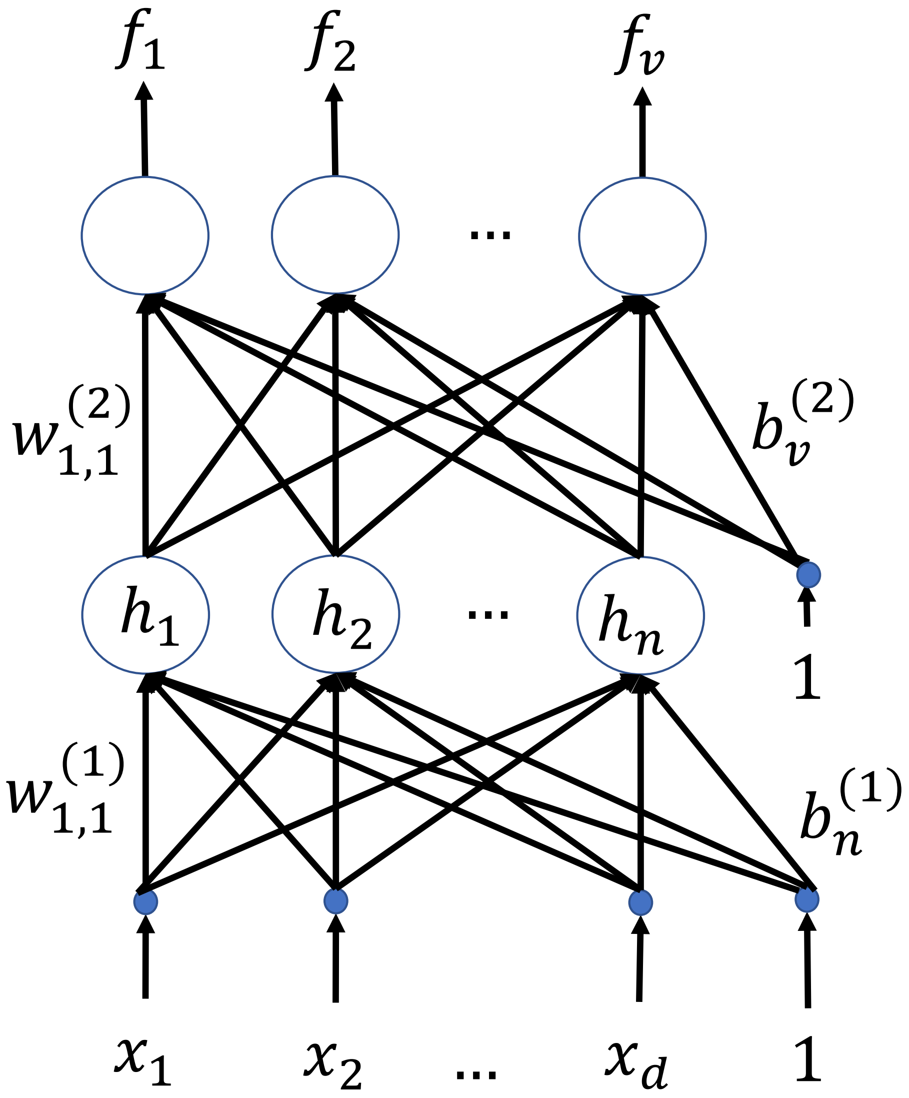
Lecture 12: Multi-Layer Neural Nets

Mark Hasegawa-Johnson

2/2024



Outline

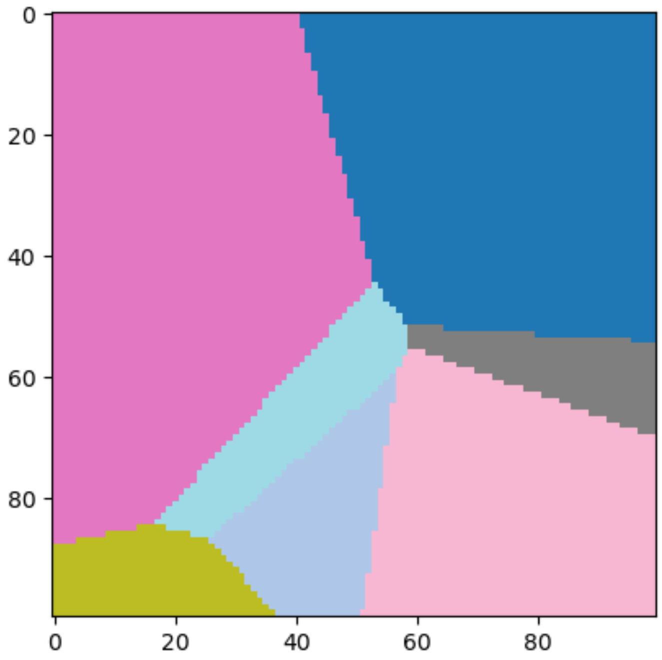
- From linear to nonlinear classifiers
- Training a two-layer network: Back-propagation

