CS395T: Foundations of Machine Learning for Systems Researchers

Fall 2025

Lecture 3: Neural Networks

Organization

- Nonlinear functions used in Neural Networks (NNs)
  - Sigmoidal: $\Re \rightarrow \Re$
  - Softmax: $\Re^n \rightarrow \Re^n$
- Examples of Multilayer Perceptrons (MLP)
  - Implementing Boolean functions
- Loss functions for probability distributions
  - KL-divergence, cross-entropy
- Convolutional neural networks (CNN)
  - Array/tensor $\rightarrow$ class label
  - Image classification

```
Function R(P,Q) {
    A = W0*P
    B = f0(A,Q)
    C = W1*B
    D = W2*B
    E = f1(C)
    F = f2(D)
    R = f3(E,F)
    return R}
```

Training data: {(pi,qi,ti)}

- Loss = mean square error
- Find gradient of Loss
- Perform gradient-descent to find optimal W0,W1,W2

Neural networks get their power from nonlinearity

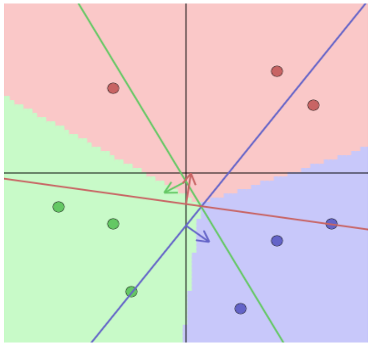
- Nonlinear base functions are critical to the power of neural networks
  - Composition of linear functions is linear!

- Example: classification*

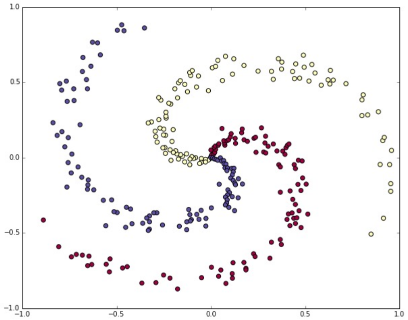
- Linear classifiers work for simple data sets such as top right one
  - Data: points in plane with one of three labels (red, green, blue)
  - Given a new point, predict its label
  - Linear classifier divides plane into three regions and uses that to predict label

- Linear classifiers not adequate for more complex data sets such as bottom right one
  - Nonlinearity is essential
  - NNs can handle this problem

* From Fei Fei Li's CS231n course notes



Linear classifiers (SVM) work



Linear classifiers fail; NNs work

Nonlinear functions used in neural networks

Sigmoidal functions, ReLU,…

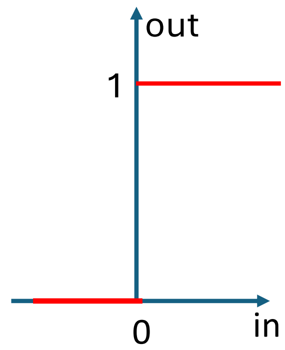
- Intuition: binary classification
  - Smooth versions of step function

Softmax

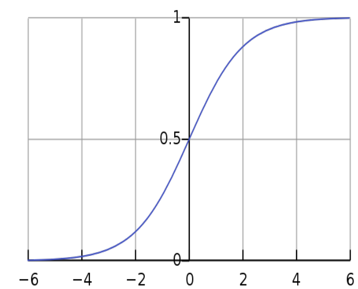
- Intuition: multiway classification
  - Smooth version of argmax

Sigmoidal functions: $\Re \rightarrow \Re$

- Step function
  - If input is +ve, output is 1 else output is 0
  - Does not have derivative at 0
- Sigmoidal functions (S-shaped):
  - Smooth approximations to step functions
  - Examples: logistic function, tanh



Step function



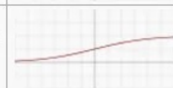
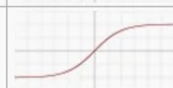
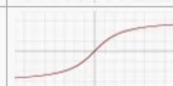
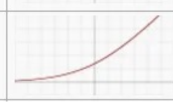
Logistic function

