\frac{\partial \ell_{reg}}{\partial w} = \overline{w} = \overline{z}\,\frac{\partial z}{\partial w} + \overline{r}\,\frac{\partial r}{\partial w}$$

$$= \overline{z}x + \overline{r}w$$

Backpropagation

Example: Logistic least-squares regression



LS ↓ CE

Derivatives via backprop

$$\overline{\ell_{reg}} = 1$$

$$\bar{r} = \overline{\ell_{reg}} \frac{\partial \ell_{reg}}{\partial r} = \overline{\ell_{reg}} \lambda$$

$$\bar{\ell} = \overline{\ell_{reg}} \frac{\partial \ell_{reg}}{\partial \ell} = \overline{\ell_{reg}}$$

$$\bar{y} = \bar{\ell} \frac{\partial \ell}{\partial y} = -\bar{\ell}\,(t - y)$$

$$\frac{\partial \ell_{reg}}{\partial \ell}$$

$$\bar{z} = \bar{y} \frac{dy}{dz} = \bar{y} \sigma'(z)$$

$$\bar{b} = \bar{z} \frac{\partial z}{\partial b} = \bar{z}$$

$$\bar{w} = \bar{z} \frac{\partial z}{\partial w} + \bar{r} \frac{\partial r}{\partial w} = \bar{z} x + \bar{r} w$$

$$\bar{v} = \frac{\partial \ell_{reg}}{\partial v}$$

- The derivation, and the final result, are **much cleaner** and **efficient**. There are no redundant computations here.
- The procedure is ***modular***: it is broken down into small chunks that can be reused for other computations. For instance, if we want to change the loss function, we'd only have to modify the formula for $\bar{y}$

Backpropagation for MLP

Multilayer Perceptron (multiple outputs)

One hidden layer

Want to compute gradients of loss function wrt all weights and biases

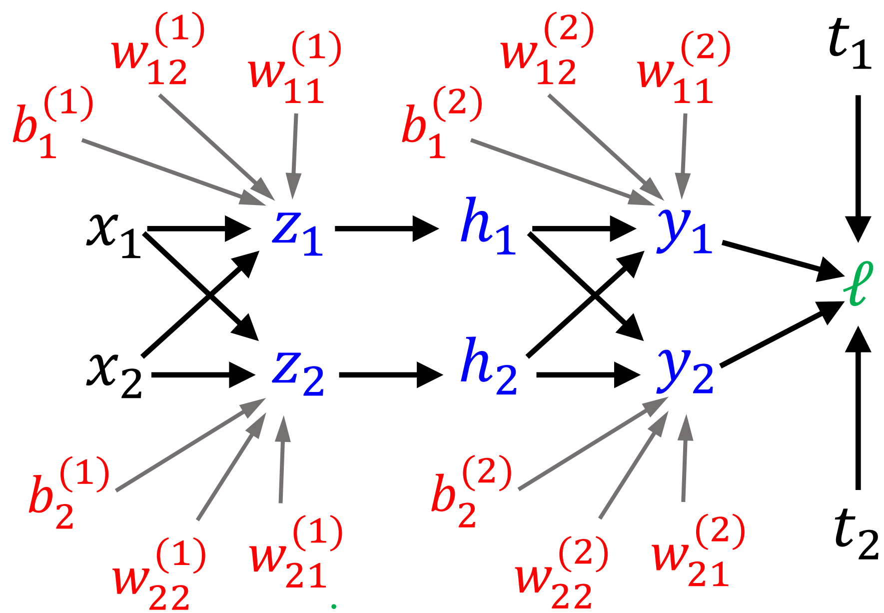
Forward pass

$$z_i = \sum_j w_{ij}^{(1)} x_j + b_i^{(1)}$$

$$h_i = \sigma(z_i)$$

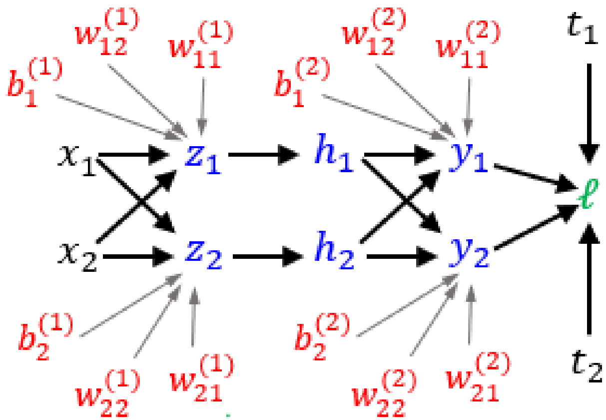
$$y_k = \sum_i w_{ki}^{(2)} h_i + b_k^{(1)}$$