Linear classifier

Review: a linear classifier computes

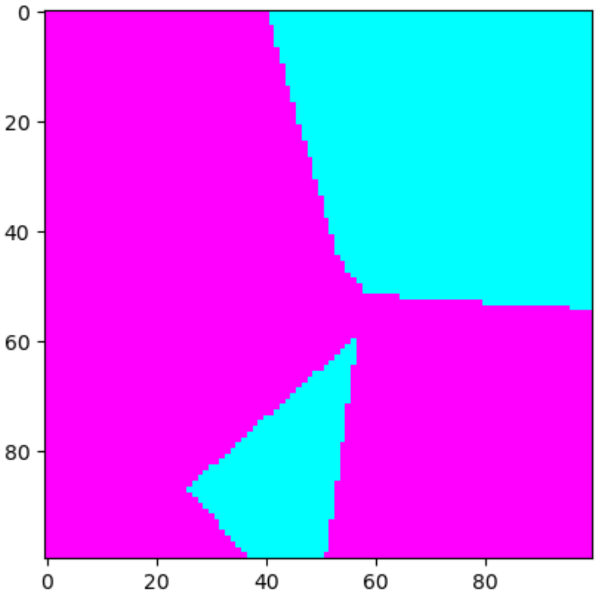
$$f(\boldsymbol{x}) = \operatorname{argmax} \boldsymbol{W}\boldsymbol{x}$$

The resulting classifier divides the x-space into Voronoi regions: convex regions with piece-wise linear boundaries



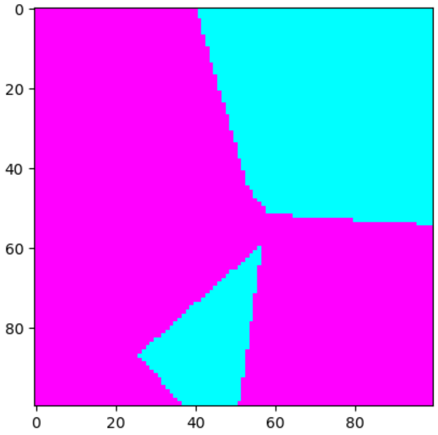
Nonlinear classifier

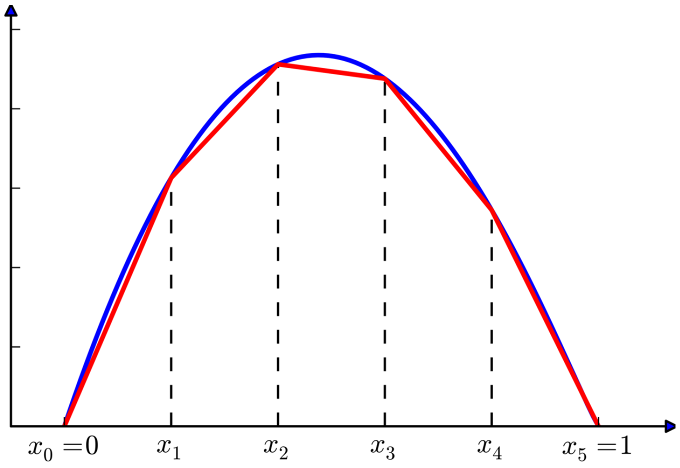
- Not all classification problems have convex decision regions with PWL boundaries!
- Here's an example problem in which class 0 (blue) includes values of **x** near $[0.8,0]^T$, but it also includes some values of **x** near $[0.4,0.9]^T$
- You can't compute this function using
$$f(\boldsymbol{x}) = \text{argmax}\, \boldsymbol{W}\boldsymbol{x}$$



The solution: Piece-wise linear functions

- Nonlinear classifiers, like this one, can be learned using piece-wise linear classification boundaries
- Nonlinear regression problems, like this one, can be learned using piece-wise linear regression
- In the limit, as the number of pieces goes to infinity, the approximation approaches the desired solution





Public domain image, Krishnavedala, 2011

Multi-layer network

A piece-wise linear function $f(\boldsymbol{x})$ can be represented by a two-layer neural network. First, the hidden nodes compute:

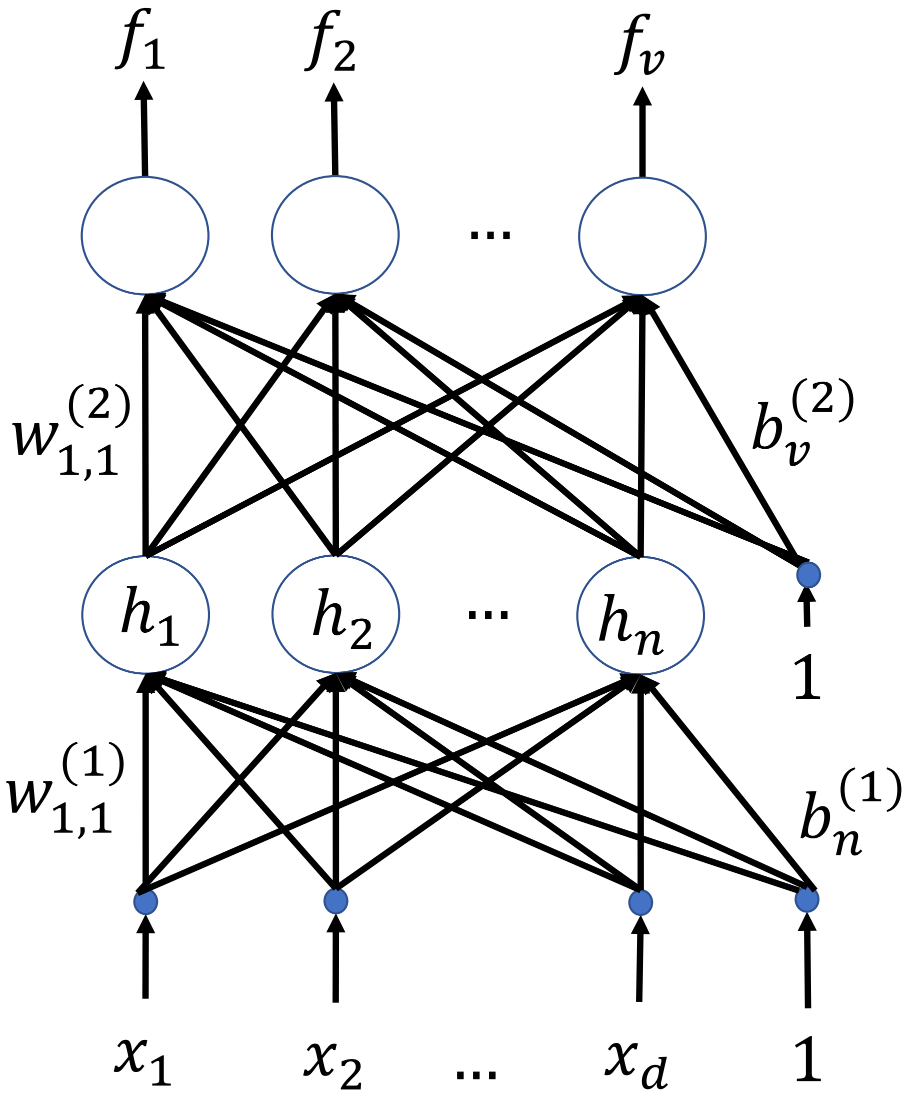
$$h_j(\boldsymbol{x}) = \max\left(0, \boldsymbol{w}_j^{(1),T}\boldsymbol{x} + b_j^{(1)}\right)$$

Then for PWL regression, the output is a weighted sum of the hidden nodes:

$$f(\boldsymbol{x}) = \boldsymbol{w}^{(2),T}\boldsymbol{x} + b^{(2)}$$

…while for PWL classification, the output is the softmax or argmax of such a sum:

$$\boldsymbol{f}(\boldsymbol{x}) = \mathrm{softmax}\left(0, \boldsymbol{W}^{(2)}\boldsymbol{x} + \boldsymbol{b}^{(2)}\right)$$



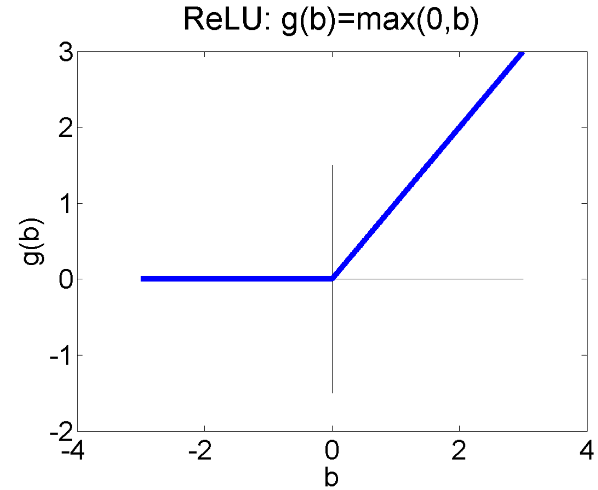
For a PWL neural net, the hidden nodes are ReLU

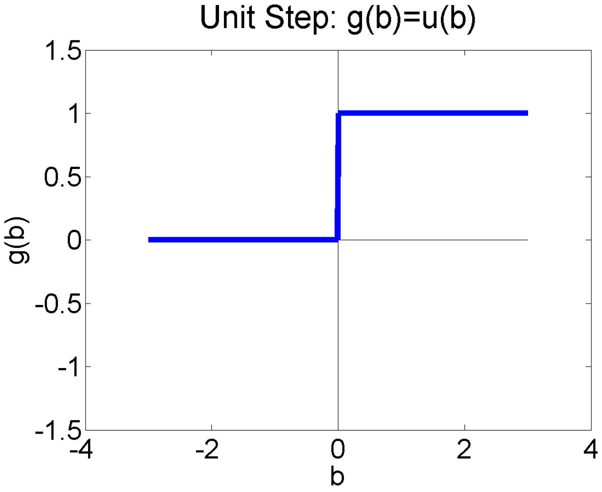
If the goal is PWL classification boundaries, we can achieve that by using hidden nodes that are the simplest possible PWL function: a Rectified Linear Unit, or ReLU:

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

This is differentiable everywhere except z=0; its derivative is the unit step function:

$$\frac{\partial \mathrm{ReLU}(z)}{\partial z} = u(z) = \begin{cases} 1 & z > 0 \\ 0 & z < 0 \end{cases}$$


ReLU: g(b)=max(0,b)


Unit Step: g(b)=u(b)

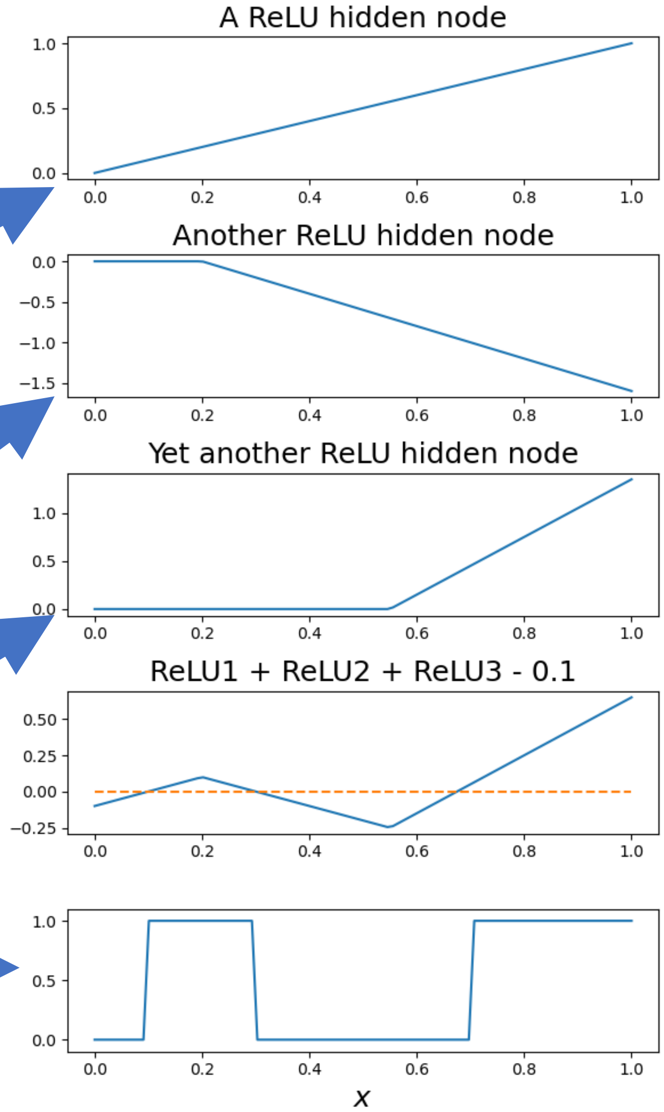
Example: Computing a non-linear classification boundary using ReLU hidden units

