Introduction to Artificial Neural Networks

STAT5241 Section 2

Statistical Machine Learning

Xiaofei Shi

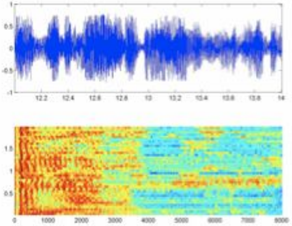
Images & Video

flickr
Google
YouTube

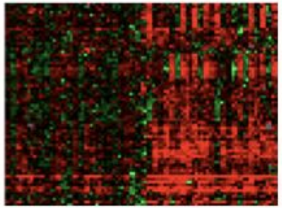
Text & Language

WIKIPEDIA
The Free Encyclopedia
REUTERS
AP Associated Press

Speech & Audio



Gene Expression

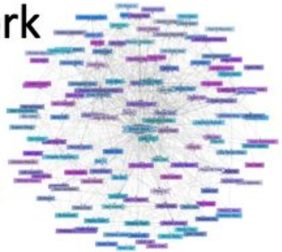


Product Recommendation

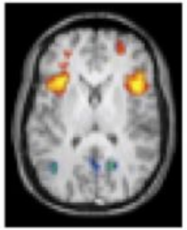
amazon
NETFLIX
ebay

Relational Data/ Social Network

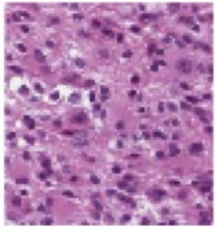
facebook
twitter



fMRI



Tumor region

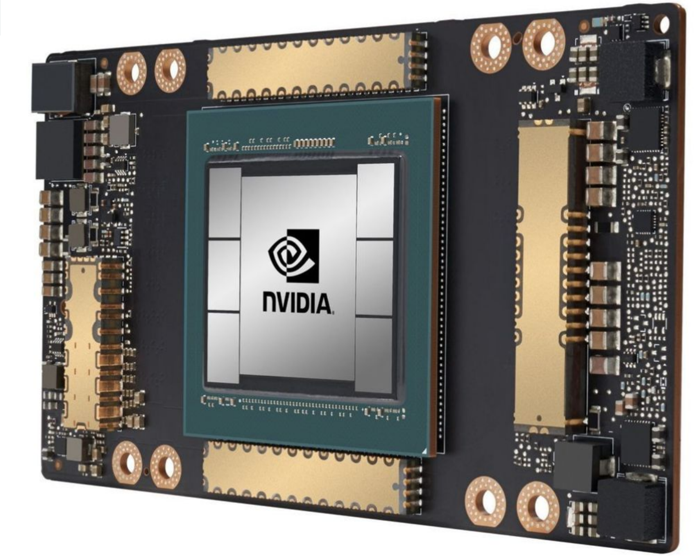


Overview

- A neural network is a supervised learning method. It can be applied to both regression and classification problems.
- The main idea is to extract linear combinations of the inputs as derived features, and then model the target as a nonlinear function of these features.
- The nonlinear transformation contributes to the model flexibility.
- Today, we will focus on the most widely used ``vanilla'' neural net, also called the single hidden layer feedforward neural networks.

COLUMBIA UNIVERSITY

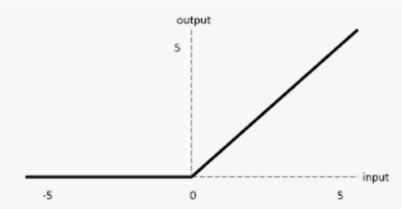
Why deep architecture works



- Enough computational power

- Large training dataset:
  MNIST, CI-FAR, IMAGENET

- Architectures adapted to
  current computational methods



COLUMBIA UNIVERSITY

Overview

- Derived features Z_m are obtained by applying the *activation function* σ to linear combinations of the inputs:

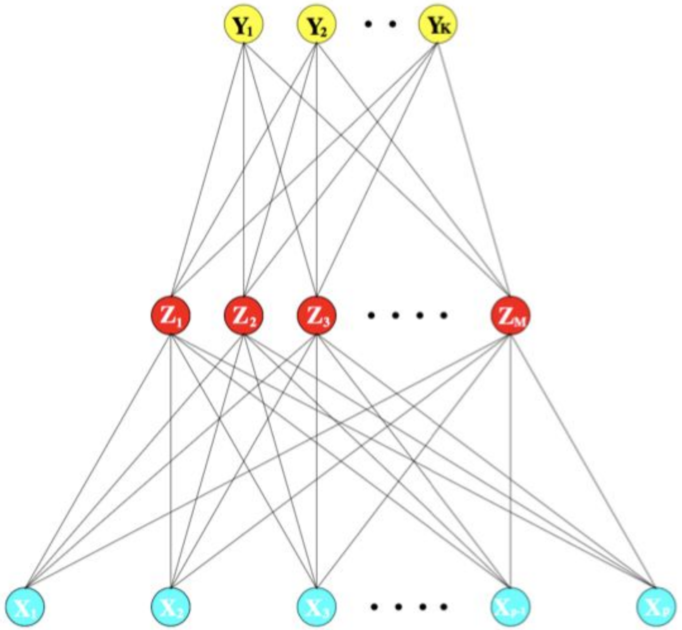
$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X), m = 1, \ldots, M.$$

- The target Y_k (or T_k in the figure) is modeled as a function of linear combinations of the Z_m:

$$T_k = \beta_{0k} + \beta_k^T Z, \quad k = 1, \ldots, K.$$

- The output function $g_k(T)$ allows a final transformation of the vector of outputs T:

$$f_k(X) = g_k(T), \quad k = 1, \ldots, K.$$

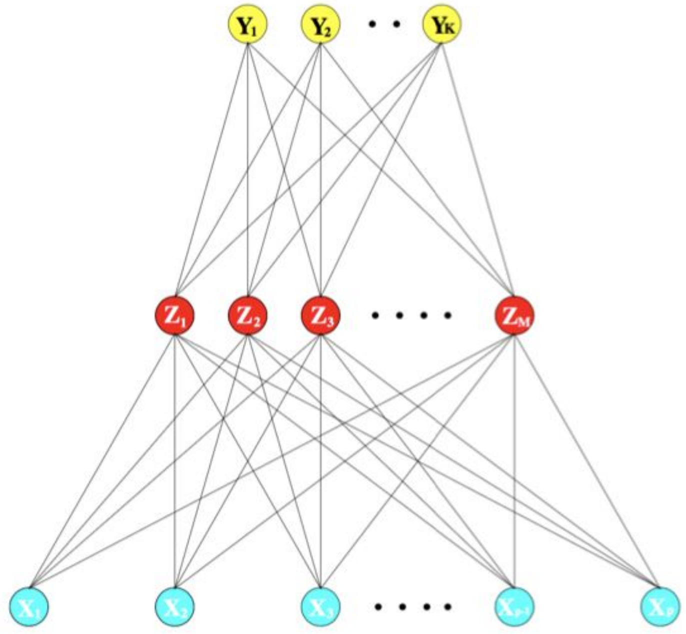


Schematic of a single hidden layer, feed-forward neural network

COLUMBIA UNIVERSITY

Artificial neurons

- Each artificial neuron has inputs and produces a single output which can be sent to multiple other neurons. The inputs can be the feature values of a sample of external data, such as images or documents, or they can be the outputs of other neurons.
- The outputs of the final output neurons of the neural net accomplish the task.



