Multimedia Indexing and Retrieval

Deep Learning for multimedia indexing and retrieval

Georges Quénot

Multimedia Information Modeling and Retrieval Group

Laboratory of Informatics of Grenoble

Outline

- Machine learning
- Loss function
- Formal neuron
- Single layer perceptron
- Multilayer perceptron
- Back-propagation
- Learning rate
- Mini-batches
- Convolutional layers
- Pooling
- Softmax
- …

Supervised learning

- Target function: $f : X \rightarrow Y$

$$x \rightarrow y = f(x)$$

  - x : input object (typically vector)
  - y : desired output (continuous value or class label)
  - X : set of valid input objects
  - Y : set of possible output values

- Training data: $S = (x_i,y_i)_{(1 \leq i \leq I)}$
  - I : number of training samples

- Learning algorithm: $L : (X \times Y)^* \rightarrow Y^X$

$$S \rightarrow f = L(S)$$

- Regression or classification system: $y = f(x) = [L(S)](x) = g(S,x)$

$$((X \times Y)^* = \cup_{n \in N} (X \times Y)^n)$$

Single-label loss function

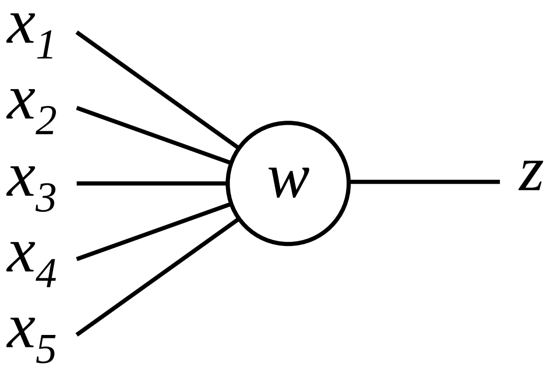
- Quantifies the cost of classification error or the empirical risk
- Example: $E_S(f) = \sum_{i=1}^{i=I} (f(x_i) - y_i)^2$
- The learning algorithm aims at minimizing the empirical risk
- If f depends on a parameter vector θ (e.g. $\theta = (w, b)$ for a linear SVM):

  $\theta^* = \underset{\theta}{\operatorname{argmax}}\ E_S(f_\theta)$

Multi-label loss function

- Predict P labels for each data sample x
- P decision functions: $f = (f_p)_{(1 \leq p \leq P)}$
- Example: $L_S(f) = \sum_{i=1}^{i=I} \sum_{p=1}^{p=P} (f_p(x_i) - y_{ip})^2$
- $\theta^* = \underset{\theta}{\operatorname{argmax}}\ E_S(f_\theta)$
- The f_p functions may take any real value

Formal neural or unit



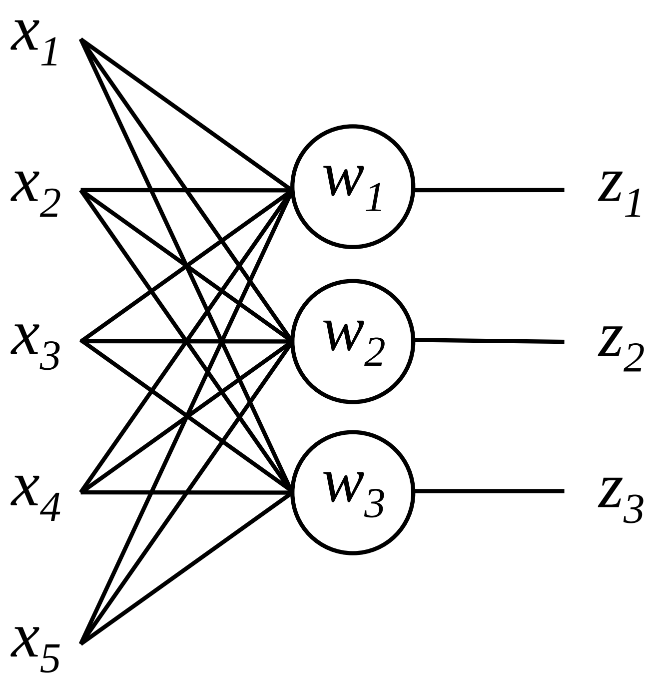
$$y = \sum_j w_j x_j$$

linear combination

$$z = \frac{1}{1 + e^y}$$

sigmoid function

Neural layer (all to all)