$$h_1(x) = ReLU(x)$$

$$w_{2,2}h_1(x) = -2ReLU(x - 0.2)$$

$$w_{2,3}h_2(x) = 3ReLU(x - 0.45)$$

$$f(x) = u(h_1 - 2h_2 + 3h_3 - 0.1)$$



A ReLU hidden node

Another ReLU hidden node

Yet another ReLU hidden node

ReLU1 + ReLU2 + ReLU3 - 0.1

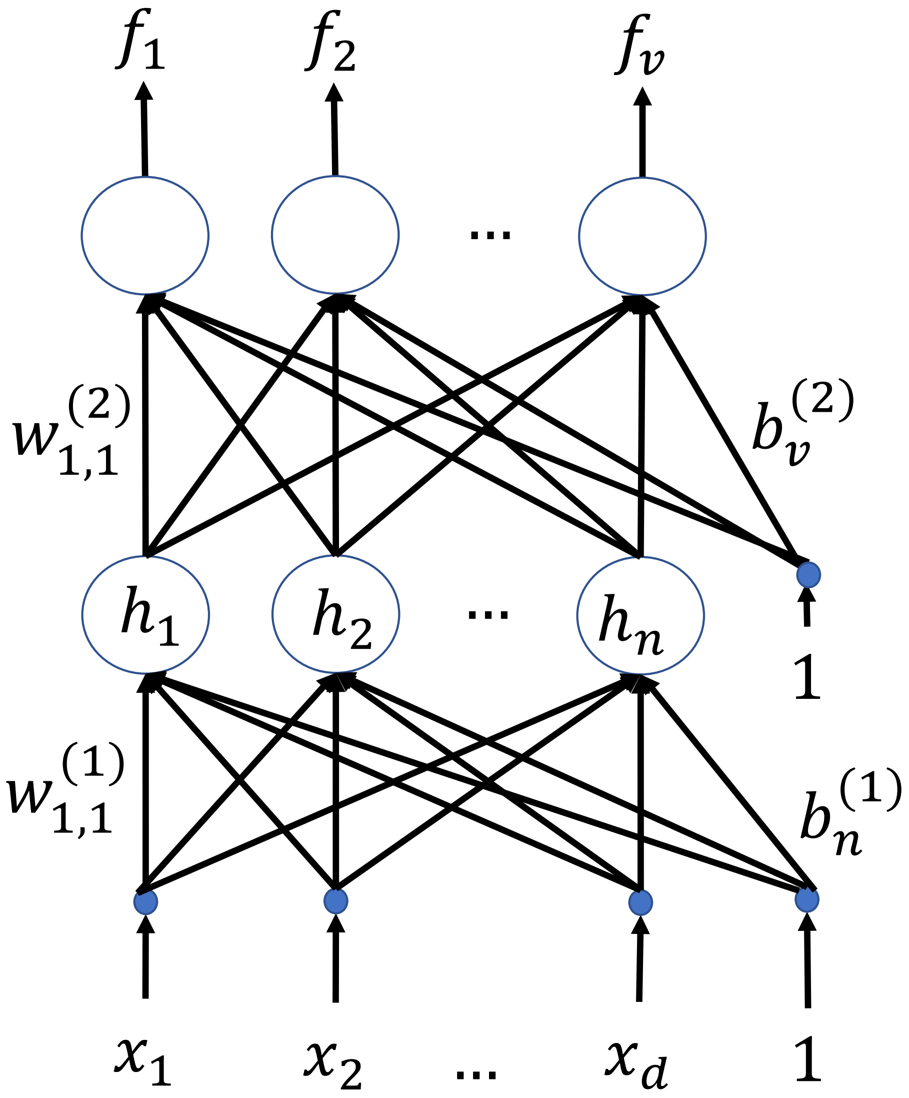
Outline

- From linear to nonlinear classifiers
- Training a two-layer network: Back-propagation

Training a neural net: Gradient descent

- Suppose we have some scalar loss function, $\mathcal{L}$, that we want to minimize
- Define the gradient of $\mathcal{L}$ w.r.t. the layer-l weight matrix, $\boldsymbol{W}^{(l)}$, as:

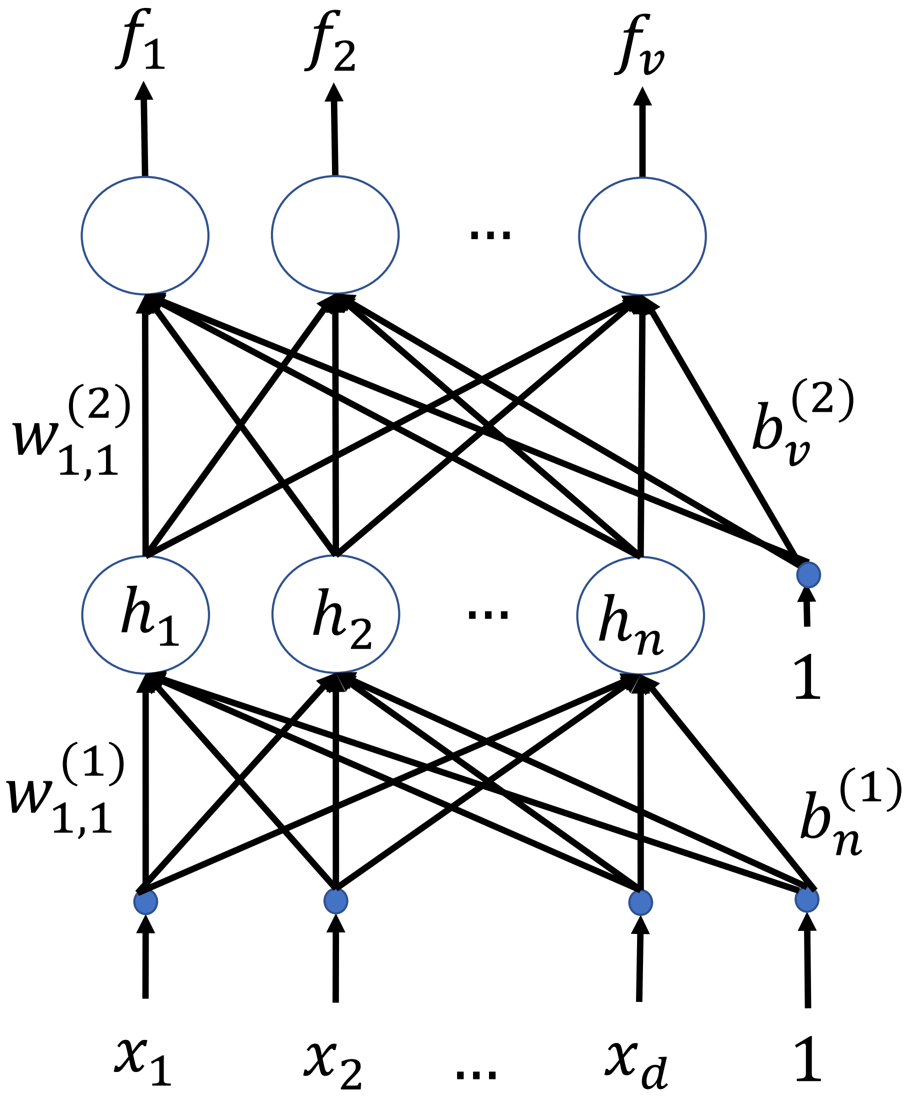
$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{(l)}} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial w_{1,1}^{(l)}} & \frac{\partial \mathcal{L}}{\partial w_{1,2}^{(l)}} & \cdots \\ \frac{\partial \mathcal{L}}{\partial w_{2,1}^{(l)}} & \frac{\partial \mathcal{L}}{\partial w_{2,2}^{(l)}} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$



