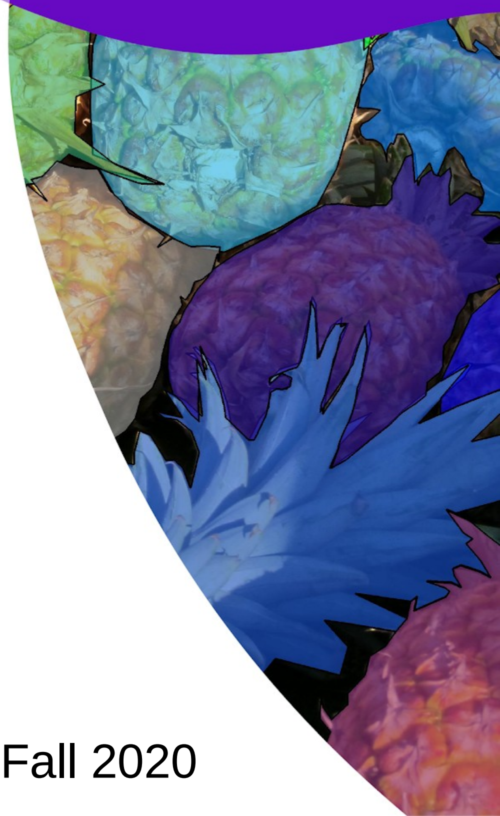
Representations, Deep Architectures & Backpropagation

Yann LeCun
NYU - Courant Institute & Center for Data Science
Facebook AI Research

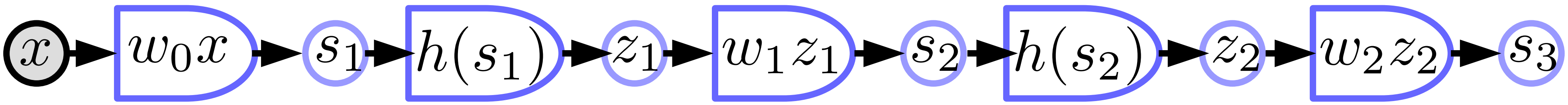


Deep Learning, NYU Fall 2020

Block Diagram of a Traditional Neural Net

- **linear blocks** $$s_{k+1} = w_k z_k$$

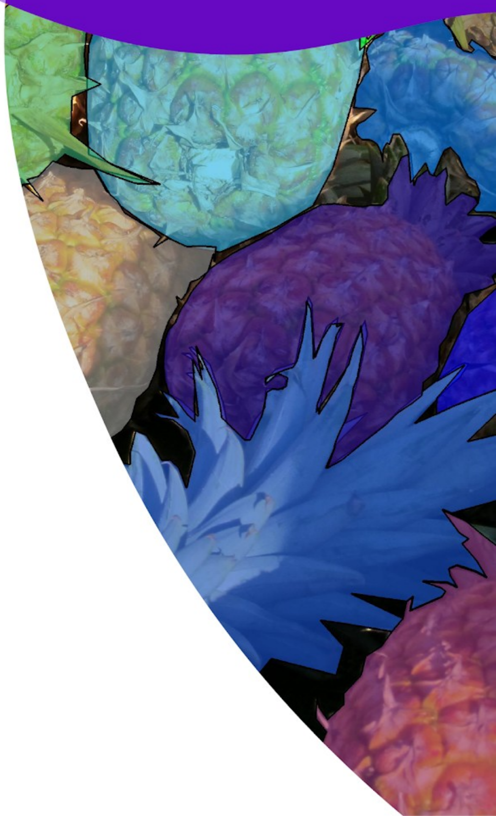
- **Non-linear blocks** $$z_k = h(s_k)$$



NEW YORK UNIVERSITY

What are Good Representations?

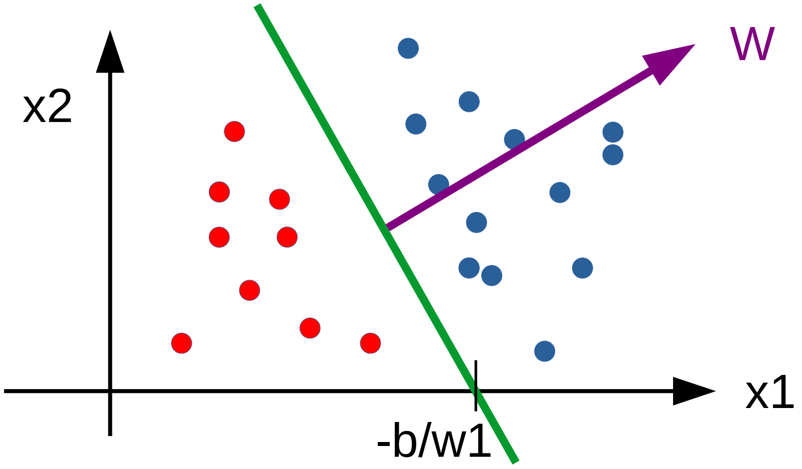
What are good features?



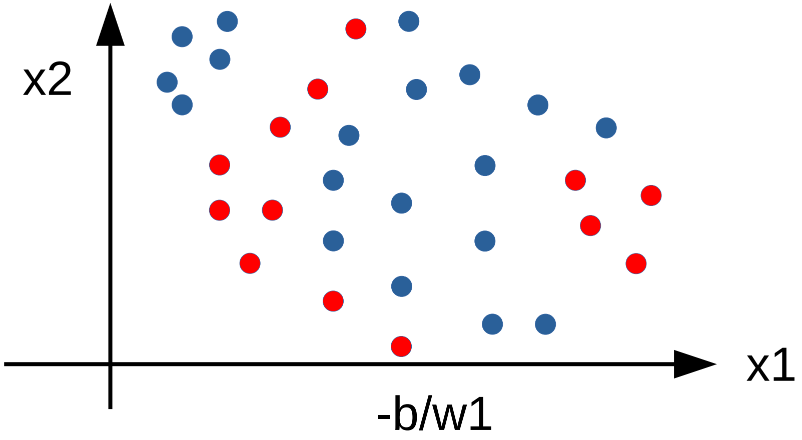
Linear Classifiers and their limitations

- **Linear classifier** $$\bar{y} = sign\left(\sum_{i=1}^{N} w_i x_i + b\right)$$
  - Partitions the space into two half spaces separated by the hyperplane:

