NPFL099 Statistical Dialogue Systems
4. Training Neural Nets

http://ufal.cz/npfl099

Ondřej Dušek

20. 10. 2025

Charles University
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

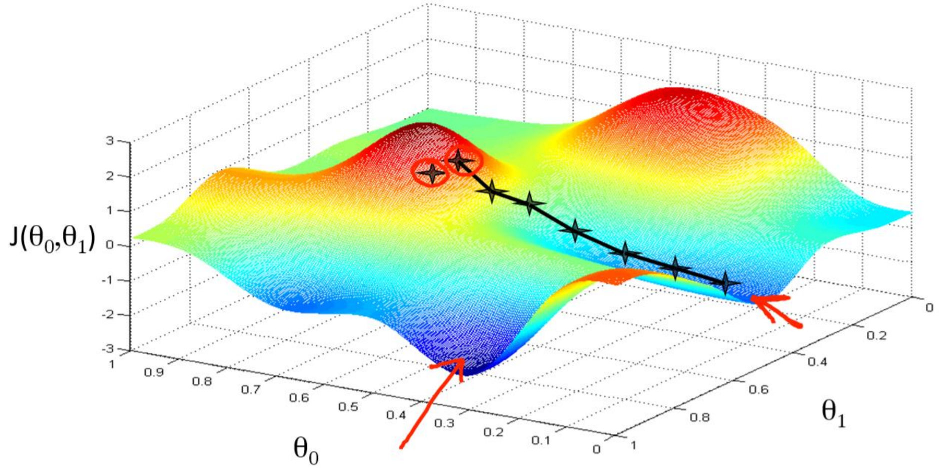
BY SA
unless otherwise stated

Recap: Neural Nets

- **complex functions, composed of simple functions** (=layers)
  - linear, ReLU, tanh, sigmoid, softmax
- **fully differentiable**
- different arrangements:
  - feed forward / multi-layer perceptron
  - CNNs
  - RNNs (LSTM/GRU)
  - attention
  - Transformer
- input: binary, float, embedding
- tasks/problems: classification, regression, structured (sequences/ranking)

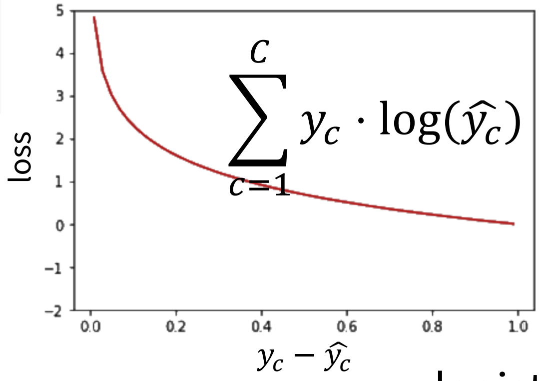
Supervised Training: Gradient Descent

- supervised training– **gradient descent** methods
  - minimizing a **cost/loss function**
    (notion of error – given system output, how far off are we?)
  - calculus: derivative = steepness/slope
  - follow the slope to find the minimum – derivative gives the direction
  - **learning rate** = how fast we go (needs to be tuned)
- gradient typically computed (=averaged) over **mini-batches**
  - random bunches of a few training instances
  - not as erratic as using just 1 instance,
    not as slow as computing over whole data
  - **stochastic gradient descent**
  - batches may be **accumulated** to fit into memory
    - e.g. your GPU only fits one instance
      → compute gradients multiple times, then do 1 update



https://hackernoon.com/gradient-descent-aynk-7cbe95a778da

Cost/Loss Functions



logistic

- differ based on what we're trying to predict
- **logistic / log loss** ("cross entropy")
  - for classification / softmax – including **word prediction**
    - classes from the whole dictionary
    - correct class <100% prob. → loss
  - pretty stupid for sequences, but works →
    - sequence shifted by 1 ⇒ everything wrong
- **squared error loss** – for regression
  - forcing the predicted float value to be close to actual one
- **hinge loss** – binary classif. (SVMs), ranking
  - forcing the correct sign
- many others, variants

