Hardware for Training

MICRO Tutorial (2016)

Website: http://eyeriss.mit.edu/tutorial.html

Joel Emer, Vivienne Sze, Yu-Hsin Chen

Cost function for Model Training

Model output:

$$\mathbf{y} = f(\mathbf{x})$$

Desired output:

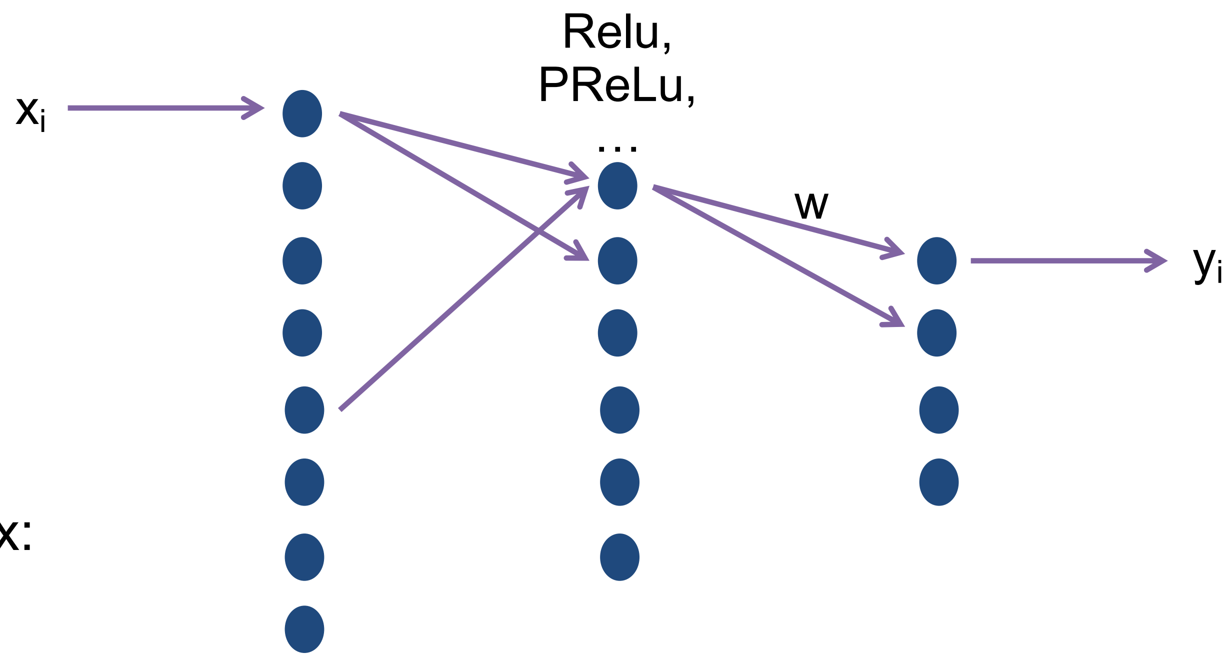
$$\mathbf{z}$$

Error:

$$e = (\mathbf{y}-\mathbf{z})$$

Over all training inputs x:

Minimize $\Sigma(\mathbf{y}-\mathbf{z})^2$



What do we vary to minimize the error?

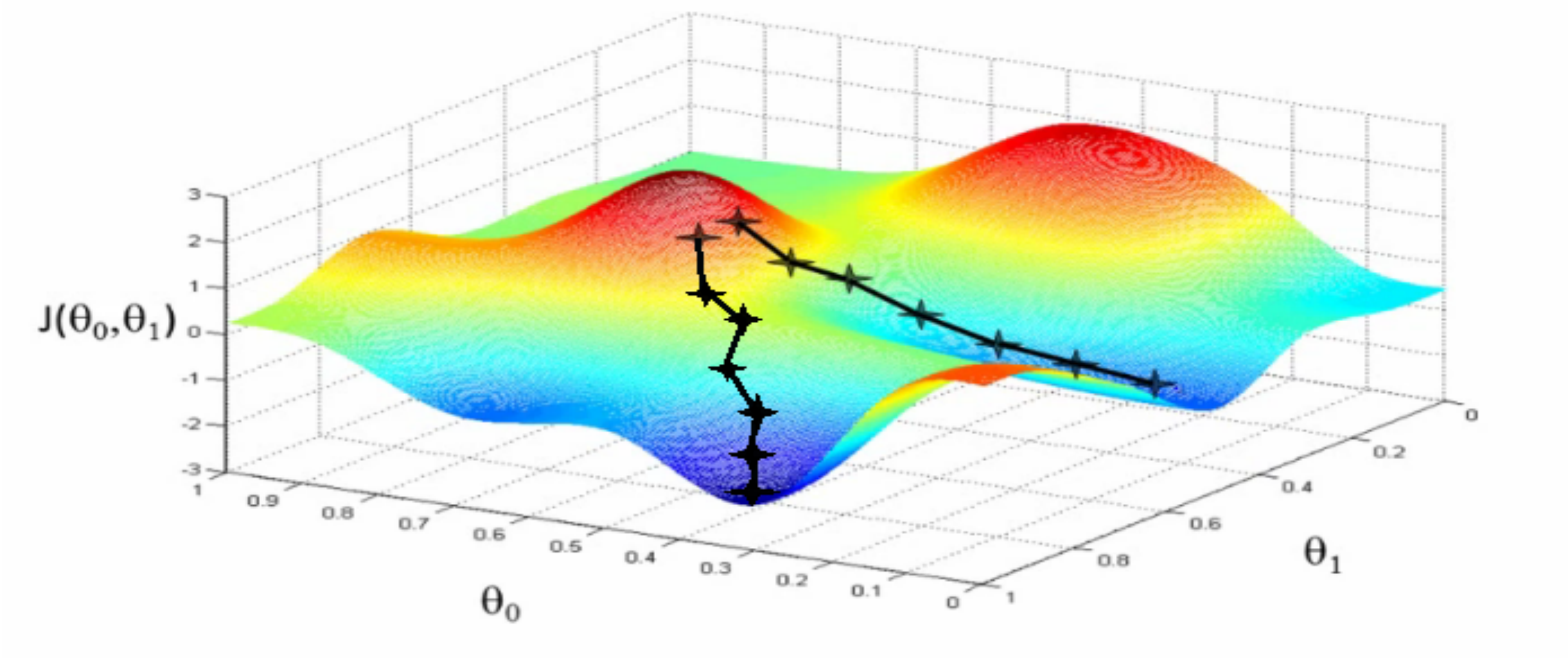
Training Optimization Problem

- **Model parameters** — θ **(include bias, weights, …)**
- **Model output** — $y(\theta) = f(x, \theta)$
- **Desired output** — z
- **Error** — $e(\theta) = y(\theta) - z$
- **Cost function*** — $E(\theta) = \Sigma e(\theta)^2$
- **Minimization** — $dE(\theta)/d\theta = 0$ **(but no closed form)**

* Over all inputs in the training set

Steepest descent

Classical first order iterative optimization scheme:
Gradient is steepest descent – follow it!



