Natural Language and Speech Processing
Lecture 12: Tips and Tricks

Tanel Alumäe

Contents

- Tips about data
- Tips about models
- Tips about training
- Debugging neural networks
- Other practical tips

Partly based on Andrej Karpathy's (Head of AI at Tesla) recent blog post
http://karpathy.github.io/2019/04/25/recipe/ (must read)

Data

- Become one with data
  - Inspect your data carefully (spend many hours)
  - Understand the distribution and look at the patterns
    - Your brain is very good at it
    - Are there duplicates?
    - Are there corrupt examples?
    - Are the labels inbalanced?
  - Think how your brain solves this task
    - Can you solve this task on local context only?
    - Or do you need larger (global) context?

How much data?

- Very general rule: you can start thinking about training a DNN if you have at least 1000 training samples
- **There is no data like more data**
- Common situation:
  - Baseline accuracy with a simple model: accuracy 60%
  - Super-well tuned “transformer with multi-head attention”: accuracy: 70%
    - Spend 1 week
  - Simple model with more data: 80%
    - Spend 2 days on data collection
- Spending time on data collection is a time well spent
  - Can always apply new models / training tricks when they are invented
  - Time spent on fine-tuning a fancy model might be wasted when suddenly a new model architecture is invented

Data augmentation

- *“The next best thing to real data is half-fake data”*
- Data augmentation: train artificial additional training data by corrupting the real data by some domain-specific way
- Often applied so that the amount of training data is increased 10 (or more) times
- Data augmentation can make a huge difference, especially if domain adaptation can be performed using augmentation

Data augmentation, continued

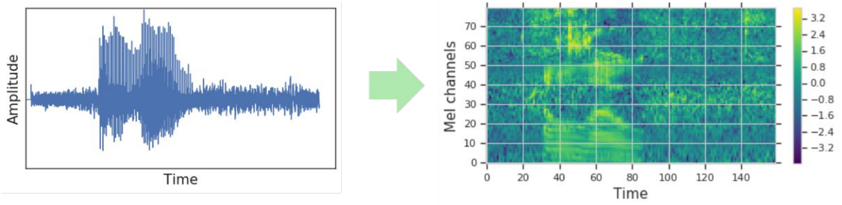
- For example:
  - Speech recognition model trained on clean data: accuracy on reverberant/noisy data **50%**
  - Add noise and reverberate clean training data: accuracy **85%**
- Data augmentation is more useful in domains where input data is “continuous”
  - Image processing
  - Speech processing
- Usually works better than trying to “fix” the test data (e.g., denoise)
- Almost never hurts accuracy

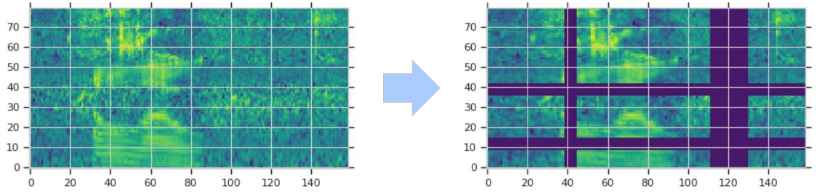
Data augmentation is speech recognition

- Speed perturbation
  - Add copies of training sentences, sped up or slown down by 10%
  - Simulates faster/slower speaking, but also different vocal tract characteristics
- Reverberation
  - Add room effects (using real or artificial **impulse responses**)
- Noise augmentation
  - Add background noise to (clean training data), e.g. street noise, engine, car, wind
  - Add “foreground noises” (e.g., door slams, claps, clicks)
  - Add music
  - Add “babble noise” (many persons speaking in the background)
  - All this with random signal-to-noise ratio, and many times

Spectral augmentation

- Proposed by Google recently (https://ai.googleblog.com/2019/04/specaugment-new-data-augmentation.html)
- Three types of modifications of the raw filterbank spectrogram, randomly chosen
  - Time warping
  - Masking blocks of consecutive filterbank features (horizontal)
  - Masking whole filterbank blocks in time (vertical)

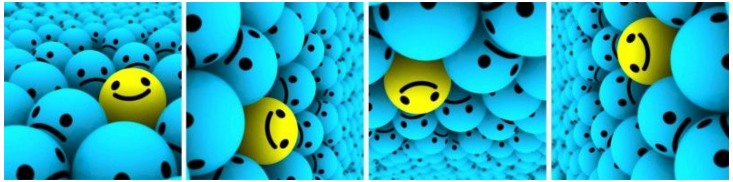




	LibriSpeech 960h		Switchboard 300h	
	test-clean	test-other	Switchboard	CallHome
Previous SOTA	2.95	7.50	8.3	17.3
Our Results	**2.5**	**5.8**	**6.8**	**14.1**

