Computer vision showcases

- Medicine
- Social media
  instagram, snapchat filters, face id, images upscaling
- OCR (optical characters recognition)
- CCTV
  car plates detections / recognition, monitoring safety vidations, person identification
- Satellites
  agricultar segmentation (crops), area calcution
- AI in defects detection
  welding seams inspection
- AI in agriculture
- AI in autonomous driving
  traffic tracking, LIDAR data processing, predicts future states of cars, pedestrians, lane and road recognition, driver perception

Tips & tricks on libraries

torch - research-friendly library for CV

torchvision - contains datasets

datasets.MNIST - hand-written digits

Dataloader - special PyTorch class to load dataset

Neural network definition in PyTorch

1. Extend nn.Module
2. Override __init__ to set up layers
3. Override forward(x)

loss function is called criterion.

Train the model

1. Put model in train mode by calling net.train()
2. Run inference by calling invoke operator ()
3. Calculate loss by calling criterion
4. Set gradients to zero by calling zero_grad()
5. Call backward from loss to calculate gradients
6. Do update by calling optimizer.step()

net.eval() puts network in the inference mode

Computational graph

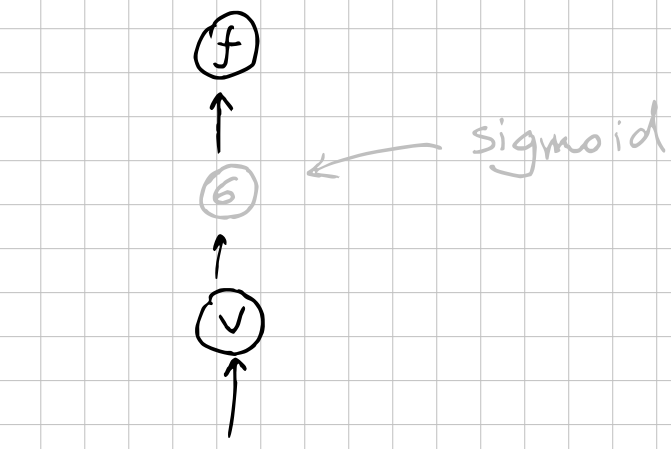
Idea: we can represent sequence of calculations as a graph;

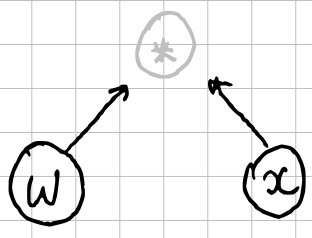
Let's take a sigmoid function as an example:

$$\sigma(x) = \frac{1}{1 + \exp(-(w \cdot x + b))},$$

where w, b are parameters

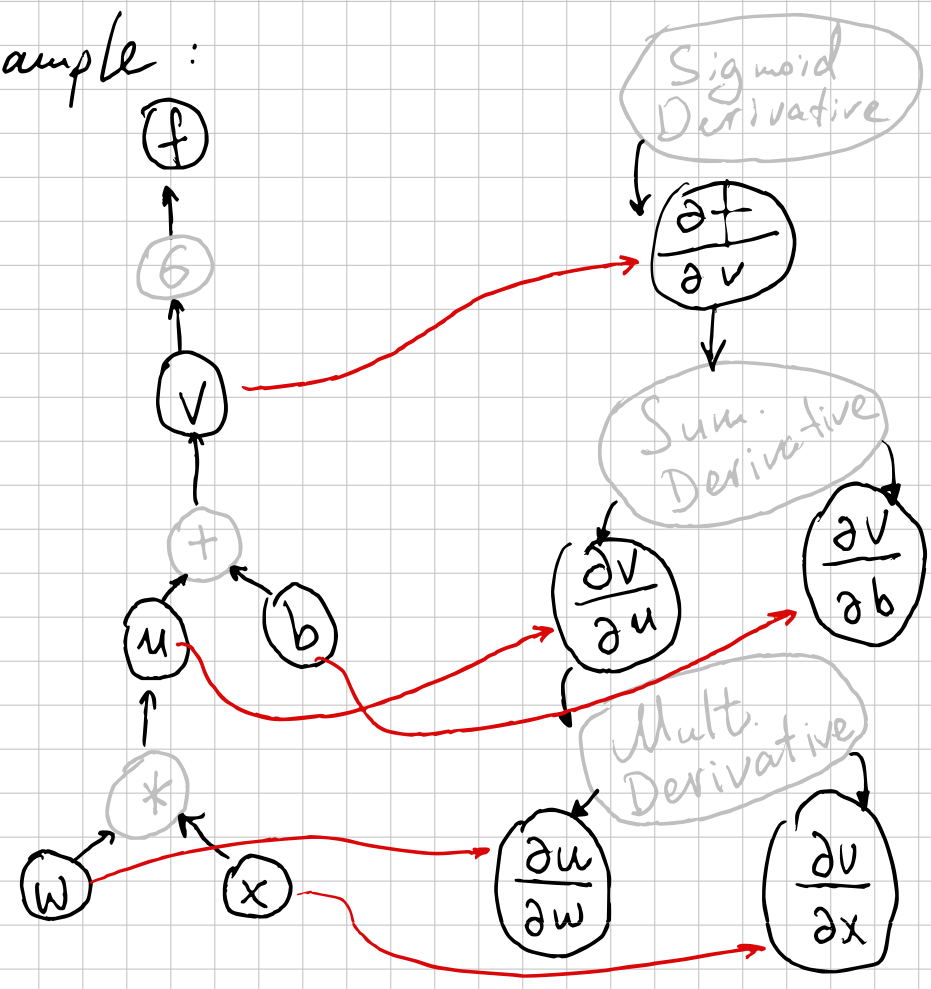
Example





- We can calculate gradients in parallel to forward pass;