Schematic of a single hidden layer, feed-forward neural network

Artificial neurons

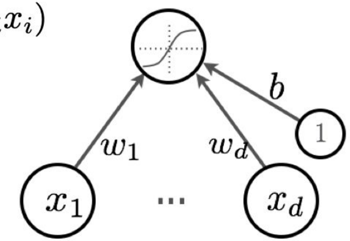
- Neuron pre-activation (or input activation):

$$a(\mathbf{x}) = b + \sum_i w_i x_i = b + \mathbf{w}^\top \mathbf{x}$$

- Neuron output activation:

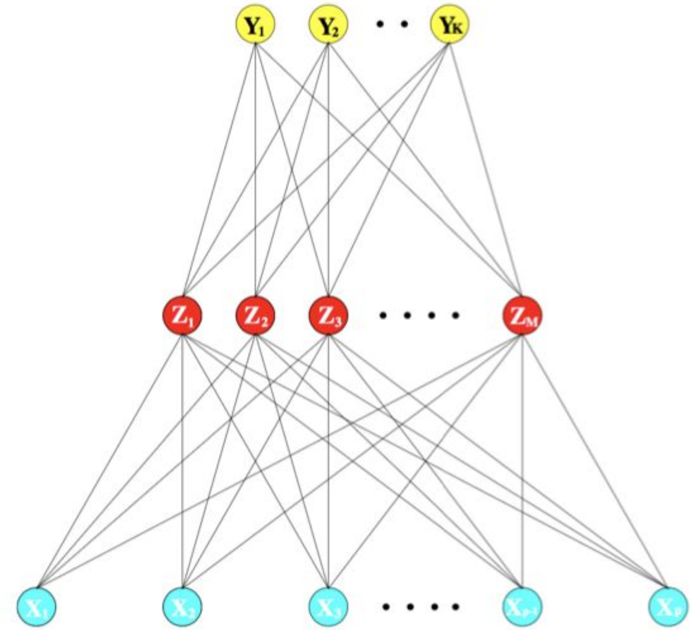
$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

where
$\mathbf{w}$ are the weights (parameters)
b is the bias term
$g(\cdot)$ is called the activation function



Activation functions

- An activation function of a node defines the output of that node given an input or set of inputs.
- Usually nonlinear.



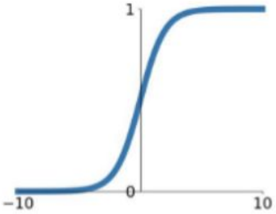
Schematic of a single hidden layer, feed-forward neural network

COLUMBIA UNIVERSITY

Activation functions

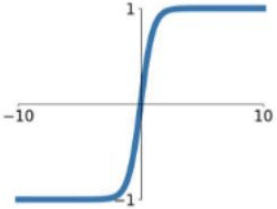
Sigmoid

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



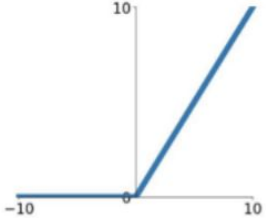
tanh

$\tanh(x)$



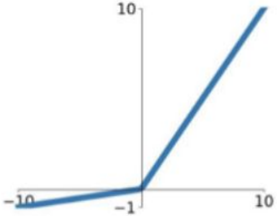
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

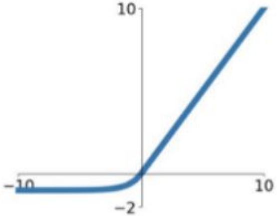


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

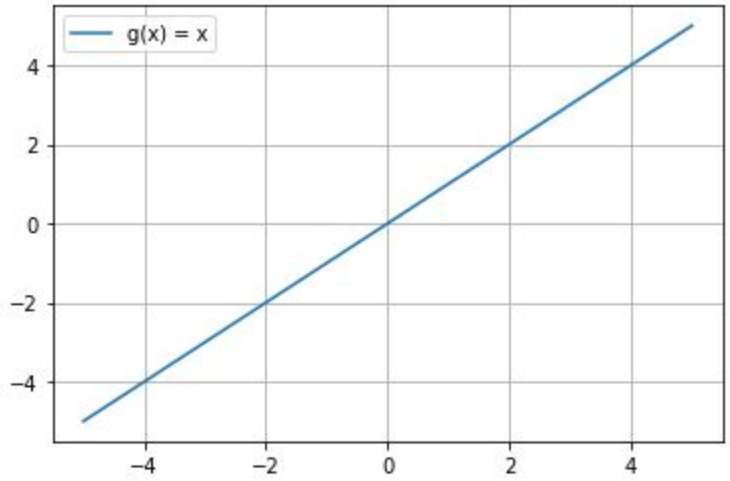
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



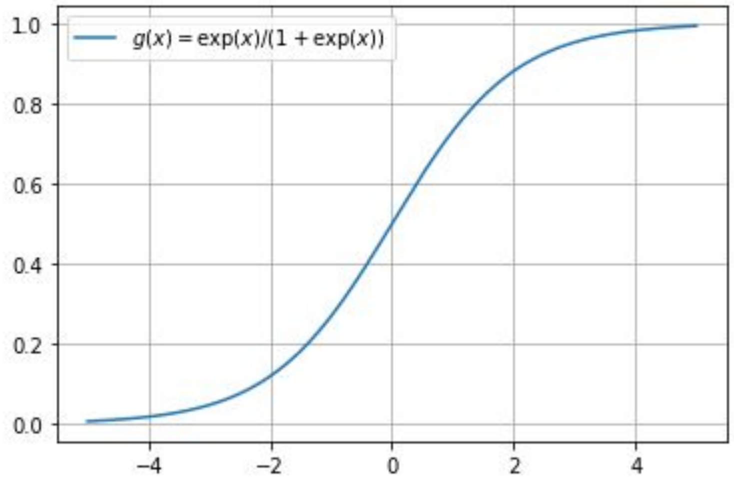
Activation function: linear

- No nonlinear transformation
- No input squashing



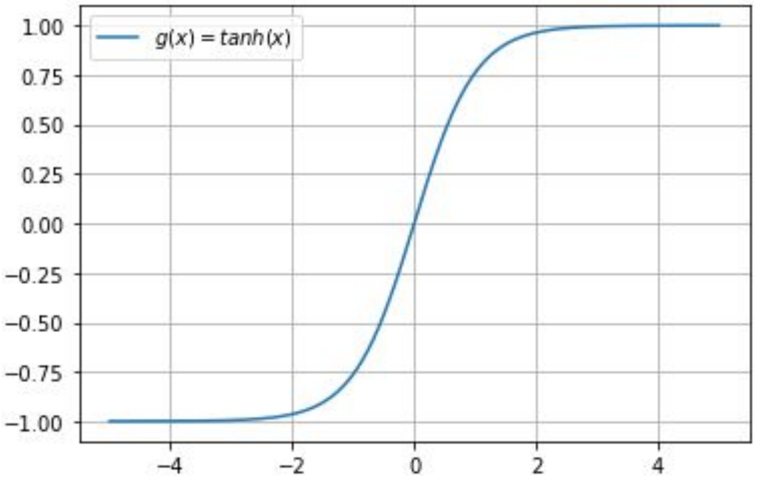
Activation function: sigmoid

- Always positive.
- Squashing the neuron's output between 0 and 1.
- Strictly increasing.



Activation function: tanh

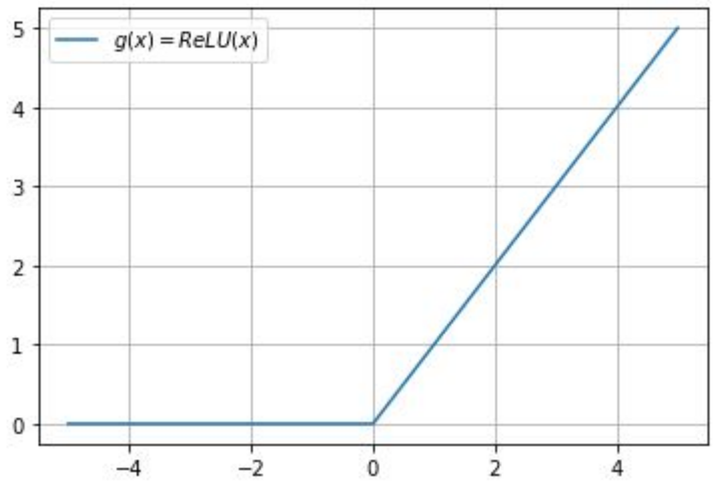
- Can be positive and negative.
- Squashing the neuron's output between -1 and 1.
- Strictly increasing.



COLUMBIA UNIVERSITY

Activation function: sigmoid

- Always positive.
- Pushing the neuron's output above 0.
- Strictly increasing.



