Lecture 14 : Backpropagation for training Neural Nets

- A neural network is a parametric model

- We will tune its parameters using optimization techniques such as SGD, ADAM, etc.

- To find suitable values for the parameters $\underline{\theta}$, we will solve the optimization problem:

$$\underline{\theta} = \begin{bmatrix} \mathrm{vec}(\underline{\underline{W}}^{(1)}) \\ \underline{b}^{(1)} \\ \vdots \\ \mathrm{vec}(\underline{\underline{W}}^{(L)}) \\ \underline{b}^{(L)} \end{bmatrix}$$

$$\hat{\underline{\theta}} = \underset{\underline{\theta}}{\mathrm{argmin}} \underbrace{J(\underline{\theta})}_{\text{cost function}}$$

where

$$J(\underline{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \underbrace{L\left(y_i, f(\underline{x}_i; \underline{\theta})\right)}_{\substack{\text{loss function} \\ \text{for data point } (\underline{x}_i,\, y_i)}}$$

$$J(\underline{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \underbrace{L\left(y_i, f(\underline{x}_i;\underline{\theta})\right)}_{\substack{\text{loss function}\\ \text{for data point } (\underline{x}_i,\, y_i)}}$$

- The functional form of the loss function depends on the problem
    - Regression problems typically use squared error loss

$$L\left(y, f(\underline{x};\underline{\theta})\right) = \left(y - \underbrace{f(\underline{x};\underline{\theta})}\right)^2$$

output of a neural net

    - Multi-class classification problems typically use cross-entropy loss (with M classes)

$$L\left(y, f(\underline{x};\underline{\theta})\right) = -\ln \overbrace{g_m\left(\underline{f}(\underline{x};\underline{\theta})\right)}^{\text{softmax}}$$

$$= -\ln g_y\left(\underline{f}(\underline{x};\underline{\theta})\right)$$

$\underline{f}$ is a vector of $M \times 1$

we use training data label y as an index variable to select the correct logit

$$\hat{\underline{\theta}} = \underset{\underline{\theta}}{\operatorname{argmin}}\ J(\underline{\theta}) \quad \text{where} \quad J(\underline{\theta}) = \frac{1}{N}\sum_{i=1}^{N} L\left(y_i, f(\underline{x}_i; \underline{\theta})\right)$$

- These optimization problems cannot be solved in closed form
  → Numerical optimization algorithms have to be used

- Numerical optimization update parameters in an iterative manner
  In deep learning, one typically uses gradient-descent algorithms

Step 1 : Pick an initial guess $\underline{\theta}^{(0)}$

Step 2 : Calculate gradient of the cost function w.r.t $\underline{\theta}^{(t)}$, $t = 0, 1, 2, \ldots$

Step 3 : Update the parameters as $\underline{\theta}^{(t+1)} \leftarrow \underline{\theta}^{(t)} - \gamma \nabla_{\underline{\theta}} J(\underline{\theta}^{(t)})$

Step 4 : Terminate when some criterion is fulfilled, and take the last $\underline{\theta}^{(t)}$ as $\hat{\underline{\theta}}$

Computational Challenges

1) Large datasets (N very large)

- The number of datapoints N is large in deep learning applications
- Makes computation of the cost function gradient very costly, due to the sum
- We resort to using a random subset of data to update parameters

  ↳ minibatch gradient descent

2) Large number of parameters $\underline{\theta}$

- The dimension of the parameter vector $\underline{\theta}$ is very large in deep learning
- To efficiently calculate the gradient $\nabla_{\underline{\theta}} J(\underline{\theta}^{(t)})$, we need to apply chain rule of calculus

this will be done using the Backpropagation algorithm

Univariate chain rule

- Let's compute the derivative of loss function w.r.t parameters w and b

$$z = wx + b$$

parameters

$$\hat{y} = \sigma(z)$$

non-linear activation function (e.g. sigmoid)