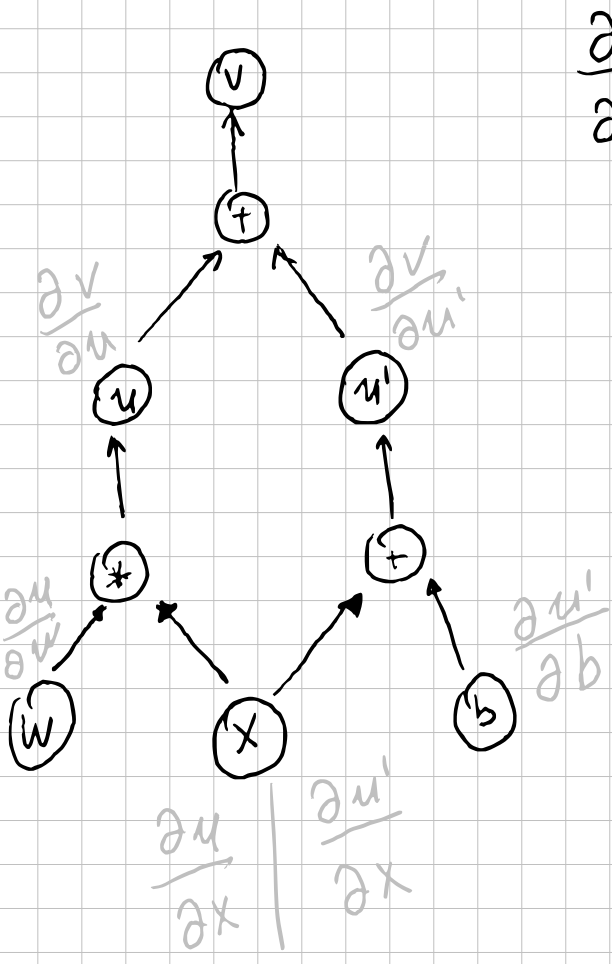
Example:



Chain rule helps us to calculate gradients of the output with respect to other variables:

> If $f(v) = f(u(v))$, then
>
> $$\frac{\partial f}{\partial v} = \frac{\partial f}{\partial u} \cdot \frac{\partial u}{\partial v}$$

In order to calculate derivative of the output with respect to some variable we need all partial derivatives along the way.



Computer vision problems

Problem statement

Input image $\underline{x}$

We want to make a prediction $\hat{y}$ based on an input image $\underline{x}$

$$\hat{y} = f(x)$$

We want to minimize empirical loss function

CV problems

- Classification: one & multiclass classification
- Detection
- Segmentation: semantic, panoptic, instance

CNNs

- Ingredients:
  1. An input image of size $\underline{H \times W \times C}$
  2. Choose the receptive field F (trad. $S \times S \times C$)
  3. For each pixel at position (i, j) calculate:

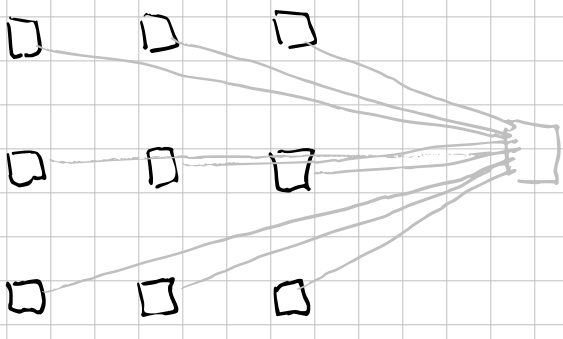
$$I'[i,j] = I[i:i+s, j:j+s, :] \odot F + b$$

  4. Do above steps as many times as you want.

Properties of CNNs

- We use parameter sharing;
- Captures "local" relations;
- Translation invariants;

Dilated convolutions



Attention mechanism

- Squeeze-and-Excitation Attention

Main CV losses

Metrics are:
- evaluation from human perspective;
- can be used in optimization (ensembling)

Examples: accuracy, f1, auc-roc, mae, rmse, iou, dice

Losses are:
- evaluation from model perspective
- can be used to evaluate by human
- can include regularizations
- in neural networks should be differentiable

Examples: bce, mse, mae, focal, dice

Binary cross-entropy

$\hat{y}, y$

X - picture

θ - parameters

$$p(y=1 \mid x; \theta) = \hat{y}_\theta(x)$$

$$p(y=0 \mid x; \theta) = 1 - \hat{y}_\theta(x)$$

$$p(y \mid x; \theta) = \hat{y}_\theta(x)^y \left(1 - \hat{y}_\theta(x)\right)^{1-y}$$

$$L = \prod_{n=1}^{N} (p(y)) \to max$$

$$\log L = \sum_{n=1}^{N} \left(y \log(\hat{y}) + (1-y)\log(1-\hat{y})\right) \to max$$

$$Loss = -\sum_{n=1}^{N} \left(y \log \hat{y} + (1-y)\log(1-\hat{y})\right) \to min$$