Columbia University

Fitting neural networks: regression tasks

Recall our model is:

$$\begin{aligned} Z_m &= \sigma(\alpha_{0m} + \alpha_m^T X), \quad m = 1, \ldots, M. \\ T_k &= \beta_{0k} + \beta_k^T Z, \quad k = 1, \ldots, K. \\ f_k(X) &= g_k(T), \quad k = 1, \ldots, K. \end{aligned}$$

The unknow parameters of the model are often called *weights*. We denote the complete set of weights by θ, which consists of

$$\begin{aligned} \{\alpha_{0m}, \alpha_m;\ m = 1, 2, \ldots, M\} &\quad M(p+1) \text{ weights}, \\ \{\beta_{0k}, \beta_k;\ k = 1, 2, \ldots, K\} &\quad K(M+1) \text{ weights}. \end{aligned}$$

For regression, we use the squared error loss

$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{n} (y_{ik} - f_k(x_i))^2.$$

Fitting neural networks: classification tasks

Recall our model is:

$$\begin{aligned} Z_m &= \sigma(\alpha_{0m} + \alpha_m^T X), \quad m = 1, \dots, M. \\ T_k &= \beta_{0k} + \beta_k^T Z, \quad k = 1, \dots, K. \\ f_k(X) &= g_k(T), \quad k = 1, \dots, K. \end{aligned}$$

The unknow parameters of the model are often called *weights*. We denote the complete set of weights by θ, which consists of

$$\begin{aligned} &\{\alpha_{0m}, \alpha_m;\ m = 1, 2, \dots, M\} \qquad M(p+1) \text{ weights}, \\ &\{\beta_{0k}, \beta_k;\ k = 1, 2, \dots, K\} \qquad K(M+1) \text{ weights}. \end{aligned}$$

For classification we use either squared error or corss-entropy

$$R(\theta) = -\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik} \log f_k(x_i),$$

and the correponding classifier is $G(x) = \text{argmax}_k f_k(x)$.

COLUMBIA UNIVERSITY

Connection to gradient descent

Assume we use squared error loss. Let $z_{mi} = \sigma(\alpha_{0m} + \alpha_m^T x_i)$ and let $z_i = (z_{1i}, z_{2i}, \ldots, z_{Mi})$. Then we have

$$R(\theta) \equiv \sum_{i=1}^{n} R_i = \sum_{i=1}^{n} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2,$$

where

$$f_k(x_i) = g_k(\beta_{0k} + \beta_k^T z_i) = g_k\left(\beta_{0k} + \sum_{m=1}^{M} \beta_{km}\sigma(\alpha_{0m} + \alpha_m^T x_i)\right).$$

The derivatives are

$$\begin{aligned}
\frac{\partial R_i}{\partial \beta_{km}} &= -2(y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i) z_{mi}, \\
\frac{\partial R_i}{\partial \alpha_{ml}} &= -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i) \beta_{km} \sigma'(\alpha_m^T x_i) x_{il},
\end{aligned}$$

Columbia University

Connection to gradient descent

Assume we use squared error loss. Let $z_{mi} = \sigma(\alpha_{0m} + \alpha_m^T x_i)$ and let $z_i = (z_{1i}, z_{2i}, \ldots, z_{Mi})$. Then we have

$$R(\theta) \equiv \sum_{i=1}^{n} R_i = \sum_{i=1}^{n} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2,$$

where

$$f_k(x_i) = g_k(\beta_{0k} + \beta_k^T z_i) = g_k\left(\beta_{0k} + \sum_{m=1}^{M} \beta_{km} \sigma(\alpha_{0m} + \alpha_m^T x_i)\right).$$

The derivatives are

$$\begin{aligned} \frac{\partial R_i}{\partial \beta_{km}} &= -2(y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i) z_{mi}, \\ \frac{\partial R_i}{\partial \alpha_{ml}} &= -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i) \beta_{km} \sigma'(\alpha_m^T x_i) x_{il}, \end{aligned}$$

Updating rule

Assume we use squared error loss. Let $z_{mi} = \sigma(\alpha_{0m} + \alpha_m^T x_i)$ and let $z_i = (z_{1i}, z_{2i}, \ldots, z_{Mi})$. Then we have

$$R(\theta) \equiv \sum_{i=1}^{n} R_i = \sum_{i=1}^{n} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2,$$

where

$$f_k(x_i) = g_k(\beta_{0k} + \beta_k^T z_i) = g_k\left(\beta_{0k} + \sum_{m=1}^{M} \beta_{km}\sigma(\alpha_{0m} + \alpha_m^T x_i)\right).$$

A gradient update at the $(r+1)$st iteration has the form

$$\begin{aligned}
\beta_{km}^{(r+1)} &= \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}}, \\
\alpha_{ml}^{(r+1)} &= \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}^{(r)}}.
\end{aligned}$$

Updating rule

If we write the gradients as

$$\frac{\partial R_i}{\partial \beta_{km}} = -2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)z_{mi},$$

$$\frac{\partial R_i}{\partial \alpha_{ml}} = -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)\beta_{km}\sigma'(\alpha_m^T x_i)x_{il}.$$

COLUMBIA UNIVERSITY

Back propagation

If we write the gradients as

$$\frac{\partial R_i}{\partial \beta_{km}} = -2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)z_{mi},$$

$$\frac{\partial R_i}{\partial \alpha_{ml}} = -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i))g_k'(\beta_k^T z_i)\beta_{km}\sigma'(\alpha_m^T x_i)x_{il}.$$

$$\frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki} z_{mi},$$

$$\frac{\partial R_i}{\partial \alpha_{ml}} = s_{mi} x_{il}.$$

COLUMBIA UNIVERSITY

Back propagation

If we write the gradients as

$$\frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki} z_{mi},$$
$$\frac{\partial R_i}{\partial \alpha_{ml}} = s_{mi} x_{il}.$$

In some sense, δ_{ki} and s_{mi} are "errors" at the output and hidden layer units. The errors satisfy

$$s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki}.$$

They are called the *back-propagation equations*. The updates can be implemented with a two-pass algorithm:

- *forward pass*: fix weights, compute the predicted values $\hat{f}_k(x_i)$.
- *backward pass*: errors δ_{ki} are computed, and back-propagated to give the errors s_{mi}. Then use both sets of errors to compute the gradients.

COLUMBIA UNIVERSITY

Starting values

- If the weights are near zero, then the operative part of the sigmoid is roughly zero.
- Usually starting values for weights are chosen to be random values near zero.
- Hence the model starts out nearly linear, and becomes nonlinear as the weights increases.

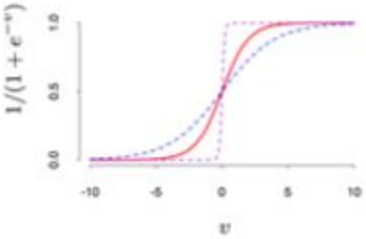


FIGURE 11.3. *Plot of the sigmoid function* $\sigma(v) = 1/(1+\exp(-v))$ *(red curve), commonly used in the hidden layer of a neural network. Included are* $\sigma(sv)$ *for* $s = \frac{1}{2}$ *(blue curve) and* $s = 10$ *(purple curve). The scale parameter* s *controls the activation rate, and we can see that large* s *amounts to a hard activation at* $v = 0$. *Note that* $\sigma(s(v - v_0))$ *shifts the activation threshold from 0 to* v_0.

Multiple minima

The error function $R(\theta)$ is nonconvex, possessing many local minima.

The solution we obtained from back-propagation is a local minimum.

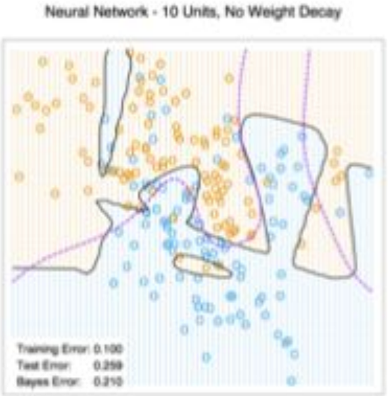
Usually, we try a number of random starting configuration, and choose the solution giving lowest error, or use the average predictions over the collection of networks as the final prediction.

