
The Ancient
Secrets
Computer Vision

Logistics:

- Homework 3 is out!
  - Optical flow
  - Won’t work on newest OpenCV 3.4
  - Should work on OpenCV 2, maybe early versions of 3
  - demo!


Previously
On
the
Ancient Secrets
of Computer Vision

Softmax: normalized exponential

Generalization of logistic

Input: vector of reals

Output: probability distribution

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

softmax([1,2,7,3,2]):

- Calculate e^x: [2.72, 7.39, 1096.63, 20.09, 7.39]
- Calculate sum(e^x): 2.72+7.39+1096.63+20.09+7.39 = 1134.22
- Normalize: e^x/sum(e^x) = [0.002, 0.007, 0.967, 0.017, 0.007]

Multinomial logistic regression

Probability of that a data point belongs to a class is the normalized, weighted sum of the input variables with the learned weights.

softmax(**wx** + **b**)

$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

Multinomial logistic regression

Probability of that a data point belongs to a class is the normalized, weighted sum of the input variables with the learned weights.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

https://www.tensorflow.org/versions/r1.1/get_started/mnist/beginners

MNIST: Handwriting recognition

50,000 images of handwriting

28 x 28 x 1 (grayscale)

Numbers 0-9

10 class softmax regression

Input is 784 pixel values

Train with SGD

> 95% accuracy

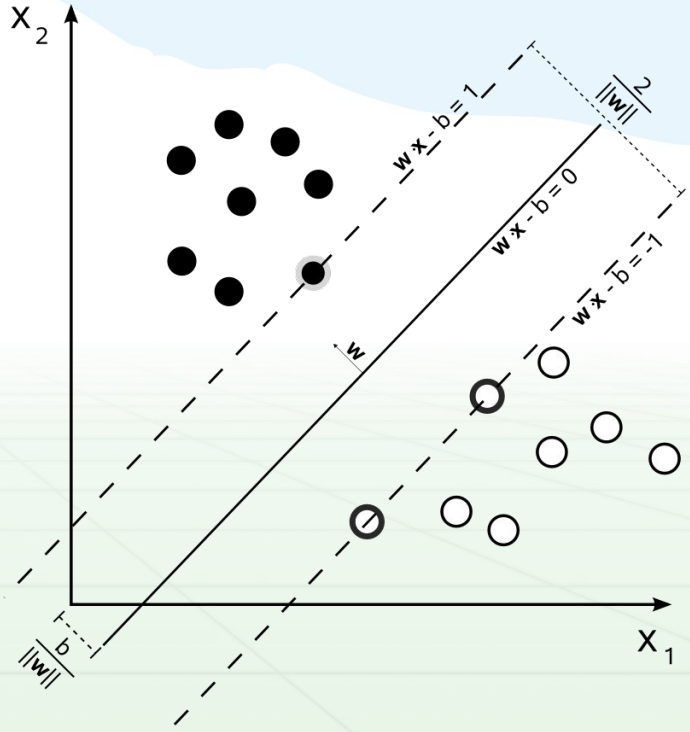
Support Vector Machine (SVM)

Find max-margin classifier. Examples on the margin are supporting data points, support vectors.

$\min ||\mathbf{w}||_2$

s.t. $y_n(\mathbf{w} \cdot \mathbf{x}_n - b) \geq 1,\ n = 1, 2 \ldots$

Or: minimize weights such that margin for each point is at least 1



Case study: Person detection

Dalal and Triggs ‘05:

Train SVM on HOG features of image

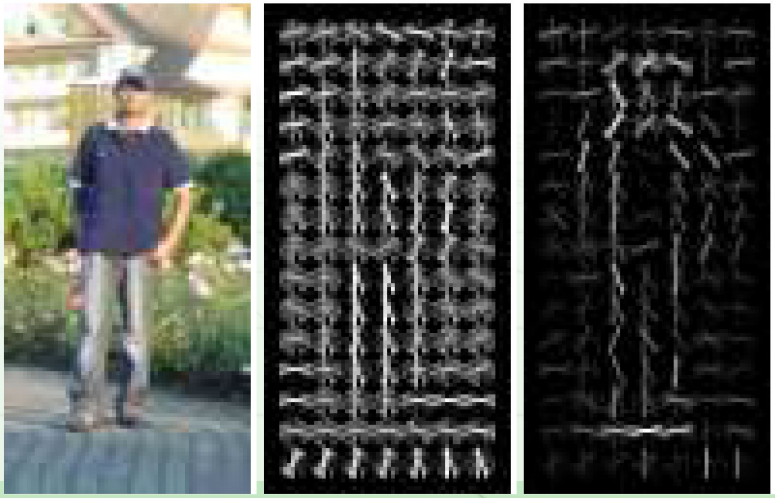
2 classes, person/not person

At test time:

Extract HOG features at many scales

Run SVM classifier at every location

High responses = person?



Case study: Person detection

Dalal and Triggs is a sliding window detector

Many scales
Every location

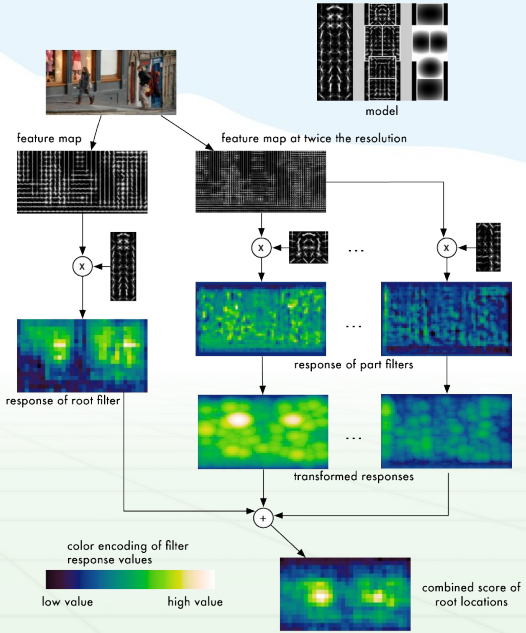
10k+ classifier evaluations per image.

Case study: Deformable parts models

Objects have parts, learn to recognize parts and where they are

Latent SVM: Learn part appearances and locations without explicit data

Hard negative mining: rebalance classes for sliding window detectors



http://cs.brown.edu/people/pfelzens/papers/lsvm-pami.pdf

Case study: Image classification

Given an image, what's in it?

Old state-of-the-art:

Extract features from image

    SIFT and Fisher Vectors

Train Linear SVM

On 1000 different classes, 54% accurate

https://lear.inrialpes.fr/~verbeek/mlcr.slides.11.12/sanchez11cvpr.pdf

What’s wrong with this?

Machine learning needs features!!

What are the right features?

HOG?

SIFT?

FV?

Why not let the algorithm decide

Neural networks: Feature extraction + linear model

Success of Neural Networks

Image classification:

54% -> 80% accuracy on 1000 classes

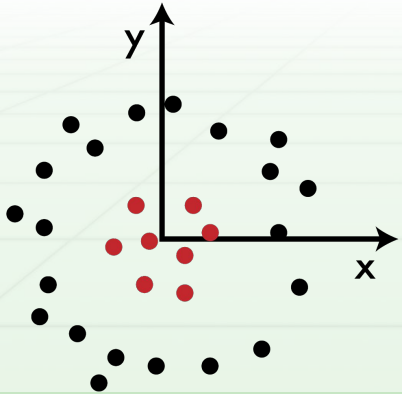
Object detection:

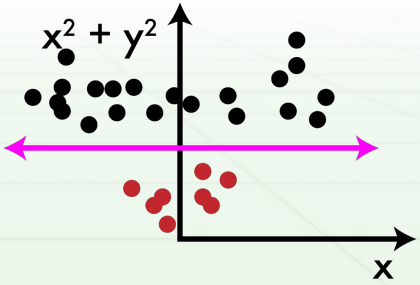
33% mAP (DPM) -> 88% mAP on 20 classes

Feature engineering

Arguably the **core** problem of machine learning (especially in practice)

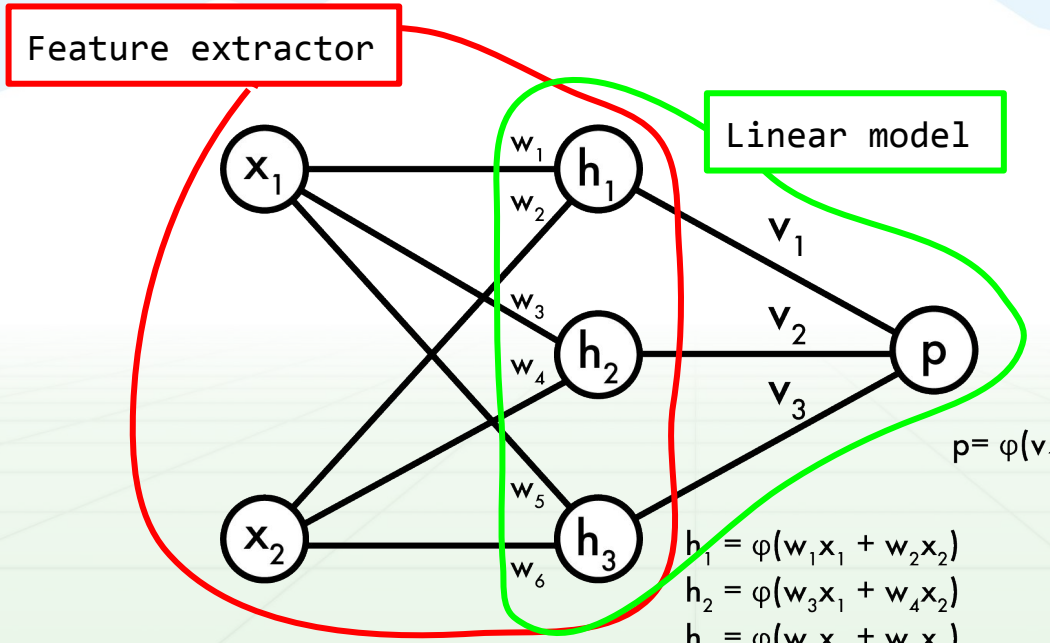
ML models work well if there is a clear relationship between the inputs and outputs of the function you are trying to model





What if we added more processing?

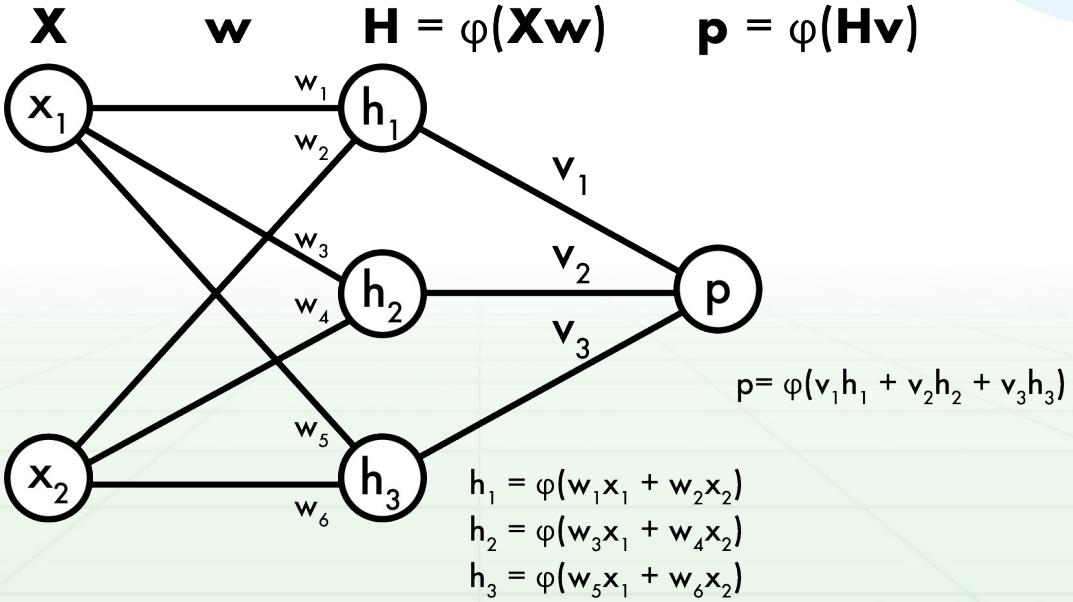
Now our prediction p is a function of our hidden layer



This is a neural network!

This one has 1 hidden layer, but can have **way** more

Each layer is just some function φ applied to linear combination of the previous layer



Universal approximation theorem

What if φ not linear?

Universal approximation theorem (Cybenko 89, Hornik 91)

φ: any nonconstant, bounded, monotonically increasing function

I_m: m-dimensional unit hypercube (interval [0-1] in m-d)

Then 1-layer neural network with φ as activation can model any continuous function f: I_m -> R

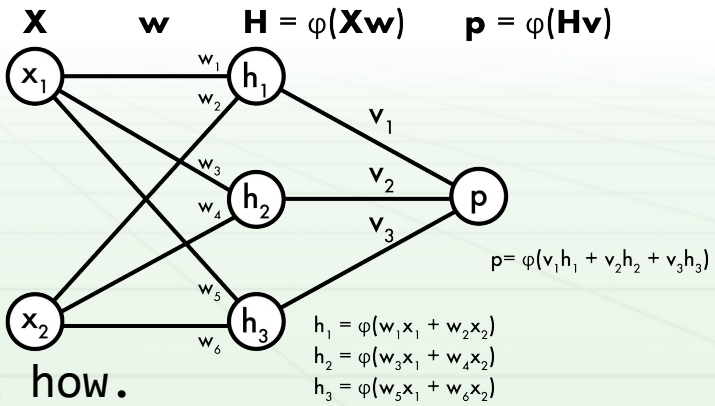
(no bound on size of hidden layer)

By extension, works on f: bounded R^m -> R

What can we learn? What can't we?

UAT just says it's possible to model, not how.

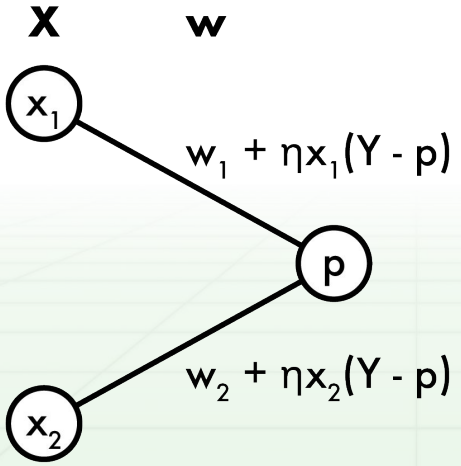
https://en.wikipedia.org/wiki/Universal_approximation_theorem



How do we learn it?

With gradient descent we calculate the partial derivatives of the loss (or likelihood) function for every weight: $\partial/\partial w_i \log L(\mathbf{w})$

Then do gradient descent (or ascent) by adding gradient to weight



$p = f(w_1x_1 + w_2x_2)$

$p = f(\mathbf{w}\cdot\mathbf{X})$

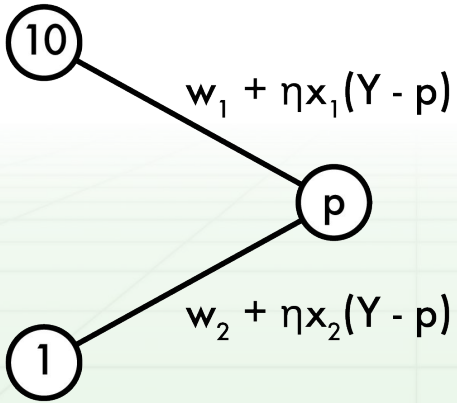
$\nabla \log L(\mathbf{w}) = \mathbf{X}[\mathbf{Y} - f(\mathbf{w}\cdot\mathbf{X})]$

Weight update rule:

$\mathbf{w} = \mathbf{w} + \eta\, \mathbf{X}[\mathbf{Y} - f(\mathbf{w}\cdot\mathbf{X})]$

How do we learn it?

Simple example, say we have a data point [10, 1] and we predict some p. We also know the “correct” label Y. Maybe our prediction p is too small and we want to make it larger. How do we adjust **w**?



Want p to be larger...

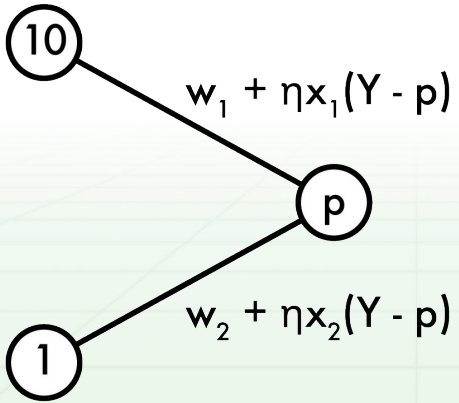
(Y - p) is positive

How do we change w_1 and w_2?

