Machine Learning
V07: ML System Design

System development: What to give priority?
Example: Learning to read checks end-to-end

With material from Andrew Y. Ng, Coursera
See also [LeCun et al, “Gradient-Based Learning…”, 1998]

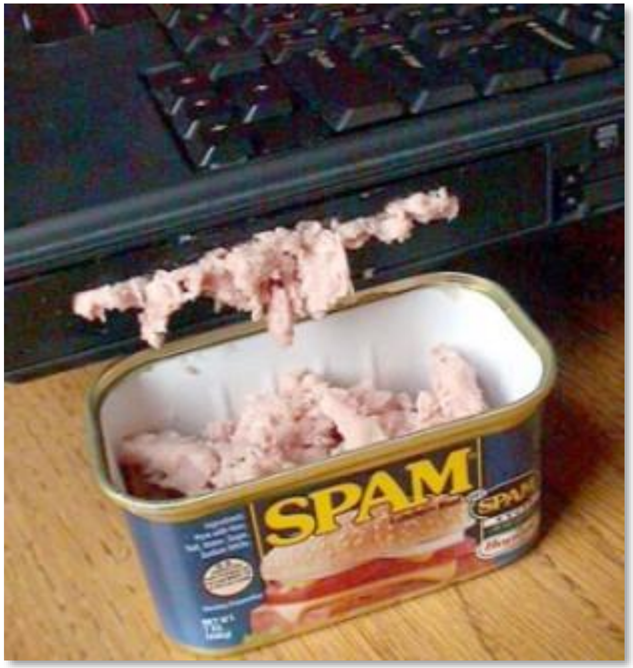
Zurich University of Applied Sciences and Arts
InIT Institute of Applied Information Technology (stdm)

Educational objectives

- **Remember** **error-** and **ceiling analysis** as well as the initial **24h hack** as tools to be successful in ML
- **Know how** to **design** and **prioritize** complete machine learning **system pipelines**
- **Appreciate** the **elegance** of the design **that enables end-to-end learning** for the check reading application of LeCun et al.

Zurich University
of Applied Sciences

1. SYSTEM DEVELOPMENT: WHAT TO GIVE PRIORITY?



Source: http://www.todayifoundout.com/index.php/2010/09/how-the-word-spam-came-to-mean-junk-message/

Example 1: Building a spam classifier

```
From: cheapsales@buystufffromme.com
To: stdm@zhaw.ch
Subject:  Buy now!

Deal of the week! Buy now!
Rolex w4tchs - $100
Med1cine (any kind) - $50
Also low cost M0rgages
available.
```

```
From: Renate Stadelmann
To: stdm@zhaw.ch
Subject:  Holiday plans

Hi Thilo,
was talking to Philipp about
plans for New Year. Sauna and
surfing in winter? ;-)
Love, Renate
```

Supervised learning task

- x: features of email ➔ y: 1 (spam) or 0 (non-spam)

Practical features

- **List of** 50'000 **most frequent words** in training set

$$x = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ \vdots \\ 1 \\ \vdots \end{bmatrix} \begin{matrix} \#Thilo \\ \#buy \\ \#deal \\ \#discount \\ \vdots \\ \#now \\ \vdots \end{matrix}, \quad x_j = \begin{cases} 1: word\ occurrs\ in\ mail \\ 0: otherwise \end{cases}$$

Example 1: Building a spam classifier (contd.)
How to prioritize *algorithmic* work?

How to best invest the time to make it work (i.e., have low error)?

- **Collect** lots of **data** (e.g., "honeypot" project)?
- **Develop** sophisticated **features**?
  - …based on email routing information from email **header**
  - …for message **body**
    ➔ Treat "discount" and "discounts" as same word? "Deal" and "Dealer"?
    ➔ Features about punctuation?
- **Develop** sophisticated **algorithm** to detect misspellings?
  ➔ e.g. "m0rtgage", "med1cine", "w4tches"

Advice
- Take **24h** to implement (rather: **hack**) a **complete system** including scoring
- Use diagnostics to decide where to improve
- In deep learning, follow **Andrej Karpathy's recipe** to stay sane (see appendix)

Recommendation

1. Start with a **simple algorithm** that can be **implement**ed **quickly**
   ➔ Implement it and test it on cross-validation data
2. Plot **learning curves** to diagnose if more data, more features, etc. are likely to help
3. **Error analysis**: Manually examine the CV examples that were misclassified
   ➔ Is there a **systematic trend** in what type of examples are **misclassified**?

Example 1: Building a spam classifier (contd.)

Error analysis

Assume the following experimental outcome

- $N_{CV} = 500$ emails in CV set
- 100 emails are misclassified

Example 1: Building a spam classifier (contd.)
Error analysis

Assume the following experimental outcome
- $N_{CV} = 500$ emails in CV set
- 100 emails are misclassified

➔ **Manually examine** the 100 **errors**

Categorization e.g. based on
1. **Type** of email
2. **Cues** (feature candidates) that would have helped the algorithm to classify correctly

Type	Number
Pharma	12
Replica / faked goods	4
Phishing	53
Other	31

Cues	Number
Deliberate misspellings (“m0rgage”, “med1cine”, etc.)	5
Unusual email routing	16
Unusual punctuation (“!!!!!!!” etc.)	32

quite rare

this might help

Example 1: Building a spam classifier (contd.)
The importance of numerical evaluation (error analysis 2)

Should a stemmer **be used** (e.g., free “Porter stemmer”)?
- Treats “discount” / ”discounts” / “discounted” / “discounting” as the same word
- Makes e.g. “universe” / “university” indistinguishable

➔ Error analysis doesn’t help much in deciding

Solution
- **Try** with & without
- **Compare numerical results** ➔ need a **single performance metric** for that (e.g. CV error; F-score)

Method	CV error
Original: without stemming	5%
With stemming	3%
Additional: distinguishing upper vs. lower case	3.2%

good idea!

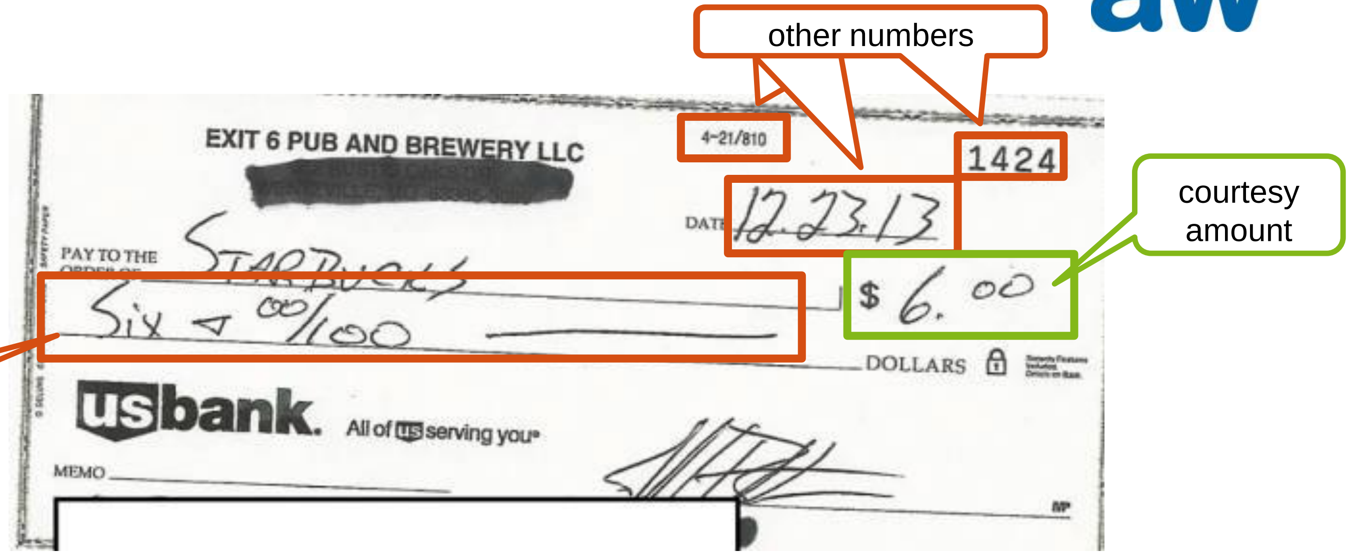
doesn’t help

- Attention: If classes are **skewed** (e.g., cancer prediction), **regard** recall-precision **trade-off**
  ➔ Use for example the F-measure instead of pure error (→ compare V03)
  ➔ Give the **rare class** the **label 1** (or: true)

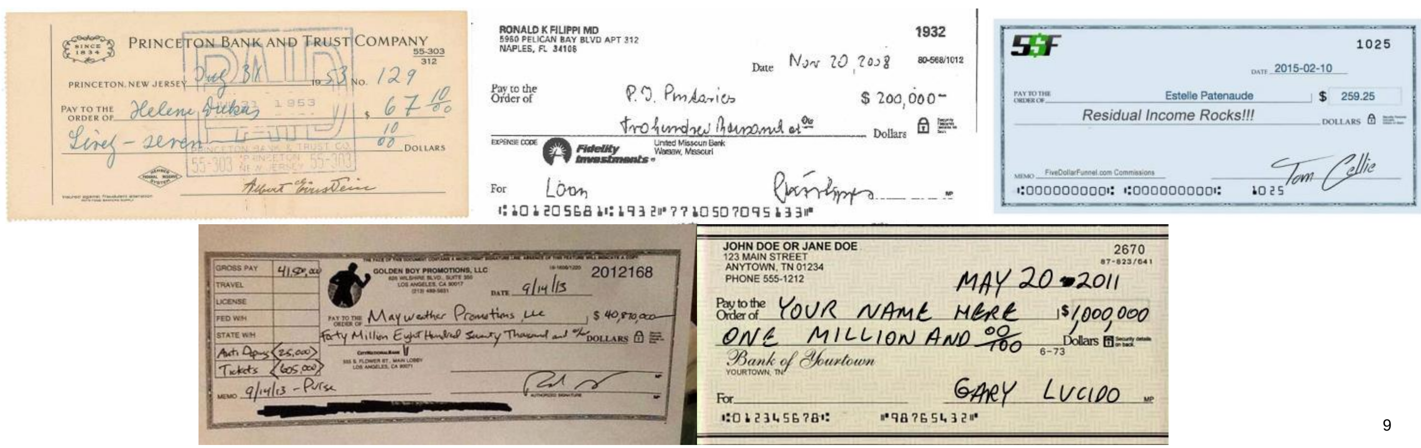
Example 2: Check reading application

Problem description:
- Read amount of $6
- Easy for humans, but time-consuming

➔ Automation wanted

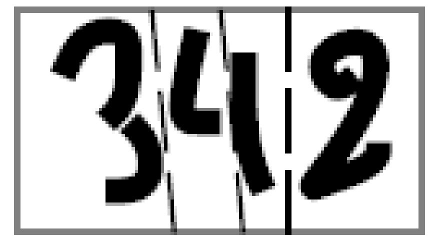


Challenge:
- Which number?
- Diversity



Example 2: Check reading application (contd.)
What part of the *pipeline* to improve next?

