Back-Propagation

Lecture 0 2

Yann Le Cun

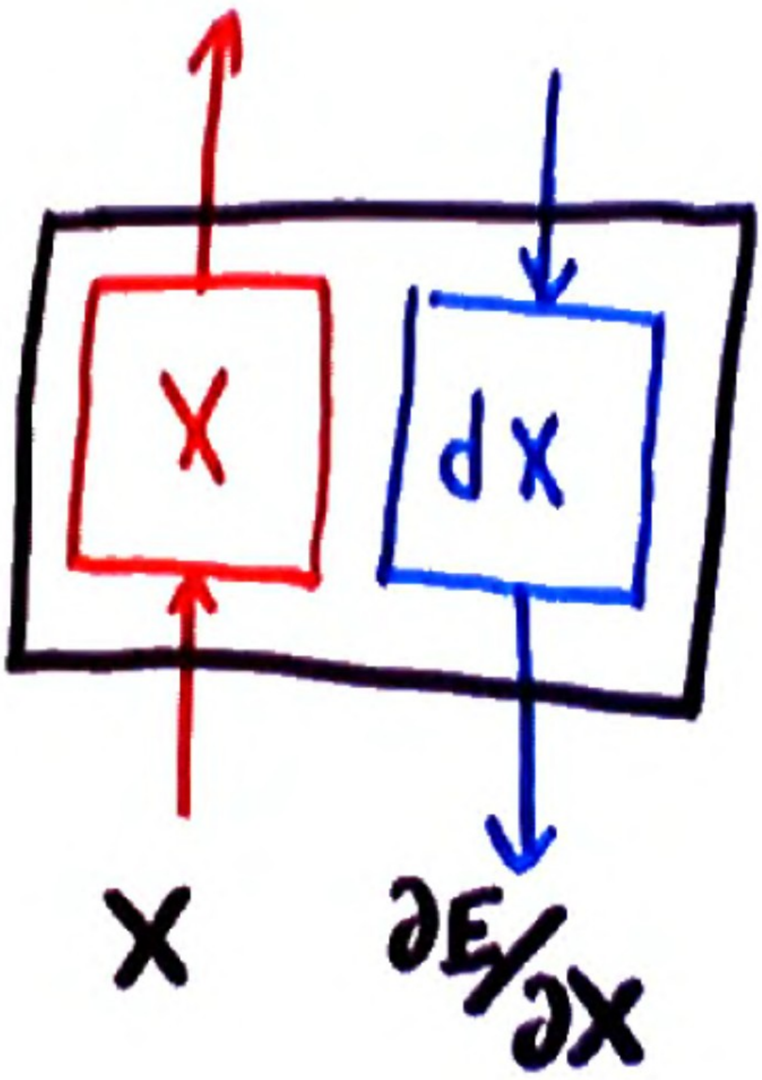
Facebook AI Research,

Center for Data Science, NYU

Courant Institute of Mathematical Sciences, NYU

http://yann.lecun.com

State Variables

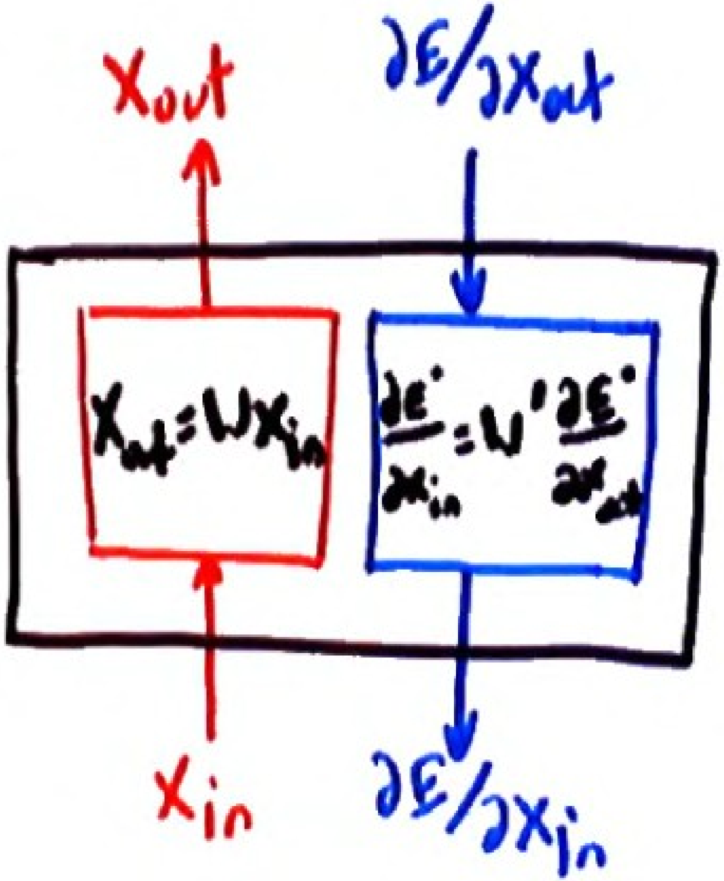


the internal state of the network will be kept in a "state" class that contains two scalars, vectors, or matrices: (1) the state proper, (2) the derivative of the energy with respect to that state.