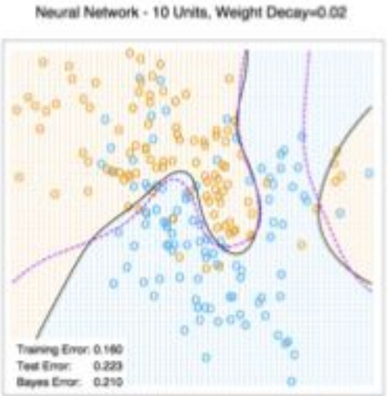
Multiple minima

- Often neural networks have too many weights and will overfit the data at the global minimum of R.
- A regularization method is *weight decay*. We add a penalty to the error function $R(\theta) + \lambda J(\theta)$, where

$$J(\theta) = \sum_{k,m} \beta_{km}^2 + \sum_{m,l} \alpha_{ml}^2.$$

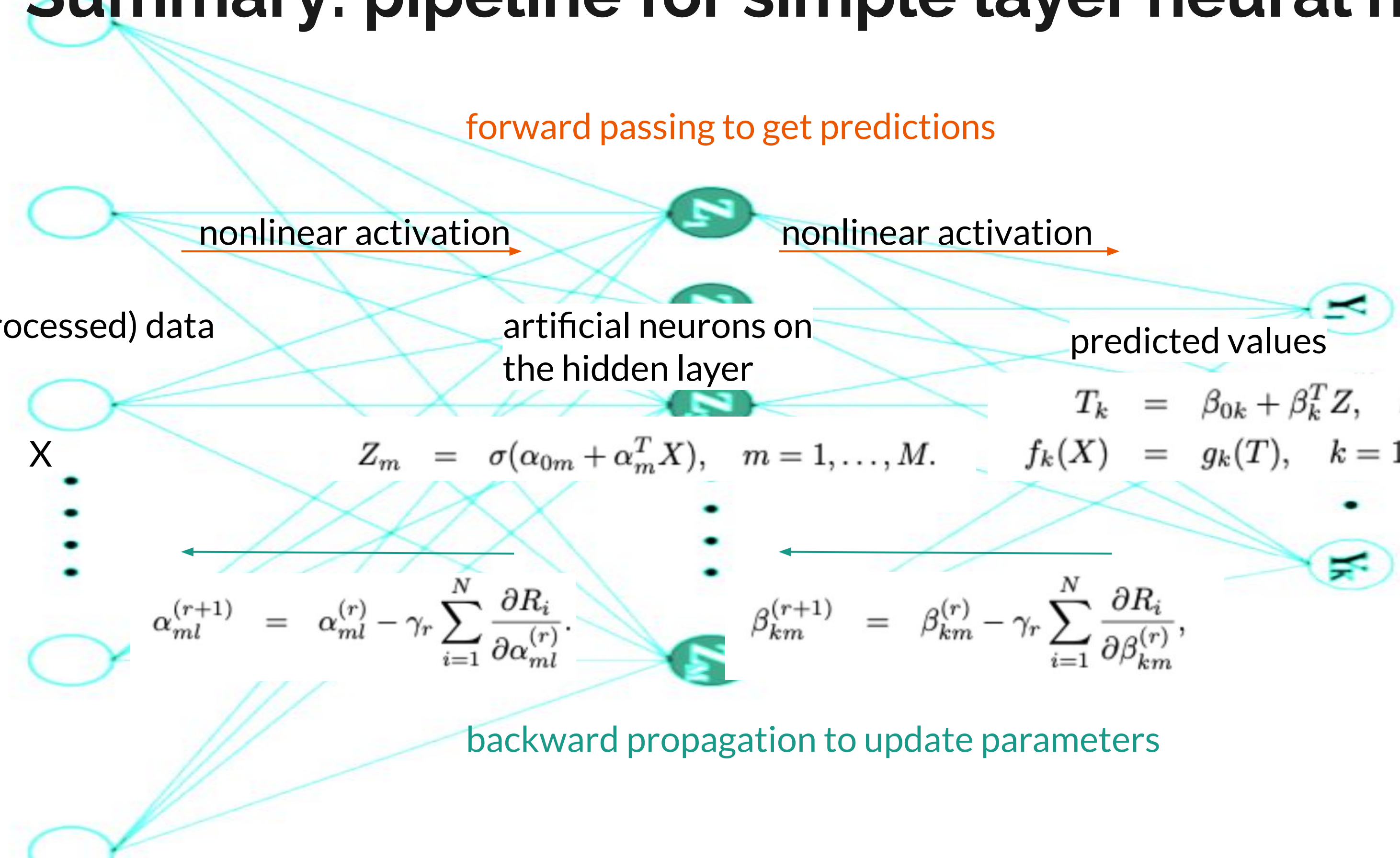
- $\lambda \geq 0$ is a tuning parameter, can be chosen by cross-validation.





Columbia University

Summary: pipeline for simple layer neural net



forward passing to get predictions

nonlinear activation

nonlinear activation

(preprocessed) data

X

artificial neurons on the hidden layer

predicted values

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X), \quad m = 1, \ldots, M.$$

$$T_k = \beta_{0k} + \beta_k^T Z, \quad k = 1, \ldots, K.$$

$$f_k(X) = g_k(T), \quad k = 1, \ldots, K.$$

$$\alpha_{ml}^{(r+1)} = \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}^{(r)}}.$$

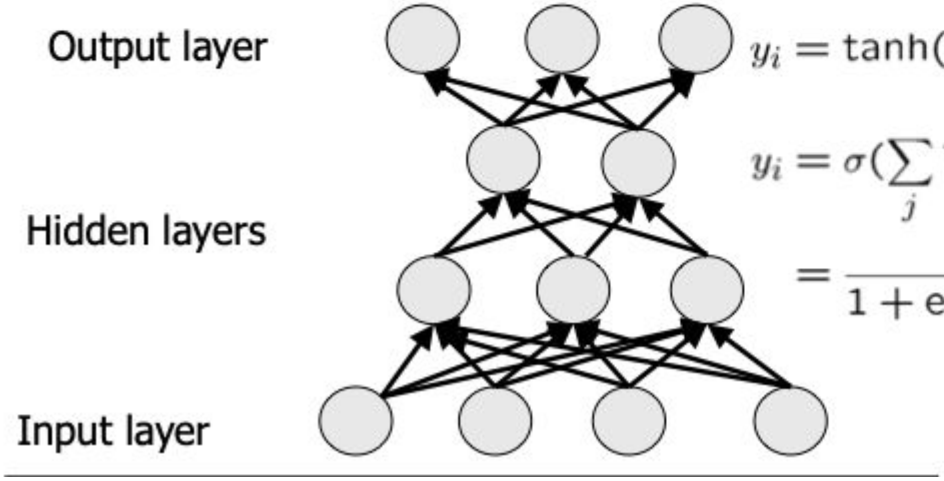
$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}},$$

backward propagation to update parameters

Columbia University

Deep neural network

Definition: Deep architectures are composed of *multiple levels* of non-linear operations, such as neural nets with many hidden layers.



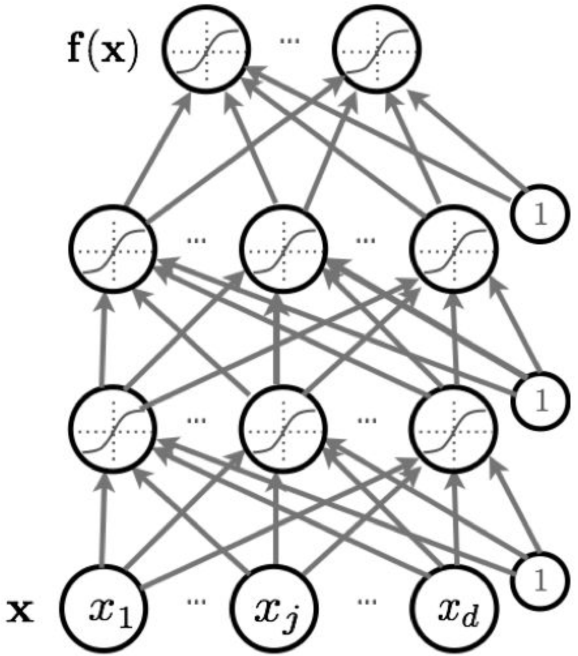
$$y_i = \tanh\left(\sum_j W_{ij} x_j + W_{i0}\right)$$

$$y_i = \sigma\left(\sum_j W_{ij} x_j + W_{i0}\right)$$

$$= \frac{1}{1 + \exp[-(\sum_j W_{ij} x_j + W_{i0})]}$$

Deep neural network: architecture

- How neural networks predict f(x) given an input x:
  - Feed forward
  - Types of activations
  - Capacity of neural networks
- How to train neural networks:
  - Loss function
  - Backward propagation with gradient descent
- More recent techniques:
  - Architecture
  - Dropout
  - SGD
  - Batch normalization



