PennState Institute for Computational and Data Sciences

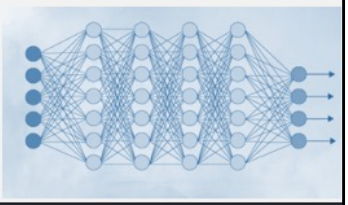
Center for Artificial Intelligence Foundations & Scientific Applications
Artificial Intelligence Research Laboratory

PennState Clinical and Translational Science Institute

Multi-layer Neural Networks and Deep Learning

Vasant G. Honavar
Dorothy Foehr Huck and J. Lloyd Huck Chair in Biomedical Data Sciences and Artificial Intelligence
Professor of Data Sciences, Informatics, Computer Science and Engineering, Bioinformatics & Genomics, Public Health Sciences and Neuroscience
Director, Center for Artificial Intelligence Foundations and Scientific Applications
Associate Director, Institute for Computational and Data Sciences
Pennsylvania State University

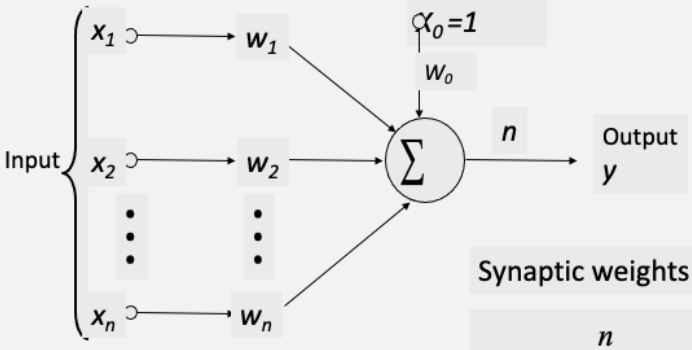
vhonavar@psu.edu
http://faculty.ist.psu.edu/vhonavar
http://ailab.ist.psu.edu



Neural Networks and Deep Learning

- Learning to approximate real-valued functions
- Bayesian recipe for learning real-valued functions
- Review: Learning linear functions using gradient descent in weight space
- Universal function approximation theorem
- Learning nonlinear functions using gradient descent in weight space
- Practical considerations and examples
- Deep Learning

Recall Linear Regression



Synaptic weights

$$y = \sum_{i=0}^{n} w_i x_i$$

Learning a linear function

$\mathbf{W} = [W_0 \ldots\ldots W_n]^T$ is the weight vector

$\mathbf{X}_p = [X_{0p} \ldots . X_{np}]^T$ is the pth training sample

$y_p = \sum_i W_i X_{ip} = \mathbf{W} \bullet \mathbf{X}_p$ is the output of the neuron for input $\mathbf{X}_p$

$\mathbf{X}_p = f(\mathbf{X}_p)$ is the desired output for input $\mathbf{X}_p$

$e_p = (d_p - y_p)$ is the *error* of the neuron on input $\mathbf{X}_p$

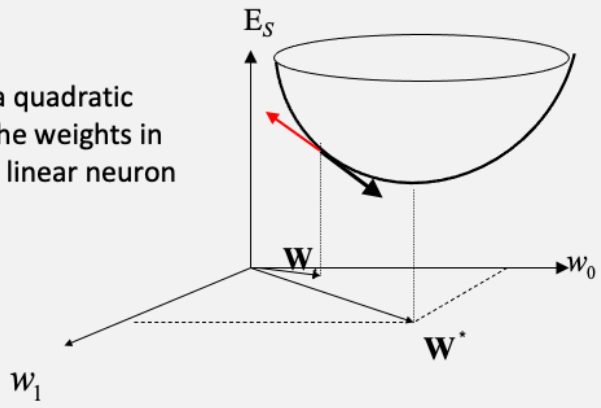
$S = \{(\mathbf{X}_p, d_p)\}$ is`a (multi) set of training examples

$E_S(\mathbf{W}) = E_S(W_0, W_1, \ldots\ldots W_n) = \frac{1}{2}\sum_p e_p^2$ is the estimated error of $\mathbf{W}$ on training set S

Goal: Find $\mathbf{W}^* = \arg\min_{\mathbf{W}} E_S(\mathbf{W})$

Learning a linear function from data – linear regression

The error is a quadratic function of the weights in the case of a linear neuron



Learning linear functions

$$w_i \leftarrow w_i - \eta \frac{\partial E}{\partial w_i}$$

$$\frac{\partial E}{\partial w_i} = \frac{1}{2}\frac{\partial}{\partial w_i}\left\{\sum_p e_p^2\right\} = \frac{1}{2}\left(\sum_p \frac{\partial}{\partial w_i}\left(e_p^2\right)\right)$$

$$= \frac{1}{2}\left(\sum_p (2e_p)\frac{\partial e_p}{\partial w_i}\right) = \sum_p e_p\left(\frac{\partial e_p}{\partial y_p}\right)\left(\frac{\partial y_p}{\partial w_i}\right) = \sum_p e_p(-1)\left(\frac{\partial}{\partial w_i}\left(\sum_{j=0}^{n} w_j x_{jp}\right)\right)$$

$$= -\sum_p (d_p - y_p)\left(\frac{\partial}{\partial w_i}\left(w_i x_{ip} + \sum_{j\neq i} w_j x_{jp}\right)\right)$$

$$= -\sum_p (d_p - y_p)\left(\frac{\partial}{\partial w_i}(w_i x_{ip}) + \frac{\partial}{\partial w_i}\left(\sum_{j\neq i} w_j x_{jp}\right)\right)$$

$$= -\sum_p (d_p - y_p) x_{ip}$$

$$w_i \leftarrow w_i + \eta \sum_p (d_p - y_p) x_{ip}$$

Least Mean Square Error (LMSE) Learning Rule or the delta rule

Batch Update $$w_i \leftarrow w_i + \eta \sum_p \left(d_p - y_p\right) x_{ip}$$

Per sample Update $$w_i \leftarrow w_i + \eta \left(d_p - y_p\right) x_{ip}$$

Momentum update

$$w_i(t+1) = w_i(t) + \Delta w_i(t)$$

$$\Delta w_i(t) = -\eta \left.\frac{\partial E}{\partial w_i}\right|_{w_i = w_i(t)} + \alpha \Delta w_i(t-1) \text{ where } 0 < \alpha < 1$$

$$= -\eta \sum_{\tau=0}^{t} \alpha^{t-\tau} \left.\frac{\partial E}{\partial w_i}\right|_{w_i = w_i(\tau)}$$

The momentum update allows effective learning rate to increase when feasible and decrease when necessary. Converges for $0 \leq \alpha < 1$

Learning nonlinear functions

- Motivations
- Universal function approximation theorem (UFAT)
- Derivation of the generalized delta rule
- Back-propagation algorithm
- Practical considerations
- Applications

Motivations

- Psychology – Empirical inadequacy of behaviorist theories of learning
  - simple reward-punishment based learning models are incapable of learning functions (e.g., exclusive OR) which are readily learned by animals (e.g., monkeys)
- Artificial Intelligence – the need for learning nonlinear functions where the form of the nonlinear relationship is unknown a-priori
- Statistics – Limitations of linear regression when the input-ouput relationship is nonlinear and is of unknown form
- Control – Need for nonlinear control methods

These considerations led multiple research communities to independently pursue generalizations of linear regression or the delta rule to the nonlinear setting

Universal function approximation theorem* (UFAT)

- Let $\varphi : \Re \rightarrow \Re$ be a non-constant, bounded (hence non-linear), monotone, continuous function.
- Let I_N be the N-dimensional unit hypercube in $\Re^N$.
- Let $C(I_N) = \{f: I_N \rightarrow \Re\}$ be the set of all continuous functions with domain I_N *and* range $\Re$.
- Then for any function $f \in C(I_N)$ and any $\varepsilon > 0$, $\exists$ an integer L and a sets of real values $\theta, \alpha_j, \theta_j, w_{ji}$ $(1 \leq j \leq L; 1 \leq I \leq N)$ such that

$$F(x_1, x_2 \ldots x_N) = \sum_{j=1}^{L} \alpha_j \phi\left(\sum_{i=1}^{N} w_{ji} x_i - \theta_j\right) - \theta$$

is a uniform approximation of f – that is,

$$\forall (x_1, \ldots x_N) \in I_N, \quad \left|F(x_1, \ldots x_N) - f(x_1, \ldots x_N)\right| < \varepsilon$$

* Cybenko, 1989

Universal function approximation theorem (UFAT)

$$F(x_1, x_2 \ldots x_n) = \sum_{j=1}^{L} \alpha_j \varphi\left(\sum_{i=1}^{N} w_{ji} x_i - \theta_j\right) - \theta$$

- The sigmoid function satisfies the UFAT requirements

$$\varphi(z) = \frac{1}{1+e^{-az}}; a > 0 \qquad \lim_{z \to -\infty} \varphi(z) = 0; \quad \lim_{z \to +\infty} \varphi(z) = 1$$

- Later it was shown that similar universal approximation properties can be guaranteed for a variety of other choices for $\varphi(z)$ – e.g., radial basis functions

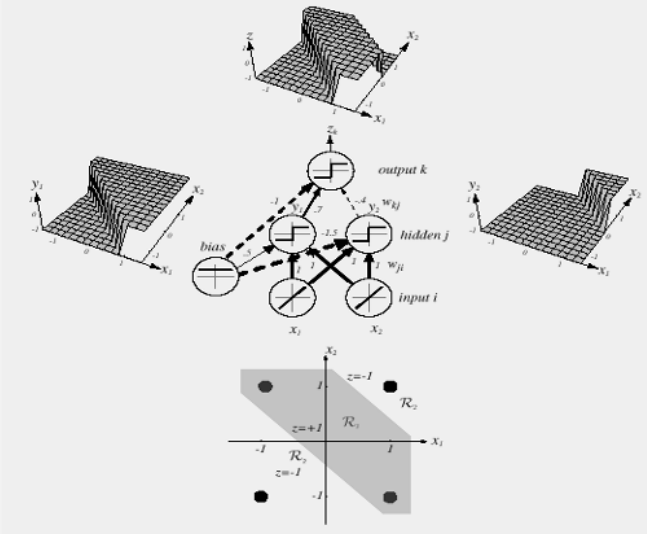
Implications of Universal function approximation theorem

- UFAT guarantees the existence of arbitrarily accurate approximations of continuous functions defined over bounded subsets of $\Re^N$
- UFAT characterizes the representational power a certain class of multi-layer networks relative to the set of continuous functions defined on bounded subsets of $\Re^N$
- UFAT is not constructive – it does not tell us how to choose the parameters to construct a desired function
- To learn an unknown nonlinear continuous function from data, we need an algorithm to search the space of multilayer networks
- By interpreting the outputs of the network as posterior probabilities of classes, we can use such networks for classification

Feed-forward neural networks

- A feed-forward n-layer network consists of n layers of nodes
- 1 layer of Input nodes
- *n*-2 layers of Hidden nodes
- 1 layer of Output nodes
- interconnected by modifiable weights from input nodes to the hidden nodes and the hidden nodes to the output nodes
- More general topologies (e.g., with connections that skip layers, e.g., direct connections between input and output nodes) are possible

A 3-layer network approximates the EXOR function



Three-layer feed-forward neural network

- A single *bias* unit is connected to each unit other than the input units
- Net *input*

$$n_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} \equiv \mathbf{W}_j \cdot \bullet \mathbf{X},$$

- where the subscript i indexes units in the input layer, j in the hidden; w_{ji} denotes the input-to-hidden layer weights at the hidden unit j.
- The output of a hidden unit is a nonlinear function of its net input. That is, $y_j = f(n_j)$ e.g., $y_j = \dfrac{1}{1+e^{-n_j}}$