$$\sum_{i=1}^{N} w_i x_i + b = 0$$

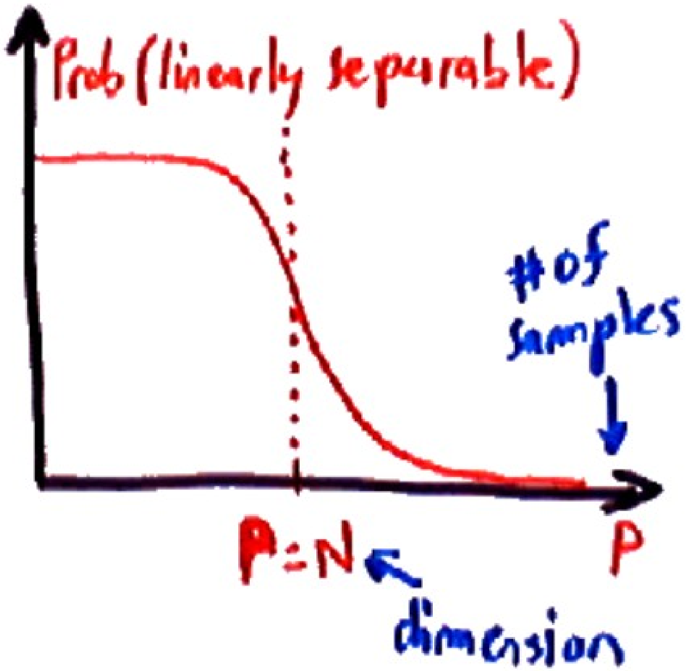


Not linearly separable dataset



Number of linearly separable dichotomies

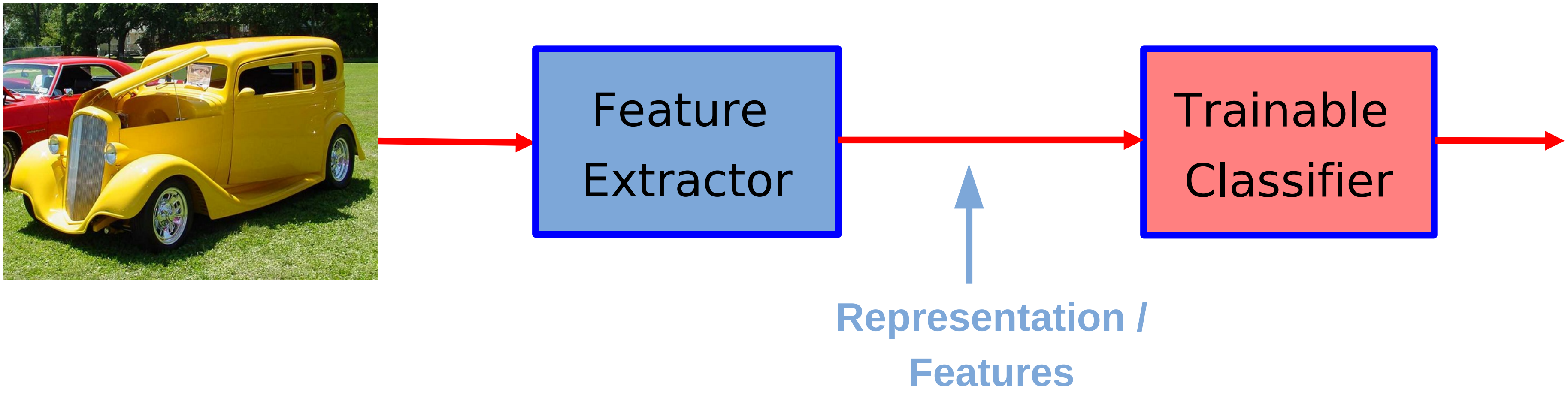
- **The probability that a dichotomy over P points in N dimensions is linearly separable goes to zero as P gets larger than N**
  - [Cover's theorem 1966]



- Problem: there are 2^P possible dichotomies of P points.
- Only about N are linearly separable.
- If P is larger than N, the probability that a random dichotomy is linearly separable is very, very small.

Solution: representations (a.k.a. features)

- **Extracting relevant features from the raw input**
- **Computing good representations of the input**
- **The feature extractor <u>must</u> be non-linear**
- **Simple solution: expand the dimension non-linearly**
  - But how?



Example: monomial features

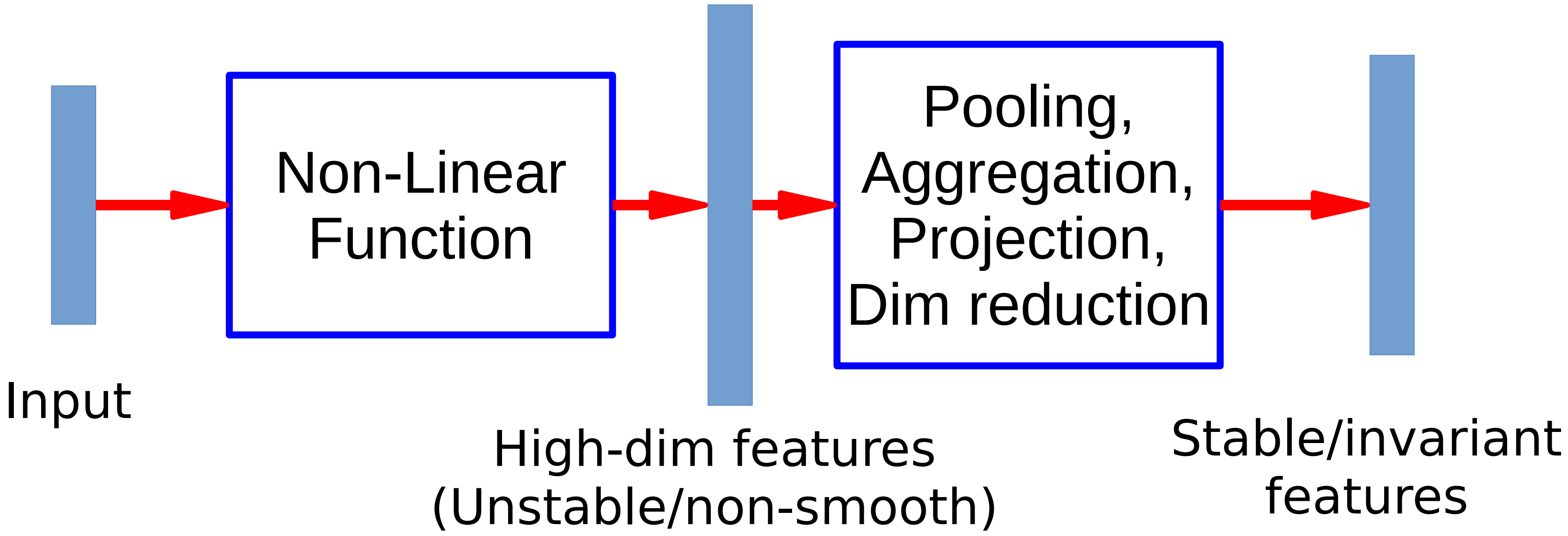
- **Feature extractor computes cross products of input variables**
- **A linear classifier on top computes a polynomial of input variables**

$$\Phi(x_1, x_2) = [1, x_1, x_2, x_1 x_2, x_1^2, x_2^2]$$

- **generalizable to degree d**
- **Unfortunately impractical for large d**
- **Number of features is d choose N, which grows like N^d**
- **But d=2 is used a lot in "attention" circuits.**