$\theta^{n+1} = \theta^n - \alpha \cdot dE(\theta^n)/d\theta$

where α is the step size along the gradient…

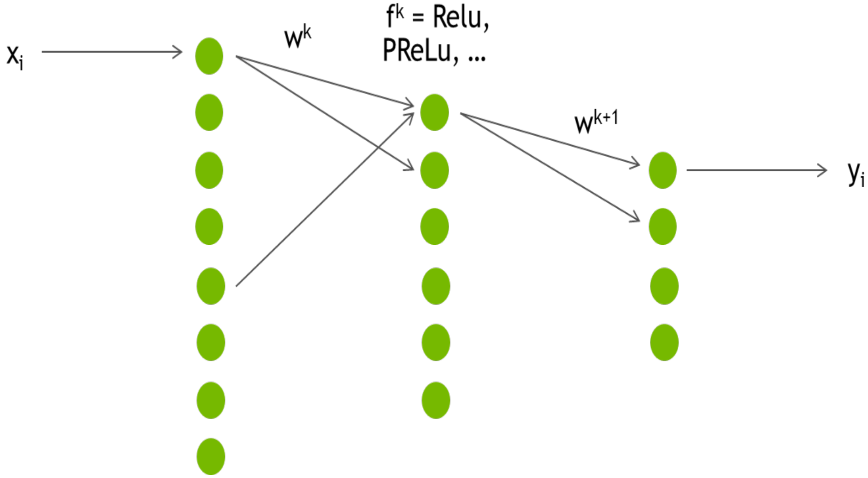
Calculating Steepest Descent

Also called error back-propagation

- **Steepest descent**
  $\boldsymbol{\theta^{n+1} = \theta^{n} - \alpha \cdot dE(\theta)/d\theta}$

- $\boldsymbol{E(\theta) = \Sigma e(\theta)^2}$
  $\boldsymbol{= \Sigma(y(\theta)-z)^2}$

- $\boldsymbol{dE(\theta)/d\theta = 2 \cdot \Sigma[(y(\theta)-z) \cdot dy(\theta)/d\theta]}$



error **e** back-propagation

Chain rule -> Back propagation

- The chain rule of calculus allows one to calculate the derivative of a layered network, i.e., a composition of functions, iteratively working backwards through the layers using the (feature map) values of the layer, i.e., function, and the derivative from the next layer.

- Back propagation is the process of doing this calculation numerically for a given input.

Per Layer Calculations

$$\mathbf{y} = f(\mathbf{x})$$

For layer k:

Inputs: x^k

Weights: w^k

Outputs: y^k

So

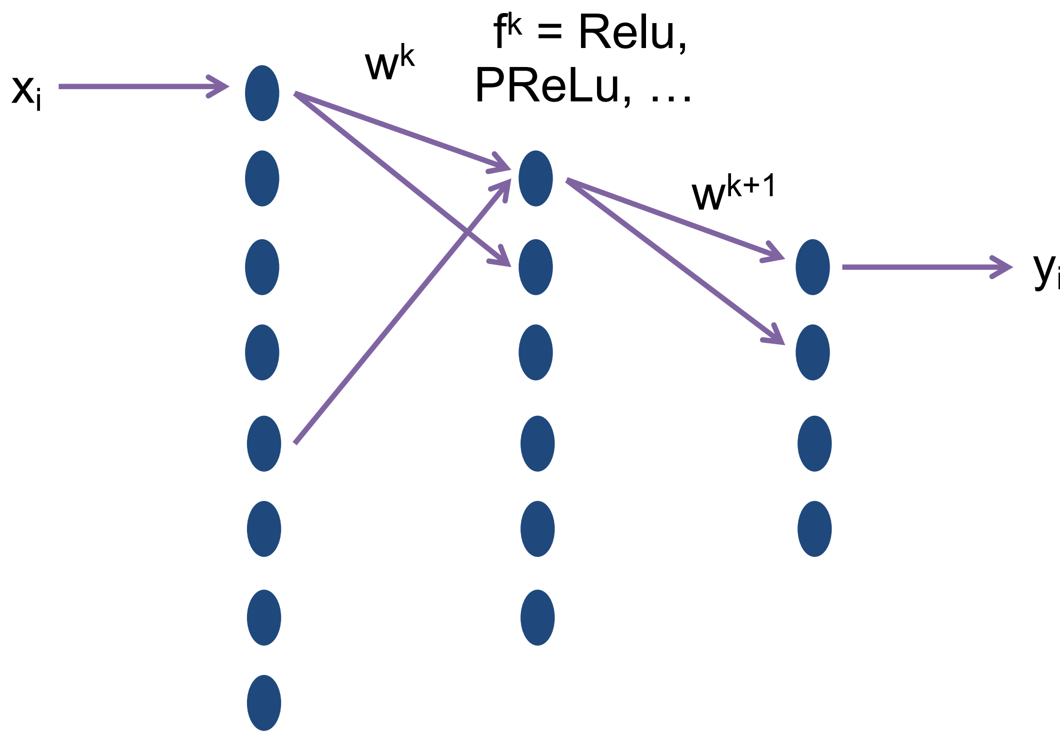
$$y_i^k = f^k\left[\Sigma(w_{ij}^k x_j^k)\right]$$

Where

$$x_j^k = y_j^{k-1}$$

or

$$\mathbf{y}^k = f^k(\mathbf{y}^{k-1},\theta)$$



Layer Operation Composition

- **Steepest descent** $\theta^{k+1} = \theta^k - \alpha \cdot dE/d\theta$
- **Derivative (1)** $dE/d\theta = 2 \cdot \Sigma[(y(\theta)-z) \cdot dy(\theta)/d\theta]$
- **Model output** $y = f(x)$
  $y^n = f^n(y^{n-1}) = f^n(f^{n-1}(y^{n-2}))$
- **Layer k** $y^k = f^k(y^{k-1}) = f^k(f^{k-1}(y^{k-2}))$