$$L = (y - \hat{y})^2$$

loss function

$$L = (y - \sigma(wx+b))^2$$

$$\frac{\partial L}{\partial w} = \frac{\partial}{\partial w}(y - \sigma(wx+b))^2$$

$$= (y - \sigma(wx+b)) \frac{\partial}{\partial w}(y - \sigma(wx+b))$$

$$= -(y - \sigma(wx+b))\, \sigma'(wx+b) \frac{\partial}{\partial w}(wx+b)$$

$$= -(y - \sigma(wx+b))\, \sigma'(wx+b)\, x$$

$$\frac{\partial L}{\partial b} = \frac{\partial}{\partial b}(y - \sigma(wx+b))^2$$

$$= (y - \sigma(wx+b)) \frac{\partial}{\partial b}(y - \sigma(wx+b))$$

$$= -(y - \sigma(wx+b))\, \sigma'(wx+b)$$

Disadvantages of this approach

- Calculations are very cumbersome

  A lot of terms have been copied from one line to the next

- Final expression has repeated terms

Univariate chain rule

A more structured approach of chain rule would be:

1) Compute the loss

$$z = wx + b$$

$$\hat{y} = \sigma(z)$$

$$L = (y - \hat{y})^2$$

2) Compute the derivatives

$$\frac{\partial L}{\partial \hat{y}} = -2(y - \hat{y})$$

computed from previous step

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial \hat{y}} \frac{d\hat{y}}{dz} = \frac{\partial L}{\partial \hat{y}} \sigma'(z)$$

evaluated at current step

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

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

This form of computation is clean and has no repeated expressions!