Deep neural network: architecture

- Consider a network with L hidden layers.
  - layer pre-activation for k>0

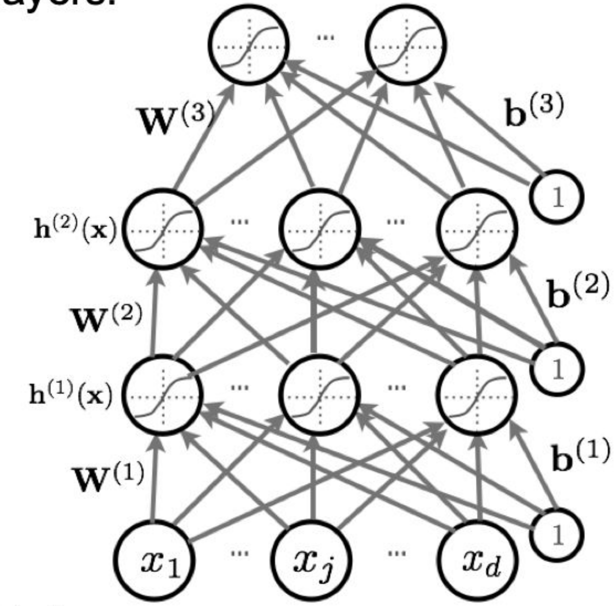
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

  - hidden layer activation from 1 to L:

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

  - output layer activation (k=L+1):

$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$

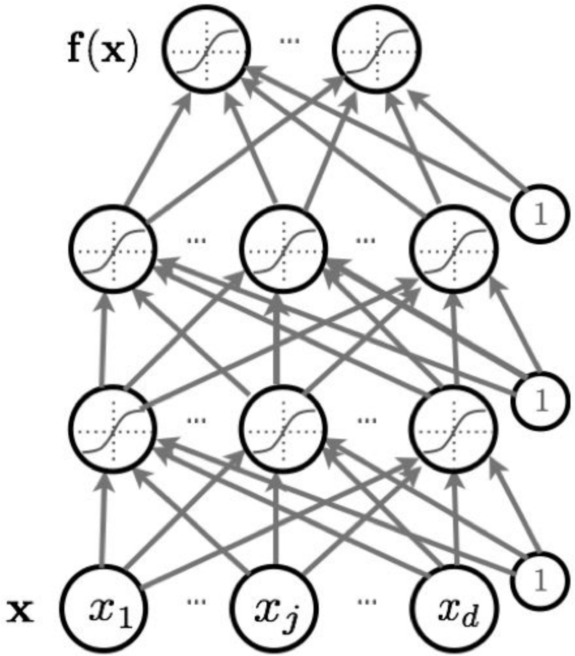


$$(\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x})$$

COLUMBIA UNIVERSITY

Neural network structure

- Fully connected layer
- Deep architecture
  - LeNet5
  - AlexNet
  - ResNet
  - ……
- Semi-supervised learning

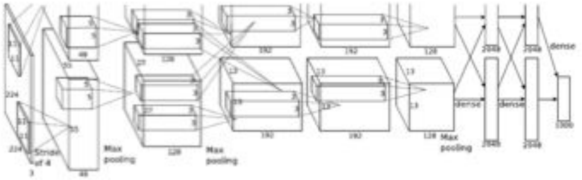


COLUMBIA UNIVERSITY

Deep neural network: important breakthrough

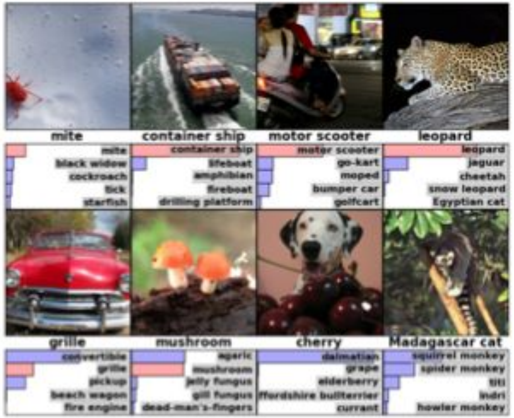
- Deep Convolutional Nets for Vision (Supervised)

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



IMAGENET

1.2 million training images
1000 classes

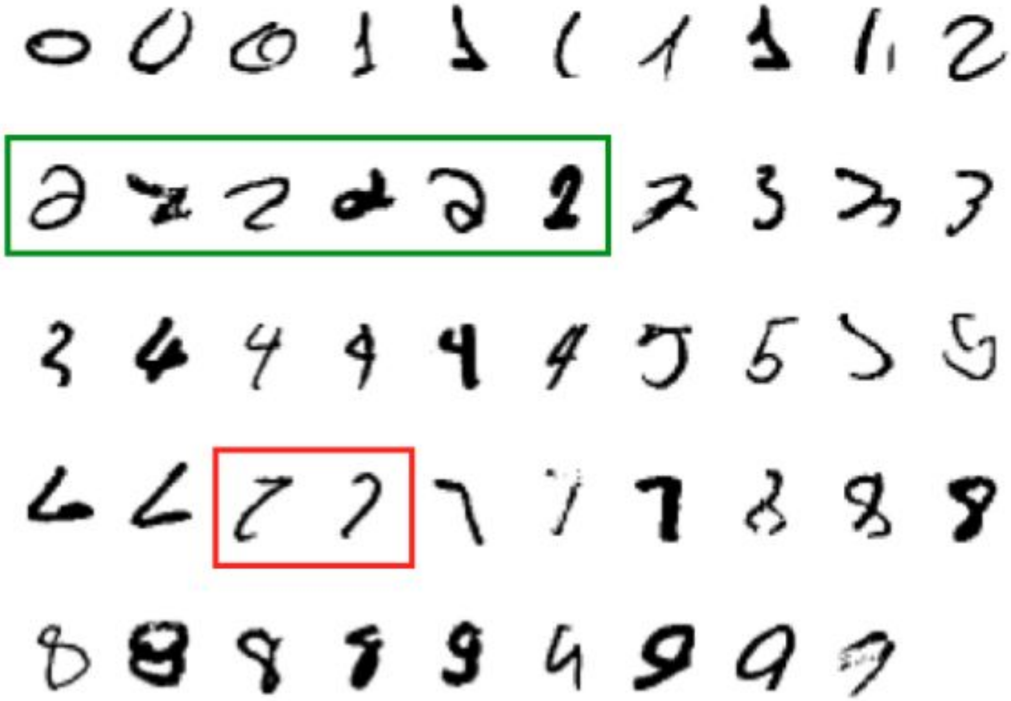


- Deep Nets for Speech (Supervised)

  Hinton et. al. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups, IEEE Signal Processing Magazine. 2012.