Chain rule

- Chain rule for functions

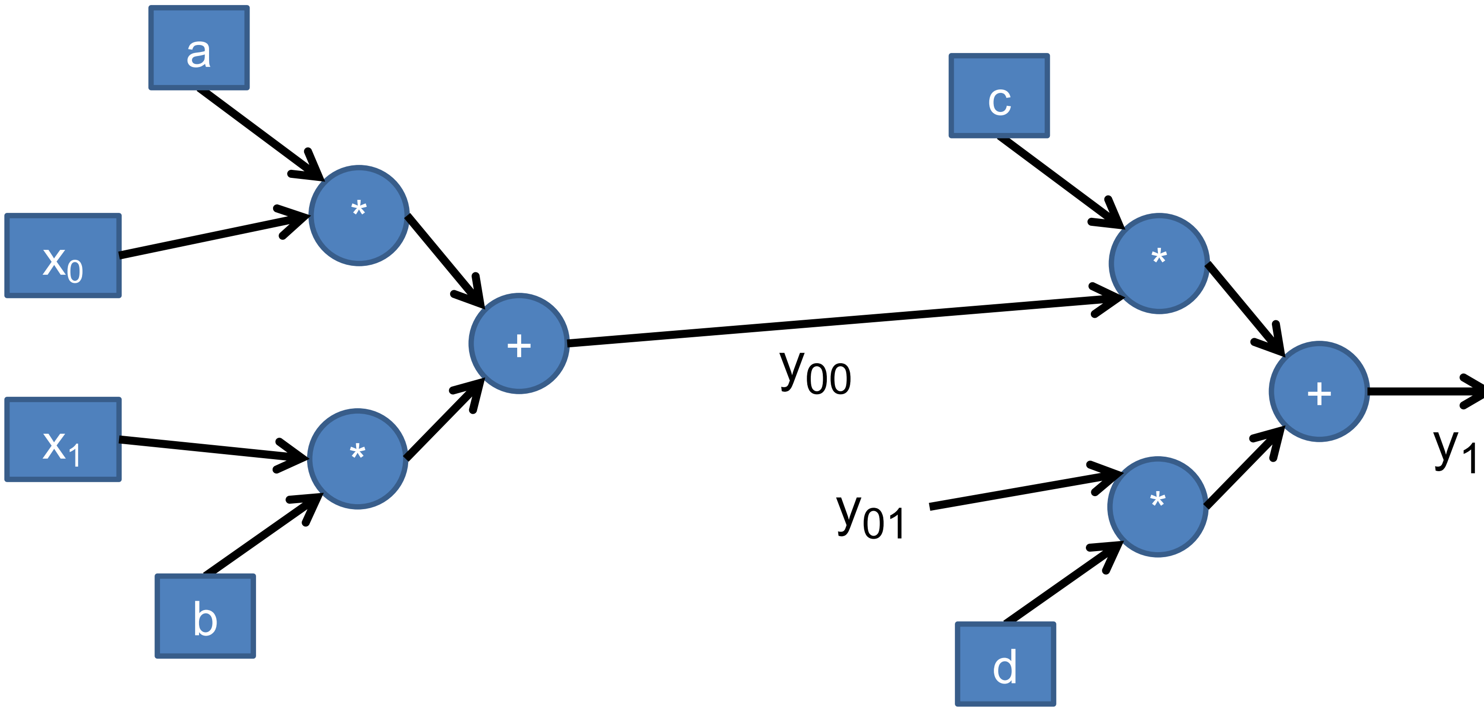
$$y = f(g(x)) \qquad y' = f'(g(x)) * g'(x)$$

$$y = f^{n}(y^{n-1}) = f^{n}(f^{n-1}(y^{n-2})) \qquad y' = f^{n\prime}(f^{n-1}(y^{n-2})) * f^{n-1\prime}(y^{n-2})$$

$$= f^{n\prime}(y^{n-1}) * f^{n-1\prime}(y^{n-2})$$

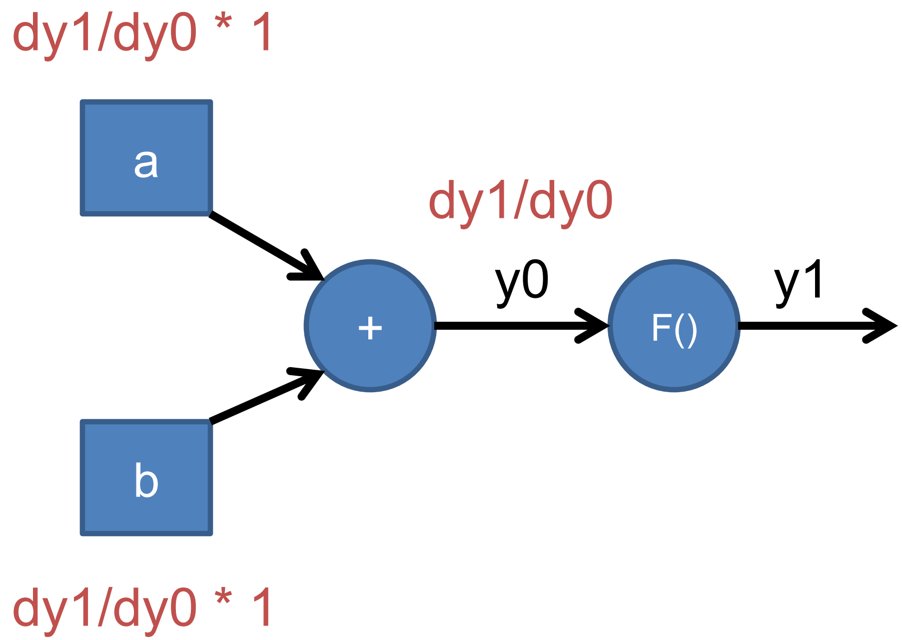
Back propagation

- $y_{00} = (a*x_0 + b*x_1)$
  $y_{01} = ...$
  $y_{10} = y_{00}*c + y_{01}*d$
- $dy_{10}/da = dy_{10}/dy_{00} * dy_{00}/da = c * x_0$