Mean squared error

$$y = \hat{y}_\theta(x) + \varepsilon, \quad \varepsilon \sim N(0, \sigma^2)$$

$$p(\varepsilon) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\varepsilon^2}{2\sigma^2}\right) \to$$

$$\to p(y) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y-\hat{y})^2}{2\sigma^2}\right)$$

$$L = \prod_{n=1}^{N} (p(y)) \to max$$

$$\log L = -C \sum_{n=1}^{N} (y-\hat{y})^2 \to max$$

$$Loss = \frac{1}{N} \sum_{n=1}^{N} (y-\hat{y})^2 \to min$$

Other losses

Classification losses:
- Cross-entropy for multiclass classification
- Focal loss
- Dice loss

Regression losses:
- Mean absolute error
- Huber loss

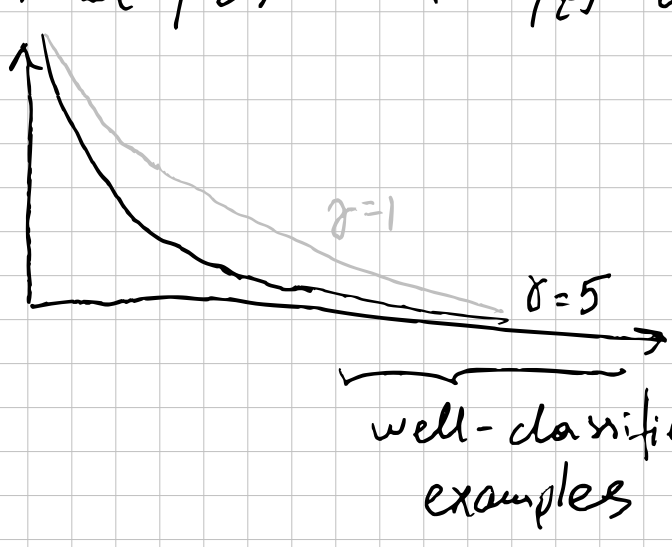
Cross-entropy for multiclass classification

$$CE = -\sum_{c=1}^{C} y \log(\hat{y}); \; C - \text{number of classes}$$

Focal loss

$$CE(p_t) = -\log(p_t)$$

$$FL(p_t) = -(1-p_t)^\gamma \log(p_t)$$



Dice loss

$$D = \frac{2\sum_i^N p_i g_i}{\sum_i^N p_i^2 + \sum_i^N g_i^2} \quad \text{i.e.}$$

$$\frac{2 \cdot |X \cap Y|}{|X| + |Y|} \approx \frac{2 \times \bigcirc}{\bigcirc + \bigcirc}$$

Mean absolute error

$$MAE = \frac{1}{N} \sum_{n=1}^{N} |y - \hat{y}|$$

* Used to ignore outliers

** Is not differentiable everywhere

Huber loss

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2, & \text{for } |a| \le \delta \\ \delta \cdot \left(|a| - \frac{1}{2}\delta\right), & \text{otherwise} \end{cases}$$

CV metrics

Classification metrics:

- Accuracy
- Precision and recall
- F1-score, Dice score
- IOU, Jaccard index
- AU-ROC, RP-curve

Regression metrics:

- Mean Absolute Error
- Mean Squared Error
- Root Mean Squared Error
- R^2

Confusion matrix:

TP	FN
FP	TN

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

$$\text{Precision} = \frac{TP}{TP+FP}$$

$$\text{Recall} = \frac{TP}{TP+FN}$$

$$\text{f1-score} = \frac{2 \times \text{precision} \times \text{recall}}{(\text{precision} + \text{recall})}$$

$$\text{IOU, Jaccard index} = \frac{TP}{TP+FP+FN}$$

ROC-AUC, RP-curve

Input: probabilities or scores

Sources of data and labels

Q: How to estimate amount of data?

A: Empirically

Depends on a task:

- Classification may require much more samples than, for example, segmentation

CIFAR10
60000
Image size: 32x32x3
Number of classes: 10
Accuracy: 99%

CIFAR100
60000
Image size: 32x32x3
Number of classes: 100
Accuracy: 96.8%

ImageNet - 1k
$1,3 \cdot 10^6$
Image size: 469x387x3
Number of classes: 1000
Accuracy: 88.3%

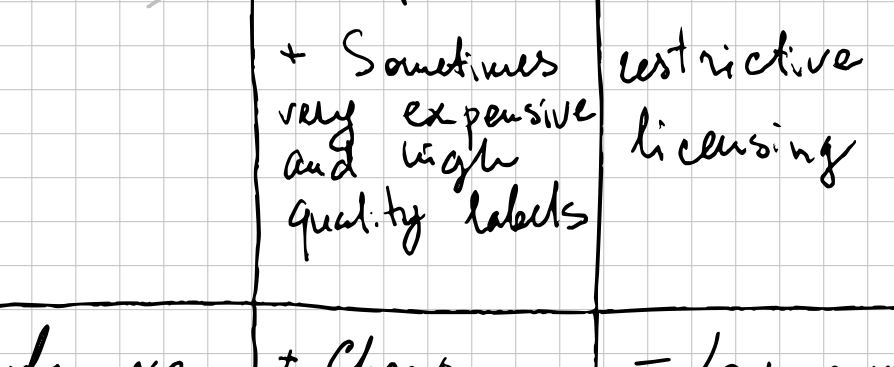
LUNA16 (lung cancer dataset)
888 CT scans
Image size: 512x512x~
Detection objects: 1186 nodules
Accuracy: 98.47% ROC-AUC

LITS2017
130 CT scans
Image size: 512x512x~
Task: Semantic segmentation
Accuracy: 89.85% IOU

Sources of data & labels

Source	Pros.	Cons
Open datasets (LUNA, MS COCO)	+ Free + Sometimes a lot of data + Sometimes very expensive and high quality labels	- Requires domain adaptation - Sometimes restrictive licensing
Crowdsource	+ Cheap + Well-developed instruments	- Low quality labels - Requires a lot of supervision
Labeled by experts	+ Full control of labeling process + Fits non-standard types of tasks	- Expensive - Requires a lot of supervision - Requires annotation tools

Labels from another source



For example, person goes to the office with their magnetic pass: we can match data from surveillance system and the pass (badge).