$$\ell = \frac{1}{2}\sum_k (t_k - y_k)^2$$



Backpropagation for MLP

Multilayer Perceptron (multiple outputs):



Forward pass

$$z_i = \sum_j w_{ij}^{(1)} x_j + b_i^{(1)}$$

$$h_i = \sigma(z_i)$$

$$y_k = \sum_i w_{ki}^{(2)} h_i + b_k^{(1)}$$

$$\ell = \frac{1}{2} \sum_k (t_k - y_k)^2$$

Backward pass

$$\overline{\ell} = 1$$

$$\overline{y}_k = \overline{\ell} \frac{\partial \ell}{\partial y_k} = -\overline{\ell} \, \underline{(t_k - y_k)}$$

$$\overline{w}_{ki}^{(2)} = \overline{y}_k \frac{\partial y_k}{\partial w_{ki}^{(2)}} = \overline{y}_k h_i$$

$$\frac{\partial \ell}{\partial b_k^{(2)}} = \overline{b}_k^{(2)} = \overline{y}_k \frac{\partial y_k}{\partial b_k^{(2)}} = \overline{y}_k$$

$$\rightarrow \overline{h}_i = \sum_k \overline{y}_k \frac{\partial y_k}{\partial h_i} = \sum_k \overline{y}_k w_{ki}^{(2)}$$

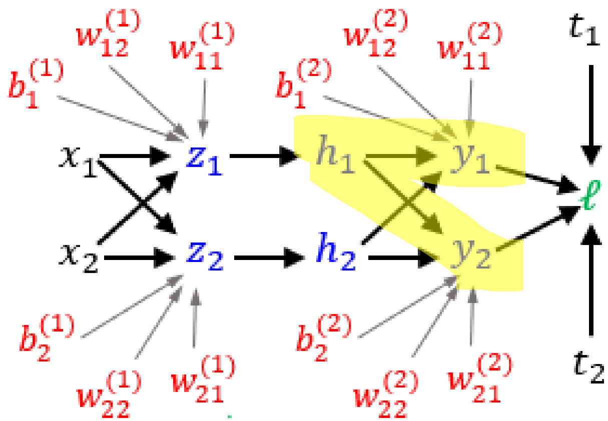
$$\overline{z}_i = \overline{h}_i \frac{\partial h_i}{\partial z_i} = \overline{h}_i \, \sigma'(z_i)$$

$$\overline{w}_{ij}^{(1)} = \overline{z}_i \frac{dz_i}{dw_{ij}^{(1)}} = \overline{z}_i x_j$$

$$\overline{b}_i^{(1)} = \overline{z}_i \frac{dz_i}{db_i^{(1)}} = \overline{z}_i$$

Backpropagation for MLP

Multilayer Perceptron (multiple outputs)**:**



Full break

$$\bar{h}_1 = \bar{y}_1 \frac{\partial y_1}{\partial h_1} + \bar{y}_2 \frac{\partial y_2}{\partial h_1} = \bar{y}_1 w_{11}^{(2)} + \bar{y}_2 w_{21}^{(2)}$$

$\frac{\partial \ell}{\partial h_1}$ $\frac{\partial \ell}{\partial y_1}$ $\frac{\partial \ell}{\partial y_2}$

