Multilayer Perceptron and Backpropagation Learning

Jenq-Neng Hwang, Professor
Department of Electrical & Computer Engineering
University of Washington, Seattle WA
hwang@uw.edu

EEP 596B: Deep Learning for Big Visual Data, Fall 2021

Machine Learning Performance Metrics

Cross Validation

- Hold-out cross-validation (early stopping): split the dataset T into three mutually disjoint subsets – training, validation, and testing.
  - The model is trained on the training subset, while the validation subset is periodically used to evaluate the model performance during the training to avoid over-training. The training is stopped, when the performance on validation subset is good enough or when it stops improving.
  - When comparing $m > 1$ computational models $L_1, \cdots, L_m$ against each other, the testing subset is used to evaluate the models' performance.
- K-Fold cross-validation: Divide T into k parts of the same size. One part forms the validation (testing) set, the other parts form the training set. This process is repeated for each validation part of the data.

Selected Relevant	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive(FP)	True Negative (TN)

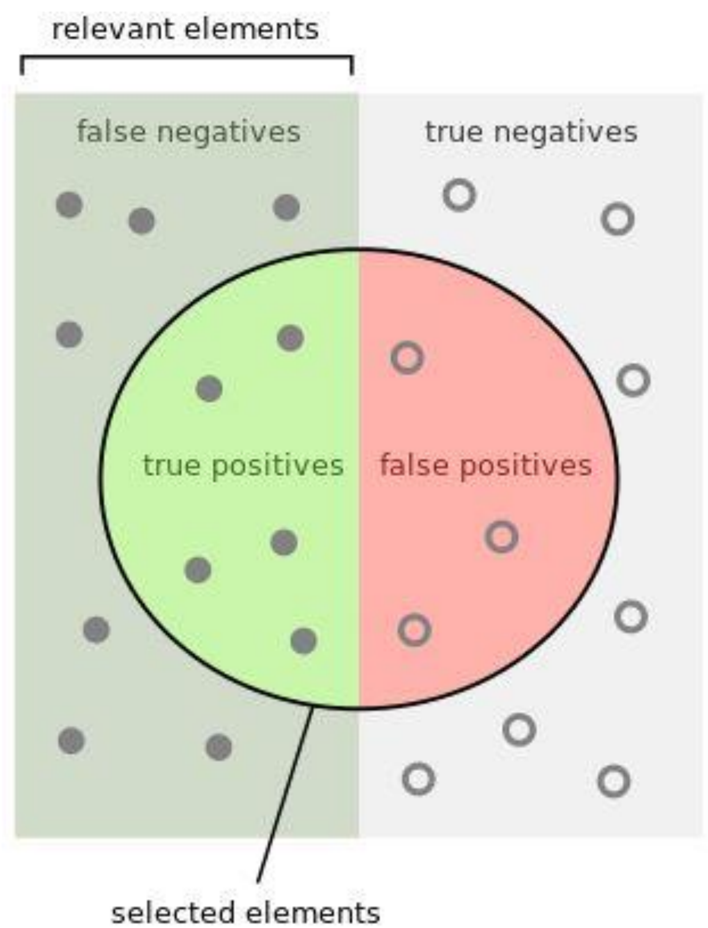
Performance Metrics

- **Accuracy (ACC)**

ACC=(TP+TN)/(TP+FP+FN+TN)

outcome

prediction



What is Wrong with Accuracy Alone?

- A 2-category classifier (imbalanced):

accuracy = (10 + 100)/(10 + 5 + 15 + 100) = **84.6%**

- A dumb "negative" classifier: (when TP < FP)

accuracy = (0 + 115)/(0 + 15 + 0 + 115) = **88.5%**.

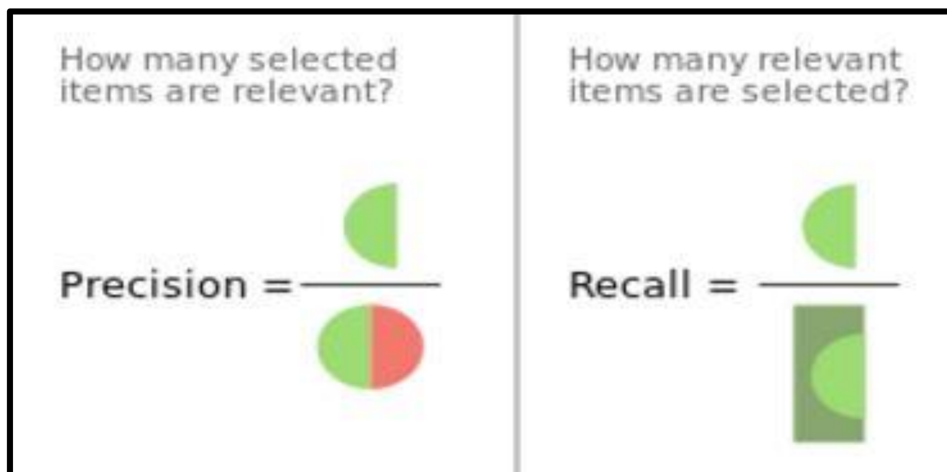
	Classified positive	Classified negative
Positive class	10	5
Negative class	15	100

	Classified positive	Classified negative
Positive class	0	15
Negative class	0	115

Precision and Recall

- Precision: positive predictive value, the correct fraction out of all the examples the classifier predicted as positive
  Pre = TP / (TP + FP) (no cancer → predict cancer)
- Recall: true positive rate, also called sensitivity, hit rate, the correct fraction out of all the positive examples
  Rec = TP / (TP + FN)

(with cancer → predict no cancer)

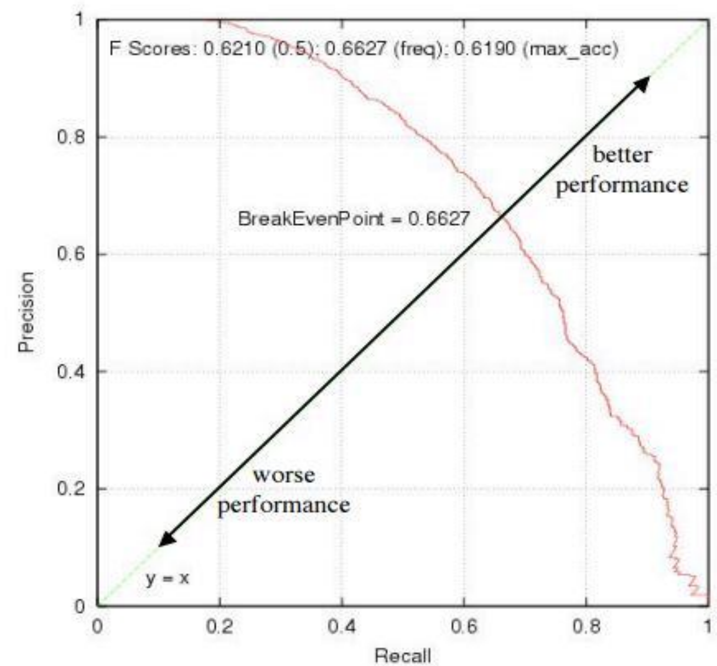


Meaning of A True Positive

- For classification tasks:
  - Threshold of confidence score (e.g., likelihood)
- For detection/segmentation tasks
  - Whether a correct object exists in the image (classification)
  - The location of the object (location, width, height, etc, a regression task).

Pre & Rec Tradeoffs

- A perfect recall (zero FN simply label all the examples as positive) → horrible precision
- Increase precision (low FP, only label the most certain examples as positive) → horrible recall
- Need to optimize a measure that combines precision and recall into a single value, such as the F1 Score.



F1 Score (F1 Measure)

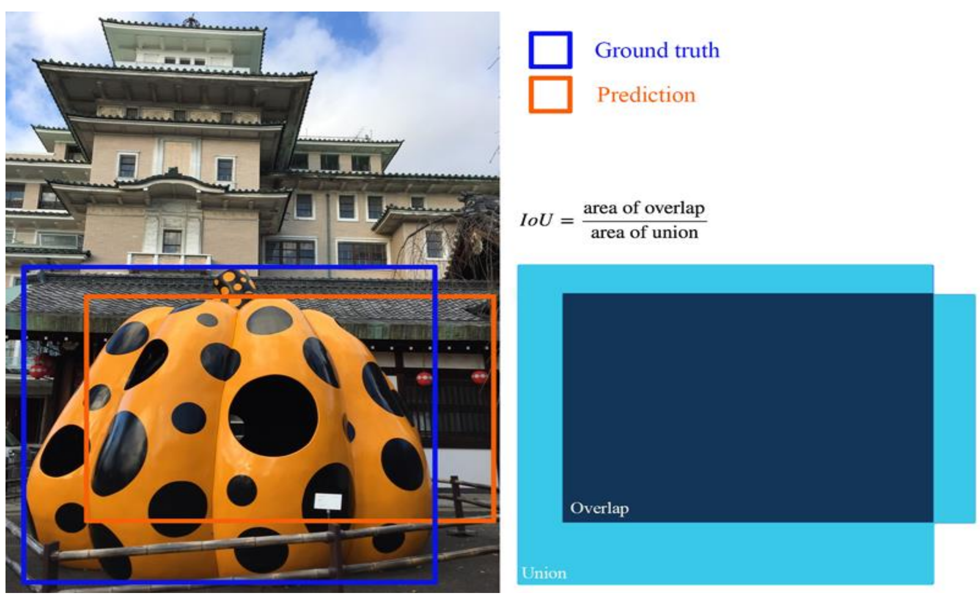
- F1 score (balanced F-score) can be interpreted as a weighted average (harmonic mean) of the precision and recall, where an F1 score reaches its best value at 1 and worst at 0.

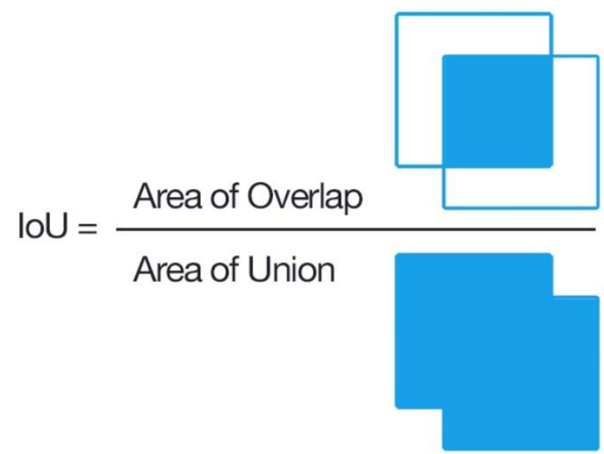
$$F1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2TP}{2TP + FP + FN}$$

- This measure is approximately the average of the two when they are close, and is more generally the square of the geometric mean divided by the arithmetic mean.

Intersection over Union (IoU) for Location TP/FP

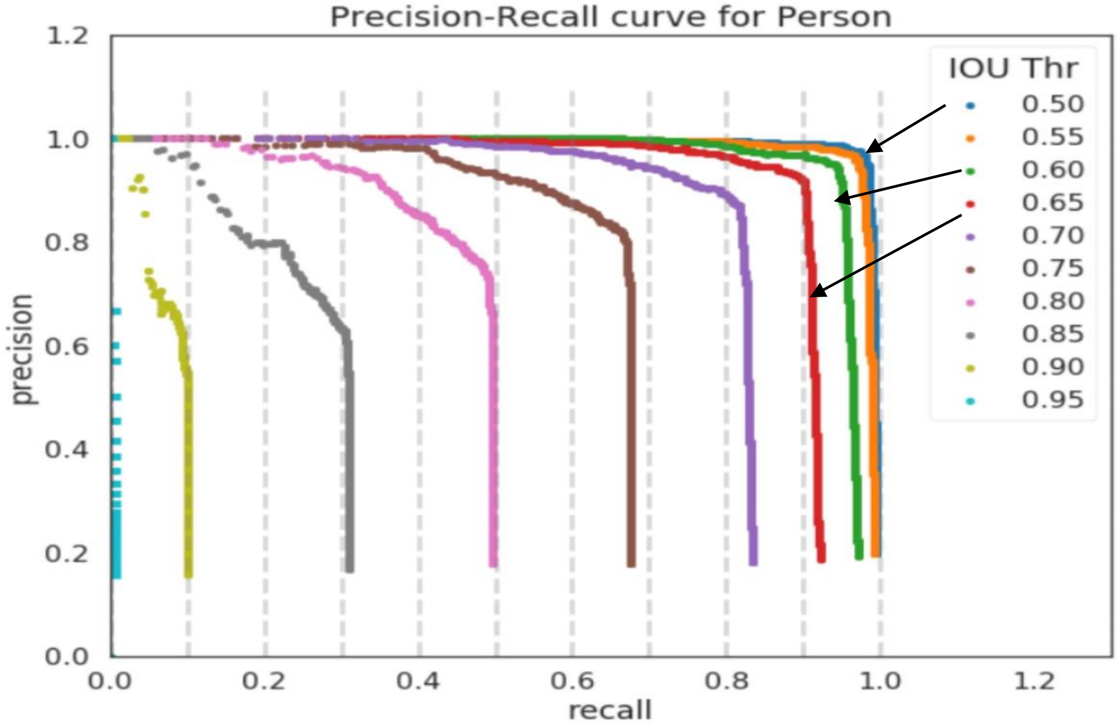
- The overlap between two (detection/segmentation) boundaries, mainly for 2D/3D data.
- Predefine an IoU threshold (say 0.5) in classifying whether the prediction is a true positive or a false positive.