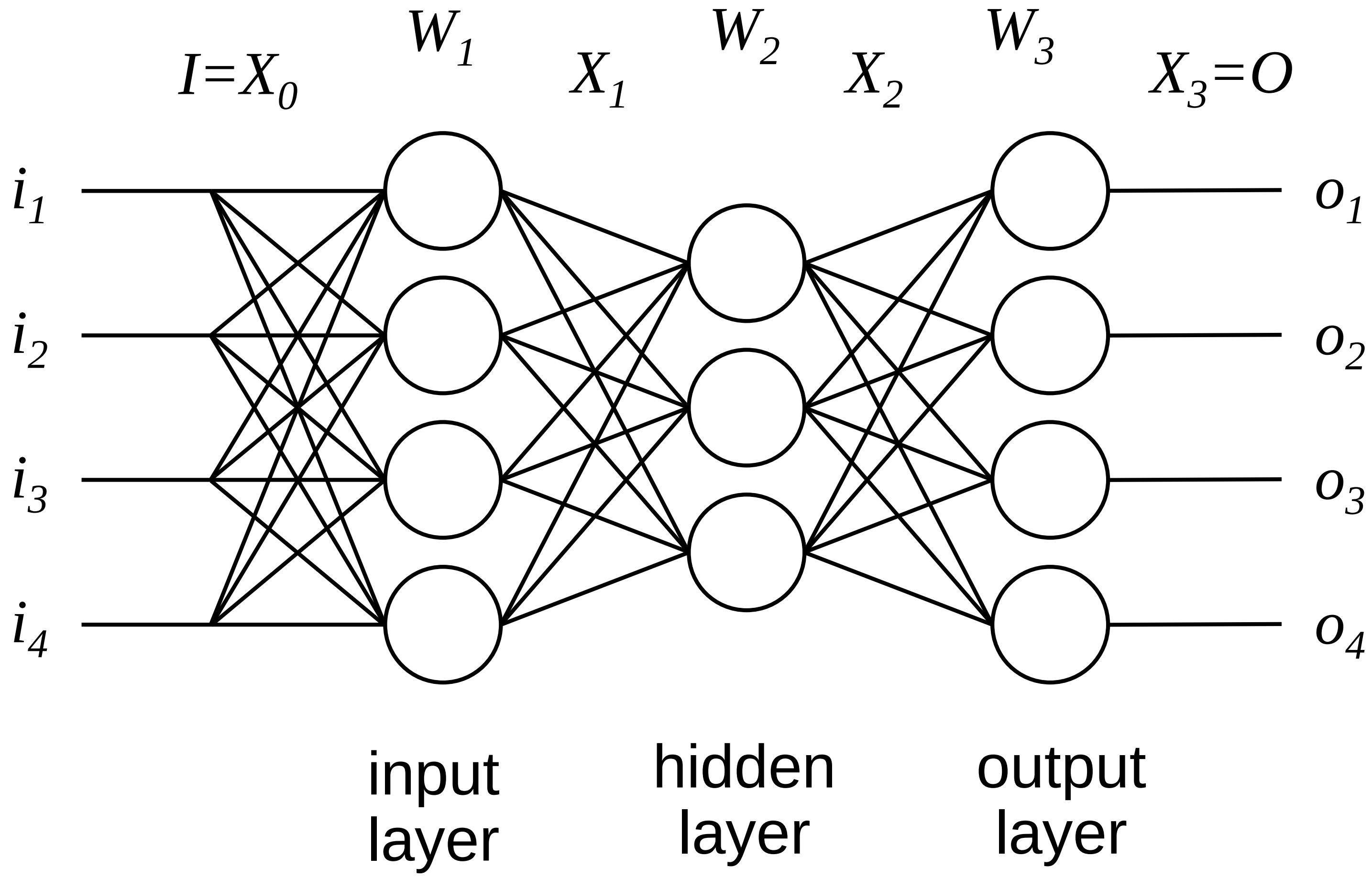
$$y_i = \sum_j w_{ij} x_j$$

matrix-vector multiplication

$$z_i = \frac{1}{1 + e^{y_i}}$$

per component operation

Multilayer perceptron (all to all)



Feed forward

- Global network definition: $O = F(W, I)$
  ($I \equiv x \quad O \equiv y \quad F \equiv f$ relative to previous notations)
- Layer values: $(X_0, X_1 \ldots X_N)$
  with $X_0 = I$ and $X_N = O$ (X_n are vectors)
- Vector of all unit parameters:
  $W = (W_1, W_2 \ \ldots \ W_N)$
  (weights by layer concatenated, W_n are matrices)
- Feed forward: $X_{n+1} = F_{n+1}(W_{n+1}, X_n)$