Training a neural net: Gradient descent

Gradient descent updates $\boldsymbol{W}^{(l)}$ as:

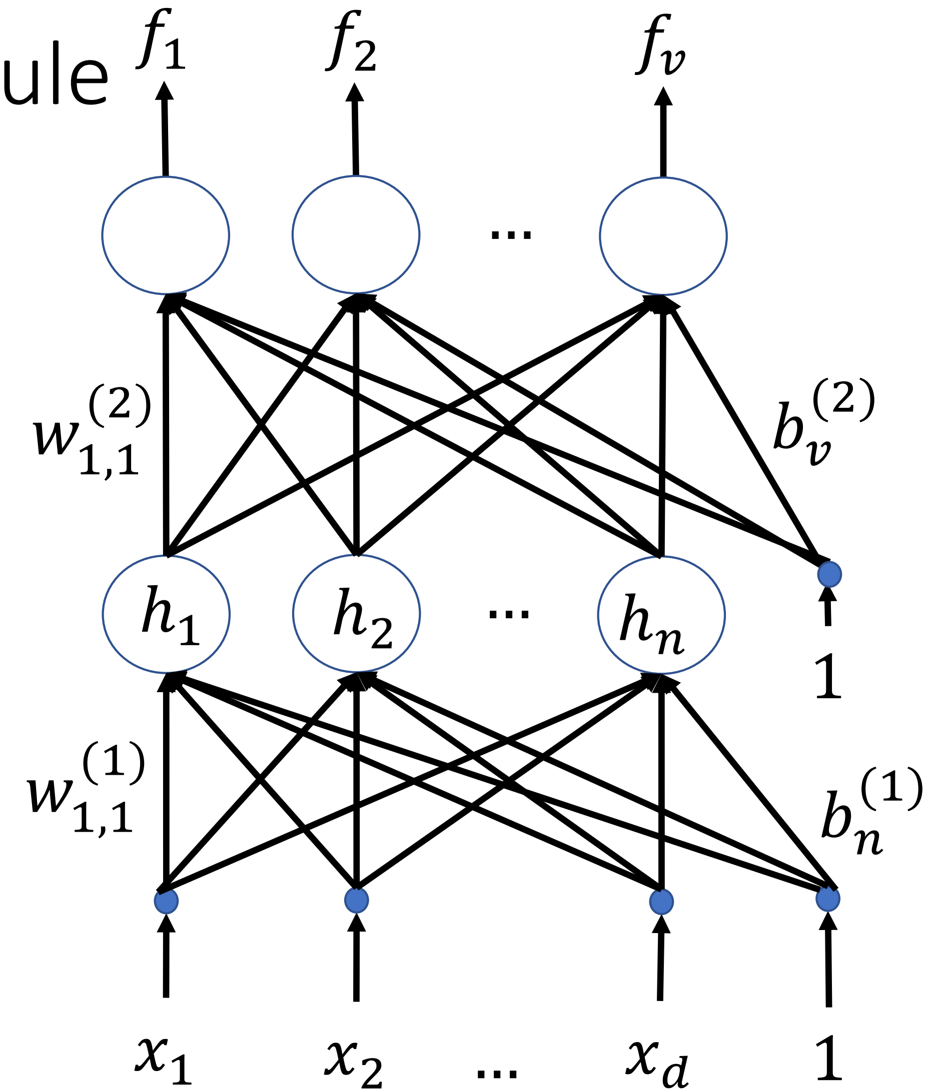
$$\boldsymbol{W}^{(l)} \leftarrow \boldsymbol{W}^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{(l)}}$$



Back-propagation = Chain rule

- Now here's the big question: how do we find $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{(l)}}$?
- Answer: use the chain rule. For example,

$$\frac{\partial \mathcal{L}}{\partial w_{i,j}^{(1)}} = \sum_{k=1}^{v} \left(\frac{\partial \mathcal{L}}{\partial f_k}\right)\left(\frac{\partial f_k}{\partial h_j}\right)\left(\frac{\partial h_j}{\partial w_{i,j}^{(1)}}\right)$$