Three-layer feed-forward neural network

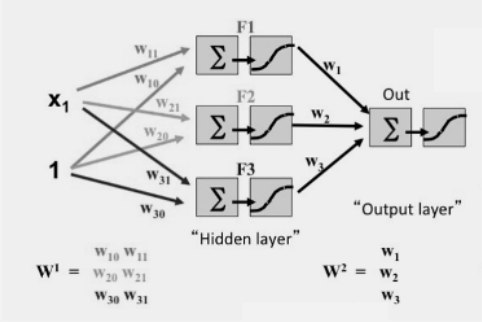
- Each output unit similarly computes its net activation based on the hidden unit signals as:

$$n_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj} = \mathbf{W}_k \bullet \mathbf{Y},$$

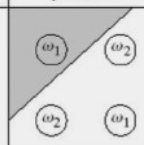
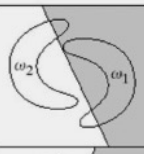
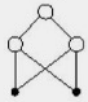
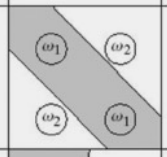
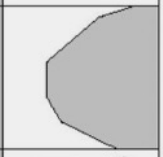
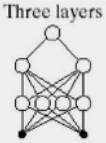
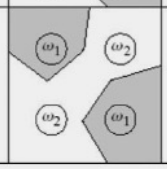
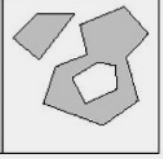
- where the subscript k indexes units in the output layer and n_H denotes the number of hidden units
- The output can be a linear or nonlinear function of the net input e.g., $z_k = n_k$

Computing nonlinear functions

- A 2-layer network with 2 inputs and 1 output
- The hidden layer and output layer nodes are sigmoid neurons



Decision boundaries realizable by multi-layer neural networks

Network structure	Type of decision region	Solution to exclusive-OR problem	Classes with meshed regions	Most general decision surface shapes
Single layer 	Single hyperplane			
Two layers 	Open or closed convex regions			
Three layers 	Arbitrary (complexity limited by the number of nodes)			

Learning nonlinear functions – nonlinear regression

- Given a training set determine
- Network structure – number of hidden nodes or more generally, network topology
  - Start small and grow the network
  - Start with a sufficiently large network and prune away the unnecessary connections
- For a given structure, determine the parameters (weights) that minimize the error on the training samples (e.g., the mean squared error)
- For now, we focus on the latter

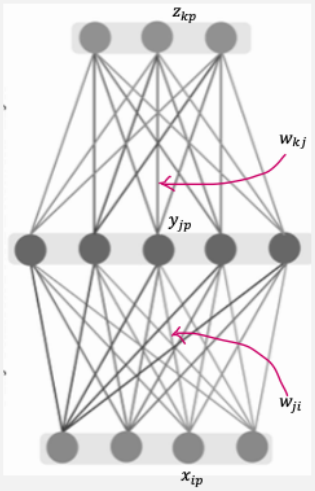
Generalized delta rule – error back-propagation

- Challenge – we know the desired outputs for nodes in the output layer, but not the hidden layer
- Need to solve the credit assignment problem – dividing the credit or blame for the performance of the output nodes among hidden nodes
- Generalized delta rule offers an elegant solution to the credit assignment problem in feed-forward neural networks in which each neuron computes a differentiable function of its inputs
- Solution can be generalized to other kinds of networks, including recurrent networks (with feedback loops)

Feed-forward networks

- Forward operation (computing output for a given input based on the current weights)
- Learning – modification of the network parameters (weights) to minimize an appropriate error measure
- Because each neuron computes a differentiable function of its inputs
  - If error is a differentiable function of the network outputs, it is a differentiable function of the weights
  - We can learn the weights by performing gradient descent!

A fully connected 2-layer network



Given the the pth sample $\mathbf{x}_p$

- Let x_{ip} be the ith input
- Let $n_{jp} = \sum_i w_{ji}\, x_{ip}$ be the net input of the jth hidden neuron
- Let $y_{jp} = \frac{1}{1+e^{-n_{jp}}}$ be the output of the jth hidden neuron
- Let $n_{kp} = \sum_j w_{kj}\, z_{jp}$ be the net input of the kth output neuron
- Let $z_{kp} = n_{kp}$ be the output of the kth output neuron

Generalized delta rule

- Let t_{kp} be the k-th target (or desired) output for input pattern $\mathbf{X}_p$ and z_{kp} be the output produced by k-th output node and let $\mathbf{W}$ represent all the weights in the network
- Training error: $E_S(\mathbf{W}) = \frac{1}{2}\sum_p \sum_{k=1}^{M} (t_{kp} - z_{kp})^2 = \sum_p E_p(\mathbf{W})$
- The weights are initialized with pseudo-random values and are changed in a direction that will reduce the error:

Batch Update $\Delta w_{ji} = -\eta \frac{\partial E_S}{\partial w_{ji}}$ $\Delta w_{kj} = -\eta \frac{\partial E_S}{\partial w_{kj}}$

Per sample update $\Delta w_{ji} = -\eta \frac{\partial E_p}{\partial w_{ji}}$ $\Delta w_{kj} = -\eta \frac{\partial E_p}{\partial w_{kj}}$

Generalized delta rule

Change in hidden to output weights $\Delta w_{kj} = -\eta \frac{\partial E_p}{\partial w_{kj}}$

$$E_p = \frac{1}{2}\sum_k (t_{kp} - z_{kp})^2$$

$$\frac{\partial E_p}{\partial w_{kj}} = \frac{\partial E_p}{\partial z_{kp}} \frac{\partial z_{kp}}{\partial w_{kj}} = \frac{\partial E_p}{\partial z_{kp}} \frac{\partial z_{kp}}{\partial n_{kp}} \frac{\partial n_{kp}}{\partial w_{kj}} = -(t_{kp} - z_{kp})(1) y_{jp}$$

$E_p \downarrow z_{kp} \downarrow n_{kp} \downarrow w_{kj}$

Let $t_{kp} - z_{kp} = \delta_{kp}$

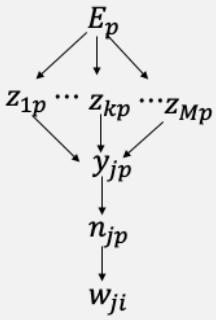
$$w_{kj} \leftarrow w_{kj} - \eta \frac{\partial E_p}{\partial w_{kj}} = w_{kj} + (t_{kp} - z_{kp}) y_{jp} = w_{kj} + \delta_{kp} y_{jp}$$

Generalized delta rule

Change in input to hidden weights $\Delta w_{ji} = -\eta \frac{\partial E_p}{\partial w_{ji}}$

$$\frac{\partial E_p}{\partial w_{ji}} = \sum_{k=1}^{M} \frac{\partial E_p}{\partial z_{kp}} \frac{\partial z_{kp}}{\partial w_{ji}} = \sum_{k=1}^{M} \frac{\partial E_p}{\partial z_{kp}} \frac{\partial z_{kp}}{\partial y_{jp}} \cdot \frac{\partial y_{jp}}{\partial n_{jp}} \cdot \frac{\partial n_{jp}}{\partial w_{ji}}$$

Chain Rule



$$= \sum_{k=1}^{M} \frac{\partial}{\partial z_{kp}} \left[\frac{1}{2} \sum_{l=1}^{M} (t_{lp} - z_{lp})^2 \right] (w_{kj})(y_{jp})(1 - y_{jp})(x_{ip})$$

$$= -\sum_{k=1}^{M} (t_{kp} - z_{kp})(w_{kj})(y_{jp})(1 - y_{jp})(x_{ip})$$

$$= -\underbrace{\left(\sum_{k=1}^{M} \delta_{kp} (w_{kj})(y_{jp})(1 - y_{jp}) \right)}_{\delta_{jp}} (x_{ip})$$

$$= -\delta_{jp} x_{ip}$$

$$w_{ji} \leftarrow w_{ji} + \eta \delta_{jp} x_{ip}$$

In the preceding slide, we have made use of the fact that

$$\frac{\partial y_{jp}}{\partial n_{jp}} = \frac{\partial}{\partial n_{jp}}\left(\frac{1}{1+e^{-n_{jp}}}\right)$$

$$= \frac{(-e^{-n_{jp}})}{(1+e^{-n_{jp}})^2}$$

$$= \left(\frac{1}{1+e^{-n_{jp}}}\right)\left(1 - \frac{1}{1+e^{-n_{jp}}}\right)$$

$$= y_{jp}(1 - y_{jp})$$

Backpropagation algorithm

- Start with small random initial weights
- Until desired stopping criterion is satisfied do
- Select a training sample from S
  - Compute the outputs of all nodes based on current weights and the input sample
  - Compute the weight updates for output nodes
  - Compute the weight updates for hidden nodes
  - Update the weights

Using neural networks for classification

- Network outputs are real valued.
- How can we use the networks for classification?

$$F(\mathbf{X}_p) = \arg\max_k z_{kp}$$

Classify a pattern by assigning it to the class that corresponds to the index of the output node with the largest output for the pattern

Training multi-layer networks – Some Useful Tricks

- Initializing weights to small random values that place the neurons in the linear portion of their operating range for most of the patterns in the training set improves speed of convergence e.g.,

$$w_{ji} = \pm \frac{1}{2N} \sum_{i=1,..,N} \frac{1}{|x_i|}$$

For input to hidden layer weights with the sign of the weight chosen at random

$$w_{kj} = \pm \frac{1}{2N} \sum_{i=1,..,N} \left(\frac{1}{\varphi\left(\sum w_{ji} x_i \right)} \right)$$

For hidden to output layer weights with the sign of the weight chosen at random

Some Useful Tricks

- Use of momentum term allows the effective learning rate for each weight to adapt as needed and helps speed up convergence – in a network with 2 layers of weights,

$$\left.\begin{aligned} w_{ji}(t+1) &= w_{ji}(t) + \Delta w_{ji}(t) \\ \Delta w_{ji}(t) &= -\eta \left.\frac{\partial E_S}{\partial w_{ji}}\right|_{w_{ji}=w_{ji}(t)} + \alpha \Delta w_{ji}(t-1) \\ w_{kj}(t+1) &= w_{kj}(t) + \Delta w_{kj}(t) \\ \Delta w_{kj}(t) &= -\eta \left.\frac{\partial E_S}{\partial w_{ji}}\right|_{w_{kj}=w_{kj}(t)} + \alpha \Delta w_{kj}(t-1) \end{aligned}\right\}$$

where $0 < \alpha, \eta < 1$ with typical values of $\eta = 0.5$ to 0.6, $\alpha = 0.8$ to 0.9

Some Useful Tricks

- Use sigmoid function which satisfies φ(–z)=–φ(z) helps speed up convergence

$$\varphi(z) = a\left(\frac{1-e^{-bz}}{1+e^{-bz}}\right)$$

$$a = 1.716, \mathrm{b} = \frac{2}{3} \Rightarrow \left.\frac{\partial \varphi}{\partial z}\right|_{z=0} \approx 1$$

$$\text{and } \varphi(z) \text{ is linear in the range } -1 < z < 1$$

Some Useful Tricks

- Randomize the order of presentation of training examples from one pass to the next helps avoid local minima
- Introduce small amounts of noise in the weight updates (or into examples) during training helps improve generalization – minimizes over fitting, makes the learned approximation more robust to noise, and helps avoid local minima
- If using the suggested sigmoid nodes in the output layer, set target output for output nodes to be 1 for target class and -1 for all others

Some useful tricks

- Regularization helps avoid over fitting and improves generalization

$$R(\mathbf{W}) = \lambda E(\mathbf{W}) + (1-\lambda) C(\mathbf{W});\ 0 \le \lambda \le 1$$

$$C(\mathbf{W}) = \frac{1}{2}\left(\sum_{ji} w_{ji}^2 + \sum_{kj} w_{kj}^2 \right)$$

$$-\frac{\partial C}{\partial w_{ji}} = -w_{ji} \text{ and } -\frac{\partial C}{\partial w_{kj}} = -w_{kj}$$

Start with λ close to 1 and gradually lower it during training. When λ <1, it tends to drive weights toward zero setting up a tension between error reduction and complexity minimization