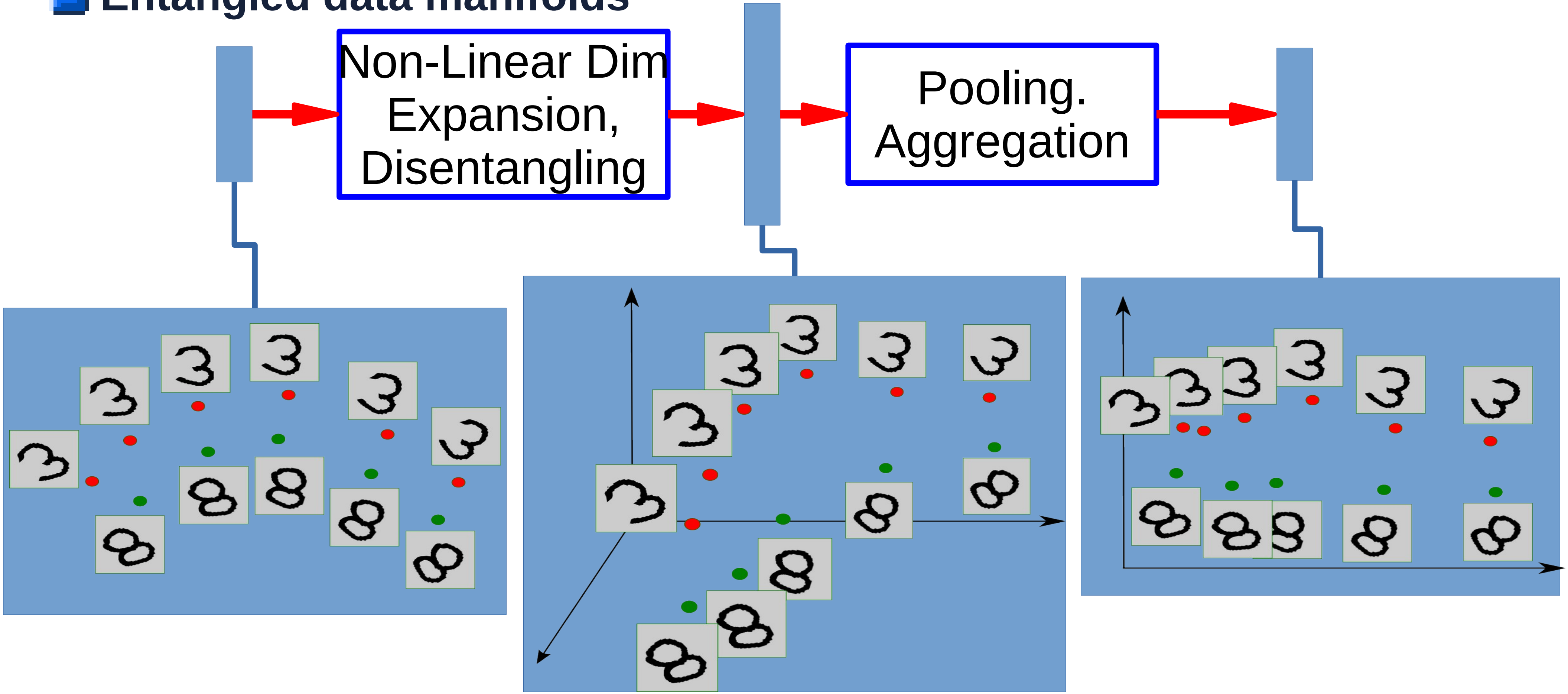
Basic Idea for Invariant Feature Learning

- **Embed the input non-linearly into a high(er) dimensional space**
  - In the new space, things that were non separable may become separable
- **Pool regions of the new space together**
  - Bringing together things that are semantically similar. Like pooling.



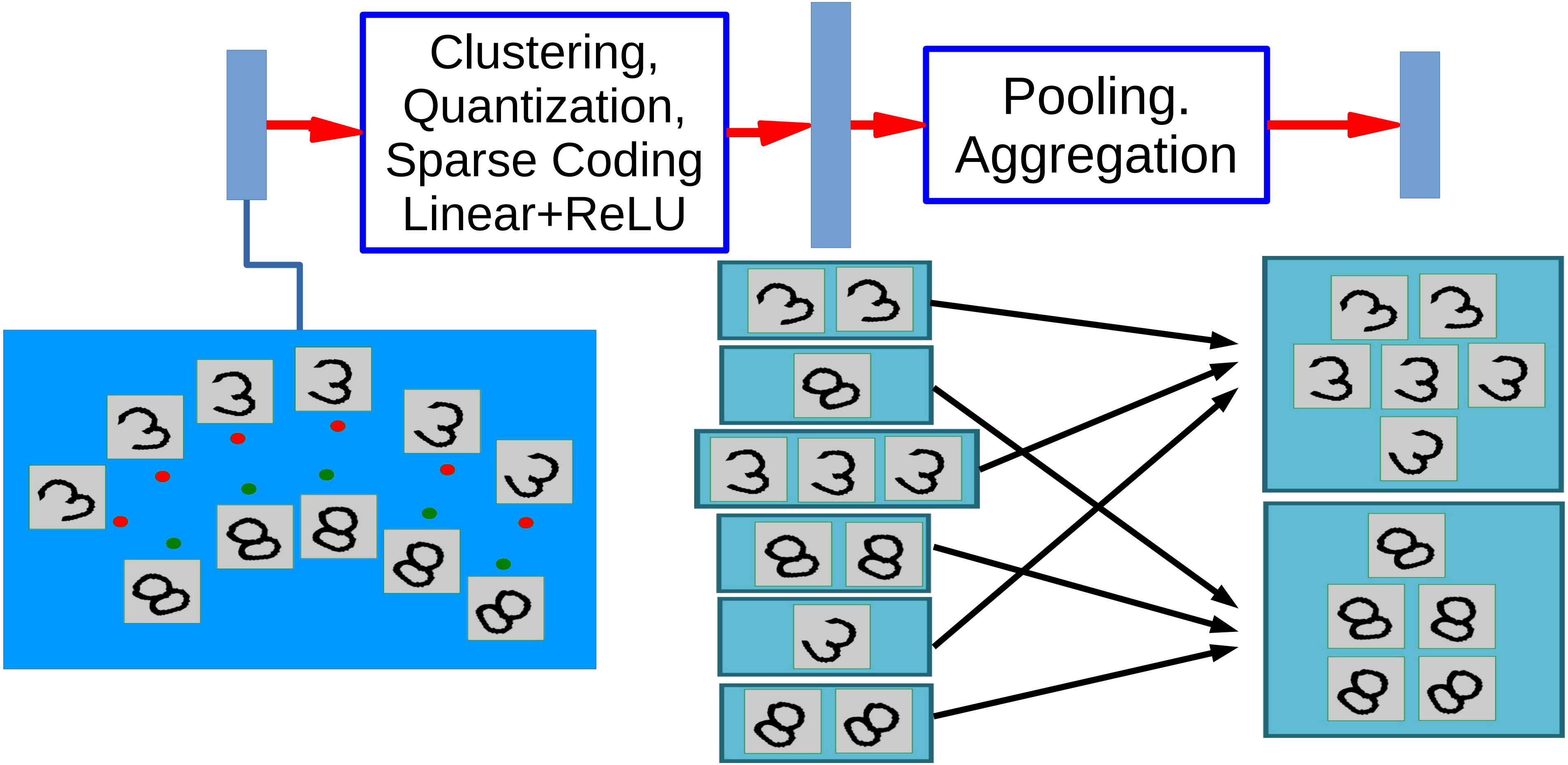
Non-Linear Expansion → Pooling

- **Entangled data manifolds**



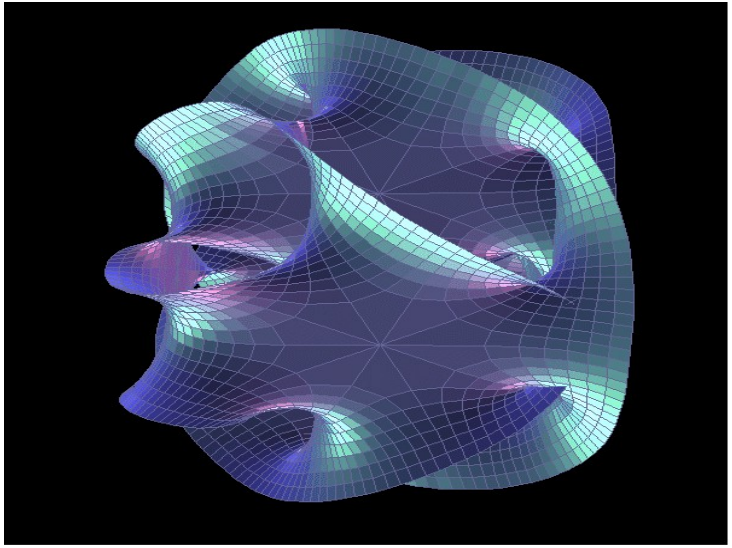
Sparse Non-Linear Expansion → Pooling

- **Use non-linear fn to break things apart, pool together similar things**



Discovering the Hidden Structure in High-Dimensional Data: The manifold hypothesis

