Course progress

- Previously
  - PCA: dimensionality reduction (simple unsupervised learning)
  - OLS, ridge regression, conjugate gradient
  - Convex optimization, linear programming, Lagrange multipliers, duality and minimax games
  - Sparse regression (LASSO); NMF, Sparse PCA,
  - Gradient descent, dual ascent, dual decomposition, augmented Lagrangians, ADMM
  - Random sampling and randomized QR and SVD factorizations
  - Compressed sensing and matrix completion
  - DFT and FFT, shift invariant and circulant matrices/2D Fourier transform/filters
  - Graphs and their matrix representation, clustering
  - SGD, ADAGRAD, ADAM
- *classification problem, perceptron, logistic regression and neural networks*

General regression problems

- Given the data $(a_1, b_1)...(a_n, b_n)$ for $a_i \in \mathbb{R}^m$ and $b_i \in \mathbb{R}$ minimize

$$L(x) = \frac{1}{n} \sum_{i=1}^{n} \ell_i(x)$$

- in the ordinary least squares the loss function ℓ is the square loss

$$\ell_i(x) = (F(x, a_i) - b_i)^2$$

and the model F is linear

$$F(x, a_i) = x^T a_i$$

- We can instead use a complicated non-linear function $F(x, a_i)$, such as a PDE solution or a neural network with parameters x.
- Sometimes L is called is called empirical risk, and the training process is called empirical risk minimization

Binary classification problems

- Given $(a_1, y_1)...(a_n, y_n)$ for $a_i \in \mathbb{R}^m$ and labels $y_i \in \{0, 1\}$,

$$L(x) = \frac{1}{n} \sum_{i=1}^{n} \ell_i(x)$$

- The 0-1 loss function is not convex or differentiable

$$\ell_i(x) = \mathbb{1}_{F(x,a_i)y_i \leq 0}$$

- Similarly the perceptron classifier (Rosenblatt 1959) loss

$$F(x, a_i) = h(x^T a_i)$$

  is not differentiable
- h is the step function (also called Heavyside function)

$$h(t) = \begin{cases} 0 & t < 0 \\ 1 & t \geq 0 \end{cases}$$

- Can handle affine data $x^T a_i + \mu = x^T a_i + \frac{1}{m} \mathbb{1}^T \begin{bmatrix} \mu \\ \vdots \\ \mu \end{bmatrix}$

Perceptron

- Trained by going through the data and updating x if the ith point is misclassified

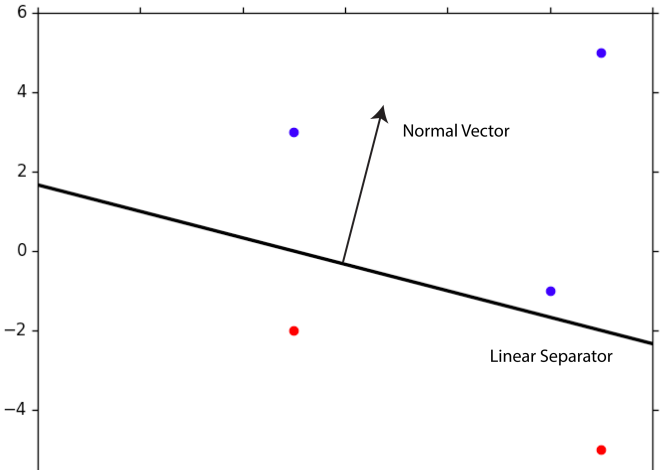
$$x \leftarrow x + r(y_i - \hat{y}_i)a_i$$

where $r > 0$ is the learning rate and $\hat{y}_i = h(x^T a_i)$ is the prediction

- Say we have $x^T a_i < 0$ which results in an incorrect classification $\hat{y}_i = 0$ for $y_i = 1$
- The training step will increase $x^T a_i$

$$(x + r(y_i - \hat{y}_i)a_i)^T a_i - x^T a_i = r(y_i - \hat{y}_i)\|a_i\| > 0$$

- Works if the data is linearly separable

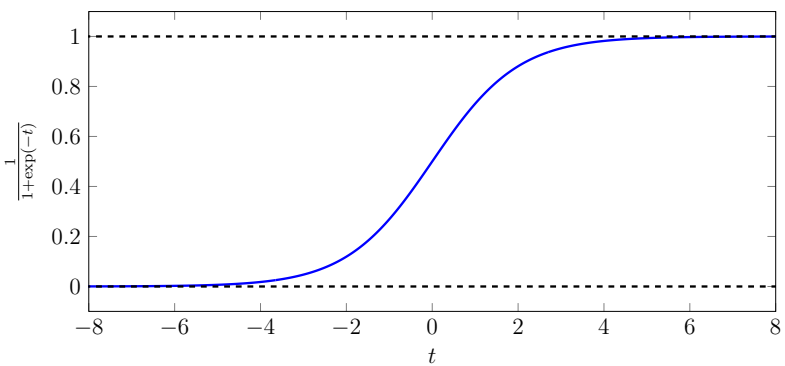


Logistic regression

- Replace the Heaviside function with a smooth version: logistic or sigmoid function