reference: *Blue Spice is expensive*

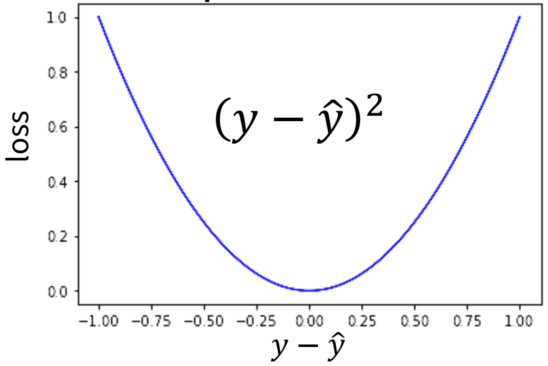
prediction: *expensive*

cheap

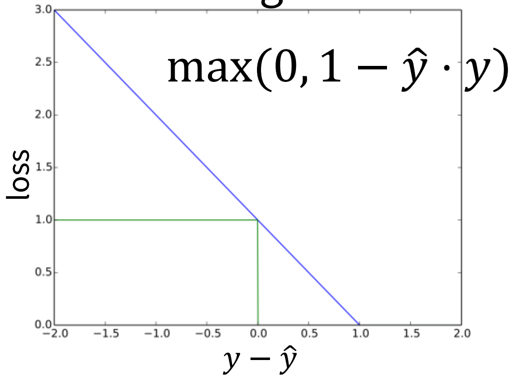
pricey

in the expensive price range

squared error



hinge



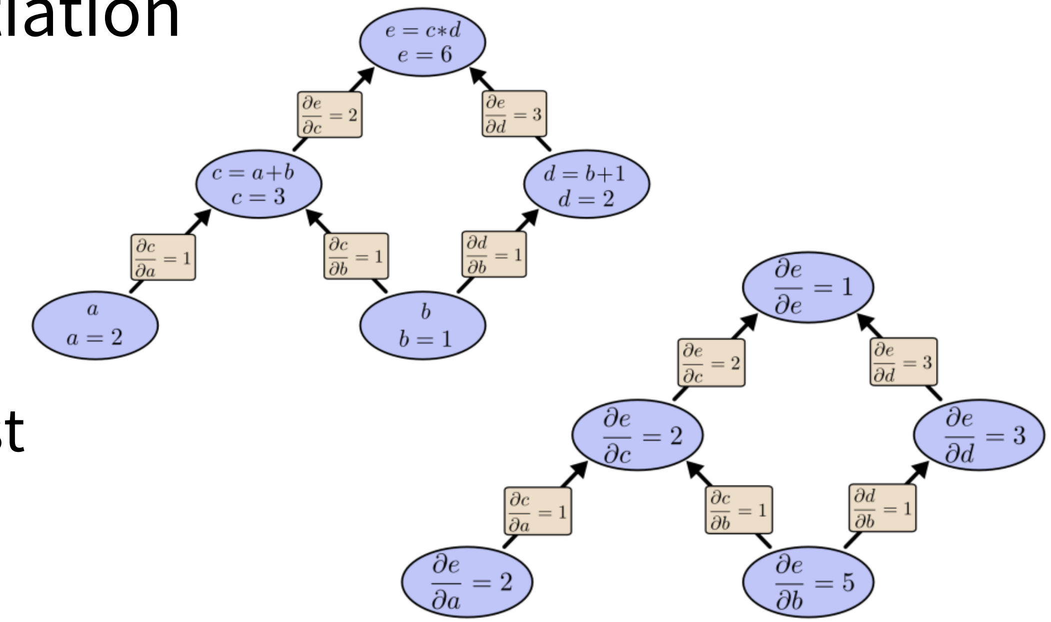
https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/
https://medium.com/@risingdeveloper/visualization-of-some-loss-functions-for-deep-learning-with-tensorflow-9f60be9d09f9 , https://en.wikipedia.org/wiki/Hinge_loss

Backpropagation

https://www.mathsisfun.com/calculus/derivatives-rules.html

Rules	Function	Derivative
Multiplication by constant	cf	cf′
Power Rule	x^n	nx^{n-1}
Sum Rule	f + g	f′ + g′
Difference Rule	f - g	f′ − g′
Product Rule	fg	f g′ + f′ g
Quotient Rule	f/g	$\frac{f'g - g'f}{g^2}$
Reciprocal Rule	1/f	$-f'/f^2$
Chain Rule (as "Composition of Functions")	f º g	(f′ º g) × g′