Excitations and Activations

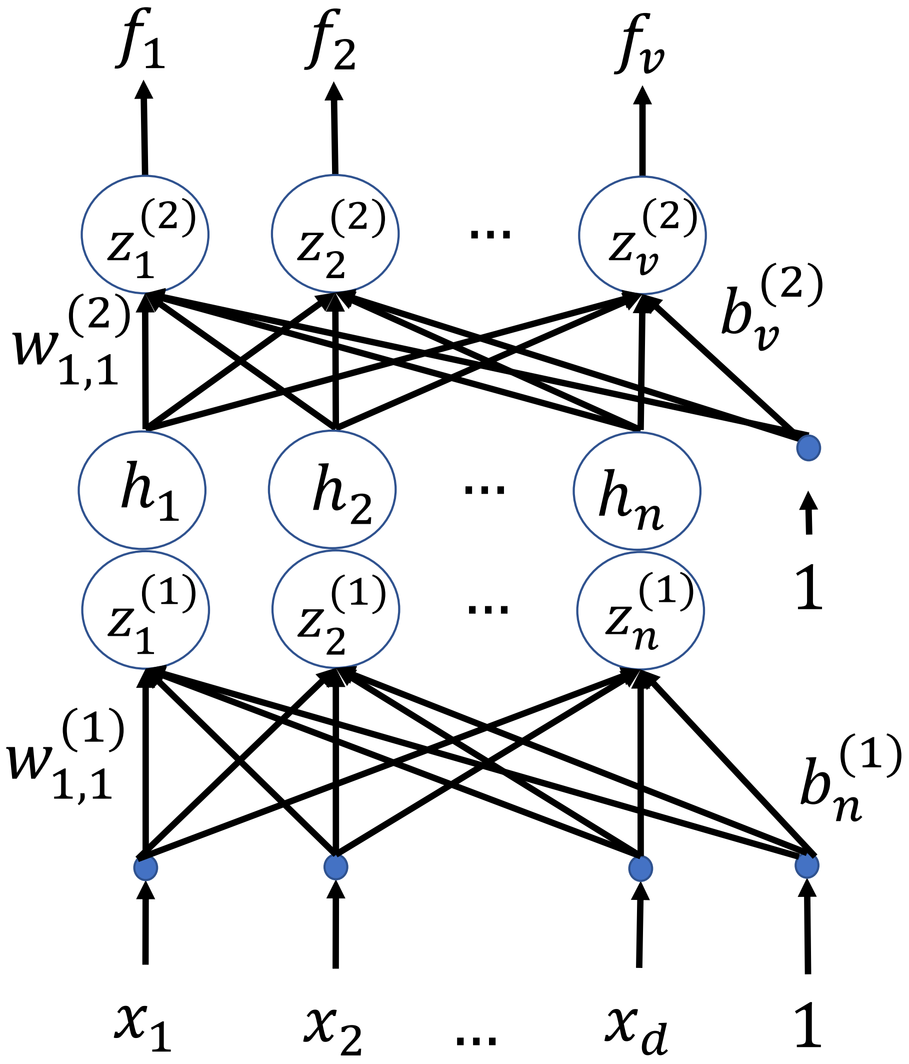
The chain rule is often easier if separate each node's excitation and activation. For example, we could have