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

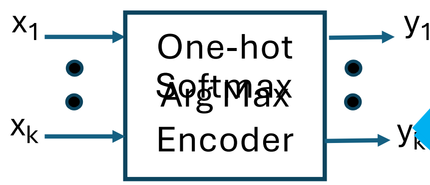
$$\frac{df}{dx} = f(x)(1-f(x))$$

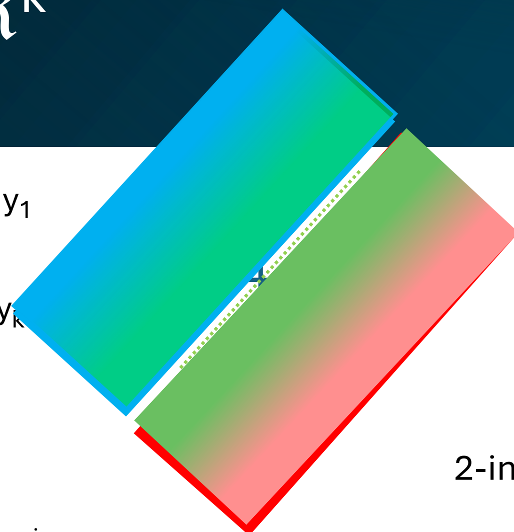
- Disadvantages of logistic function:
  - Vanishing gradients for large magnitude inputs, so weights upstream from activation function are updated slowly
  - Computationally more expensive than alternatives like ReLU

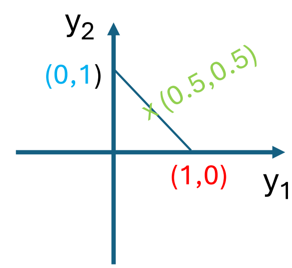
Other activation functions

Name	Plot	Equation	Derivative
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Softmax: $\Re^k \rightarrow \Re^k$







2-input one-hot arg max encoder

- One-hot arg max encoder
  - y_i is 1 if x_i is largest input, and zero otherwise
  - If m inputs are maximal, corresponding outputs are 1/m
  - Note: output values are non-negative and sum to 1
  - Outputs can be interpreted as probability distribution
- Smooth approximation to one-hot arg max encoder: softmax
  - Intuition: points on line x2 = x1+k are mapped to $(\frac{1}{e^k+1}, \frac{e^k}{e^k+1})$
  - Outputs can be interpreted as probability distribution
- 2-input encoder with one input set to 0 is logistic function

$$y_i = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}$$

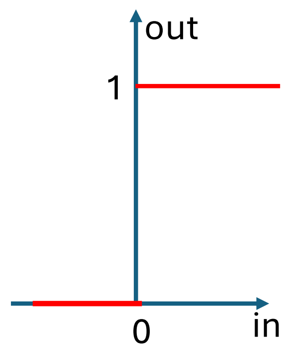
$$\frac{\partial y_i}{\partial x_i} = y_i(1-y_i)$$

$$\frac{\partial y_i}{\partial x_m} = -y_i * y_m$$

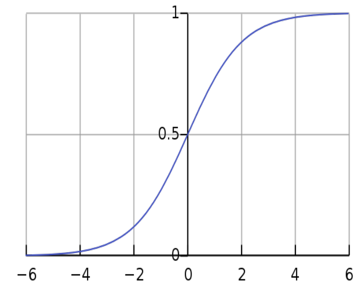
Softmax function:k inputs

MLP Examples

Example: encoding Boolean functions (I)



Step function Θ

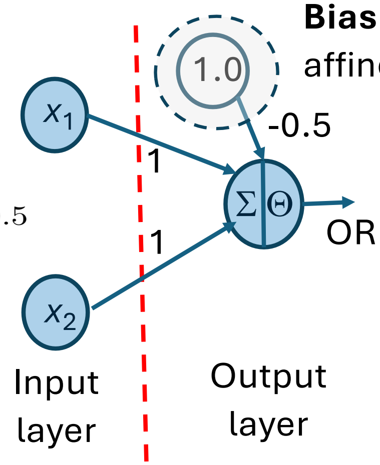


Logistic function

- Implementation: $\hat{y} = \Theta(W\underline{x} + \underline{b})$
  - Inputs and outputs: 0 for false, 1 for true
  - Network
  - Constants $\underline{b}$ referred to as bias
- NOT: $\hat{y} = \Theta(-1 * x + 0.5)$
- AND: $\hat{y} = \Theta(x_1 + x_2 - 1.5)$
- OR: $\hat{y} = \Theta(x_1 + x_2 - 0.5)$

$$\begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - 0.5$$

Bias: permits affine functions



Single-layer perceptron

Example: encoding Boolean functions (II)

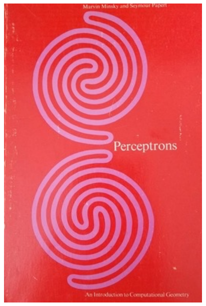
- XOR:

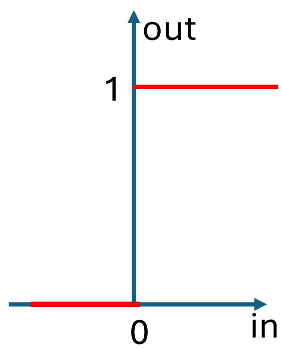
$$\hat{y} = \begin{pmatrix}1 & 1\end{pmatrix} \Theta \begin{pmatrix} x_1 - x_2 - 0.5 \\ -x_1 + x_2 - 0.5 \end{pmatrix}$$

- Implements

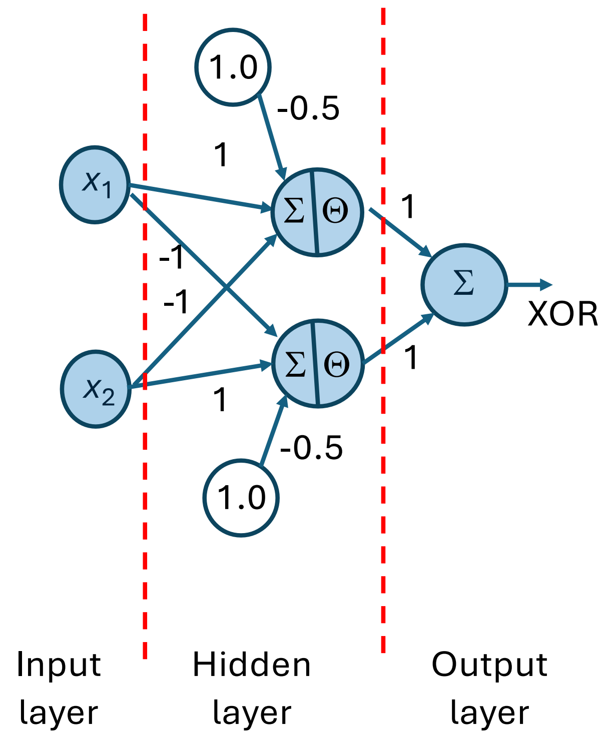
$$x_1 \oplus x_2 = (x_1 \wedge \bar{x_2}) \vee (\bar{x_1} \wedge x_2)$$

- XOR requires a hidden layer
  - Minsky & Papert





Step function Θ



Power of multi-layer neural networks

Universal approximation theorem:

- network with a linear output layer, and
- at least one hidden layer with any "squashing" activation function (such as the logistic function)

can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units.

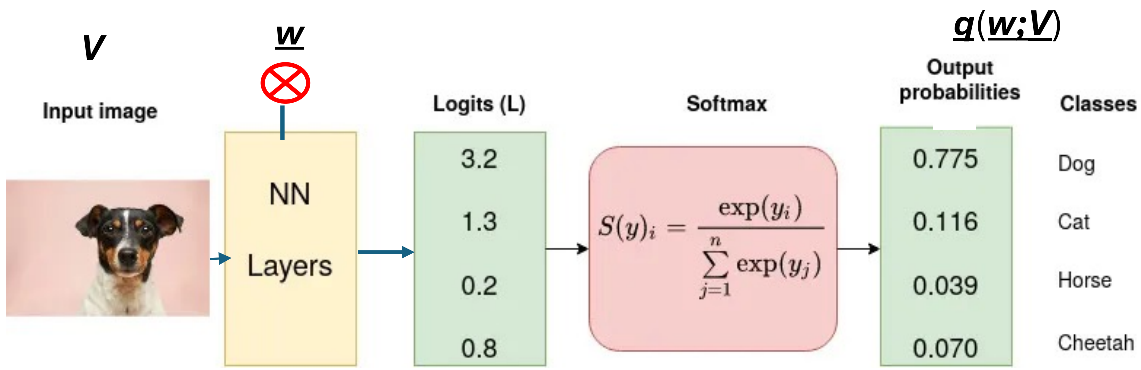
Many similar results with different assumptions

Variations of this theorem go back to McCulloch & Pitts (1943)

Wikipedia has a good overview:
https://en.wikipedia.org/wiki/Universal_approximation_theorem

Loss Functions for Probability Distributions

Example: image classification using NNs



4 image classes: dog, cat, horse, cheetah

Training data:

- Images labeled with **one-hot encoding** of class: for dog, $\begin{bmatrix} 1.0 \\ 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}$
- Interpret as **probability distribution** ***p(V)*** for classes