Some Useful Tricks

Input and output encodings

- Do not eliminate *natural* proximity in the input or output space
  - Do not normalize input patterns to be of unit length if the length is likely to be relevant for distinguishing between classes
- Do not introduce *unwarranted* proximity as an artifact
  - Do not use $\log_2$ M outputs to encode M classes, use M outputs instead to avoid spurious proximity in the output space
- Use error correcting codes when feasible

Some Useful Tricks

Examples of a good code

- Binary thermometer codes for encoding real values
  - Suppose we can use 10 bits to represent a value between -1.0 and +1.0
  - We can quantize the interval [-1, 1] into 10 equal parts
  - 0.38 in thermometer code is 1111000000
  - 0.60 in thermometer code is 1111110000
  - Note values that are close along the real number line have thermometer codes that are close in Hamming distance

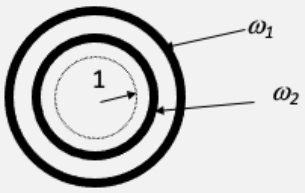
Example of a bad code

- Ordinary binary representations of integers

Some Useful Tricks

- Normalizing inputs – know when and when not to normalize
- Normalizing each input pattern so that it is of unit length is commonplace, but often inappropriate

$$\mathbf{X}_p \leftarrow \frac{\mathbf{X}_p}{\|\mathbf{X}_p\|}$$



Some Useful Tricks

- Better normalization: Scale each component of the input separately to lie between -1 and 1 with mean of 0 and standard deviation of 1

$$\mu_i = \frac{1}{P}\sum_{q=1}^{P} x_{iq}$$

$$\sigma_i^{\ 2} = \frac{1}{P}\sum_{q=1}^{P} x_{iq}^2 - \mu_i^2$$

$$x_{ip} \leftarrow \frac{\left(x_{ip} - \mu_i\right)}{\sigma_i}$$

Some Useful Tricks

Initializing weights (revisited)

Suppose weights are uniformly distributed between $-w$ and $+w$

Standardized input to a hidden neuron is distributed between $-w\sqrt{N}$ and $+w\sqrt{N}$

We want this to fall between -1 and $+1 \Rightarrow \left(w = \frac{1}{\sqrt{N}} \right)$

$$\Rightarrow -\frac{1}{\sqrt{N}} < w_{ji} < \frac{1}{\sqrt{N}}$$

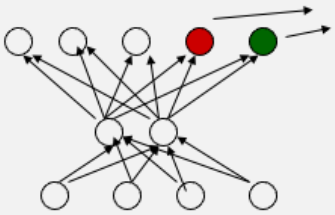
$$-\frac{1}{\sqrt{n_H}} < w_{kj} < \frac{1}{\sqrt{n_H}}$$

Some Useful Tricks

- Use of problem specific information (if known) speeds up convergence and improves generalization
- In networks designed for translation-invariant visual image classification, building in translation invariance as a constraint on the weights helps
- If we know the function to be approximated is smooth, we can build that in as part of the criterion to be minimized
  - minimize in addition to the error, the gradient of the error with respect to the inputs

Some Useful Tricks

- Manufacture training data – training networks with translated and rotated patterns if translation and rotation invariant recognition is desired
- Incorporate hints during training
- Hints are used as additional outputs during training to help shape the hidden layer representation



Hint nodes (e.g., vowels versus consonants in training a phoneme recognizer)

Some Useful Tricks

- Reducing the effective number of free parameters (degrees of freedom) helps improve generalization
- Regularization
- Preprocess the data to reduce the dimensionality of the input
  - Train an "auto encoder" neural network with output same as input, but with fewer hidden neurons than the number of inputs
  - Use the hidden layer outputs as inputs to a second network to do function approximation

Some Useful Tricks

- Choice of appropriate error function is critical – do not blindly minimize sum squared error – there are many cases where other criteria are appropriate
- Example

$$E_S(\mathbf{W}) = \sum_{p=1}^{P} \sum_{k=1}^{M} t_{kp} \ln\left(\frac{t_{kp}}{z_{kp}}\right)$$

is appropriate for minimizing the distance between the target probability distribution over the *M* output variables and the probability distribution represented by the network

Some Useful Tricks

- Interpreting the outputs as class conditional probabilities
- Use exponential output nodes

$$n_{kp} = \sum_{j=0}^{n_H} w_{kj} y_{jp}$$

$$\text{linear output } z_{kp} = \left(\frac{n_{kp}}{\sum_{l=1}^{M} n_{lp}} \right)$$

$$\text{exponential output } z_{kp} = \left(\frac{e^{n_{kp}}}{\sum_{l=1}^{M} n_{kp}} \right)$$

Bayes classification and Neural Networks

$$P(\omega_k \mid \mathbf{X}) = \frac{P(\mathbf{X} \mid \omega_k) P(\omega_k)}{\sum_{l=1}^{M} P(\mathbf{X} \mid \omega_l) P(\omega_l)}$$

$$\left.\begin{aligned} t_k(\mathbf{X}_p) = t_{kp} = 1 \text{ if } \mathbf{X}_p \in \omega_k \\ t_k(\mathbf{X}_p) = t_{kp} = 0 \text{ if } \mathbf{X}_p \notin \omega_k \end{aligned}\right\} k\text{th target output}$$

$$g_k(\mathbf{X}_p; \mathbf{W}) = k\text{th output for input } \mathbf{X}_p$$

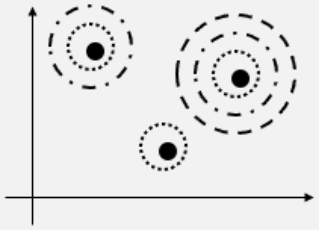
$$E_S(\mathbf{W}) = \sum_{p=1}^{P} \left(g_k(\mathbf{X}_p; \mathbf{W}) - t_{kp} \right)^2$$

We can show that the error is minimized when

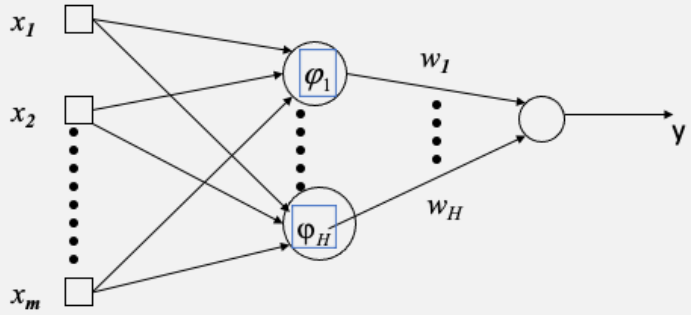
$$g_k(\mathbf{X}; \mathbf{W}) \approx P(\omega_k \mid \mathbf{X})$$

Radial-Basis Function Networks

- A function is approximated as a linear combination of radial basis functions (RBF). RBFs capture local behaviors of functions.
- RBFs represent local receptive fields



Radial Basis Function Networks

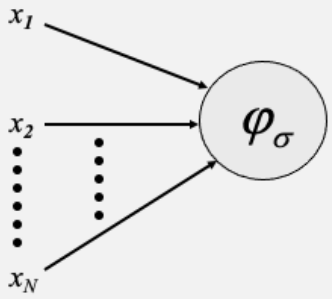


- Hidden layer applies a non-linear transformation from the input space to the hidden space.
- Output layer applies a linear transformation from the hidden space to the output space.

Example of a radial basis function

- Hidden units: use a radial basis function

$\varphi\sigma(|| \mathbf{X}- \mathbf{W}||^2)$ the output depends on the distance of the input x from the center t



$\phi_\sigma(|| \mathbf{X}- \mathbf{W}||^2)$

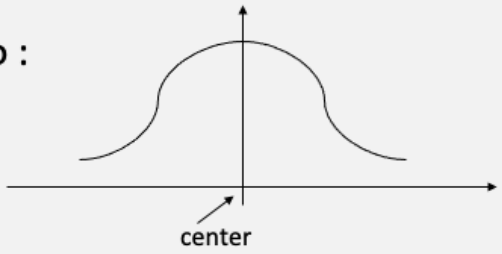
W is called center
σ is called spread
center and spread are parameters

Radial basis function

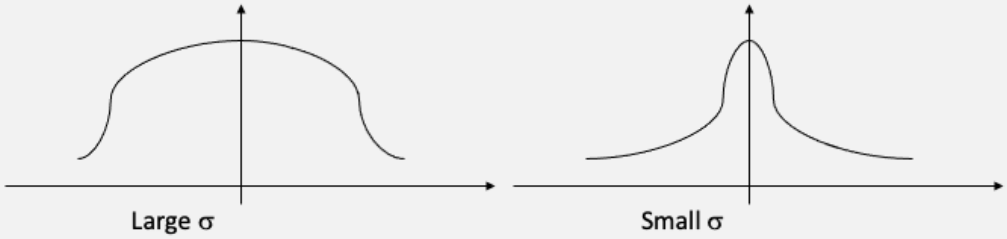
- A hidden neuron is more sensitive to data points near its center. This sensitivity may be tuned by adjusting the spread σ.
- Larger spread $\Rightarrow$ less sensitivity
- Neurons in the visual cortex have locally tuned frequency responses.

Gaussian Radial Basis Function ϕ

ϕ :



σ is a measure of how spread the curve is:



Types of radial basis functions

- Multiquadrics

$$\varphi(r) = (r^2 + c^2)^{1/2} \qquad c > 0$$

- Inverse multiquadrics

$$r = \| \mathbf{X} - \mathbf{W} \|$$

$$\varphi(r) = \frac{1}{(r^2 + c^2)^{1/2}} \qquad c > 0$$

- Gaussian functions:

$$\varphi(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right) \qquad \sigma > 0$$

RBF Learning Algorithm

$$\Delta\alpha_j = -\eta_j \frac{\partial E_S}{\partial \alpha_j}$$

$$\Delta\sigma_j = -\eta_{\sigma_j} \frac{\partial E_p}{\partial \sigma_j}$$

$$\Delta w_{ji} = -\eta_{ji} \frac{\partial E_p}{\partial w_{ji}}$$

The necessary gradients can be calculated using chain rule

RBF Learning Algorithm

$$z_{jp} = e^{-\frac{\|\mathbf{x}_p - \mathbf{w}_j\|^2}{2\sigma_j^2}}$$

$$y_p = \sum_{j=0}^{L} \alpha_j z_{jp}$$

$$E_p = \frac{1}{2}\left(t_p - y_p\right)^2$$

$$\mathbf{X}_p = \left[x_{1p} \ldots\ldots\ldots x_{Np}\right]^T$$

$$\mathbf{W}_j = \left[w_{j1} \ldots\ldots w_{jN}\right]^T$$

$$\Delta\alpha_j = -\eta_j \frac{\partial E_p}{\partial \alpha_j}$$

$$\Delta\sigma_j = -\eta_{\sigma_j} \frac{\partial E_p}{\partial \sigma_j}$$

$$\Delta w_{ji} = -\eta_{ji} \frac{\partial E_p}{\partial w_{ji}}$$

RBF Learning Algorithm

$$\Delta\alpha_j = -\eta_j \frac{\partial E_p}{\partial \alpha_j} = \eta_j (t_p - y_p) z_{jp}$$

$$\alpha_j \leftarrow \alpha_j + \eta_j (t_p - y_p) z_{jp}$$

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial y_p} \frac{\partial y_p}{\partial z_{jp}} \frac{\partial z_{jp}}{\partial w_{ji}}$$

$$= -(t_p - y_p) \alpha_j \left(\frac{z_{jp}}{\sigma_j^2} \right) (x_{ip} - w_{ji})$$

$$w_{ji} = w_{ji} + \eta_{ji} (t_p - y_p) \alpha_j \left(\frac{z_{jp}}{\sigma_j^2} \right) (x_{ip} - w_{ji})$$