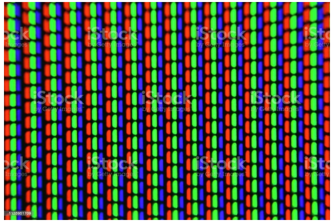
Columbia University

MNIST

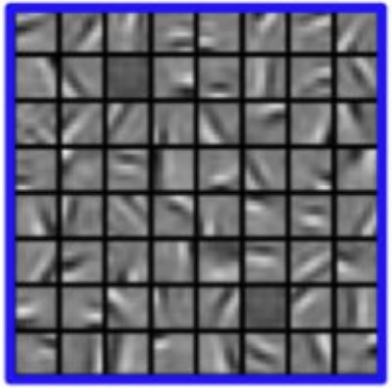


Convolutional neural network

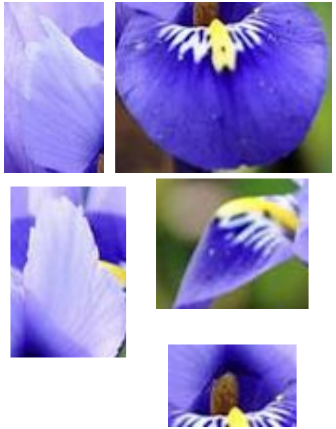
pixels



features



object parts/
combination of features

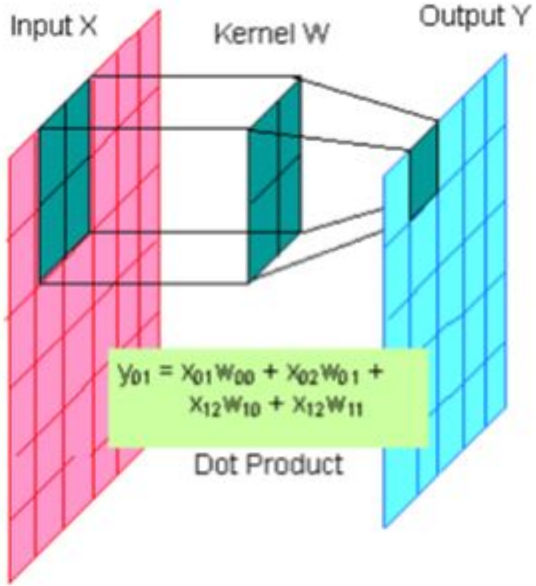


objects

Columbia University

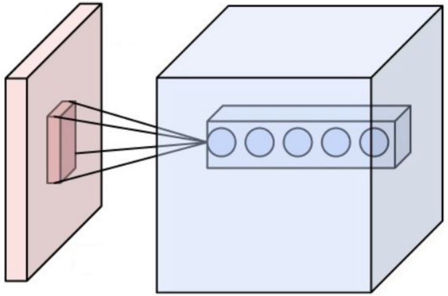
Convolutional neural network

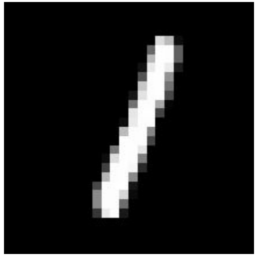
- Instead of focusing on individual, CNN provides a automatic algorithm to study groups of nearby pixels.
- Very successful in
  - computer vision (CV)
  - natural language processing (NLP)



COLUMBIA UNIVERSITY

Convolutional neural network





- Hyperparameters in convolutional layer:

  (pytorch, MNIST data)
- The input sample size is (1,28,28)

  ```
  torch.nn.Conv2d(
  in_channels    = 1,  # we only have grayscale figure
  out_channels = 16, # number of filters
  kernel_size    = 5,   # filter size
  stride             = 1,   # filter movement
  padding          = 2   # when stride = 1,
                            padding = (kernel_size -1)/2
  )
  ```
- Output size is (16,28,28)

COLUMBIA UNIVERSITY

Convolutional neural network

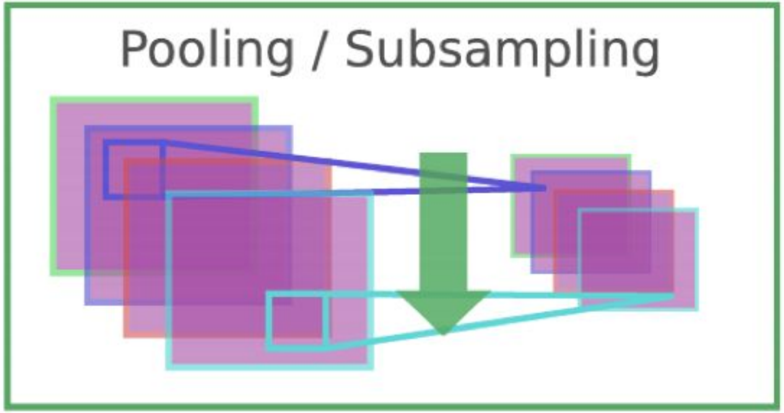
- Pooling/subsampling hidden units in same neighborhood

  - Introduces invariance to local translations

  - Reduces the number of hidden units in hidden layer

- Hyperparameters in pooling layers

  torch.nn.MaxPool2d(2)



Jarret et al. 2009

COLUMBIA UNIVERSITY

Convolutional neural network

(1,28,28)

convolutional layer

(16, 28, 28)

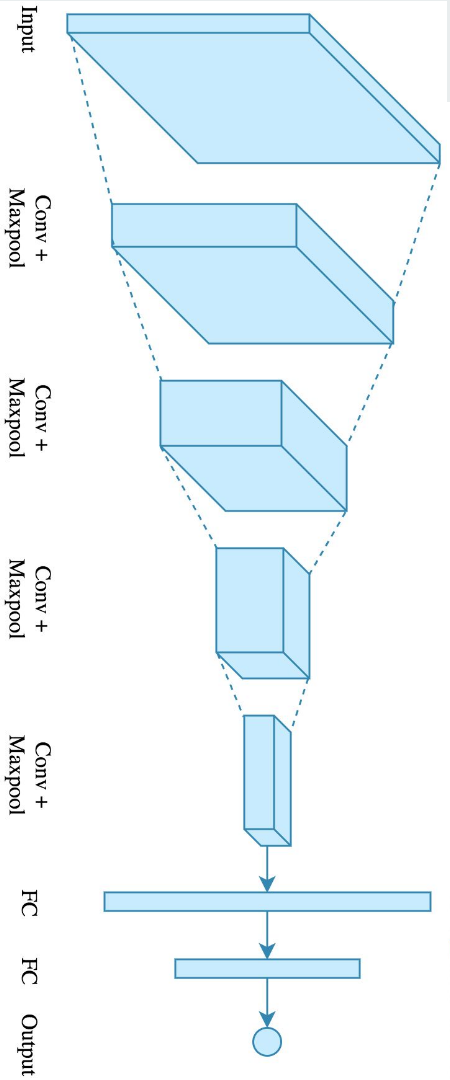
nonlinear activation

layer (usually ReLU)

(16, 28, 28)

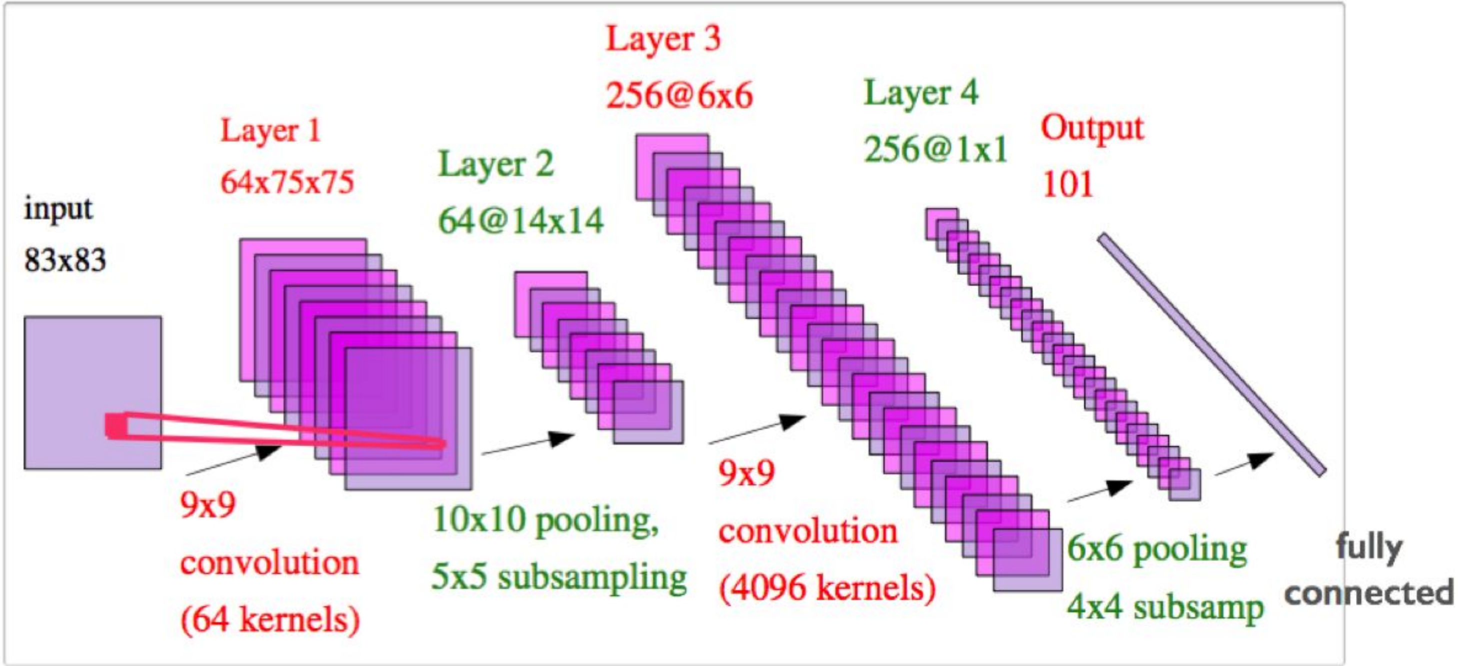
max pooling

(16, 14, 14)