NN:

- Parameters $\underline{w}$
- Output ***$\underline{q}(\underline{w};V)$*** is probability distribution for classes

Logits: raw scores that are usually converted to probabilities by softmax

Loss function: measure of how "far away" ***$\underline{p}$***(***V***) and ***$\underline{q}(\underline{w};V)$*** are from each other

- Loss across entire training set = mean of loss per image

Some possibilities

Consider label $\underline{p}(v)$ and model output $\underline{q}(\underline{w};v)$ to be points in 4-D space and compute some measure of "distance" between them

$$\|\underline{p}(v)-\underline{q}(\underline{w};v)\|_1 \qquad \|\underline{p}(v)-\underline{q}(\underline{w};v)\|_2$$

Drawback of these loss functions: slow convergence

- For classification, we do not care about values in prediction vector ($\underline{q}$), only the predicted label (bin)
- L_2 norm was used for classification problems till the late 90's

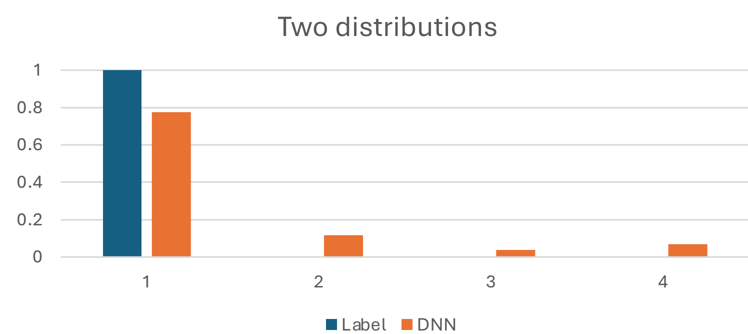
Modern approach: **divergences**

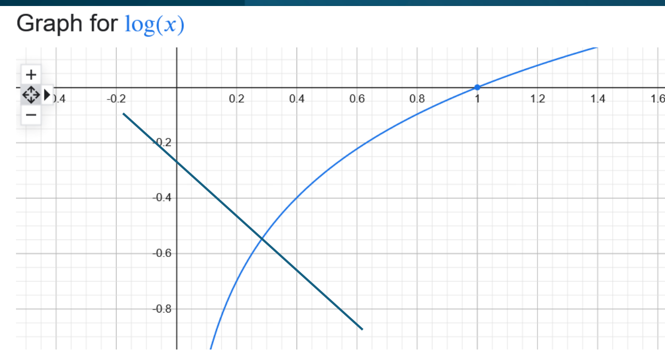
- *D(p,q):* distribution x distribution → real
- Class of functions that satisfy these properties

$$D(p,q) \geq 0$$
$$D(p,q) = 0 \leftrightarrow p = q$$

- Need not be a metric since it may not even be symmetric
- Many divergence functions in literature: L_1 and L_2 norms, Bregman divergence, f-divergence,…

Kullback-Leibler (KL) divergence: $D_{KL}(P||Q)$





- Start with $\sum_{i=1}^{c} p(i) - q(i)$

- Two key intuitions
  - If *p(i)* is small for some i, we do not care that much about *(p(i)-q(i))* $\sum_{i=1}^{c} p(i)(p(i) - q(i))$
  - Any monotonically increasing function f can be used to massage values of *p* and *q* without changing the sign of the difference
    - KL-divergence uses log (usually $\log_2$)
    - Intuition: penalize mistakes: large *p(i)*, small *q(i)* $\sum_{i=1}^{c} p(i)(\log(p(i)) - \log(q(i)))$

Cross-entropy (I)

$$D_{KL}(P||Q) = \sum_{i=1}^{c} p(i)(\log(p(i)) - \log(q(i)))$$

$$= -\sum_{i=1}^{c} p(i)\log\left(\frac{1}{p(i)}\right) + \sum_{i=1}^{c} p(i)\log\left(\frac{1}{q(i)}\right)$$