RBF Learning Algorithm

$$\frac{\partial E_p}{\partial \sigma_j} = \frac{\partial E_p}{\partial y_p}\frac{\partial y_p}{\partial z_{jp}}\frac{\partial z_{jp}}{\partial \sigma_j}$$

$$= -(t_p - y_p)\alpha_j(-z_{jp})\left(\left(\frac{2}{\sigma_j}\right)(\ln z_{jp})\right)$$

$$\sigma_j \leftarrow \sigma_j - \eta_j(t_p - y_p)\alpha_j(z_{jp})\left(\left(\frac{2}{\sigma_j}\right)(\ln z_{jp})\right)$$

Generalized RBF Learning Algorithm

Some useful facts

$$\|V\|^2 = V^T V \text{ (norm)}$$

$$\|V\|_C^2 = (CV)^T (CV) = V^T C^T CV \text{ (weighted norm)}$$

$$\|V\|_C^2 = \|V\|^2 \text{ if } C^T C = \text{identity matrix}$$

$$\frac{d}{d\mathbf{X}}(A\mathbf{X}) = A$$

$$\frac{d}{d\mathbf{X}}(\mathbf{X}^T A\mathbf{X}) = 2A\mathbf{X} \text{ (when A is a symmetric matrix)}$$

$$\frac{d}{dA}(\mathbf{X}^T A\mathbf{X}) = \mathbf{X}^T\mathbf{X}$$

Generalizd RBF Learning Algorithm

$$z_{jp} = e^{-\frac{1}{2}(\mathbf{x}_p - \mathbf{w}_j)^T \Sigma_j (\mathbf{x}_p - \mathbf{w}_j)}$$

$$y_p = \sum_{j=0}^{L} \alpha_j z_{jp}$$

$$E_p = \frac{1}{2}(t_p - y_p)^2$$

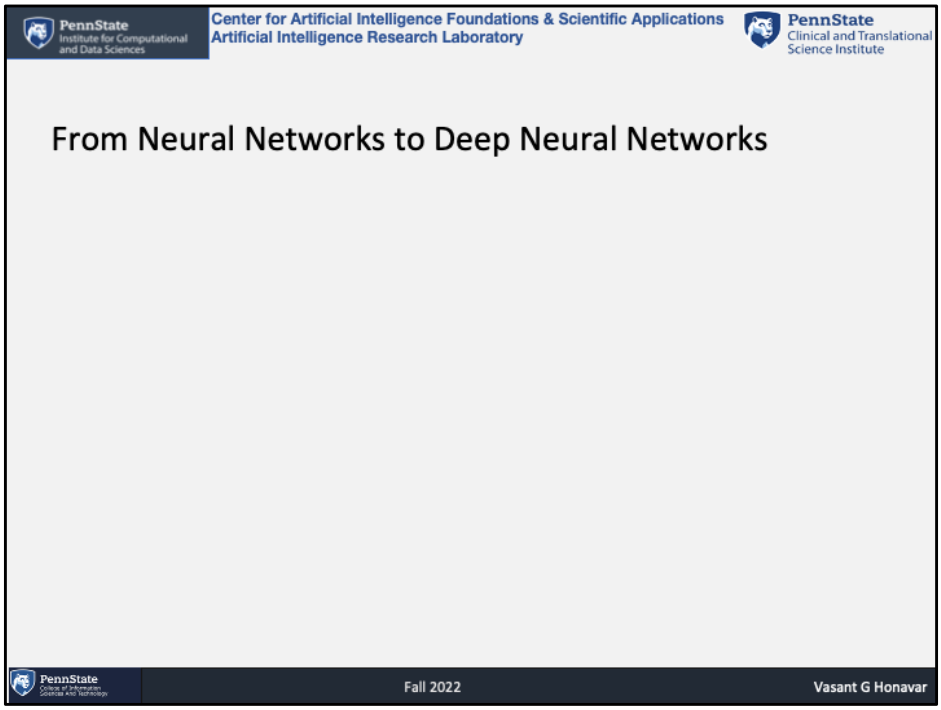
$$\mathbf{X}_p = [x_{1p} \ldots\ldots\ldots x_{Np}]^T$$

$$\mathbf{W}_j = [w_{j1} \ldots\ldots w_{jN}]^T$$

Exercise: Derive the update equations for $\mathbf{w}_j$, Σ_j and α_j

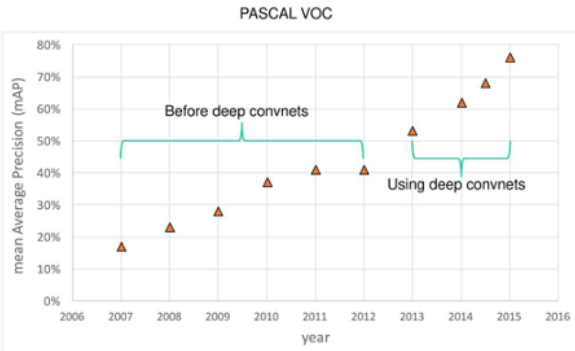
RBF Learning Algorithm

- Initialize the parameters -- centers of the hidden neurons are typically initialized to coincide with a subset of the training set
- Use gradient descent to adjust the parameters using the training data until the desired performance criterion is satisfied

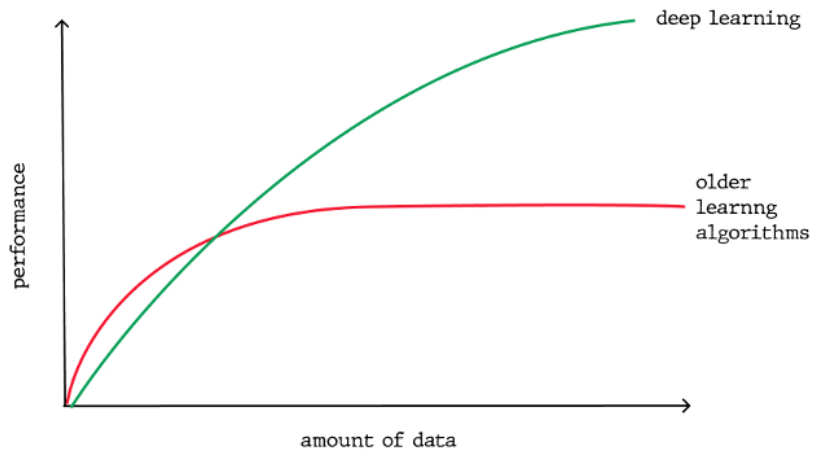

PennState
Institute for Computational and Data Sciences
Center for Artificial Intelligence Foundations & Scientific Applications
Artificial Intelligence Research Laboratory
PennState
Clinical and Translational Science Institute
From Neural Networks to Deep Neural Networks
PennState
Fall 2022
Vasant G Honavar

Why deep learning?

Recent developments in object detection



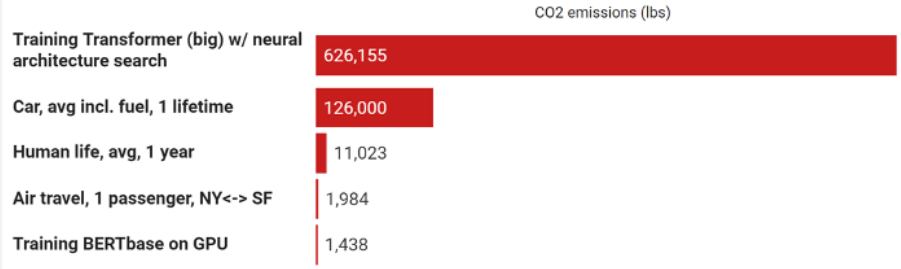
Why or why not deep learning?



Why not deep learning?

Carbon footprint comparison

Source: Strubell et al, 2019.

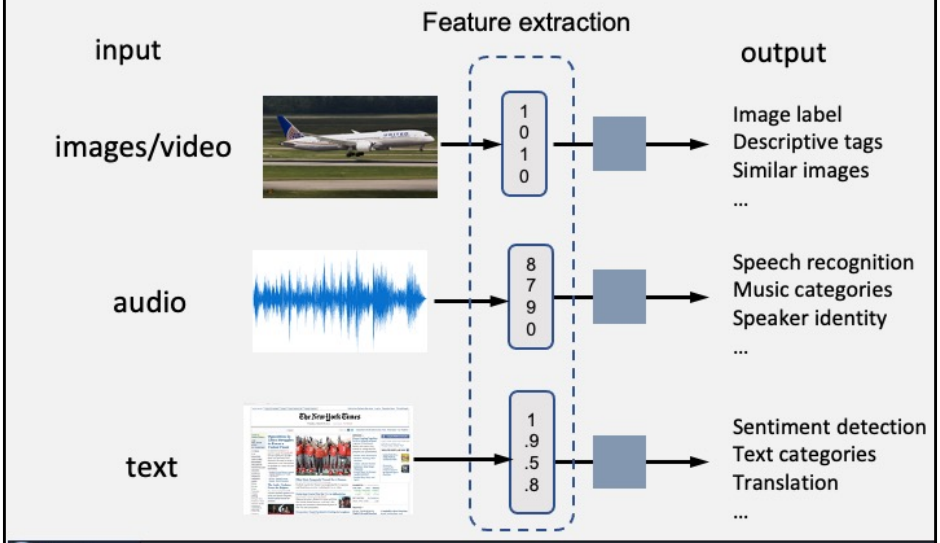


Reconstructed from: http://arxiv.org/abs/1906.02243

Typical goal of machine learning

input		output
images/video		Image label Descriptive tags Similar images …
audio		Speech recognition Music categories Speaker identity …
text		Sentiment detection Text categories Translation …

Machine learning pipeline



Object classification

ML trained model → "airplane"

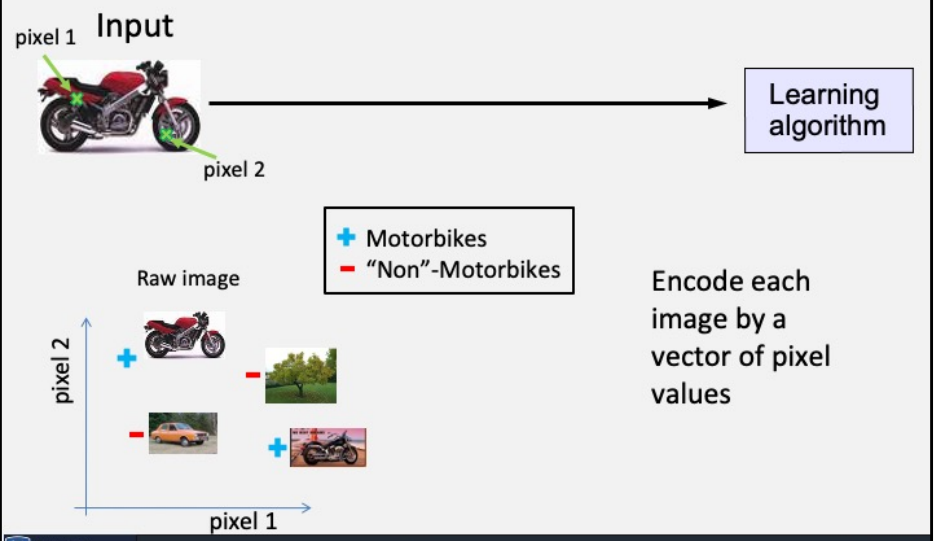
Why is object classification hard?