How do we learn it?

Simple example, say we have a data point [10, 1] and we predict some p. We also know the "correct" label Y. Maybe our prediction p is too small and we want to make it larger. How do we adjust **w**?

We adjust w_1 much more than w_2, why?



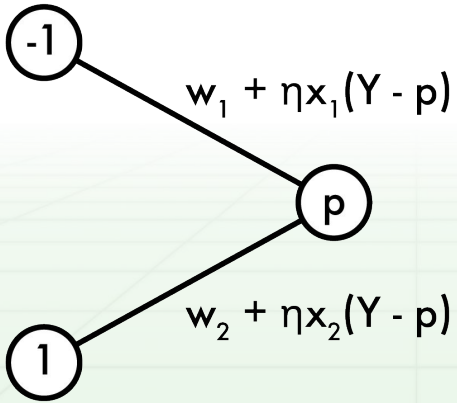
Want p to be larger...

(Y - p) is positive

How do we change w_1 and w_2?

How do we learn it?

Simple example, say we have a data point [-1, 1] and we predict some p. We also know the "correct" label Y. Maybe our prediction p is too small and we want to make it larger. How do we adjust **w**?



Want p to be larger...

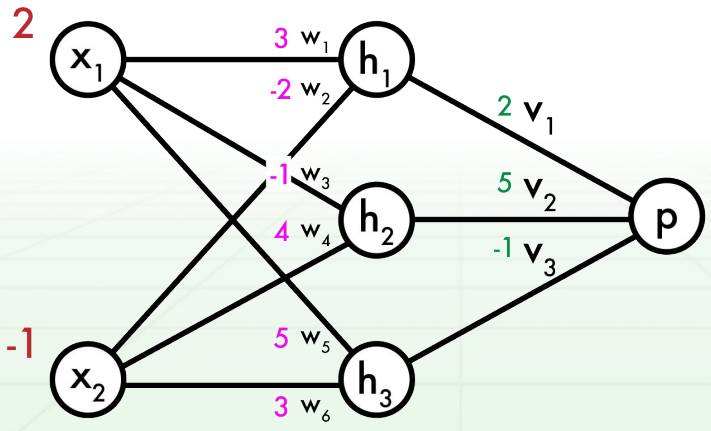
(Y - p) is positive

How do we change w_1 and w_2?


Chapter
Twelve
Neural Networks

How do we learn it?

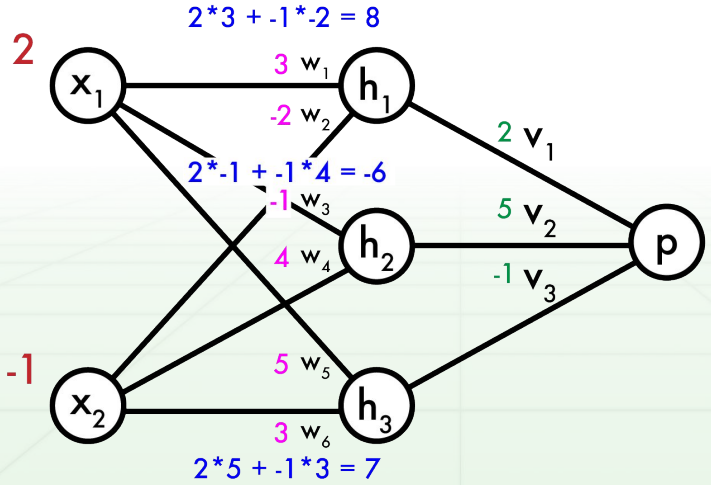
Now we have a “real” neural network (using linear activation for simplicity). How do we predict p?



How do we learn it?

Now we have a “real” neural network (using linear activation for simplicity). How do we predict p?

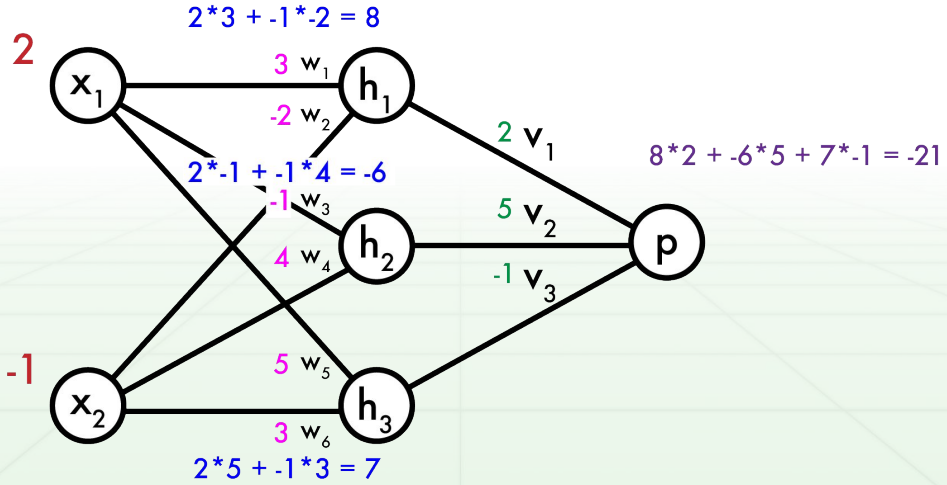
Calculate hidden layer neurons



How do we learn it?

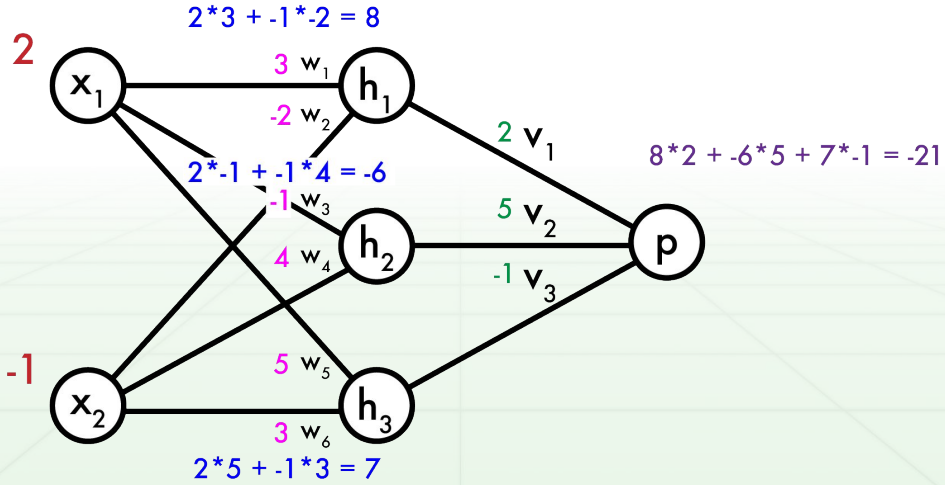
Now we have a “real” neural network (using linear activation for simplicity). How do we predict p?

- Calculate hidden layer neurons
- Calculate output p



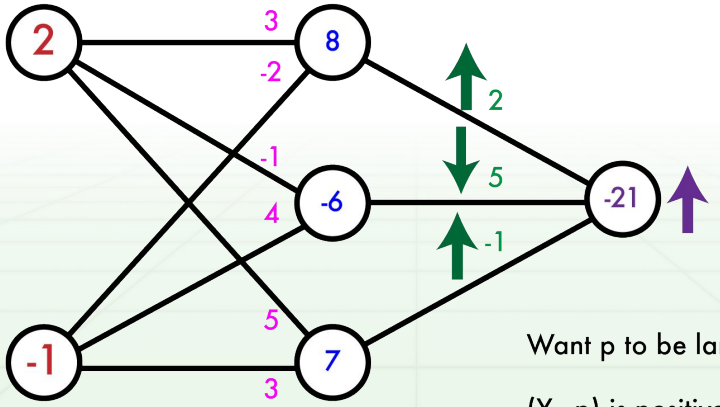
How do we learn it?

Say we want to make p larger. How do we modify the weights? The first layer is easy, same as normal linear model:



How do we learn it?

Say we want to make p larger. How do we modify the weights? The first layer is easy, same as normal linear model:



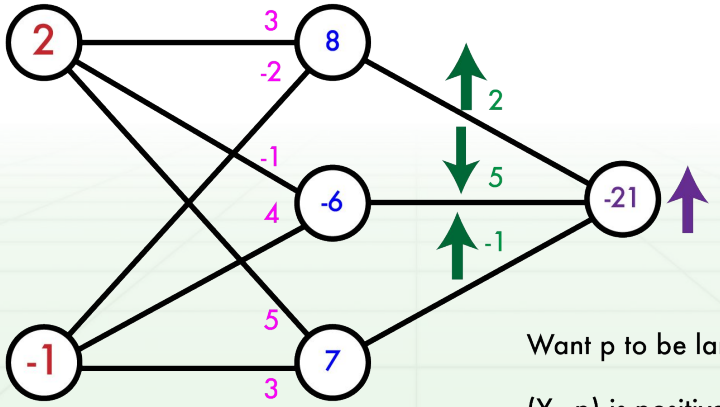
Want p to be larger...

(Y - p) is positive

How do we change our weights?

How do we learn it?

Now what? Let's calculate the "error" that the hidden layer makes. We want p to be larger, given current weights how should we adjust the hidden layer output to do that?



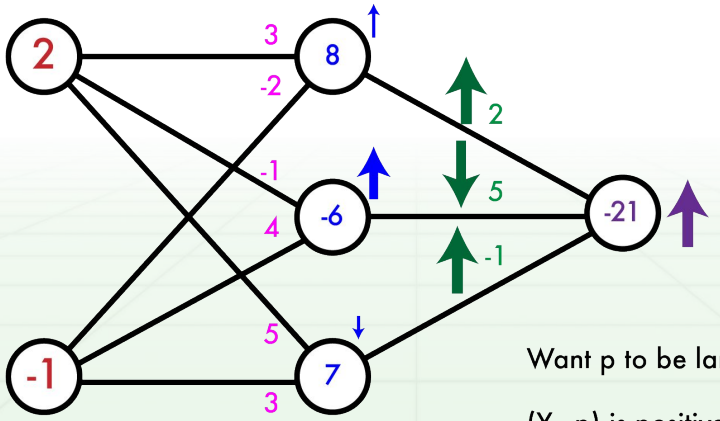
Want p to be larger...

(Y - p) is positive

How do we change our weights?

How do we learn it?

Now what? Let's calculate the "error" that the hidden layer makes. We want p to be larger, given current weights how should we adjust the hidden layer output to do that?



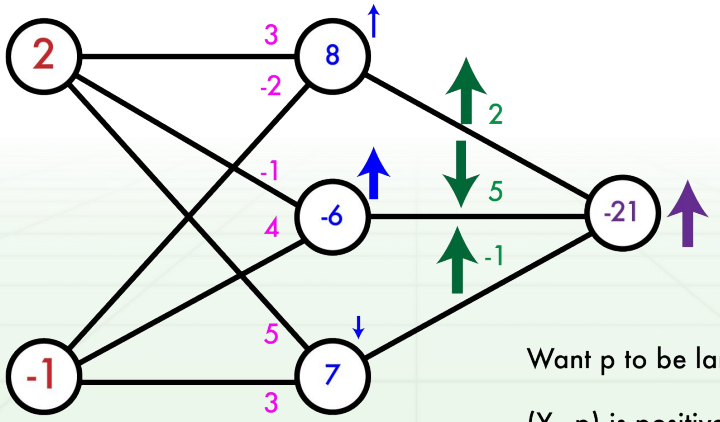
Want p to be larger...

(Y - p) is positive

How do we change our weights?

How do we learn it?

Now that we have an “error” in our hidden layer, want to modify the previous weights. Easy again, just like our linear model.



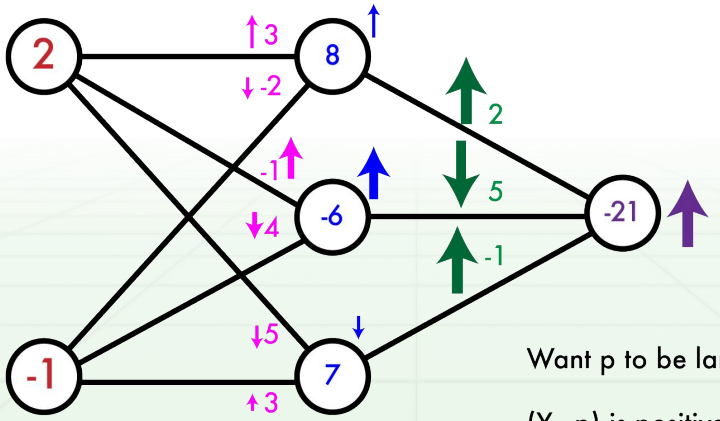
Want p to be larger...

(Y - p) is positive

How do we change our weights?

How do we learn it?

Now that we have an “error” in our hidden layer, want to modify the previous weights. Easy again, just like our linear model.



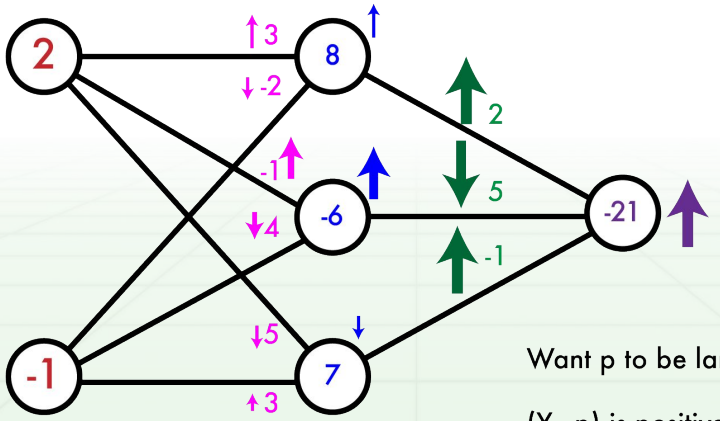
Want p to be larger...

(Y - p) is positive

How do we change our weights?

Backpropagation: just taking derivatives

This is the backpropagation algorithm. It's really just an easy way to calculate partial derivatives in a neural network. We forward-propagate information through the network, calculate our error, then backpropagate that error through network to calculate weight updates.



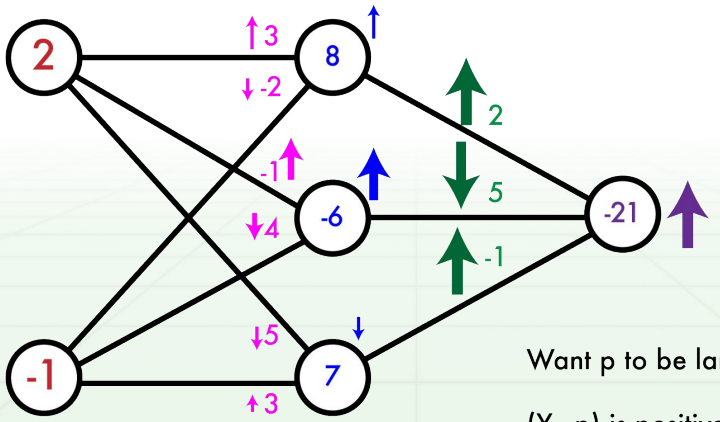
Want p to be larger...

(Y - p) is positive

How do we change our weights?

Backpropagation: just taking derivatives

This was with linear activations but the process is the same for any φ, just have to calculate φ’(x) for that neuron as well.



Want p to be larger...

(Y - p) is positive

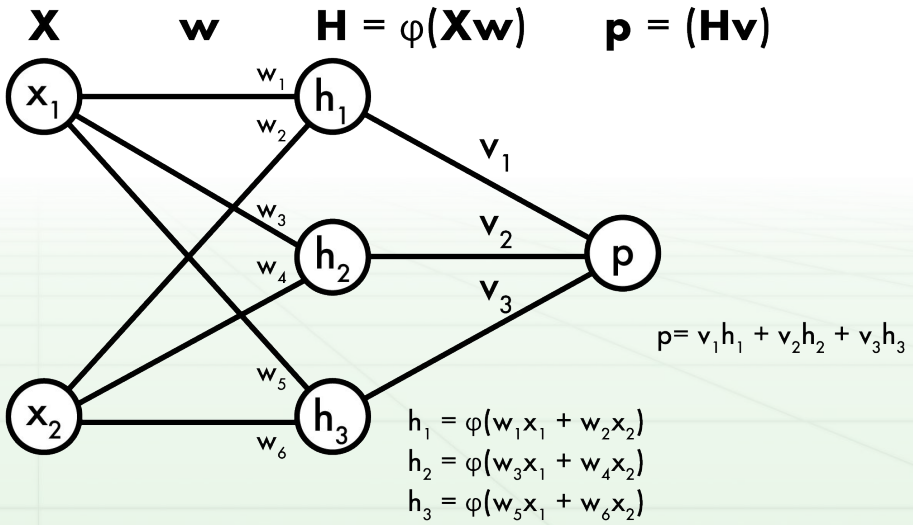
How do we change our weights?