$$= \quad -H(P) \quad + \quad H(P,Q)$$

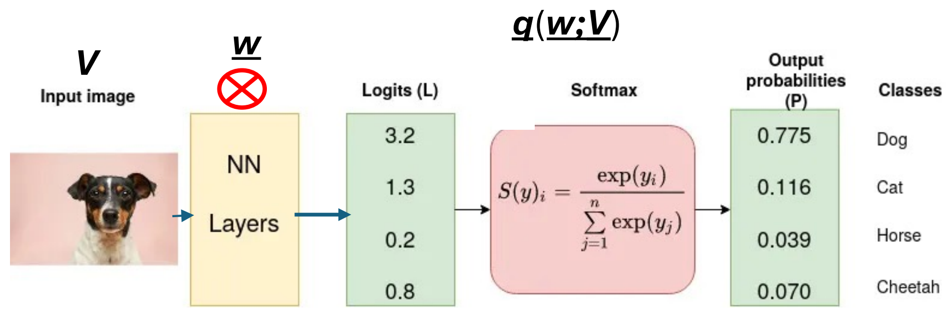
H(P,Q) is called **cross-entropy** between P and Q (non-negative value)

H(P) is constant for given training set so we can use cross-entropy for loss

In practice, try to reduce cross-entropy to less than 0.25

Cross-entropy (II)

$$H(P, Q) = \sum_{i=1}^{c} p(i) \log\left(\frac{1}{q(i)}\right)$$



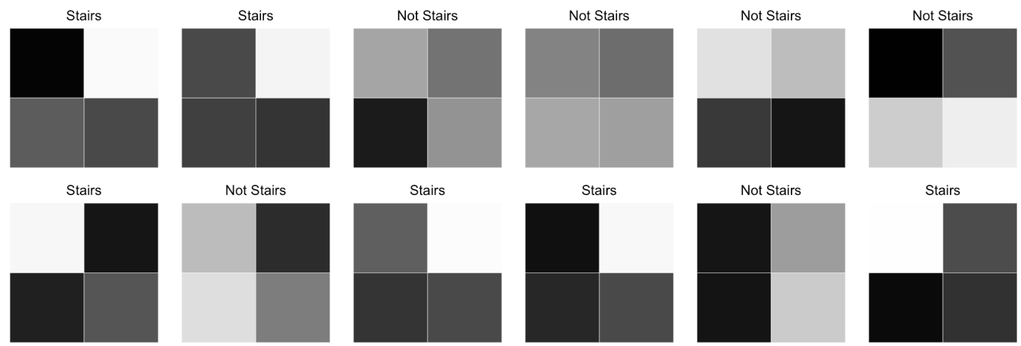
Gradient of cross-entropy loss assuming ground truth class is m
(p(m) = 1, p(i) = 0 for all other classes)

$$H(P,Q) = -\log(q(m)) \qquad \frac{\partial H(P,Q)}{\partial Wi} = -\frac{1}{q(m)} * \frac{\partial q(m)}{\partial Wi}$$

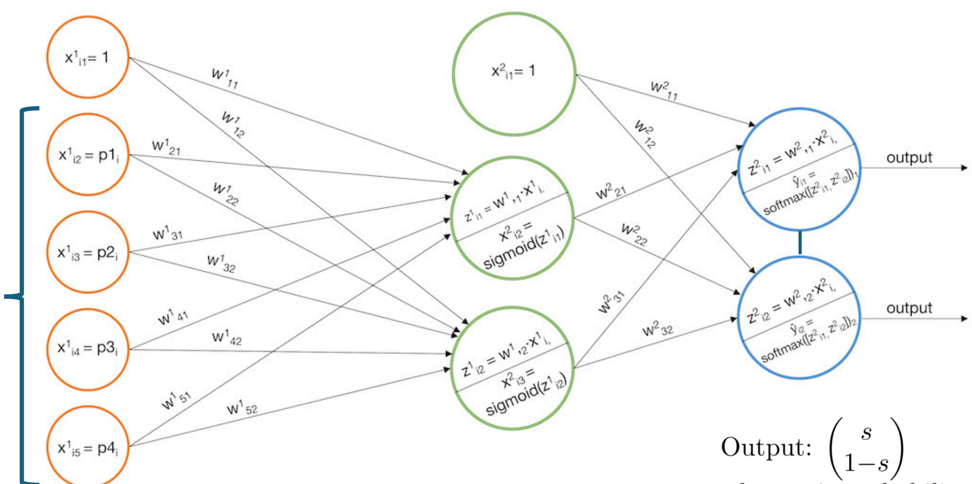
Jensen-Shannon divergence: square root is metric