$$g(t) = \frac{1}{1+e^{-t}} = \frac{1}{2} + \frac{1}{2}\tanh\left(\frac{t}{2}\right)$$

and predict according to whether $g(x^T a_i)$ is closer to 0 or 1.



Logistic regression

- From the previous slide, predict according to whether

$$g(x^T a_i) = \frac{1}{1 + e^{-x^T a_i}}$$

  is closer to 0 or 1.
- You can think about this as a simple neural network with one input layer, no internal layers, one output layer and sigmoid activation function
- Note the the activation is function is approximately piecewise linear (piecewise constant)
- It is differentiable but not convex
- However $-\log g$ is convex, which guarantees convexity of the cross-entropy loss - we'll fill in the details on the hw)
- (More complicated NN models give rise to noncovex loss functions, and a big question is why training using convex optimization methods, like SGD, works in practice)

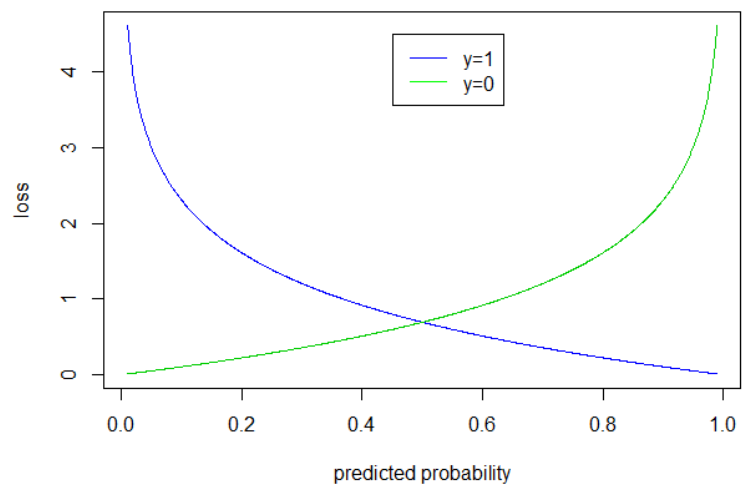
Cross-entropy loss

- Since the 0-1 loss is not convex or differentiable, can we use another loss func? So called cross-entropy:

$$L(y, g) = -\frac{1}{n} \sum_{i=1}^{n} y_i \log g_i + (1 - y_i) \log(1 - g_i)$$

- Predict g_i according to the logistic regression below or another model

$$g_i(x) = \frac{1}{1 + e^{-x^T a_i}}$$



- What is the intuition behind this type of loss function?

Cross-entropy loss

- Assume that $y_1, ..., y_n$ are i.i.d samples from Bernouilli RV with mean $Prob(y_i = 1) = g_i(x)$, i.e., coin flips with bias g_i
- The likelihood is

$$L(x) = \prod_{i=1}^{n} g_i(x)^{y_i}(1 - g_i(x))^{1-y_i}$$

- Then

$$-\frac{1}{n}\log L(x) = -\frac{1}{n}\sum_{i=1}^{n} y_i \log g_i(x) + (1 - y_i)\log(1 - g_i(x))$$

- Again g_i can be predicted according to the logistic regression or another model

$$g_i(x) = \frac{1}{1 + e^{-x^T a_i}}$$

- Although the probabilistic assumptions are often not realistic, the resulting loss function penalizes classification errors in a smooth way and is easy to optimize
- Cross-entropy is commonly used in practice

Multiclass classification

- For $(a_1, y_1)...(a_n, y_n)$ for $a_i \in \mathbb{R}^m$ and multiclass (categorical) labels y_i

$$L(x) = \frac{1}{n} \sum_{i=1}^{n} \ell_i(x)$$

- The loss function that simply counts the number wrong classifications (so called Hamming loss) is not convex or differentiable

$$\ell_i(x) = \mathbb{1}_{F(x,a_i) \neq y_i}$$

- If the model F outputs discrete labels, e.g. k-dimensional canonical basis vector e_j with 1 in the position corresponding to the predicted category j and zeros elsewhere (*one-hot* encoding), it will not be convex/differentiable either