What you see

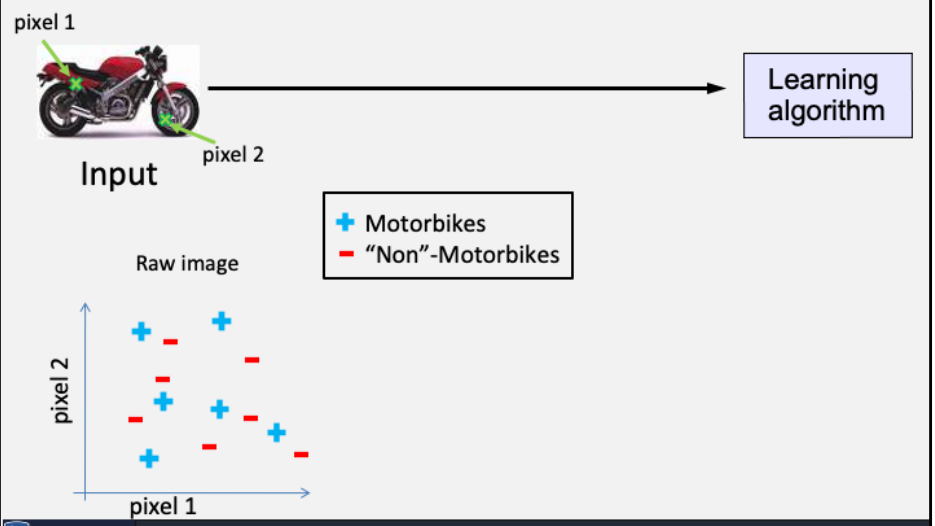
What the machine sees

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

Pixel-based representation



Pixel-based representation



Feature based representation

Input

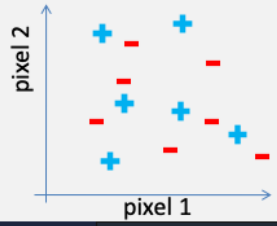
Feature representation

E.g., Does it have Handlebars? Wheels?

Learning algorithm

Motorbikes
"Non"-Motorbikes

Raw image

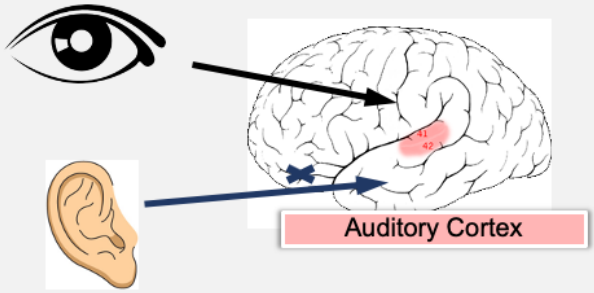


Features

Traditional Machine learning

- Machine learning 30 years ago relied on feature engineering
- Feature engineering is hard!
- It would be nice if we can avoid ad hoc feature engineering
- Kernel machines offer one solution
- Deep learning offers another solution

The brain: inspiration for deep learning



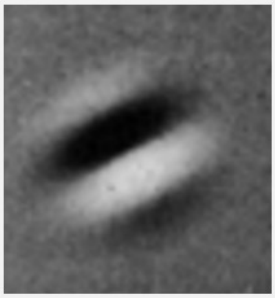
Auditory cortex learns to see!

Sur, M., Garraghty, P.E. and Roe, A.W., 1988. Experimentally induced visual projections into auditory thalamus and cortex. Science, 242(4884), pp.1437-1441.

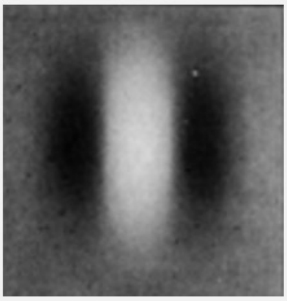
First stage of visual processing: V1

V1 is the first stage of visual processing in the brain.

Neurons in V1 act as edge detectors:



Neuron #1 of visual cortex (model)



Neuron #2 of visual cortex (model)

http://www.ldeo.columbia.edu/4d4/wavelets/dm.html

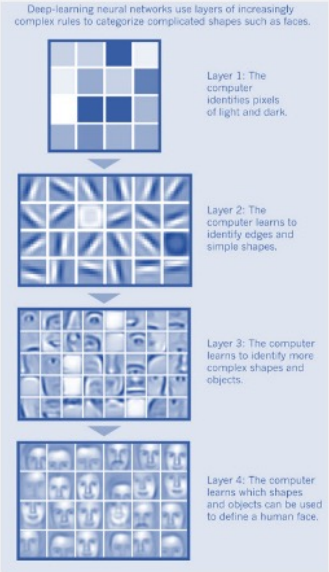
Basic idea of deep learning

- Also referred to as representation learning or unsupervised feature learning (with subtle distinctions)
- Is there some way to extract **meaningful features** from data **even without knowing the task** to be performed?
- Then, stack such representation learning layers to obtain 'deep' networks

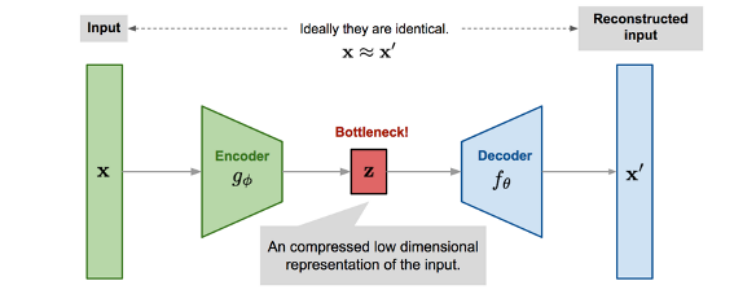
Brain inspired computer vision

Training set: Labeled images of faces.

Early work:
Uhr and students (recognition cones)
Fukushima (neocognitron)



Autoencoder



x'

A feedforward neural network that learns an information preserving representation of its input

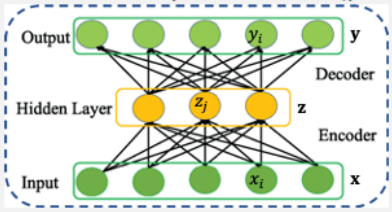
- The input-to-hidden part corresponds to an encoder
- The hidden-to-output part corresponds to a decoder
- Input and output are of the same dimension

Autoencoders

- Training a 3-layer linear autoencoder with N inputs (plus the constant input $x_{0p} = 1\ \forall p$) and 1 hidden layer of size M (plus $z_{0p} = 1\ \forall p$)

$$z_{jp} = \sum_i w_{ji}\, x_{ip}$$

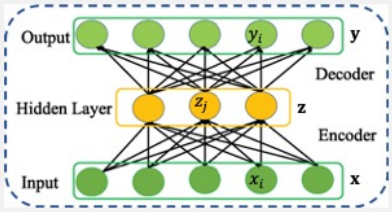
$$y_{ip} = \sum_j u_{ij}\, z_{jp}$$



Reconstruction loss $E = \sum_p E_p = \frac{1}{2}\sum_p \sum_i \left(x_{ip} - y_{ip}\right)^2$

Use backpropagation algorithm to learn $\mathbf{w}_j$ and $\mathbf{u}_i$

Training Autoencoders



$$z_{jp} = \sum_i w_{ji}\, x_{ip}$$

$$y_{ip} = \sum_j u_{ij}\, z_{jp}$$

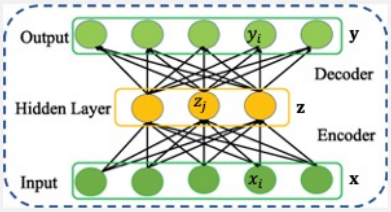
$$E = \sum_p E_p = \frac{1}{2} \sum_p \sum_i \left(x_{ip} - y_{ip}\right)^2$$

$$u_{ij} \leftarrow u_{ij} - \frac{\partial E}{\partial u_{ij}}$$

$$\frac{\partial E}{\partial u_{ij}} = \sum_p \frac{\partial E}{\partial y_{ip}} \frac{\partial y_{ip}}{\partial u_{ij}} = -\sum_p \left(x_{ip} - y_{ip}\right) z_{jp}$$

E_p
↓
y_{ip}
↓
u_{ij}

Training Autoencoders

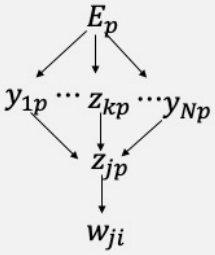


$$z_{jp} = \sum_i w_{ji}\, x_{ip}$$

$$y_{ip} = \sum_j u_{ij}\, z_{jp}$$

$$E = \sum_p E_p = \frac{1}{2} \sum_p \sum_i \left(x_{ip} - y_{ip}\right)^2$$

$$w_{ji} \leftarrow w_{ji} - \frac{\partial E}{\partial w_{ji}}$$



$$\frac{\partial E}{\partial w_{ij}} = \sum_p \sum_i \frac{\partial E}{\partial y_{ip}} \frac{\partial y_{ip}}{\partial z_{jp}} \frac{\partial z_{jp}}{\partial w_{ij}} = -\sum_p \sum_i \left(x_{ip} - y_{ip}\right) u_{ij}\, x_{ip}$$

Autoencoders

- Exercises
  - Derive the weight update equations for autoencoders when the hidden nodes compute the sigmoid (as opposed to linear function) applied to the weighted sum of the inputs.
  - Derive the weight update equations for the autoencoders when the inputs are binary as opposed to continuous valued. Hint: we want to minimize the cross-entropy between the inputs $\mathbf{x}$ and the reconstruction $\mathbf{y}$ i.e. $E = \sum_p E_p$ where $E_p = -\sum_i \left(x_{ip} \log y_{ip} + \left(1 - x_{ip}\right) \log\left(1 - y_{ip}\right)\right)$

Autoencoders

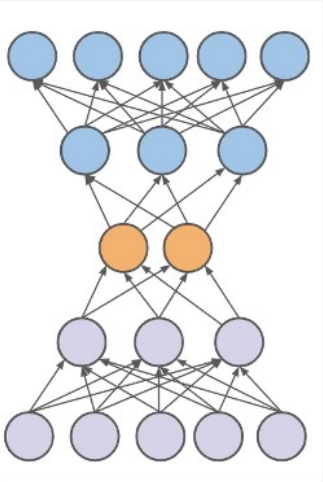
- What is the point of learning a mapping that reproduces the input?
- If the hidden layer has lower dimension than the input, the network is forced to learn a low-dimensional information preserving representation of the input
- Once such a representation is learned, we can discard the decoder and use the encoder to extract features from the input data that can then be used to train a classification or regression model

Autoencoders

- The autoencoder considered above is a linear autoencoder because it used a linear function in the hidden layer
- Such a linear autoencoder can be shown to learn an encoding that mimics principal component analysis (the number of hidden nodes correspond to the number of principal components)
- We can obtain a non-linear autoencoder by using the sigmoid activation function in the hidden layer
- We can use multiple hidden layers to learn a non-linear autoencoder

Stacked autoencoders

- We can learn hierarchical representations of input data using multi-layer nonlinear encoder
- But as we increase the number of layers, training becomes slow
- Solution: Learn a multi-layer encoder one layer at a time
  - How?