$D_{JS}(P||Q) = ½ D_{KL}(P||M) + ½ D_{KL}(Q||M)$ where $M = ½ (P+Q)$

Image classification problem



p1,p2,p3,p4:
gray scale values for
image

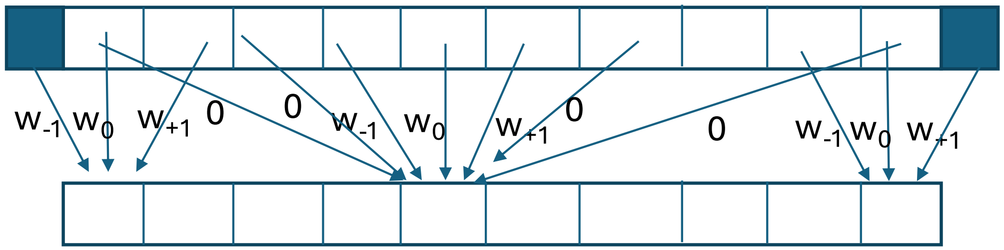


Output: $\begin{pmatrix} s \\ 1-s \end{pmatrix}$
where s is probability of "stairs"

https://www.gormanalysis.com/blog/neural-networks-a-worked-example

Convolutional Neural Networks (CNNs)

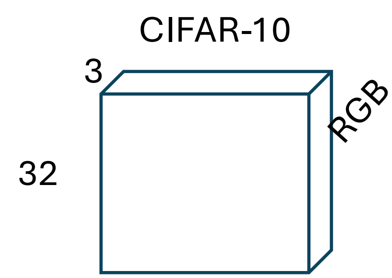
Convolution (1D example)

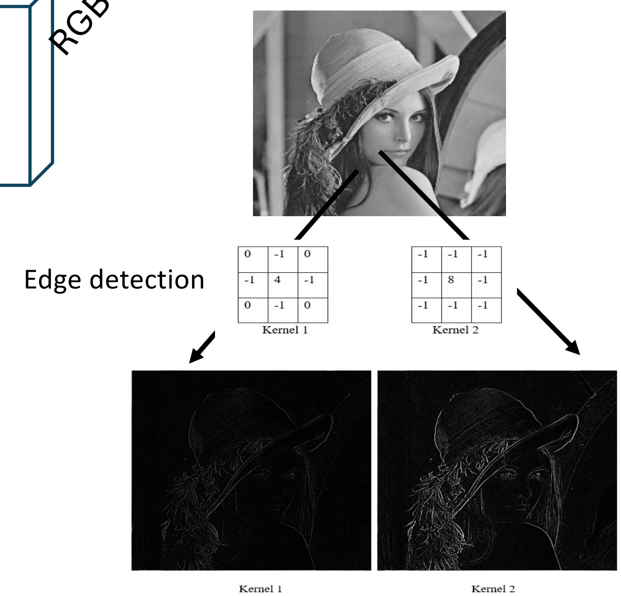


- Padding: input extended with zeros at boundaries, so output is same size
- Stride: gap between successive stencil applications
  - Output smaller than input (down-sampling)
- Weights are learned during training rather than being set heuristically
- Relation to fully-connected layers:
  - Sparse linear layer
    - Strong prior: weights of distant points is set to zero before training
  - Weight-sharing
    - Same weights used to compute all output elements

Convolutions on images

- Input data types are array/matrix/tensor
    - Image is an NxN array of pixels
    - Each pixel has grayscale value (0-255) or RGB values (channels)
    - Example: CIFAR-10 dataset has 60K, 32x32 RGB images, each labeled with one of 10 classes
- Convolution operation (aka filter)
    - Stencil operation to extract features
- Multiple filters: multiple output arrays

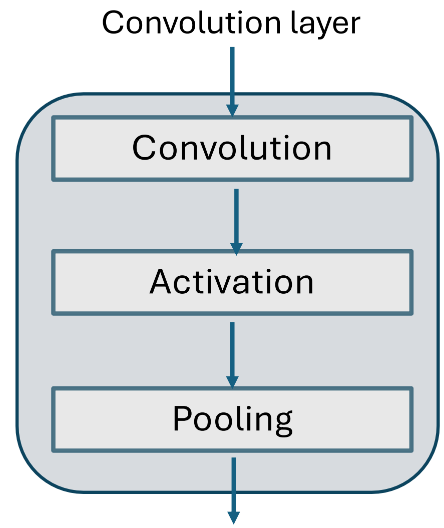


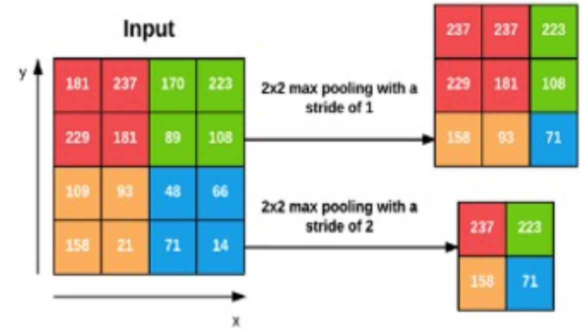


https://medium.com/@shashikadilhani97/digital-image-processing-filters-832ec6d18a73