Linear Module

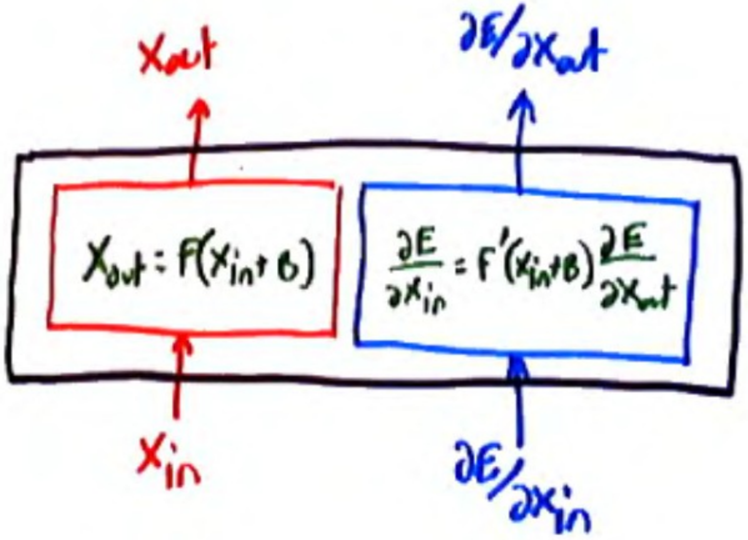
Y LeCun
MA Ranzato

The input vector is multiplied by the weight matrix.



- fprop: $X_{\text{out}} = W X_{\text{in}}$
- bprop to input:
  $$\frac{\partial E}{\partial X_{\text{in}}} = \frac{\partial E}{\partial X_{\text{out}}} \frac{\partial X_{\text{out}}}{\partial X_{\text{in}}} = \frac{\partial E}{\partial X_{\text{out}}} W$$
- by transposing, we get column vectors:
  $$\frac{\partial E}{\partial X_{\text{in}}}' = W' \frac{\partial E}{\partial X_{\text{out}}}'$$
- bprop to weights:
  $$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial X_{\text{out}i}} \frac{\partial X_{\text{out}i}}{\partial W_{ij}} = X_{\text{in}j} \frac{\partial E}{\partial X_{\text{out}i}}$$
- We can write this as an outer-product:
  $$\frac{\partial E}{\partial W}' = \frac{\partial E}{\partial X_{\text{out}}}' X'_{in}$$

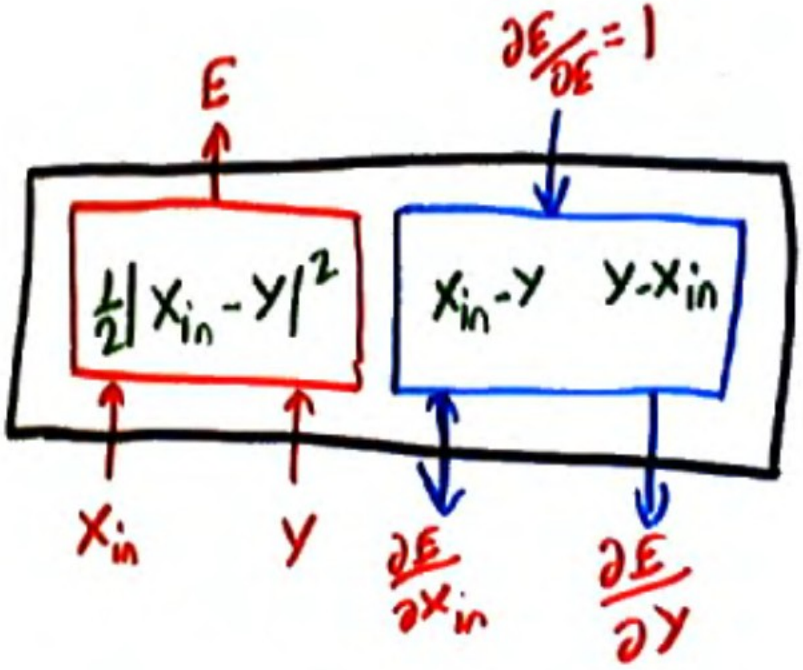
Tanh module (or any other pointwise function)