Backpropagation: the math

1-layer NN, sigmoid activation at hidden layer, linear output:

$$F(\mathbf{X}) = \varphi(\mathbf{X}\mathbf{w})\mathbf{v}$$

$$F(\mathbf{X}) = \varphi(x_1 w_1 + x_2 w_2)v_1 + \varphi(x_1 w_3 + x_2 w_4)v_2 + \varphi(x_1 w_5 + x_2 w_6)v_3$$



Backpropagation: the math

1-layer NN, sigmoid activation at hidden layer, linear output:

$$F(\mathbf{X}) = \varphi(\mathbf{X}\mathbf{w})\mathbf{v}$$

$$F(\mathbf{X}) = \varphi(x_1 w_1 + x_2 w_2)v_1 + \varphi(x_1 w_3 + x_2 w_4)v_2 + \varphi(x_1 w_5 + x_2 w_6)v_3$$

Say regression, Loss function is ½ L_2 norm, expected output is Y:

$$L_{X,Y}(\mathbf{w},\mathbf{v}) = \tfrac{1}{2}(Y - F(\mathbf{X}))^2 = \tfrac{1}{2}(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})^2$$

Backpropagation: the math

1-layer NN, sigmoid activation at hidden layer, linear output:

$$F(\mathbf{X}) = \varphi(\mathbf{X}\mathbf{w})\mathbf{v}$$

$$F(\mathbf{X}) = \varphi(x_1 w_1 + x_2 w_2)v_1 + \varphi(x_1 w_3 + x_2 w_4)v_2 + \varphi(x_1 w_5 + x_2 w_6)v_3$$

Say regression, Loss function is ½ L_2 norm, expected output is Y:

$$L_{X,Y}(\mathbf{w},\mathbf{v}) = \tfrac{1}{2}(Y - F(\mathbf{X}))^2 = \tfrac{1}{2}(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})^2$$

Want partial derivative of Loss w.r.t. weights, say v_2:

$$\partial/\partial v_2 \; L_{X,Y}(\mathbf{w},\mathbf{v}) = \partial/\partial v_2 \; \tfrac{1}{2}(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})^2$$

Backpropagation: the math

1-layer NN, sigmoid activation at hidden layer, linear output:

$$F(\mathbf{X}) = \varphi(\mathbf{X}\mathbf{w})\mathbf{v}$$

$$F(\mathbf{X}) = \varphi(x_1 w_1 + x_2 w_2)v_1 + \varphi(x_1 w_3 + x_2 w_4)v_2 + \varphi(x_1 w_5 + x_2 w_6)v_3$$

Say regression, Loss function is ½ L_2 norm, expected output is Y:

$$L_{X,Y}(\mathbf{w},\mathbf{v}) = \tfrac{1}{2}(Y - F(\mathbf{X}))^2 = \tfrac{1}{2}(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})^2$$

Want partial derivative of Loss w.r.t. weights, say v_2:

$$\partial/\partial v_2 \; L_{X,Y}(\mathbf{w},\mathbf{v}) = \partial/\partial v_2 \; \tfrac{1}{2}(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})^2$$

$$= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -[\partial/\partial v_2 \varphi(\mathbf{X}\mathbf{w})\mathbf{v}]$$

Backpropagation: the math

1-layer NN, sigmoid activation at hidden layer, linear output:

$$F(\mathbf{X}) = \varphi(\mathbf{X}\mathbf{w})\mathbf{v}$$

$$F(\mathbf{X}) = \varphi(x_1 w_1 + x_2 w_2)v_1 + \varphi(x_1 w_3 + x_2 w_4)v_2 + \varphi(x_1 w_5 + x_2 w_6)v_3$$

Say regression, Loss function is ½ L_2 norm, expected output is Y:

$$L_{X,Y}(\mathbf{w},\mathbf{v}) = \tfrac{1}{2}(Y - F(\mathbf{X}))^2 = \tfrac{1}{2}(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})^2$$

Want partial derivative of Loss w.r.t. weights, say v_2:

$$\begin{aligned}\partial/\partial v_2\ L_{X,Y}(\mathbf{w},\mathbf{v}) &= \partial/\partial v_2\ \tfrac{1}{2}(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})^2 \\ &= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -[\partial/\partial v_2 \varphi(\mathbf{X}\mathbf{w})\mathbf{v}] \\ &= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -[\partial/\partial v_2\ \varphi(x_1 w_3 + x_2 w_4)v_2]\end{aligned}$$

Backpropagation: the math

1-layer NN, sigmoid activation at hidden layer, linear output:

$$F(\mathbf{X}) = \varphi(\mathbf{X}\mathbf{w})\mathbf{v}$$

$$F(\mathbf{X}) = \varphi(x_1 w_1 + x_2 w_2)v_1 + \varphi(x_1 w_3 + x_2 w_4)v_2 + \varphi(x_1 w_5 + x_2 w_6)v_3$$

Say regression, Loss function is ½ L_2 norm, expected output is Y:

$$L_{X,Y}(\mathbf{w},\mathbf{v}) = \tfrac{1}{2}(Y - F(\mathbf{X}))^2 = \tfrac{1}{2}(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})^2$$

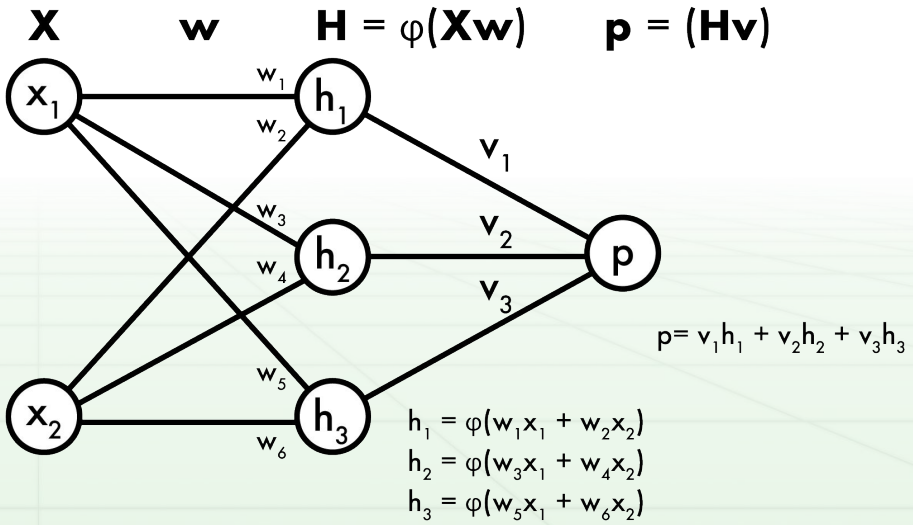
Want partial derivative of Loss w.r.t. weights, say v_2:

$$\begin{aligned}
\partial/\partial v_2 \; L_{X,Y}(\mathbf{w},\mathbf{v}) &= \partial/\partial v_2 \; \tfrac{1}{2}(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})^2 \\
&= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -[\partial/\partial v_2 \varphi(\mathbf{X}\mathbf{w})\mathbf{v}] \\
&= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -[\partial/\partial v_2 \; \varphi(x_1 w_3 + x_2 w_4)v_2] \\
&= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -\varphi(x_1 w_3 + x_2 w_4)
\end{aligned}$$

Backpropagation: the math

Want partial derivative of Loss w.r.t. weights, say v_2:

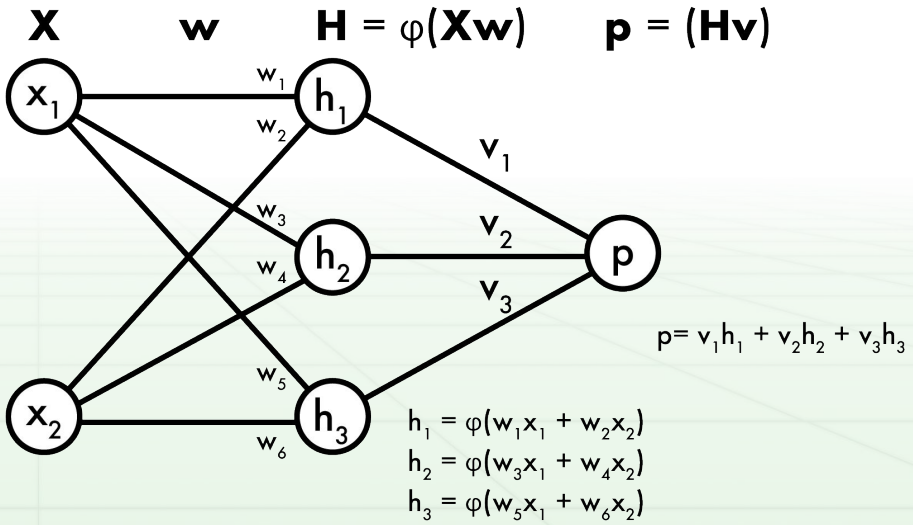
$$\partial/\partial v_2 \; L_{x,y}(\mathbf{w},\mathbf{v}) = (Y - \varphi(\mathbf{Xw})\mathbf{v}) * -\varphi(x_1 w_3 + x_2 w_4)$$



Backpropagation: the math

Want partial derivative of Loss w.r.t. weights, say v_2:

$$\partial/\partial v_2 \; L_{x,Y}(\mathbf{w},\mathbf{v}) = -(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})*h_2$$



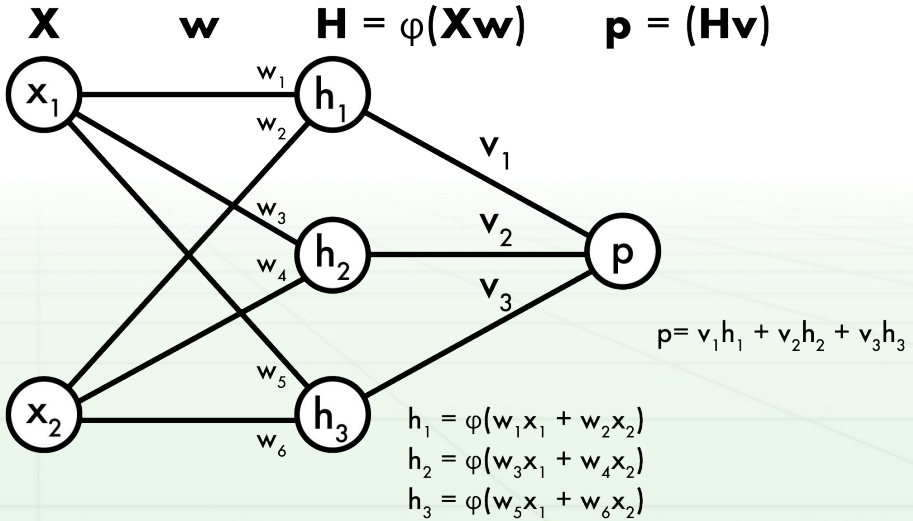
Backpropagation: the math

Want partial derivative of Loss w.r.t. weights, say v_2:

$$\partial/\partial v_2 \; L_{x,Y}(\mathbf{w},\mathbf{v}) = -(Y - \varphi(\mathbf{Xw})\mathbf{v})*h_2$$

Weight update rule (remember descend on loss):

$$v_2 = v_2 + \eta(Y - \varphi(\mathbf{Xw})\mathbf{v})*h_2$$



Backpropagation: the math

1-layer NN, sigmoid activation at hidden layer, linear output:

$$F(\mathbf{X}) = \varphi(\mathbf{X}\mathbf{w})\mathbf{v}$$

$$F(\mathbf{X}) = \varphi(x_1 w_1 + x_2 w_2)v_1 + \varphi(x_1 w_3 + x_2 w_4)v_2 + \varphi(x_1 w_5 + x_2 w_6)v_3$$

Say regression, Loss function is ½ L_2 norm, expected output is Y:

$$L_{X,Y}(\mathbf{w},\mathbf{v}) = \tfrac{1}{2}(Y - F(\mathbf{X}))^2 = \tfrac{1}{2}(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})^2$$

Want partial derivative of Loss w.r.t. weights, say w_2:

$$\partial/\partial w_2 \; L_{X,Y}(\mathbf{w},\mathbf{v}) = \partial/\partial w_2 \; \tfrac{1}{2}(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})^2$$

$$= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -[\partial/\partial w_2 \; \varphi(\mathbf{X}\mathbf{w})\mathbf{v}]$$

Backpropagation: the math

1-layer NN, sigmoid activation at hidden layer, linear output:

$$F(\mathbf{X}) = \varphi(\mathbf{X}\mathbf{w})\mathbf{v}$$

$$F(\mathbf{X}) = \varphi(x_1 w_1 + x_2 w_2)v_1 + \varphi(x_1 w_3 + x_2 w_4)v_2 + \varphi(x_1 w_5 + x_2 w_6)v_3$$

Say regression, Loss function is ½ L_2 norm, expected output is Y:

$$L_{X,Y}(\mathbf{w},\mathbf{v}) = \tfrac{1}{2}(Y - F(\mathbf{X}))^2 = \tfrac{1}{2}(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})^2$$

Want partial derivative of Loss w.r.t. weights, say w_2:

$$\begin{aligned} \partial/\partial w_2 \; L_{X,Y}(\mathbf{w},\mathbf{v}) &= \partial/\partial w_2 \; \tfrac{1}{2}(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})^2 \\ &= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -[\partial/\partial w_2 \; \varphi(\mathbf{X}\mathbf{w})\mathbf{v}] \\ &= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -[\partial/\partial w_2 \; \varphi(x_1 w_1 + x_2 w_2)v_1] \end{aligned}$$

Backpropagation: the math

1-layer NN, sigmoid activation at hidden layer, linear output:

$$F(\mathbf{X}) = \varphi(\mathbf{X}\mathbf{w})\mathbf{v}$$

$$F(\mathbf{X}) = \varphi(x_1 w_1 + x_2 w_2)v_1 + \varphi(x_1 w_3 + x_2 w_4)v_2 + \varphi(x_1 w_5 + x_2 w_6)v_3$$

Say regression, Loss function is ½ L_2 norm, expected output is Y:

$$L_{X,Y}(\mathbf{w},\mathbf{v}) = \tfrac{1}{2}(Y - F(\mathbf{X}))^2 = \tfrac{1}{2}(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})^2$$

Want partial derivative of Loss w.r.t. weights, say w_2:

$$\begin{aligned} \partial/\partial w_2 \; L_{X,Y}(\mathbf{w},\mathbf{v}) &= \partial/\partial w_2 \; \tfrac{1}{2}(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})^2 \\ &= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -[\partial/\partial w_2 \; \varphi(\mathbf{X}\mathbf{w})\mathbf{v}] \\ &= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -v_1[\partial/\partial w_2 \; \varphi(x_1 w_1 + x_2 w_2)] \end{aligned}$$

Backpropagation: the math

1-layer NN, sigmoid activation at hidden layer, linear output:

$$F(\mathbf{X}) = \varphi(\mathbf{X}\mathbf{w})\mathbf{v}$$

$$F(\mathbf{X}) = \varphi(x_1 w_1 + x_2 w_2)v_1 + \varphi(x_1 w_3 + x_2 w_4)v_2 + \varphi(x_1 w_5 + x_2 w_6)v_3$$

Say regression, Loss function is ½ L_2 norm, expected output is Y:

$$L_{X,Y}(\mathbf{w},\mathbf{v}) = \tfrac{1}{2}(Y - F(\mathbf{X}))^2 = \tfrac{1}{2}(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})^2$$

Want partial derivative of Loss w.r.t. weights, say w_i:

$$\begin{aligned}
\partial/\partial w_2 \; L_{X,Y}(\mathbf{w},\mathbf{v}) &= \partial/\partial w_2 \; \tfrac{1}{2}(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})^2 \\
&= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -[\partial/\partial w_2 \; \varphi(\mathbf{X}\mathbf{w})\mathbf{v}] \\
&= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -v_1[\partial/\partial w_2 \; \varphi(x_1 w_1 + x_2 w_2)] \\
&= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -v_1\varphi'(x_1 w_1 + x_2 w_2)[\partial/\partial w_2 \; (x_1 w_1 + x_2 w_2)]
\end{aligned}$$

Chain rule!