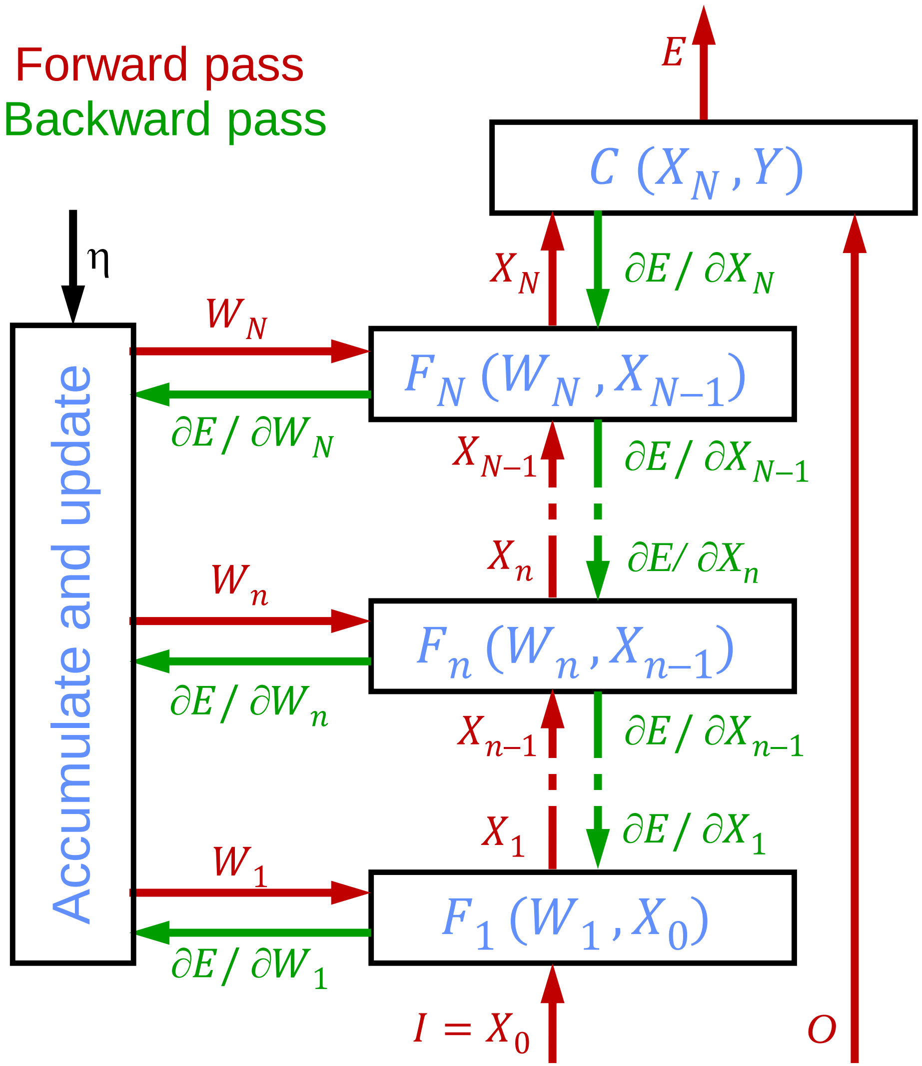
Error back-propagation

- Training set: $\left(I_p, O_p\right)_{(1 \leq p \leq P)}$ input-output samples
- $X_{p,0} = I_{p,0}$ and $X_{p,n+1} = F_{n+1}\left(W_{n+1}, X_{p,n}\right)$
- Note: regarding this notation the vector-matrix multiplication counts as one layer and the element-wise non-linearity counts as another one (not mandatory but greatly simplifies the layer modules' implementation)
- Error (empirical risk) on the training set:
  $E(W) = \sum_p \left(F\left(W, I_p\right) - O_p\right)^2 = \sum_p \left(X_{p,N} - O_p\right)^2$
- Minimization of $E(W)$ by gradient descent

Error back-propagation

- Minimization of $E(W)$ by gradient descent:
  - Randomly initialize $W(0)$
  - Iterate $W(t+1) = W(t) - \eta \frac{\partial E}{\partial W}(t)$ $\quad \eta = f(t)$ or $\eta = \left(\frac{\partial^2 E}{\partial W^2}(t)\right)^{-1}$
  - Back-propagation: $\frac{\partial E}{\partial W_n}$ is computed by backward recurrence from $\frac{\partial F_n}{\partial W_n}$ and $\frac{\partial F_n}{\partial X_{n-1}}$ applying iteratively $(g\ o\ f)' = (g'o\ f).f'$

Error back-propagation (adapted from Yann Le Cun)



Forward pass, for $1 \leq n \leq N$:

$X_n = F_n\,(W_n, Xn_{-1})$

Partial derivatives with respect to W_n. For $1 \leq n \leq N$:

$$\frac{\partial E}{\partial W_n} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(W_n, X_{n-1})}{\partial W_n}$$

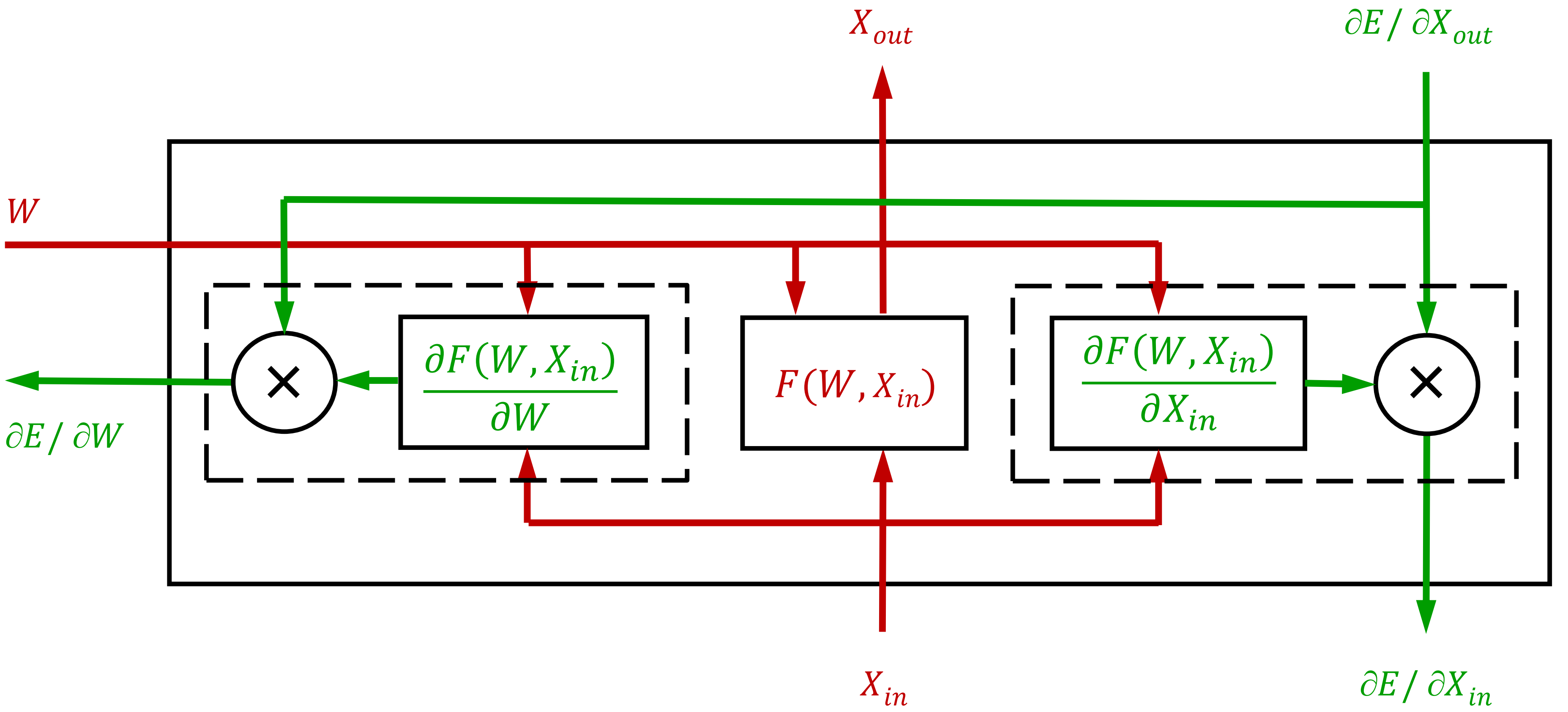
We need partial derivatives with respect to X_n. For N :

$$\frac{\partial E}{\partial X_N} = \frac{\partial C(X_N, O)}{\partial X_N}$$

Then backward recurrence:

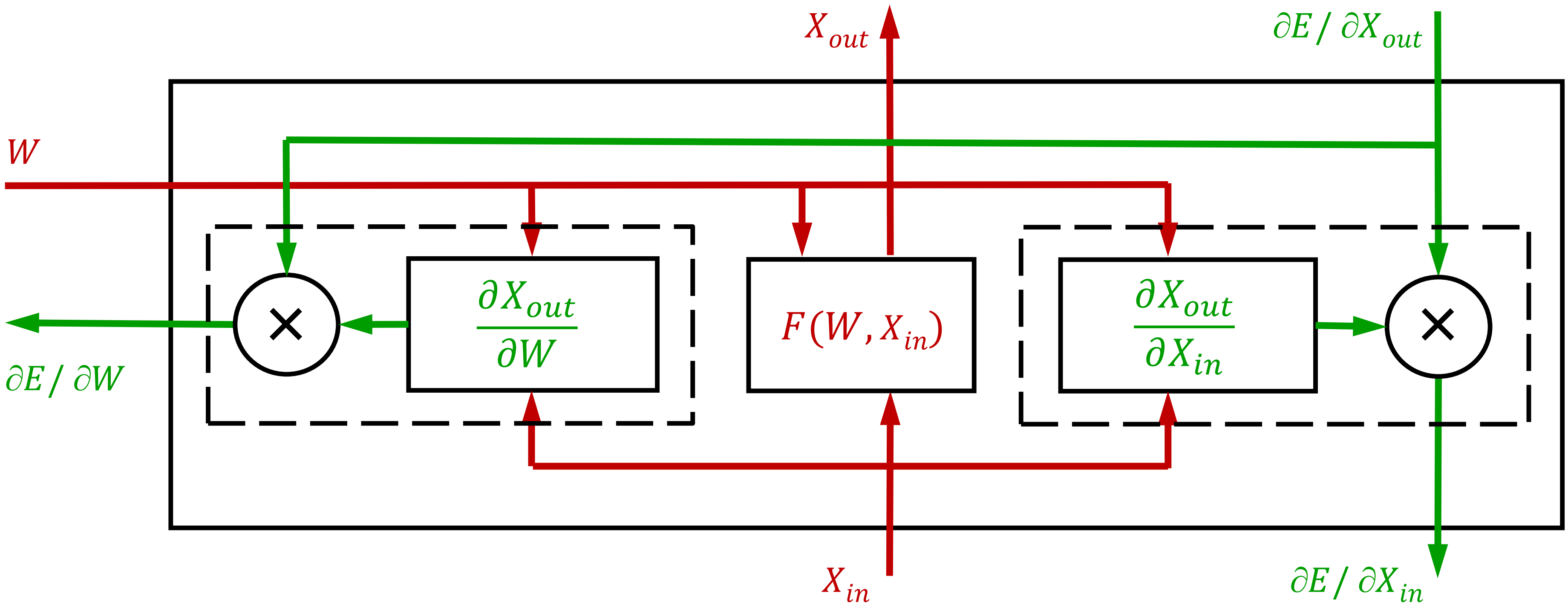
$$\frac{\partial E}{\partial X_{n-1}} = \frac{\partial E}{\partial X_n} \frac{\partial F_n(W_n, X_{n-1})}{\partial X_{n-1}}$$

Layer module (adapted from Yann Le Cun)



Notes: $X_{in} \equiv X_{n-1}$, $X_{out} \equiv X_n$, $W \equiv W_n$ and $F \equiv F_n$

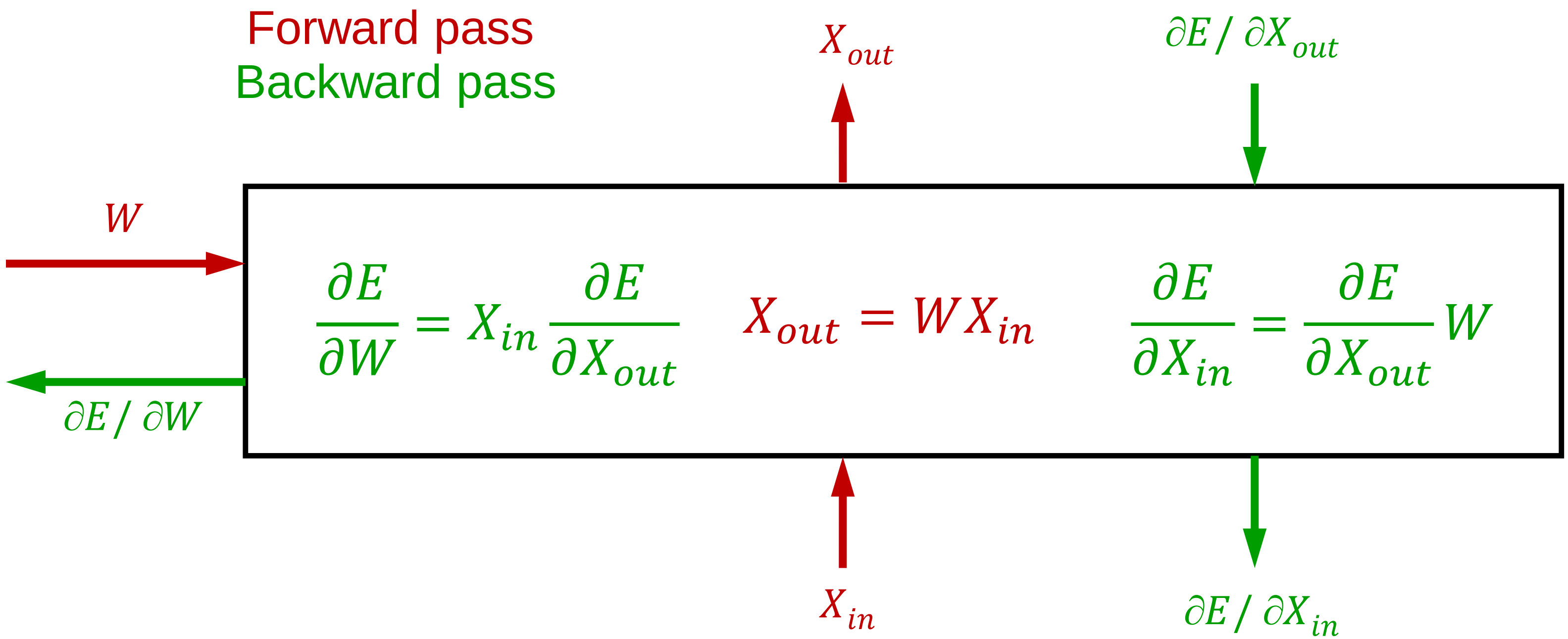
Layer module (adapted from Yann Le Cun)