Computational graph

- The computations can be plotted using a computational graph

1) Compute the loss

$$z = wx + b$$

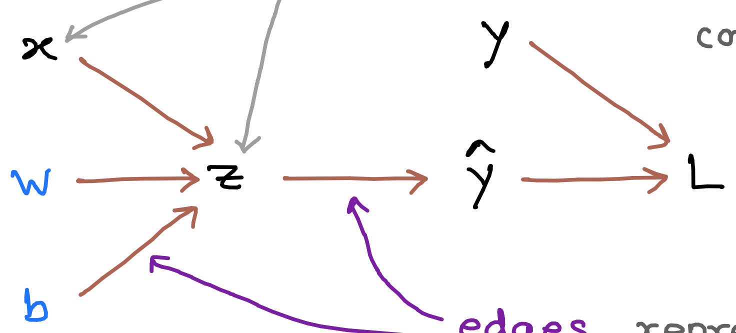
$$\hat{y} = \sigma(z)$$

$$L = (y - \hat{y})^2$$

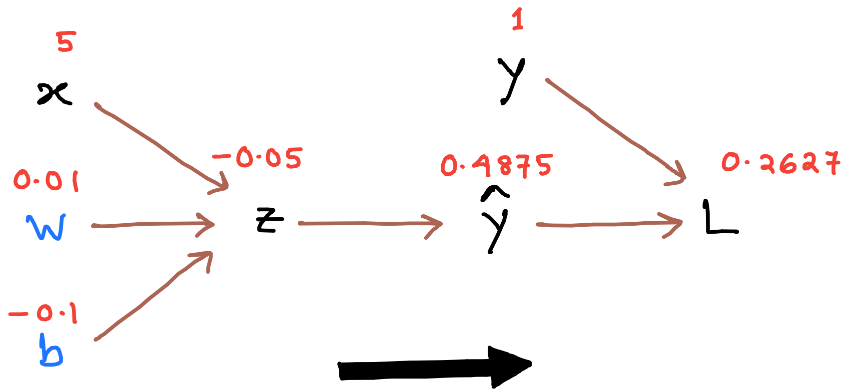
sigmoid activation

$$\frac{e^z}{1+e^z}$$

Nodes represent all inputs and computed quantities



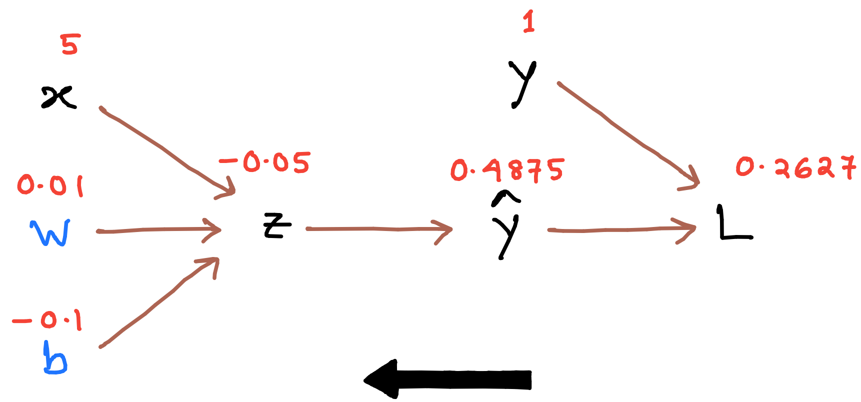
edges represent which nodes are computed direct as a function of other nodes



Forward pass for loss : Parents ⟶ Children

For backprop, we will use notation

$$\bar{v} = \frac{\partial L}{\partial v}, \quad \bar{x} = \frac{\partial L}{\partial x}, \text{ etc.}$$



Backward pass for gradients

Parents ⟵ Children

1) Compute the loss

$$z = wx + b$$

$$\hat{y} = \sigma(z)$$

$$L = (y - \hat{y})^2$$

2) Compute the derivatives

$$\frac{\partial L}{\partial \hat{y}} = -2(y - \hat{y})$$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial \hat{y}} \frac{d\hat{y}}{dz} = \frac{\partial L}{\partial \hat{y}} \sigma'(z)$$

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

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

$$\bar{\hat{y}} = -2(y - \hat{y})$$

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

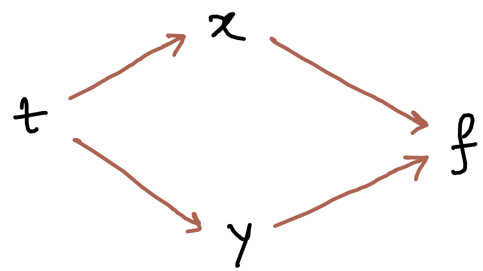
$$\bar{w} = \bar{z}\, x$$

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

Multivariate Chain Rule

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

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

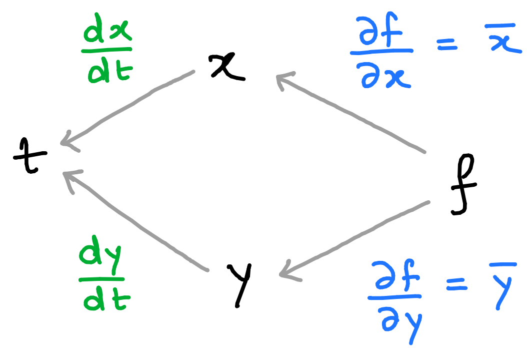


- In the context of gradient computation (backward pass)

these values will be computed first

- $$\frac{\partial f}{\partial t} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$

Parents ⟵ Children



- $$\frac{\partial f}{\partial t} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$

these will be evaluated next

$$\boxed{\begin{array}{c}\text{In our notation} \\ \bar{t} = \bar{x}\,\frac{dx}{dt} + \bar{y}\,\frac{dy}{dt}\end{array}}$$

Backpropagation Algorithm

- Let $v_1, v_2, \dots, v_p$ be a topological ordering of the computational graph (i.e. where parents come before children)

Ex:

$$\begin{tikzcd}
v_1 \arrow[dr] & & & v_6 \arrow[d] \\
v_2 \arrow[r] & v_4 \arrow[r] & v_5 \arrow[r] & v_7 \\
v_3 \arrow[ur] & & &
\end{tikzcd}$$

- v_p denotes the variable we are trying to compute the derivatives of

In our case $v_p \equiv L$ (loss function)