Softmax regression

- Let's predict the jth category according to

$$p(\theta) = \begin{bmatrix} p(y = e_1 | a, \theta_1) \\ \dots \\ p(y = e_k | a, \theta_k) \end{bmatrix} = \frac{1}{S} \begin{bmatrix} \exp(a^T \theta_1) \\ \dots \\ \exp(a^T \theta_k) \end{bmatrix}$$

  where $S = \sum_{j=1}^{k} \exp(a^T \theta_j)$ and predict the label of a datapoint a according to

$$\arg_j \max p_j(\theta)$$

- This generalizes logistic regression in the multiclass setting

Softmax regression in the binary setting

$$
\begin{aligned}
p(\theta) &= \frac{1}{S}\begin{bmatrix}\exp(a^T\theta_1)\\ \exp(a^T\theta_2)\end{bmatrix} \\
&= \frac{1}{\exp(a^T(\theta_1-\theta_2)+\exp(a^T 0)}\begin{bmatrix}\exp(a^T(\theta_1-\theta_2)\\ \exp(a^T 0)\end{bmatrix} \\
&= \frac{1}{\exp(-a^T x)+1}\begin{bmatrix}\exp(-a^T x)\\ 1\end{bmatrix} = \begin{bmatrix}1-g(x)\\ g(x)\end{bmatrix}
\end{aligned}
$$

where $S = \exp(a^T\theta_1) + \exp(a^T\theta_2)$, $x = -(\theta_1 - \theta_2)$ and g is the logistic function

Cross-entropy loss in the multiclass setting

- We have

$$L(y, g) = -\frac{1}{n} \sum_{i=1}^{n} \ell_i(y_i, g_i)$$

  where $y_i \in \mathbb{R}^k$ is the label associated with a_i and $g_i \in \mathbb{R}^k$ is the prediction of y_i, e.g. softmax.

- Then the loss function is given by

$$\ell_i(y_i, q_i) = \sum_{j=1}^{k} y_{j,i} \log g_{j,i}$$

- This is also known as a KL divergence between two probability distributions y_i and g_i

Limitations of linear models

- Linear classifiers, i.e. $g(x^T a)$, won't properly classify nonlinear data, such as XOR
- Can attempt to solve this problem by piecewise linear functions, such as

$$ReLU(x) = x_+ = max(0, x)$$

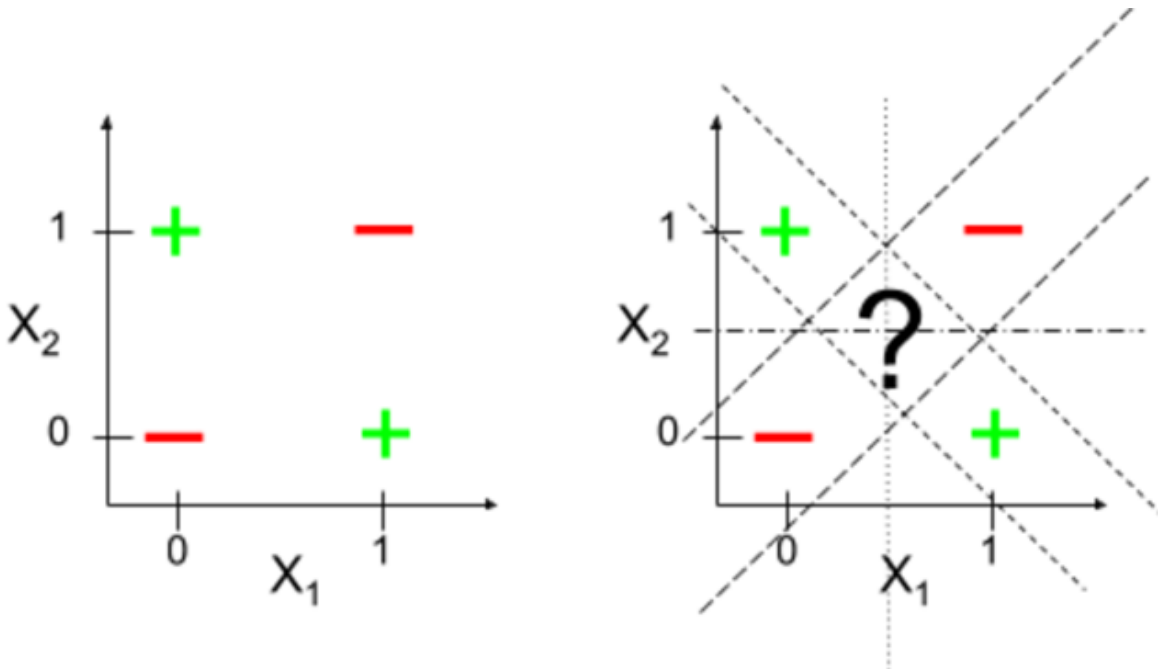


Figure: Fig from [3]