- fprop: $(X_{\text{out}})_i = \tanh((X_{\text{in}})_i + B_i)$
- bprop to input:
  $(\frac{\partial E}{\partial X_{\text{in}}})_i = (\frac{\partial E}{\partial X_{\text{out}}})_i \tanh'((X_{\text{in}})_i + B_i)$
- bprop to bias:
  $\frac{\partial E}{\partial B_i} = (\frac{\partial E}{\partial X_{\text{out}}})_i \tanh'((X_{\text{in}})_i + B_i)$
- $\tanh(x) = \frac{2}{1+\exp -x} - 1 = \frac{1-\exp(-x)}{1+\exp(-x)}$

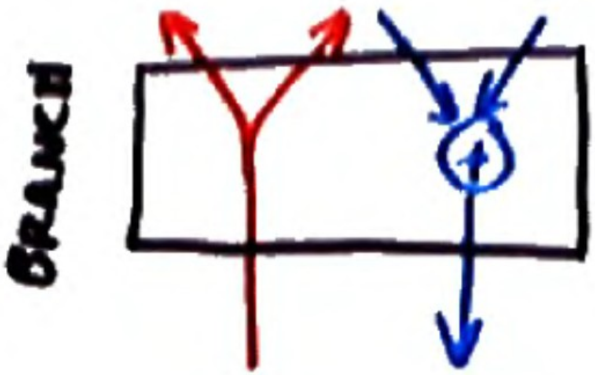
Euclidean Distance Module (Squared Error)

Y LeCun
MA Ranzato



- fprop: $X_{\text{out}} = \frac{1}{2}||X_{\text{in}} - Y||^2$
- bprop to X input: $\frac{\partial E}{\partial X_{\text{in}}} = X_{\text{in}} - Y$
- bprop to Y input: $\frac{\partial E}{\partial Y} = Y - X_{\text{in}}$

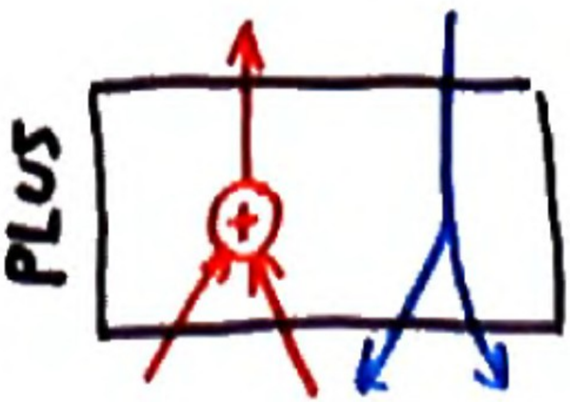
Y connector and Addition modules



- The PLUS module: a module with K inputs $X_1, \ldots, X_K$ (of any type) that computes the sum of its inputs:

$$X_{\text{out}} = \sum_k X_k$$

back-prop: $\frac{\partial E}{\partial X_k} = \frac{\partial E}{\partial X_{\text{out}}} \quad \forall k$