Challenge
- Identify correct character string (e.g., *«342»*) on a piece of paper
- Therefore:
  1. **Detect** all handwritten strings
  2. [**Identify** correct string (containing the amount)]
  3. Find correct **segmentation**
  4. **Recognize** individual characters



System pipeline
- **What part** of the pipeline should you **spend** the **most time** trying to improve**?**
- Note: Identification of the correct string is omitted here (could be placed at the end)

Image → Text detection → Character segmentation → Character recognition

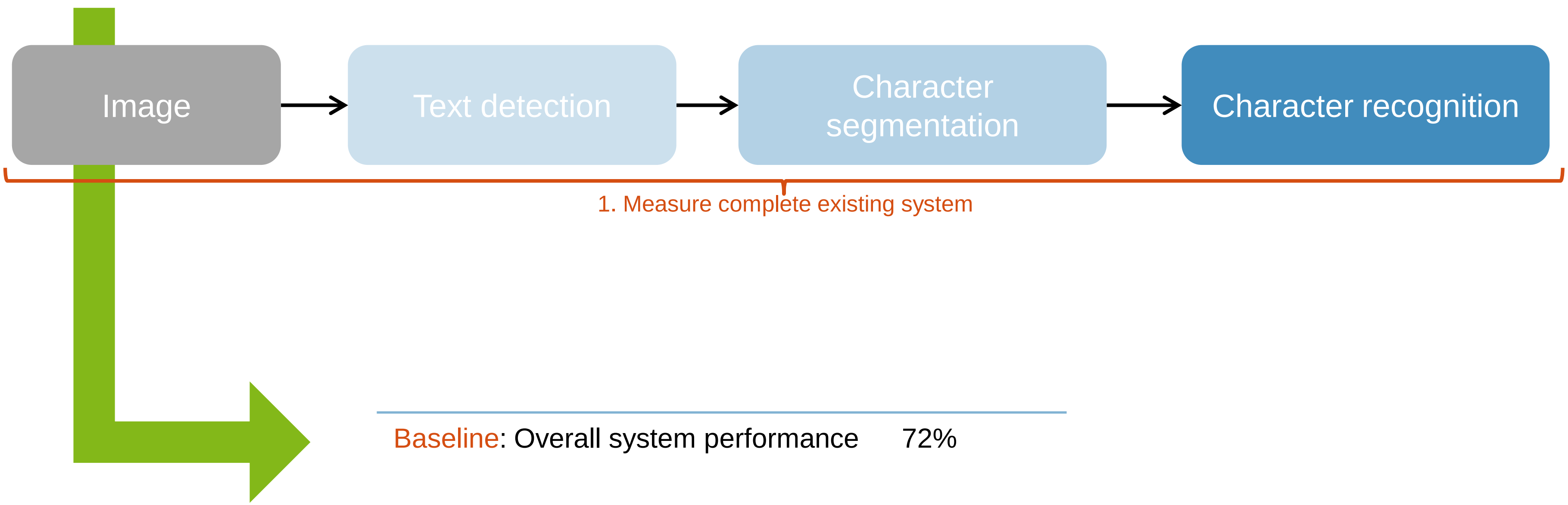
Zurich University
of Applied Sciences

Example 2: Check reading application (contd.)

Ceiling analysis: Attributing errors to individual components

Ceiling analysis

1. Baseline ➔ measure the (CV) performance of the complete pipeline

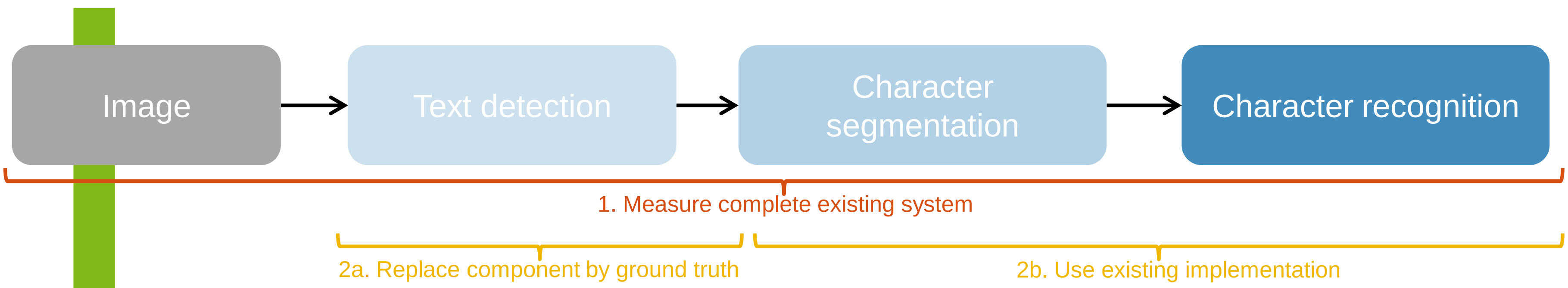


Example 2: Check reading application (contd.)

Ceiling analysis: Attributing errors to individual components

Ceiling analysis

1. Baseline ➔ measure the (CV) performance of the complete pipeline
2. Replace first component with ground truth (perfect results) ➔ measure performance



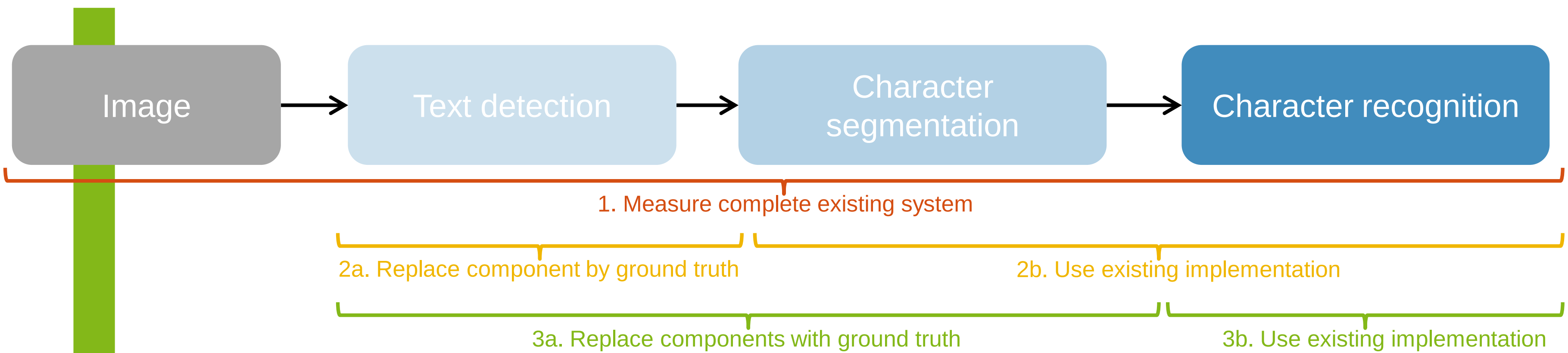
Baseline: Overall system performance	72%
Perfect components up to…	**Accuracy**
Text detection	89%

Example 2: Check reading application (contd.)

Ceiling analysis: Attributing errors to individual components

Ceiling analysis

1. Baseline ➔ measure the (CV) performance of the complete pipeline
2. Replace first component with ground truth (perfect results) ➔ measure performance
3. Replace next component with ground truth ➔ measure performance



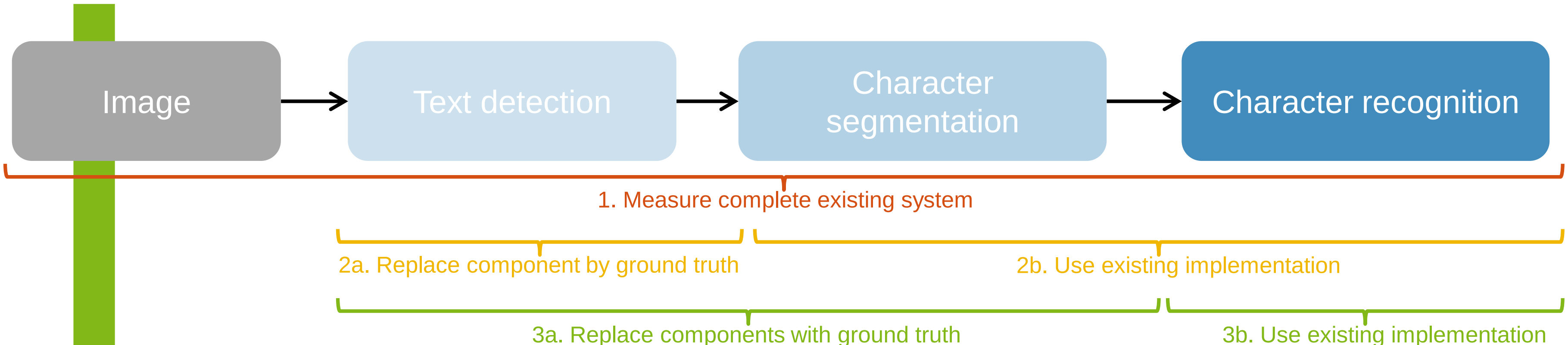
Baseline: Overall system performance	72%
Perfect components up to…	**Accuracy**
Text detection	89%
Character Segmentation	90%

Example 2: Check reading application (contd.)