Feedforward neural networks

$$w = f(v) = A_2(A_1 v + b_1)_+$$

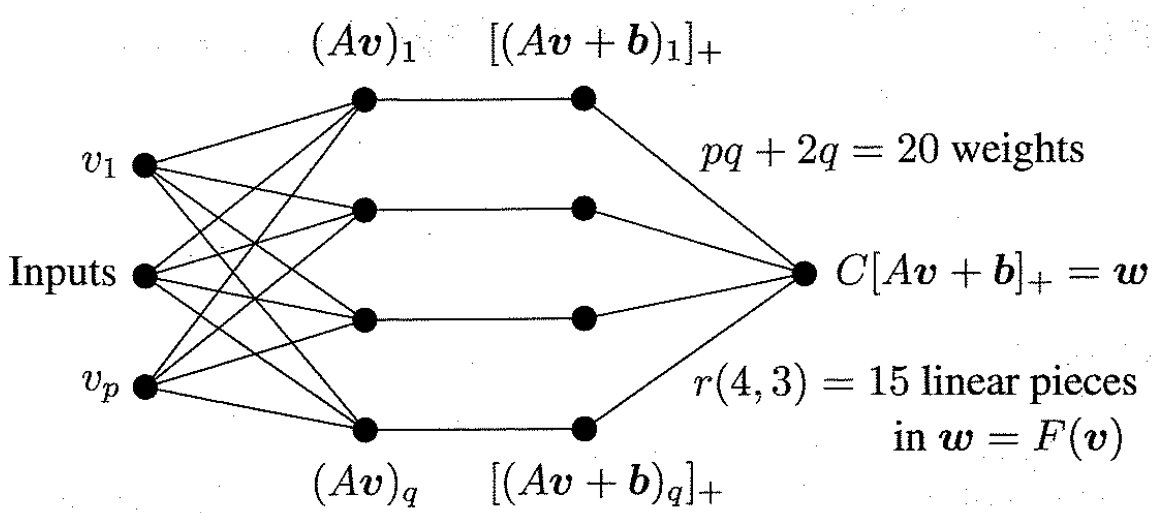


Figure: Fig from p 373 in [1]

