Deep Learning in a Nutshell

TODAY:

1) linear regression
2) logistic regression
3) neural networks
   a) fully connected layers
   b) convolutional layers

recall: ML problem:

1) data X
2) model: $\theta \longmapsto g_\theta$, θ: model params
3) cost function:

$$X, g_\theta, \theta \longmapsto E(X, g_\theta)$$

evaluates how well the model works on the data

→ decompose data: $X = X_{training} \cup X_{test}$

1) training data/set → determine optimal θ

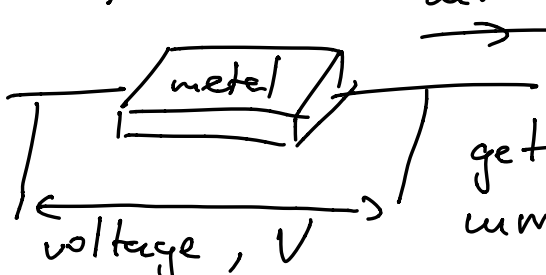
2) test data/set → evaluate the final model performance

$X_{train} \cap X_{test} = \emptyset$

3) validation set → (HW, review Mehta, et al)

Linear Regression

example:



get a measurement of current I at a fixed voltage V

$X = \{(I_i, V_i)\}_{i=1}^{n}$, n : measurements

want: to know the dependence b/w V & I

assume a model: $V_i = g_\theta(I_i)$

↳ model params, trainables, learnables

e.g. $g_\theta(I) = \theta I$: θ is some proportionality const. (resistance)

or $g_\theta(I) = \theta I^2$

or $g_{\vec{\theta}}(I) = \theta_0 + \theta_1 I + \theta_2 I^2$, $\vec{\theta} = (\theta_0, \theta_1, \theta_2)$

g_θ is linear function of $\vec{\theta}$!

- more generally: p variables $(x^1, x^2, \dots, x^p) = \vec{x}$

$\Rightarrow \mathcal{X} = \{(\vec{x}^{(i)}, y_i)\}_{i=1}^{n}$

- cost function for linear regression

$E(\theta) = \frac{1}{n} \sum_{i=1}^{n} \| y_i - g_\theta(\vec{x}^{(i)}) \|^2$ least square loss

linear regression: $g_\theta(\vec{x}^{(i)}) = \vec{\theta}^t \cdot \vec{x}^{(i)}$

$\vec{\theta} \in \mathbb{R}^p$

→ can find exactly (analytically) the optimal parameters θ which minimize $E(\theta)$ → (HW, review)

problem: applying GD may fine-tune the optimal θ to the training data (b/c test data is not used for training)

→ testing (i.e. evaluating model g_θ on test data) won't work well

→ overfitting

- regularization: want to avoid/prevent overfitting

a) L^2 regularizer: add $\|\vec{\theta}\|_2^2$ to cost function

$$E(\theta) = \frac{1}{n}\sum_{j=1}^{n}\left[y_j - g_\theta(\vec{x}^{(j)})\right]^2 + \lambda\|\vec{\theta}\|_2^2$$

$$\|\vec{\theta}\|_2^2 := \sum_{u=1}^{p}\theta_u^2 = \vec{\theta}^{T}\cdot\vec{\theta}$$

λ: regularization strength (hyperparameter)

(one can still find exact solution for optimal θ $\longrightarrow$ Ridge regression)

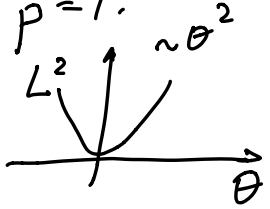
b) L^1 regularizer: add $\|\vec{\theta}\|_1$ to cost function