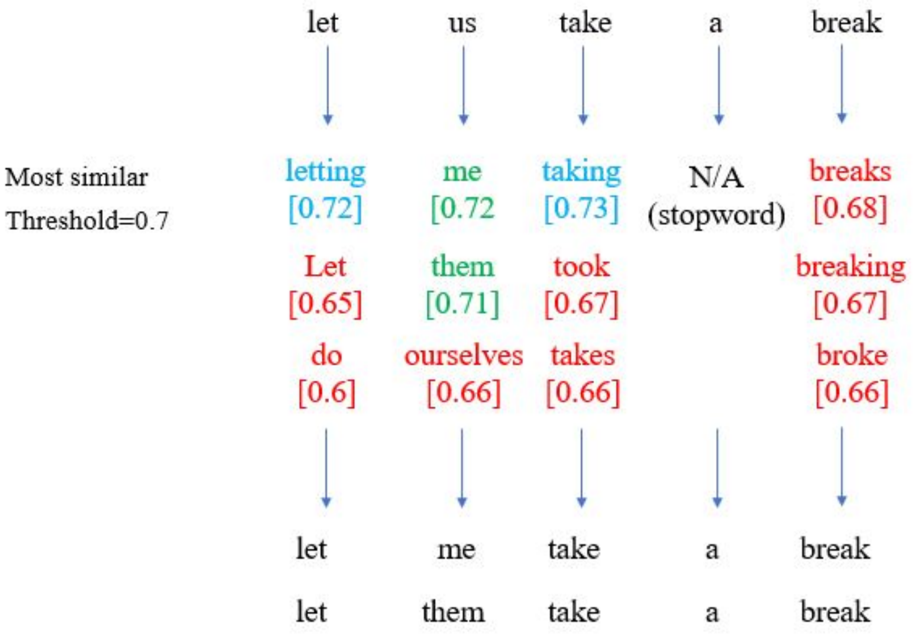
Data augmentation with images

- Flipping
- Rotation
- Translation
- Scaling
- Adding noise
- ...



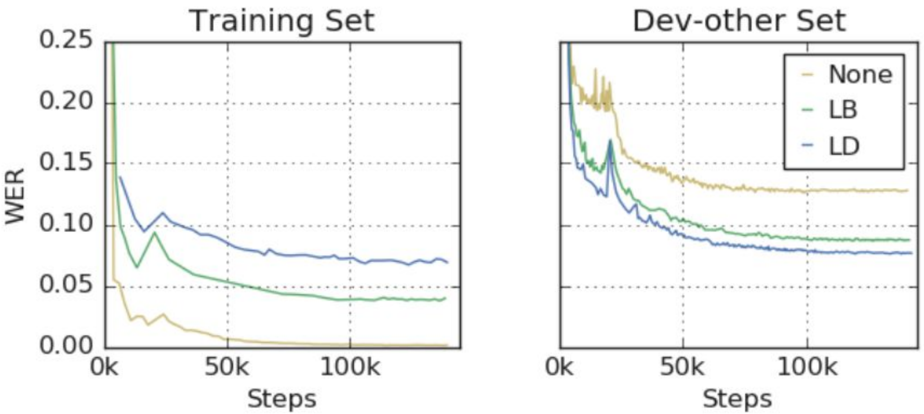
Data augmentation for NLP

- For text data, data augmentation is less useful
- One idea: replace words in training data with words that have similar embeddings
    - But usually better to just use pretrained word embeddings (or pretrained contextualized representations, like BERT)
- Alternative: use domain knowledge
    - E.g., replace company names in text with other company names from the business registry (if having a good coverage of company names is relevant to your task)



Data augmentation under-fits training data

- Data augmentation turns over-fitting problem into under-fitting problem
  - Loss when training augmented data is worse than when training on clean data
  - But loss (and error rate) on test data is better
- Thus, common methods that work against under-fitting help:
  - Using larger (wider and deeper) models
  - Training for more epochs