Back Propagation for Addition

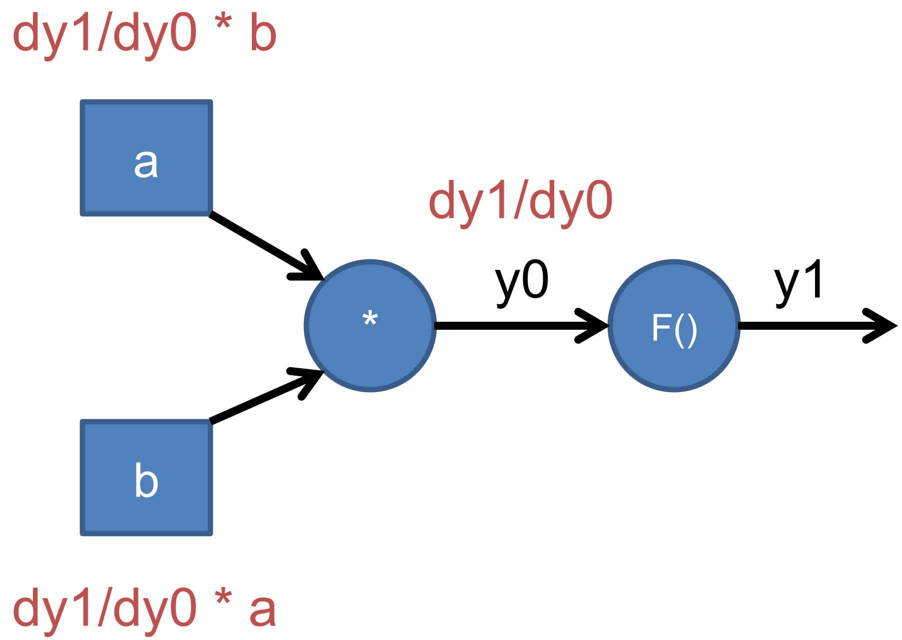
- $y_0 = a + b$
- $y_1 = f(y_0)$



- $dy_0/da = 1$
- $dy_0/db = 1$
- $dy_1/dy_0 = f'(y_0)$
- $dy_1/da = dy_1/dy_0 * dy_0/da = dy_1/dy_0 * 1 = dy_1/dy_0$
- $dy_1/db = dy_1/dy_0 * dy_0/db = dy_1/dy_0 * 1 = dy1/dy0$

Back Propagation for Multiplication

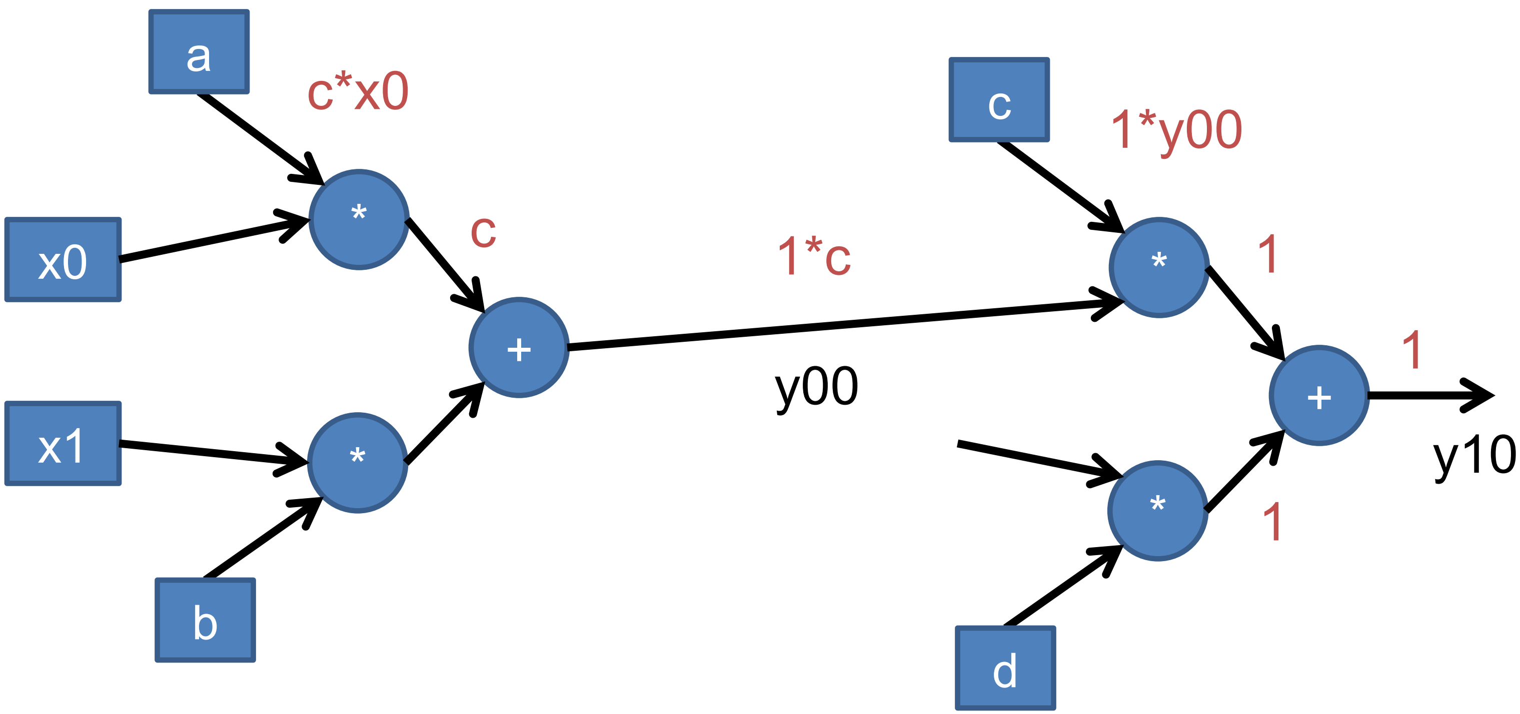
- y0 = a * b
- y1 = f(y0)



- dy0/da = b
- dy0/db = a
- dy1/dy0 = f'(y0)
- dy1/da = dy1/dy0 * dy0/da = dy1/dy0 * b
- dy1/db = dy1/dy0 * dy0/db = dy1/dy0 * a

Back propagation for Network

- y00 = (a*x0 + b*x1)
  y01 = ….
  y10 = y00*c + y01*d
- dy10/da = dy10/dy00 * dy00/da = c * x0



Back Propagation Recipe

Start point

- Select a initial set of weights (θ) and an input (x)

Forward pass

- For all layers
  - Compute layer outputs use as input for next layer (and save for later)