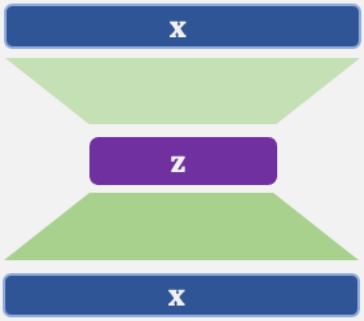


Stacked encoders

- Learn a multi-layer encoder one layer at a time
  - First learn an autoencoder $\mathbf{x} \rightarrow \mathbf{z} \rightarrow \mathbf{x}$
  - Strip off the decoder $\mathbf{z} \rightarrow \mathbf{x}$ and keep $\mathbf{x} \rightarrow \mathbf{z}$
  - Now learn an autoencoder $\mathbf{z} \rightarrow \mathbf{u} \rightarrow \mathbf{z}$
  - Strip off the decoder $\mathbf{u} \rightarrow \mathbf{z}$ and keep $\mathbf{z} \rightarrow \mathbf{u}$
  - Stack the encoders $\mathbf{x} \rightarrow \mathbf{z}$ and $\mathbf{z} \rightarrow \mathbf{u}$ to obtain the stacked encoder $\mathbf{x} \rightarrow \mathbf{z} \rightarrow \mathbf{u}$
  - Repeat the preceding steps if more layers are desired

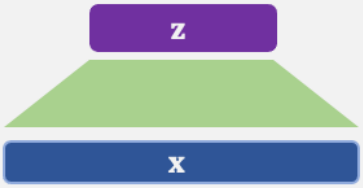
Learning stacked encoders

First learn an autoencoder $\mathbf{x} \rightarrow \boldsymbol{z} \rightarrow \mathbf{x}$



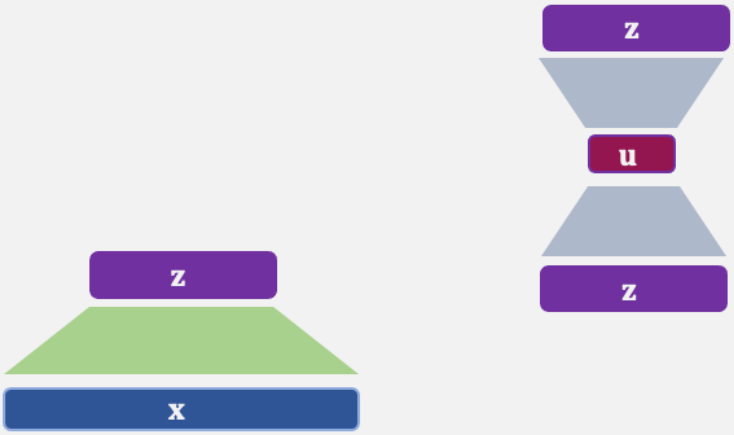
Learning stacked encoders

Strip off the decoder $\mathbf{z} \rightarrow \mathbf{x}$ and keep the encoder $\mathbf{x} \rightarrow \mathbf{z}$



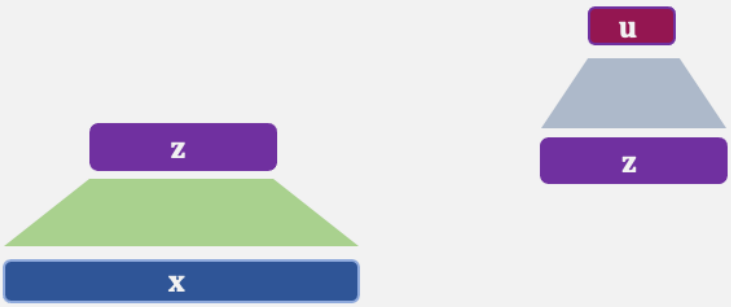
Learning stacked encoders

Learn an autoencoder $\mathbf{z} \rightarrow \mathbf{u} \rightarrow \mathbf{z}$



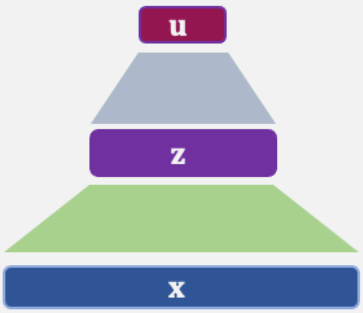
Learning stacked encoders

Strip off the decoder $\mathbf{u} \rightarrow \mathbf{z}$ and keep the encoder $\mathbf{z} \rightarrow \mathbf{u}$



Learning stacked encoders

Stack the encoders $\mathbf{x} \rightarrow \mathbf{z}$ and $\mathbf{z} \rightarrow \mathbf{u}$ to obtain the stacked encoder $\mathbf{x} \rightarrow \mathbf{z} \rightarrow \mathbf{u}$



Stacked encoders

- Learn a multi-layer encoder one layer at a time
  - First learn an autoencoder $\mathbf{x} \rightarrow \mathbf{z} \rightarrow \mathbf{x}$
  - Strip off the decoder $\mathbf{z} \rightarrow \mathbf{x}$ and keep $\mathbf{x} \rightarrow \mathbf{z}$
  - Now learn an autoencoder $\mathbf{z} \rightarrow \mathbf{u} \rightarrow \mathbf{z}$
  - Strip off the decoder $\mathbf{u} \rightarrow \mathbf{z}$ and keep $\mathbf{z} \rightarrow \mathbf{u}$
  - Stack the encoders $\mathbf{x} \rightarrow \mathbf{z}$ and $\mathbf{z} \rightarrow \mathbf{u}$ to obtain the stacked encoder $\mathbf{x} \rightarrow \mathbf{z} \rightarrow \mathbf{u}$
  - Repeat the preceding steps if more layers are desired

Autoencoders

- Autoencoders are only able to compress or extract useful features from data that are similar to training data
- The decoded output will be a noisy reconstruction of input
- Can be trained in an unsupervised fashion on large data sets
- The resulting encoding can be used to extract useful features from for supervised training of classifiers from much smaller data sets

Can autoencoders overfit their training data?

- Like other neural networks, autoencoders are susceptible to overfitting when the network capacity (the number of weights) is is too large given the amount of training data
- Overfitting can be mitigated by a combination of
  - Information bottlenecks – code with fewer dimensions than the input
  - Training to denoise
  - Regularization

Information bottleneck

- Suppose the training data consist of $N \times N$ images and the size of the code is $M \ll N^2$
- Then the code is too compact to perfectly reconstruct the images
- In such a setting the autoencoder learns an encoding that captures the important features of the training data that are adequate for approximate reconstruction of the training data

Denoising autoencoders

- The basic autoencoder minimizes the loss between $\mathbf{x}$ and the reconstruction $g\big(f(\mathbf{x})\big)$ where f is the encoder and g the decoder.
- Denoising autoencoders minimizes the loss between $\mathbf{x}$ and $g\big(f(\mathbf{x}+\mathbf{w})\big)$, where $\mathbf{w}$ is gaussian random noise
- Input and output of a denoising autoencoder

- Noise added to the input forces a denoising autoencoder to learn a mapping from noise perturbed training data to noise-free training data

Sparse autoencoders

- Add a term to the loss function that forces sparse encoding of training data, e.g., sum of absolute values of the activations of the nodes in the hidden layer
- As a result, different subsets of the hidden nodes are activated by different inputs
- This is often combined by penalties for large weights (e.g., the square of the norm of the weight vectors).

Contractive autoencoders

- Contractive autoencoders make the *feature extraction function* (ie. encoder) robust in the presence of small perturbations of the input
- How?
- Instead of minimizing the loss, minimize the loss plus a term proportional to the magnitude of $\nabla_{\mathbf{x}} E$ i.e., the gradient of the loss with respect to the input $\mathbf{x}$

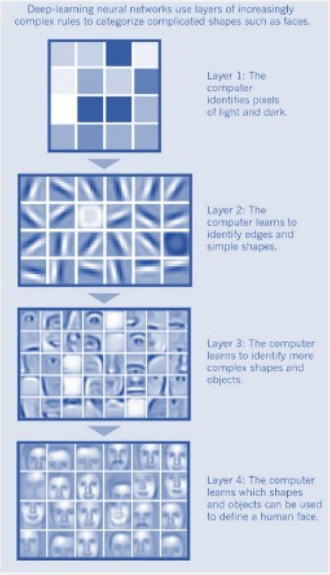
Stochastic Encoders and Decoders

- Modern autoencoders use stochastic mappings
- We can generalize the notion of the encoding and decoding functions to encoding and decoding distributions
- The resulting encoders are called variational autoencoders (detailed discussion omitted)

Brain inspired computer vision

Training set: Labeled images of faces.

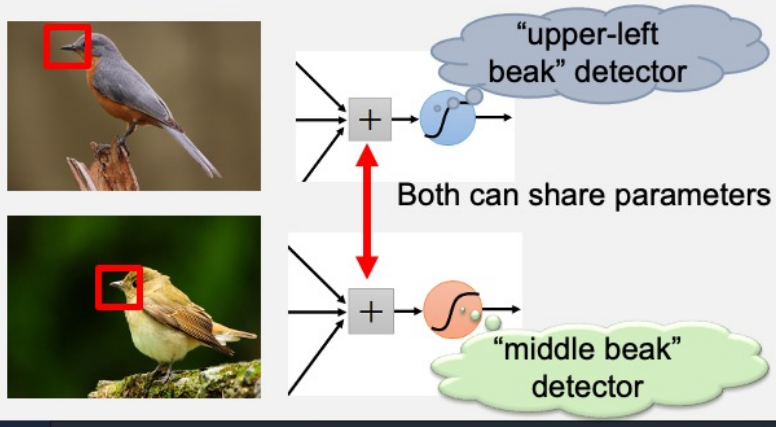
Early work:
Uhr and students (recognition cones)
Fukushima (neocognitron)



Vision

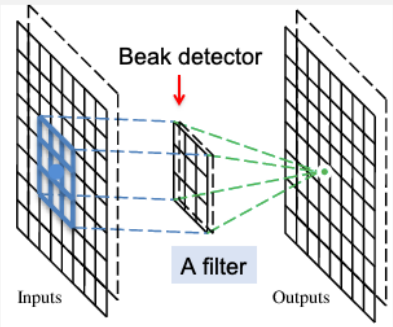
- Images have features at different spatial scales
- Need to be able to recognize objects regardless of where they appear in an image
- How can we modify neural networks to accommodate these considerations?
- Extract local features from images

Translation invariance



Solution: convolutional neural network

- A CNN is a neural network with one or more convolutional layers
- A convolutional layer performs convolutional operations.



Convolution

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

These are the network parameters to be learned.

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

⋮ ⋮

Each filter detects a small pattern (3 x 3).

Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

If stride =1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Dot product → 3 -1