Forward pass

$$z_i = \sum_j w_{ij}^{(1)} x_j + b_i^{(1)}$$

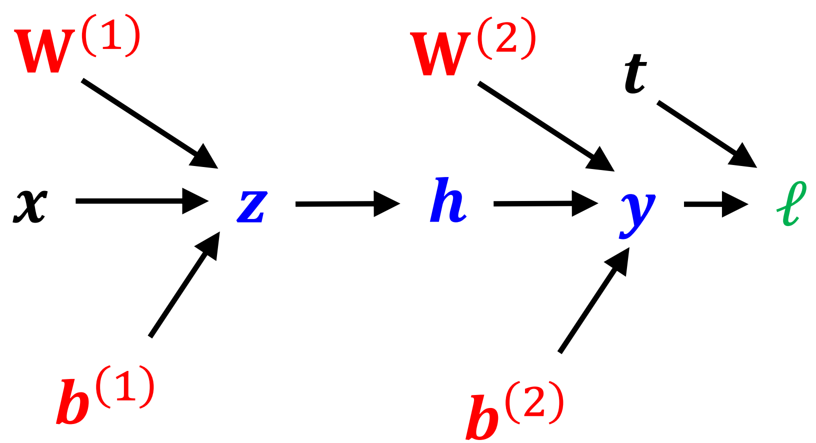
$$h_i = \sigma(z_i)$$

$$y_k = \sum_i w_{ki}^{(2)} h_i + b_k^{(1)}$$

$$\ell = \frac{1}{2} \sum_k (t_k - y_k)^2$$

Vector form

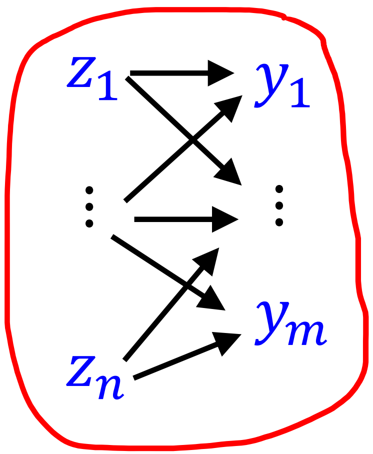
- Computation graphs showing individual units are cumbersome
- We can draw graphs over the **vectorized** variables



- We pass the gradients back in the same way as for the scalar-valued nodes

Vector form

- Consider this partial computation graph:



$$\boldsymbol{z} \longrightarrow \boldsymbol{y}$$

$$\bar{\boldsymbol{y}} = \frac{\partial L}{\partial \boldsymbol{y}}$$

$m \times 1$

- Backprop rules: $\bar{z}_j = \sum_{k=1}^{m} \bar{y}_k \frac{\partial y_k}{\partial z_j}$

$$\underset{n\times 1}{\bar{\boldsymbol{z}}} = \underset{n\times m}{\left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{z}}\right)^T} \underset{m\times 1}{\bar{\boldsymbol{y}}}$$

vector

matrix

VJP

where $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{z}}$ is the **Jacobian matrix**

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{z}} = \begin{pmatrix} \frac{\partial y_1}{\partial z_1} & \cdots & \frac{\partial y_1}{\partial z_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial z_1} & \cdots & \frac{\partial y_m}{\partial z_n} \end{pmatrix}_{m\times n}$$

Vector form

Full backpropagation algorithm (vector form):

Let $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_N$ be a topological ordering of the computation graph (i.e. parents come before children)

$\boldsymbol{v}_N$ denotes the variable we are trying to compute derivatives of (e.g. loss function)

Forward pass
Compute values

For $i = 1, \dots, N$

Compute $\boldsymbol{v}_i$ as a function of Pa($\boldsymbol{v}_i$)

Backward pass
Compute derivatives

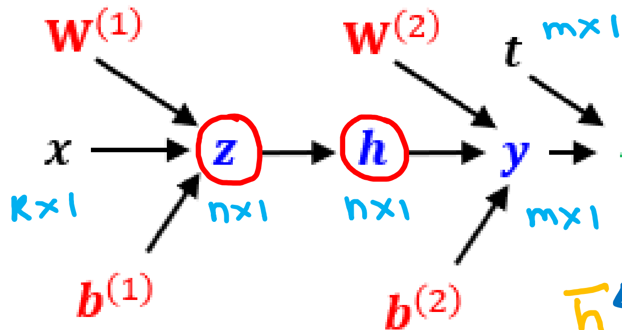
$\bar{v}_N = 1$

For $i = N-1, \dots, 1$

$$\bar{\boldsymbol{v}}_i = \sum_{j \in \mathrm{Ch}(\bar{v}_i)} \left(\frac{\partial \boldsymbol{v}_j}{\partial \boldsymbol{v}_i} \right)^T \bar{\boldsymbol{v}}_j$$