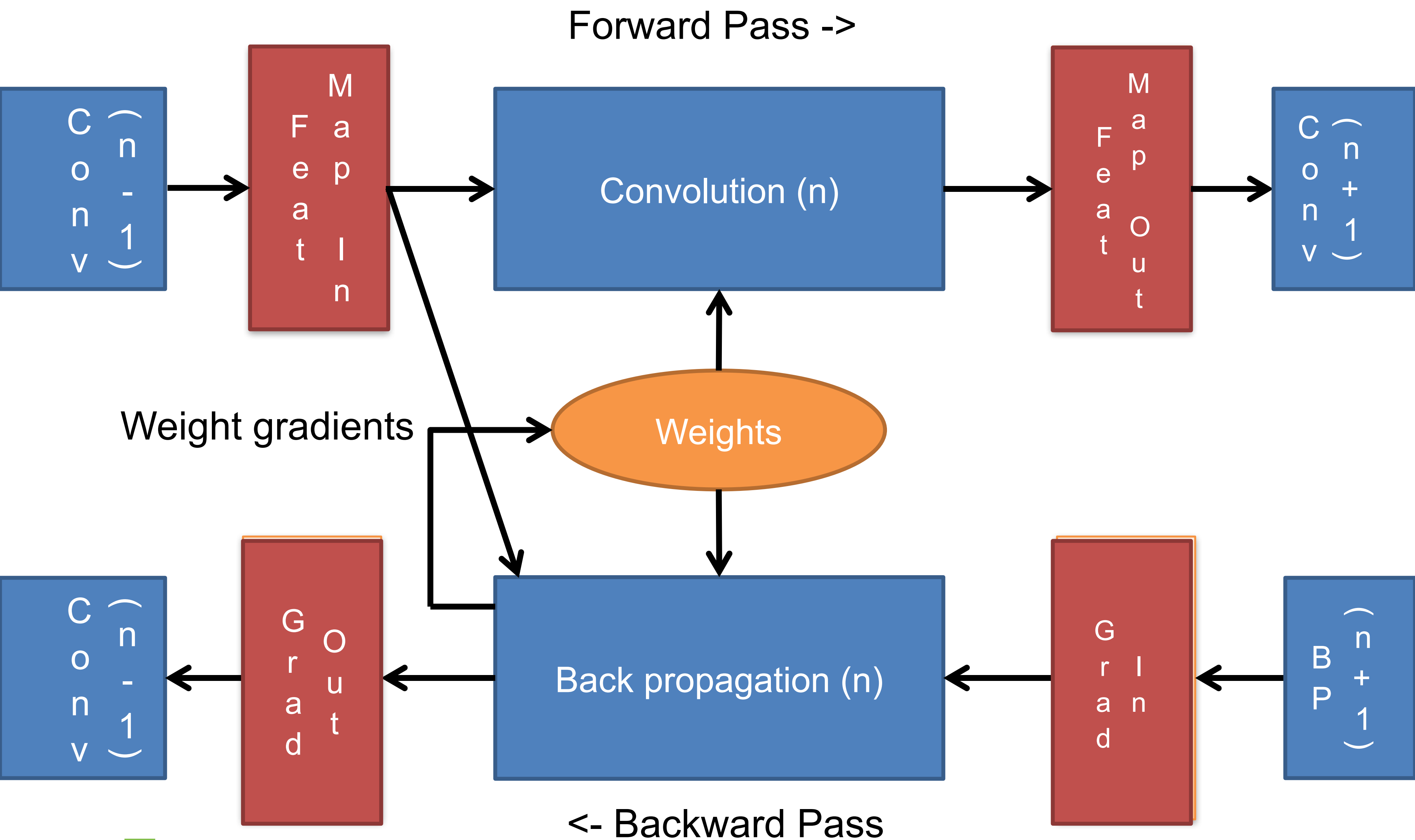
Backward pass

- For all layers (with output of previous layer and gradient of next layer)
  - Compute gradient, i.e., (partial) derivative, for layer
  - Back-propagate gradient to previous layer
  - Compute (partial) derivatives for (local) weights of layer