If $F(x) = f(g(x))$

$F'(x) = f'(g(x))g'(x)$

Backpropagation: the math

1-layer NN, sigmoid activation at hidden layer, linear output:

$$F(\mathbf{X}) = \varphi(\mathbf{X}\mathbf{w})\mathbf{v}$$

$$F(\mathbf{X}) = \varphi(x_1 w_1 + x_2 w_2)v_1 + \varphi(x_1 w_3 + x_2 w_4)v_2 + \varphi(x_1 w_5 + x_2 w_6)v_3$$

Say regression, Loss function is ½ L_2 norm, expected output is Y:

$$L_{X,Y}(\mathbf{w},\mathbf{v}) = \tfrac{1}{2}(Y - F(\mathbf{X}))^2 = \tfrac{1}{2}(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})^2$$

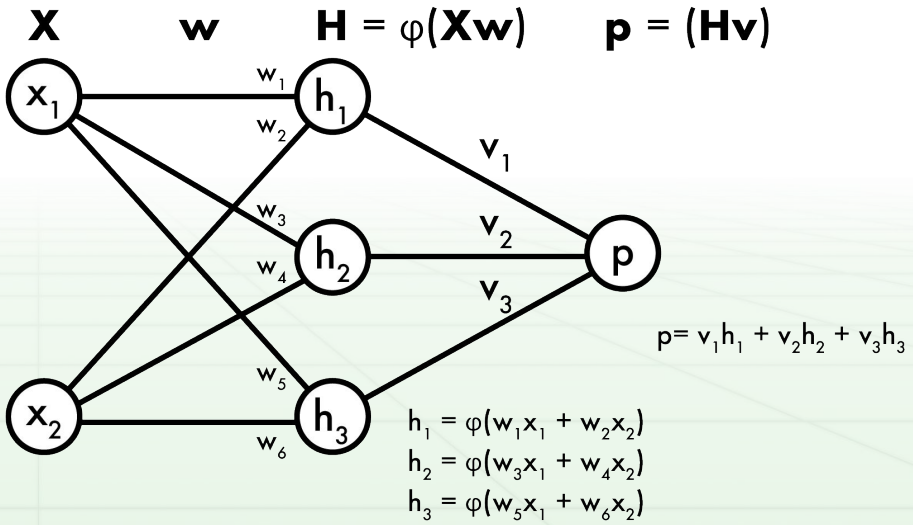
Want partial derivative of Loss w.r.t. weights, say w_2:

$$\begin{aligned}
\partial/\partial w_2 \; L_{X,Y}(\mathbf{w},\mathbf{v}) &= \partial/\partial w_2 \; \tfrac{1}{2}(Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})^2 \\
&= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -[\partial/\partial w_2 \; \varphi(\mathbf{X}\mathbf{w})\mathbf{v}] \\
&= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -v_1[\partial/\partial w_2 \; \varphi(x_1 w_1 + x_2 w_2)] \\
&= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -v_1 \varphi'(x_1 w_1 + x_2 w_2)[\partial/\partial w_2 \; (x_1 w_1 + x_2 w_2)] \\
&= (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -v_1 \varphi'(x_1 w_1 + x_2 w_2) * x_2
\end{aligned}$$

Backpropagation: the math

Want partial derivative of Loss w.r.t. weights, say w_2:

$$\partial/\partial w_2 \; L_{x,y}(\mathbf{w},\mathbf{v}) = (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v}) * -v_1\varphi'(x_1w_1+x_2w_2) * x_2$$

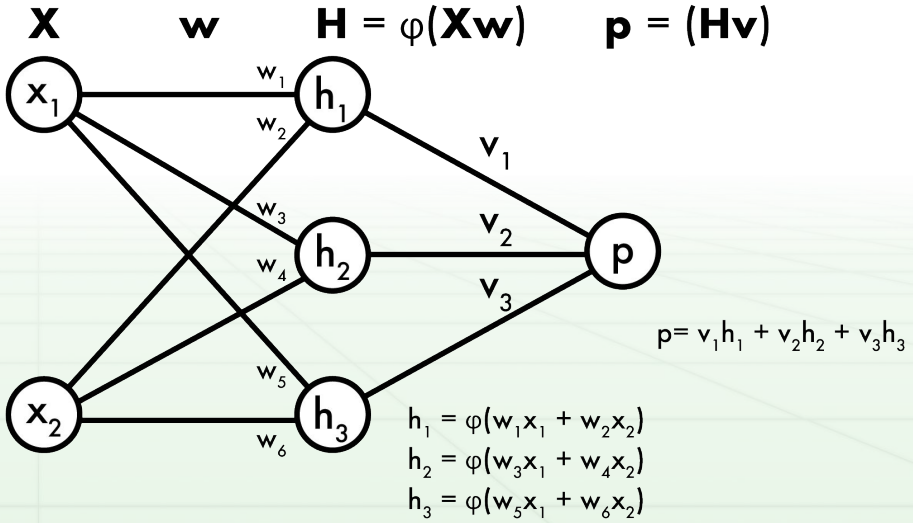


Backpropagation: the math

Want partial derivative of Loss w.r.t. weights, say w_2:

$$\partial/\partial w_2 \; L_{X,Y}(\mathbf{w},\mathbf{v}) = (Y - \varphi(\mathbf{Xw})\mathbf{v}) * -v_1\varphi'(x_1w_1+x_2w_2) * x_2$$

Model error at p



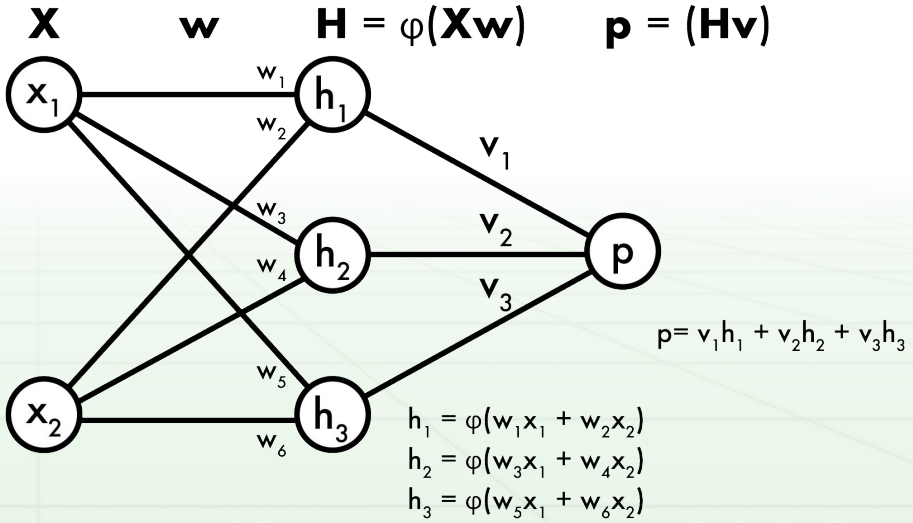
Backpropagation: the math

Want partial derivative of Loss w.r.t. weights, say w_2:

$$\partial/\partial w_2 \; L_{X,Y}(\mathbf{w},\mathbf{v}) = (Y - \varphi(\mathbf{Xw})\mathbf{v}) * -v_1\varphi'(x_1w_1+x_2w_2) * x_2$$

Model error at p

Backpropagate through v_1



Backpropagation: the math

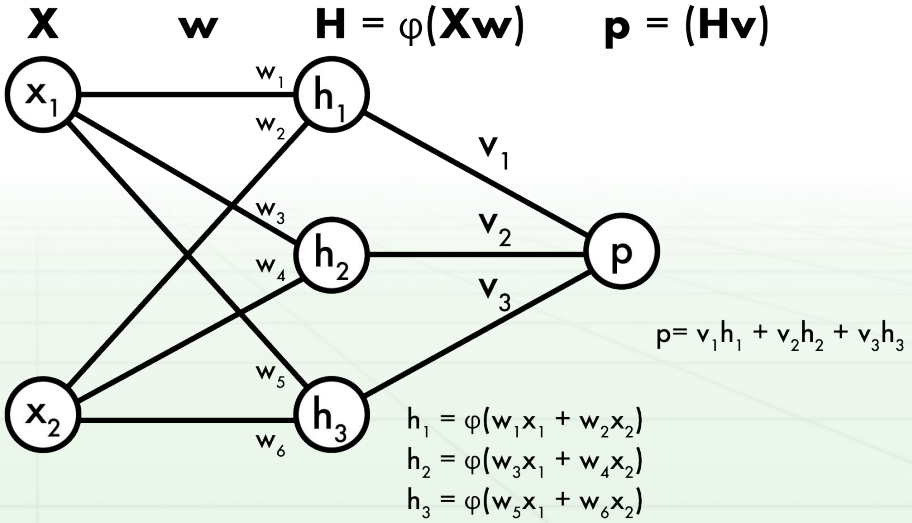
Want partial derivative of Loss w.r.t. weights, say w_2:

$$\partial/\partial w_2 \; L_{x,y}(\mathbf{w},\mathbf{v}) = (Y - \varphi(\mathbf{Xw})\mathbf{v}) * -v_1\varphi'(x_1w_1+x_2w_2) * x_2$$

Model error at p

Model error at h_1

Backpropagate through v_1



Backpropagation: the math

Want partial derivative of Loss w.r.t. weights, say w_2:

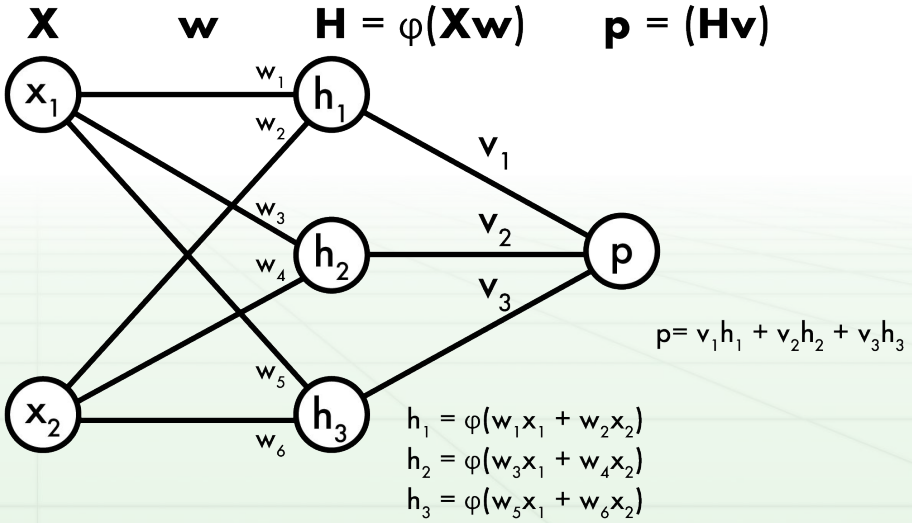
$$\partial/\partial w_2 \; L_{X,Y}(\mathbf{w},\mathbf{v}) = (Y - \varphi(\mathbf{Xw})\mathbf{v}) * -v_1\varphi'(x_1w_1+x_2w_2) * x_2$$