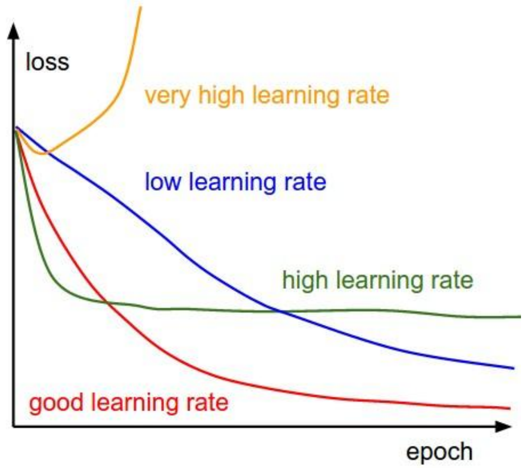
- network ~ computational graph
  - reflects function/layer composition
- composed function derivatives – simple rules
  - basically summing over different paths
  - factoring ~ merging paths at every node
- **backpropagation** = reverse-mode differentiation
  - going back from output to input
  - ~ how every node affects the output
  - your graph **output = cost function**
  - → derivatives of all parameters w. r. t. cost
  - one pass through the network only → easy & fast
  - NN frameworks do this automatically



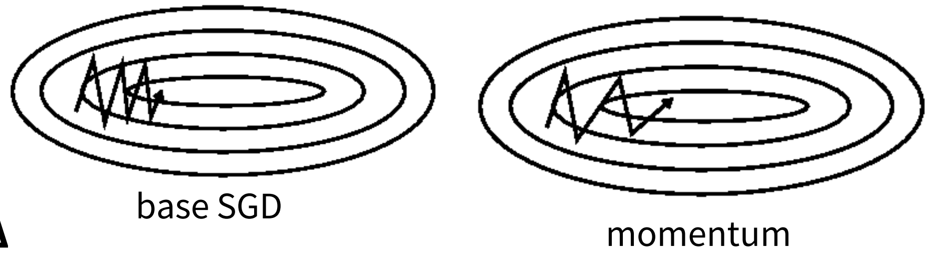
http://colah.github.io/posts/2015-08-Backprop/

Learning Rate (α) & Momentum

- α: **most important parameter** in (stochastic) gradient descent
- tricky to tune:
  - too high α = may not find optimum
  - too low α = may take forever
- **Learning rate decay**: start high, lower α gradually
  - make bigger steps (to speed learning)
  - slow down when you're almost there (to avoid overshooting)
  - linear, stepwise, exponential
  - **reduce-on-plateau** – check every now and then if we're still improving, reduce LR if not
- **Momentum**: moving average of gradients
  - make learning less erratic
  - $m = \beta \cdot m + (1 - \beta) \cdot \Delta$, update by m instead of Δ



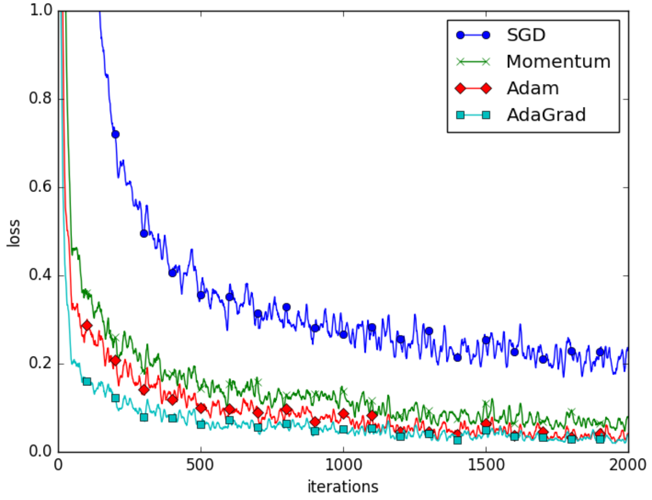
http://cs231n.github.io/neural-networks-3/