- **Learning Representations of Data:**
  - **Discovering & disentangling the independent explanatory factors**
- **The Manifold Hypothesis:**
  - Natural data lives in a low-dimensional (non-linear) manifold
  - Because variables in natural data are mutually dependent



Discovering the Hidden Structure in High-Dimensional Data

- **Example: all face images of a person**
  - 1000x1000 pixels = 1,000,000 dimensions
  - But the face has 3 Cartesian coordinates and 3 Euler angles
  - And humans have less than about 50 muscles in the face
  - Hence the manifold of face images for a person has <56 dimensions
- **The perfect representations of a face image:**
  - Its coordinates on the face manifold
  - Its coordinates away from the manifold
- **We do not have good and general methods to learn functions that turns an image into this kind of representation**

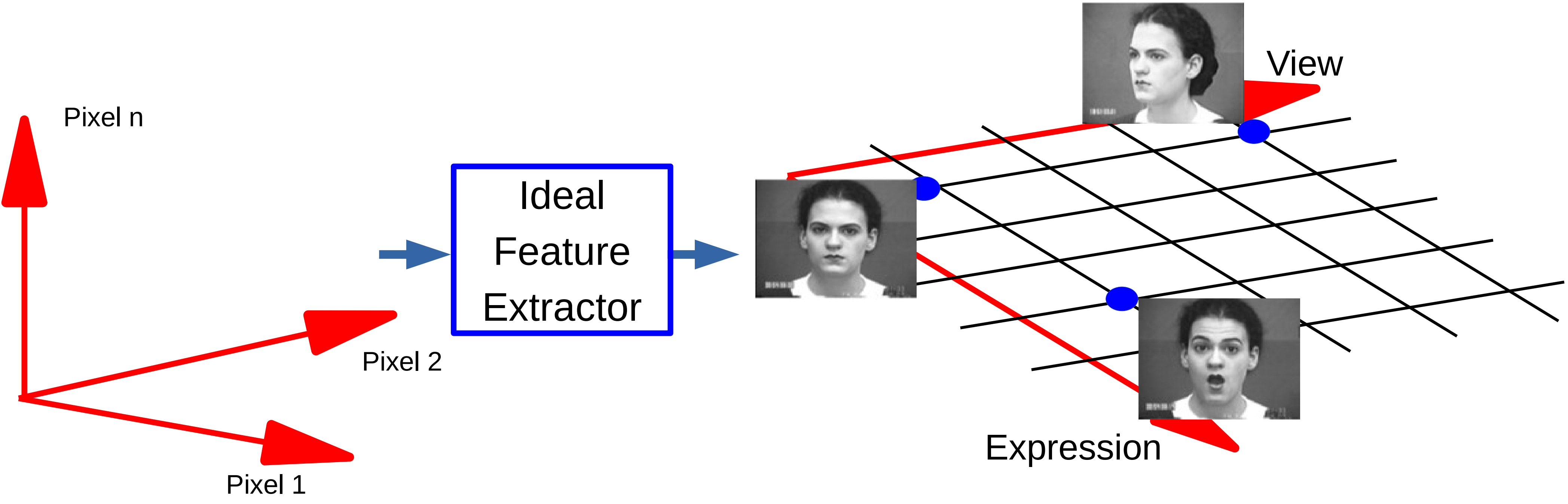
Ideal Feature Extractor

$$\begin{bmatrix} 1.2 \\ -3 \\ 0.2 \\ -2\ldots \end{bmatrix}$$