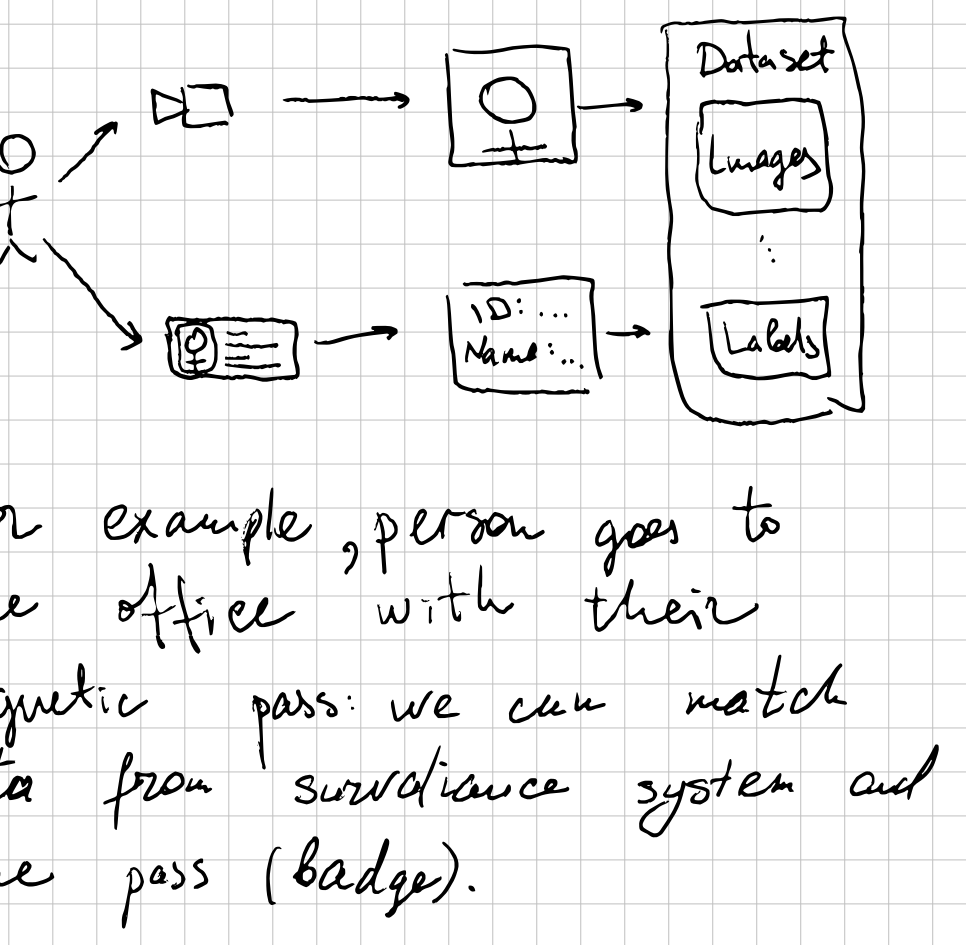
Pros

+ Opens new opportunities to get data
+ Sometimes free

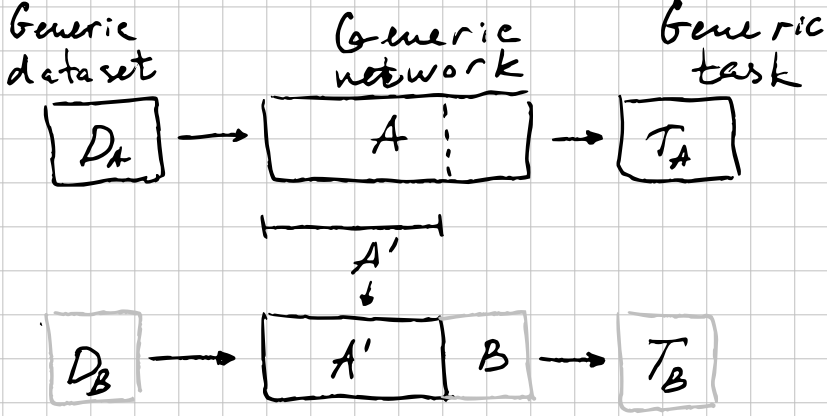
Cons

- Sometimes requires additional infrastructure

Labels from future



Transfer learning

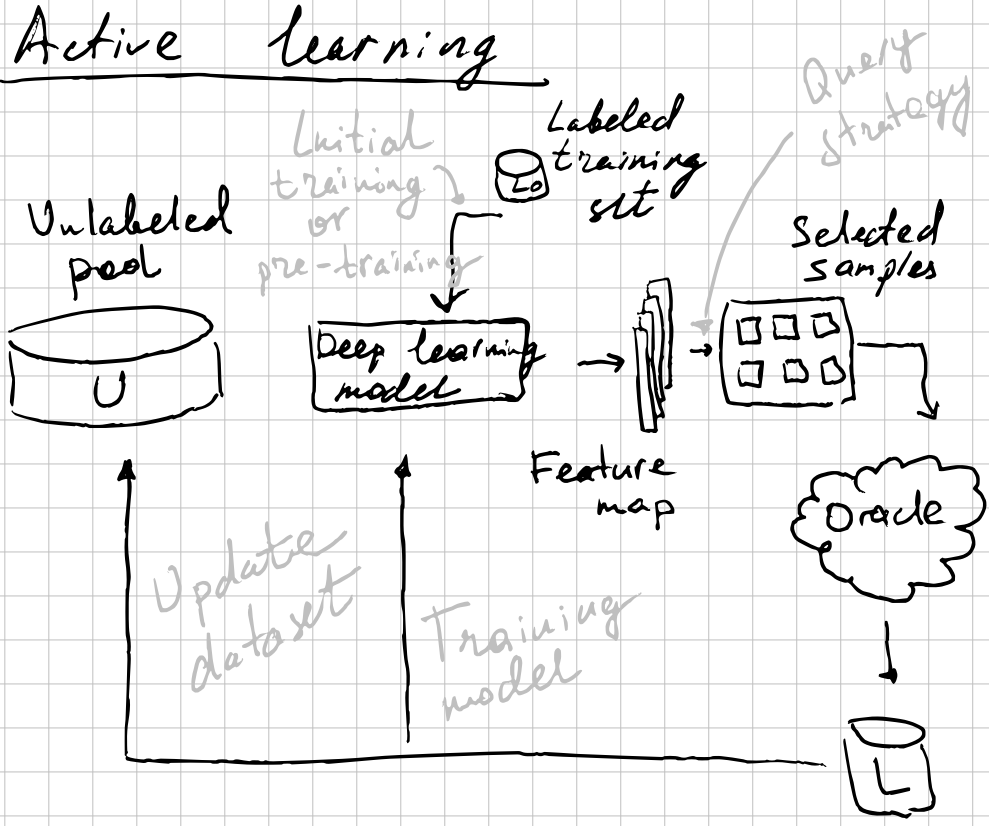


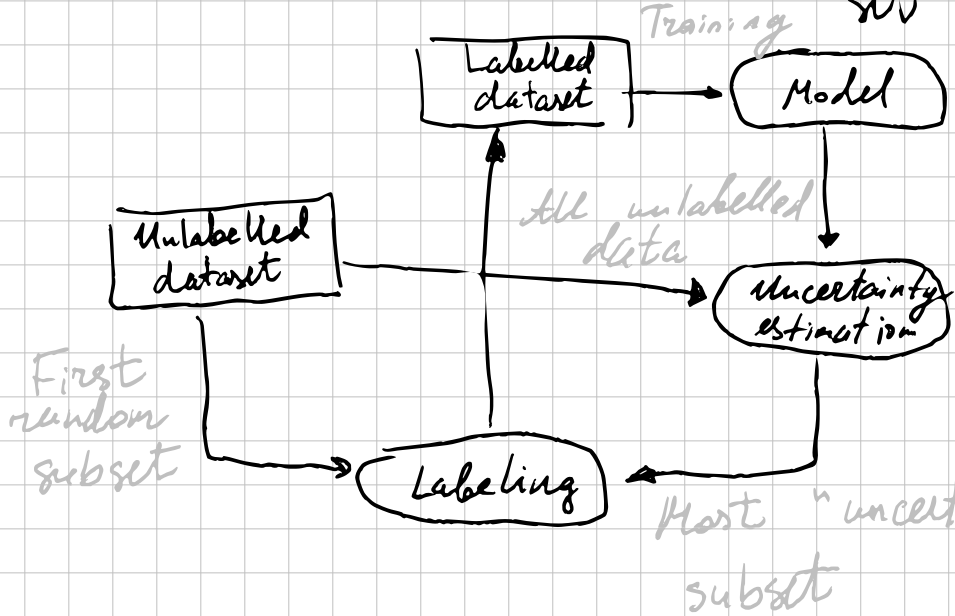
Specific dataset — Pretrained — Trainable — Specific Task

Usually, we can train model A efficiently on big generic dataset. We assume A posses some knowledge about the system.

We train only our parameters of the network (B).

Active learning





Pick examples that produces the most uncertain results, label them and return to model.