Feedforward neural networks

$$v_2 = F(A_k, b_k, v_0) = A_2 v_1 + b_2 = A_2(ReLU(A_1 v_0 + b_1))$$

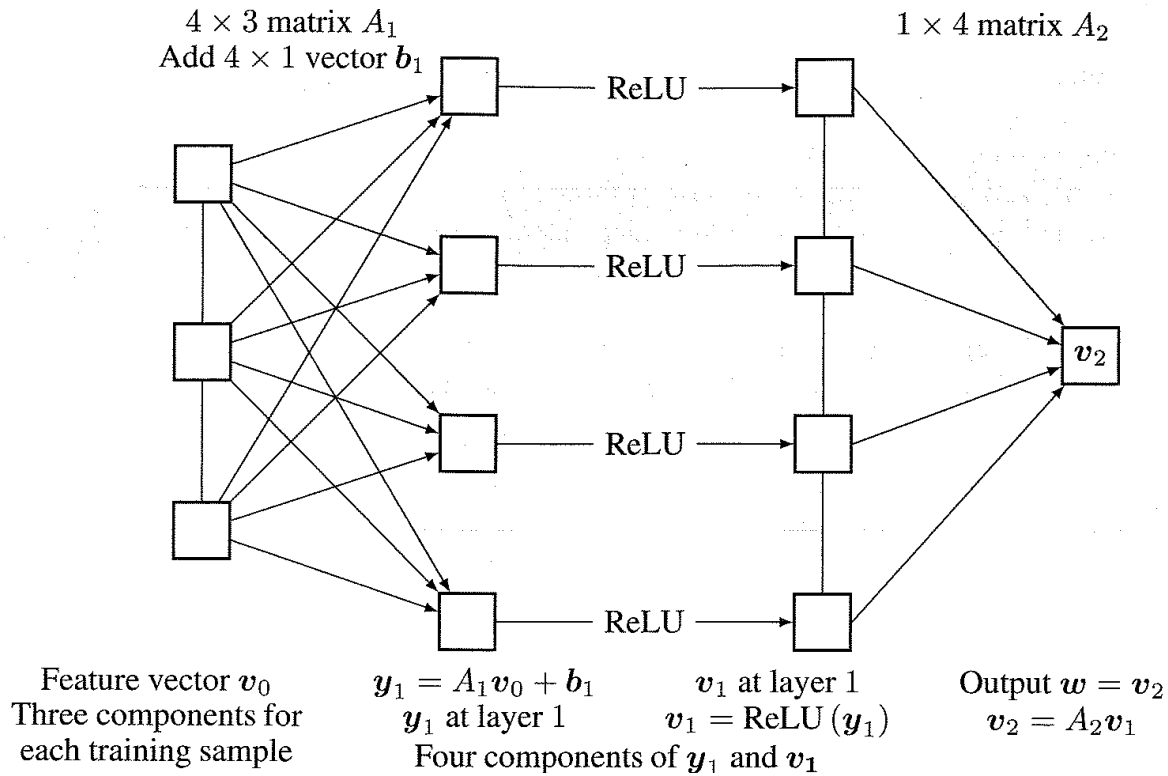


Figure: A feedforward neural net with 4 neurons in one internal layer. The output v_2 provides a binary classification of the input v_0 using 20 weights in A_k, b_1 Fig VII.2 from [1]

Activation functions

- Popular choices of nonlinear activation functions,

$$ReLU(x) = \max(0, x)$$

and $\tanh(x)$ so called sigmoid function (rescaled logistic function)

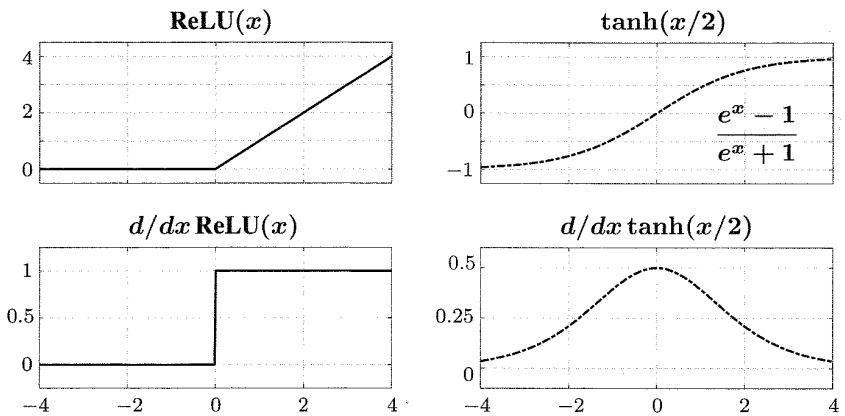


Figure: ReLU and sigmoid activation functions Fig VII.1 from [1]

Training

- Given $(a_1, y_1)...(a_n, y_n)$ for $a_i \in \mathbb{R}^m$ and binary $y_i \in \{0, 1\}$, minimize suitable loss function, such as the cross-entropy loss

$$L(x) = -\frac{1}{n} \sum_{i=1}^{n} y_i \log F_i(x) + (1 - y_i) \log(1 - F_i(x))$$

  where $x = (A_1, b_1), ..., (A_k, b_l)$ and F is the function that represents the NN.
- Typically use random initialization with a suitably chosen variance
- Too small variance will make too the norm of the weights too small, and therefore the gradient may vanish and too big variance may make the gradient diverge (if the loss is non-convex)
- Deep learning APIs like keras, provide standard initialization methods

Dimensionality reduction and max pooling

- We previously used subsampling of matrix columns to reduce the size of the matrix.
- Here, say for a weights matrix $A^{128\times128}$, we remove every second row $->$ $A^{64\times128}$ (stride 2)
- The size of the weights is more informative of the relationship between the inputs and outputs
- Max pooling for every pair, e.g. $(Ax)_2$ and $(Ax)_3$ take the largest
- Also can do this for 2D signal, e.g., for every 2x2 patch of an image, take the largest element