Vector form

MLP example in vectorized form:



$\bar{h}$ ← vector

$$\bar{z} = \frac{\partial \ell}{\partial z} = \boxed{\frac{\partial \ell}{\partial h}}\ \boxed{\frac{\partial h}{\partial z}}$$

$\frac{\partial h}{\partial z}$: $\frac{\sigma'(z)}{\text{vector}}$

Forward pass

$$\underset{n\times 1}{z} = \underset{n\times k}{\mathbf{W}^{(1)}}\,\underset{k\times 1}{x} + \underset{n\times 1}{b^{(1)}}$$

$$\underset{n\times 1}{h} = \sigma(z)$$

$$\underset{m\times 1}{y} = \underset{m\times n}{\mathbf{W}^{(2)}}\,\underset{n\times 1}{h} + \underset{m\times 1}{b^{(2)}}$$

$$\ell = \frac{1}{2}\underset{1\times m}{(t-y)^T}\,\underset{m\times 1}{(t-y)}$$

$$\frac{\partial \ell}{\partial W^{(2)}} = \frac{\partial \ell}{\partial y} \times \boxed{\frac{\partial y}{\partial W^{(2)}}}$$

Backward pass

$$\underset{1\times 1}{\bar{\ell}} = 1$$

$$\underset{m\times 1}{\bar{y}} = -\bar{\ell}\,\underset{m\times 1}{(t-y)}$$

$$\frac{\partial \ell}{\partial W^{(2)}} \leftarrow \underset{m\times n}{\bar{\mathbf{W}}^{(2)}} = \bar{y}\,\underset{1\times n}{h^T}$$

$$\underset{m\times 1}{\bar{b}^{(2)}} = \underset{m\times 1}{\bar{y}}$$

$$\underset{n\times 1}{\bar{h}} = \underset{n\times m}{\mathbf{W}^{(2)\mathrm{T}}}\,\underset{m\times 1}{\bar{y}}$$

Elementwise multiplication

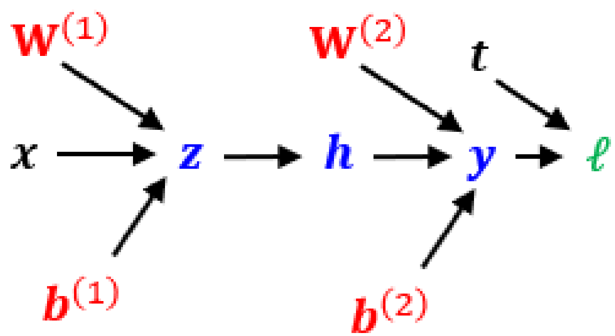
$$\underset{n\times 1}{\bar{z}} = \underset{n\times 1}{\bar{h}} \circ \underset{n\times 1}{\sigma'(z)}$$

$$\underset{n\times k}{\bar{\mathbf{W}}^{(1)}} = \bar{z}\,\underset{1\times k}{x^T}$$

$$\underset{n\times 1}{\bar{b}^{(1)}} = \underset{n\times 1}{\bar{z}}$$

Vector form

MLP example in vectorized form:



Forward pass

$$\boldsymbol{z} = \mathbf{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}$$

$$\boldsymbol{h} = \sigma(\boldsymbol{z})$$

$$\boldsymbol{y} = \mathbf{W}^{(2)}\boldsymbol{h} + \boldsymbol{b}^{(2)}$$

$$\ell = \frac{1}{2}(\boldsymbol{t} - \boldsymbol{y})^{T}(\boldsymbol{t} - \boldsymbol{y})$$