Face/not face
Pose
Lighting
Expression

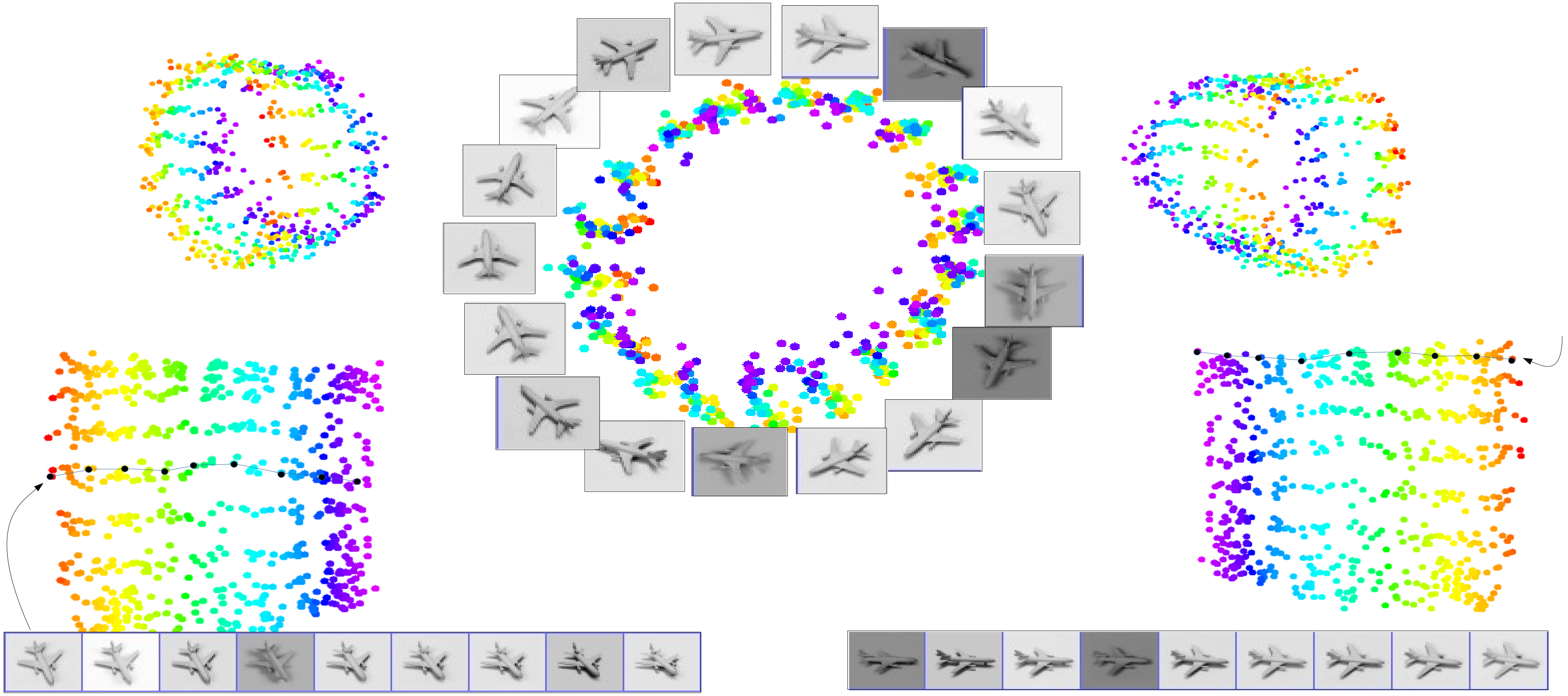
Disentangling factors of variation

The Ideal Disentangling Feature Extractor



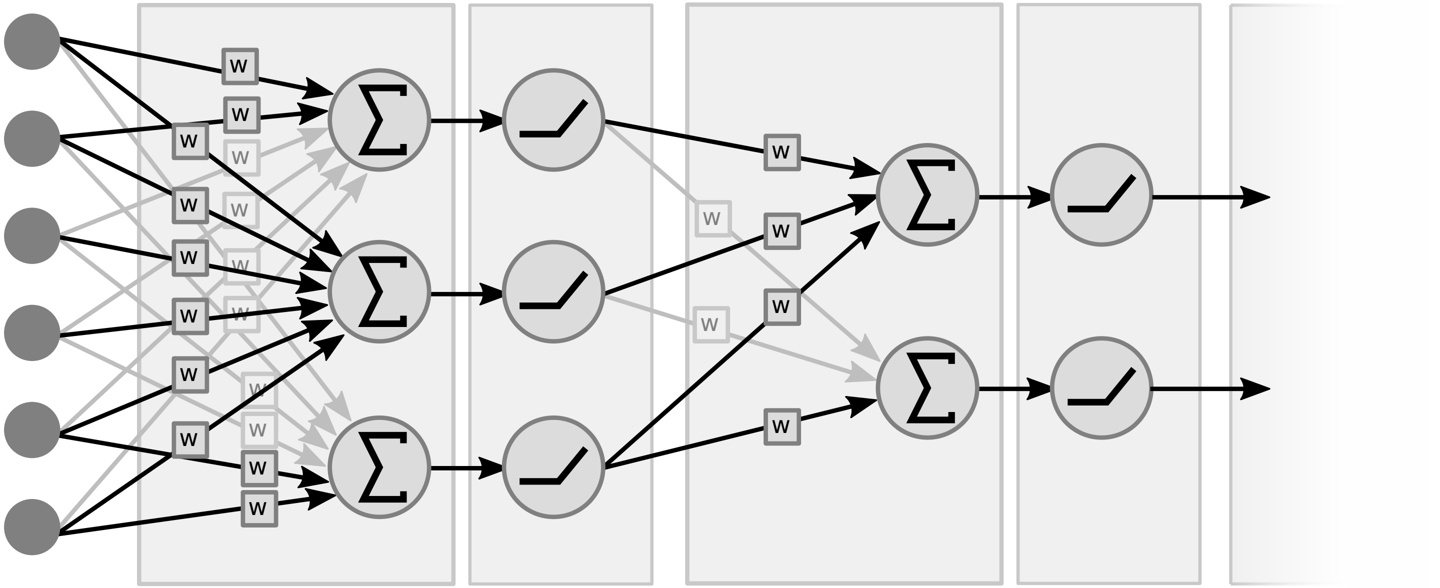
Data Manifold

[Hadsell et al. CVPR 2006]



Traditional Neural Net

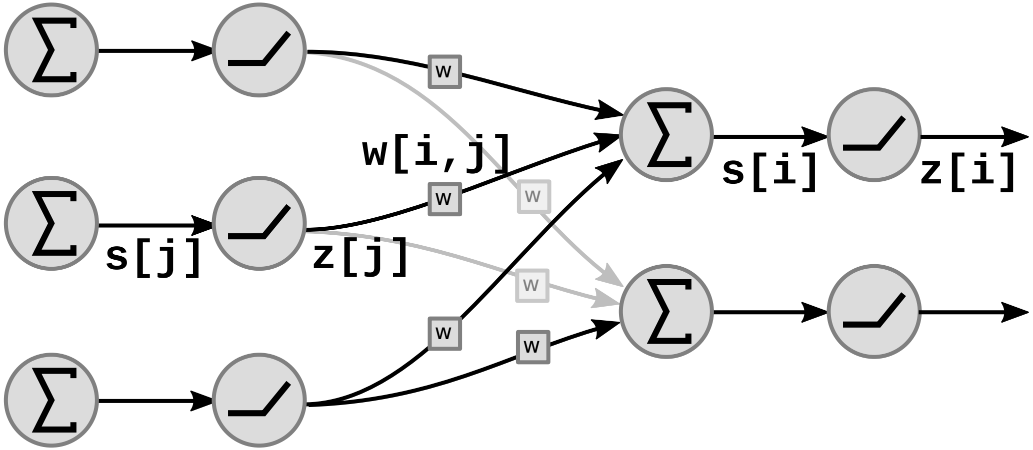
- **Stacked linear and non-linear functional blocks**
  - Weighted sums, matrix-vector product
  - Point-wise non-linearities (e.g. ReLu, tanh, ….)



Traditional Neural Net

- **Stacked linear and non-linear functional blocks**

$$s[i] = \sum_{j \in \mathrm{UP}(i)} w[i,j] \cdot z[j] \qquad\qquad z[i] = f(s[i])$$



Backprop through a non-linear function

- **Chain rule:**

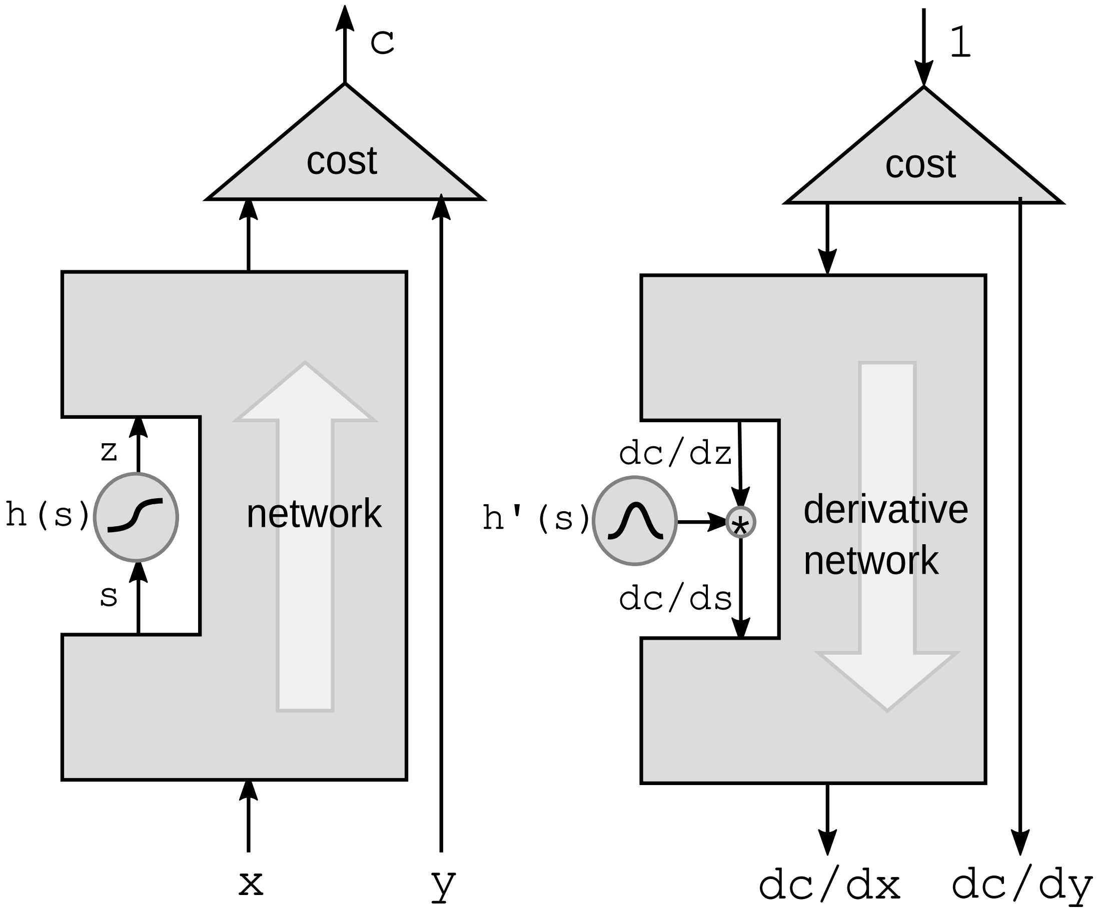
  g(h(s))' = g'(h(s)).h'(s)

  dc/ds = dc/dz*dz/ds

  dc/ds = dc/dz*h'(s)

- **Perturbations:**
  - Perturbing s by ds will perturb z by: dz=ds*h'(s)
  - This will perturb c by

    dc = dz*dc/dz = ds*h'(s)*dc/dz
  - Hence: dc/ds = dc/dz*h'(s)