Model error at p

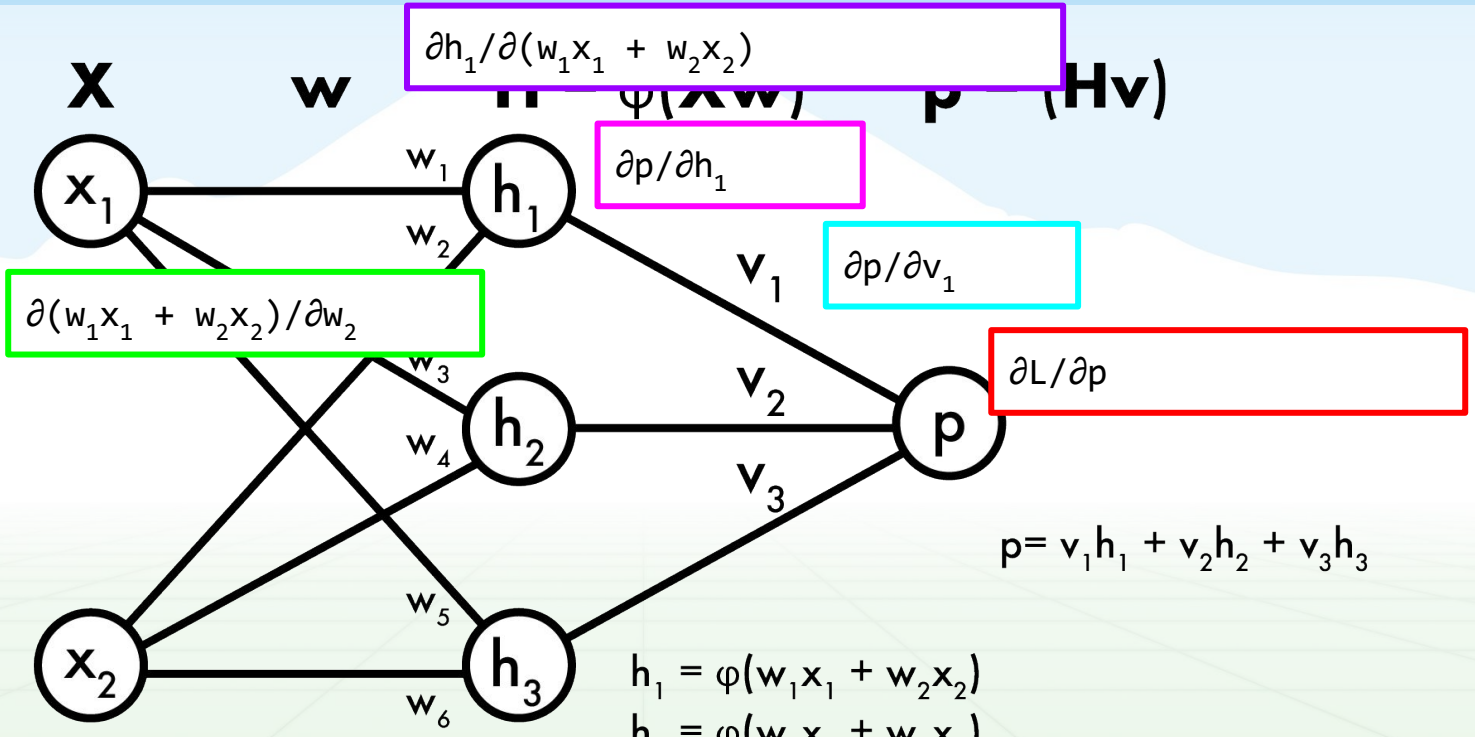
Model error at h_1

Multiply by x_2: gradient w.r.t. w_2

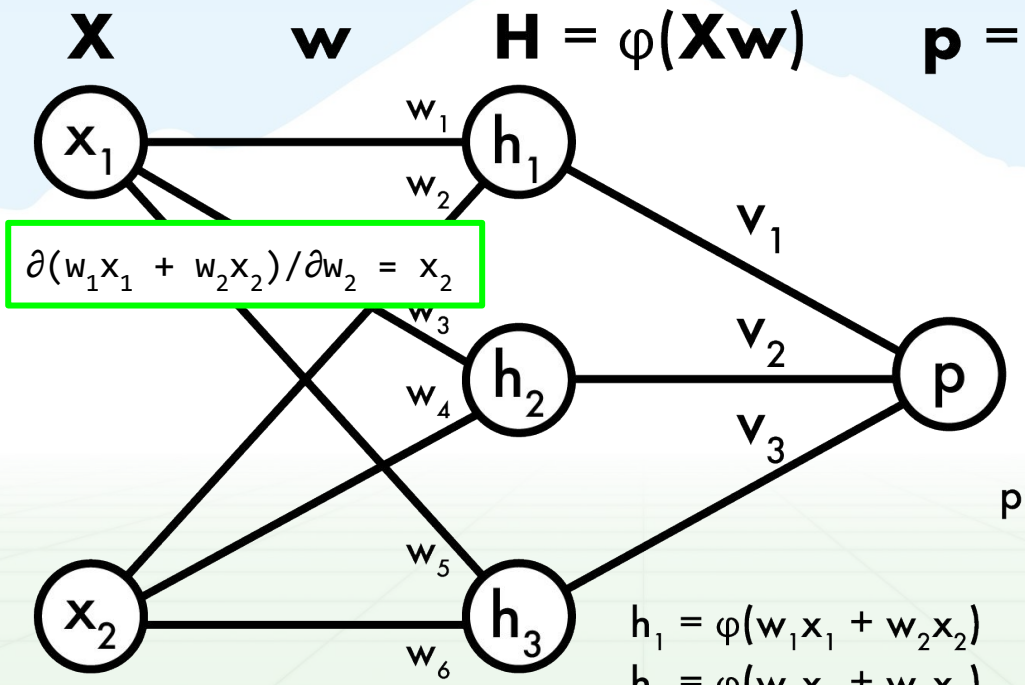
Backpropagate through v_1



Backpropagation: the math



Backpropagation: the math



$p = v_1h_1 + v_2h_2 + v_3h_3$

$h_1 = \varphi(w_1x_1 + w_2x_2)$

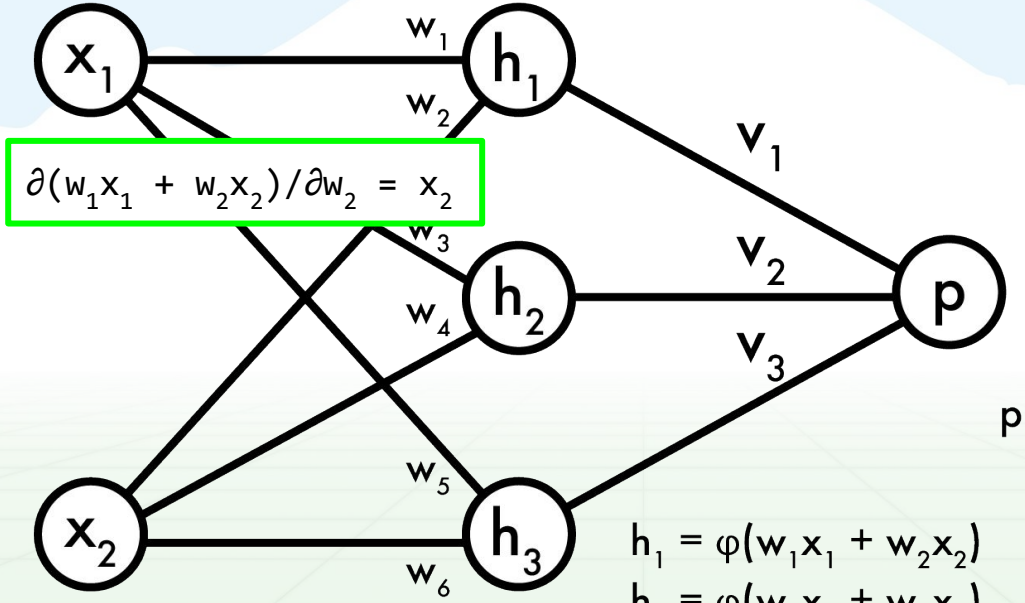
$h_2 = \varphi(w_3x_1 + w_4x_2)$

$h_3 = \varphi(w_5x_1 + w_6x_2)$

Backpropagation: the math

$\partial h_1/\partial(w_1x_1 + w_2x_2) =$
$\varphi'(x_1w_1+x_2w_2)$

X w (Xw) p (Hv)

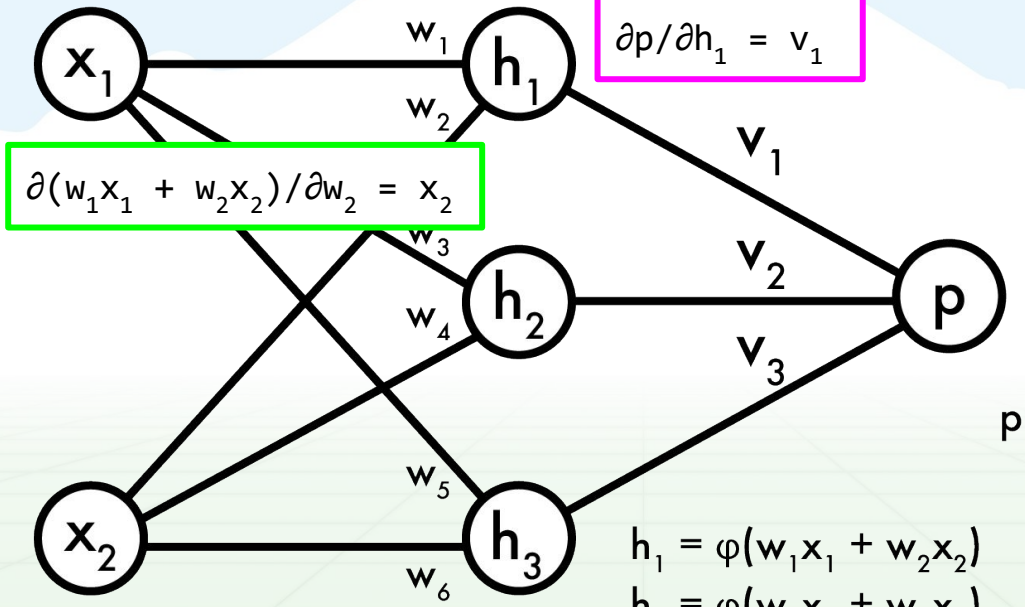


$\partial(w_1x_1 + w_2x_2)/\partial w_2 = x_2$

$p = v_1h_1 + v_2h_2 + v_3h_3$

$h_1 = \varphi(w_1x_1 + w_2x_2)$
$h_2 = \varphi(w_3x_1 + w_4x_2)$
$h_3 = \varphi(w_5x_1 + w_6x_2)$

Backpropagation: the math



$\partial h_1/\partial(w_1x_1 + w_2x_2) =$
$\varphi'(x_1w_1+x_2w_2)$

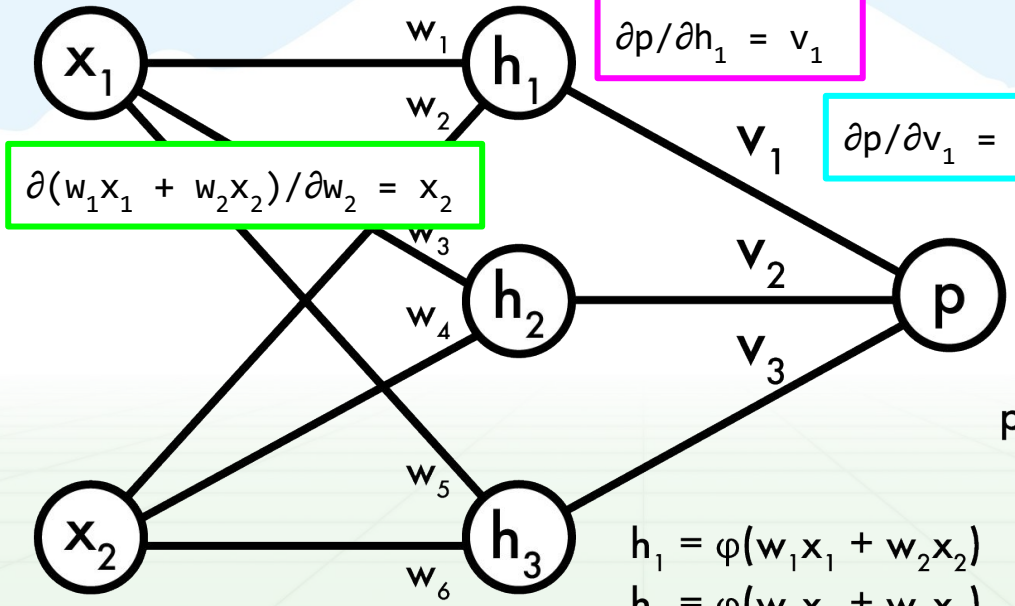
$\partial p/\partial h_1 = v_1$

$\partial(w_1x_1 + w_2x_2)/\partial w_2 = x_2$

$p = v_1h_1 + v_2h_2 + v_3h_3$

$h_1 = \varphi(w_1x_1 + w_2x_2)$
$h_2 = \varphi(w_3x_1 + w_4x_2)$
$h_3 = \varphi(w_5x_1 + w_6x_2)$