Definition of Average Precision (AP)

- The areas under the precision-recall curves (of a specific IoU threshold) or average over all IoUs

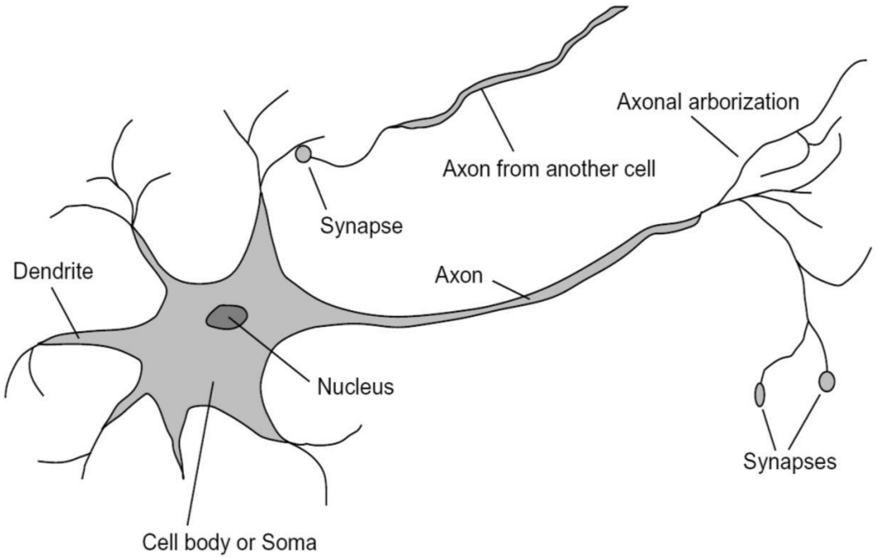


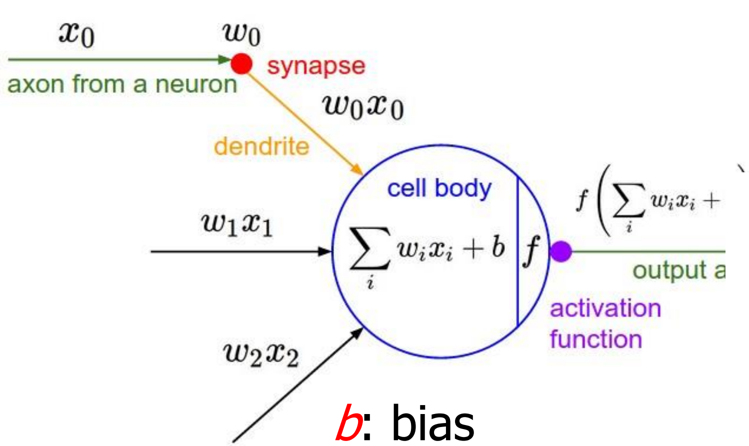
AP and mAP Definitions in COCO Competition

- For COCO, AP is the average over multiple IoUs (the minimum IoU to consider a positive match) of one class or over multiple categories.
- **AP@[.5:.95]** corresponds to the average AP for IoU from 0.5 to 0.95 with a step size of 0.05. For the COCO competition, AP is the average over 10 IoU levels on 80 categories (AP@[.50:.05:.95]: start from 0.5 to 0.95 with a step size of 0.05)
- *mAP is averaged over all categories (same as AP in COCO).*

Biological & Artificial Neurons

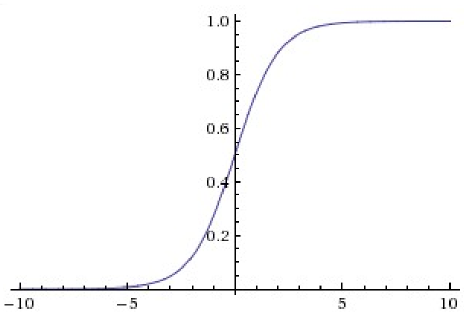
- 10^{11} neurons in our brain and 10^3 synapses per neuron.
- Neuron is a complex electrochemical device with membrane potential.





b: bias

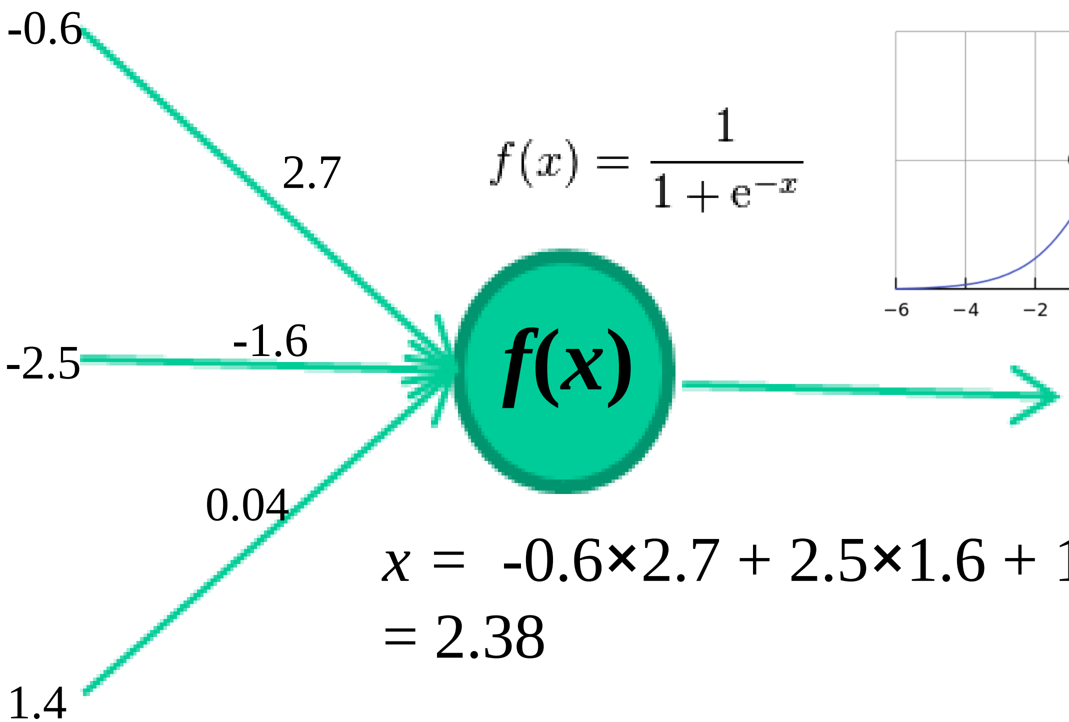
voltage x conductance = current



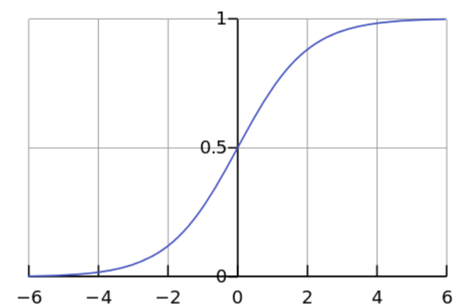
$$f(\mathrm{u}) = \frac{1}{1+e^{-u}} \, (sigmoid)$$

