MATH 565: Lecture 27 (04/21/2026)

Today: * back-propagation in neural networks

Recall In CGs,

$$z(i,j) = \frac{\partial y(j)}{\partial y(i)} \quad \text{edge derivative}$$

$$S(i,t) = \sum_{j \in A(i)} S(j,t) \cdot z(i,j) \quad \text{sensitivity recursion}$$

We now consider the case of feed forward neural networks or NNs in more detail. These are DACGs. But, unlike in generic CGs, each node in an NN typically applies an activation function Φ to a weighted sum $s(j)$ of its inputs.

For a single edge (i,j), we have

$$y(j) = \Phi(w_{ij}\, y(i)).$$

More generally,

$$y(j) = \Phi\left(\sum_{i \in AI(j)} w_{ij}\, y(i)\right) = \Phi(s(j))$$

$AI(j)$: tail nodes at end of in-arcs of node j

$s(j)$: weighted sum

$s(j)$: pre-activation at node j

LO4ML uses $a(j)$, but that could be confusing, since one may confuse that with the post-activation function. Some other books use $z(j)$.

Edge derivatives

We use the chain rule through pre-activation:

$$z(i,j) = \frac{\partial y(j)}{\partial y(i)} = \frac{\partial \Phi(s(j))}{\partial s(j)} \cdot \frac{\partial s(j)}{\partial y(i)}$$

$$\Rightarrow \quad z(i,j) = \Phi'(s(j)) \cdot w_{ij} \qquad \text{——— (1)}$$

In words, edge derivative of $(i,j) \in E$ in the NN is the product of the derivative of activation at pre-activation of node j and the synaptic weight w_{ij}.

Sensitivity Recursion

$$S(i,t) = \frac{\partial y(t)}{\partial y(i)} \equiv \text{sensitivity}$$

↗ how sensitive is the output function to a unit increase in $y(i)$ value.

The recursion works just as before:

$$S(i,t) = \sum_{j \in A(i)} S(j,t) \cdot z(i,j).$$

$$\Rightarrow \quad S(i,t) = \sum_{j \in A(i)} S(j,t) \cdot \Phi'(s(j)) \cdot w_{ij} \qquad \text{——— (2)}$$

with initialization (or boundary condition) $S(t,t) = 1$.

w_{ij} here "carries the signal" backward through the weight.

$\Phi'(s(j))$: "gates" how much gradient passes through node j.

$\Rightarrow$ When $\Phi' \approx 0$, the gate is "closed", and gives rise to the problem of vanishing gradients.

Derivatives of common activation functions

Common activation functions used in NNs: (as seen in Lecture 24)

$\Phi(v) = \frac{1}{1+e^{-v}}$ sigmoid function

$\Phi(v) = \frac{e^{2v}-1}{e^{2v}+1}$ tanh function — hyperbolic tangent

$\Phi(v) = \max\{v, 0\}$ ReLU (Rectified linear unit)

$\Phi(v) = \max\{\min\{v, 1\}, -1\}$ hard tanh function

We compute the derivative in each case.

1. Sigmoid: $\Phi(s) = \frac{1}{1+e^{-s}}$

$$\Rightarrow \frac{\partial \Phi(s)}{\partial s} = \frac{-1 \cdot -1 \cdot e^{-s}}{(1+e^{-s})^2} = \frac{1}{1+e^{-s}}\left(1 - \frac{1}{1+e^{-s}}\right) = \Phi(s)(1-\Phi(s))$$

2. tanh: $\Phi(s) = \frac{e^{2s}-1}{e^{2s}+1}$

$$\Rightarrow \frac{\partial \Phi}{\partial s} = \frac{(e^{2s}+1)2e^{2s} - (e^{2s}-1)2e^{2s}}{(e^{2s}+1)^2} = \frac{4e^{2s}}{(e^{2s}+1)^2}$$

$$= 1 - \left(\frac{e^{2s}-1}{e^{2s}+1}\right)^2 = 1 - (\Phi(s))^2.$$

3. ReLU $\Phi(s) = \max\{s, 0\}$.

$$\Rightarrow \Phi'(s) = \begin{cases} 1 & \text{if } s > 0, \\ 0 & \text{o.w.} \end{cases}$$

4. hard tanh $\Phi(s) = \max\{\min\{s, 1\}, -1\}$

$$\Rightarrow \Phi(s) = \begin{cases} 1 & \text{if } s \in [-1, 1] \\ 0 & \text{o.w.} \end{cases}$$

→ or just as a constant!

We can express Φ' conveniently in terms of Φ itself in each case!

Proof of backpropagation recursion

Lemma 14 Let $G_C = (\mathcal{N}, \mathcal{E})$ be a DACG with a single output node t, and let $S(i,t) = \frac{\partial y(t)}{\partial y(i)} \quad \forall i \in \mathcal{N}$.

Then S satisfies the recursion

$$S(i,t) = \sum_{j \in A(i)} S(j,t) \cdot z(i,j), \quad \text{with } S(t,t) = 1$$

where $z(i,j) = \frac{\partial y(j)}{\partial y(i)}$.