Backprop through a weighted sum

- **Perturbations:**
  - Perturbing z by dz will perturb s[0],s[1],s[2] by ds[0]=w[0]*dz, ds[1]=w[1]*dz, ds[2]=w[2]*dz
  - This will perturb c by

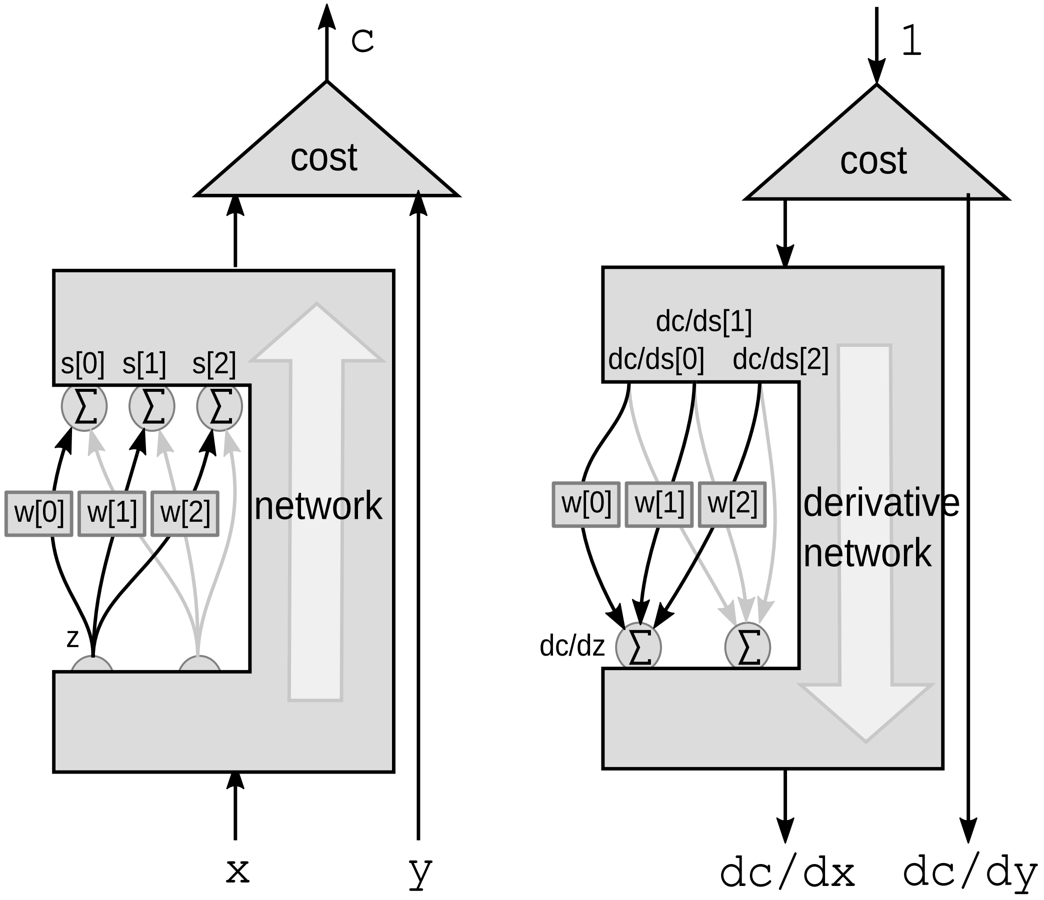
    dc = ds[0]*dc/ds[0]+

    ds[1]*dc/ds[1]+

    ds[2]*dc/ds[2]
  - Hence: dc/dz = dc/ds[0]*w[0]+

    dc/ds[1]*w[1]+

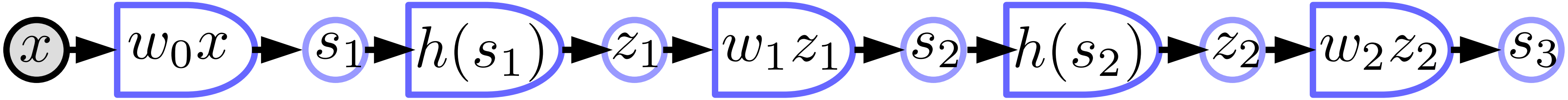
    dc/ds[2]*w[2]+



Block Diagram of a Traditional Neural Net

Y. LeCun

- **linear blocks** $s_{k+1} = w_k z_k$
- **Non-linear blocks** $z_k = h(s_k)$



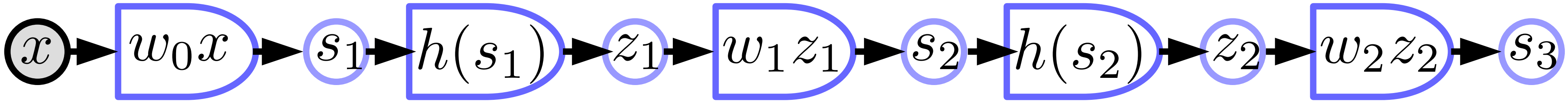
PyTorch definition

- **Object-oriented version**
  - Uses predefined nn.Linear class, (which includes a bias vector)
  - Uses torch.relu function
  - State variables are temporary

```
import torch
from torch import nn
image = torch.randn(3, 10, 20)
d0 = image.nelement()

class mynet(nn.Module):
    def __init__(self, d0,d1,d2,d3):
        super().__init__()
        self.m0 = nn.Linear(d0, d1)
        self.m1 = nn.Linear(d1, d2)
        self.m2 = nn.Linear(d2, d3)
    def forward(self, x):
        z0 = x.view(-1)  ## flatten input tensor
        s1 = self.m0(x)
        z1 = torch.relu(s1)
        s2 = self.m1(z1)
        z2 = torch.relu(s2)
        s3 = self.m2(z2)
        return s3

model = mynet(d0,60,40,10)
out = model(image)
```



Backprop through a functional module

- **Using chain rule for vector functions**

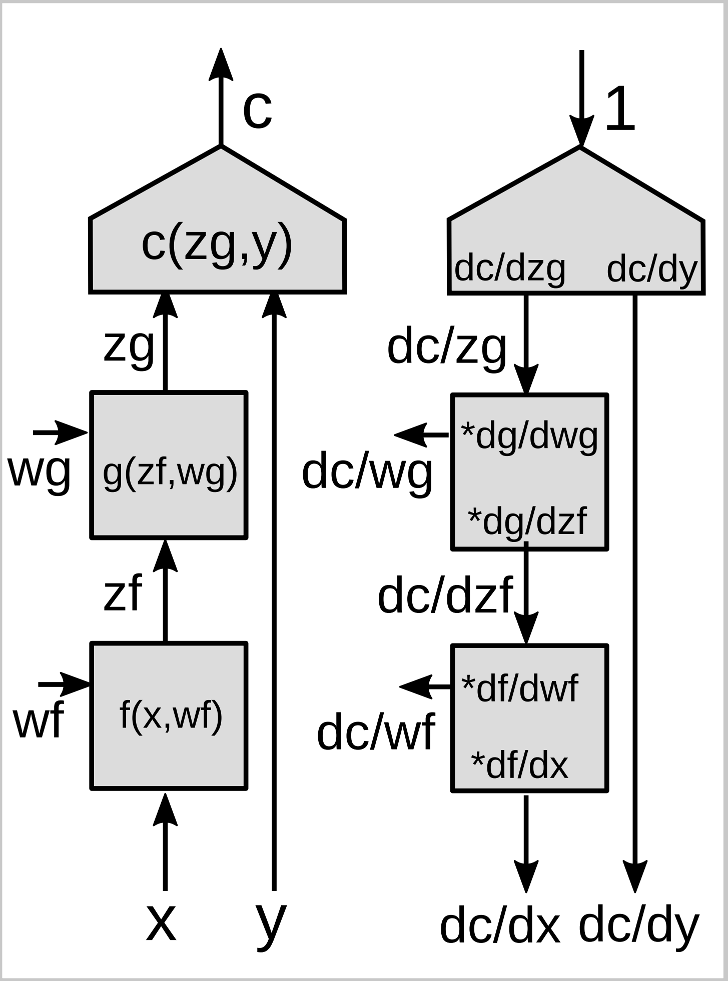
$$z_g : [d_g \times 1] \quad z_f : [d_f \times 1]$$