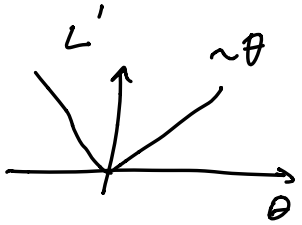
$$E(\theta) = \frac{1}{n}\sum_{j=1}^{n}\left[y_j - g_\theta(\vec{x}^{(j)})\right]^2 + \lambda\|\vec{\theta}\|_1$$

$$\|\vec{\theta}\|_1 = \sum_{j=1}^{p}|\theta_j|$$

- enforces sparsity of $\vec{\theta}$, i.e. only a few of $\vec{\theta}$ will be non-zero

for $p=1$:





- Rmk: 1) $y_i \in \mathbb{R}$ continuous variable

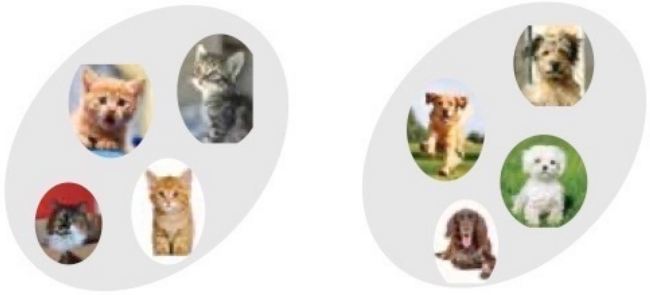
2) in RL, if we want to learn approx. to V_π or Q_π, then we can "regress" on Bellman error $\delta_t = r_{t+1} + V(s_{t+1}) - V(s_t)$

Classification

y_i are discrete, e.g. only take values from $n = 0, 1, \ldots, M-1$

where M: number of classes/categories

example: binary classifier: tell if an image is showing a cat or a dog



$y_i = 0$ $\quad$ $y_i = 1$

Perceptron model:

def.: $s_j = \vec{w}^t \cdot \vec{x}_j + b = \vec{\theta}^t \cdot \vec{x}$

$\vec{w}$: params of model

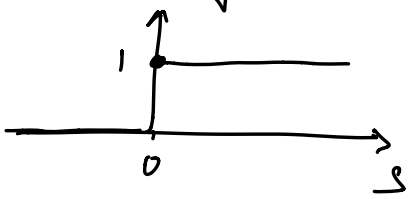
$\vec{\theta} = (\vec{w}, b)$

$\vec{x} = (\vec{x}, 1)$

$$g_\theta(\vec{x}_j) = \frac{1}{2}\left[\text{sign}(s_j) + 1\right] = \begin{cases} 0, & \text{if } s_j < 0 \\ 1, & \text{if } s_j \geq 0 \end{cases}$$

↑ two categories

$f(s) = \frac{1}{2}(\text{sign}(s) + 1)$



→ use linear regression

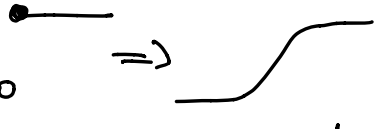
problem: make large mistakes whenever $s_j \approx 0$

idea: instead of learning labes $\{0, 1\}$, learn the probability for a data point x_j to in category 0 or 1

then: classification:

$$y_{pred} = \underset{j \text{ in categories}}{\text{argmax}} \; P(y = j \mid x, \theta)$$

Logistic regression:

smooth out sign-function: 

def:

sigmoid $\sigma(s) := \frac{1}{1+e^{-s}}$ (Fermi-Dirac distribution)

def:

$$P(y_i = 1 \mid \vec{x}_i, \theta) = \sigma(\theta^t \cdot \vec{x}) = \frac{1}{1+e^{-\vec{\theta}^t \vec{x}}}$$

$$P(y_i = 0 \mid \vec{x}_i, \theta) = 1 - P(y_i = 1 \mid \vec{x}_i, \theta) = 1 - \sigma(\theta^t \cdot x_i)$$