$$\frac{df(u)}{du} = f' = f \times (1-f)$$

A Simple Example



$$f(x) = \frac{1}{1+e^{-x}}$$



$x =$ -0.6×2.7 + 2.5×1.6 + 1.4×0.04+b

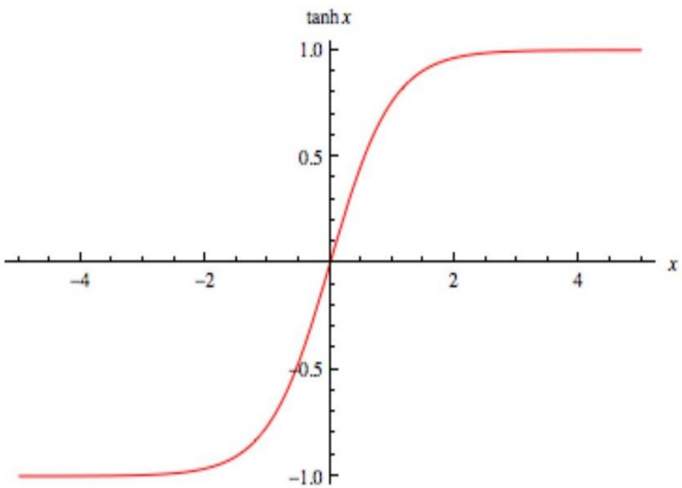
= 2.38

f(2.38)=0.916

Other Nonlinear Activation Functions

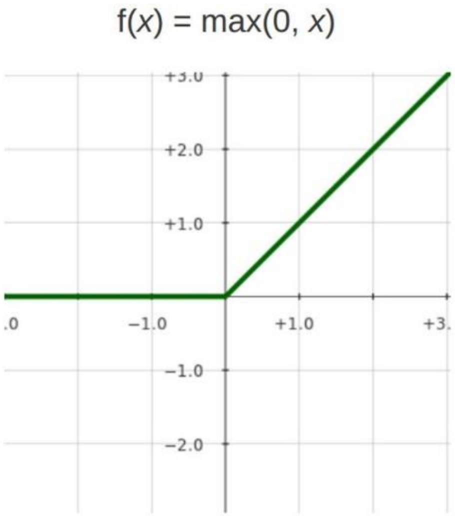
hyperbolic tangent

$$tanh(z) = \frac{exp(z) - exp(-z)}{exp(z) + exp(-z)}$$

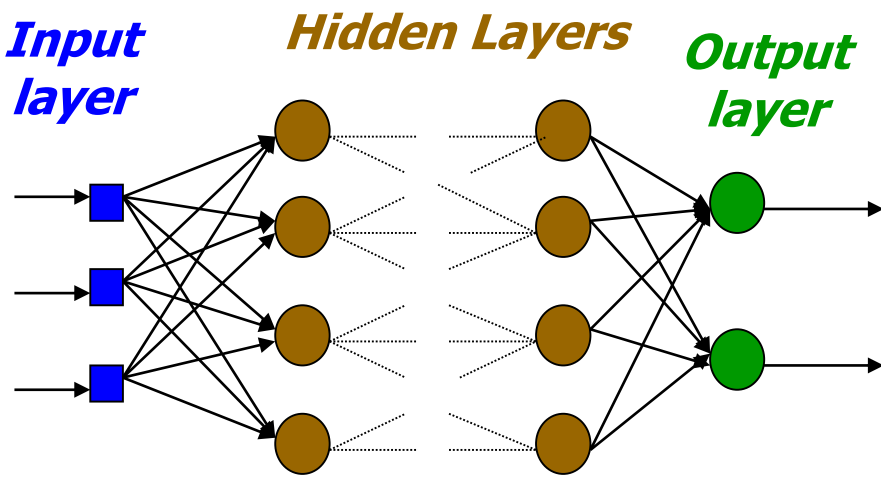


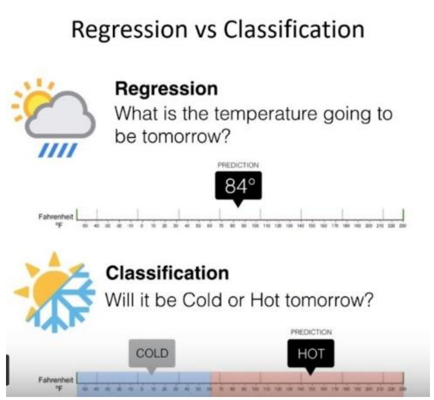
rectified linear unit

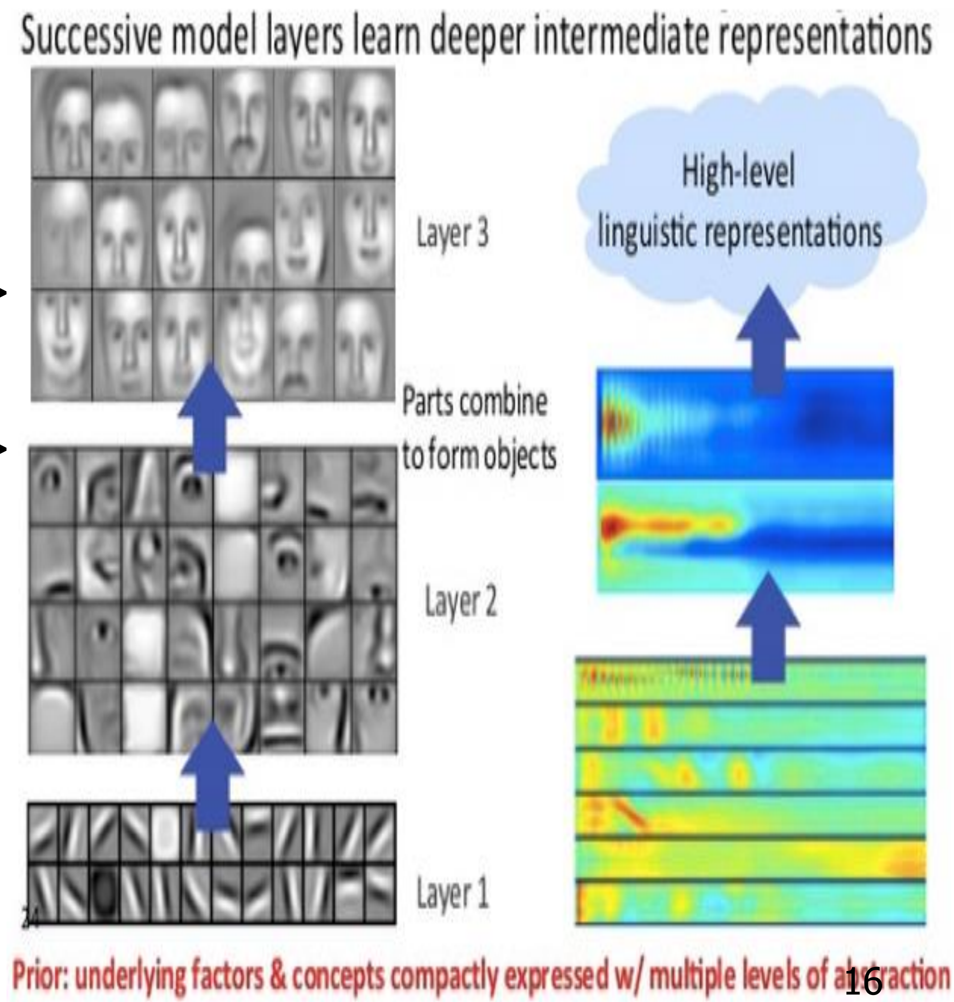
$$relu(z) = \max(0, z)$$



A Multilayer Perceptron (MLP) Neural Network





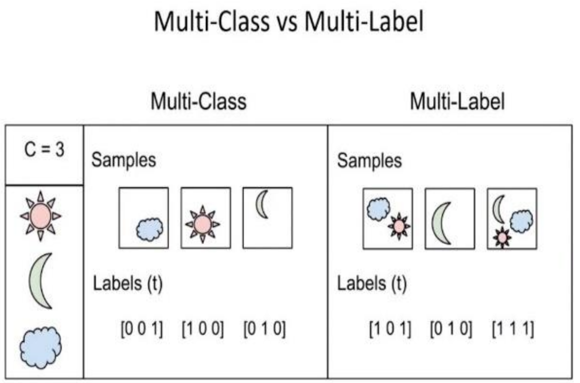


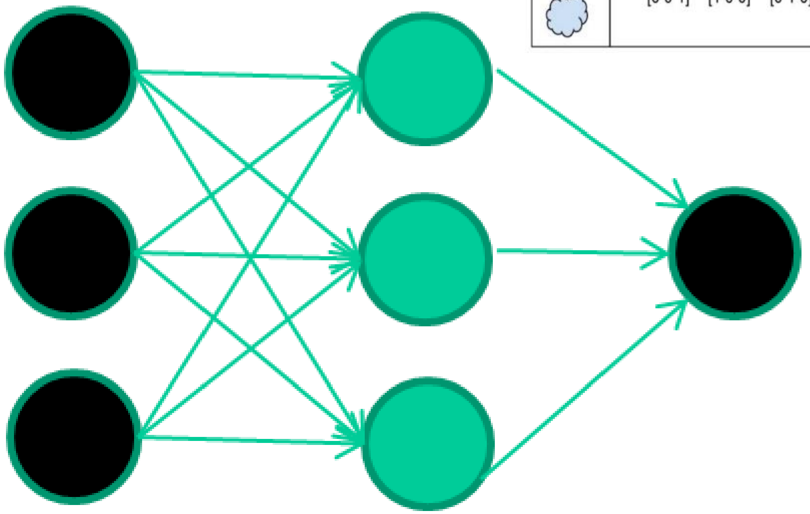
Neural Network Training

A dataset

inputs			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	1
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …



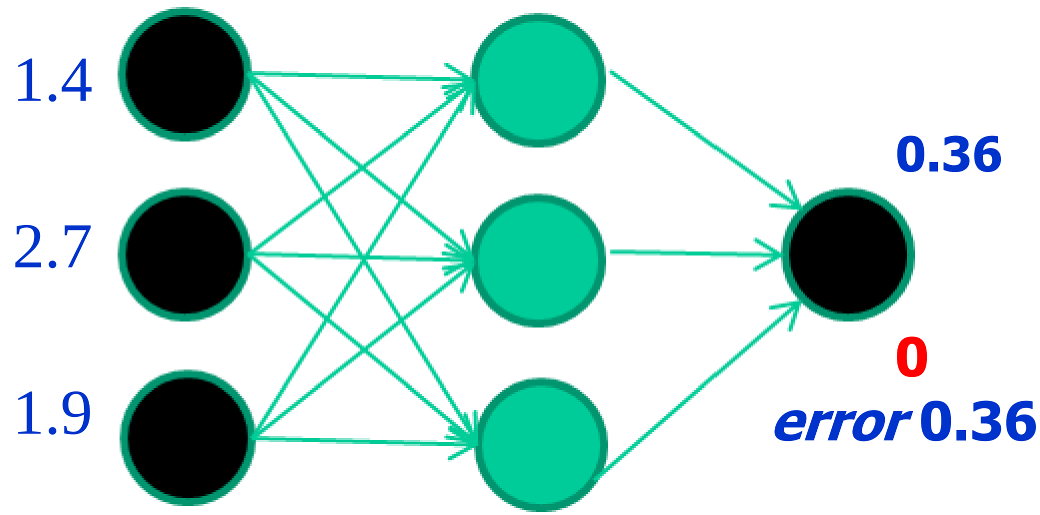


Initialize with random weights

Neural Network Training

Training data

inputs			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	1
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			



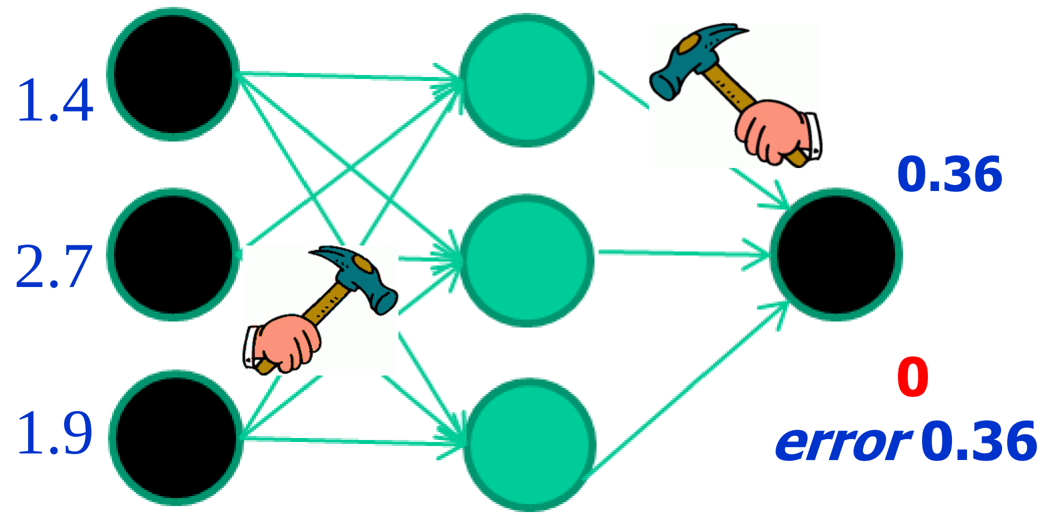
D. Rumelhart, et al., "Learning Internal Representations by Error Propagation“, *Chap. 8 in Parallel Distributed Processing : Explorations in the Microstructure of Cognition*. Volume 1 : Foundations. Cambridge: MIT Press. 1986

- **Present a training pattern**
- **Feed it through to get output**
- **Compare with target output**

Neural Network Training

Training data

inputs			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	1
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			



$$w_{ij}(l) \leftarrow w_{ij}(l) - \eta \frac{\partial E}{\partial w_{ij}(l)}$$

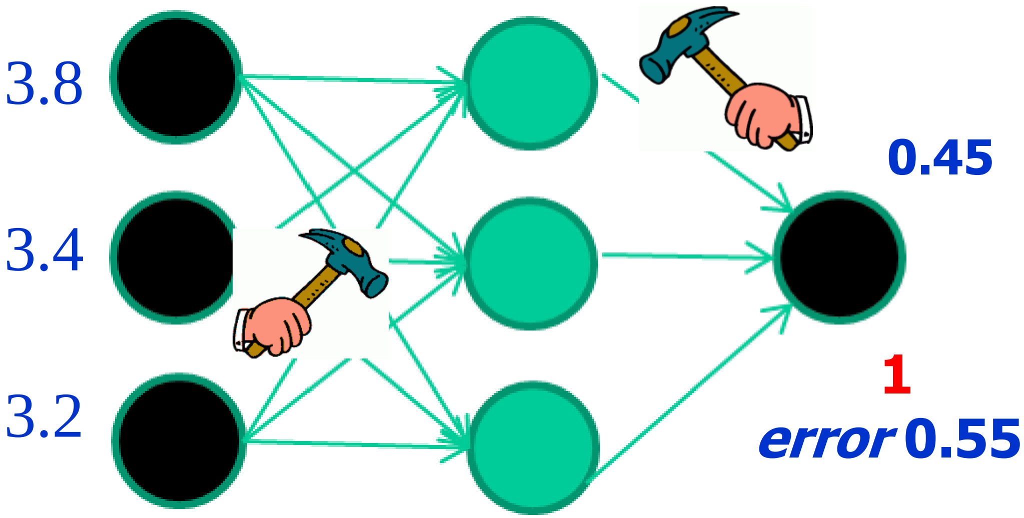
$$E = 1/2 \sum_{j=1}^{N_L} (t_i - a_j(L))^2$$

- **Adjust weights based on error**
- **Minimize the squared error or cross entropy**
- **A gradient descent search is used**

Neural Network Training

Training data

inputs			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	1
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