6 x 6 image

Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

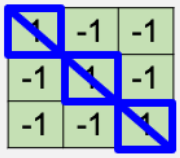
If stride =2

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

3 -3

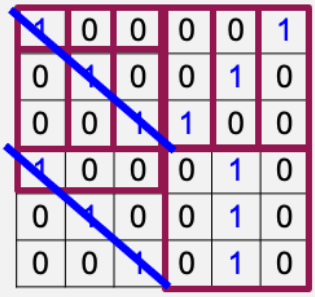
6 x 6 image

Convolution

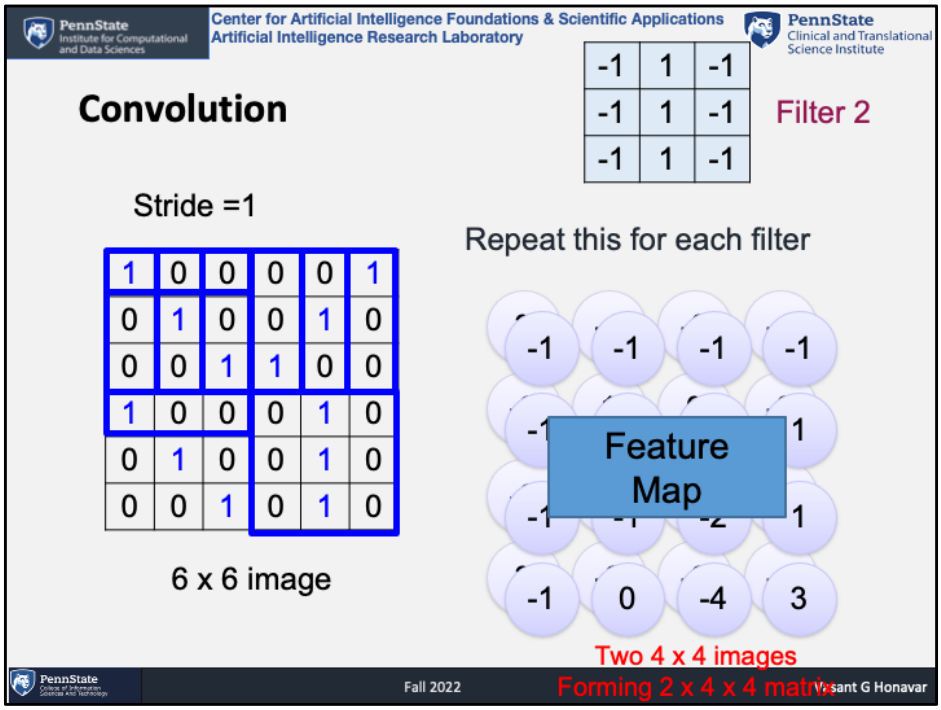


Filter 1

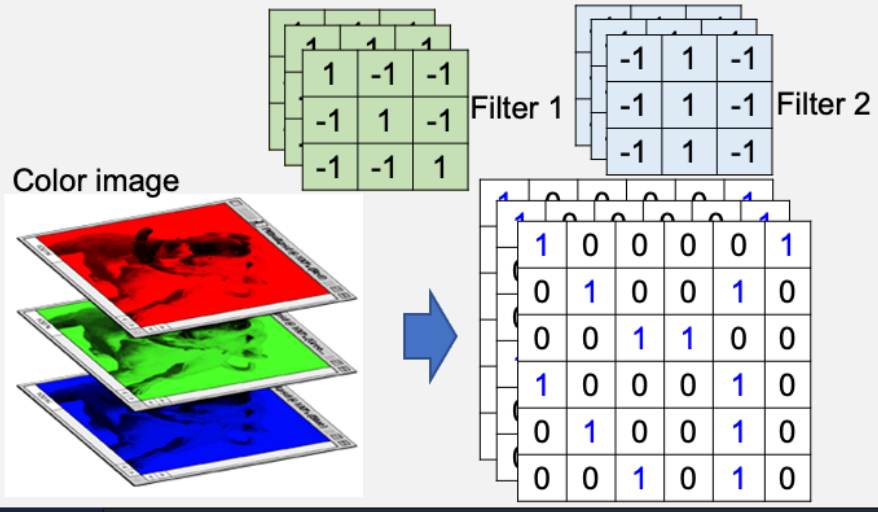
Stride =1



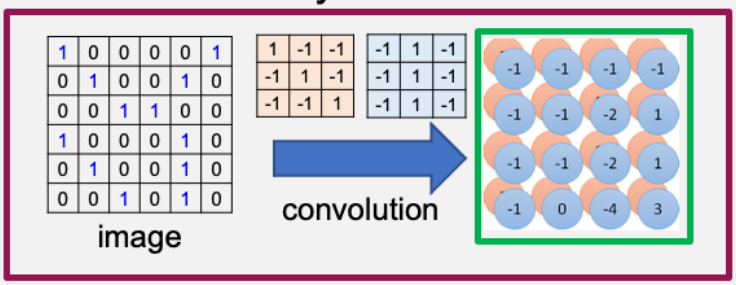
6 x 6 image


PennState Institute for Computational and Data Sciences
Center for Artificial Intelligence Foundations & Scientific Applications
Artificial Intelligence Research Laboratory
PennState Clinical and Translational Science Institute
Convolution
-1 1 -1
-1 1 -1
-1 1 -1
Filter 2
Stride =1
1 0 0 0 0 1
0 1 0 0 1 0
0 0 1 1 0 0
1 0 0 0 1 0
0 1 0 0 1 0
0 0 1 0 1 0
6 x 6 image
Repeat this for each filter
Feature Map
-1 -1 -1 -1
-1 0 -4 3
Two 4 x 4 images
Forming 2 x 4 x 4 matrix
PennState College of Information Sciences And Technology
Fall 2022
Vasant G Honavar

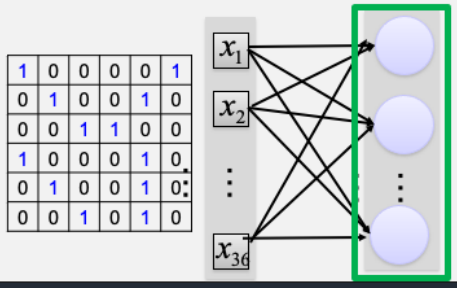
Color image: RGB 3 channels

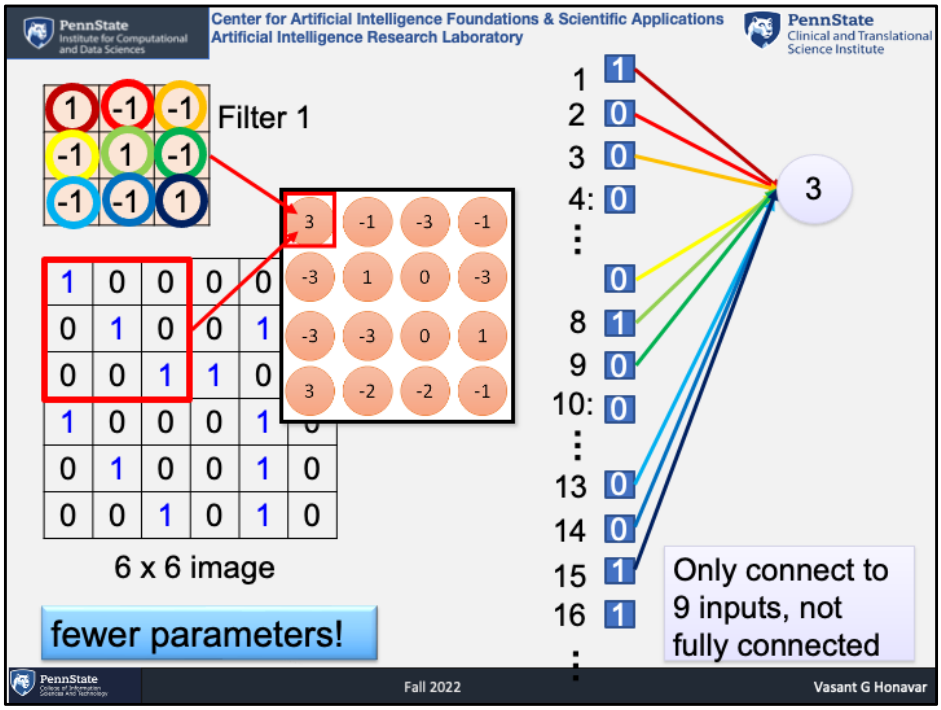


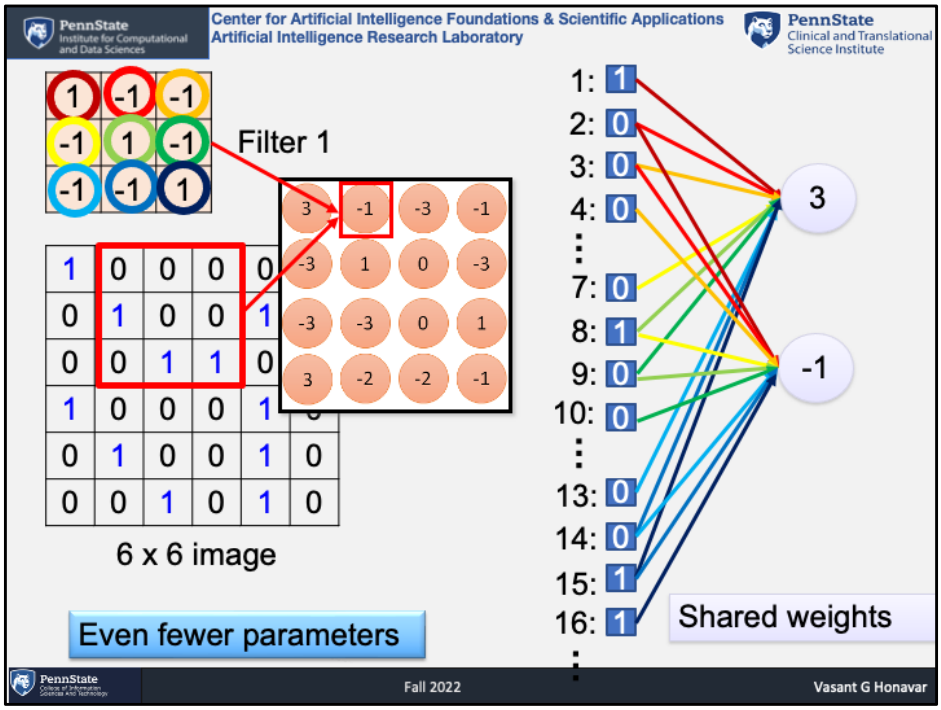
Convolution v.s. Fully Connected



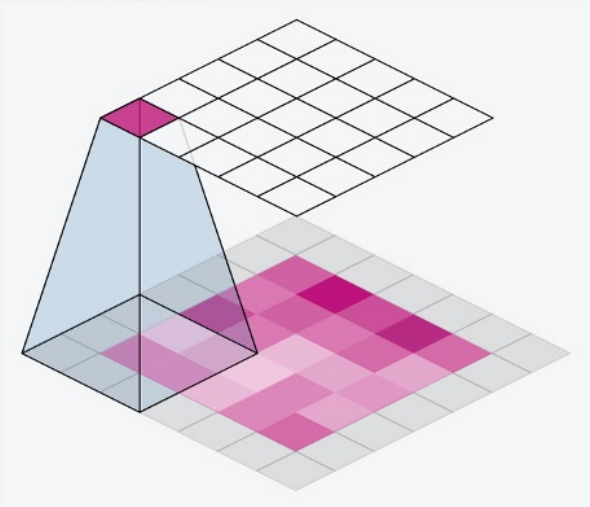
Fully-connected




PennState Institute for Computational and Data Sciences
Center for Artificial Intelligence Foundations & Scientific Applications
Artificial Intelligence Research Laboratory
PennState Clinical and Translational Science Institute
Filter 1
6 x 6 image
fewer parameters!
Only connect to 9 inputs, not fully connected
3
Fall 2022
Vasant G Honavar


PennState Institute for Computational and Data Sciences
Center for Artificial Intelligence Foundations & Scientific Applications
Artificial Intelligence Research Laboratory
PennState Clinical and Translational Science Institute
Filter 1
6 x 6 image
Even fewer parameters
Shared weights
3
-1
Fall 2022
Vasant G Honavar

Convolution at work



Pooling

Max

3	1	1	3
2	5	0	2
1	4	2	1
4	7	2	4

=

5	3
7	4

Avg

3	1	1	3
2	5	0	2
1	4	2	1
4	7	2	4

=

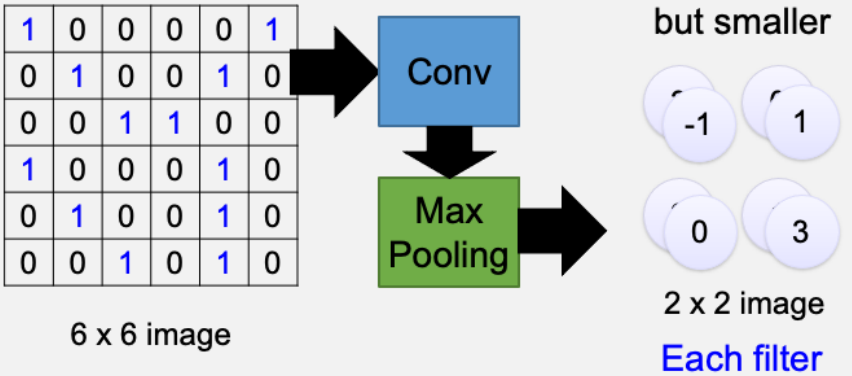
2.75	1.5
4	2.25

Pooling

- Down-samples the feature maps
- Reduces resolution
- Summarizes the features in a region
- Reduces the number of parameters to be learned
- Combines simpler features into more complex ones

Pooling can be traced back to early work on computer vision, e.g., recognition cones (Uhr, Li, Honavar), pyramids (Rosenfeld, Dyer, Tanimoto, Tsotsos)