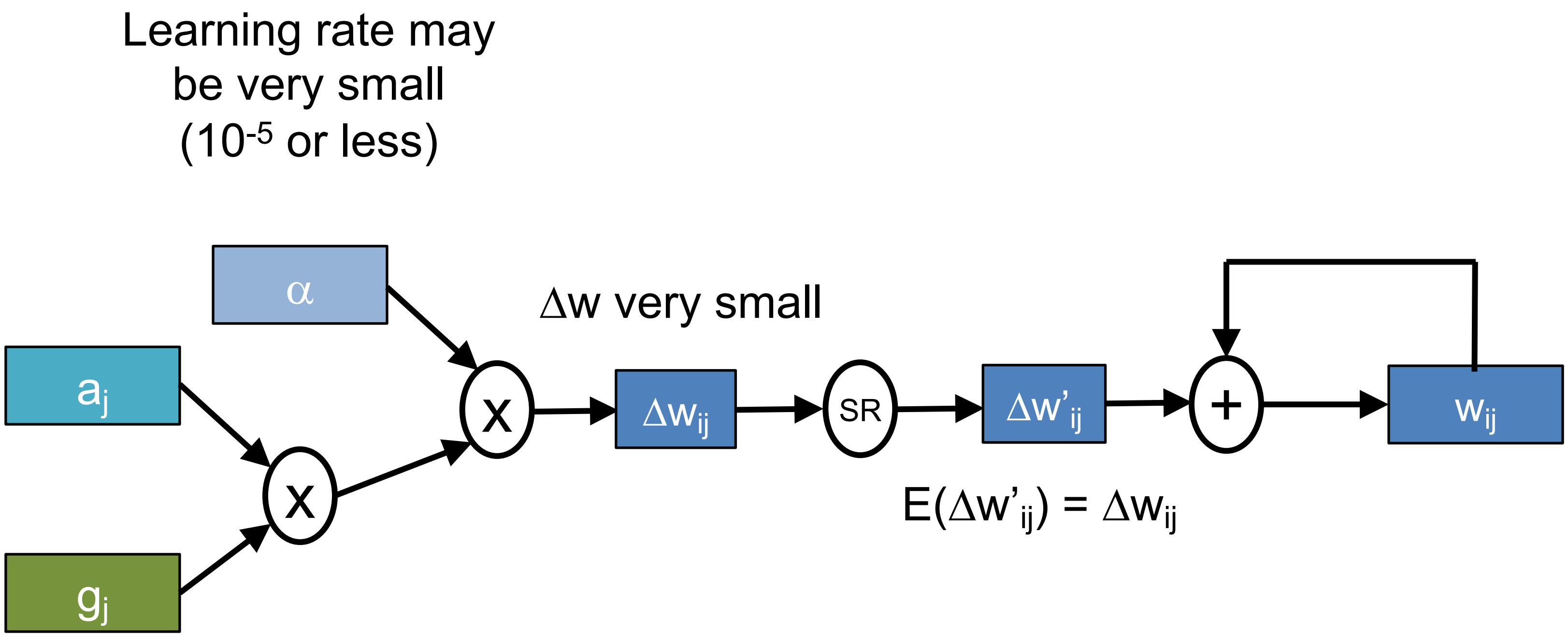
Calculate next set of weights

- $\theta^{k+1} = \theta^{k} - \alpha \cdot dE/d\theta$

Back Propagation



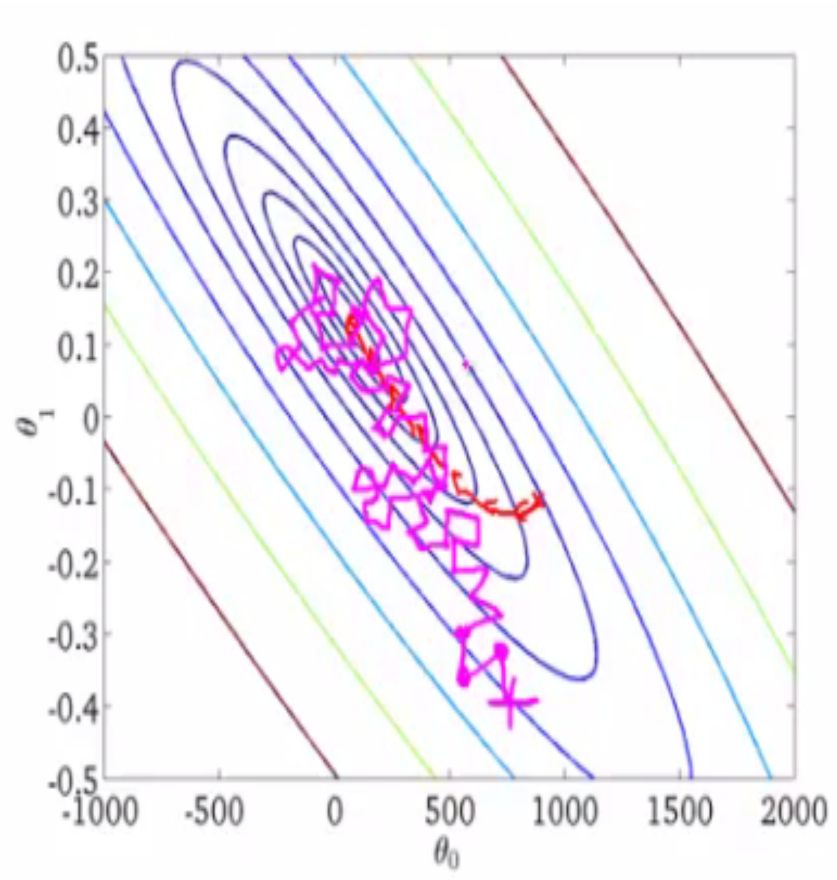
Precision on Training



- Beware truncating changes to zero
- Rounding can bias result -> use stochastic rounding

[Gupta et al., ICML 2015]

Back Propagation Batches



Issue:

- **N = 1 is often too noisy, weights may oscillate around the minimum**

Solution:

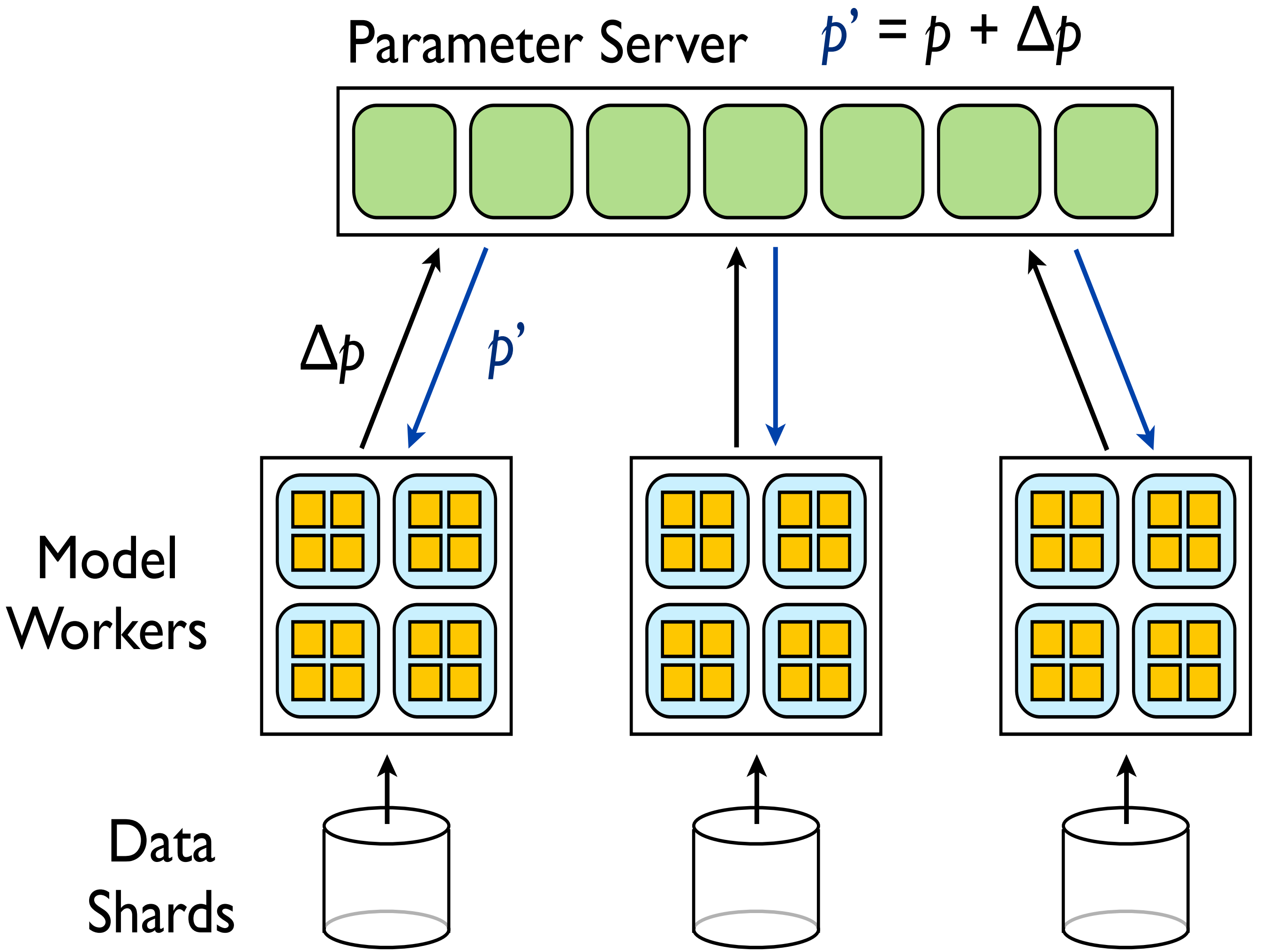
- **Use batches of N inputs…**
- **Max theoretical speed up: N**

Parallel creation of gradient

- **Steepest descent** $\theta^{k+1} = \theta^k - \alpha \cdot dE/d\theta$
- **Derivative** $dE/d\theta = 2 \cdot \Sigma[(y(\theta)-z) \cdot dy(\theta)/d\theta]$

Split sum of pieces of $dE/d\theta$
across different nodes!