https://ruder.io/optimizing-gradient-descent/

Optimizers

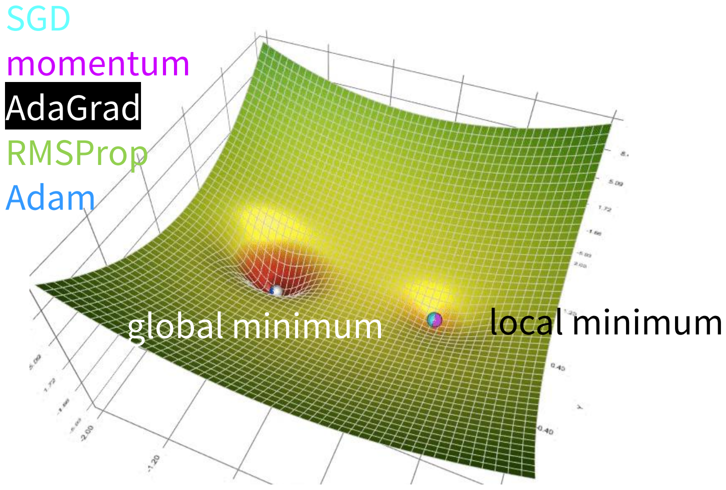
http://kaeken.hatenablog.com/entry/2016/11/10/203151



- Better LR management
  - change LR based on gradients, less sensitive to settings
- **AdaGrad** – all history
  - remember sum of total gradients squared: $\sum_t \Delta_t^2$
  - divide LR by $\sqrt{\sum \Delta_t^2}$
  - variants: **Adadelta**, **RMSProp** – slower LR drop
- **Adam** – per-parameter momentum
  - moving averages for Δ & Δ^2:

    $m = \beta_1 \cdot m + (1 - \beta_1)\Delta, v = \beta_2 \cdot v + (1 - \beta_2)\Delta^2$
  - use m instead of Δ, divide LR by $\sqrt{v}$
  - often used as default nowadays

(Kingma & Ba, 2015)
https://arxiv.org/abs/1412.6980



- variant: **AdamW** – better regularization
  - not much difference though

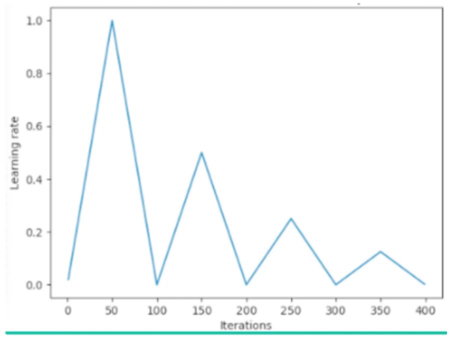
(Loshchilov & Hutter, 2019)
https://arxiv.org/abs/1711.05101

https://ruder.io/optimizing-gradient-descent/

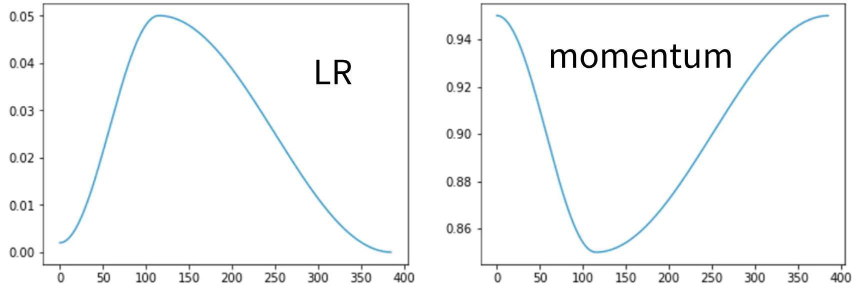
https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