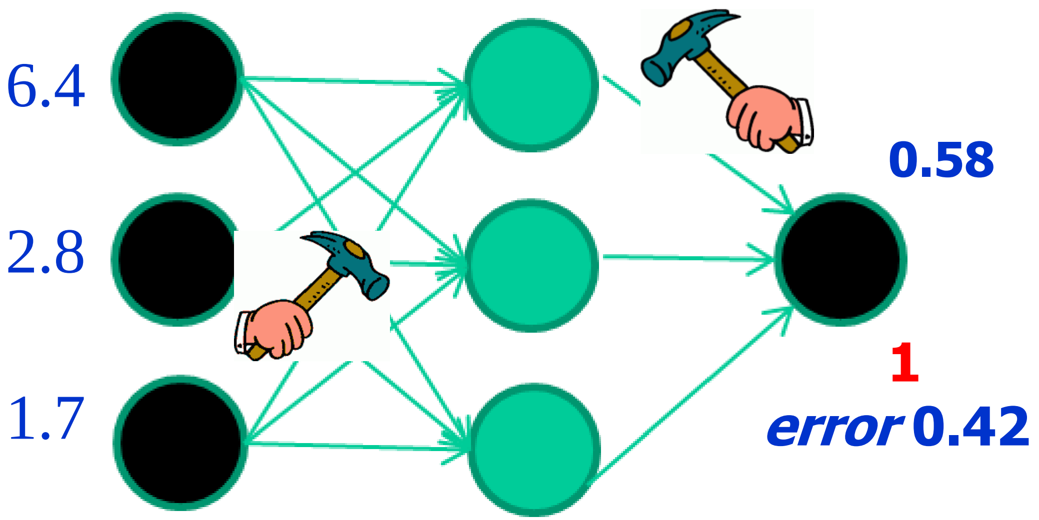


- **Present another training pattern**
- **Feed it through to get output**
- **Compare with target output**
- **Update the weights again**

Neural Network Training

Training data

inputs			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	1
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			



- **Continue to present training patterns one-by-one randomly and update weights**
- **Can last thousands or millions iterations**

A Very Nonlinear Model with Single-Stage ML

conventional linear approacher (MA, AR, ARMA)

e.g., $\mathrm{y(k)} \approx \mathrm{a}_0 x(k) + \mathrm{a}_1 x(k-1) + \cdots + \mathrm{a}_\mathrm{p} x(k-p)$

or $\mathrm{y(k)} \approx \mathrm{a}_0 x(k) + \cdots + \mathrm{a}_\mathrm{p} x(k-p) + b_1 y(k-1) + \cdots + b_\mathrm{p} y(k-q)$

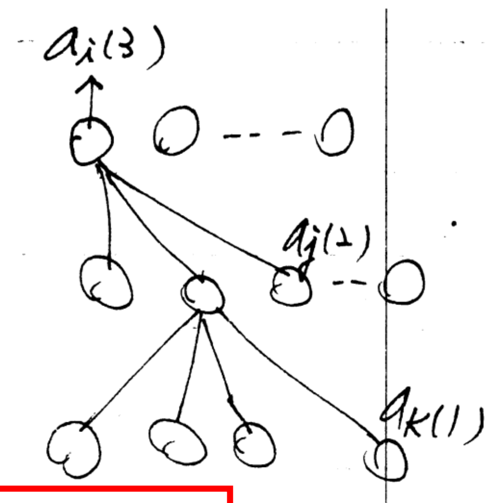
conventional nonlinear regression

e.g., (polynomial, rational polynomial)

$$\mathrm{y(k)} = \mathrm{a}_0 x(k) + \mathrm{a}_1 x^2(k) + \mathrm{a}_2 x(k) x(k-1) + \cdots$$

$$or\ \mathrm{y(k)} = \frac{\mathrm{a}_0 x^2(k) + \mathrm{a}_1 x^3(k-1)}{\mathrm{a}_2 x(k-2) x(k) + 3}$$

How about BP network?

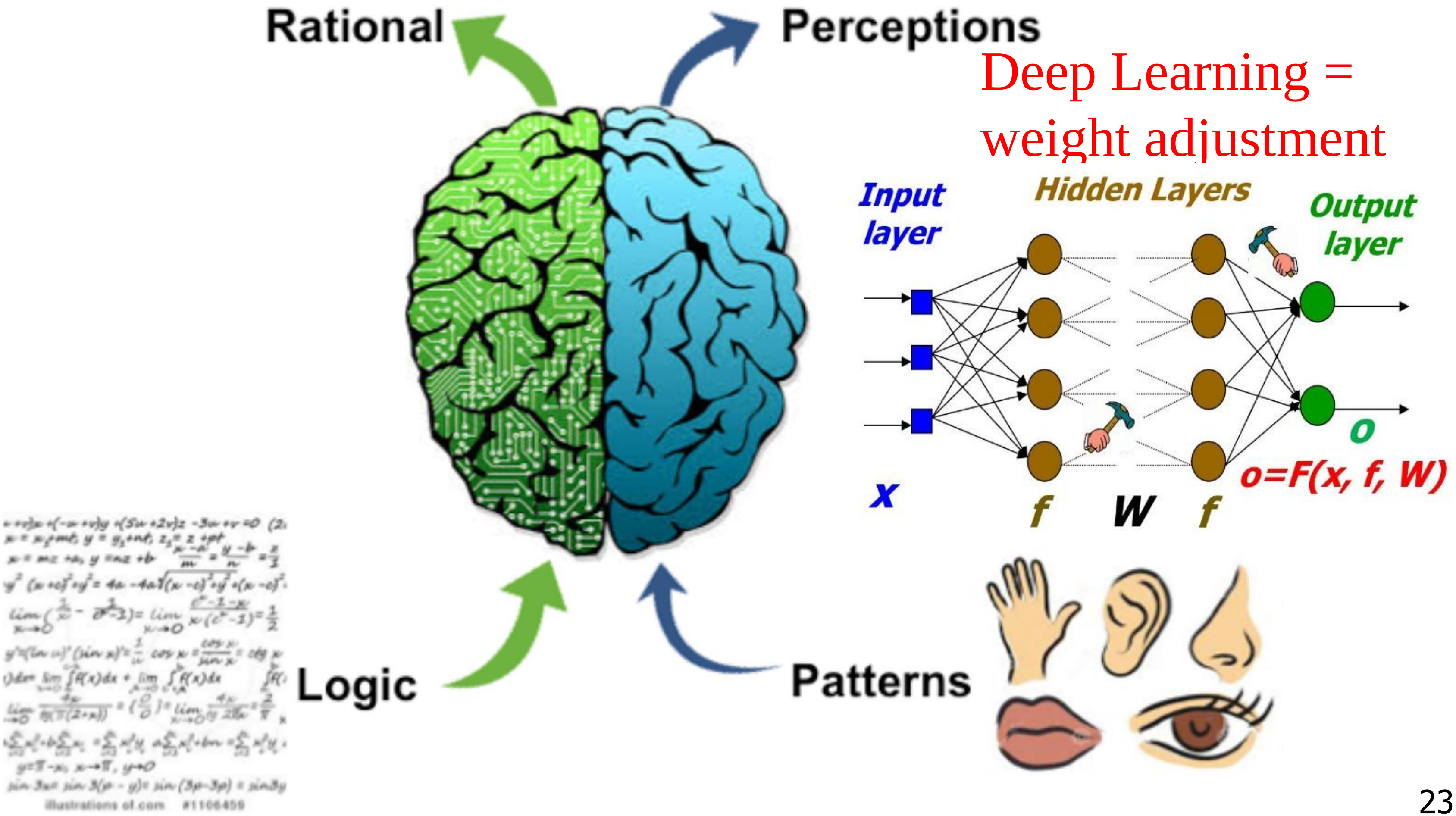


$$\mathrm{a_i}(3) = \frac{1}{1+e^{-u_i(3)}} = \frac{1}{1+e^{-(\sum_j w_{ij}(3) a_j(2) + \theta_i(3))}}$$

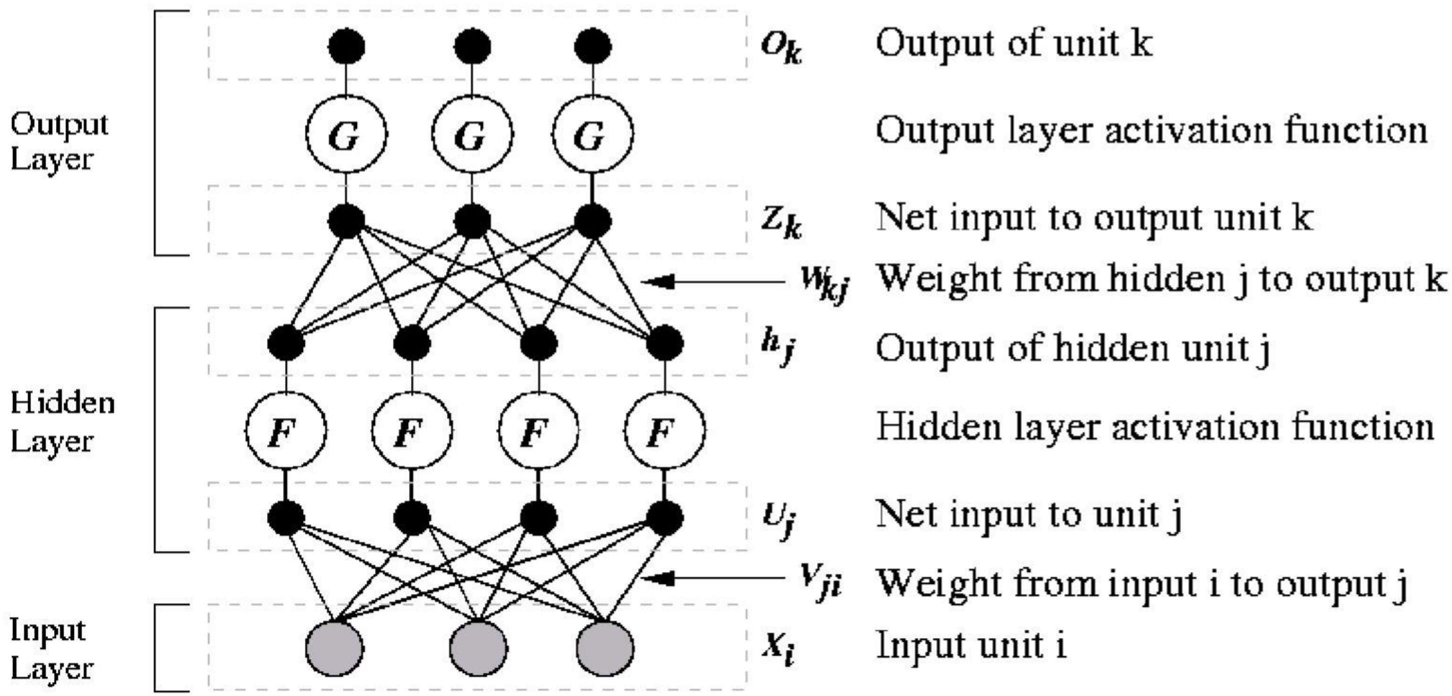
$$= \frac{1}{1+e^{-(\sum_j w_{ij}(3) \frac{1}{1+e^{-u_j(2)}} + \theta_i(3))}} = \frac{1}{1+e^{-(\sum_j w_{ij}(3) \frac{1}{1+e^{-(\sum_j w_{jk}(2) a_k(1) + \theta_j(2))}} + \theta_i(3))}}$$

Deep Learning based AI

Deep Learning =
weight adjustment



BackPropagation (BP) Learning for An MLP



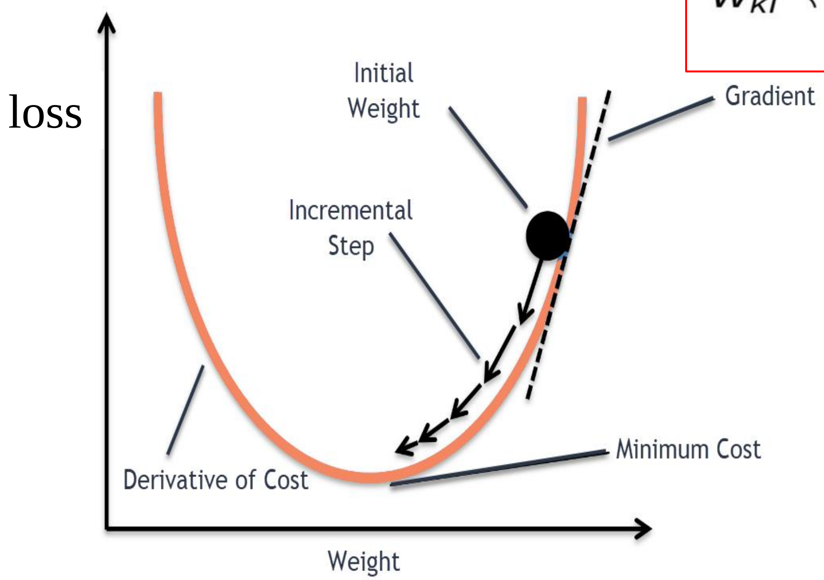
mean squared error (MSE) $E = \frac{1}{2}\sum_k (o_k - t_k)^2;$ $\qquad \frac{\partial E}{\partial o_k} = o_k - t_k$