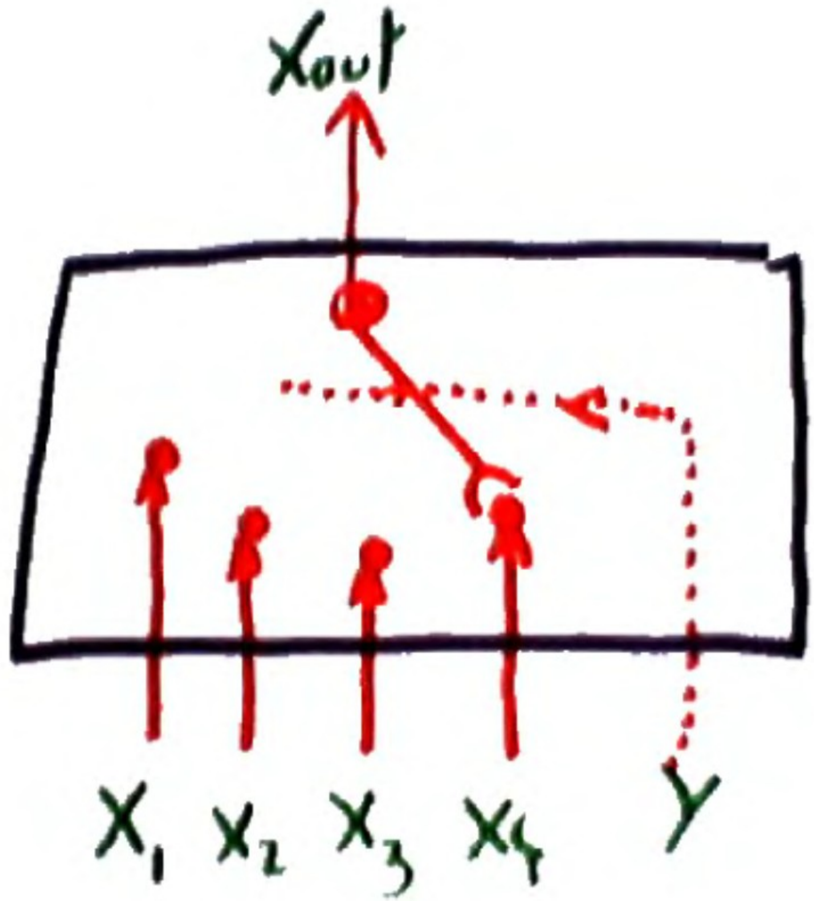


- The BRANCH module: a module with one input and K outputs $X_1, \ldots, X_K$ (of any type) that simply copies its input on its outputs:

$$X_k = X_{\text{in}} \quad \forall k \in [1..K]$$

back-prop: $\frac{\partial E}{\partial \text{in}} = \sum_k \frac{\partial E}{\partial X_k}$

Switch Module



- A module with K inputs $X_1, \ldots, X_K$ (of any type) and one additional discrete-valued input Y.
- The value of the discrete input determines which of the N inputs is copied to the output.
-

$$X_{\text{out}} = \sum_k \delta(Y-k) X_k$$

$$\frac{\partial E}{\partial X_k} = \delta(Y-k) \frac{\partial E}{\partial X_{\text{out}}}$$

the gradient with respect to the output is copied to the gradient with respect to the switched-in input. The gradients of all other inputs are zero.

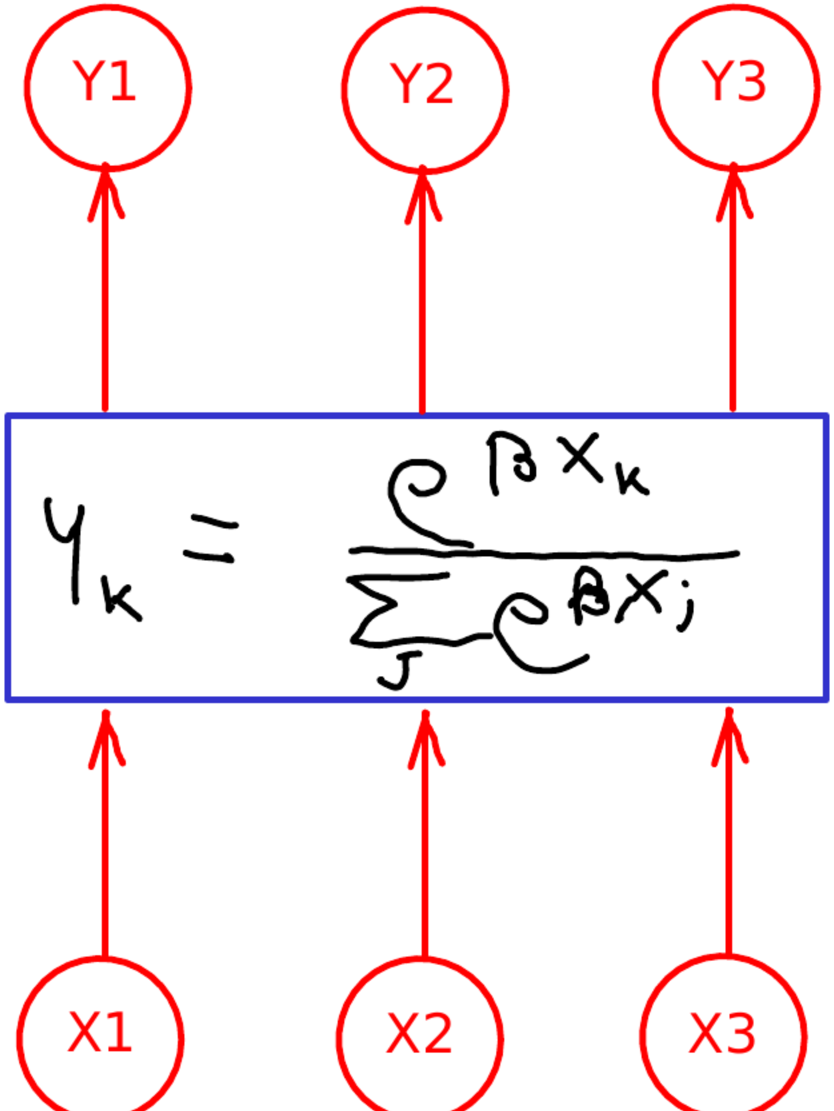
SoftMax Module (should really be called SoftArgMax)

Y LeCun
MA Ranzato

- Transforms scores into a discrete probability distribution
  - Positive numbers that sum to one.