COLUMBIA UNIVERSITY

Convolutional neural network

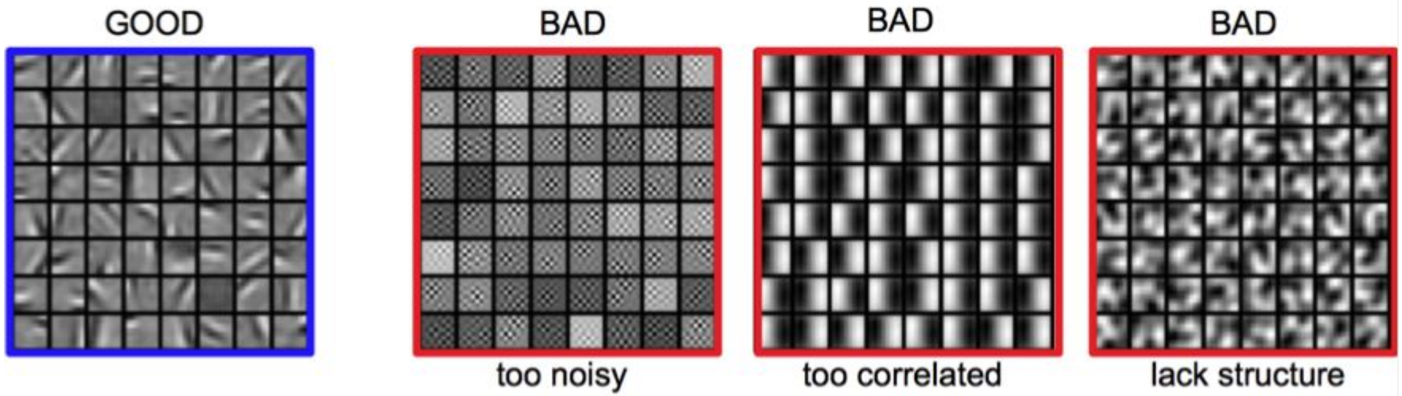


From Yann LeCun's slides

Columbia University

Convolutional neural network

- Visualize parameters:
  learned features should exhibit structure and should be uncorrelated and are uncorrelated

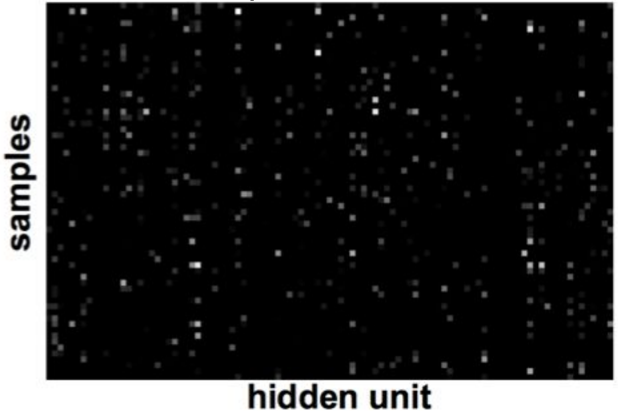


[From Marc'Aurelio Ranzato, CVPR 2014 tutorial]

Convolutional neural network

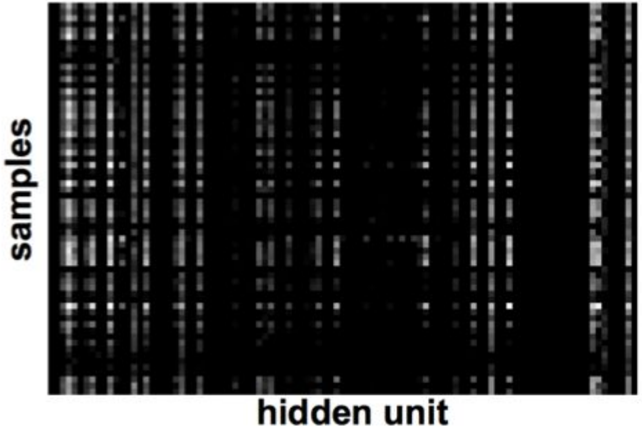
- Visualize features (feature maps need to be uncorrelated)

Good training:
hidden units are sparse
across samples.



[From Marc'Aurelio Ranzato, CVPR 2014 tutorial]

Bad training:
hidden units are highly
correlated.



[From Marc'Aurelio Ranzato, CVPR 2014 tutorial]

Columbia University

Why training is hard

- Underfitting: use better optimization:

  - use better optimization tools (e.g. batch-normalization, 2nd-order methods).

  - use GPUs, distributed computing.
- Overfitting: use better regularization:

  - unsupervised pre-training

  - stochastic drop-out training
- For many large-scale practical problems, have to scale up:

  - ReLu nonlinearity

  - initialization (e.g. Kaiming He's initialization)

  - stochastic gradient descent

  - momentum, batch-normalization, and drop-out

Preprocessing

- One-hot representation: class 0 or class 1 → (1,0) or (0,1)
- Normalizing the inputs will speed up training (Lecun et al. 1998)

  - could normalization be useful at the level of the hidden layers?
- Batch normalization is an attempt to do that

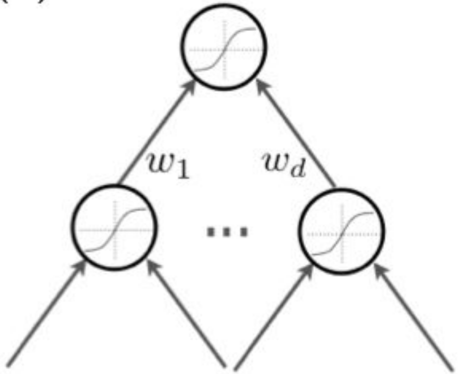
  - each unit's pre-activation is normalized (mean subtraction, stddev division)

  - during training, mean and stddev is computed for each minibatch

  - backpropagation takes into account the normalization

  - at test time, the global mean / stddev is used

$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$



Initialization of parameters

- Initialize biases to 0
- For weights
  - Can not initialize weights to 0 with tanh activation
    - ➢ All gradients would be zero (saddle point)
  - Can not initialize all weights to the same value
    - ➢ All hidden units in a layer will always behave the same
    - ➢ Need to break symmetry
  - Sample $\mathbf{W}_{i,j}^{(k)}$ from $U\left[-b, b\right]$, where