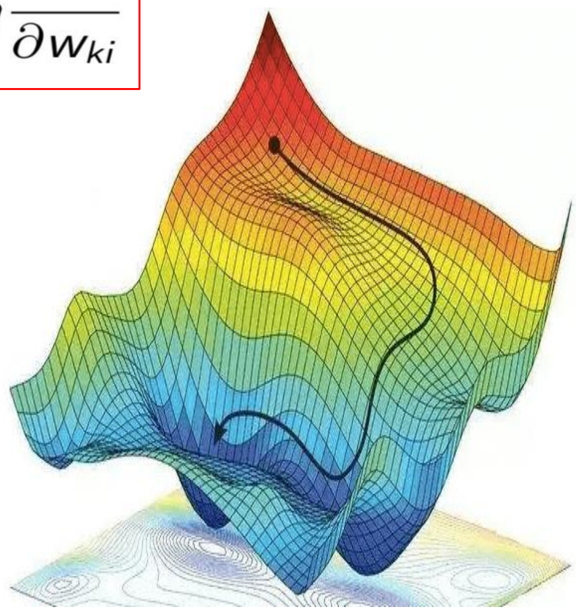
Stochastic gradient descent $w_{ki} \leftarrow w_{ki} - \eta \frac{\partial E}{\partial w_{ki}}$ $\qquad \boxed{\eta \text{ :learning rate}}$

David Rumelhart

Stochastic Gradient Descent (SGD) Search

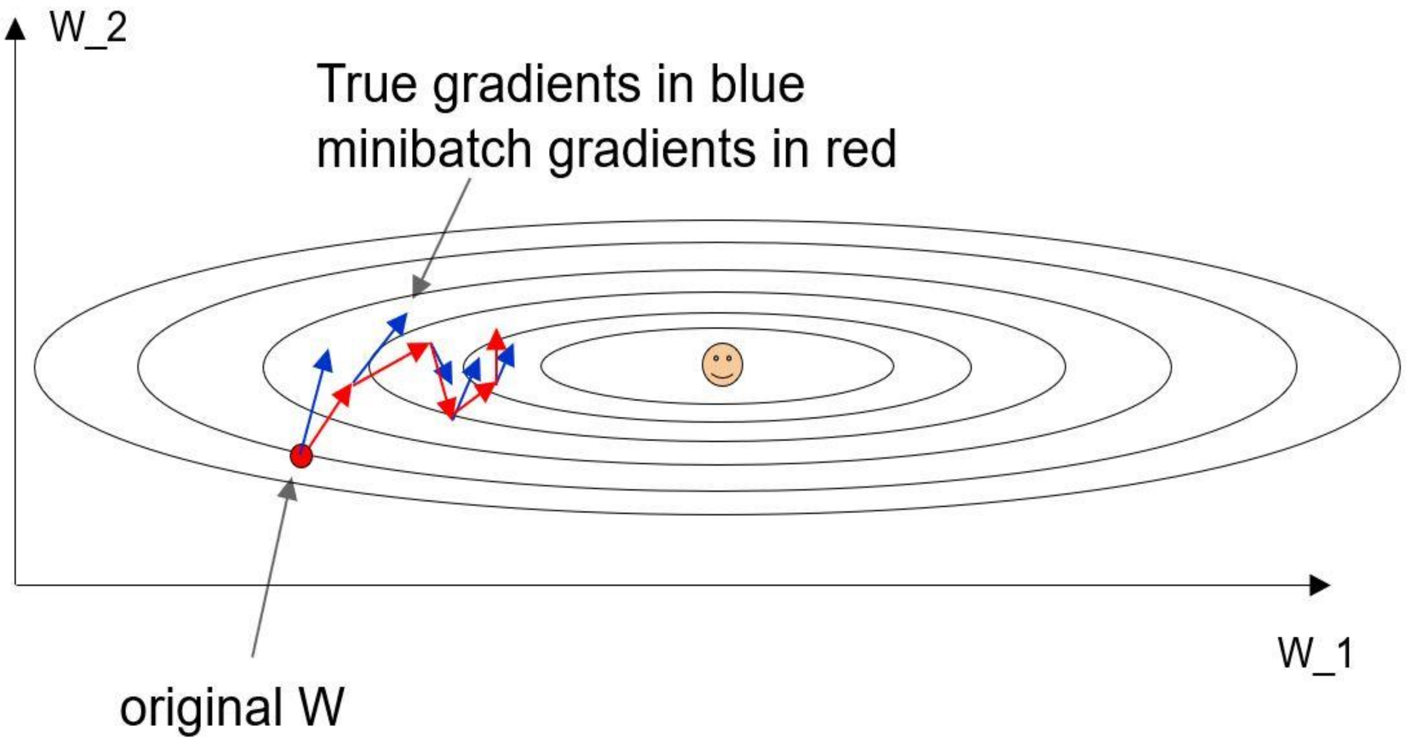
$$w_{ki} \leftarrow w_{ki} - \eta \frac{\partial E}{\partial w_{ki}}$$





Stochastic gradient descent can be regarded as a stochastic approximation of gradient descent optimization, since it replaces the actual gradient (calculated from the entire data set) by an estimate thereof (calculated from a randomly selected subset of the data, mini-batch)

Mini Batch Gradients



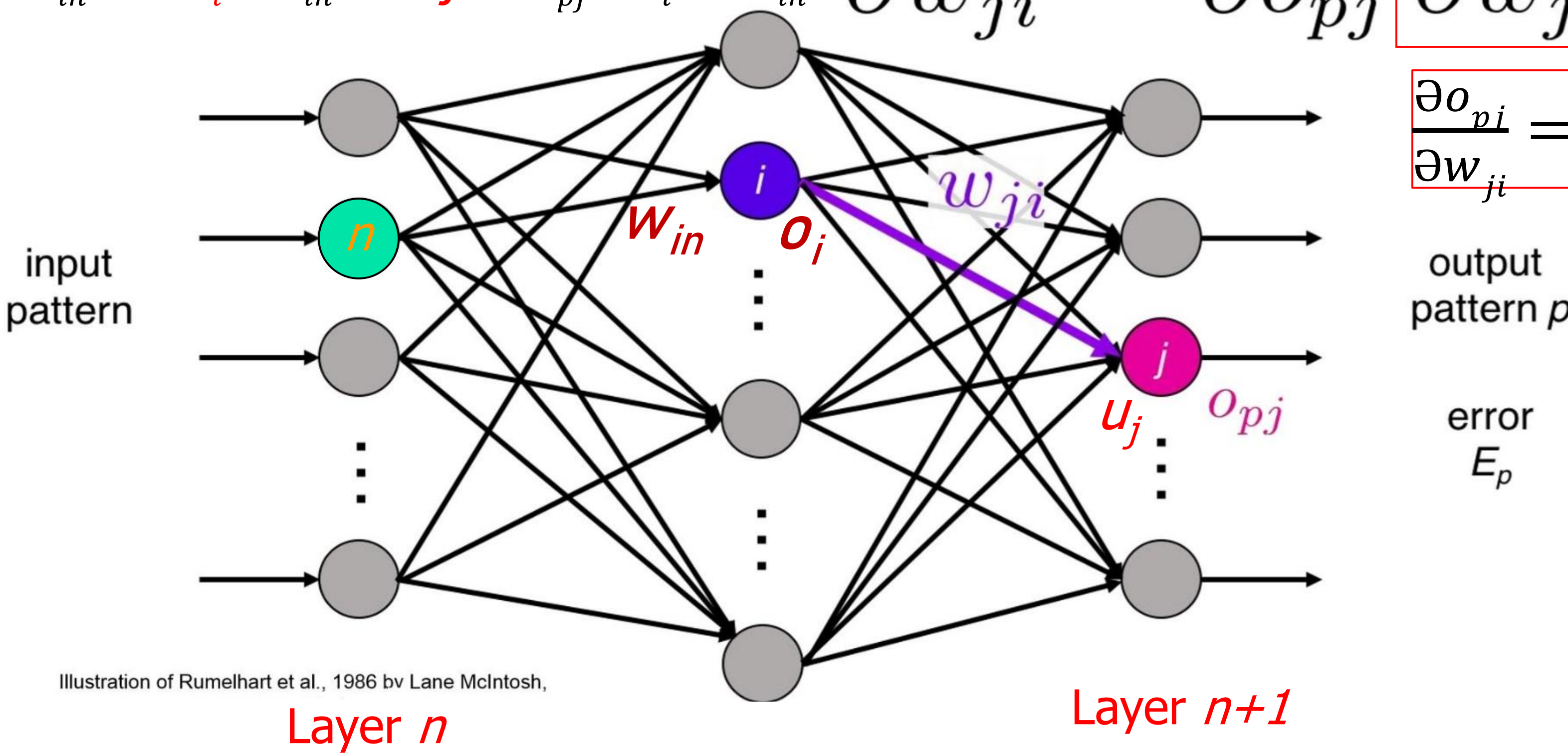
Gradients are noisy but still make good progress on average

Chain Rule of Derivatives

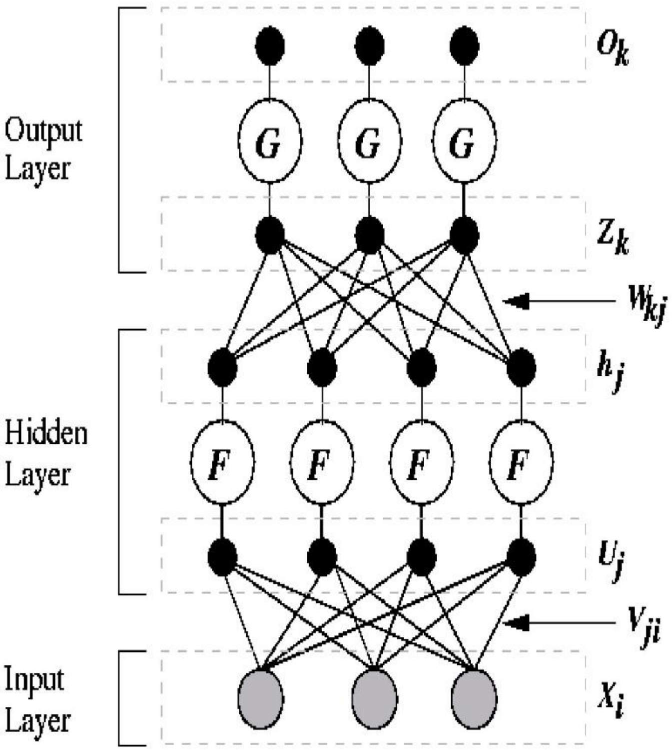
$$\frac{\partial E_p}{\partial w_{in}} = \overset{\delta_n}{\frac{\partial E_p}{\partial o_i}} \frac{\partial o_i}{\partial w_{in}} = \overset{\delta_{n+1}}{\left(\sum_j \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial o_i}\right)} \frac{\partial o_i}{\partial w_{in}}$$

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \boxed{\frac{\partial o_{pj}}{\partial w_{ji}}}$$

$$\boxed{\frac{\partial o_{pj}}{\partial w_{ji}} = \frac{\partial o_{pj}}{\partial u_j} \frac{\partial u_j}{\partial w_{ji}}}$$



Different Loss for BP



- When using a neural network as a function approximator (regressor), a sigmoid activation and MSE as loss function work well

$$E = \frac{1}{2}\sum_k (o_k - t_k)^2; \qquad \frac{\partial E}{\partial o_k} = o_k - t_k$$