Schedulers

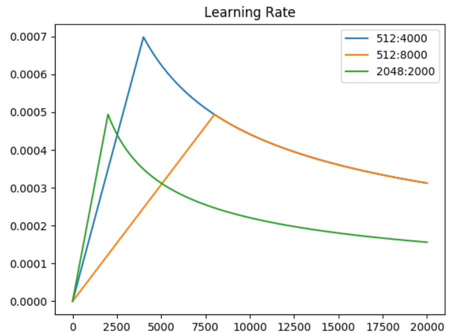
- more fiddling with LR – **warm-ups**
  - start learning slowly, then increase LR, then reduce again
  - may be repeated (**warm restarts**),
    with lowered maximum LR
    - allow to diverge slightly – work around local minima
- multiple options:
  - cyclical (=warm restarts) – linear, cosine annealing
  - **one cycle** – same, just don't restart
  - **Noam scheduler** – linear warm-up, decay by $\sqrt{\text{steps}}$
- combine with base SGD or Adam/Adadelta etc.
  - momentum updated inversely to LR
  - may have less effect with optimizers
    - trade-off: speed vs. sensitivity to parameter settings



cyclical scheduler (warm restarts)



one cycle with cosine annealing

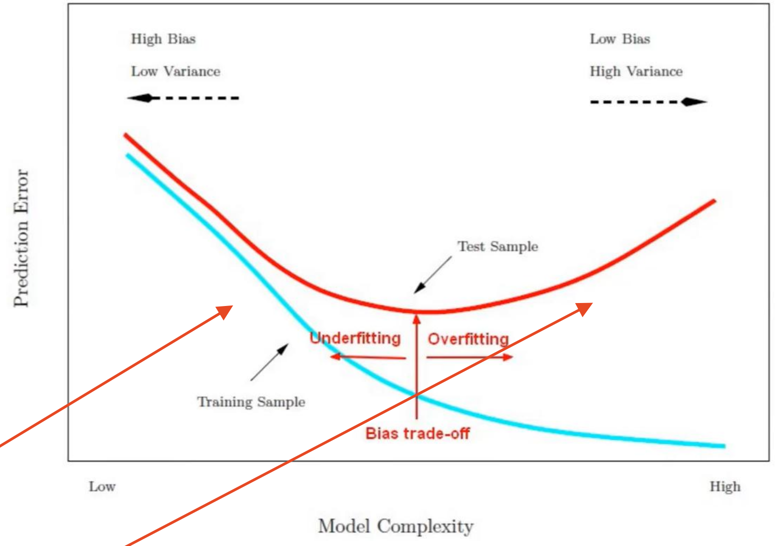


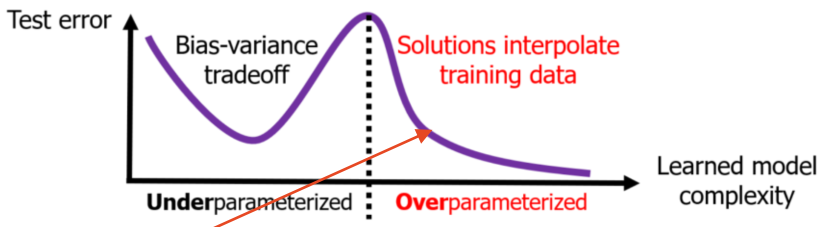
Noam scheduler with different parameters

https://spell.ml/blog/lr-schedulers-and-adaptive-optimizers-YHmwMhAAACYADm6F
https://nn.labml.ai/optimizers/noam.html

When to stop training

https://www.andreaperlato.com/theorypost/bias-variance-trade-off/





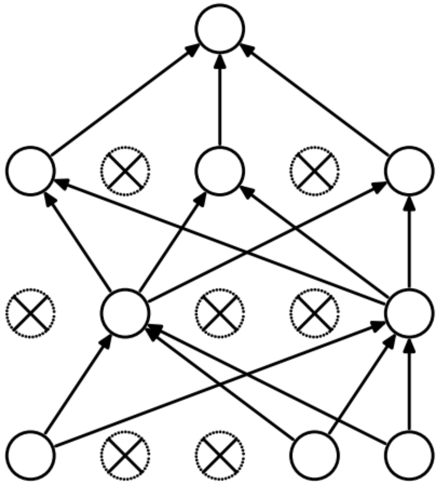
- generally, when cost stops going down
  - despite all the LR fiddling
- problem: **overfitting**
  - cost low on training set, high on validation set
  - network essentially memorized the training set
  - → **check on validation set** after each epoch (pass through data)
  - stop when cost goes up on validation set
  - regularization (see →) helps delay overfitting
- **bias-variance** trade-off:
  - smaller models may underfit (high bias, low variance = not flexible enough)
  - larger models likely to overfit (too flexible, memorize data)
  - **grokking** – models overfit soo much they start to interpolate/generalize