$$\frac{\partial c}{\partial z_f} = \frac{\partial c}{\boxed{\partial z_g}} \frac{\boxed{\partial z_g}}{\partial z_f}$$

$$[1 \times d_f] = [1 \times d_g] * [d_g \times d_f]$$

- **Jacobian matrix**
  - Partial derivative of i-th output w.r.t. j-th input

$$\left(\frac{\partial z_g}{\partial z_f}\right)_{ij} = \frac{(\partial z_g)_i}{(\partial z_f)_j}$$



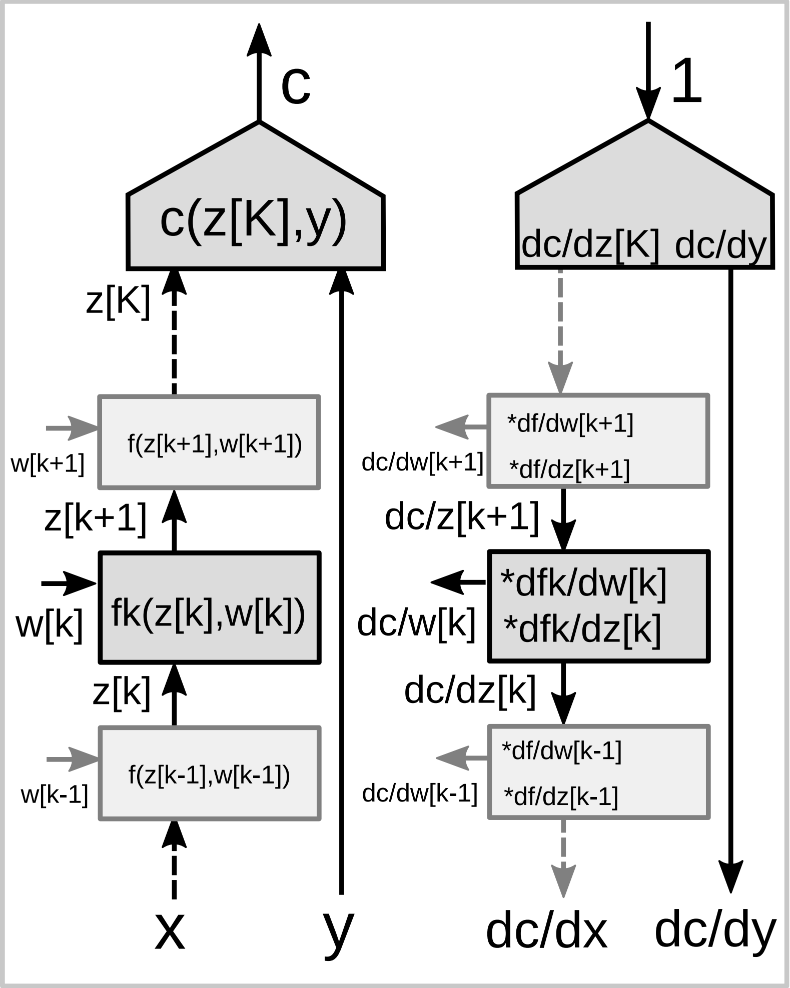
Backprop through a multi-stage graph

- **Using chain rule for vector functions**

$$\frac{\partial c}{\partial z_k} = \frac{\partial c}{\partial z_{k+1}} \frac{\partial z_{k+1}}{\partial z_k} = \frac{\partial c}{\partial z_{k+1}} \frac{\partial f_k(z_k, w_k)}{\partial z_k}$$

$$\frac{\partial c}{\partial w_k} = \frac{\partial c}{\partial z_{k+1}} \frac{\partial z_{k+1}}{\partial w_k} = \frac{\partial c}{\partial z_{k+1}} \frac{\partial f_k(z_k, w_k)}{\partial w_k}$$

- **Two Jacobian matrices for the module:**
  - One with respect to z[k]
  - One with respect to w[k]



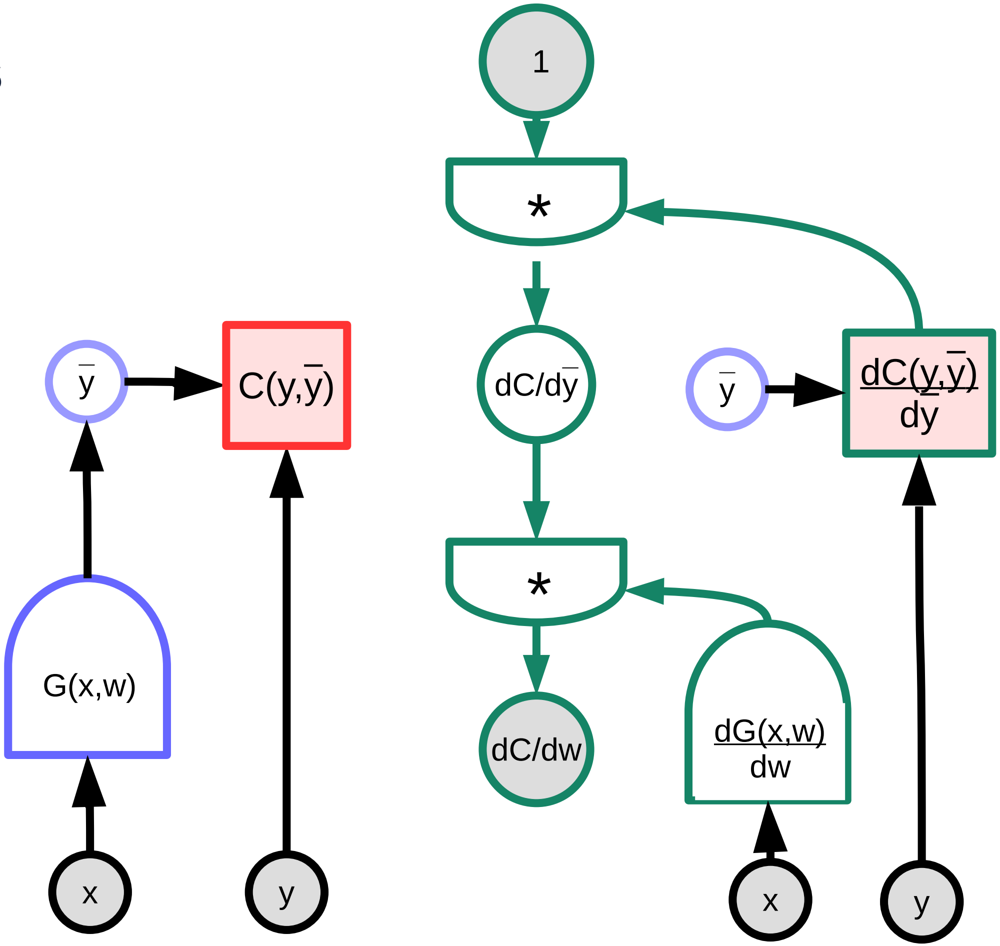
Backprop = propagation through a transformed graph

- **Derivative of composed functions**

$$C(G(w))' = C'(G(w))G'(w)$$

$$\frac{\partial C(y,\bar{y})}{\partial w} = \frac{\partial C(y,\bar{y})}{\boxed{\partial \bar{y}}} \frac{\boxed{\partial \bar{y}}}{\partial w}$$

$$\frac{\partial C(y,\bar{y})}{\partial w} = \frac{\partial C(y,\bar{y})}{\partial \bar{y}} \frac{\partial G(x,w)}{\partial w}$$



Gradient, Jacobian, ….