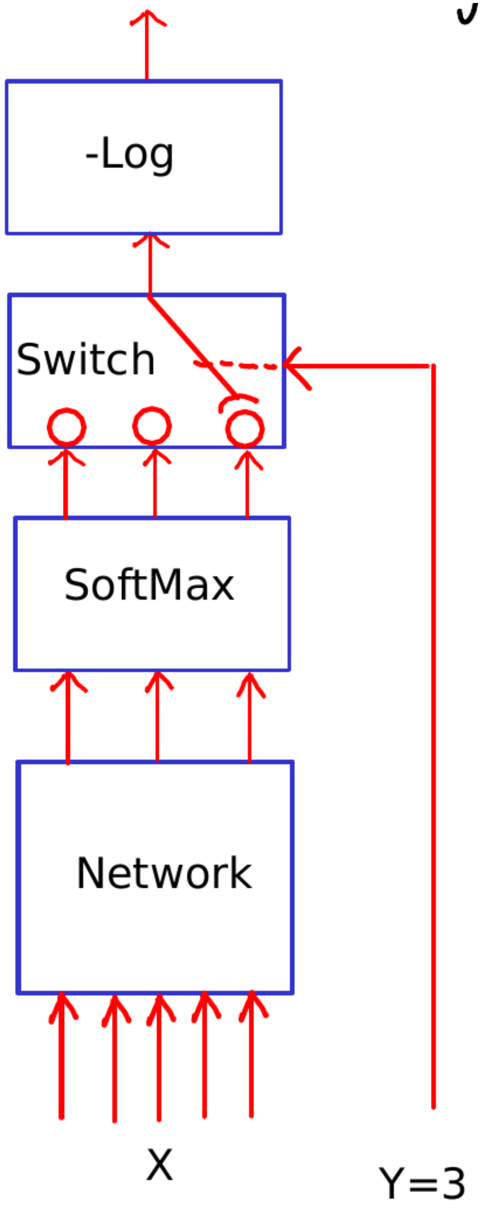
- Used in multi-class classification

$$P_k = \frac{e^{\beta x_k}}{\sum_j e^{\beta x_j}}$$

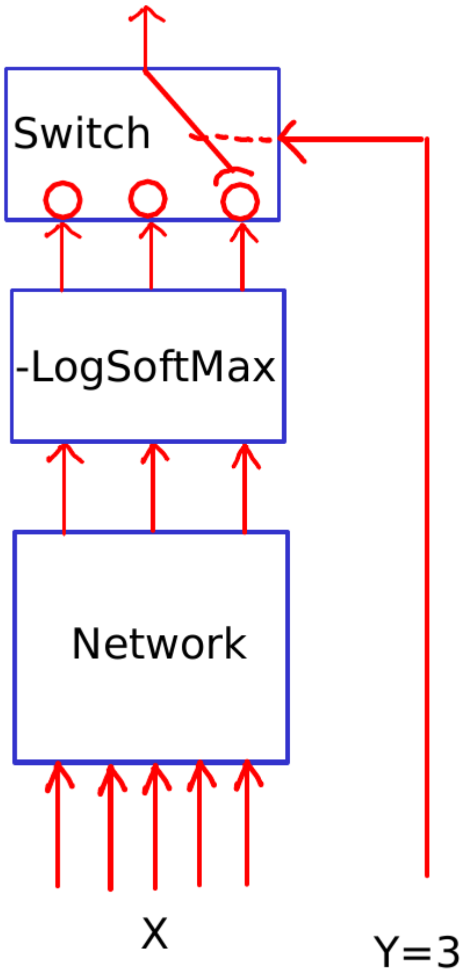


SoftMax Module: Loss Function for Classification

Y LeCun
MA Ranzato

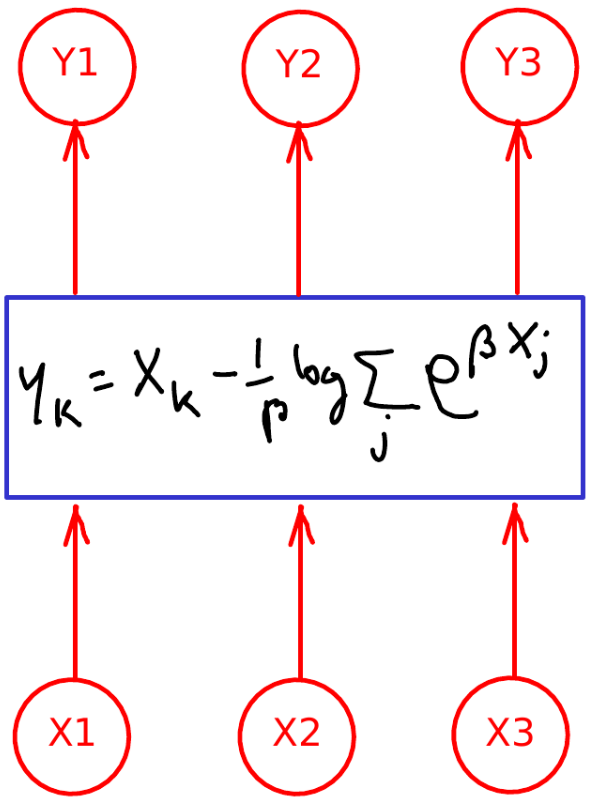