Forward pass

Compute values

For $i = 1, \dots, p$

Compute v_i as a function of Parents(v_i)

Backward pass

Compute derivatives

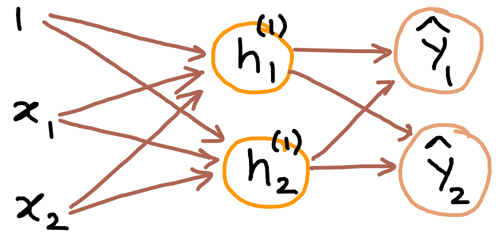
Treat $\bar{v}_p = \dfrac{\partial v_p}{\partial v_p} = 1$

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

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

Backpropagation for Neural Nets

Neural net with 1-hidden layer
(with multiple outputs)



Forward pass

(to compute loss)

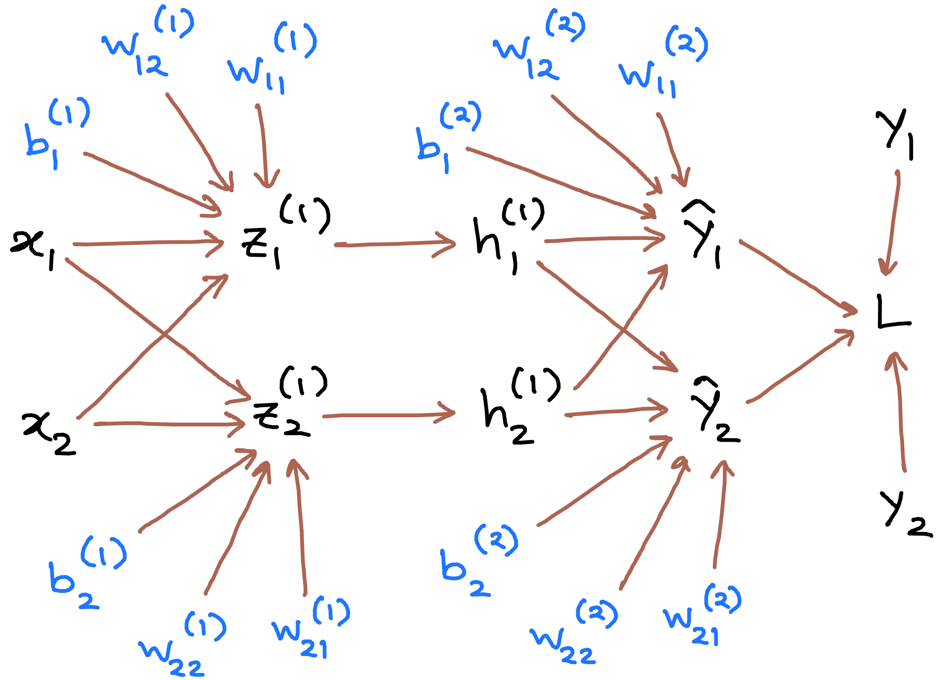
$$z_i^{(1)} = \sum_j w_{ij}^{(1)} x_j + b_i^{(1)}$$

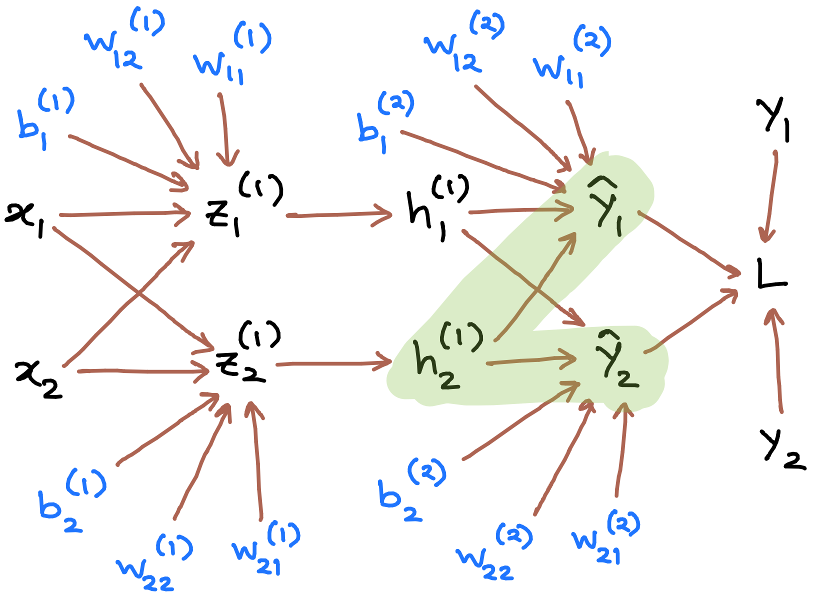
$$h_i^{(1)} = \sigma\left(z_i^{(1)}\right)$$