Ceiling analysis: Attributing errors to individual components

Ceiling analysis

1. Baseline ➔ measure the (CV) performance of the complete pipeline
2. Replace first component with ground truth (perfect results) ➔ measure performance
3. Replace next component with ground truth ➔ measure performance
4. …

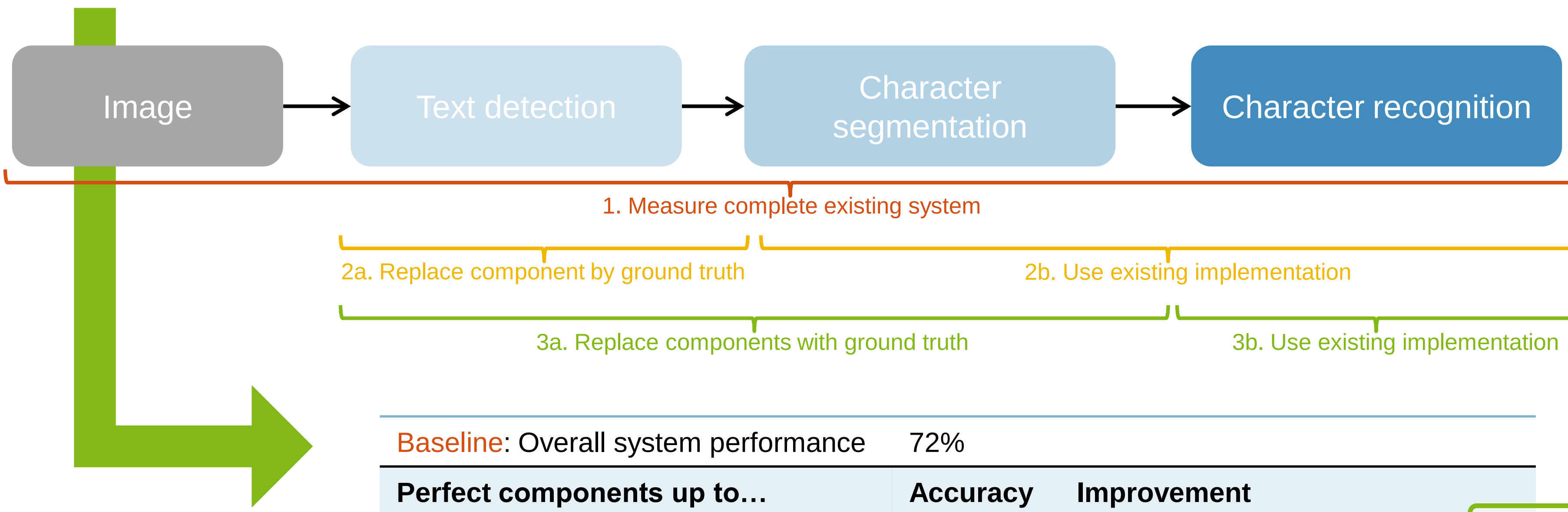


Baseline: Overall system performance	72%
Perfect components up to…	**Accuracy**
Text detection	89%
Character Segmentation	90%
Character Recognition	100%

Example 2: Check reading application (contd.)

Ceiling analysis: Attributing errors to individual components

Ceiling analysis
1. Baseline ➔ measure the (CV) performance of the complete pipeline
2. Replace first component with ground truth (perfect results) ➔ measure performance
3. Replace next component with ground truth ➔ measure performance
4. …



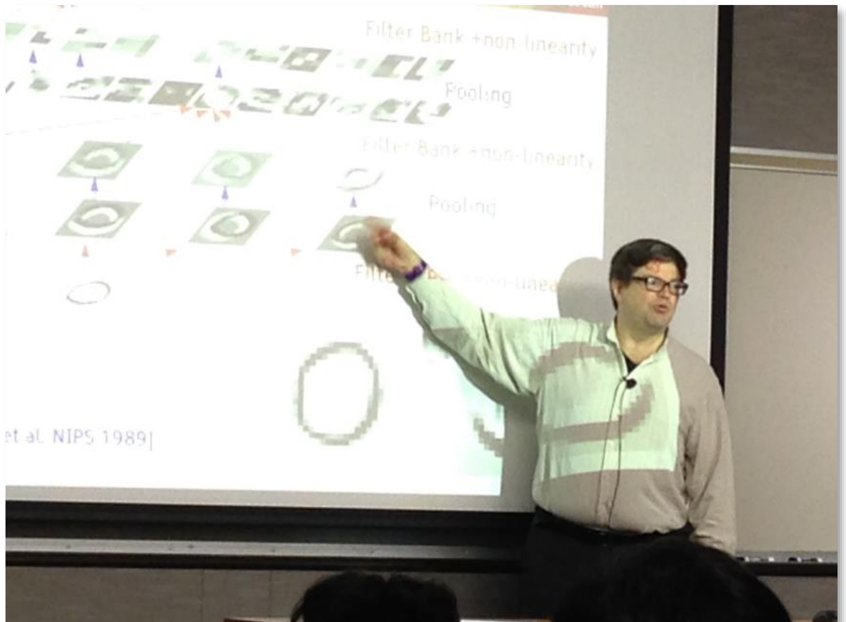
Baseline: Overall system performance	72%	
Perfect components up to…	**Accuracy**	**Improvement** [in percentage points over previous row]
Text detection	89%	17%
Character Segmentation	90%	1%
Character Recognition	100%	10%

good idea!

doesn't help

good idea!

2. SYSTEM EXAMPLE: LEARNING TO READ CHECKS END-TO-END



Source: https://en.wikipedia.org/wiki/Yann_LeCun#/media/File:Yann_LeCun_at_the_University_of_Minnesota.jpg

A landmark work in Machine *Learning*

LeCun et al., “Gradient-Based Learning Applied to Document Recognition”, 1998


PROC. OF THE IEEE, NOVEMBER 1998 1

Gradient-Based Learning Applied to Document Recognition -> LeNet 5

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

Abstract— **Multilayer Neural Networks trained with the backpropagation algorithm constitute the best example of a successful Gradient-Based Learning technique. Given an appropriate network architecture, Gradient-Based Learning algorithms can be used to synthesize a complex decision surface that can classify high-dimensional patterns such as handwritten characters, with minimal preprocessing. This paper reviews various methods applied to handwritten character recognition and compares them on a standard handwritten digit recognition task. Convolutional Neural Networks, that are specifically designed to deal with the variability of 2D shapes, are shown to outperform all other techniques.**

Real-life document recognition systems are composed of multiple modules including field extraction, segmentation, recognition, and language modeling. A new learning paradigm, called Graph Transformer Networks (GTN), allows such multi-module systems to be trained globally using Gradient-Based methods so as to minimize an overall performance measure.

Two systems for on-line handwriting recognition are described. Experiments demonstrate the advantage of global training, and the flexibility of Graph Transformer Networks.

A Graph Transformer Network for reading bank check is also described. It uses Convolutional Neural Network character recognizers combined with global training techniques to provides record accuracy on business and personal checks. It is deployed commercially and reads several million checks per day.

Keywords— **Neural Networks, OCR, Document Recognition, Machine Learning, Gradient-Based Learning, Convolutional Neural Networks, Graph Transformer Networks, Finite State Transducers.**

NOMENCLATURE

- GT Graph transformer.
- GTN Graph transformer network.
- HMM Hidden Markov model.
- HOS Heuristic oversegmentation.
- K-NN K-nearest neighbor.
- NN Neural network.
- OCR Optical character recognition.
- PCA Principal component analysis.
- RBF Radial basis function.
- RS-SVM Reduced-set support vector method.
- SDNN Space displacement neural network.
- SVM Support vector method.
- TDNN Time delay neural network.
- V-SVM Virtual support vector method.

The authors are with the Speech and Image Processing Services Research Laboratory, AT&T Labs-Research, 100 Schulz Drive Red Bank, NJ 07701. E-mail: {yann,leonb,yoshua,haffner}@research.att.com. Yoshua Bengio is also with the Département d'Informatique et de Recherche Opérationelle, Université de Montréal, C.P. 6128 Succ. Centre-Ville, 2920 Chemin de la Tour, Montréal, Québec, Canada H3C 3J7.

I. INTRODUCTION

Over the last several years, machine learning techniques, particularly when applied to neural networks, have played an increasingly important role in the design of pattern recognition systems. In fact, it could be argued that the availability of learning techniques has been a crucial factor in the recent success of pattern recognition applications such as continuous speech recognition and handwriting recognition.

The main message of this paper is that better pattern recognition systems can be built by relying more on automatic learning, and less on hand-designed heuristics. This is made possible by recent progress in machine learning and computer technology. Using character recognition as a case study, we show that hand-crafted feature extraction can be advantageously replaced by carefully designed learning machines that operate directly on pixel images. Using document understanding as a case study, we show that the traditional way of building recognition systems by manually integrating individually designed modules can be replaced by a unified and well-principled design paradigm, called *Graph Transformer Networks*, that allows training all the modules to optimize a global performance criterion.