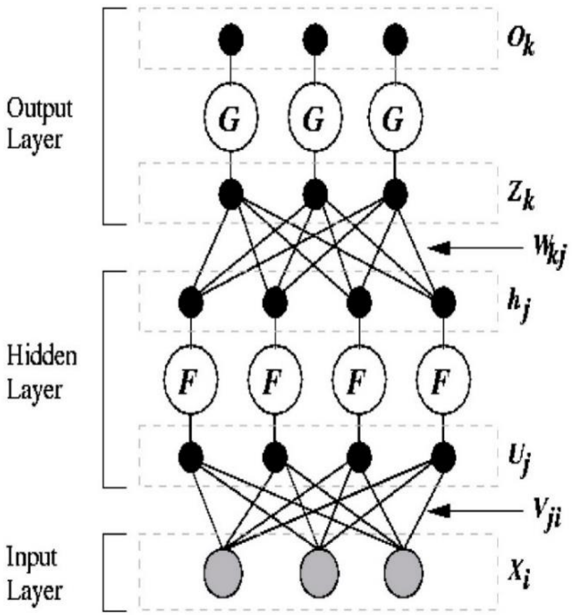
- The chain rule for δ_{n+1}

$$\frac{\partial E}{\partial o} = o - t$$

$$\frac{\partial o}{\partial z} = o(1-o)$$

$$\frac{\partial E}{\partial z} = \frac{\partial E}{\partial o}\frac{\partial o}{\partial z} = (o-t)o(1-o)$$

Different Loss for BP



- For classification, if it is a binary (2-class) problem, then cross-entropy error function often does better

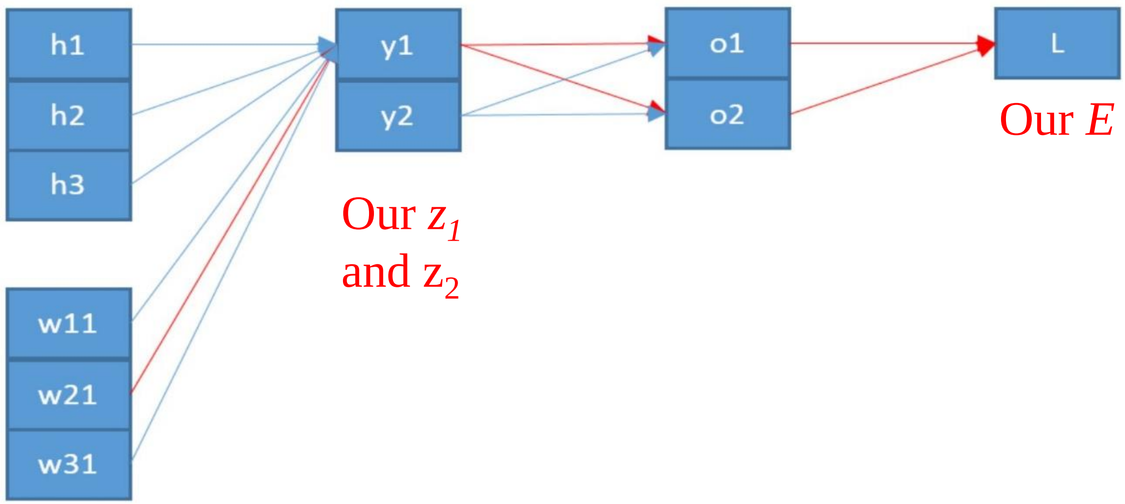
$$E = -\sum_{n=1}^{N} t^{(n)} \log o^{(n)} + (1 - t^{(n)}) \log(1 - o^{(n)})$$

$$o^{(n)} = (1 + \exp(-z^{(n)}))^{-1} \quad \text{sigmoid}$$

- For multi-class classification problems, use the softmax activation (prob.)

$$\boxed{E = -\sum_{n} \sum_{k} t_k^{(n)} \log o_k^{(n)} \qquad o_k^{(n)} = \frac{\exp(z_k^{(n)})}{\sum_j \exp(z_j^{(n)})}}$$

Gradient for Cross-Entropy



$$L = -t_1 \log o_1 - t_2 \log o_2$$

$$o_1 = \frac{\exp(y_1)}{\exp(y_1) + \exp(y_2)}$$

$$o_2 = \frac{\exp(y_2)}{\exp(y_1) + \exp(y_2)}$$

$$y_1 = w_{11} h_1 + w_{21} h_2 + w_{31} h_3$$

$$y_2 = w_{12} h_1 + w_{22} h_2 + w_{32} h_3$$

$$\frac{\partial L}{\partial o_1} = -\frac{t_1}{o_1}$$

$$\frac{\partial L}{\partial o_2} = -\frac{t_2}{o_2}$$

$$\frac{\partial o_1}{\partial y_1} = \frac{\exp(y_1)}{\exp(y_1) + \exp(y_2)} - \left(\frac{\exp(y_1)}{\exp(y_1) + \exp(y_2)}\right)^2 = o_1(1 - o_1)$$

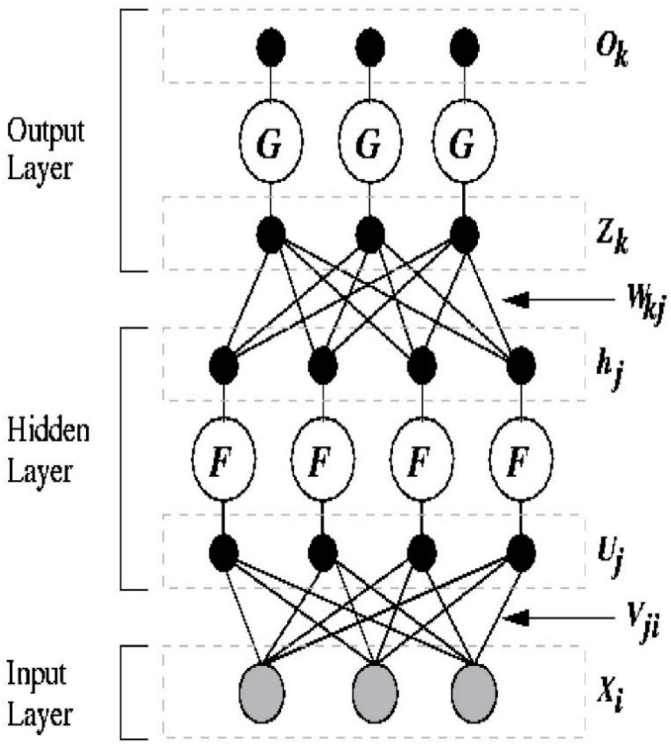
$$\frac{\partial o_2}{\partial y_1} = \frac{-\exp(y_2)\exp(y_1)}{(\exp(y_1) + \exp(y_2))^2} = -o_2 o_1$$

$$\frac{\partial y_1}{\partial w_{21}} = h_2$$

$$\begin{aligned}
\frac{\partial L}{\partial w_{21}} &= \frac{\partial L}{\partial o_1}\frac{\partial o_1}{\partial y_1}\frac{\partial y_1}{\partial w_{21}} + \frac{\partial L}{\partial o_2}\frac{\partial o_2}{\partial y_1}\frac{\partial y_1}{\partial w_{21}} \\
&= \frac{-t_1}{o_1}[o_1(1 - o_1)]h_2 + \frac{-t_2}{o_2}(-o_2 o_1)h_2 \\
&= h_2(t_2 o_1 - t_1 + t_1 o_1) \\
&= h_2(o_1(t_1 + t_2) - t_1) \\
&= h_2(o_1 - t_1)
\end{aligned}$$

$t_1+t_2=1$, because the vector $\boldsymbol{t}$ is a one-hot vector

Gradient Descent Updates



- How often to update
  - After each training sample ($N=1$) or
  - After a mini-batch of N training patterns (e.g., N=32, 64, 128, 256, etc)

$$w_{ki} \leftarrow w_{ki} - \eta \frac{\partial E}{\partial w_{ki}} = w_{ki} - \eta \sum_{n=1}^{N} (o_k^{(n)} - t_k^{(n)}) o_k^{(n)} (1 - o_k^{(n)}) x_i^{(n)}$$

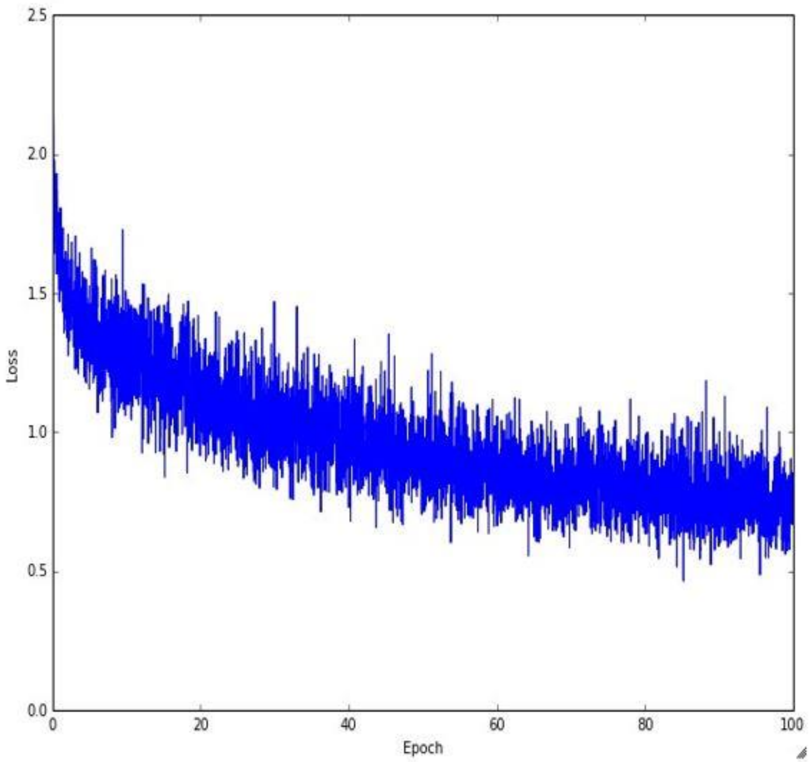
- How much to update
  - Use a fixed or an adaptive learning rate
  - Add a momentum term with leakage

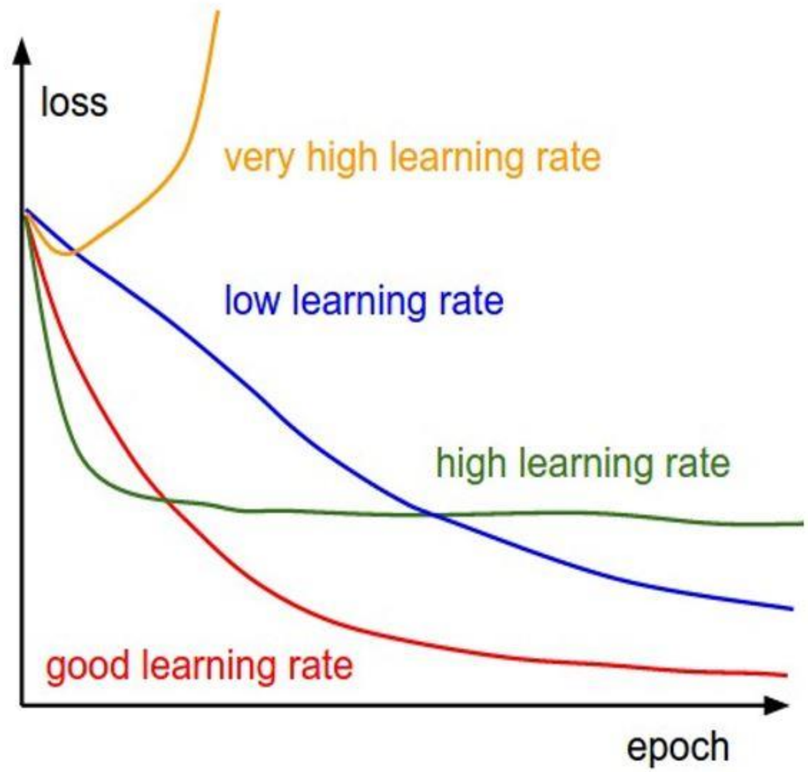
$$\begin{aligned} V_t &= \mu V_{t-1} - \alpha \nabla L_t(W_{t-1}) \\ W_t &= W_{t-1} + V_t \end{aligned}$$

- $\alpha > 0$ – *learning rate* (typical choices: 0.01, 0.1)
- $\mu \in [0, 1)$ – *momentum* (typical choices: 0.9, 0.95, 0.99)

Momentum smooths updates, enhancing stability and speed.