Batch Parameter Update

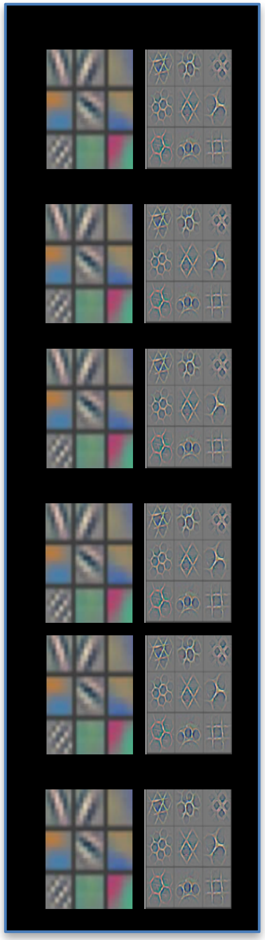


[Dean et al., NIPS 2012]

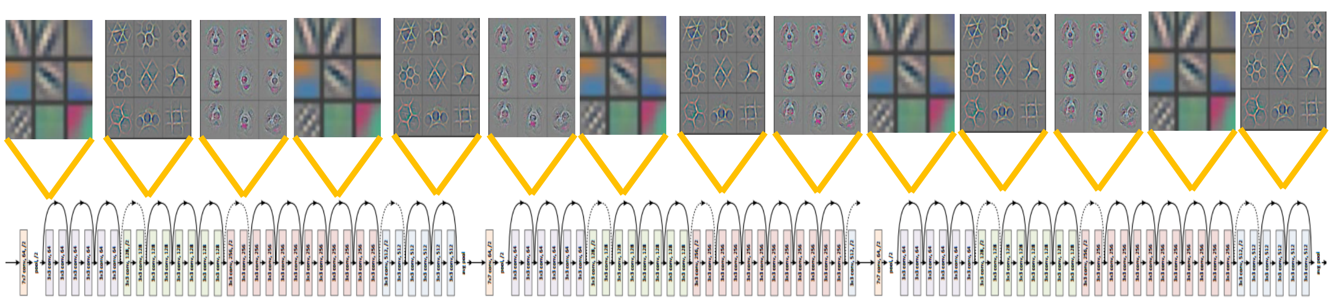
Training Uses a Lot of Memory

GPU memory usage proportional to network depth

GPU memory



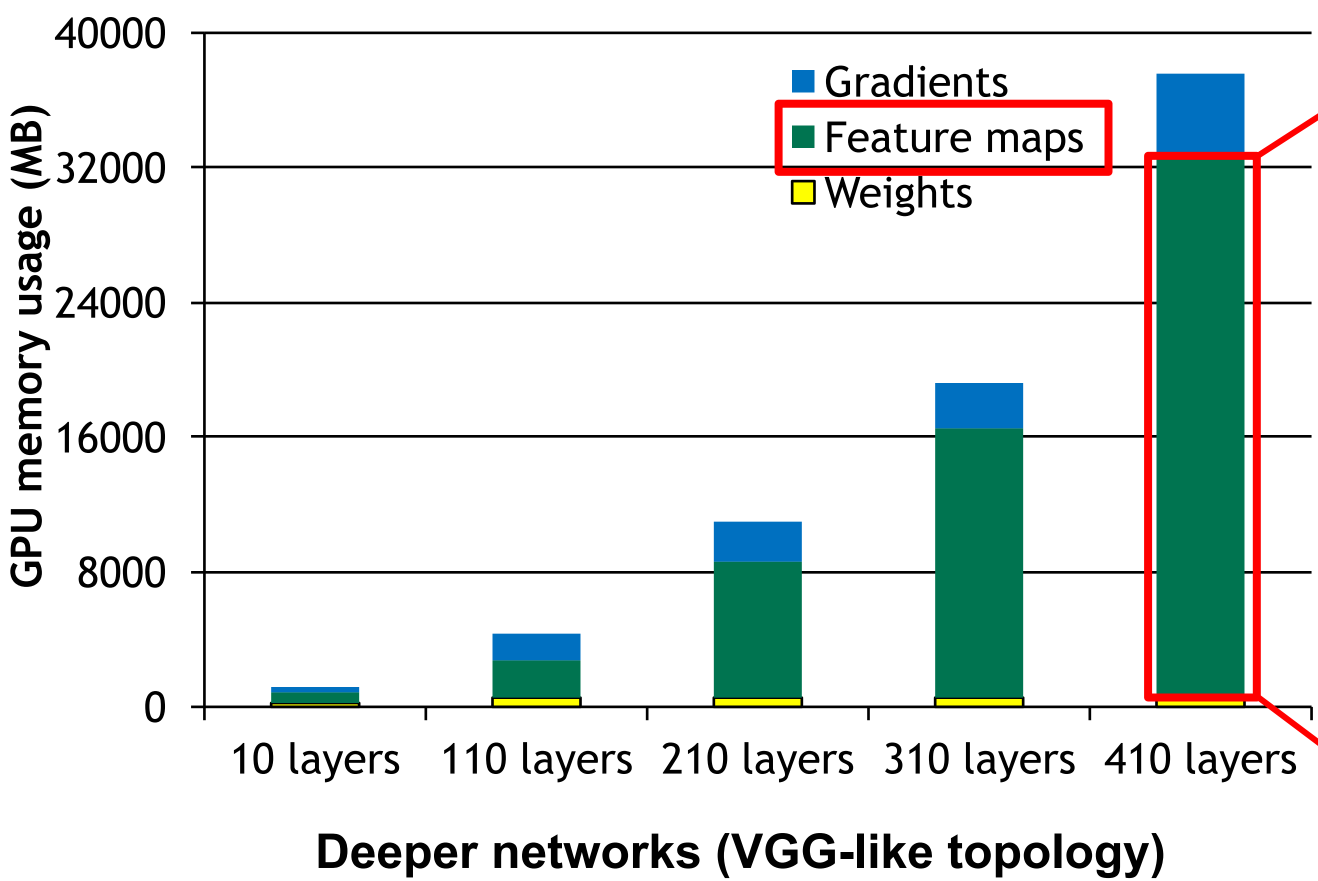
Feature maps

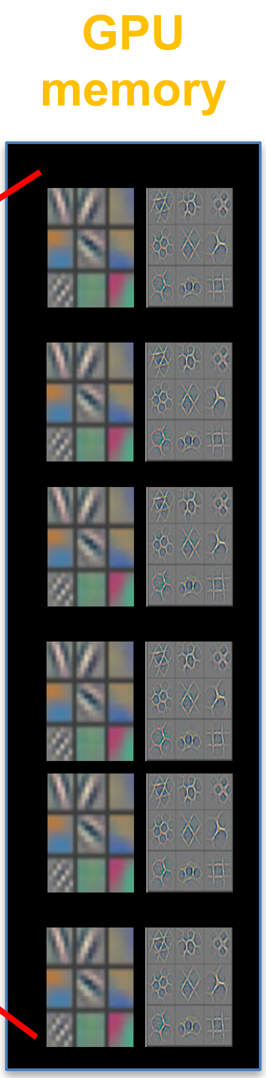


[Rhu et al., vDNN, MICRO 2016]

How Much Memory Is It?