- cost function: cross entropy loss

1) p_{labels}: delta-function prob (deterministic)
→ know label with certainty

2) $p_{data} = P(y_i \mid x_i, \theta) = \sigma(\theta \cdot x_i)$
↑ unknown parameters

goal: find θ, s.t. $p_{labels} \approx p_{data}$

def: distance on space of distributions

Kullback-Leibler divergence:

$$D_{KL}(p_{labels} \,||\, p_{data}) = \sum_{m=0}^{1} p_{m,labels} \log \frac{p_{m,labels}}{p_{m,data}}$$

$D_{KL}(p_{labels} \,||\, p_{data})$

$$= \sum_m \overbrace{p_{m,labels} \log p_{m,labels}}^{\text{indep. of } \theta} - p_{m,labels} \log p_{m,data}$$

minimize D_{KL} w.r.t. θ

cross entropy loss:

$$E(\theta) = -\sum_i \sum_{m=0}^{1} p_{m,labels}(x_j^i) \log p_{m,data}(x_j^i)$$

only two classes

$$= -\sum_i y_j \log \sigma(\theta^t \cdot x_j) + (1-y_i) \log (1-\sigma(\theta^t \cdot x_j))$$

→ use GD or ADAM to find optimal θ

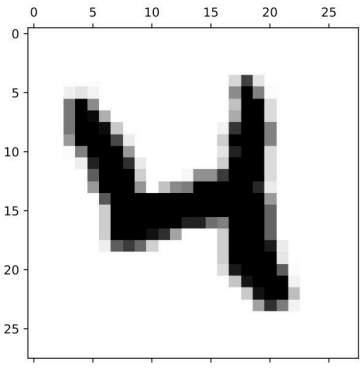
Softmax regression

what if we have more than two categories?

example: MNIST problem

→ classify hand-written digits

[0] [1] [2] ... [9]



classify each hand-written image into its category, i.e. the which digit is shown on image

→ a total of 10 digits: 0, 1, .., 9 ⇒ $M = 10$

want to generalize logistic regression:

example (statistical mechanics)

$M=2$: consider a two-state system coupled to a thermal bath ($T=\beta^{-1}$)

→ $y_i = \{0, 1\}$ with energies ϵ_0, ϵ_1

— $1, \epsilon_1$

— $0, \epsilon_0$

find the probability for system to be in state y_i!

$$p(y_i=1) = \frac{e^{-\beta\epsilon_1}}{e^{-\beta\epsilon_0} + e^{-\beta\epsilon_1}} = \frac{1}{1+e^{-\beta\Delta\epsilon}}$$

$$p(y_i=0) = \frac{e^{-\beta\epsilon_0}}{e^{-\beta\epsilon_0} + e^{-\beta\epsilon_1}} = \frac{1}{1+e^{+\beta\Delta\epsilon}}$$

$$\Delta\epsilon = \epsilon_1 - \epsilon_0$$

sigmoid

→ back to softmax

def: one-hot embedding

0	1	0	1	2
↓	↓	↓	↓	↓
[1,0]	[0,1]	[1,0,0]	[0,1,0]	[0,0,1]
$M=2$				

$\Rightarrow y_j \longrightarrow \vec{y}_j : \quad y_{jm} \in \{0,1\}$

$m = 0, \dots, M-1$

$\hookrightarrow$ # classes

— softmax prob. $\vec{\theta}_1, \dots, \vec{\theta}_{M-1}$

$$P(y_{im'} = 1 \mid \vec{x}_i, \{\vec{\theta}_k\}_{k=0}^{M-1}) = \frac{e^{-\vec{\theta}_{m'}^t \cdot \vec{x}_i}}{\sum_{m=0}^{M-1} e^{-\vec{\theta}_m^t \cdot \vec{x}_i}}$$

data point ↑ ↑ category

— cost function: (cross entropy)

$$E(\theta) = -\sum_{j=1}^{n} \sum_{m=0}^{M-1} y_{jm} \log P(y_{jm} = 1 \mid \vec{x}_j, \{\vec{\theta}\})$$