- -LogSoftMax: $-\frac{1}{\beta}\log P_k = -X_k + \frac{1}{\beta}\log \sum_j e^{\beta X_j}$
- Maximum conditional likelihood
- Minimize -log of the probability of the correct class.

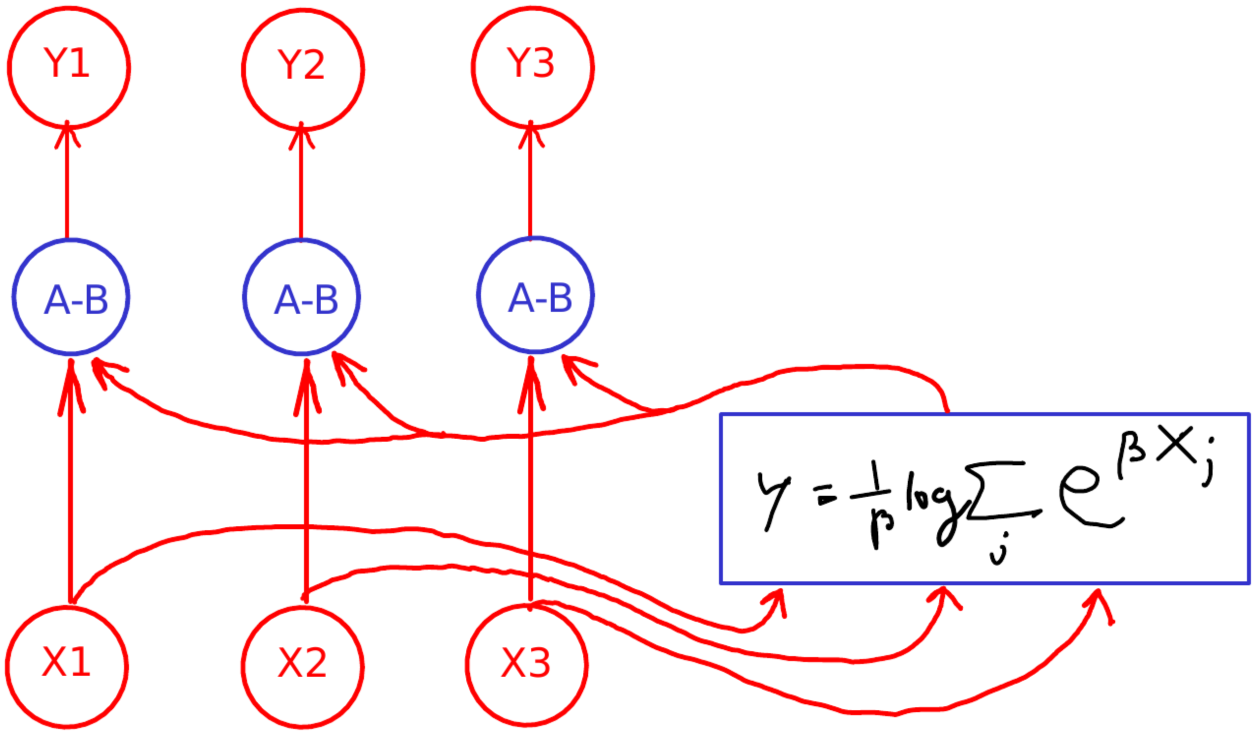


LogSoftMax Module

Y LeCun
MA Ranzato



- Transforms scores into a discrete probability distribution
- LogSoftMax = Identity - LogSumExp