$$\frac{\partial F(W, X_{in})}{\partial X_{in}} \equiv \frac{\partial X_{out}}{\partial X_{in}} \qquad \frac{\partial E}{\partial X_{in}} = \frac{\partial X_{out}}{\partial X_{in}} \frac{\partial E}{\partial X_{out}}$$

$$\frac{\partial F(W, X_{in})}{\partial W} \equiv \frac{\partial X_{out}}{\partial W} \qquad \frac{\partial E}{\partial W} = \frac{\partial X_{out}}{\partial W} \frac{\partial E}{\partial X_{out}}$$

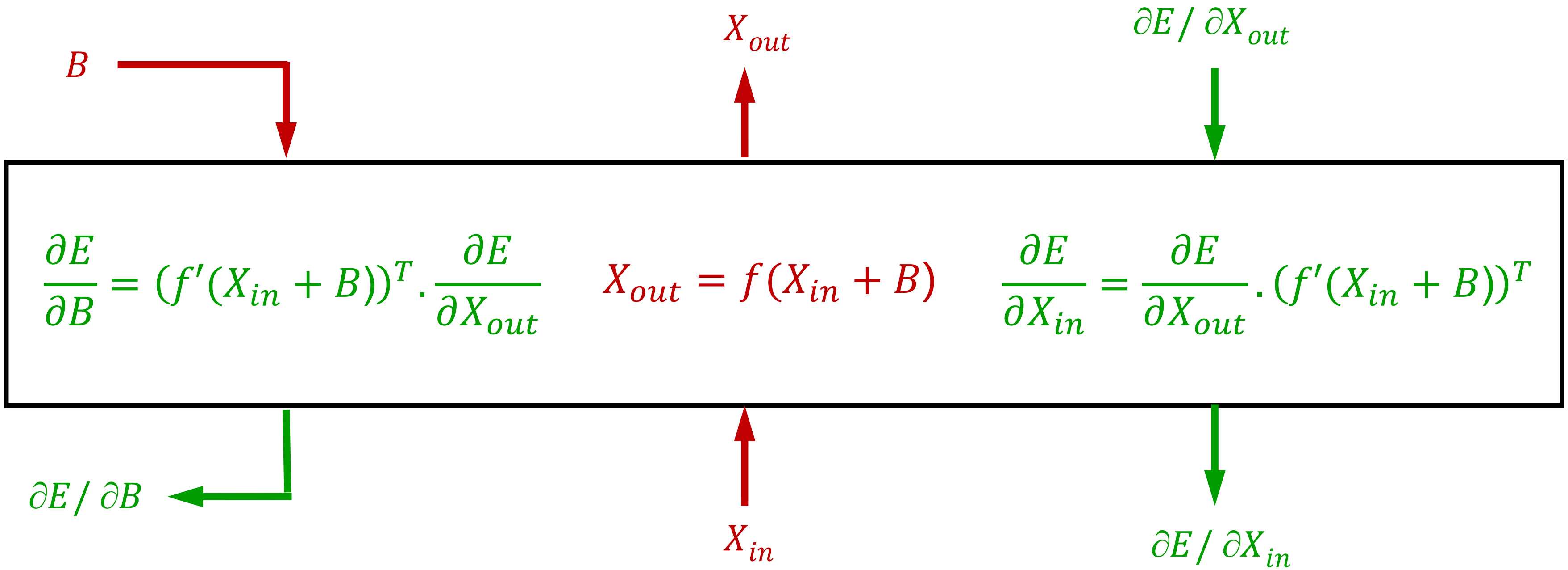
Linear module (adapted from Yann Le Cun)



Note: X_{in} and X_{out} are regular (column) vectors and W is a matrix while $\partial E / \partial X_{in}$ and $\partial E / \partial X_{out}$ are transpose (row) vectors and $\partial E / \partial W$ is a transpose matrix (for the vectors, this is because $\mathrm{d}E = (\partial E / \partial X).\mathrm{d}X$). $\partial E / \partial W$ is the outer product of the regular and transpose vectors X_{in} and $\partial E / \partial X_{out}$.

.