Since the early days of pattern recognition it has been known that the variability and richness of natural data, be it speech, glyphs, or other types of patterns, make it almost impossible to build an accurate recognition system entirely by hand. Consequently, most pattern recognition systems are built using a combination of automatic learning techniques and hand-crafted algorithms. The usual method of recognizing individual patterns consists in dividing the system into two main modules shown in figure 1. The first module, called the feature extractor, transforms the input patterns so that they can be represented by low-dimensional vectors or short strings of symbols that (a) can be easily matched or compared, and (b) are relatively invariant with respect to transformations and distortions of the input patterns that do not change their nature. The feature extractor contains most of the prior knowledge and is rather specific to the task. It is also the focus of most of the design effort, because it is often entirely hand-crafted. The classifier, on the other hand, is often general-purpose and trainable. One of the main problems with this approach is that the recognition accuracy is largely determined by the ability of the designer to come up with an appropriate set of features. This turns out to be a daunting task which, unfortunately, must be redone for each new problem. A large amount of the pattern recognition literature is devoted to describing and comparing the relative

Outline

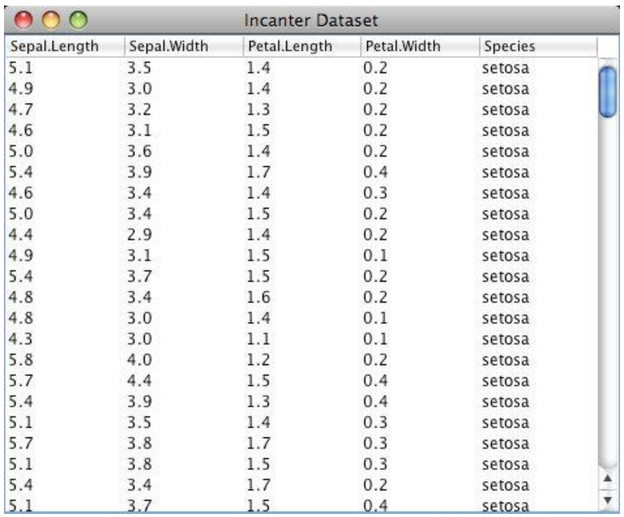
- Gradient-Based ML ✓
- Convolutional Neural Nets ✓ (→ DL module)
- Comparison with other Methods
- Multi-Module Systems & Graph Transformer Networks (GTNs)
- Multiple Object Recognition & Heuristic Oversegmentation
- Space Displacement Neural Networks
- GTN’s as General Transducers
- On-Line Handwriting Recognition System
- Check Reading System

➔ **GTNs** have **not been adopted widely**, but pioneered end-to-end deep learning

➔ Here explained in some completeness as an **historical example**

Standard and sequential supervised learning

Supervised Learning



Incanter Dataset

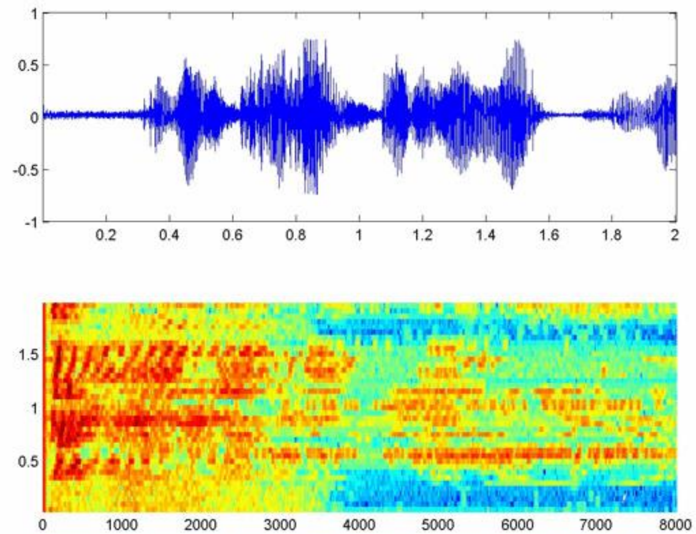
Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa
5.4	3.4	1.7	0.2	setosa
5.1	3.7	1.5	0.4	setosa

feature vectors labels

Typical assumptions on data:

- i.i.d.
- Surrounding tasks deemed simple(r)

Sequential Supervised Learning



Typical assumptions on data:

- Sequence information matters
- Overall task has many challenging components
  (e.g., segmentation → recognition → sequence assembly)

See M. Gori, «What's Wrong with Computer Vision?», ANNPR'18

Approaches to classifying sequential data

«A bird in the hand…» approach

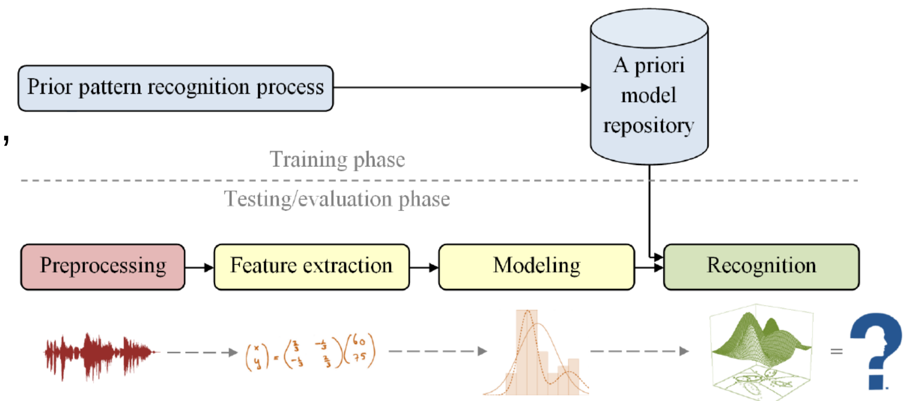
- Train standard classifier, extend it using a sliding window and post-processing (e.g., smoothing)

Direct modeling approach

- Train a generative (statistical) model of the sequence generation process (e.g., HMM)

«…two in the bush» approach

- Build a unified pattern recognition processing chain, optimize it globally with a unique criterion

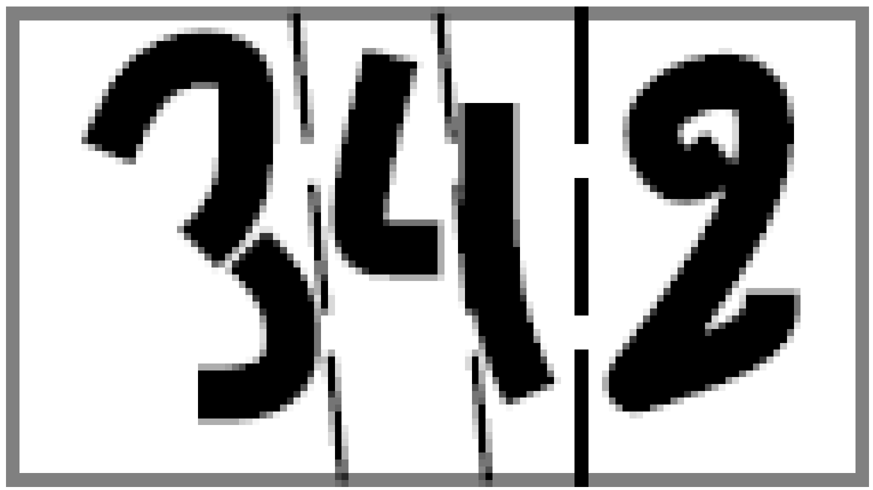


See also:

- T.G. Dietterich, «Machine Learning for Sequential Data – A Review», 2002
- J. Choi, «Deep Learning for Sequential Data», 2018 (online: http://sail.unist.ac.kr/talks/Deep_Learning_Winter_School_Time_Series.pdf)

Proposed Solution: Global Learning
Example: Reading handwritten strings

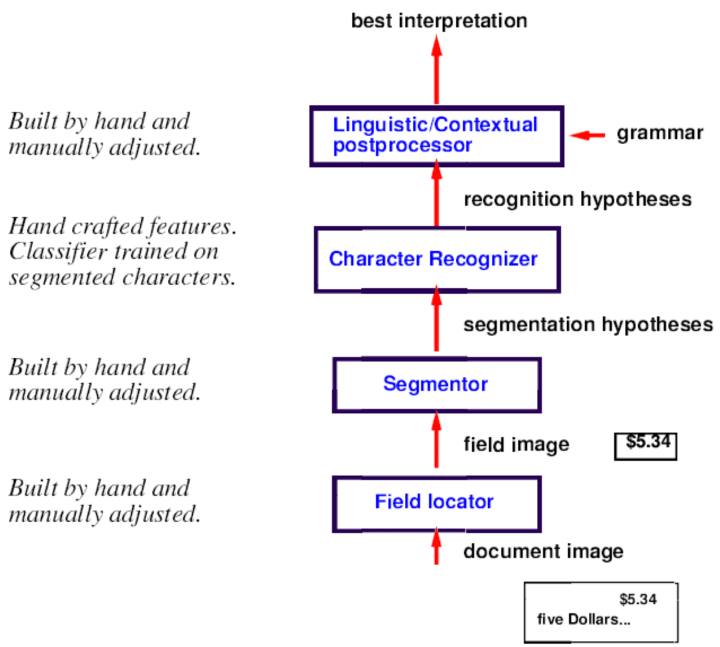
Challenge
- Identify correct character string («*342*») on a piece of paper
- Therefore: Find correct segmentation & recognize individual characters



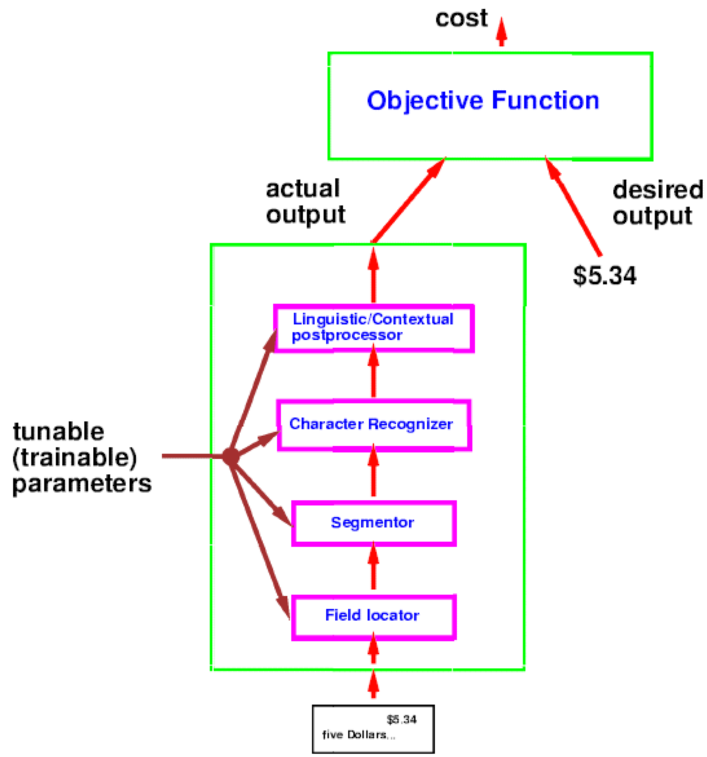
Images sources for this section: → see references slide in appendix