$$f_k = \operatorname*{softmax}_k \boldsymbol{z}^{(2)}$$

$$z_k^{(2)} = b_k^{(2)} + \sum_{j=1}^{n} w_{k,j}^{(2)} h_j$$

$$h_j = \mathrm{ReLU}\left(z_j^{(1)}\right)$$

$$z_j^{(1)} = b_j^{(1)} + \sum_{i=1}^{d} w_{j,i}^{(1)} x_i$$



Example

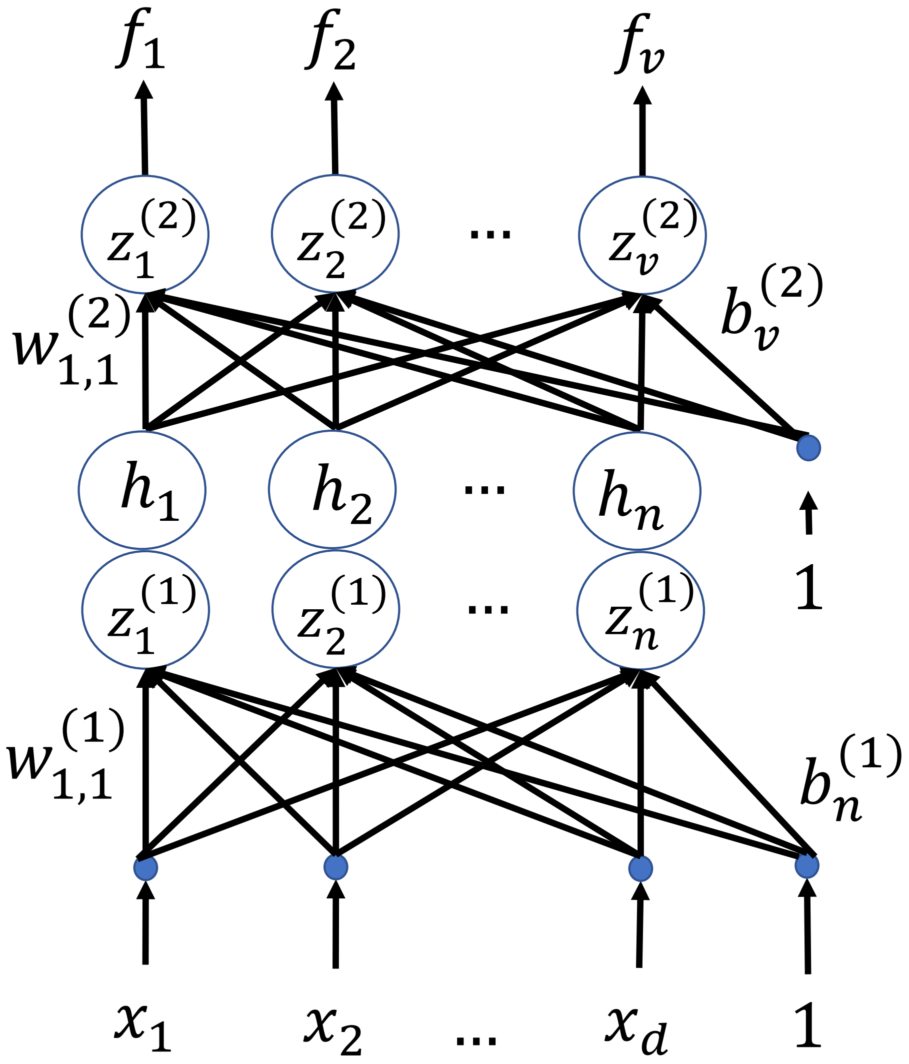
If the loss is cross-entropy, then

$$\frac{\partial \mathcal{L}}{\partial z_k^{(2)}} = f_k - \mathbb{1}_{y=k}$$

So the weight gradient is:

$$\frac{\partial \mathcal{L}}{\partial w_{i,j}^{(1)}} = \sum_{k=1}^{v} \left(\frac{\partial \mathcal{L}}{\partial z_k^{(2)}}\right)\left(\frac{\partial z_k^{(2)}}{\partial h_j}\right)\left(\frac{\partial h_j}{\partial w_{i,j}^{(1)}}\right)$$

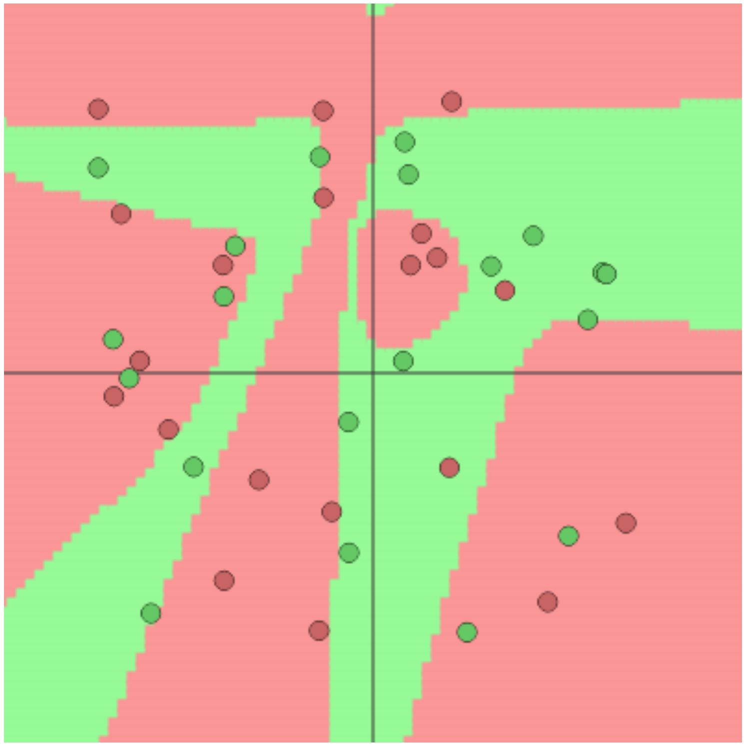
$$= \sum_{k=1}^{v} (f_k - \mathbb{1}_{y=k}) w_{k,j}^{(2)} \mathbb{1}_{h_j>0} x_i$$



Try the quiz!

Try the quiz:
https://us.prairielearn.com/pl/course_instance/147925/assessment/2397863

Approximating an arbitrary nonlinear boundary using a two-layer network



https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

How to train a neural network

- From a very large training dataset, randomly choose a training token $(\boldsymbol{x}_i, y_i)$
- Calculate the neural net prediction, $\boldsymbol{f}(\boldsymbol{x}_i)$
- Calculate the loss, e.g., $\mathcal{L} = -\log f_{y_i}(\boldsymbol{x}_i)$
- Back-propagate to find the gradients, $\frac{\partial\mathcal{L}}{\partial\boldsymbol{W}^{(2)}}$ and $\frac{\partial\mathcal{L}}{\partial\boldsymbol{W}^{(1)}}$
- Do a gradient update step, $\boldsymbol{W}^{(l)} \leftarrow \boldsymbol{W}^{(l)} - \eta\frac{\partial\mathcal{L}}{\partial\boldsymbol{W}^{(l)}}$
- Repeat until the loss is small enough.