Pointwise module (adapted from Yann Le Cun)



Note: B is a bias vector on the input. X_{in}, X_{out} and B are regular (column) vectors while $\partial E / \partial X_{in}$ and $\partial E / \partial X_{out}$ and $\partial E / \partial B$ are transpose vectors. f is a scalar function applied pointwise on $X_{in} + B$. f' is the derivative of f and is also applied pointwise. The multiplication by $(f'(X_{in} + B))^T$ is also performed pointwise.

Convolutional layers

- Alternative to the “all to all” connections
- Preserves the image topology via “feature maps”
- Each layer is a “stack” of features maps
- Each map points is connected to the map points of a neighborhood in the previous layer
- Weights between maps are shared so that they are invariant by translation
- Resolution changes across layers: stride and pooling
- Example: AlexNet

Convolutional layers

Classical image convolution (2D to 2D):

$$O(i,j) = (I * K)(i,j) = \sum_{(m,n)} I(i-m, j-n) K(m,n)$$

Convolutional layer (3D to 3D):

$$O(i,j,l) = (I * K)(i,j,l) = B(l) + \sum_{k} \sum_{(m,n)} I(i-m, j-n, k) K(m,n,k,l)$$

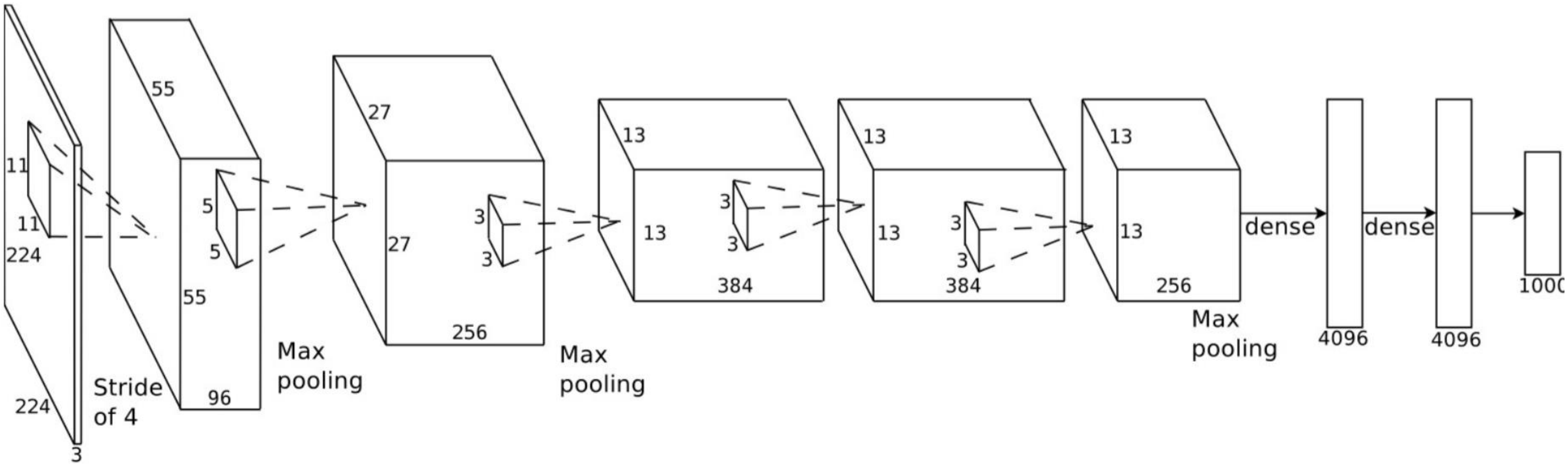
k and l: indices of the feature maps in the input and output layers

m and n: within a window around the current location, corresponding to the feature size

ImageNet Challenge 2012

[Krizhevsky et al., 2012]

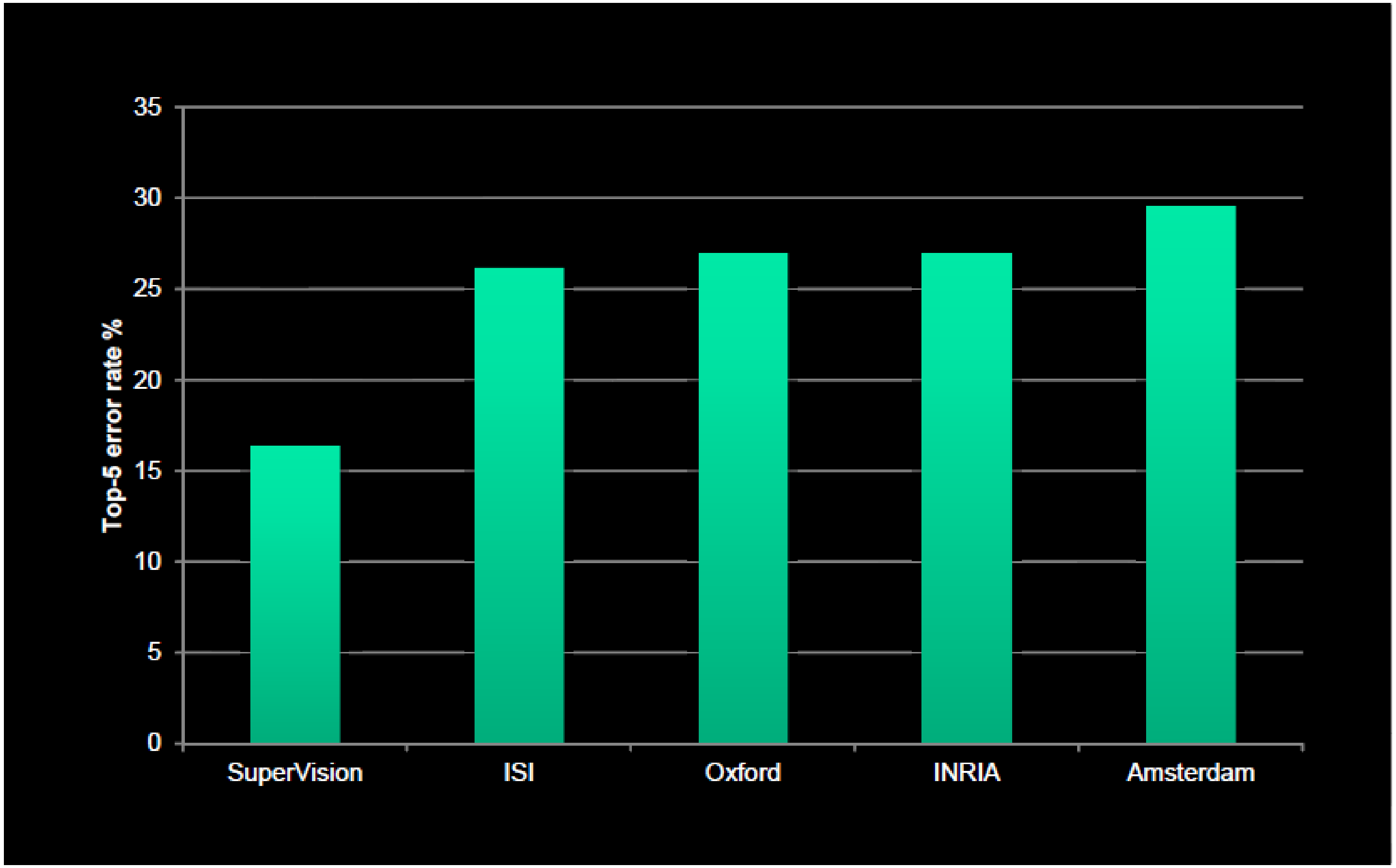
- 7 hidden layers, 650K units, 60M parameters (W)
- GPU implementation (50× speed-up over CPU)
- Trained on two GPUs for a week