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

$$L = \sum_k \left(y_k - \hat{y}_k\right)^2$$

Computational graph





Backward pass

(to compute gradients)

$$\bar{L} = 1$$

$$\bar{\hat{y}}_k = \bar{L}\,\frac{\partial L}{\partial \hat{y}_k} = -2\bar{L}\,(y_k - \hat{y}_k)$$

$$\bar{w}_{ki}^{(2)} = \bar{\hat{y}}_k\,\frac{\partial \hat{y}_k}{\partial w_{ki}^{(2)}} = \bar{\hat{y}}_k\, h_i^{(1)}$$

$$\bar{b}_k^{(2)} = \bar{\hat{y}}_k\,\frac{\partial \hat{y}_k}{\partial b_k^{(2)}} = \bar{\hat{y}}_k$$

$$\frac{\partial L}{\partial h_2^{(1)}} = \frac{\partial L}{\partial \hat{y}_1}\frac{\partial \hat{y}_1}{\partial h_2^{(1)}} + \frac{\partial L}{\partial \hat{y}_2}\frac{\partial \hat{y}_2}{\partial h_2^{(1)}}$$

$$\bar{h}_i^{(1)} = \sum_k \bar{\hat{y}}_k\,\frac{\partial \hat{y}_k}{\partial h_i^{(1)}} = \sum_k \bar{\hat{y}}_k\, w_{ki}^{(2)}$$

$$\bar{z}_i^{(1)} = \bar{h}_i^{(1)}\,\frac{\partial h_i^{(1)}}{\partial z_i^{(1)}} = \bar{h}_i^{(1)}\,\sigma'\!\left(z_i^{(1)}\right)$$

$$\bar{w}_{ij}^{(1)} = \bar{z}_i^{(1)}\,\frac{\partial z_i^{(1)}}{\partial w_{ij}^{(1)}} = \bar{z}_i^{(1)}\, x_j$$

$$\bar{b}_i^{(1)} = \bar{z}_i^{(1)}\,\frac{\partial z_i^{(1)}}{\partial b_i^{(1)}} = \bar{z}_i^{(1)}$$

Forward pass

(to compute loss)

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

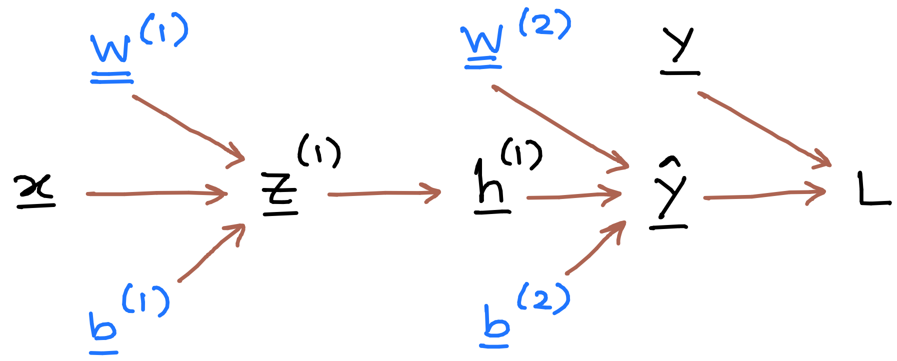
$$h_i^{(1)} = \sigma\left(z_i^{(1)}\right)$$

$$\hat{y}_k = \sum_i w_{ki}^{(2)}\, h_i^{(1)} + b_k^{(2)}$$

$$L = \sum_k \left(y_k - \hat{y}_k\right)^2$$

Vectorized form of BackProp

- Computational graphs showing individual units are cumbersome
- Instead draw graphs over the vectorized variables