Choice of Learning Rate





More Variants of Updates

- Adaptive Gradient (AdaGrad)

$$W_t = W_{t-1} - \alpha \frac{\nabla L_t(W_{t-1})}{\sqrt{\sum_{t'=1}^{t} \nabla L_{t'}(W_{t'-1})^2}}$$

- Root Mean Square Propagation (RMSProp)

$$\begin{aligned} R_t &= \gamma R_{t-1} + (1-\gamma)\nabla L_t(W_{t-1})^2 \\ W_t &= W_{t-1} - \alpha \frac{\nabla L_t(W_{t-1})}{\sqrt{R_t}} \end{aligned}$$

Adaptive Moment (Adam) Estimation Updates

- Combine the advantages of:
  - AdaGrad – works well with sparse gradients
  - RMSProp – works well in non-stationary settings
- Maintain exponential moving averages of gradient and its square
- Update proportional to $\frac{\text{average gradient}}{\sqrt{\text{average squared gradient}}}$

$M_0 = \mathbf{0}, R_0 = \mathbf{0}$ (Initialization)

For $t = 1, \ldots, T$:

$$M_t = \beta_1 M_{t-1} + (1 - \beta_1)\nabla L_t(W_{t-1}) \quad \text{(1st moment estimate)}$$

$$R_t = \beta_2 R_{t-1} + (1 - \beta_2)\nabla L_t(W_{t-1})^2 \quad \text{(2nd moment estimate)}$$

$$\hat{M}_t = M_t / (1 - (\beta_1)^t) \quad \text{(1st moment bias correction)}$$

$$\hat{R}_t = R_t / (1 - (\beta_2)^t) \quad \text{(2nd moment bias correction)}$$

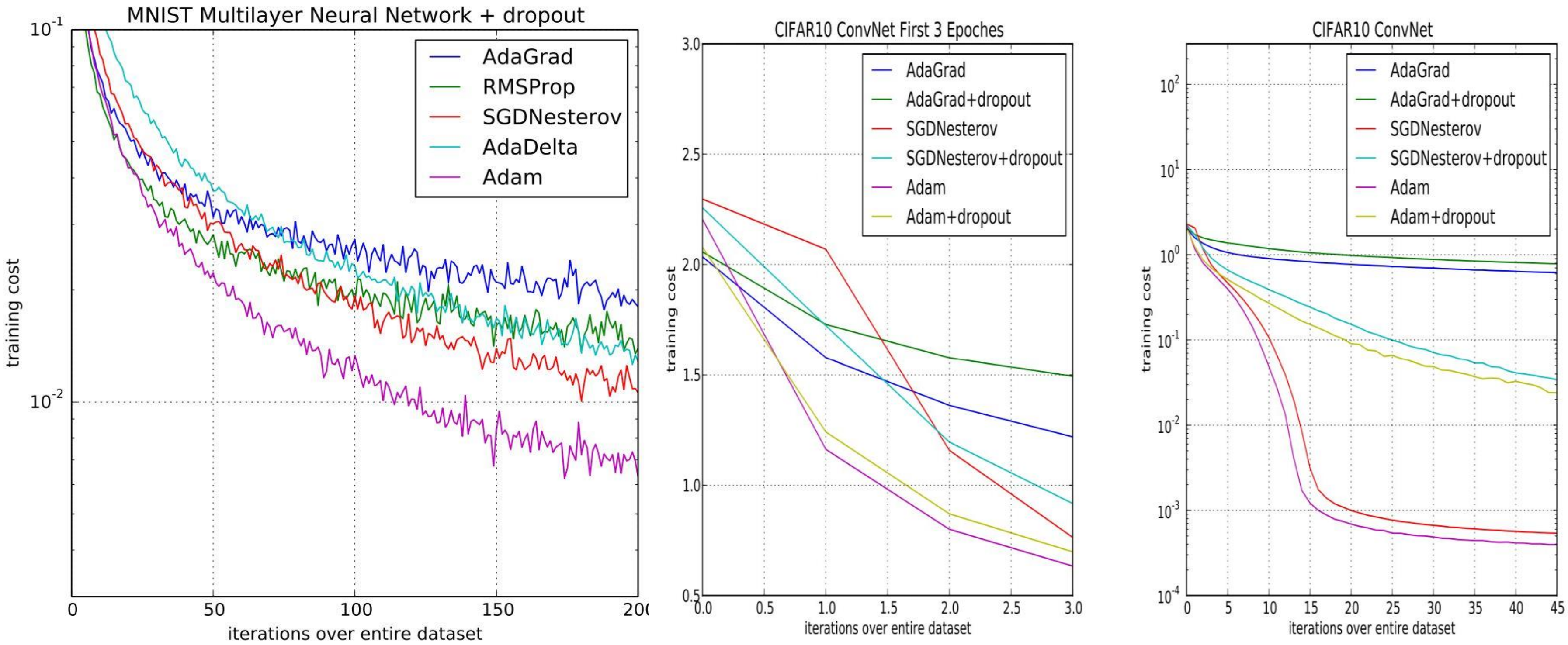
$$W_t = W_{t-1} - \alpha \frac{\hat{M}_t}{\sqrt{\hat{R}_t + \epsilon}} \quad \text{(Update)}$$

Return W_T

Hyper-parameters:

- $\alpha > 0$ – learning rate (typical choice: 0.001)
- $\beta_1 \in [0, 1)$ – 1st moment decay rate (typical choice: 0.9)
- $\beta_2 \in [0, 1)$ – 2nd moment decay rate (typical choice: 0.999)
- $\epsilon > 0$ – numerical term (typical choice: 10^{-8})

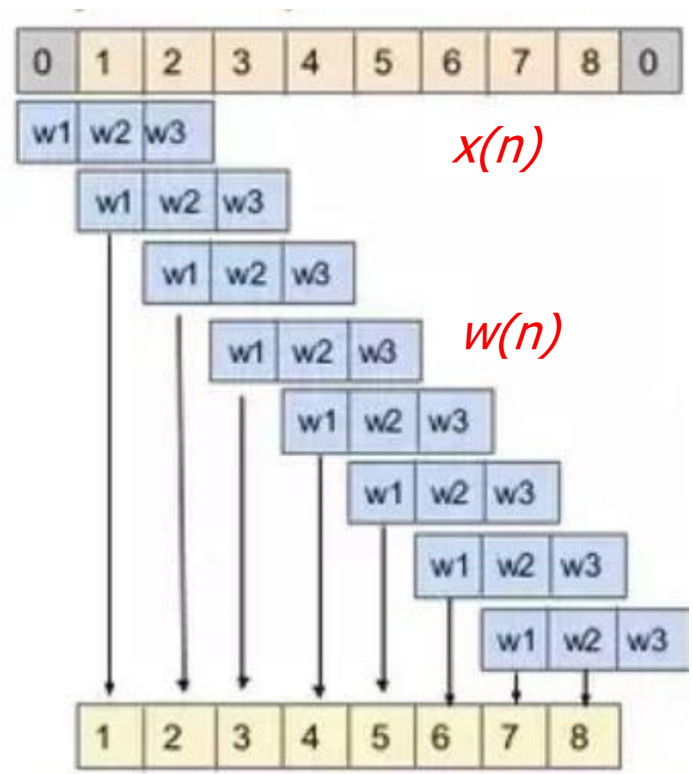
Adam: A Method for Stochastic Optimization

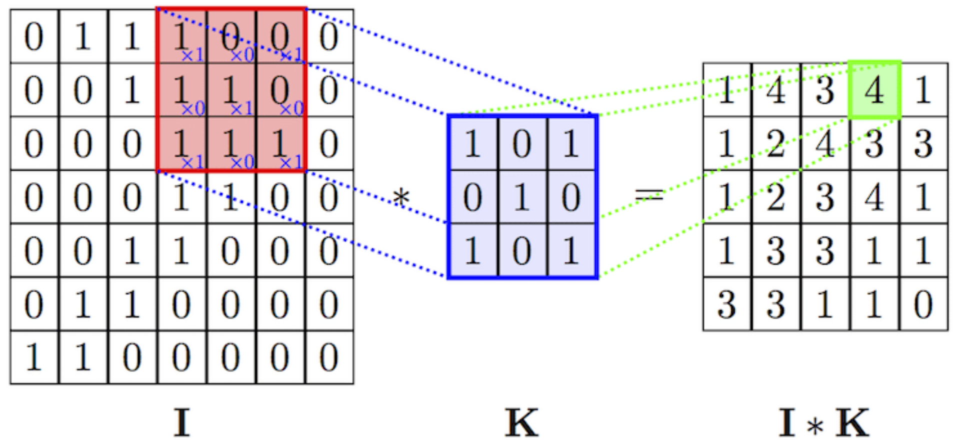


Diederik P. Kingma, Jimmy Ba, "Adam: A Method for Stochastic Optimization," ICLR 2015, https://arxiv.org/abs/1412.6980

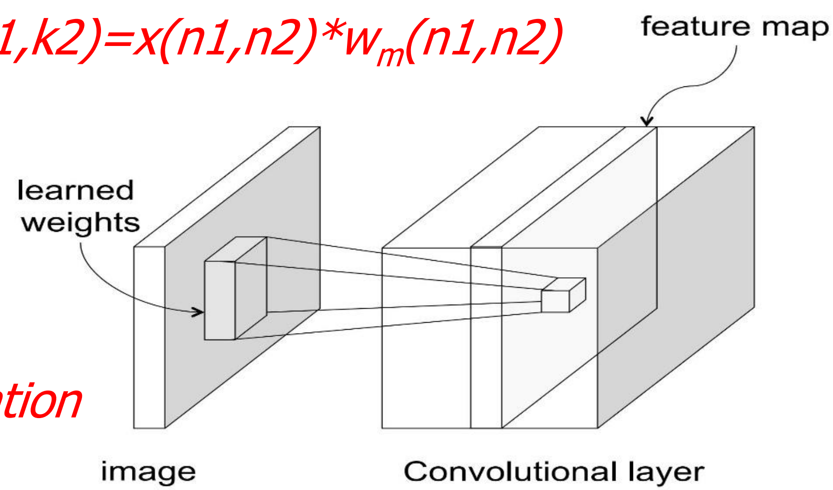
Deep Convolution Neural Networks (CNNs)