Convolutional neural networks (CNNs)

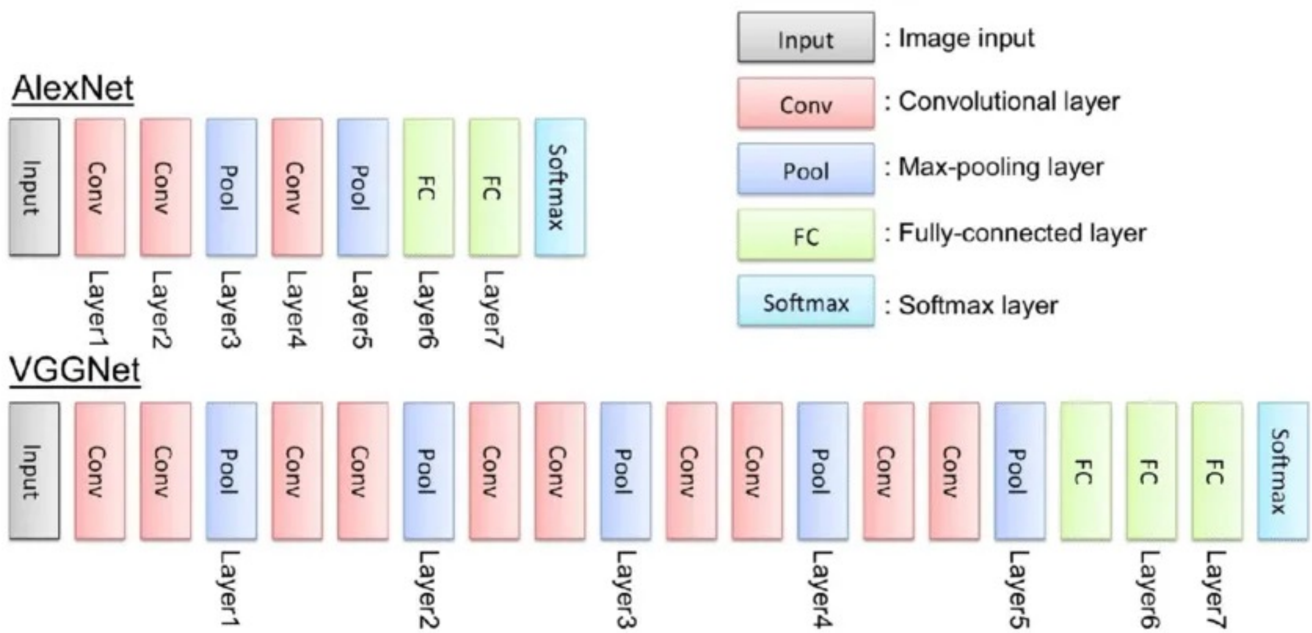
- Input data types are array/matrix/tensor
  - Image is an NxN array or tensor
- Structure of convolution layer
  - Convolution: sparse linear layer w/ weight sharing
  - Activation functions (usually ReLU)
  - Pooling: reduction operation on portions of array
- CNN:
  - Multiple convolutional layers
  - Fully-connected layers
  - Softmax





Example: VGGNet

- Input image: 224x224x3
- Conv: 3x3 convolutions/stride 1/padding 1, ReLU
- Pool: 2x2 max pooling/stride 2/no padding



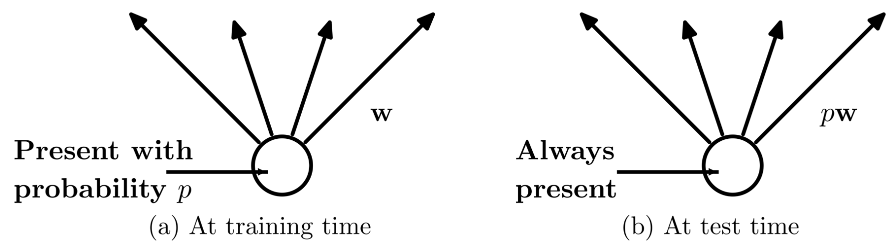
Parameters: 138,344,128
Memory: 200MB/image