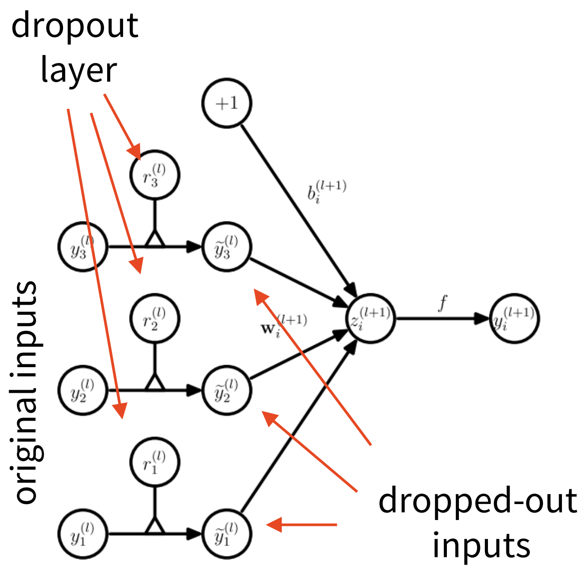
(Dar et al., 2021) https://arxiv.org/abs/2109.02355 (Wang et al., 2024) https://arxiv.org/abs/2405.15071
(Power et al., 2022) http://arxiv.org/abs/2201.02177 (Li et al., 2025) https://arxiv.org/abs/2506.21551

Regularization: Dropout

(Srivastava et al., 2014)
http://jmlr.org/papers/v15/srivastava14a.html

- regularization: preventing overfitting
  - making it harder for the network to learn, adding noise
- **Dropout** – simple regularization technique
  - more effective than e.g. weight decay (L2)
  - **zero out some neurons/connections**
    in the network at random
  - technically: multiply by dropout layer
    - 0/1 with some probability (typically 0.5–0.8)
  - at training time only – full network for prediction
  - weights scaled down after training
    - they end up larger than normal because there’s fewer nodes
    - done by libraries automatically
  - may need larger networks to compensate