Backpropagation: the math



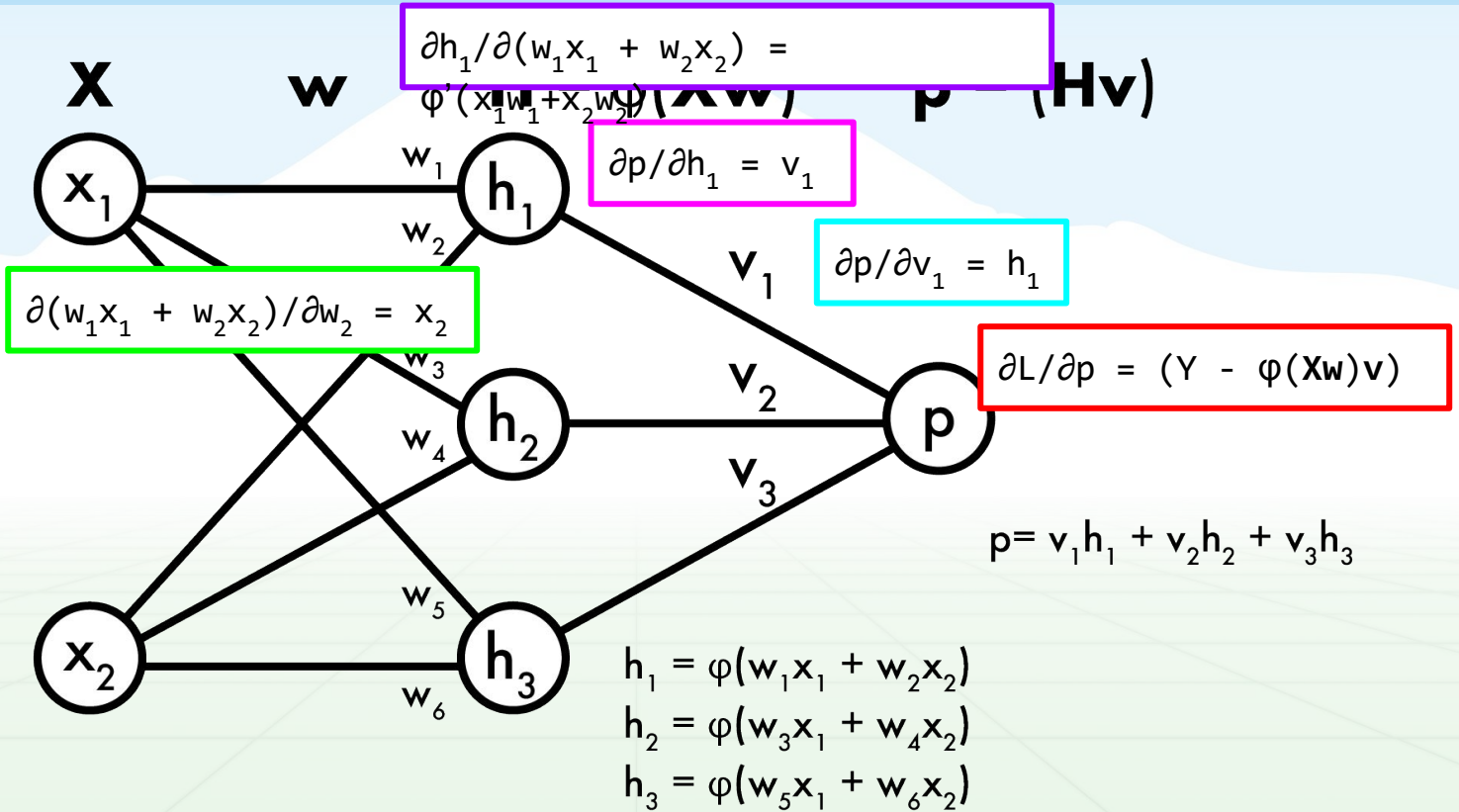
$p = v_1h_1 + v_2h_2 + v_3h_3$

$h_1 = \varphi(w_1x_1 + w_2x_2)$

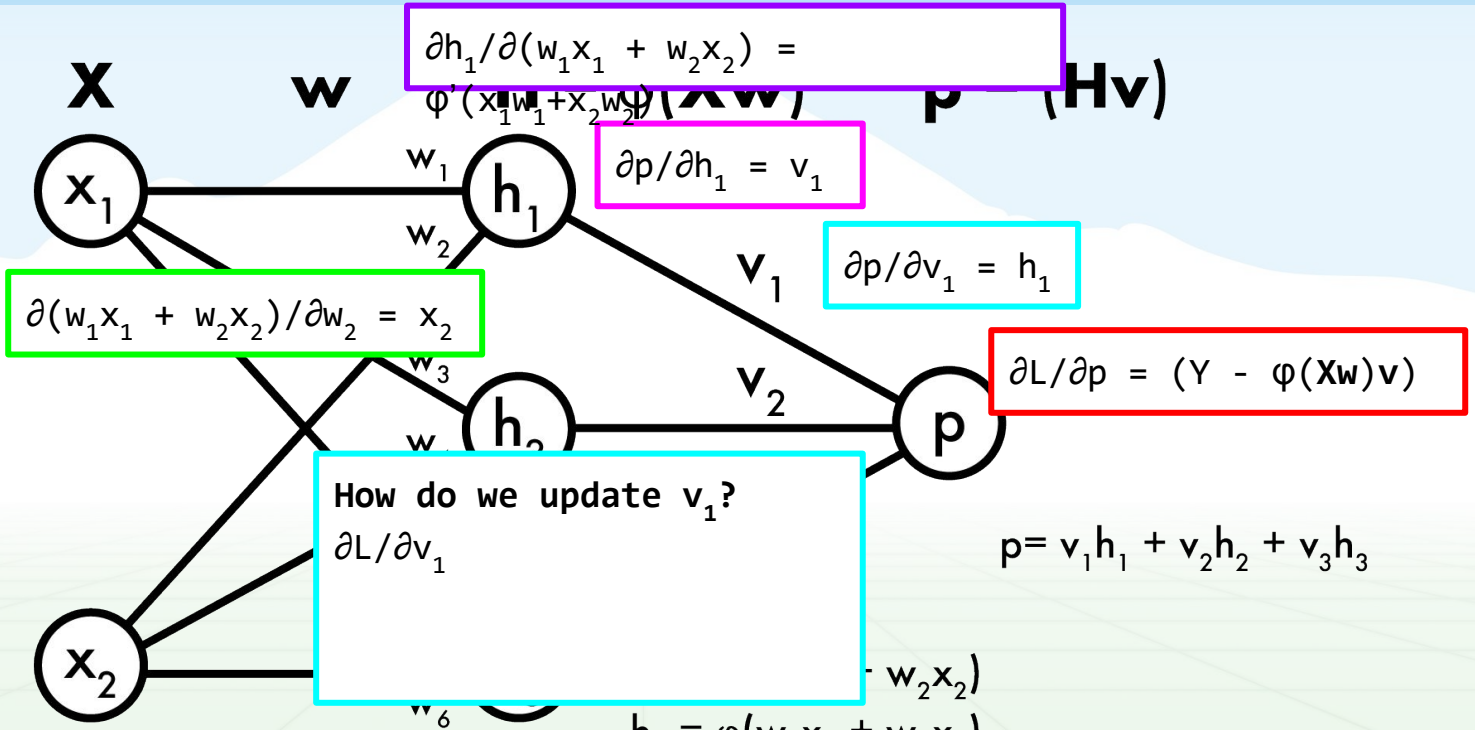
$h_2 = \varphi(w_3x_1 + w_4x_2)$

$h_3 = \varphi(w_5x_1 + w_6x_2)$

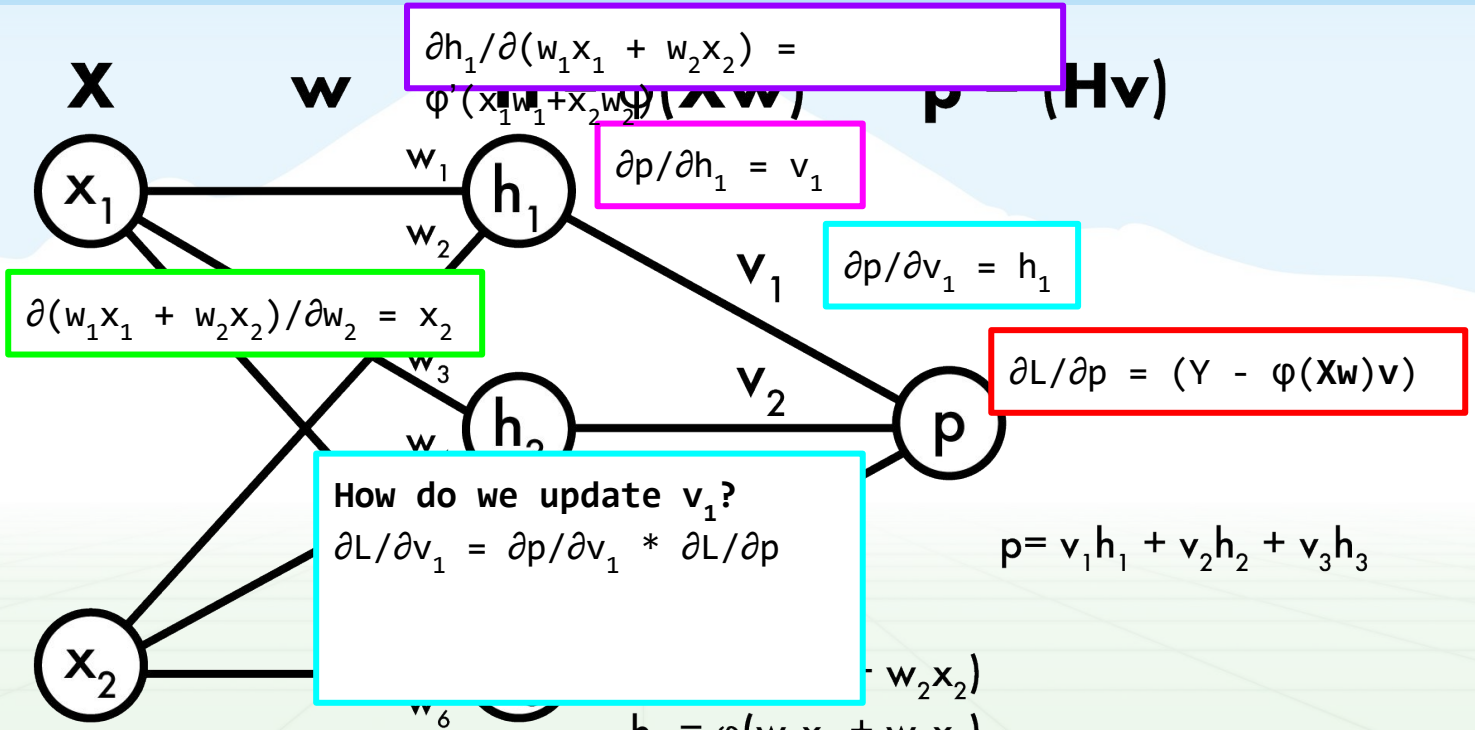
Backpropagation: the math



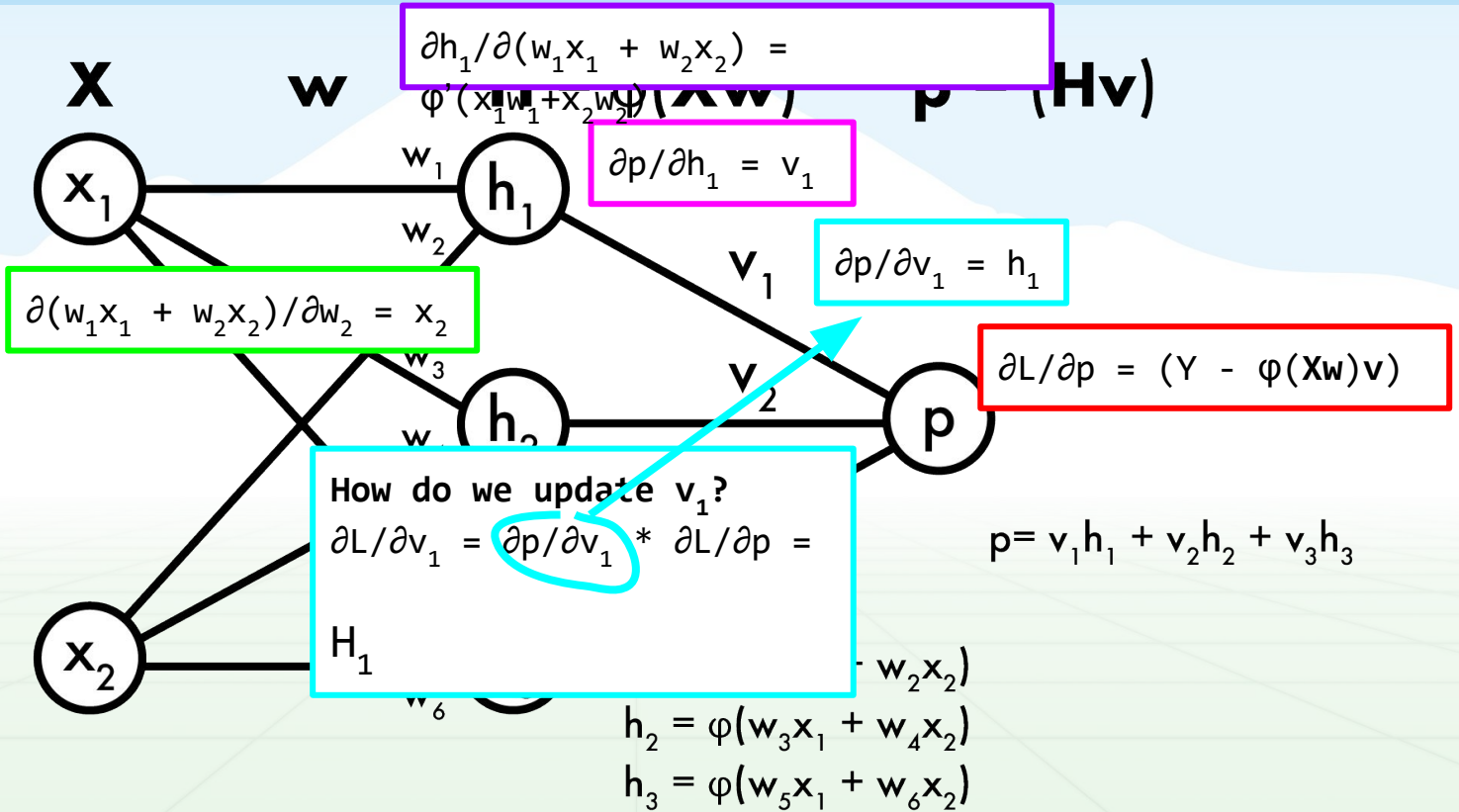
Backpropagation: the math



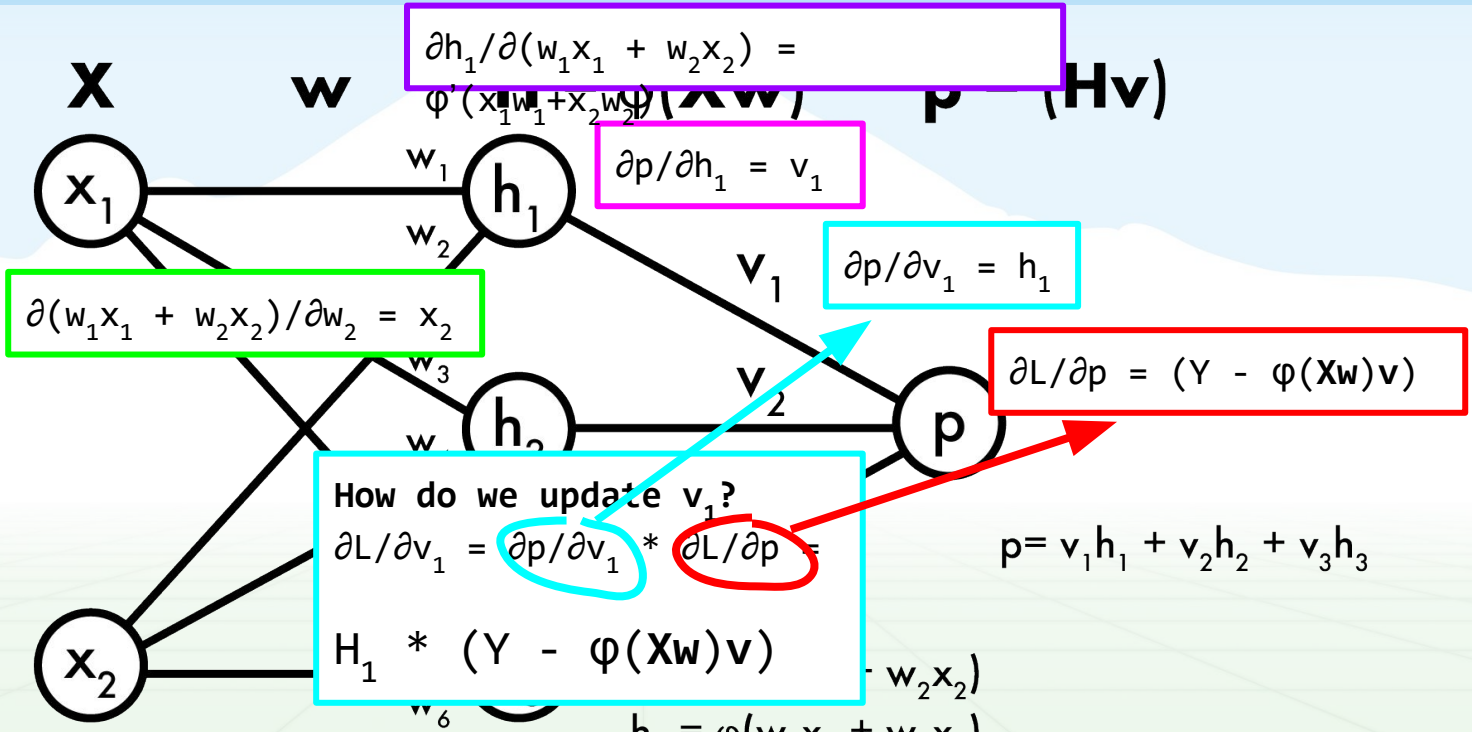
Backpropagation: the math



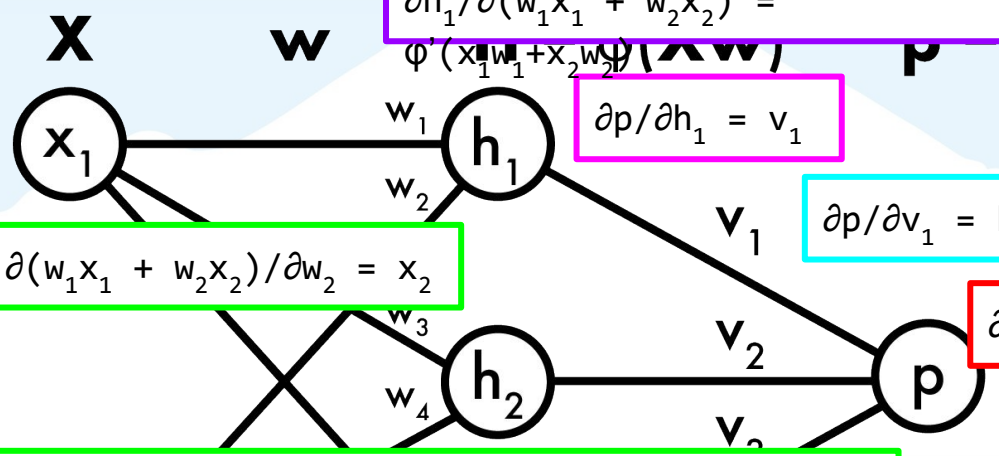
Backpropagation: the math



Backpropagation: the math



Backpropagation: the math



$\partial h_1/\partial(w_1x_1 + w_2x_2) =$
$\varphi'(x_1w_1+x_2w_2)$

$\partial p/\partial h_1 = v_1$

$\partial p/\partial v_1 = h_1$

$\partial(w_1x_1 + w_2x_2)/\partial w_2 = x_2$

$\partial L/\partial p = (Y - \varphi(\mathbf{Xw})\mathbf{v})$

How do we update w_2?

$\partial L/\partial w_2$

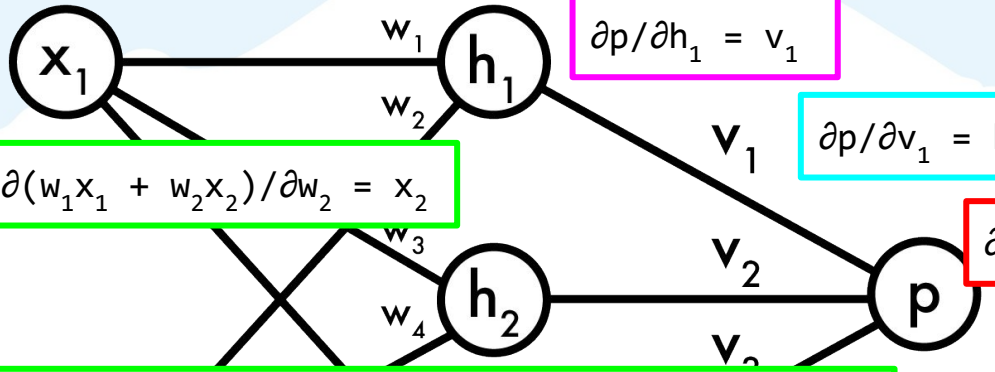
$p = v_1h_1 + v_2h_2 + v_3h_3$

$_2x_2)$

$_4x_2)$

$h_3 = \varphi(w_5x_1 + w_6x_2)$

Backpropagation: the math



$\partial h_1/\partial(w_1x_1 + w_2x_2) = \varphi'(x_1w_1+x_2w_2)$

$\partial p/\partial h_1 = v_1$

$\partial p/\partial v_1 = h_1$

$\partial(w_1x_1 + w_2x_2)/\partial w_2 = x_2$

$\partial L/\partial p = (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})$

How do we update w_2?

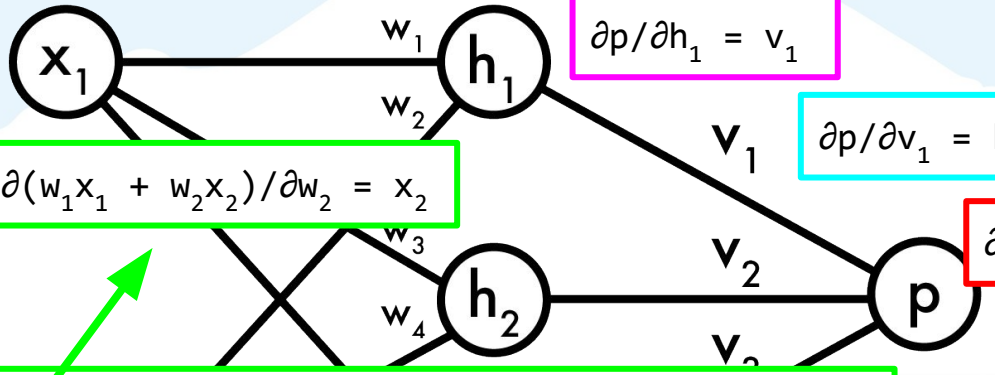
$\partial L/\partial w_2 =$

$\partial(w_1x_1 + w_2x_2)/\partial w_2 * \partial h_1/\partial(w_1x_1 + w_2x_2) * \partial p/\partial h_1 * \partial L/\partial p$

$p = v_1h_1 + v_2h_2 + v_3h_3$

$\varphi(w_5x_1 + w_6x_2)$

Backpropagation: the math



$\partial h_1/\partial(w_1x_1 + w_2x_2) = \varphi'(x_1w_1+x_2w_2)$

$\partial p/\partial h_1 = v_1$

$\partial p/\partial v_1 = h_1$

$\partial(w_1x_1 + w_2x_2)/\partial w_2 = x_2$

$\partial L/\partial p = (Y - \varphi(\mathbf{X}\mathbf{w})\mathbf{v})$

How do we update w_2?