LogSumExp Module

- Log of normalization term for SoftMax
- Fprop

$$X_{out} = \frac{1}{\beta} \sum_j e^{\beta X_j}$$

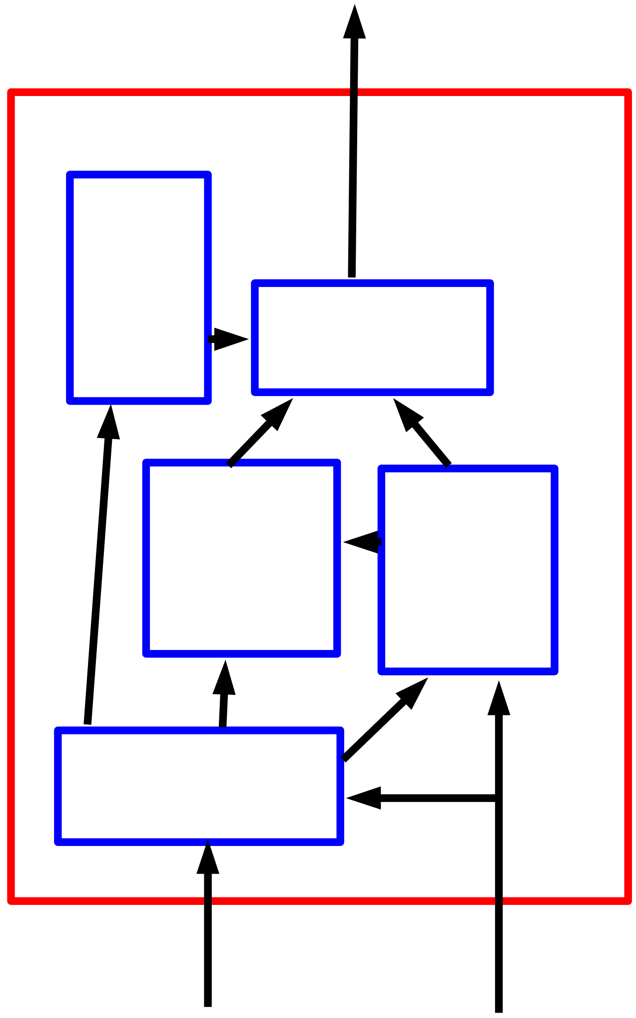
- Bprop

$$\frac{\partial E}{\partial X_k} = \frac{\partial E}{\partial X_{out}} \cdot \frac{e^{\beta X_k}}{\sum_j e^{\beta X_j}}$$

- Or:

$$\frac{\partial E}{\partial X_k} = \frac{\partial E}{\partial X_{out}} \cdot P_k \qquad P_k = \frac{e^{\beta X_k}}{\sum_j e^{\beta X_j}}$$

Backprop works through any modular architecture



- **Any connection is permissible**
  - Networks with loops must be "unfolded in time".
- **Any module is permissible**
  - As long as it is continuous and differentiable almost everywhere with respect to the parameters, and with respect to non-terminal inputs.