Global Learning
Learning end-to-end

What we know: Traditional pattern recognition system architecture



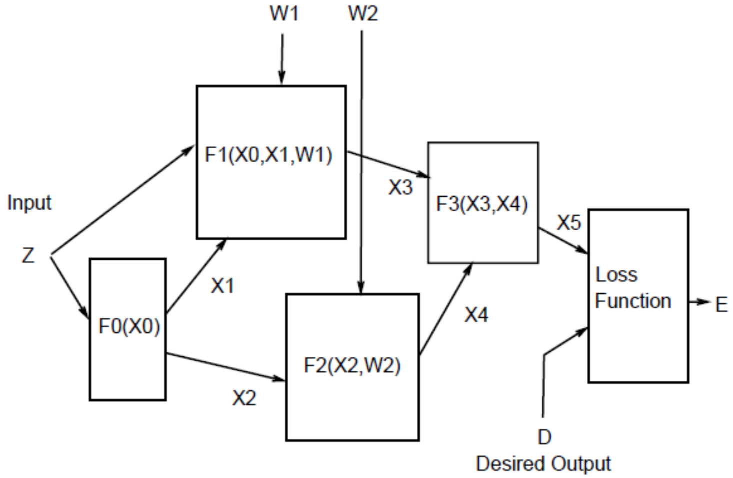
What we want: Train all parameters to optimize a global performance criterion



Foundation: Gradient-based learning

i.e., gradient descent

A trainable system composed of heterogeneous modules:



Backpropagation can be used if...

- cost (loss) function is differentiable w.r.t. parameters
- modules are differentiable w.r.t. parameters

➔ Gradient-based learning is the unifying concept behind many machine learning methods (→ see V02)

➔ Object-oriented design approach: Each module is a class with a `fprop()` and `bprop()` method

➔ Graph transformer network (GTN)

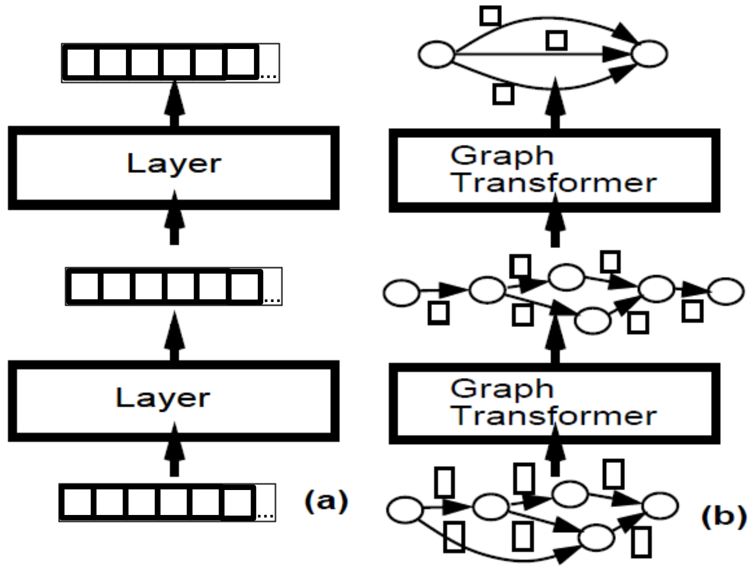
- General architecture to **train individual components collectively** via backpropagation
- …using **graph structures** as input and output

Graph Transformer Networks

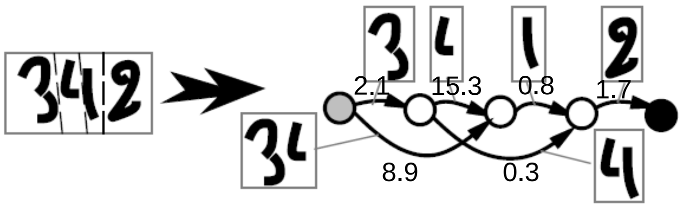
Network of pattern recognition modules that successively refine graph representations of the input

GTNs

- Operate on graphs of the input (b) instead of fixed-size feature vectors (a)



- Graph: DAG with numerical information ("penalties") at the arcs



➔ GTN takes gradients w.r.t. both module parameters and numerical data at input arcs

Example: Heuristic over-segmentation
…for reading handwritten strings

Loss function: Average penalty (over all training data) of best (i.e., lowest penalty) correct path (i.e. associated with correct sequence)

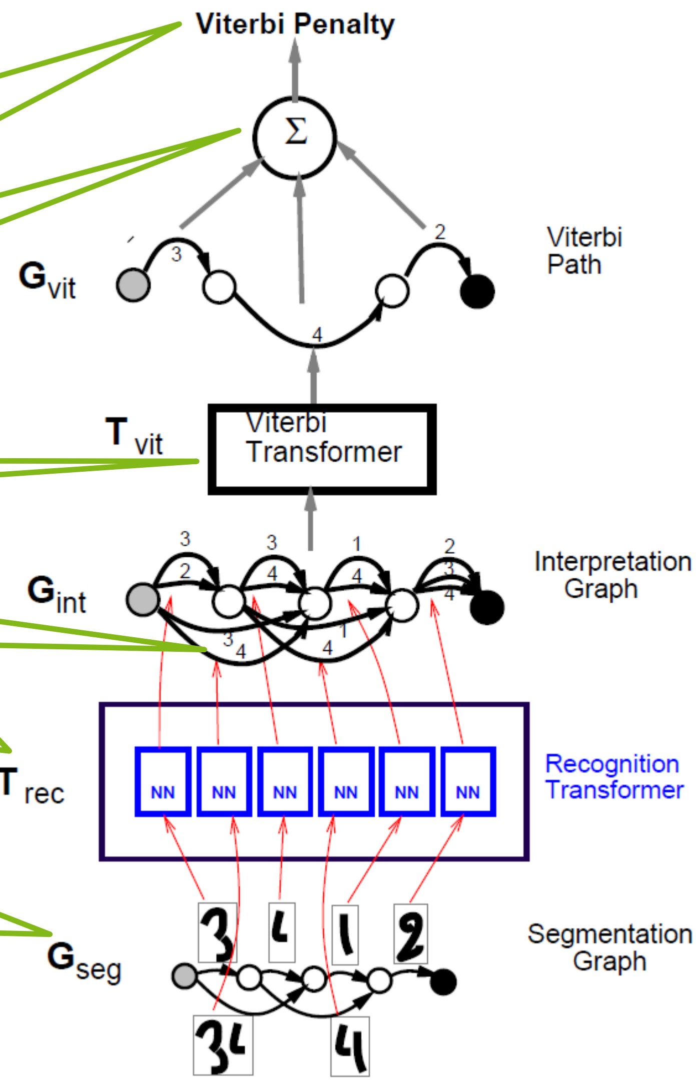
Sum up all penalties

Select path with least cum. penalty

Combined penalties
➔ handled via integrated training

Recognizes individual characters on single images (e.g. CNN)

Contains all possible segmentations (arcs: penalties & images)



Example: Heuristic over-segmentation

…for reading handwritten strings

Loss function: Average penalty (over all training data) of best (i.e., lowest penalty) correct path (i.e. associated with correct sequence)