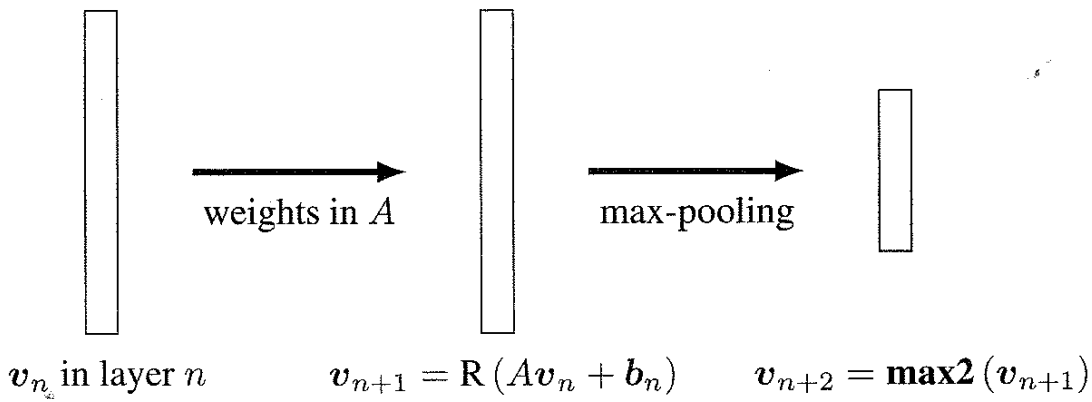


Figure: Illustration of max-pooling from p 379 from [1]

Expressivity of neural networks

- For a linear combination of nonlinear activation functions

$$F(A_{1:2}, b_{1:2}, v_0) = A_2(ReLU(A_1 v_0 + b_1)) + b_2$$

- ReLU applied to $A_1 v_0 \in R^4$ for $v_0 \in \mathbb{R}^3$, divides $\mathbb{R}^3$ nonlineraly, e.g.,

$$ReLU((A_1 v_0)_1), ReLU((A_1 v_0)_2), ReLU((A_1 v_0)_3)$$

  has "folds" along $(A_1 v_0)_i = 0$ dividing R^3 into 8 pieces.
- # folds is proportionate to the expressivity of the neural network
- Can give a more nuanced classification patterns than the hyperplanes in the case of linear classifiers.

Feedforward neural networks

- The graph of i-th component of $(A_1 v + b_1)_+$ has 2 half-planes

$$[(A_1 v + b_1)_+]_i = \begin{cases} 0 & \text{if } (A_1 v)_i + (b_1)_i < 0 \\ (A_1 v)_i + (b_1)_i & \text{otherwise} \end{cases}$$

- So if $A_1 \in \mathbb{R}^{q \times p}$, the input space $\mathbb{R}^p$ is sliced by q hyperplanes into r pieces

$$r(p, q) = \binom{q}{0} + \binom{q}{1} .... + \binom{q}{p}$$

$$r(4, 3) = 15$$

Convolution

- As you recall from previous lecture, the (linear) convolution between x and y is

$$(v * x)_k = \sum_j v_j x_{k-j},$$

- Multiplying by a Toeplitz matrix (i.e., matrix with constant diagonals) computes this convolution

$$A = \begin{bmatrix} x_1 & x_0 & x_{-1} & 0 & 0 & 0 \\ 0 & x_1 & x_0 & x_{-1} & 0 & 0 \\ 0 & 0 & x_1 & x_0 & x_{-1} & 0 \\ 0 & 0 & 0 & x_1 & x_0 & x_{-1} \end{bmatrix}$$

Figure: Conv net matrix of connections from p. 387 in [1]

- $k - j \in \{1, 0, -1\}$ implies $j \in k - \{1, 0, -1\}$
- E.g., for $k = 1$, we have $j \in \{0, 1, 2\}$, and

$$(Av)_1 = x_1 v_0 + x_0 v_1 + x_{-1} v_2$$

Convolution

- A is a combination of shift matrices: left(L), center (C), right(R)

$$A = x_1 L + x_0 C + x_{-1} R$$

- Then the derivatives of

$$y = Av$$

are

$$\frac{\partial y}{\partial x_1} = Lv, \ \frac{\partial y}{\partial x_0} = Cv, \text{ and } \frac{\partial y}{\partial x_{-1}} = Rv.$$