Backprop rules

Forward Pass

Compute values

For $i = 1, \dots, P$

Compute $\underline{v}_i$ as a function of Parents $(\underline{v}_i)$

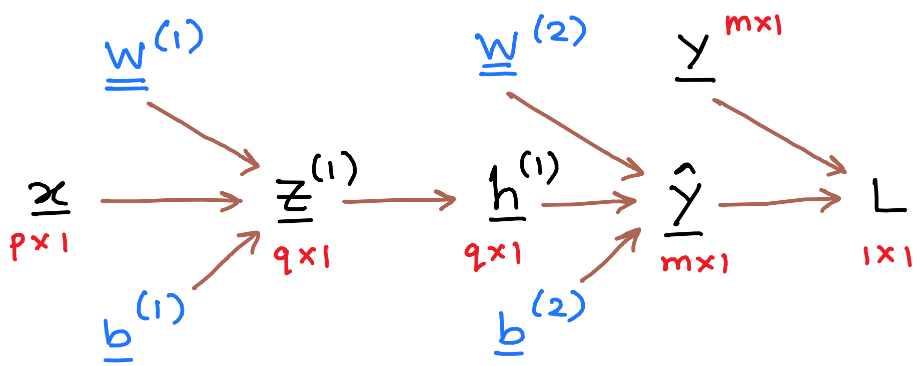
Backward Pass: Compute derivatives

Treat $\overline{\underline{v}}_P = 1$

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

$$\overline{\underline{v}}_i = \sum_{j \in \text{Children}(\underline{v}_i)} \overline{\underline{v}}_j \frac{\partial \underline{v}_j}{\partial \underline{v}_i}$$

BackProp example in vectorized form



Forward Pass

$$\underset{q\times1}{\underline{z}^{(1)}} = \underset{q\times p}{\underline{\underline{W}}^{(1)}}\ \underset{p\times1}{\underline{x}} + \underset{q\times1}{\underline{b}^{(1)}}$$

$$\underset{q\times1}{\underline{h}^{(1)}} = \sigma\big(\underset{q\times1}{\underline{z}^{(1)}}\big)$$

$$\underset{m\times1}{\hat{\underline{y}}} = \underset{m\times q}{\underline{\underline{W}}^{(2)}}\ \underset{q\times1}{\underline{h}^{(1)}} + \underset{m\times1}{\underline{b}^{(2)}}$$

$$\underset{1\times1}{L} = \underset{1\times m}{(\underline{y}-\hat{\underline{y}})^T}\ \underset{m\times1}{(\underline{y}-\hat{\underline{y}})}$$

Backward Pass

$$\underset{1\times1}{\bar{L}} = 1$$

$\frac{\partial L}{\partial \hat{\underline{y}}}$

$$\underset{m\times1}{\bar{\hat{\underline{y}}}} = -2\bar{L}\ \underset{m\times1}{(\underline{y}-\hat{\underline{y}})}$$

$\frac{\partial L}{\partial \underline{\underline{W}}^{(2)}}$

$$\underset{m\times q}{\bar{\underline{\underline{W}}}^{(2)}} = \underset{m\times1}{\bar{\hat{\underline{y}}}}\ \underset{1\times q}{\underline{h}^{(1)T}}$$

$$\underset{m\times1}{\bar{\underline{b}}^{(2)}} = \underset{m\times1}{\bar{\hat{\underline{y}}}}$$