Proof We proceed by induction on $d(i) =$ length of longest directed path from i to t. (in terms of # arcs)

Base case $d(i) = 0 \Rightarrow i = t$. $S(t,t) = 1$. ✓

Induction Assume result holds for $\forall i' \in \mathcal{N}$ with $d(i') < d(i)$.
As G_C is a DAG, every directed path from i to t passes through exactly one $j \in A(i)$. By multivariate chain rule,

$$\frac{\partial y(t)}{\partial y(i)} = \sum_{j \in A(i)} \frac{\partial y(t)}{\partial y(j)} \cdot \frac{\partial y(j)}{\partial y(i)}$$

$$= \sum_{j \in A(i)} S(j,t) \cdot z(i,j)$$

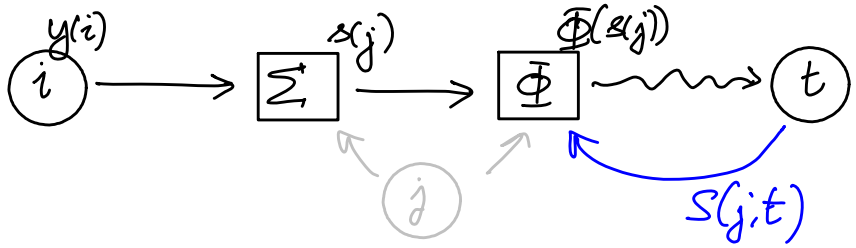
as each $j \in A(i)$ has $d(j) < d(i)$ and hence the induction hypothesis applies. □

Gradient w.r.t. edge weights

Assume G_c has a single output node t, and $y(t) = J$, the loss function. Then

$$\frac{\partial y(t)}{\partial w_{ij}} = \frac{\partial y(t)}{\partial y(j)} \cdot \frac{\partial y(j)}{\partial w_{ij}} = S(j,t) \cdot \Phi'(s(j)) \cdot y(i)$$

This gradient breaks into the product of three terms
* downstream sensitivity, $S(j,t)$
* local gate $\Phi'(s(j))$
* upstream (incoming) activation $y(i)$.



If $\Phi'(s(j)) \approx 0$, the gate is closed. Each such edge multiplies $S(i,t)$ by a very small number. Hence, over many layers, $S(i,t)$ becomes really small.
$\Rightarrow$ NN cannot learn!
One good reason to use ReLU, as $\Phi'(s) = 1$ for $s > 0$!

Note: $y(i)$ is available from forward pass, while $S(j,t)$ is computed using recursion running backwards.
$\Rightarrow$ Forward and backward passes together give all needed quantities for gradient computation.

Decoupled Formulation

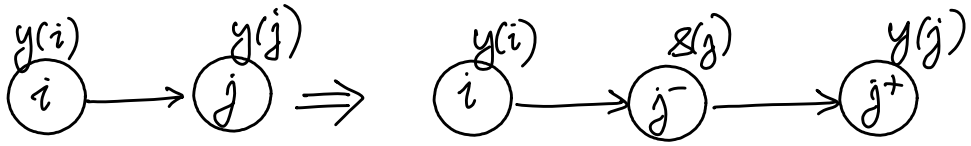
As in previous figure, we split each node j into two subnodes:

* linear subnode j^-: computes weighted sum

$$s(j) = \sum_{i \in \mathrm{AI}(j)} w_{ij}\, y(i)$$

* activation subnode j^+: computes output

$$y(j) = \Phi(s(j)).$$



So, each $(i,j) \in \mathcal{E}$ is replaced by a 2-edge path through sum and activation subnodes.

Edge Derivative in Decoupled NN

$$z(i, j^-) = \frac{\partial s(j)}{\partial y(i)} = w_{ij}$$

↳ just weight; no activation

↙ into the linear subnode

$$z(j^-, j^+) = \frac{\partial y(j)}{\partial s(j)} = \Phi'(s(j))$$

↳ just activation, no weight

↙ through the activation subnode.

Sensitivity Recursion in Decoupled NN

Recall, $S(i,t) = \sum_{j \in A(i)} S(j,t) \cdot z(i,j)$. This recursion applies to j^- and j^+ separately here.

$$S(j^-,t) = S(j^+,t) \cdot \bar{\Phi}'(s(j))$$

$$S(i,t) = \sum_{j \in A(i)} S(j^-,t) \cdot w_{ij}$$

$$= \sum_{j \in A(i)} S(j^+,t) \cdot \bar{\Phi}'(s(j)) \cdot w_{ij}$$

Gradient w.r.t. w_{ij}

Note that w_{ij} appears only in (i,j^-).

$$\Rightarrow \frac{\partial y(t)}{\partial w_{ij}} = S(j^-,t) \cdot y(i)$$

$$= S(j^+,t) \cdot \bar{\Phi}'(s(j)) \cdot y(i)$$

These formulae can easily generalize to the vector case in layerwise NNs.