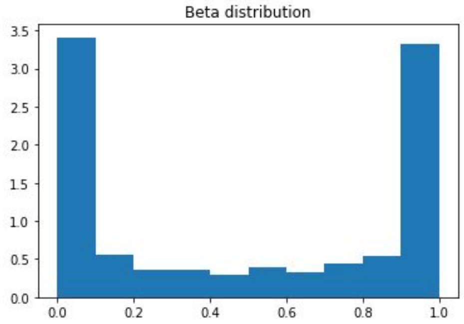


Mixup

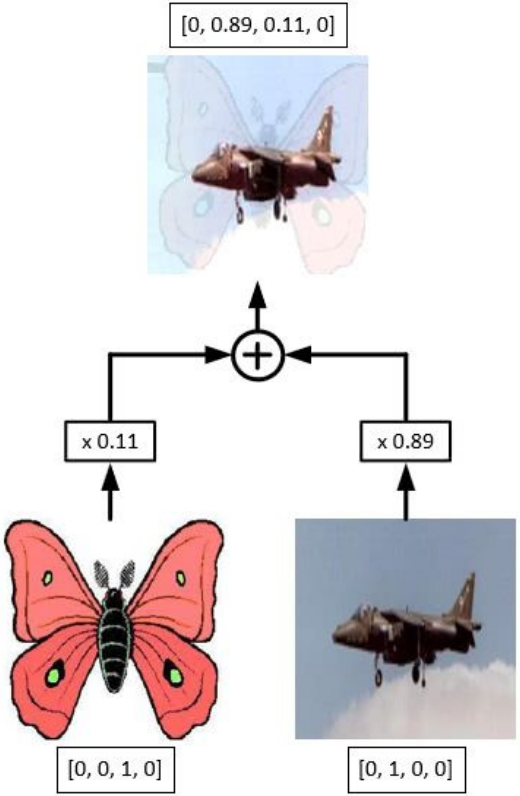
- During training, mix two training samples and the corresponding labels

$$\tilde{x} = \lambda x_i + (1-\lambda)x_j, \qquad \text{where } x_i, x_j \text{ are raw input vectors}$$
$$\tilde{y} = \lambda y_i + (1-\lambda)y_j, \qquad \text{where } y_i, y_j \text{ are one-hot label encodings}$$