A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

ImageNet Classification 2012 Results

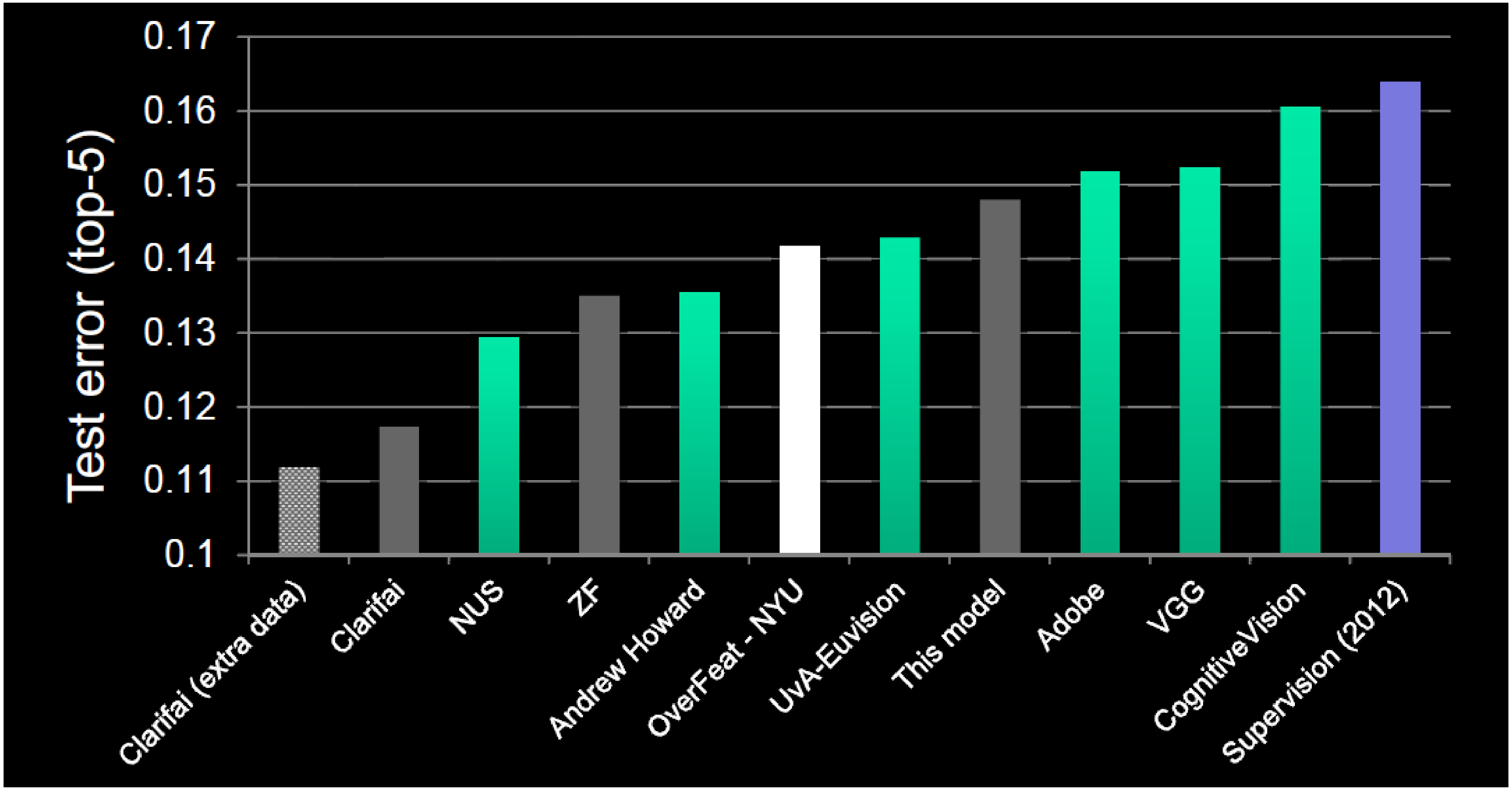
Krizhevsky et al. -- **16.4% error** (top-5)
Next best (non-convnet) – **26.2% error**