- **Dimensions:**

$$y, \bar{y} : [M \times 1] \quad w : [N \times 1]$$

$$\frac{\partial C(y,\bar{y})}{\partial w} = \frac{\partial C(y,\bar{y})}{\partial \bar{y}} \frac{\partial \bar{y}}{\partial w}$$

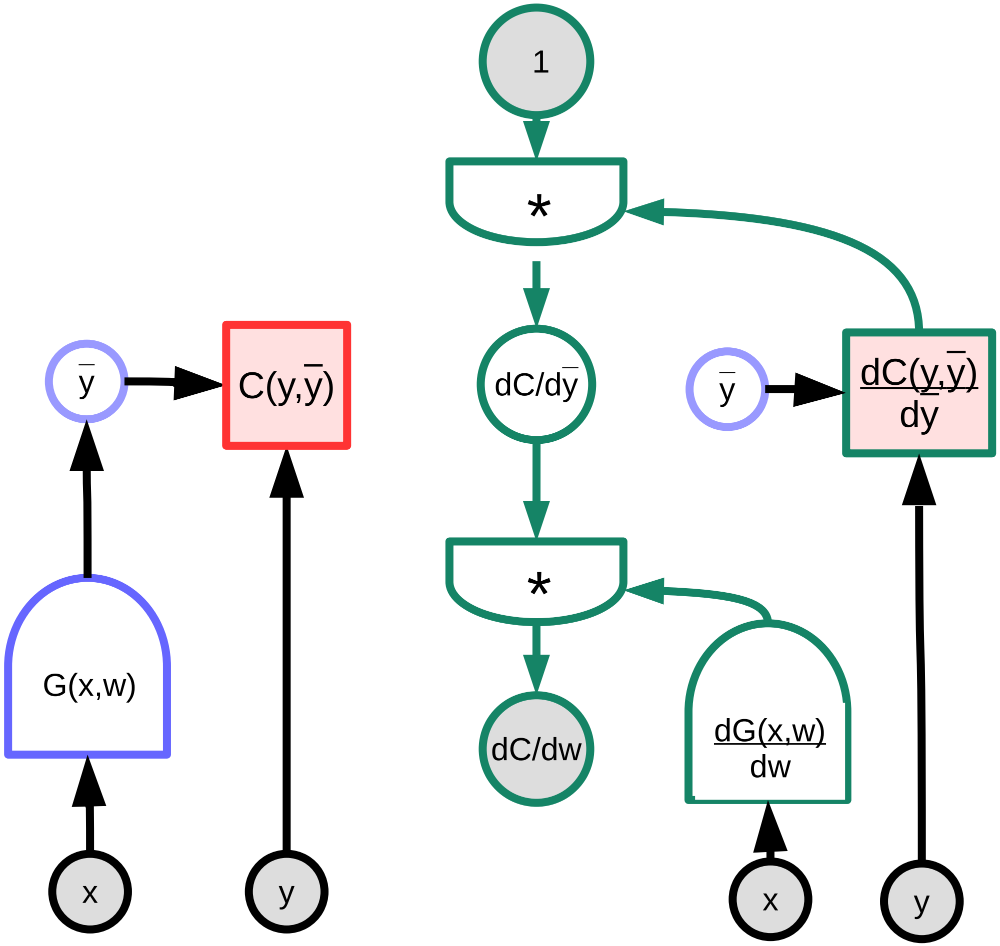
$$[1 \times N] = [1 \times M] \cdot [M \times N]$$

- **Row vector = row vector . matrix**

$$\frac{\partial C(y,\bar{y})}{\partial w} = \frac{\partial C(y,\bar{y})}{\partial \bar{y}} \frac{\partial G(x,w)}{\partial w}$$

$$[1 \times N] = [1 \times M] \cdot [M \times N]$$

- **Gradient = gradient . Jacobian**



Basic Modules

Linear	$Y = W.X$; $dC/dX = W^T . dC/dY$; $dC/dW = X\ dC/dY$
ReLU	$y = ReLU(x)$; if $(x<0)$ $dC/dx = 0$ else $dC/dx = dC/dy$
Duplicate	$Y1 = X, Y2 = X$; $dC/dX = dC/dY1 + dC/dY2$
Add	$Y = X1 + X2$; $dC/dX1 = dC/dY$; $dC/dX2 = dC/dY$
Max	$y = max(x1,x2)$; if $(x1>x2)$ $dC/dx1 = dC/dy$ else $dC/dx1=0$
LogSoftMax	$Yi = Xi - \log[\sum_j \exp(Xj)]$; .....???

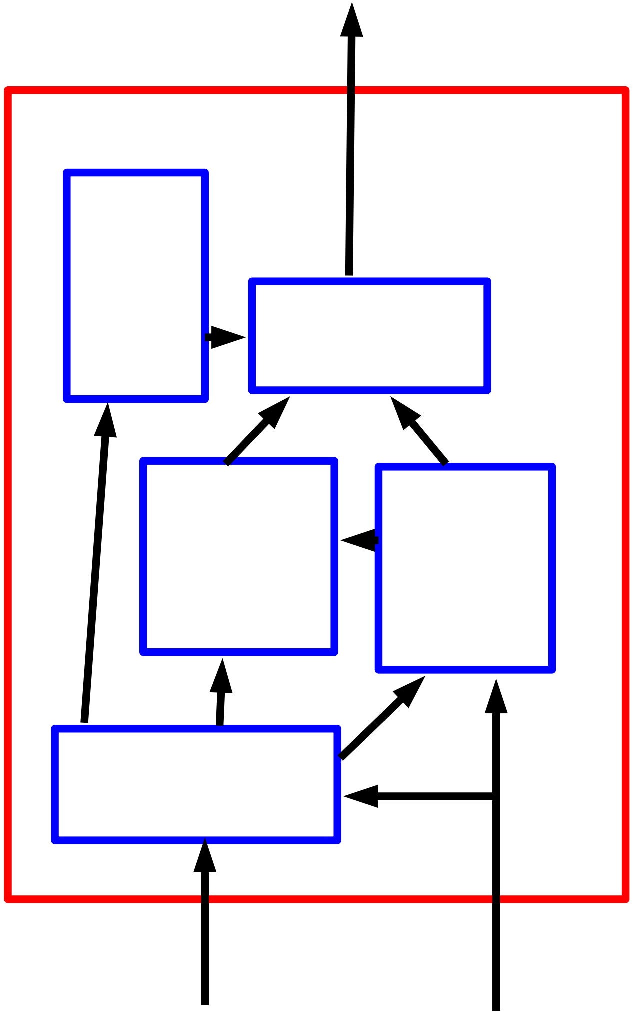
Non-Linear functions and Loss functions in PyTorch

- **ReLu, sigmoids and variations**
- **Squared error, cross-entropy, hinge, ranking loss and variants**

Any directed acyclic graph is OK for backprop

- **As long as there exist a partial order on the modules**

- **If the graph has loops, we need to “unroll” them.**
  - Recurrent networks and bakprop through time



Backprop in Practice

- **Use ReLU non-linearities (tanh and logistic are falling out of favor)**
- **Initialize the weights properly**
- **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 recent papers.**