For more details
Kaggle notebook

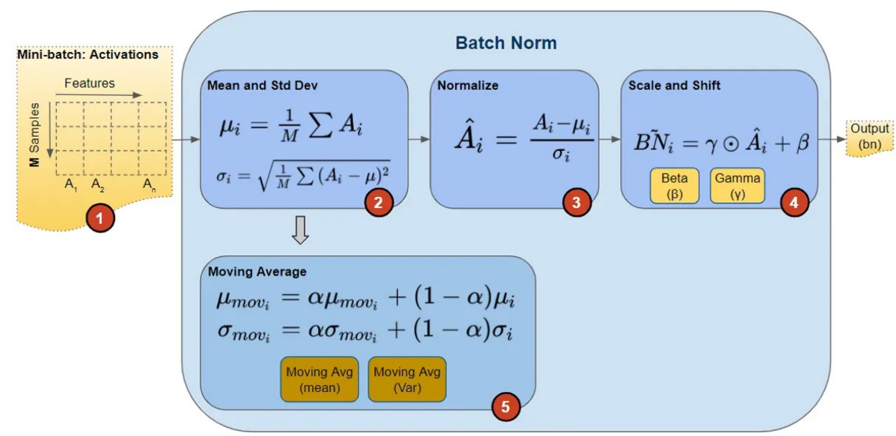
Variations (useful for other NNs as well)(I)

Drop-out: technique to avoid over-fitting

https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf



Batch normalization: like data normalization but performed on the fly during training on outputs of hidden layers

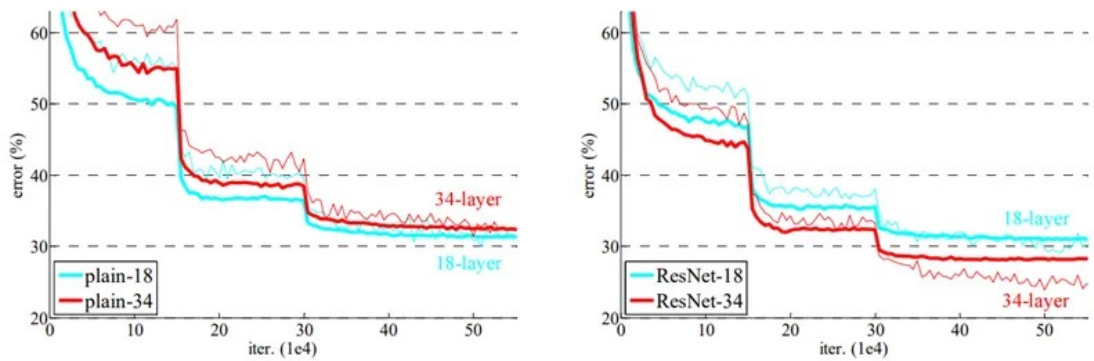


Informal introduction to batch normalization

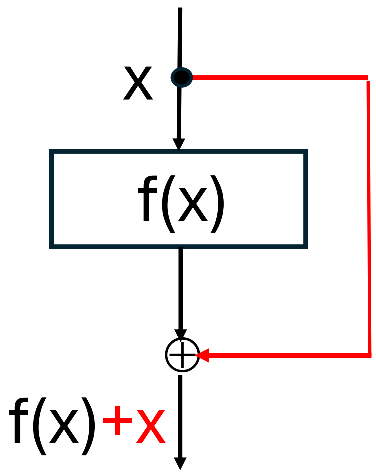
Variations (useful for other NNs as well)(II)

Residual (skip) connections

- Ameliorate problem of vanishing gradients
- First popularized in ResNet but used much earlier by McCulloch and Pitts, and Rosenblatt
- Permitted stacking of many more layers in CNNs



From ResNet paper



Online resources

CIFAR-10 demo running in your browser

- From Andrej Karpathy
- Shows improvements in accuracy for CIFAR-10 dataset during training

Fei-Fei Li's CNN course notes

- Stanford course cs231
- Leading expert on CNNs and image classification

Summary

Nonlinear functions used in NNs

- Sigmoid: $\Re \to \Re$
  - > Logistic function, ReLU, Leaky ReLU,....
- Softmax: $\Re^n \to \Re^n$
  - > Output can be interpreted as a probability distribution

Examples of NNs

- Implementing Boolean functions
- Stair image classifier

Loss functions for distributions

- KL-divergence, cross-entropy

Convolutional neural networks (CNNs)

- Inputs are matrices/tensors