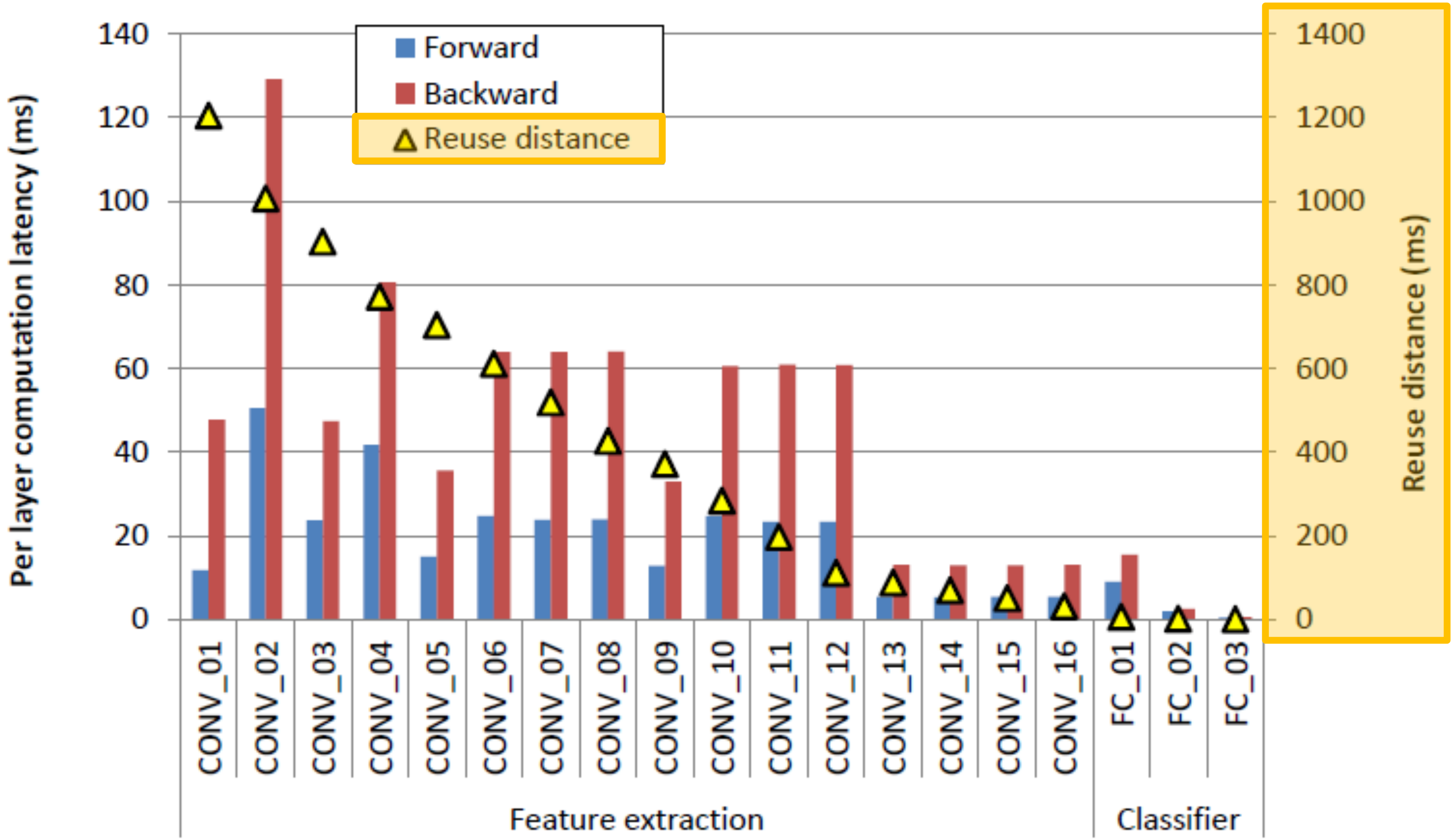
Up to Tens of Gigabytes





[Rhu et al., vDNN, MICRO 2016]

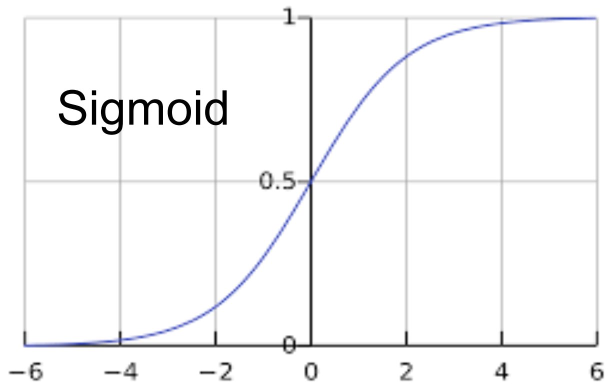
Reuse Distance of Feature Maps

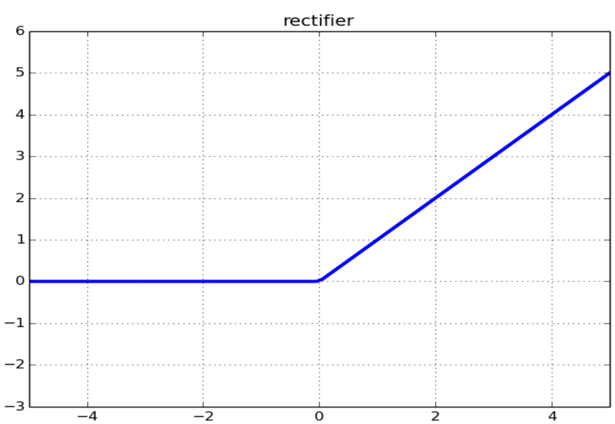


VGG-16

[Rhu et al., vDNN, MICRO 2016]

Problems with saturation





Issue

- A null gradient results in no learning, which happens if:
    - the sigmoid saturates, or
    - the ReLU saturates

Solution

- Initialize weighs so the average value is zero, i.e., work in the interesting zone of the activation functions
- Normalize data (zero mean)

Non-differential operations

Issue

- Discrete activation function / weights
  - extreme case is binary net
- Derivative not well defined

Solution

- Use approximate derivative, or
- Discretize a-posteriori

Model Overfitting

Problem:

- Neural net learns too specifically from input set, rather than generalizing from input, called overfitting
- Overfitting can be a result of too many parameters in model

Solution:

- Dropout – turn off neurons at random; other neurons will take care of their job.
  - + Reliability
  - - Redundancy (-> pruning)

Architecture Challenges for Training

- Floating point accuracy
- Where to store the gradients
- Synchronization for parallel processing