1-D and 2-D Convolution





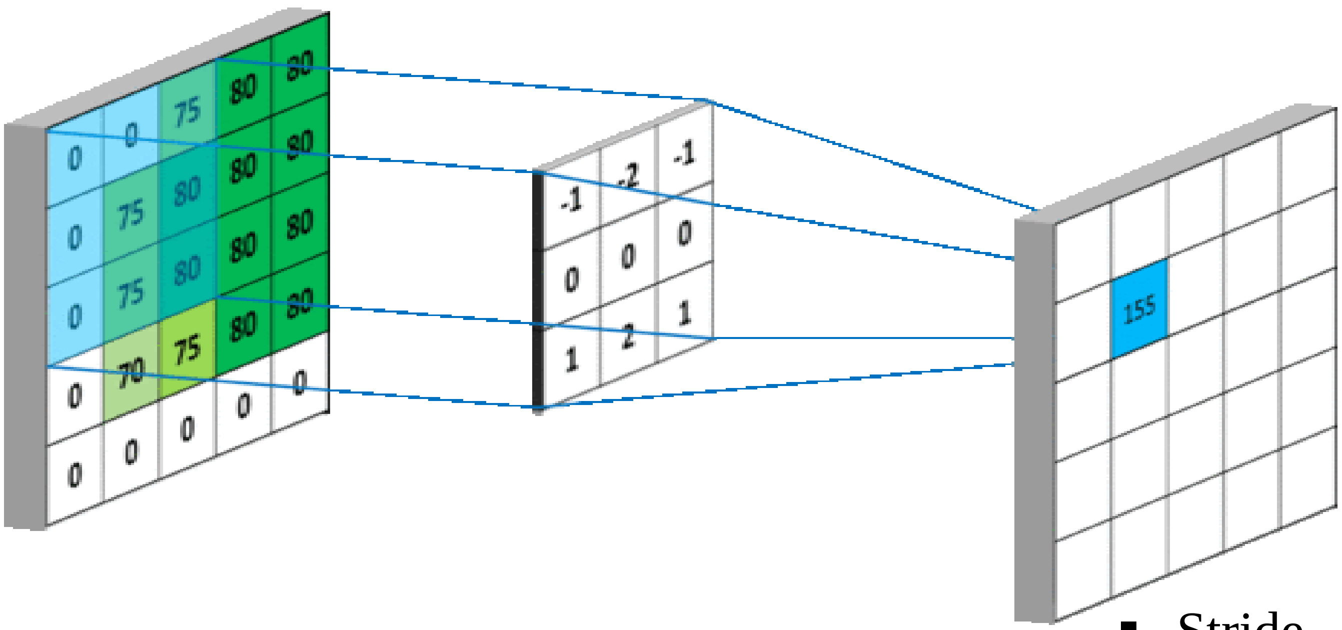
$F_m(k1,k2)=x(n1,n2)*w_m(n1,n2)$



$y(k)=x(n)*w(n)=sum_n\ x(n+k)w(k)$ *correlation*

should be x(n-k)w(k) mathematically

2D Convolution Example



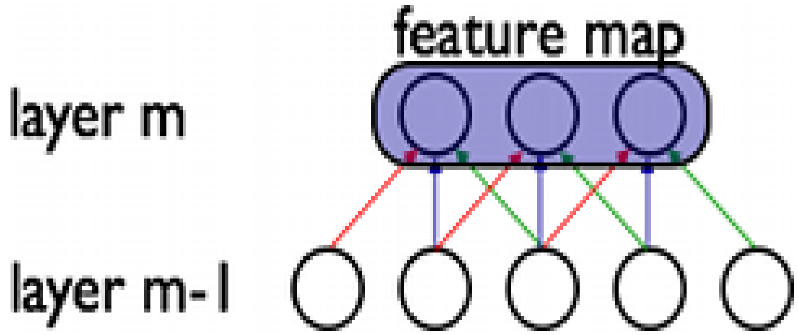
$F_m(k1,k2)=x(n1,n2)*w_m(n1,n2)$

- Stride
- Padding
- Pooling (Subsampling)

Let the convolution kernels $w_m(n1, n2)$ learnable in an MLP

Shared Convolutional Kernels

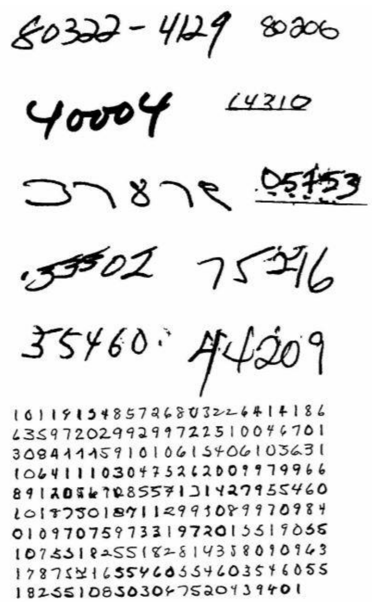
- Replicating units in this way allows for features to be "detected" regardless of their position in the visual field.
- Additionally, weight sharing increases learning efficiency by greatly reducing the number of learned free parameters.
- The constraints on the model enable CNNs to achieve better generalization on vision problems.

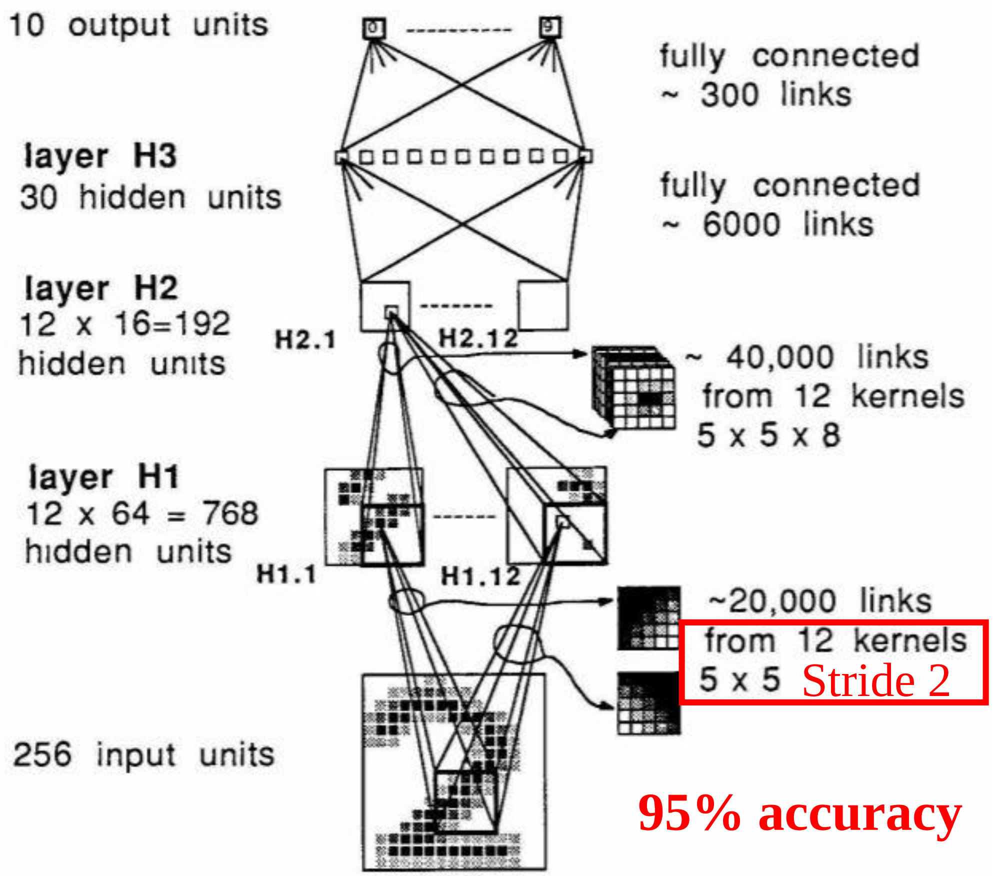


Convolution & Pooling

Shared weights and simplified nonlinearity [LeCun 1989]

LeNet for Handwritten Digits (1989)





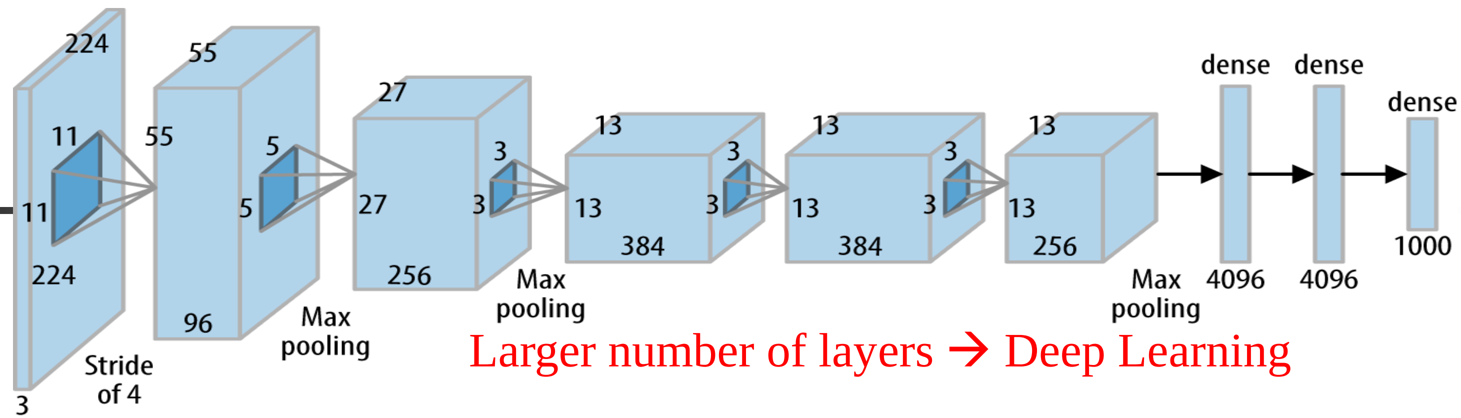
95% accuracy

- For layer H1: 768 hidden units, 768x256=199608 connections, but only 1068 trainable weights (25x12+768 biases)

Y. LeCun, et al, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, Winter 1989.

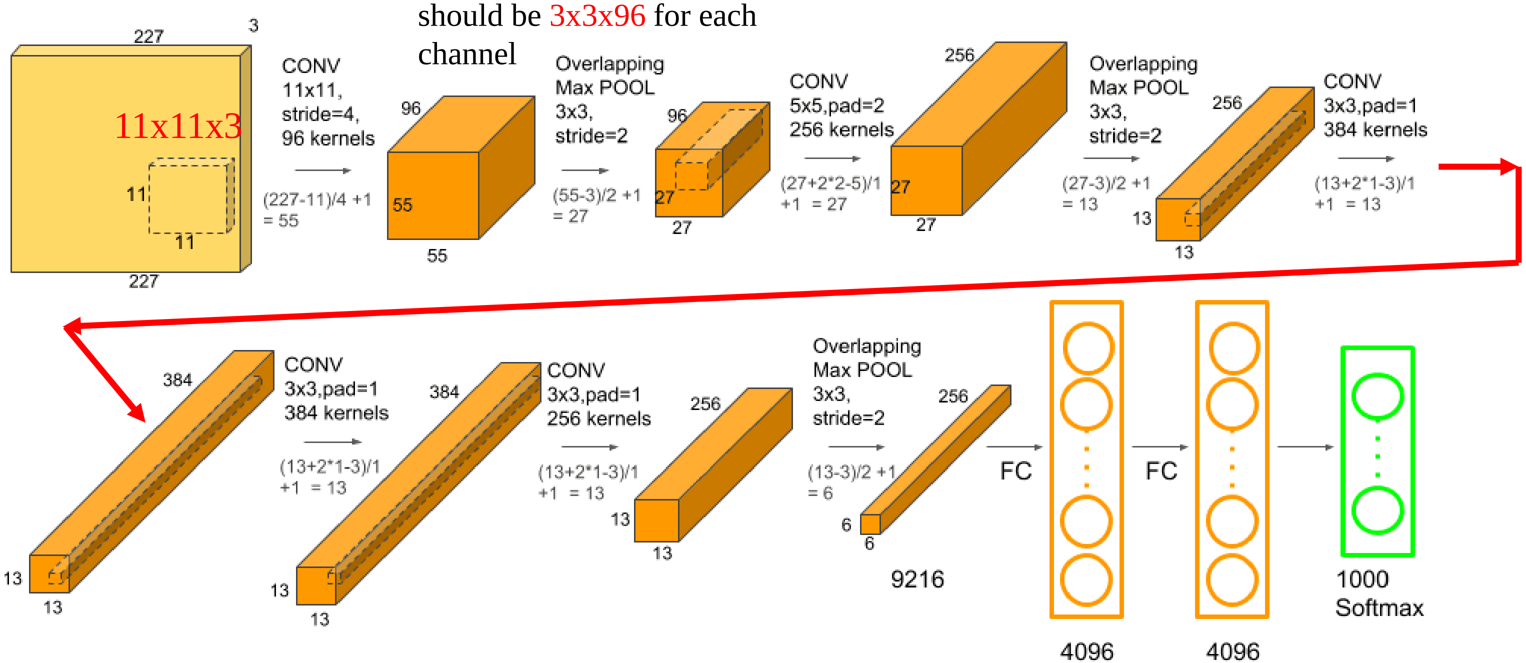
Ice Age of Neural Network based Learning 1992-2012

AlexNet for ImageNet



Larger number of layers → Deep Learning

Note that the 2D convolution should be 3x3x96 for each channel



Alex Krizhevsky, et al. "ImageNet Classification with Deep Convolutional Neural Networks," NIPS 2012.