Estimation of dataset quality

1. Quality of a dataset

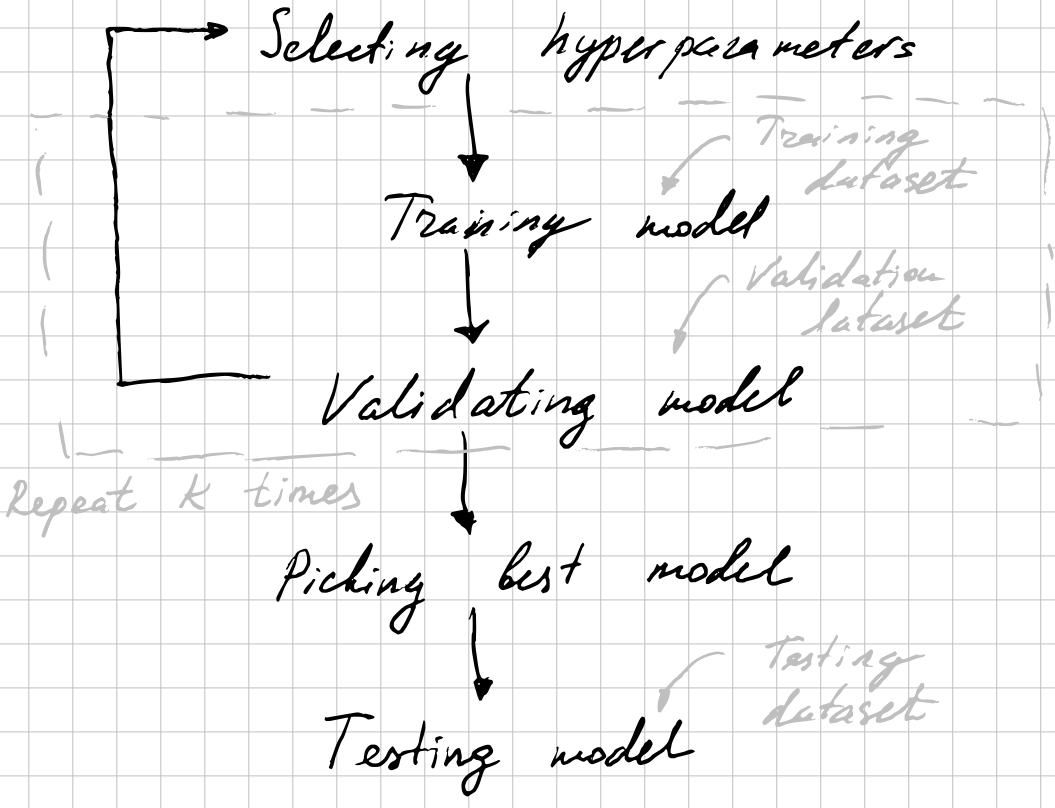
- Quantity: the more data we have - better
- Generalisation: how good data that we have represents general case

2. Quality of data

- Instrumental noise
- Exploratory data analysis: usually some general statistical error

3. Quality of labels

Model validation



- Test and validation sets should not have leaks (data from test set is in validation set)
- Test and validation sets should be close to production data; no augmentations.
- If multiple separate test datasets - calculate separate metrics.
- Separate data into cohorts can be helpful

Learning rate optimizations and scheduling

We have a model $f(\cdot)$
let $\hat{y}_i = f(x_i)$ denote prediction

We penalise our model using loss function $l(y_i, \hat{y}_i)$, where y_i is ground truth

Empirical loss function is defined as $L = \sum_i l(y_i, \hat{y}_i)$

We want to minimise empirical loss function.

Stochastic gradient descent

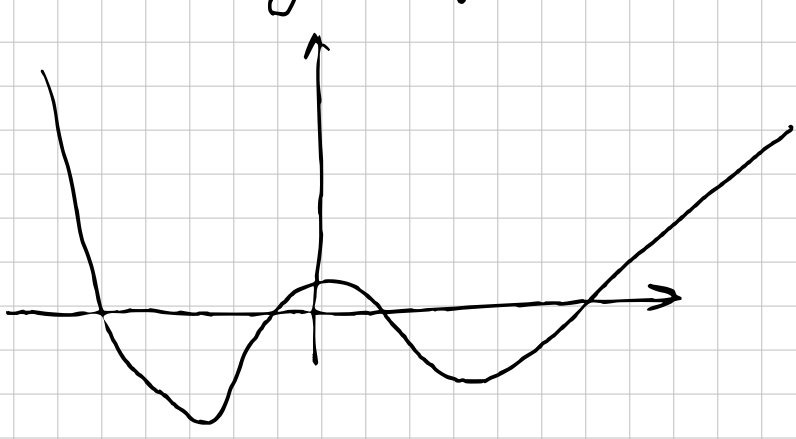
- Initialize parameters randomly
- Choose some learning rate λ;
- Choose randomly k elements from the dataset;
- Update

$$w_{k+1} = w_k - \lambda \frac{1}{k} \sum_{i=1}^{k} \nabla_w l(y_i, \hat{y}_i)$$

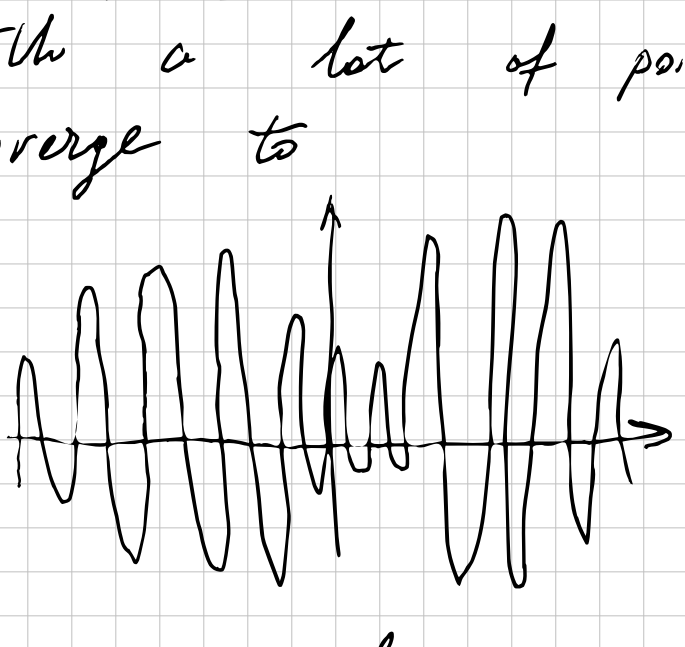
Update step:

$$a := a - \lambda \frac{d\, l(y_i, \hat{y}_i)}{da}$$

There are "good" functions with easy to find local mins

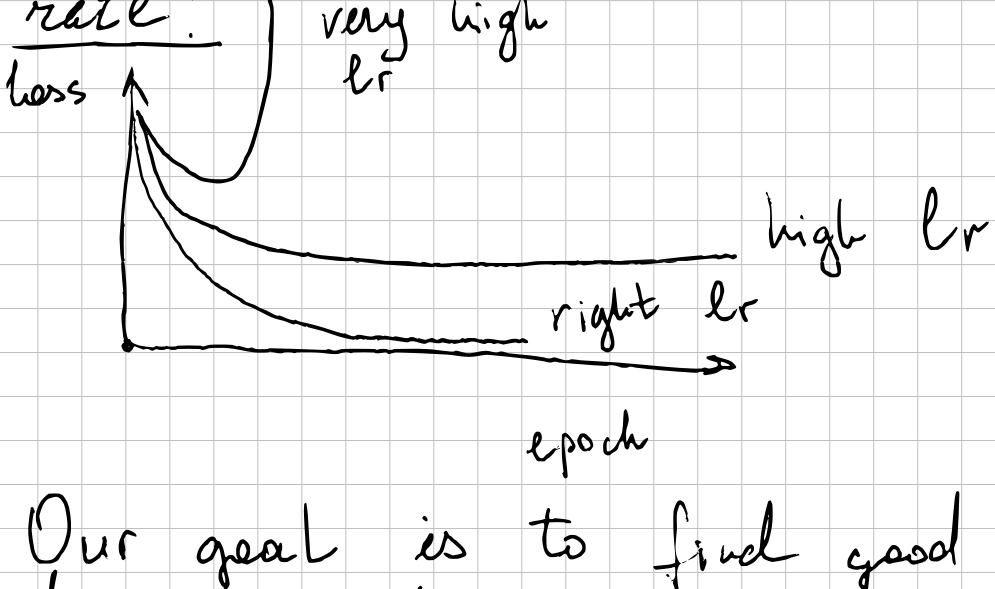


Or there are "bad" functions with a lot of points to converge to



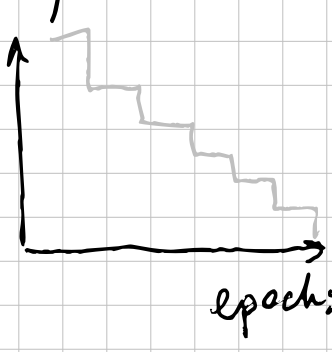
Learning rate

How to find learning rate?



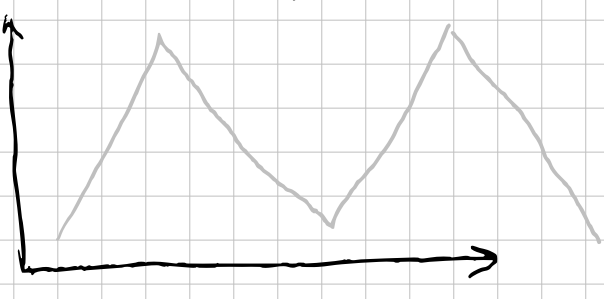
Our goal is to find good learning rate.

Step decay: decrease lr every k epochs



Cycle learning rate:

$$1 - \left| \frac{\text{iterations}}{\text{step-size}} - 2\,\text{cycle} + 1 \right|$$



Benefits of cycle LR:

- Helps to avoid sharp minima;
- Helps to avoid saddle points;

Schedulers help with that.

Data augmentation

Motivation:

- NNs are overparametrized;
- NNs can produce very "sharp" predictions, i.e. in a vicinity of point x in parameter space it could predict different classes.

To solve the problem: we can "enrich" our dataset by generating x' similar to x.

Pros:

- We will train our model on a dataset with more data samples;
- Hopefully the model will generalize better to unseen examples;

How to produce them:

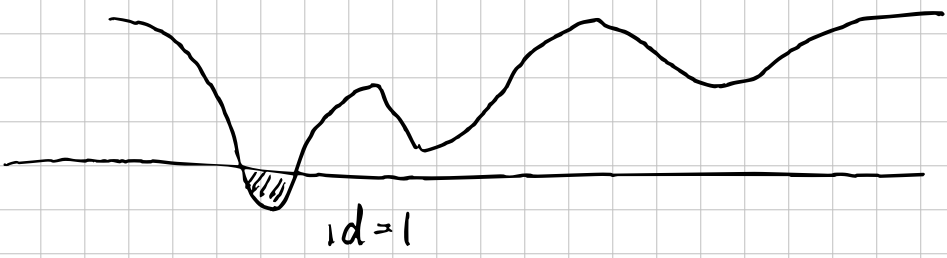
1. Train generative model $p(x|y)$ ← hard way
2. Data augmentations

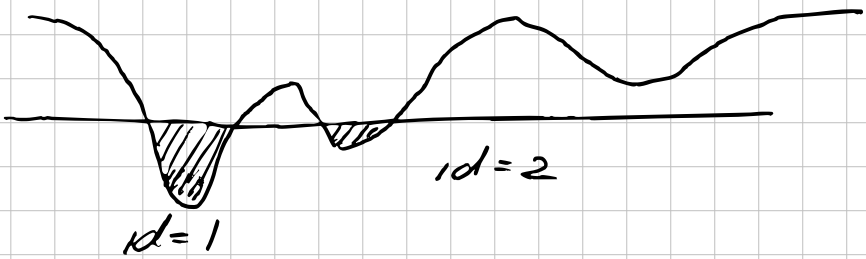
Augmentations:

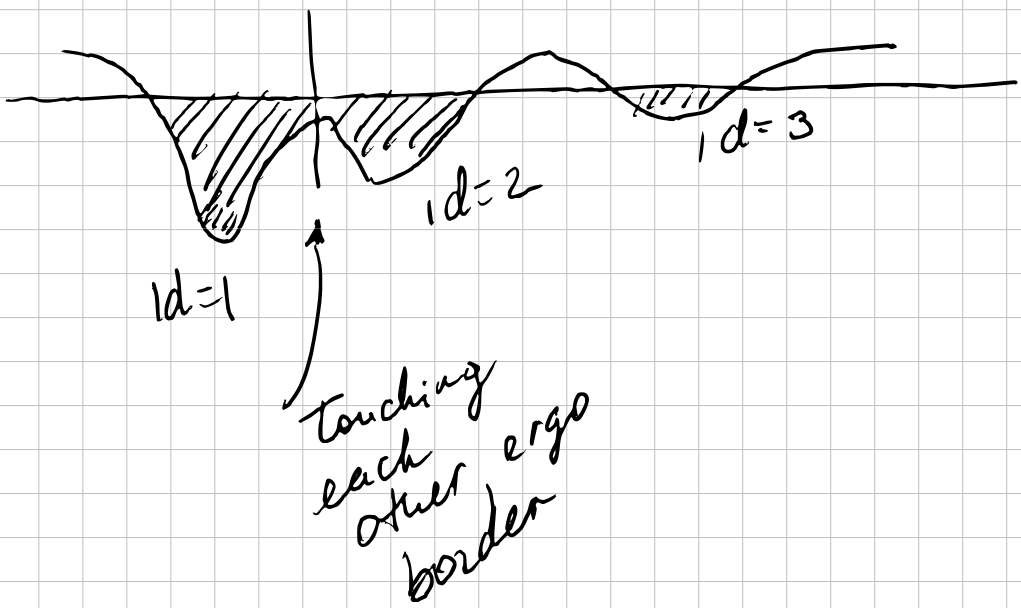
- Rotation
- Crop
- Coarse dropout
- Channel dropout
- Adding noise

Postprocessing

- Non-maximum suppression
- Watershed algorithm
  1. Represent 1-channel image as height map
  2. Pouring water onto it







- Erosion / Dilation

The curse of dimensionality

Imagine that we have a function $f(x)$ and we want to calculate its values in uniformly distributed points x_i

1d - 100 points
2d - 100^2 points
10d - 100^{10} points
etc

Hyperparameters optimisation

What if we want to optimise: learning rate, batch size, loss function, β in momentum.

First approach: Grid Search

We try every combination of hyperparameters.

- Bayesian optimization

① $x \in R$, $f(x) = const$.

$\hat{y} = f(x) \in R$

$l_i = l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2 = (y_i - c)^2$

$\mathcal{L} = \sum_i l_i = \sum_i (y_i - c)^2$

$\mathcal{L}_c' = \sum_i -2(y_i - c) = 0$

$\sum_i (y_i - c) = 0 \Leftrightarrow \sum y_i - \sum c = 0$

$\sum y_i - nc = 0 \Leftrightarrow nc = \sum y_i$

$c = \frac{\sum y_i}{n}$

② $\hat{y}_i = f(x) = ax_i + b$

$l_i = l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2 =$

$= (y_i - ax_i - b)^2 = (ax_i + b - y_i)^2$

$\mathcal{L} = \sum_i l_i = \sum_i (ax_i + b - y_i)^2$

$\mathcal{L}_a' = \sum_i 2(ax_i + b - y_i) \cdot x_i$

$\mathcal{L}_b' = \sum_i 2(ax_i + b - y_i)$

$\sum ax_i + b - y_i = 0$

$nb = \sum y_i - ax_i$

$b = \frac{(\sum y_i - ax_i)}{n}$

$\bar{y} = \frac{\sum y_i}{n}$, $\bar{x} = \frac{\sum x_i}{n}$ $\Big\} \; b = \bar{y} - a\bar{x}$

$\sum (ax_i + b - y_i) x_i = 0$

$a \sum x_i^2 = \sum x_i y_i - b \sum x_i$

$a = \frac{\sum x_i y_i - b \sum x_i}{\sum x_i^2}$

Ensembling

- In order to produce ensemble of models that performs better than individual models, model should be:
  - Accurate (better than random guessing)
  - Diverse (f_1 & f_2 are diverse if $\ell(f_1(x), y) \neq \ell(f_2(x), y)$ on new data points)

Example:

10 model

$p = 2/3$

- we use majority voting
- error is

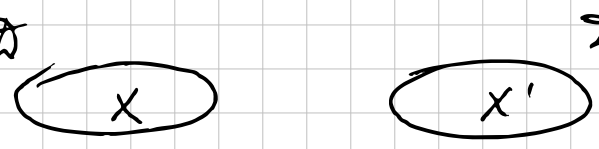
$$\sum_{i=6}^{10} \binom{10}{i} \frac{1}{3}^{i} \frac{2}{3}^{10-i} \approx 0.073$$

How to build ensembles:

- Bagging
- Cross-validation
- Stacking

Transfer learning

- Often we do not have enough data to train a neural network
- Pre-trained model was trained on a domain D
- Domains could either intersect or not:

1. 

2. 

If new targets are <u>not far away</u>

- Freeze a backbone, fine-tune classification head

New targets <u>are far</u>:

- Freeze some deep layers, and fine-tune top

Unsupervised learning

- Often we don't have an access to massive labeled data

Contrastive learning

- Split on positive and negative examples (that are far away)