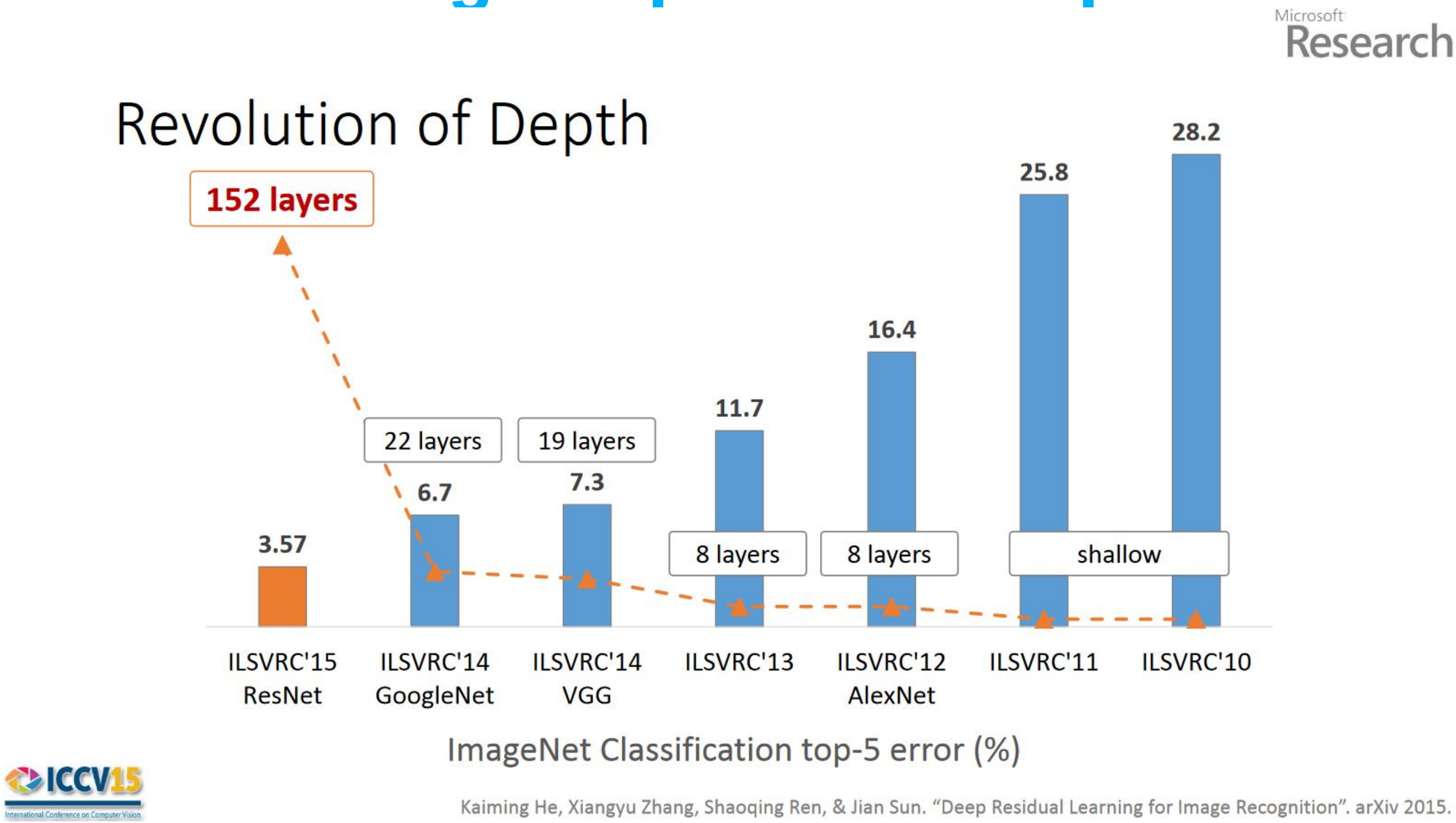
ImageNet Classification 2013 Results

http://www.image-net.org/challenges/LSVRC/2013/results.php

Demo: http://www.clarifai.com/



Going deeper and deeper



For comparison, human performance is 5.1% (Russakovsky et al.)

Engineered versus learned descriptors

- Classical "classification pipeline"
  - Extraction(s) – [aggregation] – optimization(s) – classifier(s) – one or more levels of fusion – re-scoring (non exhaustive example)
  - Most of the stages are explicitly engineered: the form of descriptors or processing steps has been thought and designed by a skilled engineer or researcher
  - *Lots* of experience and acquired expertise by thousands of smart people over tens of years
  - Learning concerns only the classifier(s) stages and a few hyper-parameters controlling the other ones
  - Almost everything has been tried
  - The more you incorporate, the more you get (at a cost)

Engineered versus learned descriptors

- Deep learning pipeline: MLP with about 8 layers
  - Advances in computing power (Tflops): large networks possible
  - Algorithmic advance: combination of convolutional layers for the lower stages with all-to-all layers; the topology of the image is preserved in the lower layers with weights shared between the units within a layer
  - Algorithmic advances: NN researchers finally find out how to have back-propagation working for MLP with more than three layers
  - Image pixels are entered *directly* into the first layer
  - The first (resp. intermediate, last) layers practically compute low-level (resp. intermediate level, semantic) descriptors
  - Everything is made using a unique and homogeneous architecture
  - A single network can be used for detecting many target concepts
  - All the level are jointly optimized at once
  - Requires *huge* amounts of training data