- The mixup factor is drawn randomly from a beta distribution

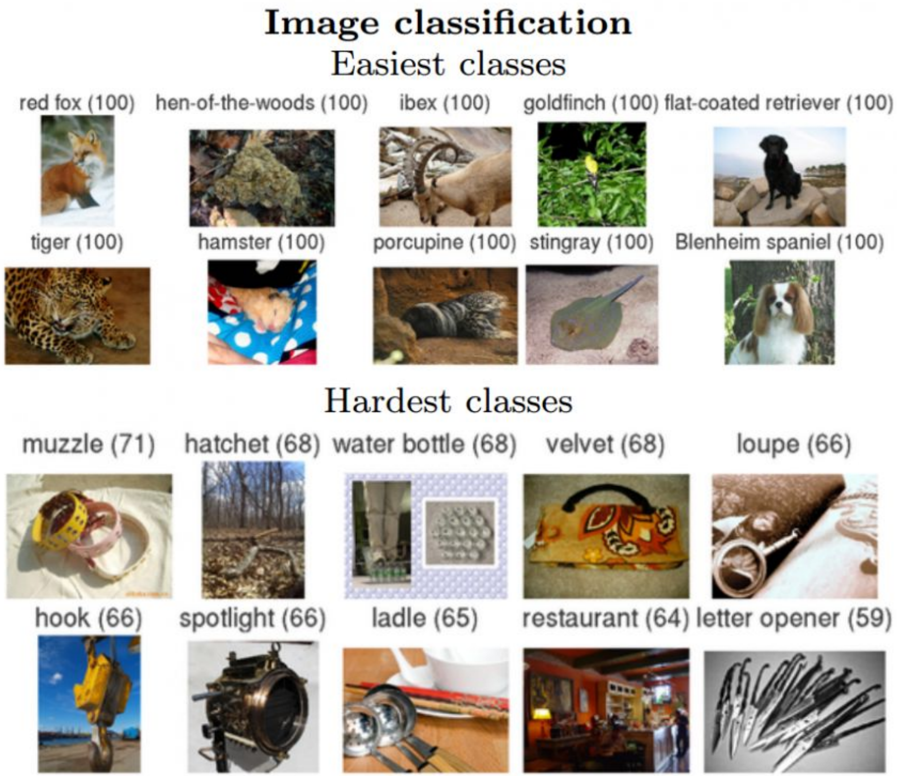


- Kind of surprising that it helps
- Not really applicable to text data



Pretraining

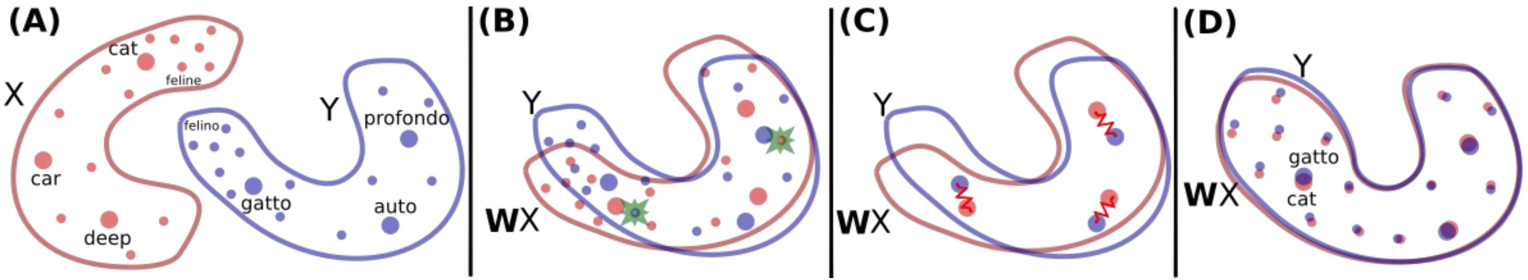
- *It rarely ever hurts to use a pretrained network if you can, even if you have enough data*
- Pretraining was first popularized by the image recognition community
    - Pretrain a ConvNet on ImageNet
    - ImageNet: dataset of 1.2M images, 1000 categories
    - Models trained on ImageNet can be used to initialize models **for completely other datasets** and improve performance significantly
    - Works even if you have only a few examples per object



Pretraining for NLP

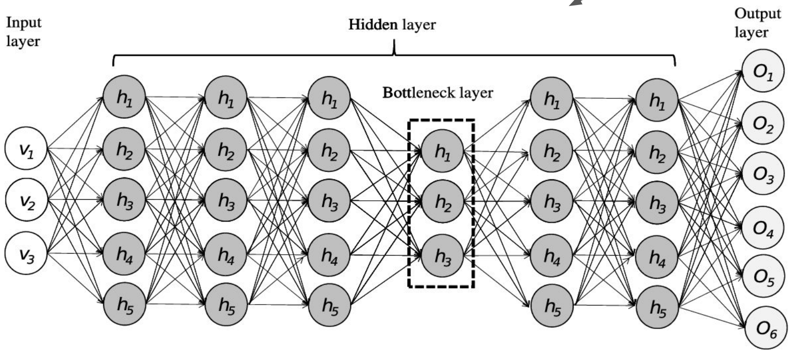
- Pretraining can have huge benefits on NLP tasks
  - Pretrained embeddings
  - Pretrained contextualized embeddings (e.g. BERT)
- Use pretrained model “found” from internet (many Estonian models available) or train your own

- Cross-lingual pretraining
- Main idea:
  - Train word embeddings for two languages (e.g., English and Italian)
  - Align English and Italian embeddings
  - Use English labeled data to train a text classification model
  - The model also works for Italian!
  - https://github.com/facebookresearch/MUSE



Pretraining for speech recognition

- Pretraining also works for speech recognition
- Example:
  - Train a model on a large *universal* corpus
  - Use a small corpus (e.g. 20 h) of in-domain data to finetune the model



- Alternative: (multilingual) **bottleneck** features
- Useful in a situation when you have very little data (e.g. 40h) from a particular language
- Idea:
  - Train a DNN acoustic model on other language(s)
  - The DNN has a narrow *bottleneck* layer before the output
  - Use the bottleneck as a **feature extractor**
  - Train a speech recognition model on the low-resource language on bottleneck features, instead of the usual filterbanks

Pretraining for speaker recognition

- Pretraining is **crucial** for speaker recognition (e.g., speaker identification and verification)
- Idea:
    - Use a large speaker database (speech-speaker pairs) to train a speaker identification model (convolutional model with global pooling)
    - Use the model's pooling layer output as a feature extractor
    - Now, use the feature extractor to get *speaker embeddings* (called x-vectors) for speakers you need to cover
    - Only a few sentences per speaker is now enough

VoxCeleb

- But how to get the “large speaker database”, needed for training the voice feature extractor?
- VoxCeleb:
  - Query YouTube, using e.g. “Elon Musk interview”
  - Use Face Recognition to identify frames where Elon Musk is in the frame
  - Use lip-syncing to check that lips and audio are in sync
  - Extract all such frames and use this as training data
  - Alternative: given several videos of Elon Musk, the segments where he is speaking can be found automatically (invented in our lab, based on a Master thesis)

What model to use in NLP?