$\partial L/\partial w_2 =$

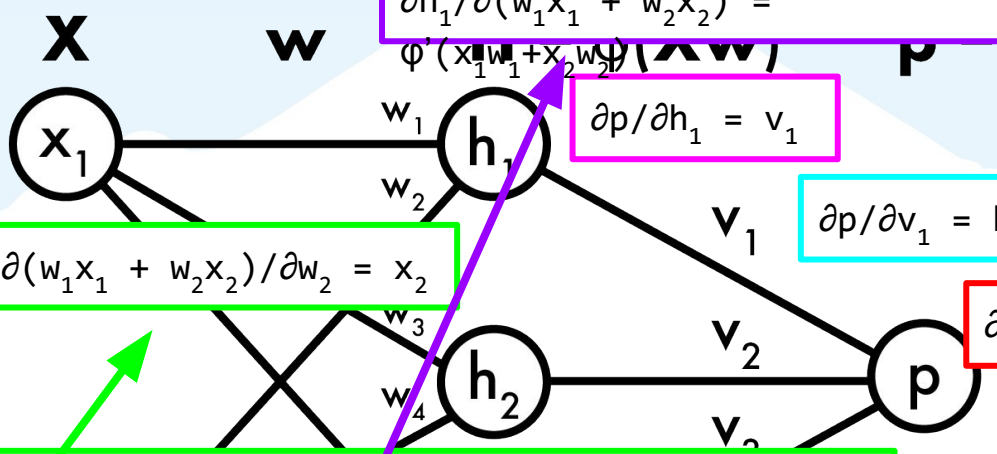
$\partial(w_1x_1 + w_2x_2)/\partial w_2 * \partial h_1/\partial(w_1x_1 + w_2x_2) * \partial p/\partial h_1 * \partial L/\partial p =$

x_2

$p = v_1h_1 + v_2h_2 + v_3h_3$

$\varphi(w_5x_1 + w_6x_2)$

Backpropagation: the math



$\partial h_1/\partial(w_1x_1 + w_2x_2) = \varphi'(x_1w_1+x_2w_2)$

$\partial p/\partial h_1 = v_1$

$\partial p/\partial v_1 = h_1$

$\partial(w_1x_1 + w_2x_2)/\partial w_2 = x_2$

$\partial L/\partial p = (Y - \varphi(\mathbf{Xw})\mathbf{v})$

How do we update w_2?

$\partial L/\partial w_2 =$

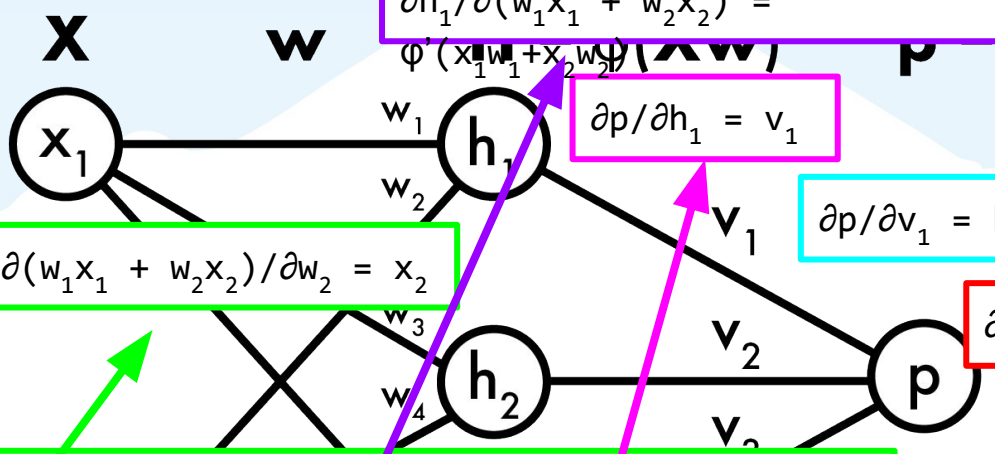
$\partial(w_1x_1 + w_2x_2)/\partial w_2 * \partial h_1/\partial(w_1x_1 + w_2x_2) * \partial p/\partial h_1 * \partial L/\partial p =$

$x_2 * \varphi'(x_1w_1+x_2w_2)$

$p = v_1h_1 + v_2h_2 + v_3h_3$

$\varphi(w_5x_1 + w_6x_2)$

Backpropagation: the math



$\partial h_1/\partial(w_1x_1 + w_2x_2) = \varphi'(x_1w_1+x_2w_2)$

$\partial p/\partial h_1 = v_1$

$\partial p/\partial v_1 = h_1$

$\partial(w_1x_1 + w_2x_2)/\partial w_2 = x_2$

$\partial L/\partial p = (Y - \varphi(\mathbf{Xw})\mathbf{v})$

How do we update w_2?

$\partial L/\partial w_2 =$

$\partial(w_1x_1 + w_2x_2)/\partial w_2$ * $\partial h_1/\partial(w_1x_1 + w_2x_2)$ * $\partial p/\partial h_1$ * $\partial L/\partial p$ =

$x_2 * \varphi'(x_1w_1+x_2w_2) * v_1$

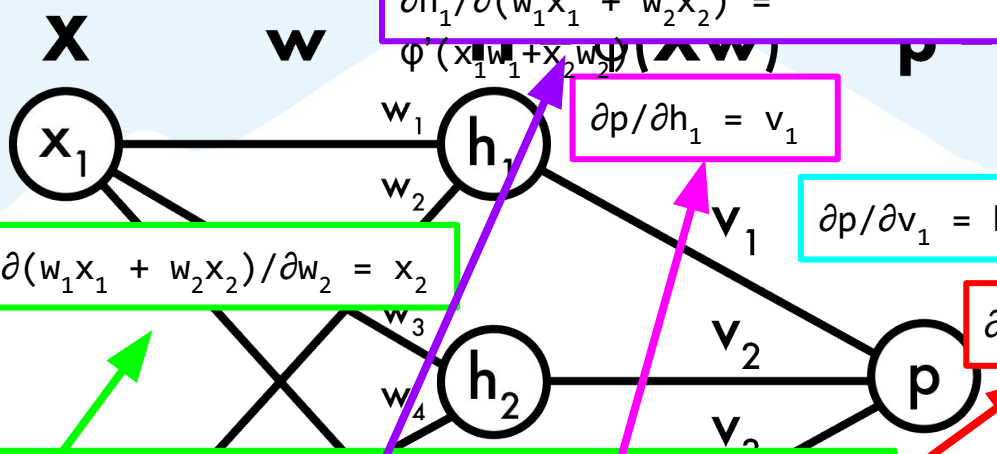
$p = v_1h_1 + v_2h_2 + v_3h_3$

$h_1 = \varphi(w_1x_1 + w_2x_2)$

$h_2 = \varphi(w_3x_1 + w_4x_2)$

$h_3 = \varphi(w_5x_1 + w_6x_2)$

Backpropagation: the math



$\partial h_1/\partial(w_1x_1 + w_2x_2) = \varphi'(x_1w_1+x_2w_2)$

$\partial p/\partial h_1 = v_1$

$\partial p/\partial v_1 = h_1$

$\partial(w_1x_1 + w_2x_2)/\partial w_2 = x_2$

$\partial L/\partial p = (Y - \varphi(\mathbf{Xw})\mathbf{v})$

$p = v_1h_1 + v_2h_2 + v_3h_3$

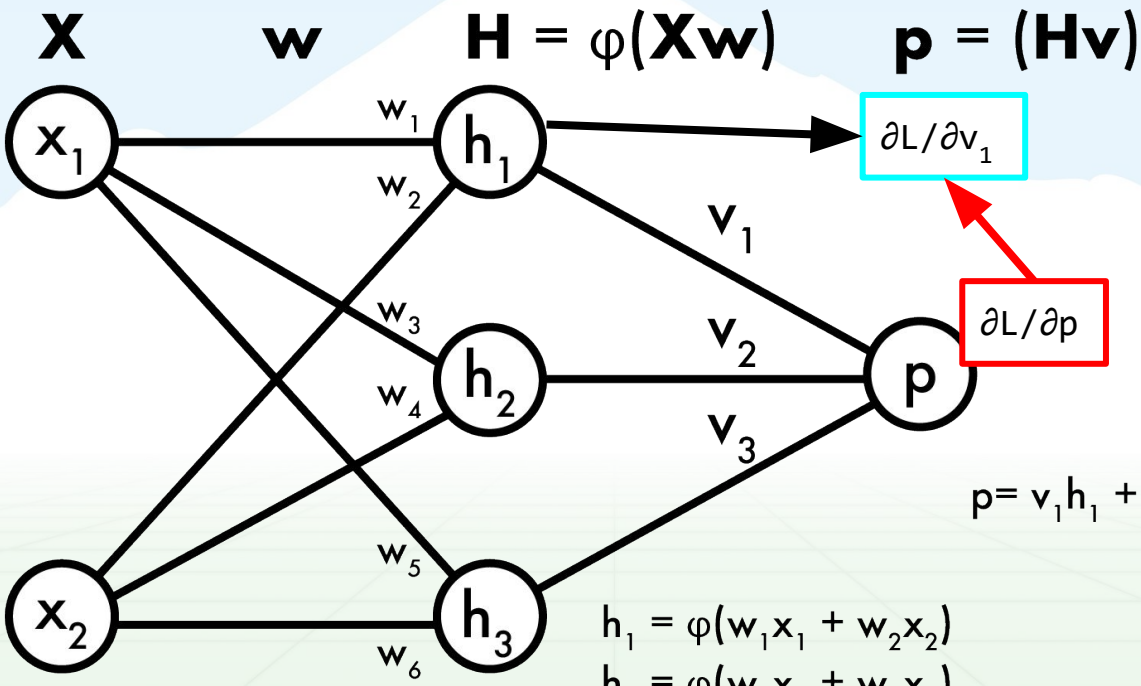
How do we update w_2?

$\partial L/\partial w_2 =$

$\partial(w_1x_1 + w_2x_2)/\partial w_2 * \partial h_1/\partial(w_1x_1 + w_2x_2) * \partial p/\partial h_1 * \partial L/\partial p =$