Neural Networks

recall:

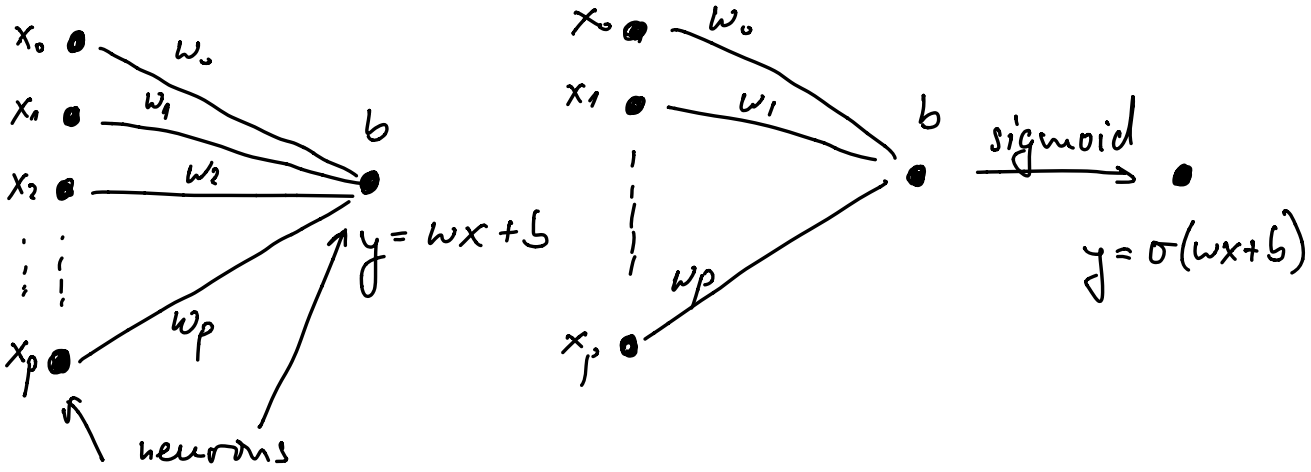
linear regression

$y = w \cdot x + b$

logistic regression

$y = \sigma(wx + b)$

graphical representation:



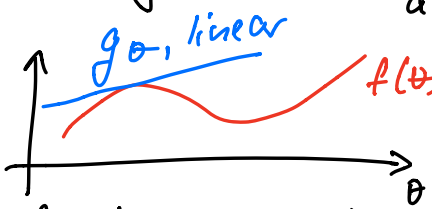
linear regression is a single-layer neural network w/o activation function
(DNN which are linear model linear regression)

logistic regression
a single-layer NN with a sigmoid activation function

issues:

1) (linear regression): what if true model is actually non-linear?

e.g.



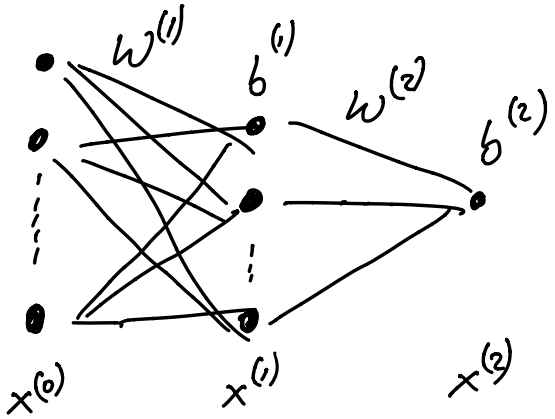
→ design non-linear features by hand
(recall $g_\theta = \theta_0 + \theta_1 I + \theta_2 I^2$)

Hard!

we'd like to be able to learn features, i.e. determine features variationally
→ need a nonlinear model! (e.g. sigmoid)

2) cannot learn correlations between values of input neurons with a single layer

idea: add another layer of neurons
— Deep Neural Network (DNN)