Convolution in 2D

- We saw that a 2D convolution is

$$(v * x)_{a,b} = \sum_j \sum_k v_{j,k} x_{a-j,b-k}$$

- Then we have 9 shifts instead of 3,

$$\begin{aligned} A = & x_{11} LU + x_{01} CU + x_{-11} RU \\ & + x_{10} L + x_{00} C + x_{-10} R \\ & + x_{1-1} LD + x_{0-1} CD + x_{-1-1} RD \end{aligned}$$

$$\textbf{Weights} \begin{bmatrix} x_{11} & x_{01} & x_{-11} \\ x_{10} & x_{00} & x_{-10} \\ x_{1-1} & x_{0-1} & x_{-1-1} \end{bmatrix} \quad \begin{array}{lll} \textbf{Input image} & v_{ij} & i,j \text{ from} (0,0) \text{ to} (N+1, N+1) \\ \textbf{Output image} & y_{ij} & i,j \text{ from } (1,1) \text{ to } (N,N) \\ \textbf{Shifts L, C, R, U, D} & & = \text{Left, Center, Right, Up, Down} \end{array}$$

Figure: Block of 2D conv net matrix of connections from p. 388 in [1]

Convolution in 2D

- Then the derivatives of

$$\begin{aligned} y = Av = &(x_{11}LU + x_{01}CU + x_{-11}RU \\ &+ x_{10}L + x_{00}C + x_{-10}R \\ &+ x_{1-1}LD + x_{0-1}CD + x_{-1-1}RD)v \end{aligned}$$

  are

$$\frac{\partial y}{\partial x_{11}} = LUv,\ \frac{\partial y}{\partial x_{01}} = CUv,\ \frac{\partial y}{\partial x_{-11}} = RUv, ..., \frac{\partial y}{\partial x_{-1-1}} = RDv$$

- CNN can accommodate multiple channels, including features like RGB
- Then these weight matrices become order 3 tensors

CNNs

- State of the art for many tasks, like image recognition
- HW: detected edges by convolution with a Laplacian-like filter
- Other filters like Gaussian, provide smoothing/denoising

$$Gf(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) * f$$
$$= \frac{1}{\sqrt{2\pi}\sigma}\exp(-x^2/(2\sigma^2)) * \frac{1}{\sqrt{2\pi}\sigma}\exp(-y^2/(2\sigma^2)) * f$$

(convolution is associative; don't need brackets on the RHS)

- Gaussian is positive and integrates to one - weighted average
- in the discrete case, the radial symmetry is lost and the product of 1D filters is not exact

$$G = \frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \approx \frac{1}{289}\begin{bmatrix} 1 \\ 4 \\ 7 \\ 4 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \end{bmatrix} \quad (3)$$

Figure: Equation (3) from p. 389 in [1]

Gradient detection

- We saw that a convolution between x and y is

$$(v * x)_k = \sum_j v_j x_{k-j},$$

- Is equivalent to multiplying by a Toeplitz matrix

$$A = \begin{bmatrix} x_1 & x_0 & x_{-1} & 0 & 0 & 0 \\ 0 & x_1 & x_0 & x_{-1} & 0 & 0 \\ 0 & 0 & x_1 & x_0 & x_{-1} & 0 \\ 0 & 0 & 0 & x_1 & x_0 & x_{-1} \end{bmatrix}$$

Figure: Conv net matrix of connections from p. 387 in [1]

- $k - j \in \{1, 0, -1\}$ implies $j \in k - \{1, 0, -1\}$
- Taking $(x_1, x_0, x_{-1}) = (-1/2, 0, 1/2)$ gives us centered differences
- E.g., $k = 1$, $j \in \{0, 1, 2\}$, and

$$(Av)_1 = x_1 v_0 + x_0 v_1 + x_{-1} v_2 = \frac{1}{2}(v_2 - v_0)$$

Gradient detection in 2D

- We saw that a 2D convolution is

$$(v * x)_{a,b} = \sum_j \sum_k v_{j,k} x_{a-j,b-k}$$

- The Sobel operators combine smoothing (averaging) with differentiation

$$\frac{\partial}{\partial x} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1/2 & 0 & 1/2 \end{bmatrix} \text{ and } \frac{\partial}{\partial x} = \begin{bmatrix} -1/2 \\ 0 \\ 1/2 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

- Also while in the above filters we prescribed coefficients corresponding to smoothing/averaging, gradient/edge detection, etc, in a CNN these coefficients will be learned from the data during training

CNNs and Fourier representation

- We have been able to compute convolutions efficiently in the Fourier domain
- But we don't expect that the Fourier representation of the signal will be well behaved with respect to the nonlinear activation functions and max-pooling layers in the spatial domain in a typical CNN
- Therefore, we normally don't use FFT, etc to optimize/speedup a CNN

Dimensionality reduction and max pooling

- We can also use (lossy) filters to reduce dimensionality

Stride $S = 2$

$$A = \begin{bmatrix} x_1 & x_0 & x_{-1} & 0 & 0 \\ 0 & 0 & x_1 & x_0 & x_{-1} \end{bmatrix} \tag{6}$$