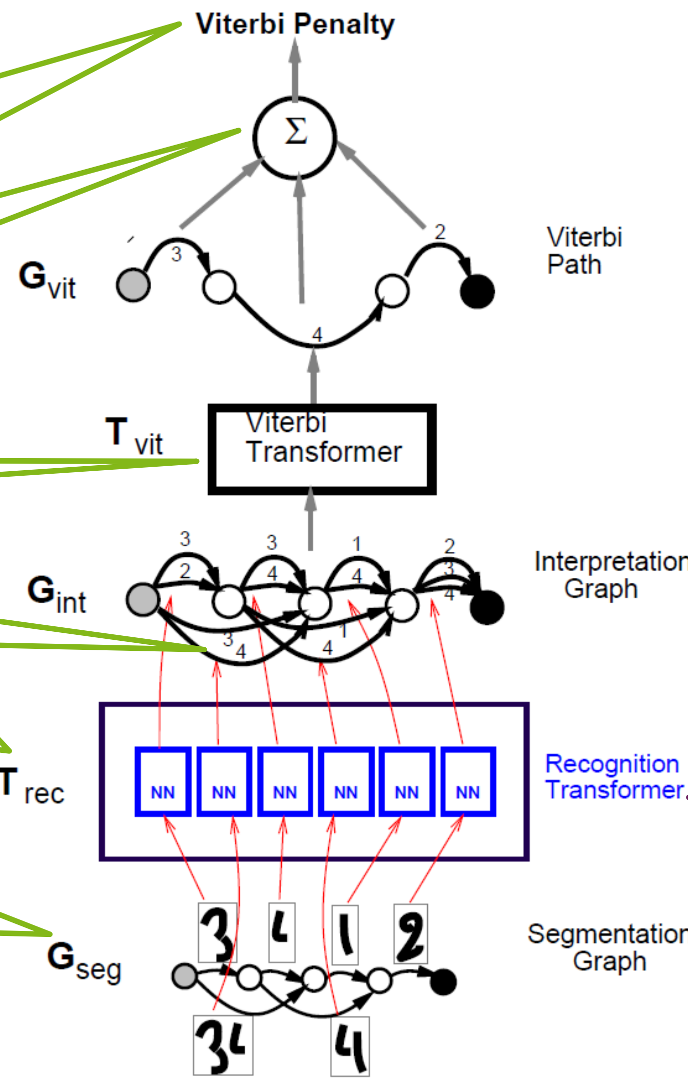
Sum up all penalties

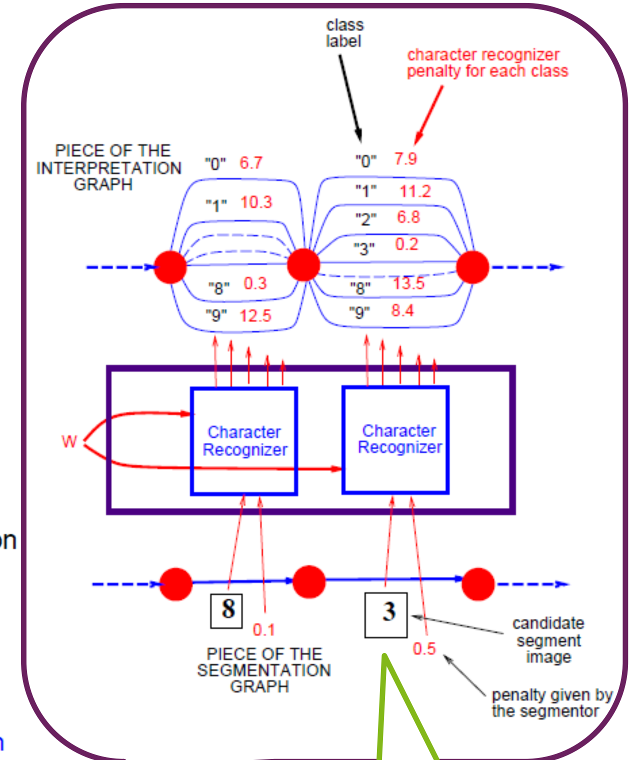
Select path with least cum. penalty

Combined penalties
➔ handled via integrated training

Recognizes individual characters on single images (e.g. CNN)

Contains all possible segmentations (arcs: penalties & images)

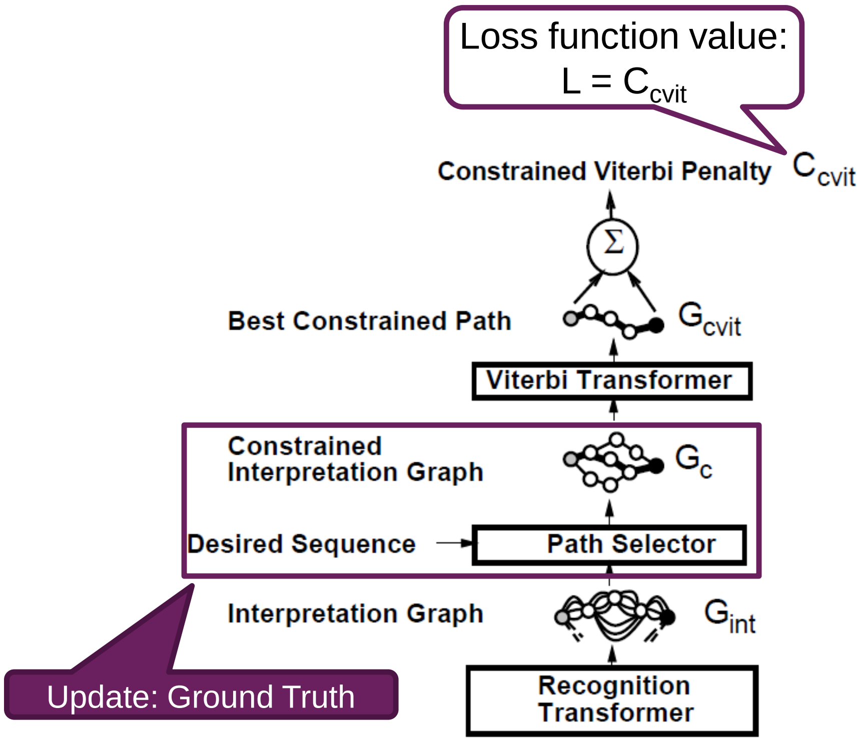




For each arc in G_{seg}: Create new one per (character) class, attach penalty and label

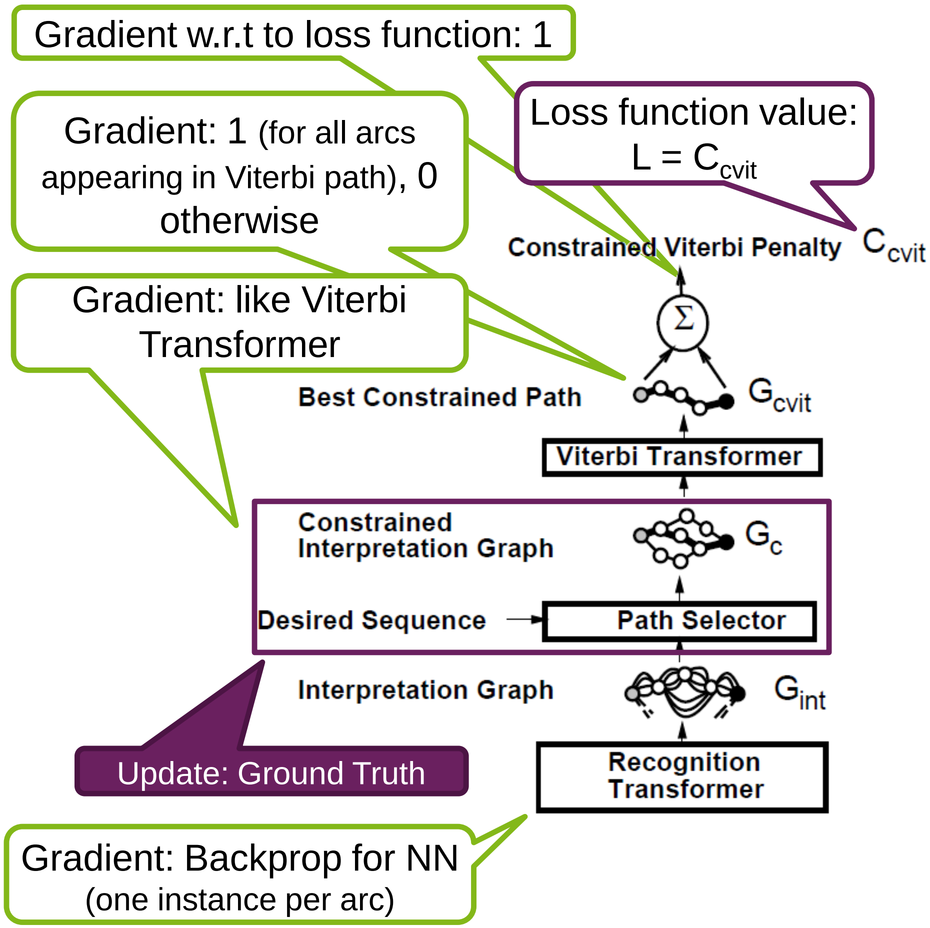
How to train? Discriminative training wins

«Viterbi» training



How to train? Discriminative training wins

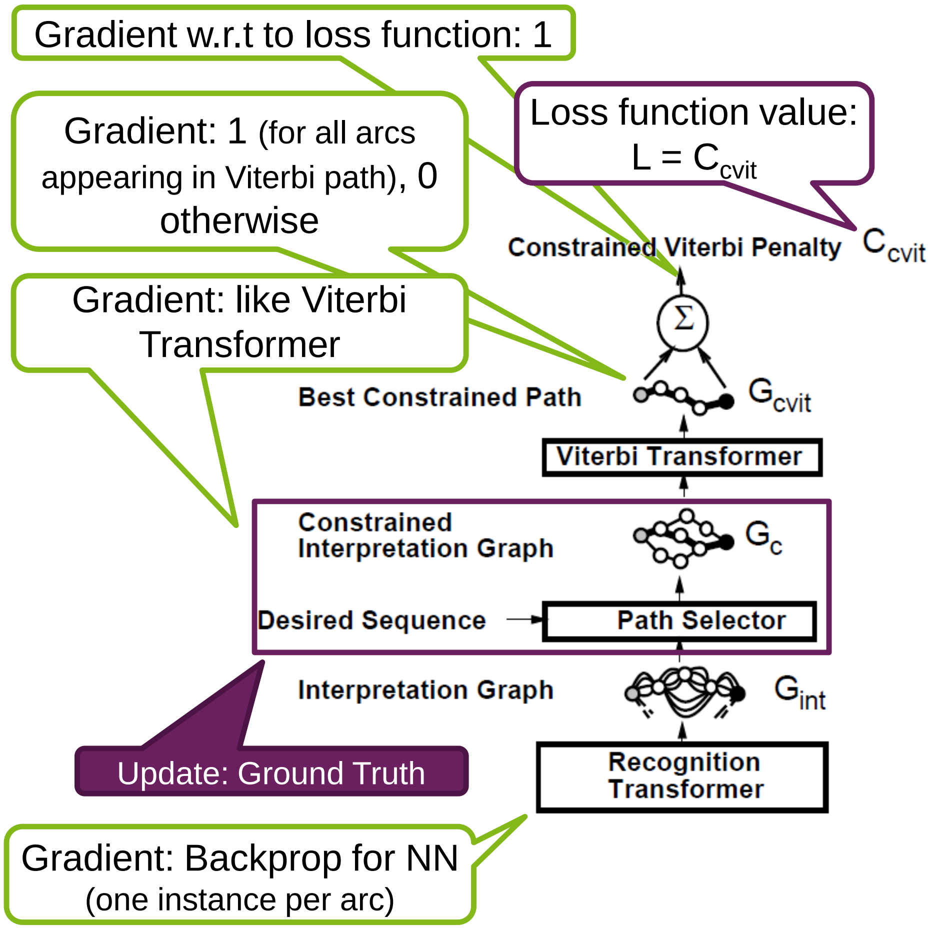
«Viterbi» training



How to train? Discriminative training wins

Problems:
1. Trivial solution possible (Recognizer ignores input & sets all outputs to small values)
2. Penalty does not take competing answers into account (i.e., ignores training signals)