$x^{(0)}$: input data/layer

$$x_i^{(1)} = f^{(1)}\left(W_{ij}^{(1)} x_j^{(0)} + b_i^{(1)}\right)$$

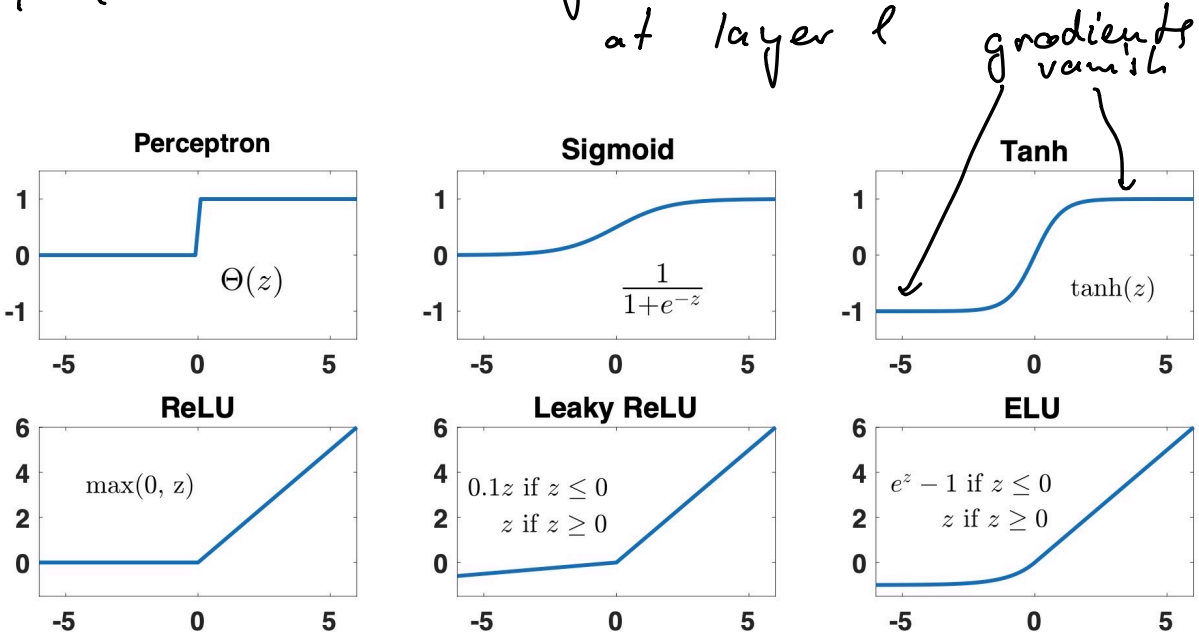
$$x_i^{(2)} = f^{(2)}\left(W_{ij}^{(2)} x_j^{(1)} + b_i^{(2)}\right)$$

⋮

$W_{ij}^{(\ell)}$: weight matrix of layer ℓ

$b_i^{(\ell)}$: bias vector of layer ℓ

$f^{(\ell)}(z)$: non-linearity / activation function at layer ℓ



- how do we find parameters ?

$$\theta = \{(W^{(\ell)}, b^{(\ell)})\}_{\ell=0}$$

- if $y \in \mathbb{R}$ is continuous

→ least square loss : $E(\theta) = \frac{1}{n} \sum_{i=1}^{n} [y_i - g_\theta(x_i)]^2$ (DNN_θ)

- if $y \in \mathbb{Z}$ is discrete

→ cross entropy : $E(\theta)$

- to do GD, we need partial derivatives

$\frac{\partial E}{\partial w_{ij}^{(l)}}$, $\frac{\partial E}{\partial b_i^{(l)}}$ → use Backpropagation (→ review, Chapter 9)