Figure: Equation (6) from p. 390 in [1]

- Since the size of the weights is more informative of the relationship between the inputs and outputs
- Max pooling for every pair, e.g. $(Ax)_2$ and $(Ax)_3$ take the largest
- Also can do this for 2D signal, e.g., for every 2x2 patch of an image, take the largest element

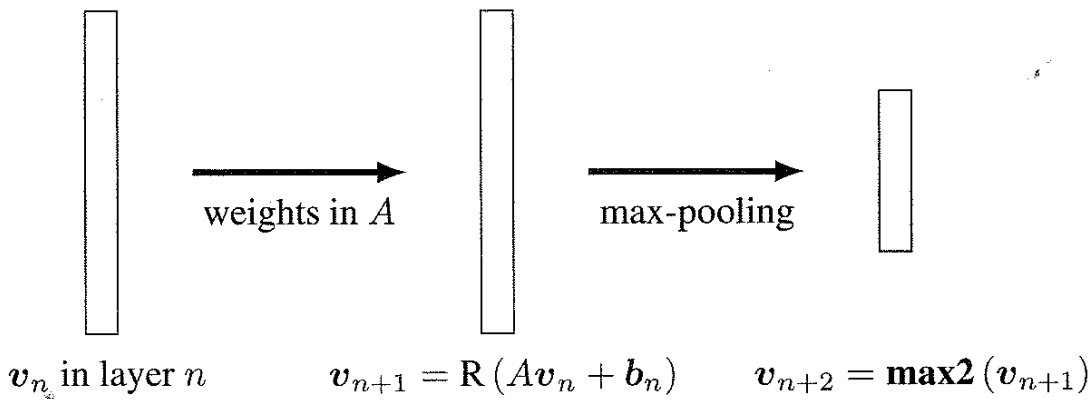
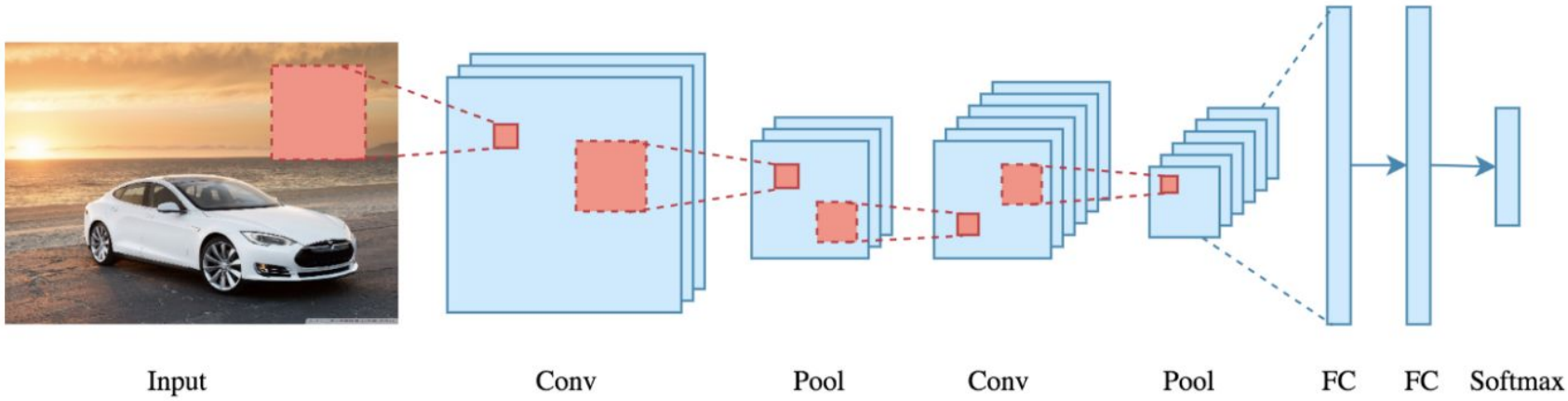


Figure: Illustration of max-pooling from p 379 from [1]

CNN architecture



https://towardsdatascience.com/

applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2#9722

- AlexNet: The original ICLR paper's primary result was that the depth of the model was essential for its high performance on the ImageNet data set
- Used 11x11, 5x5 and three 3x3 convolution kernels
- Uses mostly max pooling but also some strides
- Ended with several fully connected layers and a softmax layer

Next steps

- Back-propagation and chain rule (Sec. VII.3)
- Hyperparameters (Sec. VII.4)