Max Pooling

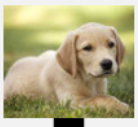


6 x 6 image

2 x 2 image

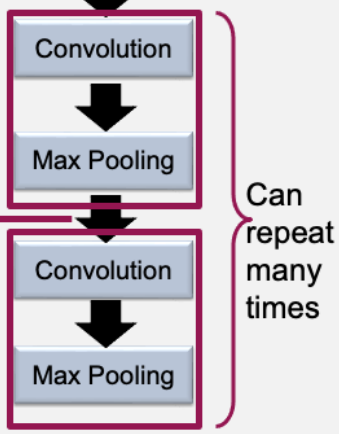
Each filter is a channel

The structure of CNN

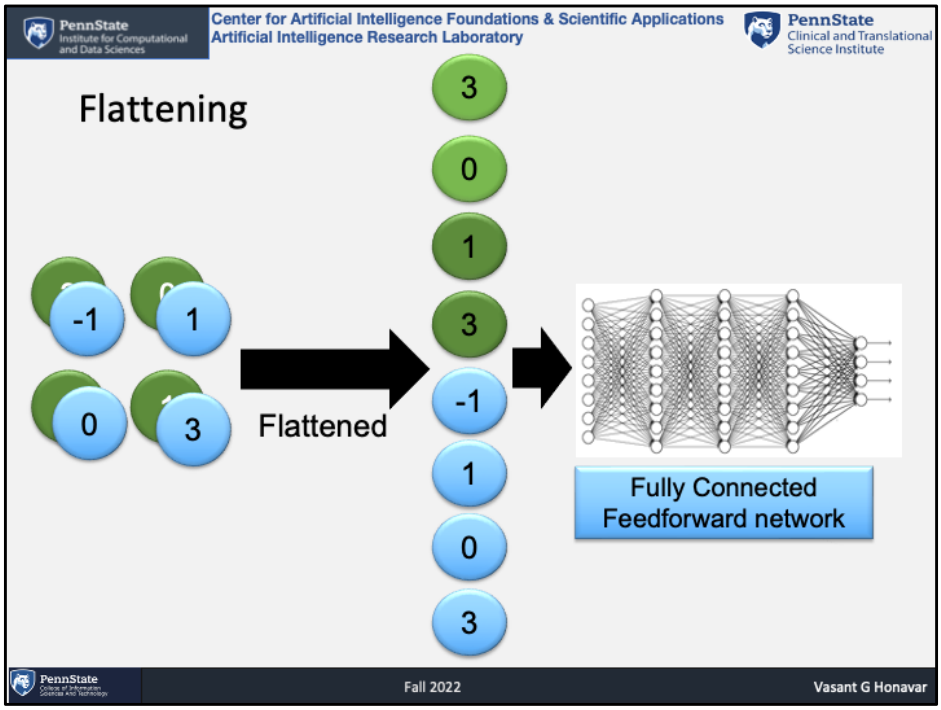


A new image

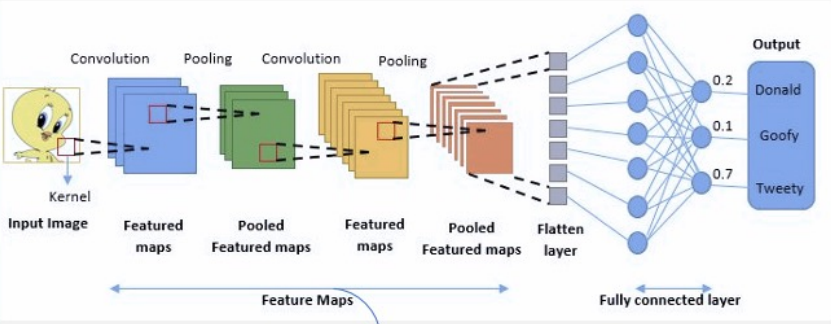
Smaller than the original image
The number of channels is the number of filters



Can repeat many times


PennState Institute for Computational and Data Sciences
Center for Artificial Intelligence Foundations & Scientific Applications
Artificial Intelligence Research Laboratory
PennState Clinical and Translational Science Institute
Flattening
-1
1
0
3
Flattened
3
0
1
3
-1
1
0
3
Fully Connected Feedforward network
PennState College of Information Sciences And Technology
Fall 2022
Vasant G Honavar

Complete convolutional neural network (CNN)



- Can be repeated many times
- Modern deep neural nets can have tens or hundreds of feature maps as well as layers

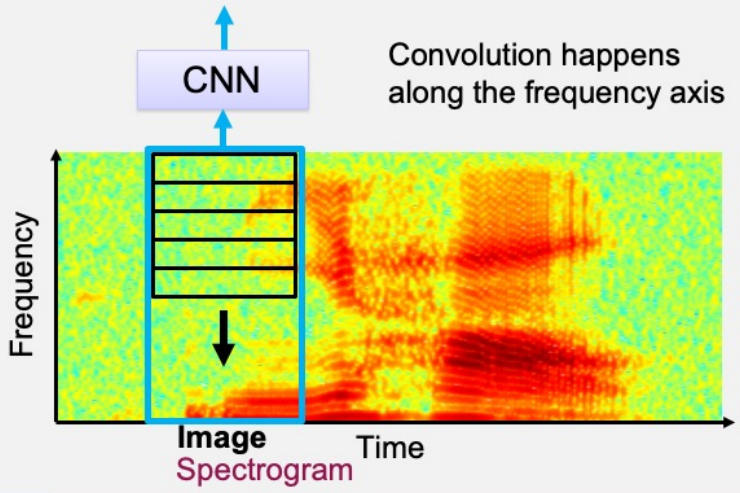
A CNN compresses a fully connected network

- Reduces the number of connections
- Shared weights on the edges
- Max pooling further reduces the complexity

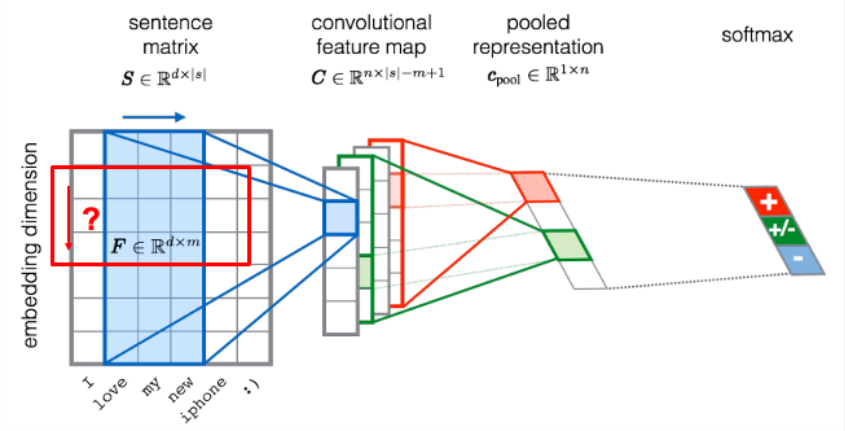
CNN Applications

- Image classification
- Speech recognition
- Biomolecular sequence classification
- Text classification

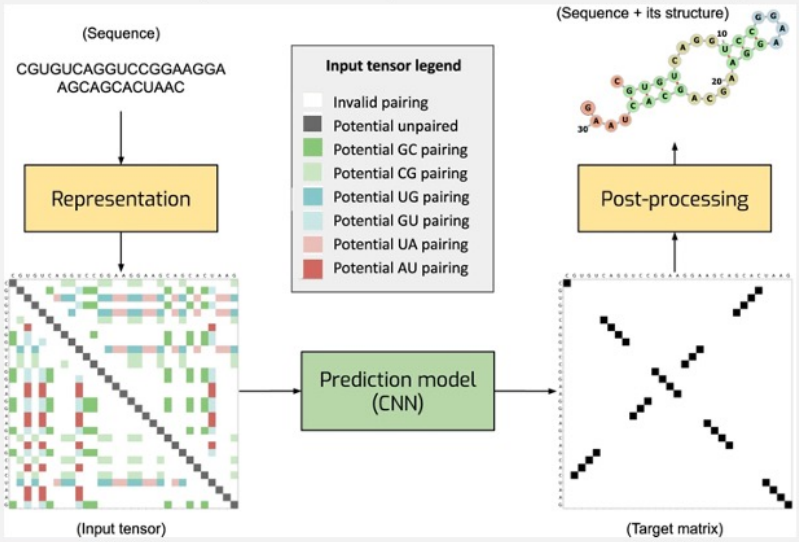
Speech recognition using CNN



Text classification using CNN



RNA secondary structure prediction using deep CNN

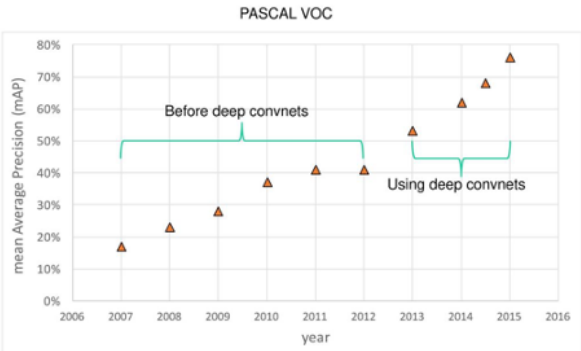


Not covered in this course

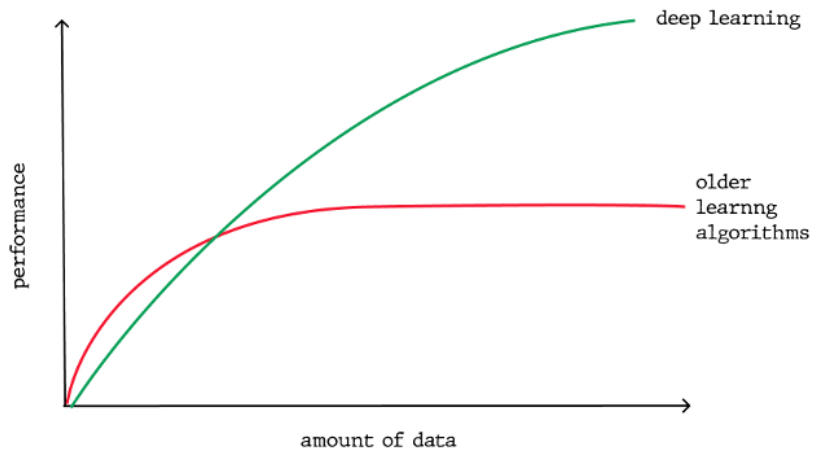
- Foundation models
- Graph neural networks
- Transformers
- Adversarial learning
- Deep reinforcement learning
- Causal inference
- Algorithmic bias
- …

Why deep learning?

Recent developments in object detection



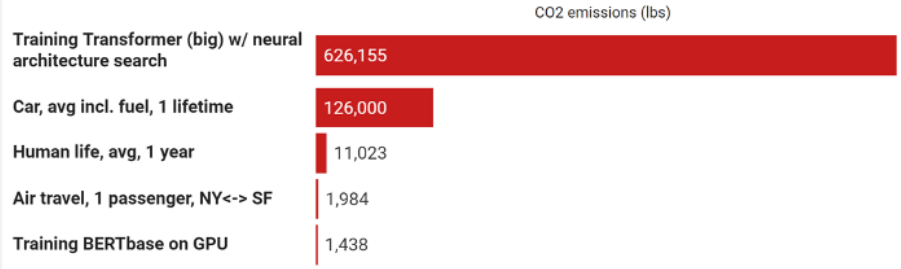
Why or why not deep learning?



Why not deep learning?

Carbon footprint comparison

Source: Strubell et al, 2019.



Reconstructed from: http://arxiv.org/abs/1906.02243

Concluding remarks

- Unsupervised representation learning offers a powerful mechanism for learning useful features from complex data
- Multi-layer deep networks, given sufficient data, can be trained to perform many tasks that were once beyond the reach of machine learning
- Powerful tools Keras, Pytorch allow rapid prototyping of deep learning solutions
- But deep learning, as it is practiced today, is an environmentalist's nightmare – need better approaches
- Deep learning systems can be easily fooled by adversarial data samples - need learning methods that are robust to adversarial attacks
- Deep learning produces black boxes that are hard to understand and explain –need better tools to explain the results of deep learning