- there exists a theorem (expressivity theorem) which states that one can approximate any function with a single layer non-linear neural network, provided that the layer can be arbitrarily wide (i.e. # of neurons can go to infinity)
→ DNN's are called universal approximators

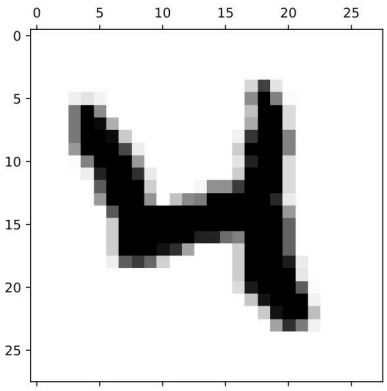
Convolutional Neural Nets

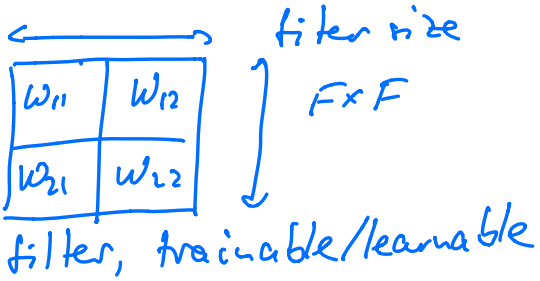
- how can we capture spatial correlations in the data?

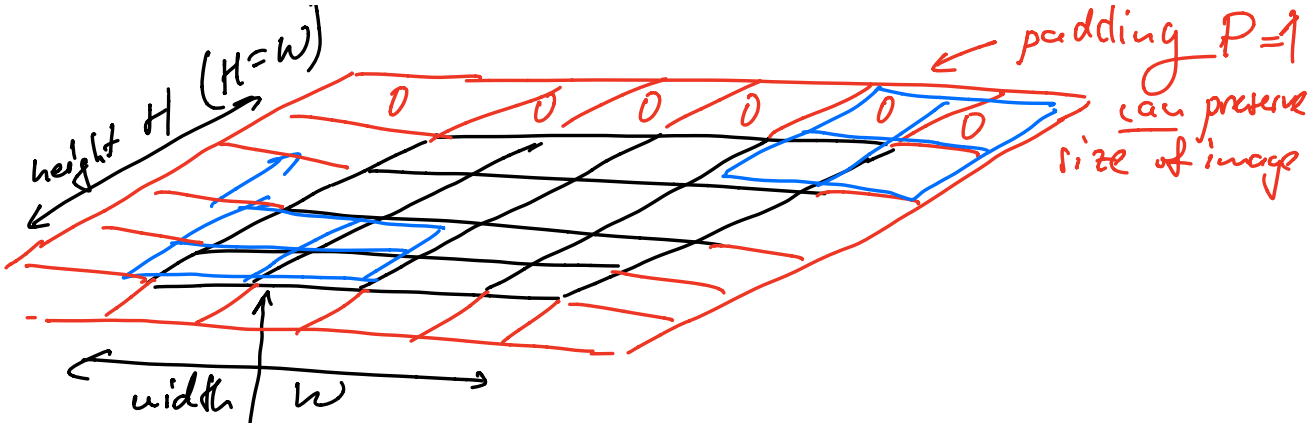
idea: consider so-called filters: local maps over input of size

$F \times F \times D$

- F: width of filter
- F: height of filter
- D: number of filters







padding P=1 can preserve size of image

data point (2D image)