Backward pass

$$\overline{\ell} = 1$$

$\frac{\partial \ell}{\partial \underset{\sim}{y}}$ → $$\overline{\boldsymbol{y}} = -\overline{\ell}\,(\boldsymbol{t} - \boldsymbol{y})$$ $\frac{\partial \ell}{\partial \underset{\sim}{y}}$

$$\overline{\mathbf{W}}^{(2)} = \overline{\boldsymbol{y}}\,\boldsymbol{h}^{T}$$

$$\overline{\boldsymbol{b}}^{(2)} = \overline{\boldsymbol{y}}$$

$$\overline{\boldsymbol{h}} = \mathbf{W}^{(2)\mathrm{T}}\,\overline{\boldsymbol{y}}$$

Elementwise multiplication

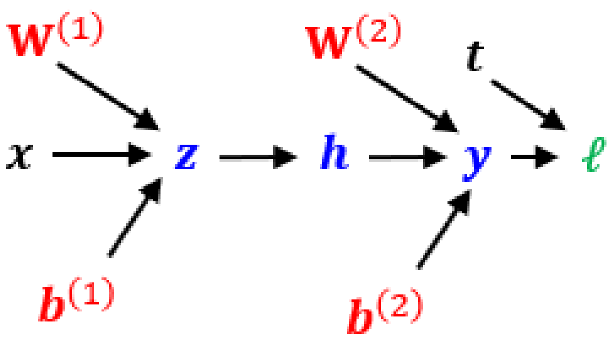
$$\overline{\boldsymbol{z}} = \overline{\boldsymbol{h}} \circ \sigma'(\boldsymbol{z})$$

$$\overline{\mathbf{W}}^{(1)} = \overline{\boldsymbol{z}}\,\boldsymbol{x}^{T}$$

$$\overline{\boldsymbol{b}}^{(1)} = \overline{\boldsymbol{z}}$$

Backpropagation in MLPs

MLP example in vectorized form:



Forward pass

$$\boldsymbol{z} = \mathbf{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}$$
$$\boldsymbol{h} = \sigma(\boldsymbol{z})$$
$$\boldsymbol{y} = \mathbf{W}^{(2)}\boldsymbol{h} + \boldsymbol{b}^{(2)}$$
$$\ell = \frac{1}{2}(\boldsymbol{t}-\boldsymbol{y})^T(\boldsymbol{t}-\boldsymbol{y})$$

Backward pass

$$\overline{\ell} = 1$$
$$\overline{\boldsymbol{y}} = -\overline{\ell}\,(\boldsymbol{t}-\boldsymbol{y})$$
$$\overline{\mathbf{W}}^{(2)} = \overline{\boldsymbol{y}}\,\boldsymbol{h}^T$$
$$\overline{\boldsymbol{b}}^{(2)} = \overline{\boldsymbol{y}}$$
$$\overline{\boldsymbol{h}} = \mathbf{W}^{(2)\mathrm{T}}\,\overline{\boldsymbol{y}}$$
$$\overline{\boldsymbol{z}} = \overline{\boldsymbol{h}} \circ \sigma'(\boldsymbol{z})$$
$$\overline{\mathbf{W}}^{(1)} = \overline{\boldsymbol{z}}\,\boldsymbol{x}^T$$
$$\overline{\boldsymbol{b}}^{(1)} = \overline{\boldsymbol{z}}$$

- Backpropagation in MLPs are commonly implemented as matrix-vector multiplications
- Here, these matrix-vector multiplications can be called **vector Jacobian products (VJPs)**

Closing remarks

- Backprop is used to train most of the neural nets you will find today
- Even optimization algorithms much fancier than gradient descent (e.g. second-order methods) use backprop to compute the gradients
- Backprop is based on the computation graph, and it basically works backwards through the graph, applying the chain rule at each node
- Once the derivatives w.r.t. the weights and biases are computed using backprop, the updates are applied to the weights and biases using some optimization scheme

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}}$$

$$\boldsymbol{b} \leftarrow \boldsymbol{b} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}}$$

- However, here we wrote out the computation graph and calculated the derivatives by hand ourselves; this hand calculation is avoided by **automatic differentiation**