Backprop in Practice

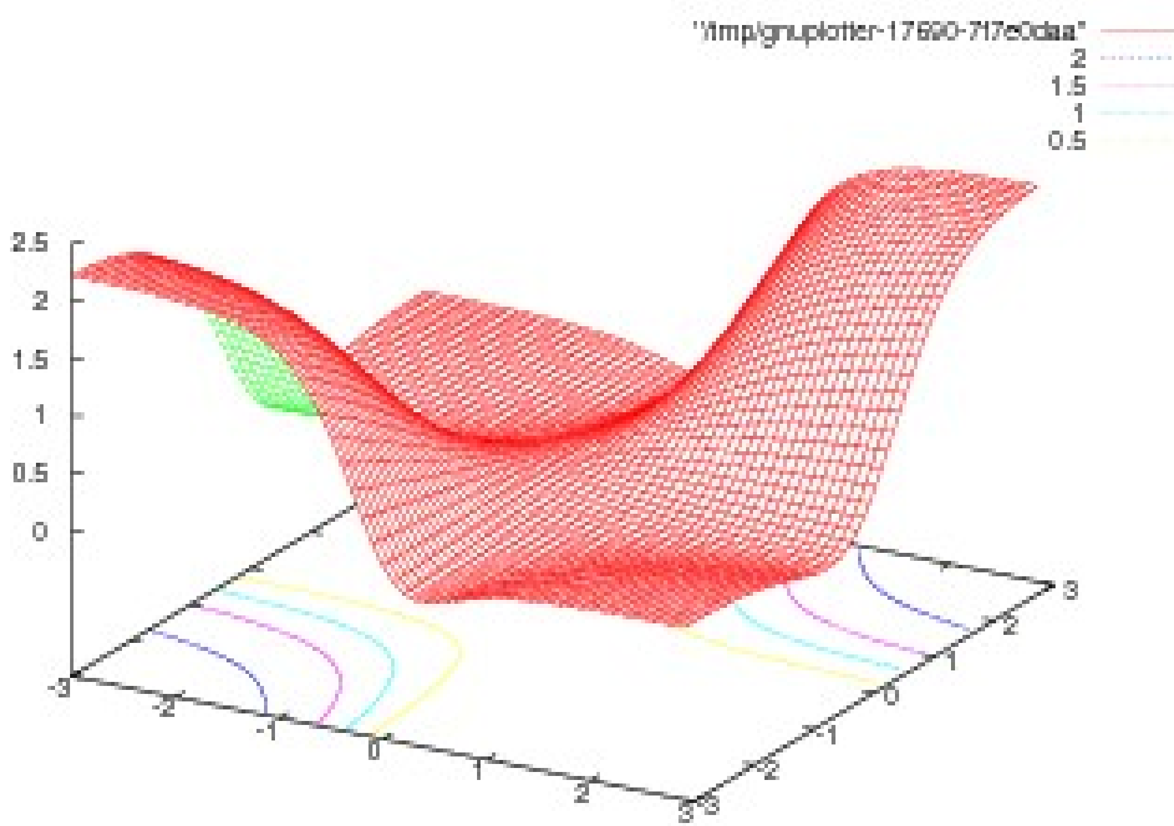
Y LeCun
MA Ranzato

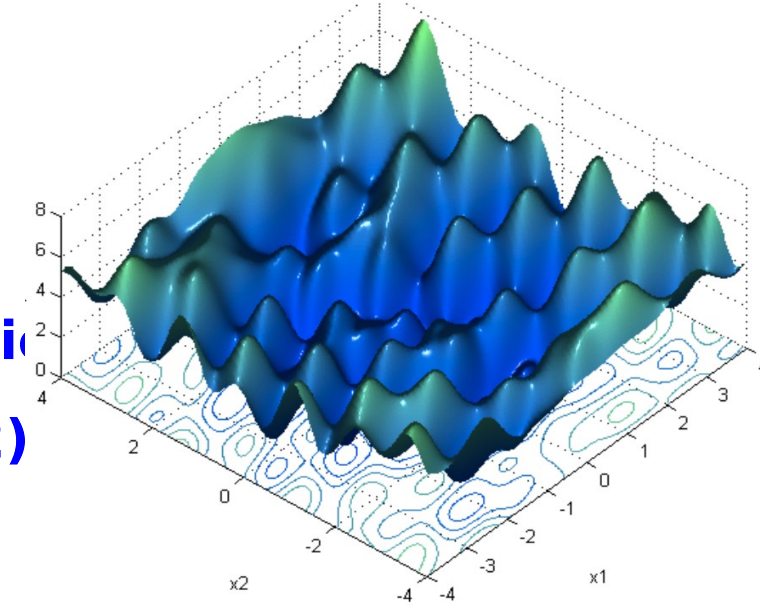
- **Use ReLU non-linearities (tanh and logistic are falling out of favor)**
- **Use cross-entropy loss for classification**
- **Use Stochastic Gradient Descent on minibatches**
- **Shuffle the training samples**
- **Normalize the input variables (zero mean, unit variance)**
- **Schedule to decrease the learning rate**
- **Use a bit of L1 or L2 regularization on the weights (or a combination)**
  - But it's best to turn it on after a couple of epochs
- **Use “dropout” for regularization**
  - Hinton et al 2012 http://arxiv.org/abs/1207.0580
- **Lots more in [LeCun et al. “Efficient Backprop” 1998]**
- **Lots, lots more in “Neural Networks, Tricks of the Trade” (2012 edition) edited by G. Montavon, G. B. Orr, and K-R Müller (Springer)**

Deep Learning is Non-Convex

- **Example: what is the loss function for the simplest 2-layer neural net ever**
  - Function: 1-1-1 neural net. Map 0.5 to 0.5 and -0.5 to -0.5 (identity function) with quadratic cost:

$$y = \tanh(W_1 \tanh(W_0.x)) \qquad L = (0.5 - \tanh(W_1 \tanh(W_0 0.5)^2$$





1-1-1 network

– Y = W1*W2*X

Objective: identity function with quadrati

One sample: X=1, Y=1 L(W) = (1-W1*W2)