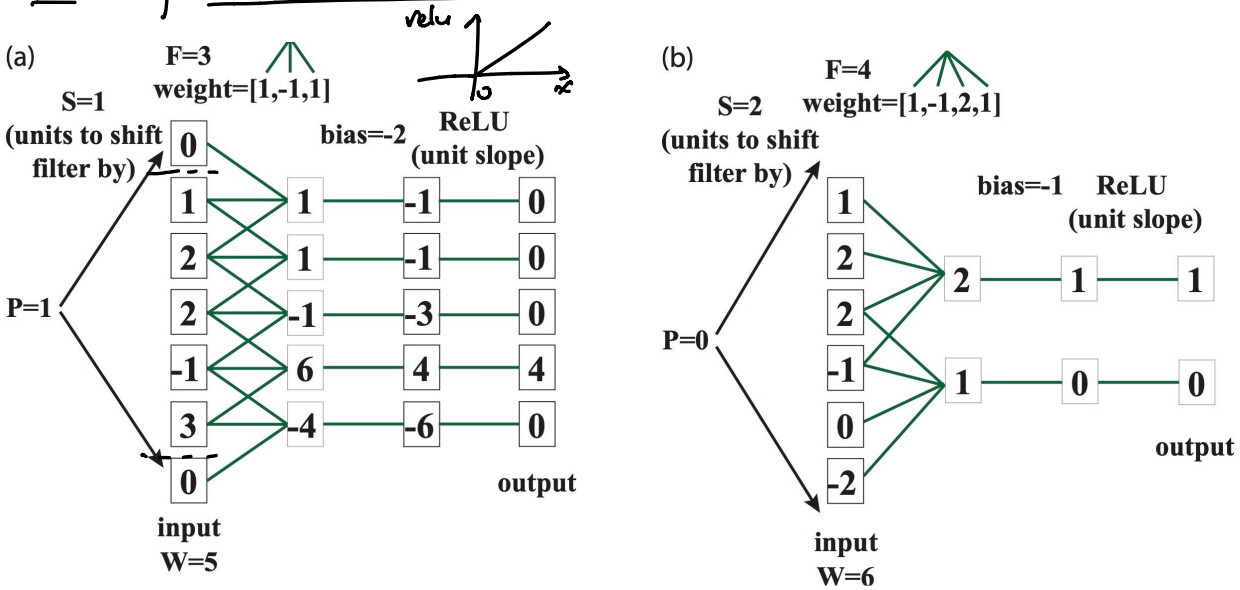
number of neurons in the filter-output layer:

$$\frac{W - F + 2P}{S} + 1$$

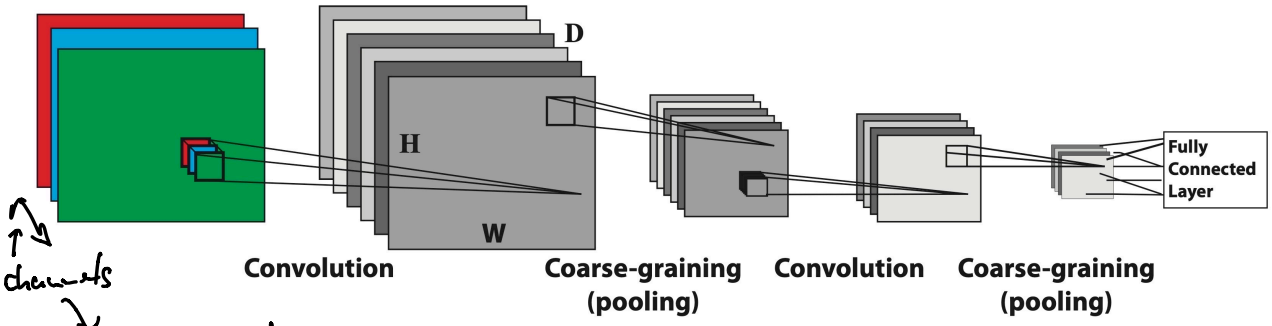
(F: filter size; P: padding; W: width of image (W=H))

S: stride: how many neurons we shift the filter by

– examples of 1D convolution:



- example: generic CNN architecture



(1, 3, W, H)

↑ # images

- other types of deep networks:

- recurrent neural net (RNN)
  - → capture causality: e.g. data is a time series
    - LSTM
- residual neural nets
  - → skip connections, etc.

HW: explore JAX documentation