- What model to use for NLP task (e.g. text classification, information extraction), given that you have reasonable amount of training data?
- For most NLP tasks, **Transformer** (multi-layer multi-head attention with position encodings) is currently the best choice
- For English, use BERT for getting contextualized word embeddings
- For Estonian, train your own BERT?

Debugging neural networks

- *Neural net training fails silently*
- Tricky to unit test
- Your net can still (shockingly) work pretty well even if your training data is corrupted somehow
  - E.g., maybe you feed it columns instead of rows of data
  - It’s because neural nets can memorize the data to some extent
- Most of the time when you screw something up, it will train but silently work a bit worse

Recipe for training neural nets

- Verify your data
- Start with simple models
  - No data augmentation, fancy learning rate schedules, etc
- Train an input-independent baseline, (e.g. easiest is to just set all your inputs to zero)
  - This should perform worse than when you actually plug in your data without zeroing it out. Does it?
- Overfit a single batch of only a few examples (e.g. as little as two).
  - Verify that we can reach the lowest achievable loss (e.g. zero)

Recipe, continued

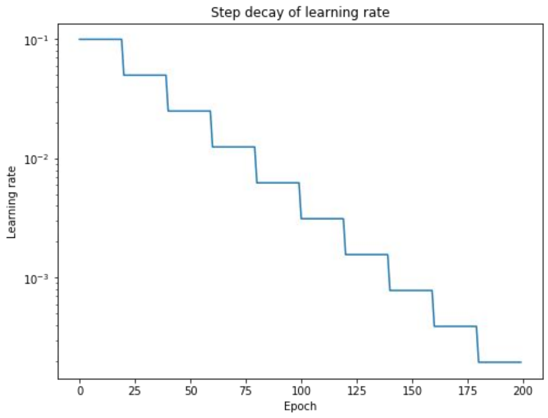
- Don’t be a hero
  - Start with a model that is known to give good results on similar tasks
  - You’re allowed to do something more custom later and beat this
- One complexity at a time
  - At fanciness to your model one by one and make sure it improves the model
- Leave learning rate tuning to the end

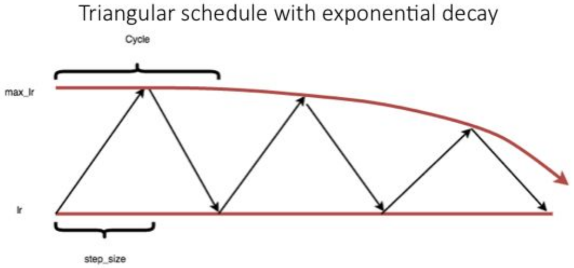
Recipe, continued

- Regularize (avoid overfitting)
  - Try to get more data
  - Data augmentation, try to be creative
  - Pretraining rarely hurts
  - Unsupervised learning usually gives very little improvement, and is very complicated
  - Try with less features
  - Try smaller model size
  - Add dropout
  - Add L2 regularization (weight decay)
  - Early stopping
  - Try larger model, but stop early

Some tricks: learning rate schedule

- Learning rate schedule
  - E.g., step decay (0.1 at the beginning, 0.0001 at the end)
  - Or: constant learning rate, but decrease it when the loss on dev data doesn't decrease any more (`ReduceLROnPlateau` in Pytorch)
  - More recent: triangular learning rates
    - Start at learning rate 0.001
    - Gradually increase it to 0.01 at epoch 25%
    - Then gradually decrease it until the end
  - Cyclic learning rates





Some tricks: dropout, label smoothing

- Similarly to learning rate, dropout can be also scheduled
  - E.g.: 0 dropout at the beginning,
  - Increase dropout gradually to 0.3 at epoch 50%
  - Then gradually decrease it back to 0

- Label smoothing
  - Very simple idea: redistribute 0.1 of probability mass from the correct label to all other labels
  - Often used in Google’s papers
  - Results in less loss fluctuations on dev data during epochs in my experience
  - Maybe disable label smoothing during the last training phase

Ensembles

- *Model ensembles are a pretty much guaranteed way to gain 2% of accuracy on anything*
  - Ensembles of models of different architecture result in the best performance
  - Even if a particular model is relatively bad (compared to the best ones in the ensemble), combining its predictions with others can give surprising gains
- *If you can’t afford the computation at test time, try distilling your ensemble into a network using dark knowledge*
  - Compute posteriors (target probabilities) for your training data using the ensemble, and train a final model on the posteriors (instead of real targets)
  - This way, the final model learns to mimic the ensemble