«Viterbi» training



How to train? Discriminative training wins

Problems:
1. Trivial solution possible (Recognizer ignores input & sets all outputs to small values)
2. Penalty does not take competing answers into account (i.e., ignores training signals)

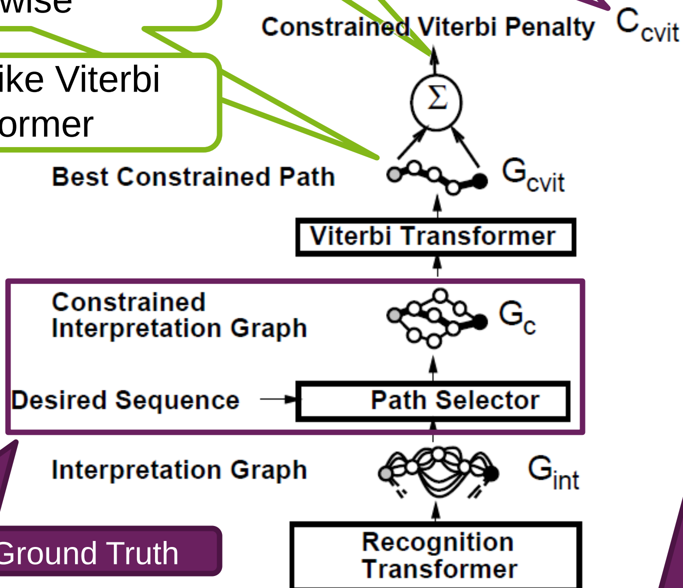
«Viterbi» training

Gradient w.r.t to loss function: 1

Gradient: 1 (for all arcs appearing in Viterbi path), 0 otherwise

Loss function value: $L = C_{cvit}$

Gradient: like Viterbi Transformer

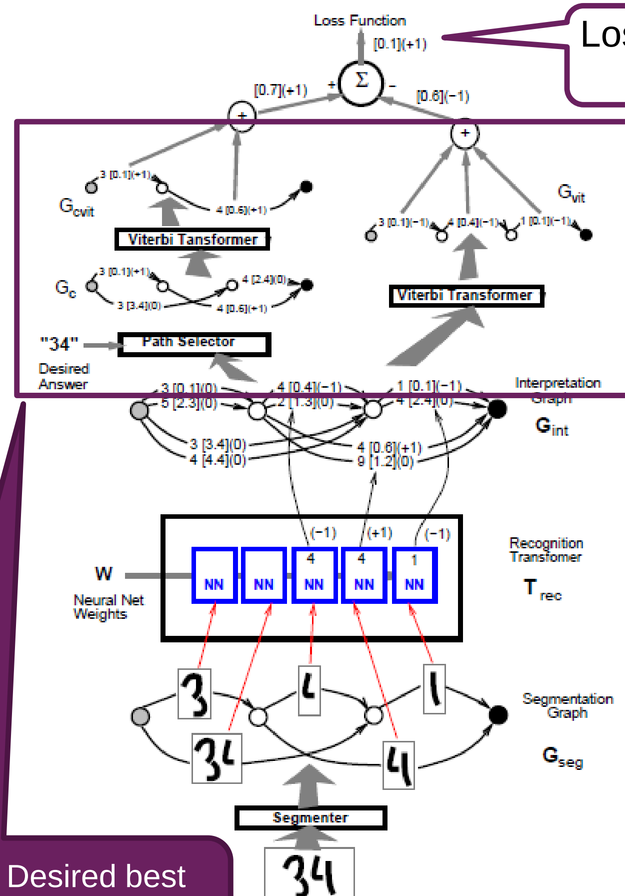


Update: Ground Truth

Gradient: Backprop for NN (one instance per arc)

Discriminative training

Loss function value: $L = C_{cvit} - C_{vit}$



Update: Desired best output (left) vs. best output (correct or not; right)

Solved: Discriminative training builds the class-*"separating surfaces rather than modeling individual classes independently of each other"*
➔ L=0 if best path is a correct path.

How to train? Discriminative training wins

Problems:
1. Trivial solution possible (Recognizer ignores input & sets all outputs to small values)
2. Penalty does not take competing answers into account (i.e., ignores training signals)

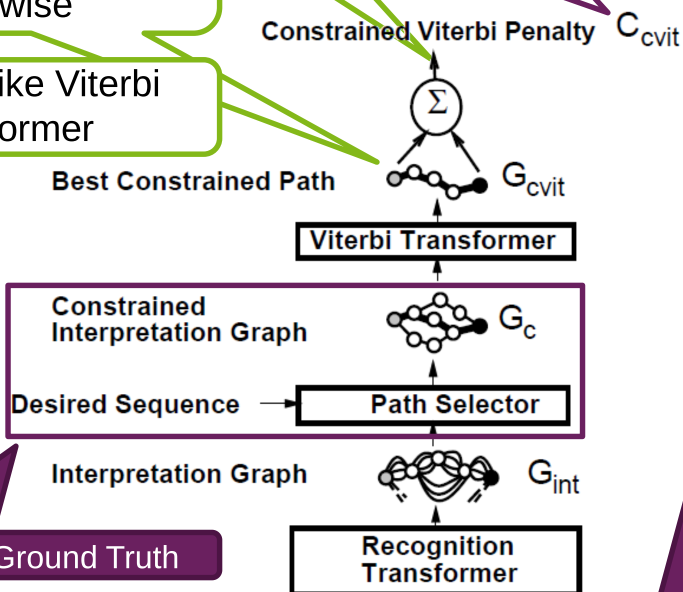
«Viterbi» training

Gradient w.r.t to loss function: 1

Gradient: 1 (for all arcs appearing in Viterbi path), 0 otherwise

Gradient: like Viterbi Transformer

Loss function value: $L = C_{cvit}$

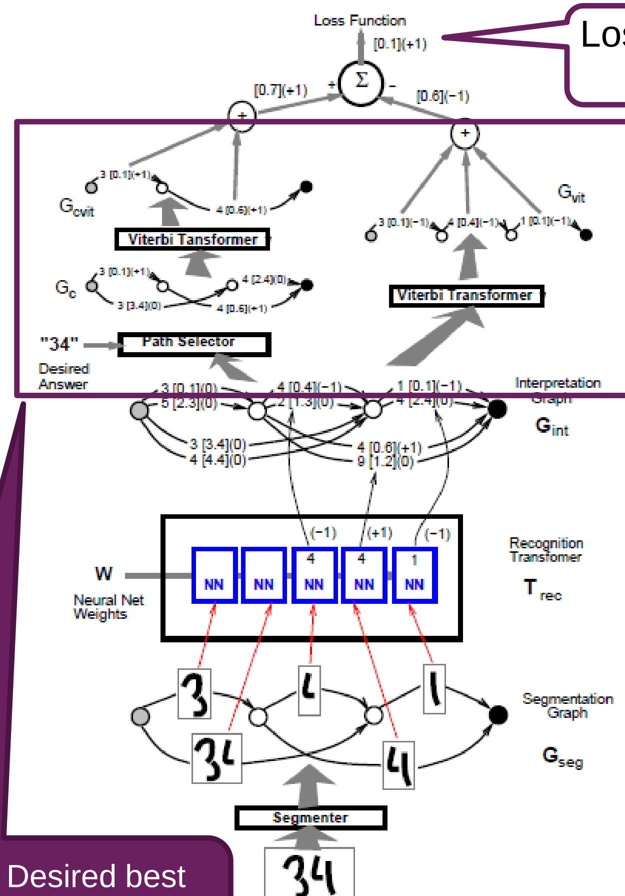


Update: Ground Truth

Gradient: Backprop for NN (one instance per arc)

Discriminative training

Loss function value: $L = C_{cvit} - C_{vit}$



Update: Desired best output (left) vs. best output (correct or not; right)

Solved: Discriminative training builds the class-*"separating surfaces rather than modeling individual classes independently of each other"*
➔ L=0 if best path is a correct path.

Zurich University of Applied Sciences

How to train? Discriminative training wins

Problems:
1. Trivial solution possible (Recognizer ignores input & sets all outputs to small values)
2. Penalty does not take competing answers into account (i.e., ignores training signals)