$x_2 * \varphi'(x_1w_1+x_2w_2) * v_1 * (Y - \varphi(\mathbf{Xw})\mathbf{v})$

Backpropagation: the math



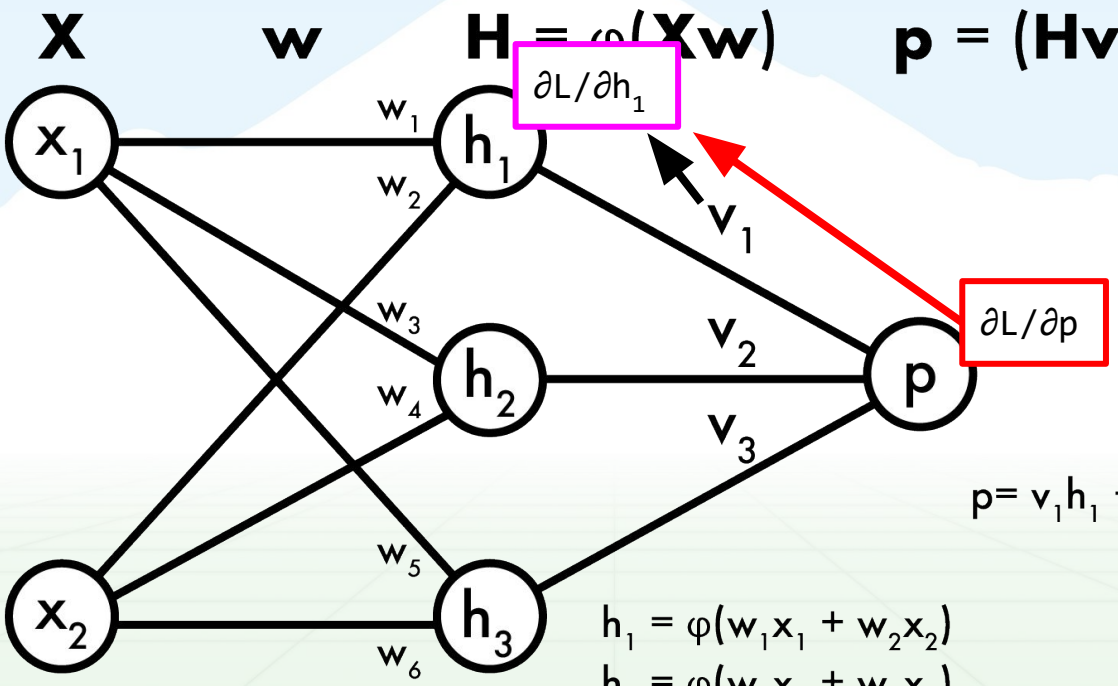
$p = v_1h_1 + v_2h_2 + v_3h_3$

$h_1 = \varphi(w_1x_1 + w_2x_2)$

$h_2 = \varphi(w_3x_1 + w_4x_2)$

$h_3 = \varphi(w_5x_1 + w_6x_2)$

Backpropagation: the math



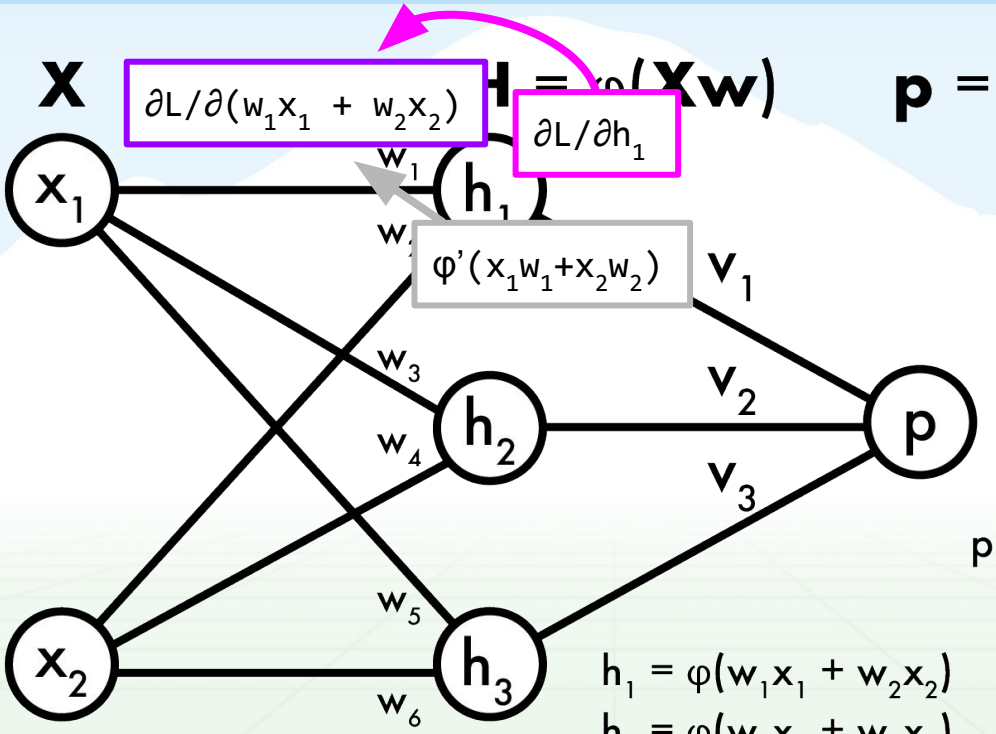
$p = v_1h_1 + v_2h_2 + v_3h_3$

$h_1 = \varphi(w_1x_1 + w_2x_2)$

$h_2 = \varphi(w_3x_1 + w_4x_2)$

$h_3 = \varphi(w_5x_1 + w_6x_2)$

Backpropagation: the math



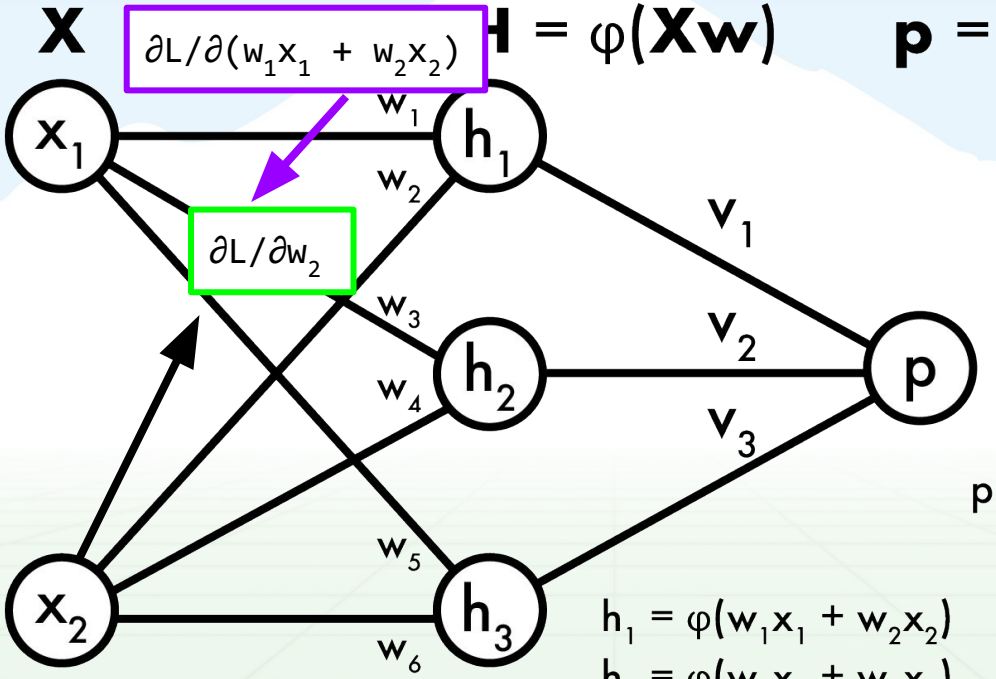
$p = v_1h_1 + v_2h_2 + v_3h_3$

$h_1 = \varphi(w_1x_1 + w_2x_2)$

$h_2 = \varphi(w_3x_1 + w_4x_2)$

$h_3 = \varphi(w_5x_1 + w_6x_2)$

Backpropagation: the math



$p = v_1h_1 + v_2h_2 + v_3h_3$

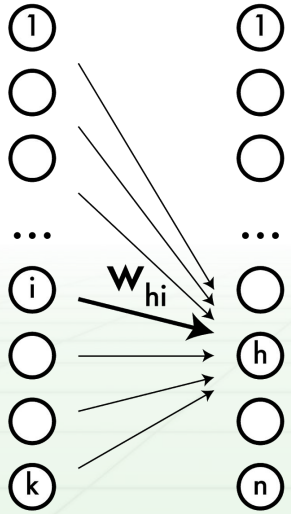
$h_1 = \varphi(w_1x_1 + w_2x_2)$

$h_2 = \varphi(w_3x_1 + w_4x_2)$

$h_3 = \varphi(w_5x_1 + w_6x_2)$

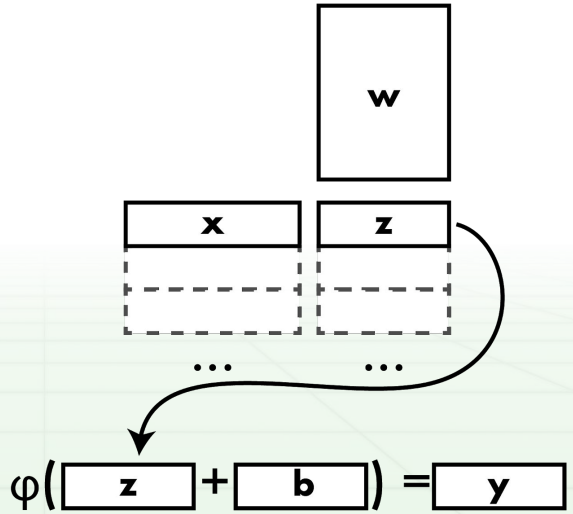
Forward propagation

$$y_h = \varphi(\sum x_i w_{hi} + b)$$



Layer x Layer y

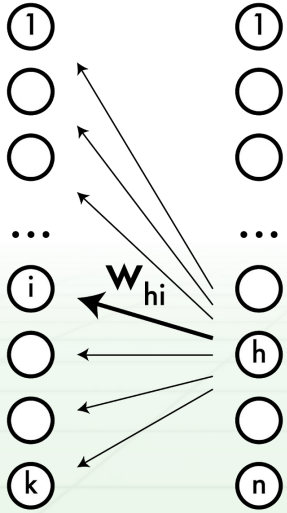
$$\mathbf{y} = \varphi(\mathbf{xw} + \mathbf{b})$$

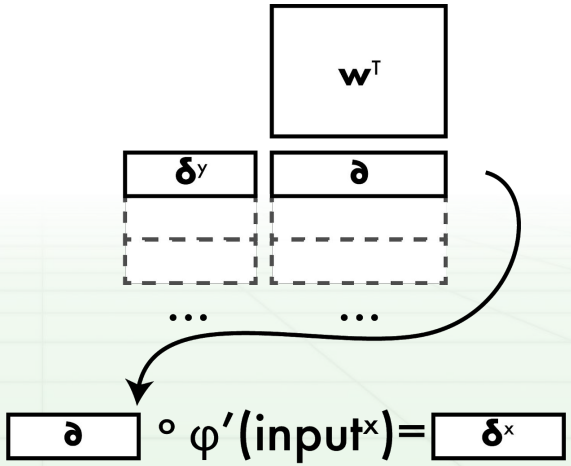


x: m×k **w**: k×n **z**: m×n

Backward propagation

$$\delta_i^x = \left(\sum \delta_j^y w_{hi}\right) \varphi'(input_i^x) \qquad \boldsymbol{\delta}^x = \boldsymbol{\delta}^y \mathbf{w}^T \circ \varphi'(input^x)$$



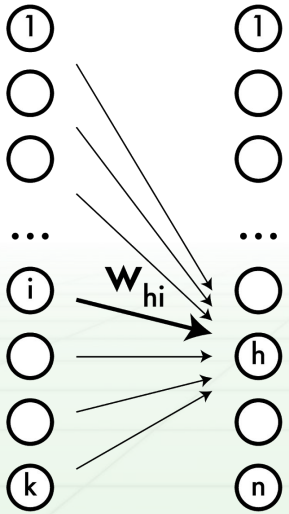


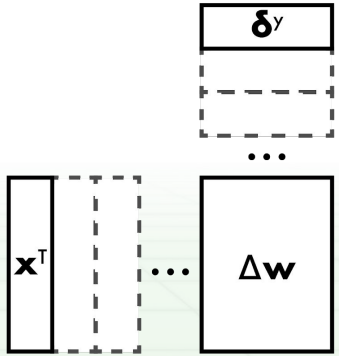
Layer x Layer y $\boldsymbol{\delta}^y$: m×n $\mathbf{w}^T$: n×k $\boldsymbol{\delta}^x$: m×k

Weight updates

$$\Delta w_{hi} = \delta_h^y x_i$$

$$\Delta \mathbf{w} = \mathbf{x}^T \boldsymbol{\delta}^y$$





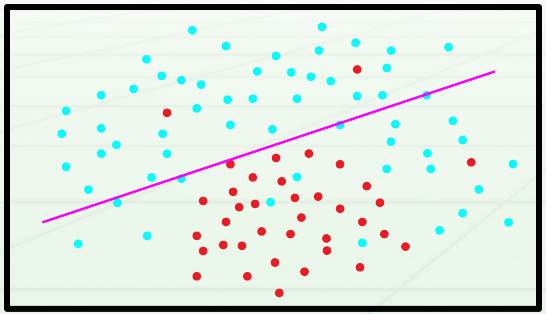
Layer x Layer y $\mathbf{x}^T$: k×m $\boldsymbol{\delta}^y$: m×n $\Delta \mathbf{w}$: k×n

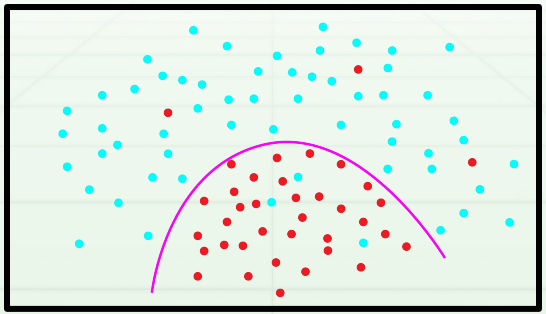
Under and Overfitting

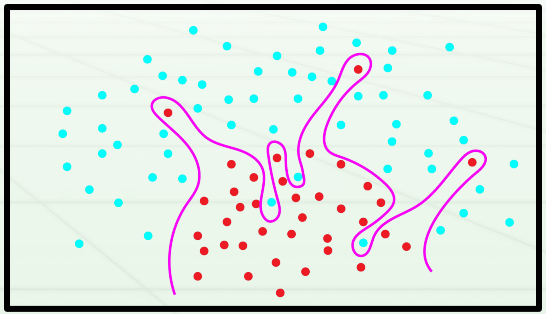
Underfitting: model not powerful enough, too much bias

Overfitting: model too powerful, fits to noise, doesn’t generalize well

Want the happy medium, how?







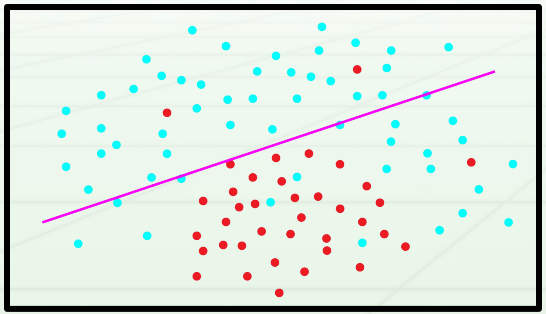
Under and Overfitting

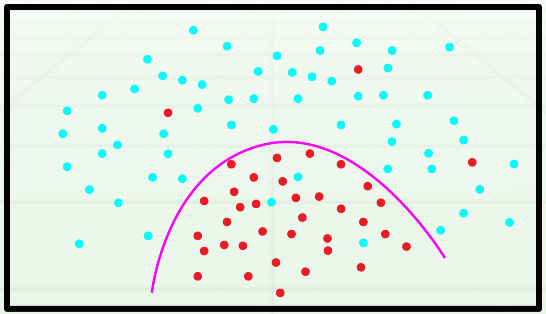
Want the happy medium, how?

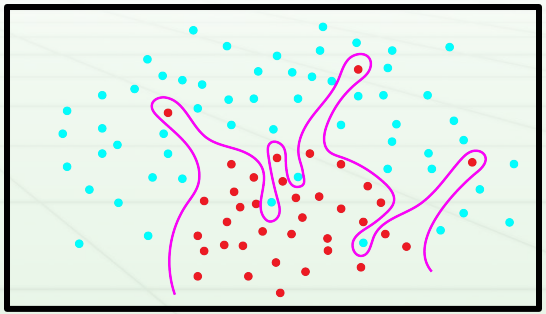
Pick the right model, but very hard to know a priori

Make weak model more powerful: boosting! (or other ways)

Make strong model less likely to overfit: *regularization*







With great power comes great *overfitting*

Neural networks are (sort of) all powerful! Which is not necessarily a good thing.

With great power comes great *overfitting*

Like SVMs, put limits on model that make it generalize better!

SVM:

$\min ||\mathbf{w}||_2$

$\text{s.t. } y_n(\mathbf{w} \cdot \mathbf{x}_n - b) \geq 1, \; n = 1, 2 \ldots$

Neural net:

Minimize loss function and weight magnitude

Before: $\text{argmin}_w \, L_X(\mathbf{w})$

Now: $\text{argmin}_w \, L_X(\mathbf{w}) + \lambda \, ||\mathbf{w}||_2$

Weight decay: neural network regularization

$\text{argmin}_w L_x(\mathbf{w}) + \lambda ||\mathbf{w}||_2$

λ: regularization parameter

Higher: more penalty for large weights, less powerful model

Lower: less penalty, more overfitting

Commonly use L_2 norm to regularize, *weight decay*

Gradient descent update rule:

$$w_{t+1} = w_t - \eta[\partial/\partial w_t \, L(w_t) + \lambda w_t]$$

$$= w_t - \eta\partial/\partial w_t \, L(w_t) - \eta\lambda w_t$$

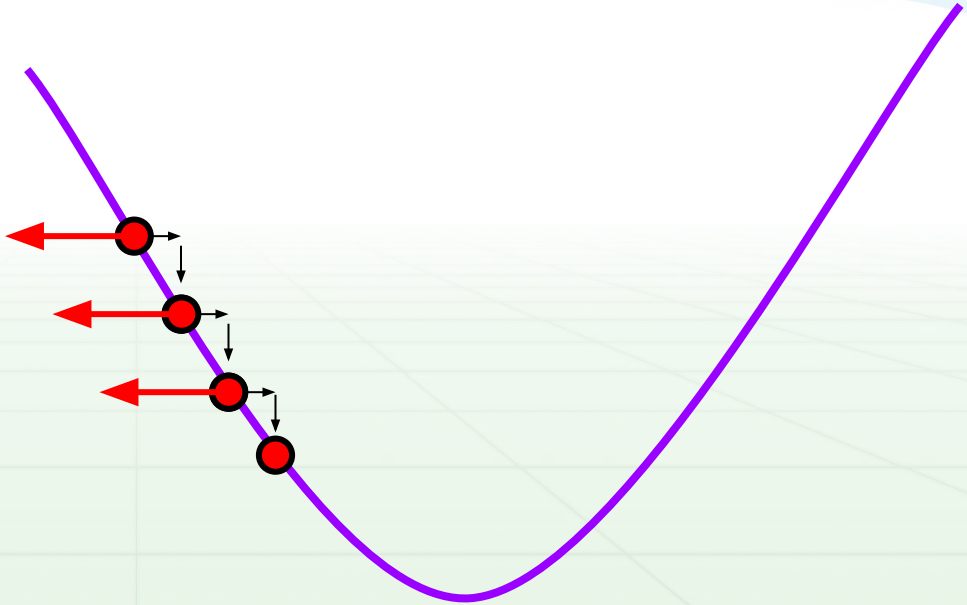
Subtract a little bit of weight every iteration

Sometimes training is SLOW

With SGD we make LOTS of little steps along the gradient

Sometimes we move in the same direction for a long time...

Maybe we should speed up in that direction!



Momentum: speeding up SGD

If we keep moving in same direction we should move further every round

Before:

$$\Delta w_t = -\partial/\partial w_t \; L(w_t)$$

Now:

$$\Delta w_t = -\partial/\partial w_t \; L(w_t) + m\Delta w_{t-1}$$

$$w_{t+1} = w_t + \eta \Delta w_t$$

Side effect: **smooths** out updates if gradient is in different directions

NN updates with weight decay and momentum

$$\Delta w_t = -\partial/\partial w_t \, L(w_t) - \lambda w_t + m\Delta w_{t-1}$$

$$w_{t+1} = w_t + \eta \Delta w_t$$

NN updates with weight decay and momentum

$$\Delta w_t = -\partial/\partial w_t \; L(w_t) - \lambda w_t + m\Delta w_{t-1}$$

Gradient of loss

$$w_{t+1} = w_t + \eta \Delta w_t$$

NN updates with weight decay and momentum

$$\Delta w_t = -\partial/\partial w_t \; L(w_t) - \lambda w_t + m\Delta w_{t-1}$$

Gradient of loss

Weight decay

$$w_{t+1} = w_t + \eta \Delta w_t$$

NN updates with weight decay and momentum

$$\Delta w_t = -\partial/\partial w_t \; L(w_t) - \lambda w_t + m\Delta w_{t-1}$$

Gradient of loss

Weight decay

Momentum

$$w_{t+1} = w_t + \eta \Delta w_t$$

NN updates with weight decay and momentum

$$\Delta w_t = -\partial/\partial w_t \; L(w_t) - \lambda w_t + m\Delta w_{t-1}$$

Gradient of loss

Weight decay

Momentum

$$w_{t+1} = w_t + \eta \Delta w_t$$

Learning rate

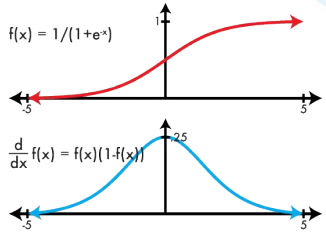
What about our activation functions φ

Many options, want them to be easy to take derivative

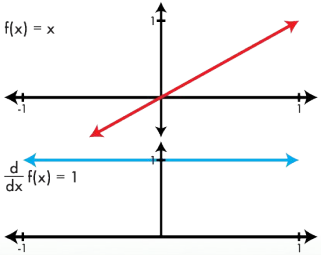
UAT holds when bounded, in practice bounds can be problematic

Common activation functions φ

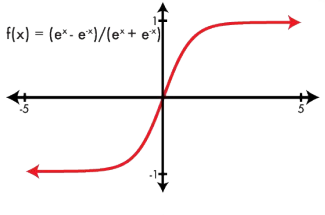
logistic



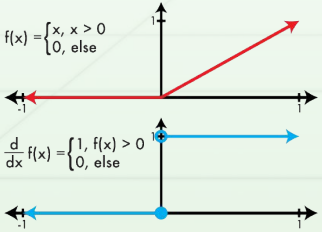
linear



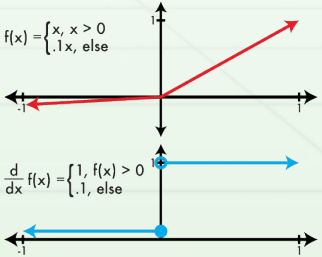
tanh



REctified Linear Unit (RELU)



Leaky RELU



So many hyper parameters!!

How do we know what to use??

Hyper Parameter Dark Magic

What follows are the one, true, correct, and only set of hyperparameters. Praise be the NetLord!

$\eta = [.0001 - .01]$

$\lambda = .0005$

$m = .9$

$\varphi =$ leaky relu