$$b = \frac{\sqrt{6}}{\sqrt{H_k + H_{k-1}}}$$

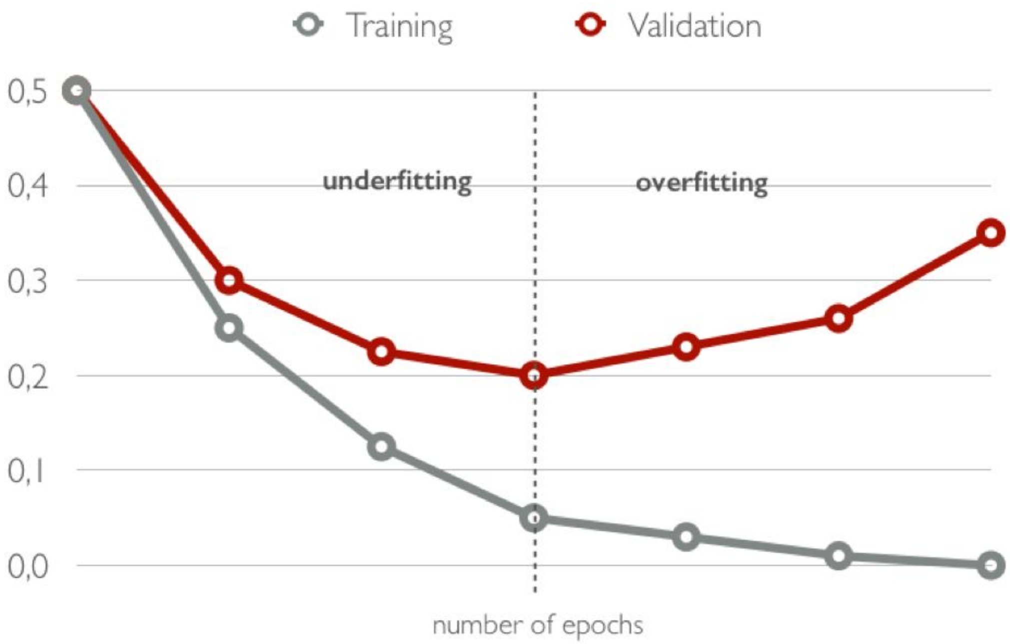
Sample around 0 and break symmetry

Size of $\mathbf{h}^{(k)}(\mathbf{x})$

COLUMBIA UNIVERSITY

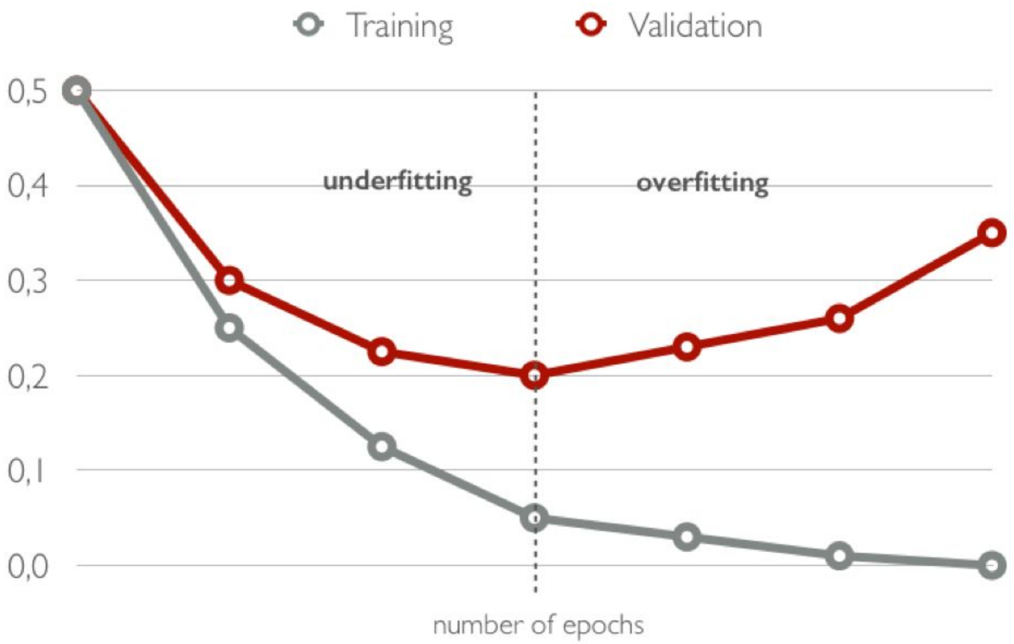
Deep neural network: overfitting

- Overfitting often occurs in applications of neural networks.
- Ways to overcome:
  - Early stopping:
    Stop training process early.
  - Dropout:
    Use random binary masks.



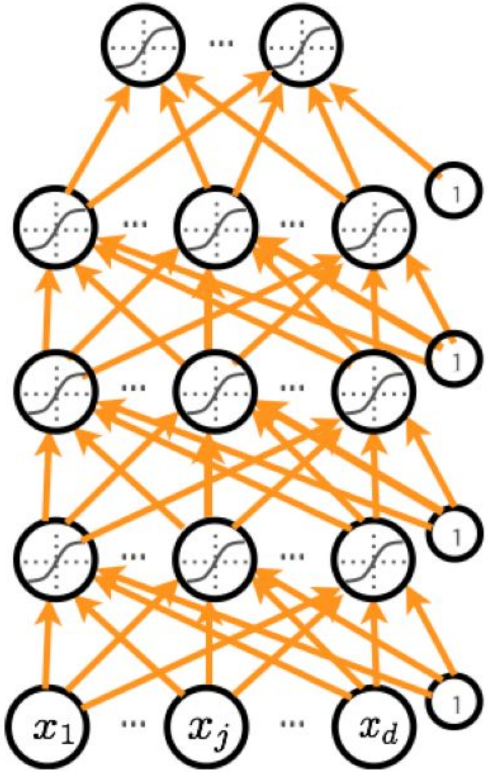
COLUMBIA UNIVERSITY

Early stopping



Dropouts

- Cripple neural network by removing hidden units stochastically
  - each hidden unit is set to 0 with probability 0.5
  - hidden units cannot co-adapt to other units
  - hidden units must be more generally useful
- Could use a different dropout probability, but 0.5 usually works well



COLUMBIA UNIVERSITY

Model selection

- Training Protocol:
  - Train your model on the Training Set $\mathcal{D}^{\text{train}}$
  - For model selection, use Validation Set $\mathcal{D}^{\text{valid}}$
    - Hyper-parameter search: hidden layer size, learning rate, number of iterations/epochs, etc.
  - Estimate generalization performance using the Test Set $\mathcal{D}^{\text{test}}$
- Remember: Generalization is the behavior of the model on **unseen examples**.

COLUMBIA UNIVERSITY

Optimization

- SGD with momentum, batch-normalization, and dropout usually works very well
- Pick learning rate by running on a subset of the data

  - Start with large learning rate & divide by 2 until loss does not diverge

  - Decay learning rate by a factor of ~100 or more by the end of training

  - Use ReLU nonlinearity

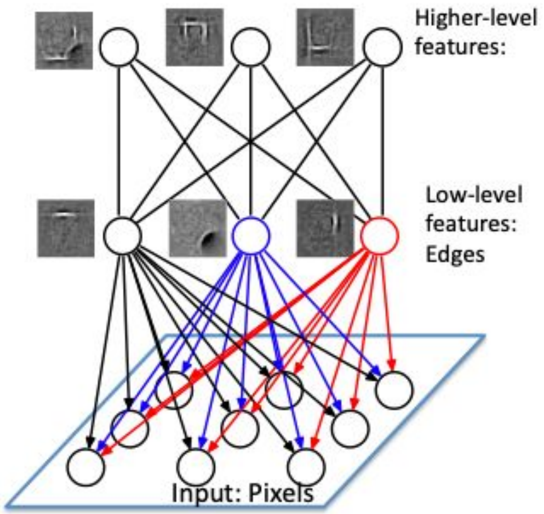
  - Initialize parameters so that each feature across layers has similar variance. Avoid units in saturation.
- Use adapted learning rate

Columbia University

Deep neural network: important breakthrough

- Deep Belief Networks, 2006 (Unsupervised)

  Hinton, G. E., Osindero, S. and Teh, Y., A fast learning algorithm for deep belief nets, Neural Computation, 2006.



Theoretical Breakthrough:

- Adding additional layers improves variational lower-bound.

Efficient Learning and Inference with multiple layers:

- Efficient greedy layer-by-layer learning learning algorithm.

- Inferring the states of the hidden variables in the top most layer is easy.

COLUMBIA UNIVERSITY

Deep neural network: important breakthrough

- Conditional generative model P(zebra images| horse images)

- Style Transfer

Input Image

Monet

Van Gogh

Zhou el al., Cycle GAN 2017

Columbia University

Summary: pipeline for deep neural net

forward passing to get predictions

nonlinear activation | nonlinear activation | nonlinear activation | nonlinear activation | nonlinear activation

(preprocessed) data | hidden layers | hidden layers | hidden layers | …… | hidden layers | predicted values

backward propagation to update parameters

Summary:

- Usually the loss function is non-convex
- To have better performance, we can

  - Try different loss function

  - Try different penalty on learnt parameters

  - Try different initializations

  - Try different learning rate (1e-2, 1e-3)/ adapted learning rate

  - Try dropout

  - Try early stopping

  - Try different optimizer (SGD+momentum, Adam, RMSprop)

  - Try different activation function + batch normalization

  - Try different/deeper architecture

COLUMBIA UNIVERSITY

References

- Christopher Bishop: Pattern Recognition and Machine Learning, Chapter 5
- Ziv Bar-Joseph, Tom Mitchell, Pradeep Ravikumar and Aarti Singh: CMU 10-701
- Ryan Tibshirani: CMU 10-725
- Ruslan Salakhutdinov: CMU 10-703
- https://towardsdatascience.com/neural-network-architectures-156e5bad51ba
- https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2