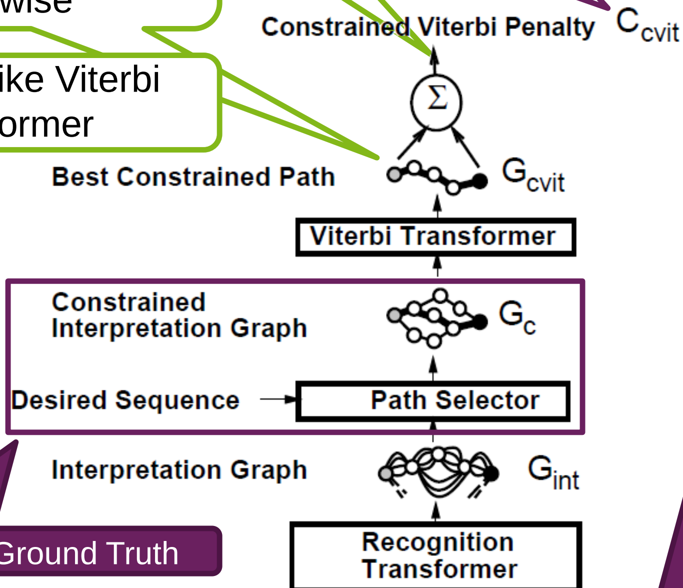
«Viterbi» training

Gradient w.r.t to loss function: 1

Gradient: 1 (for all arcs appearing in Viterbi path), 0 otherwise

Loss function value: $L = C_{cvit}$

Gradient: like Viterbi Transformer

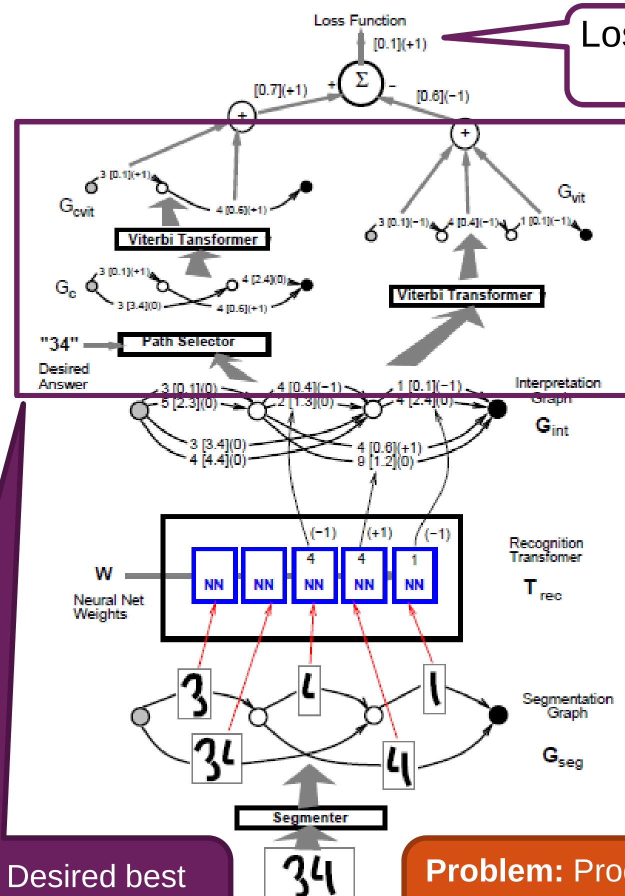


Update: Ground Truth

Gradient: Backprop for NN (one instance per arc)

Discriminative training

Loss function value: $L = C_{cvit} - C_{vit}$



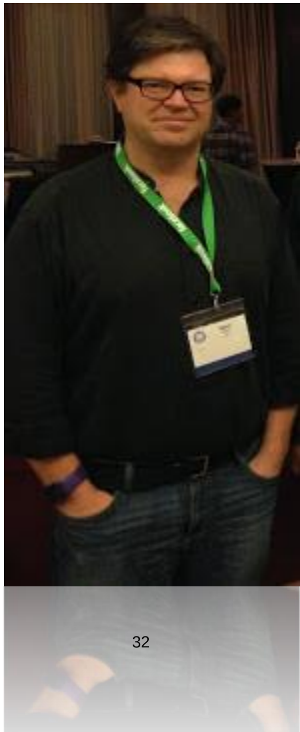
Update: Desired best output (left) vs. best output (correct or not; right)

Problem: Produces no margin as long as classification is correct (see paper for solution)

Remarks

Discriminative training
- Uses **all** available training **signals**
- Utilizes "penalties", **not probabilities**
  - ➔ **No** need for **normalization**
  - ➔ Enforcing normalization is *"complex, inefficient, time consuming, ill-conditions the loss function"* [according to paper]
- Is the ***easiest/direct* way** to achieve the objective of classification (**as opposed to Generative training**, that solves the more complex density estimation task as an intermediary result)

List of possible GT modules
- All **building blocks of (C)NNs** (layers, nonlinearities etc.)
- **Multiplexer** (though not differentiable w.r.t. to switching input)
  → can be used to dynamically rewire GTN architecture per input
- **min**-function (though not differentiable everywhere)
- **Loss** function



Conclusions?

possible
without

note-by-note
segmentation?

Zurich University
of Applied Sciences

Less need for manual labeling

- **Ground truth only** needed **for final** result
  (not for every intermediate result like e.g. segmentation)

Early errors can be adjusted later due to…

- …**unified training of all** pattern recognition **modules** under one regime
- …**postponing hard decisions** until the very end

No call upon probability theory for modeling / justification

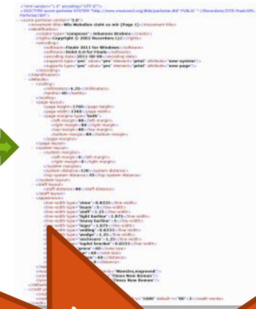
- Occam’s razor: Choose easier discriminative model over generative one
  Vapnik: Don’t solve a more complex problem than necessary
- No need for normalization when dealing with “penalties”
  instead of probabilities → no “other class” examples needed
- **Less constrains on system architecture** and module selection

possible without crop marks?

Zurich University
of Applied Sciences

Conclusions?

possible
without
note-by-note
segmentation?



No – in both projects*, stronger (application-based) inductive biases served learning better

Less need for manual labeling

- **Ground truth only** needed **for final** result
  (not for every intermediate result like e.g. segmentation)

Early errors can be adjusted later due to…

- …**unified training of all** pattern recognition **modules** under one regime
- …**postponing hard decisions** until the very end

No call upon probability theory for modeling / justification

- Occam's razor: Choose easier discriminative model over generative one
  Vapnik: Don't solve a more complex problem than necessary
- No need for normalization when dealing with "penalties"
  instead of probabilities → no "other class" examples needed
- **Less constrains on system architecture** and module selection

possible without crop marks?

***)** Meier, Stadelmann, Stampfli, Arnold & Cieliebak (2017). *«Fully Convolutional Neural Networks for Newspaper Article Segmentation»*. ICDAR'2017.
Tuggener, Elezi, Schmidhuber & Stadelmann (2018). *«Deep Watershed Detector for Music Object Recognition»*. ISMIR'2018.
Stadelmann et al. (2018). *«Deep Learning in the Wild»*. ANNPR'2018.

Review

- ML systems are **pipelines composed of** individual **components** that can be **develope**d **collaboratively** in a team
- Do **ceiling analysis** to **decide which component** in the pipeline is most likely to alter the result for good
- Do **qualitative analysis of wrong**ly predicted **examples** to get **insight** what is going wrong
- Do **numerical error analysis** (i.e., compare CV scores) to **prioritize algorithmic ideas**
- Have a **single performance metric** (e.g., error or F-measure) to **monitor** the **evolution** of your system **continuously**
- Consider **end-to-end training** (global optimization via deep learning)

System design, expanded
Reading & discussion task

GTNs are an historical example of end-to-end training; nowadays, deep learning is frequently used in this respect, and researchers strive to make more and more general functions learnable. Read the article *«Reinforcement Learning, Fast and Slow»* by Botvinick et al. (Trends in Cog. Sci., Vol. 23, No. 5, 2019*****) and get an overview.

Then discuss at your table:
- What is reinforcement learning (RL)
- What is fast and slow learning in ML? In biological learning?
- How can current trends in RL help ML systems spread the range of their applicability?
- Is there a connection to fast and slow *thinking* (Kahneman, *«Thinking, fast and slow»*, 2011******)?

***)** Online: https://www.cell.com/trends/cognitive-sciences/fulltext/S1364-6613(19)30061-0)
****)** Online: https://en.wikipedia.org/wiki/Thinking,_Fast_and_Slow

APPENDIX

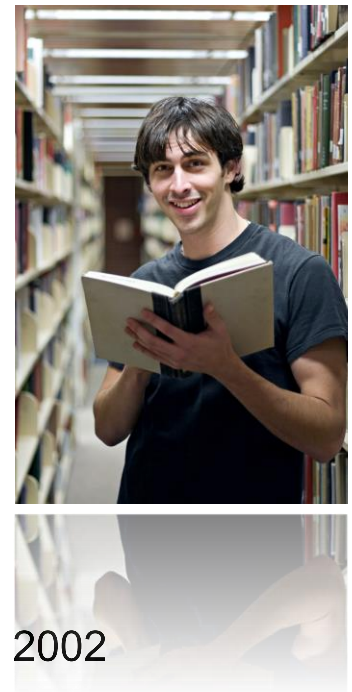
Karpathy's recipe for neural network training
See http://karpathy.github.io/2019/04/25/recipe/

1. Become one with the data
2. Set up the end-to-end training/evaluation skeleton + get dumb baselines
   a) fix random seed
   b) simplify (no augmentation, no fanciness, …)
   c) evaluate on full test set to add significance
   d) verify loss @ init
   e) initialize well
   f) human baseline
   g) input-independent baseline
   h) overfit one batch
   i) verify decreasing training loss
   j) visualize just before the net
   k) visualize prediction dynamics on fixed test batch
   l) use backprop to chart dependencies
   m) generalize a special case
3. Overfit
   a) picking the model (don't be a hero)
   b) adam is safe
   c) complexify only one at a time
   d) do not trust learning rate decay defaults
4. Regularize
   a) get more data
   b) data augment
   c) creative augmentation
   d) pretrain
   e) stick with supervised learning
   f) smaller input dimensionality
   g) smaller model size
   h) decrease the batch size
   i) use dropout (2D for CNNs; careful with batchnorm)
   j) increase weight decay
   k) early stopping to catch best model before overfitting
   l) try a larger (early stopped) model

5. Tune
   a) random over grid search
   b) hyper-parameter optimization
6. Squeeze out the juice
   a) Ensembles (tops out after ~5 models)
   b) leave it training

For RL-specific advice and a general research methodology, see:
- http://amid.fish/reproducing-deep-rl
- https://stdm.github.io/Great-methodology-delivers-great-theses/

Further reading for end-to-end learning



- Original short paper: Bottou, Bengio & LeCun, “Global Training of Document Processing Systems using Graph Transformer Networks”, 1997
  http://www.iro.umontreal.ca/~lisa/pointeurs/bottou-lecun-bengio-97.pdf

- Landmark long paper: LeCun, Bottou, Bengio & Haffner, “Gradient-Based Learning Applied to Document Recognition”, 1998
  http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

- Slide set by the original authors: Bottou, “Graph Transformer Networks”, 2001
  http://leon.bottou.org/talks/gtn

- Overview: Dietterich, “Machine Learning for Sequential Data: A Review”, 2002
  http://eecs.oregonstate.edu/~tgd/publications/mlsd-ssspr.pdf

- Recent work: Collobert, “Deep Learning for Efficient Discriminative Parsing”, 2011
  http://ronan.collobert.com/pub/matos/2011_parsing_aistats.pdf