$\frac{\partial L}{\partial \underline{h}^{(1)}}$

$$\underset{q\times1}{\bar{\underline{h}}^{(1)}} = \underset{q\times m}{\underline{\underline{W}}^{(2)T}}\ \underset{m\times1}{\hat{\underline{y}}}$$

$$\underset{q\times1}{\bar{\underline{z}}^{(1)}} = \underset{q\times1}{\bar{\underline{h}}^{(1)}} \circ \sigma'\big(\underset{q\times1}{\underline{z}^{(1)}}\big)$$

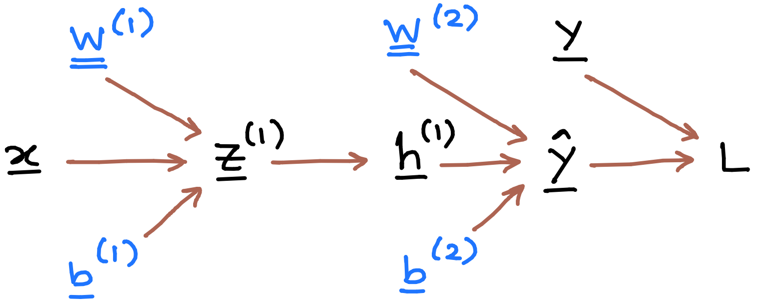
elementwise product

$\frac{\partial L}{\partial \underline{\underline{W}}^{(1)}}$

$$\underset{q\times p}{\bar{\underline{\underline{W}}}^{(1)}} = \underset{q\times1}{\bar{\underline{z}}^{(1)}}\ \underset{1\times p}{\underline{x}^{T}}$$

$$\underset{q\times1}{\bar{\underline{b}}^{(1)}} = \underset{q\times1}{\bar{\underline{z}}^{(1)}}$$

Back Prop example in vectorized form



Forward Pass

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

$$\underline{h}^{(1)} = \sigma(\underline{z}^{(1)})$$

$$\underline{\hat{y}} = \underline{\underline{W}}^{(2)} \underline{h}^{(1)} + \underline{b}^{(2)}$$

$$L = (\underline{y} - \underline{\hat{y}})^T (\underline{y} - \underline{\hat{y}})$$

Backward Pass

$$\bar{L} = 1$$

$$\bar{\underline{\hat{y}}} = -2\bar{L}\,(\underline{y} - \underline{\hat{y}})$$

$$\bar{\underline{\underline{W}}}^{(2)} = \bar{\underline{\hat{y}}}\, \underline{h}^{(1)^T}$$

$$\bar{\underline{b}}^{(2)} = \bar{\underline{\hat{y}}}$$

$$\bar{\underline{h}}^{(1)} = \bar{\underline{\underline{W}}}^{(2)^T}\, \underline{\hat{y}}$$

$$\bar{\underline{z}}^{(1)} = \bar{\underline{h}}^{(1)} \circ \sigma'(\underline{z}^{(1)})$$

$$\bar{\underline{\underline{W}}}^{(1)} = \bar{\underline{z}}^{(1)}\, \underline{x}^T$$

$$\bar{\underline{b}}^{(1)} = \bar{\underline{z}}^{(1)}$$

- Backprop in neural networks are commonly implemented as matrix-vector multiplications
- These matrix-vector multiplications are called vector Jacobian products (VJPs)

Closing Remarks

- Backprop is based on the computational graph, and it basically works backwards through the graph, applying the chain rule at each node

- Backprop is used to train most neural nets you will find these days

- Even optimization algorithms much fancier than gradient descent (such as second-order methods) use backprop to compute gradients

- 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

$$\underline{\underline{W}}^{(t+1)} \leftarrow \underline{\underline{W}}^{(t)} - \eta \left. \frac{\partial J}{\partial \underline{\underline{W}}} \right|_{\underline{\underline{W}}^{(t)}} \qquad \underline{b}^{(t+1)} \leftarrow \underline{b}^{(t)} - \eta \left. \frac{\partial J}{\partial \underline{b}} \right|_{\underline{b}^{(t)}}$$

- Hand-calculation of derivatives are replaced with automatic differentiation