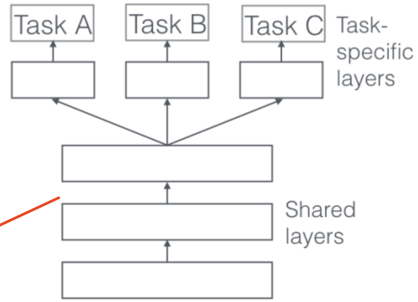


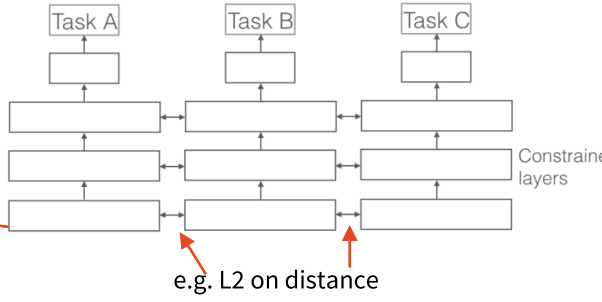
(b) Dropout network

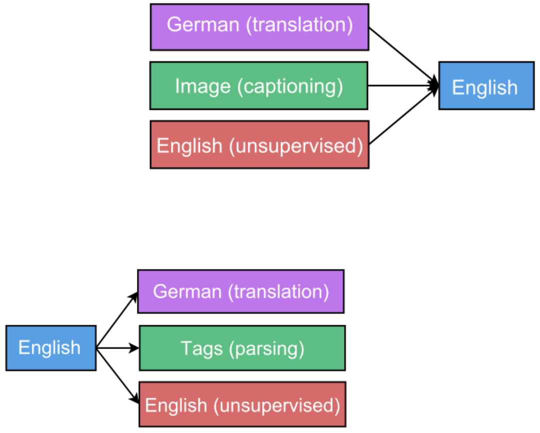
Regularization: Multi-task Learning

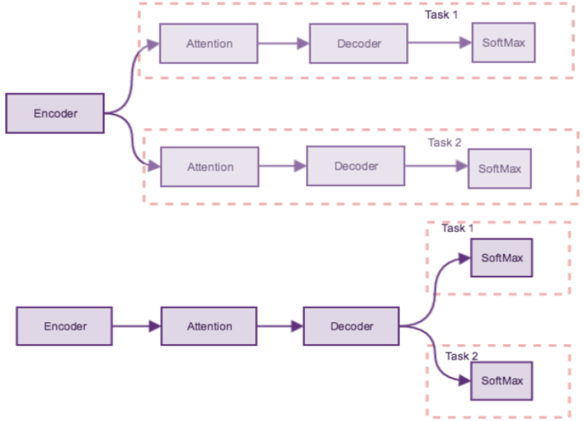
(Ruder, 2017)
http://arxiv.org/abs/1706.05098
(Fan et al., 2017)
http://arxiv.org/abs/1706.04326
(Luong et al., 2016)
http://arxiv.org/abs/1511.06114

- achieve better generalization by **learning more things at once**
  - a form of regularization
  - implicit data augmentation
  - biasing/focusing the model
    - e.g. by explicitly training for an important subtask
- parts of network shared, parts task-specific
  - hard sharing = parameters truly shared (most common)
  - soft sharing = regularization by parameter distance
  - different approaches w. r. t. what to share
- training – **alternating** between tasks
  - **catastrophic forgetting:**
    if you don't alternate,
    the network forgets previous tasks



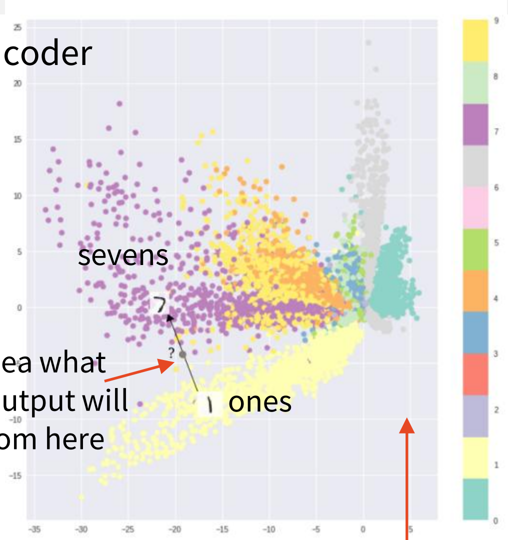




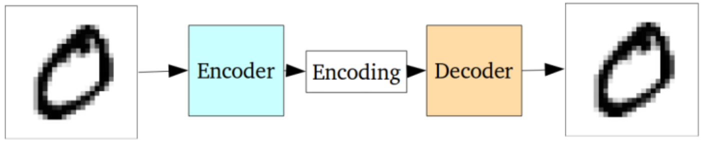


Autoencoders

MNIST digits autoencoder
latent space



- Using NNs as **generative models**
  - more than just classification – modelling the whole distribution
    - (of e.g. possible texts, images)
  - generate new instances that look similar to training data
- **Autoencoder**: input → encoding → input
  - encoding ~ "embedding" in latent space (i.e. some vector)

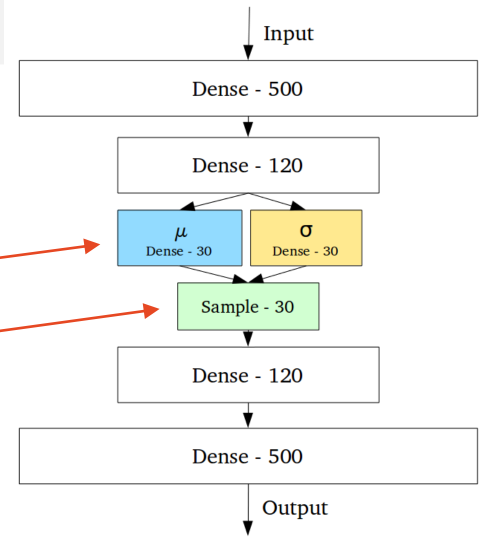


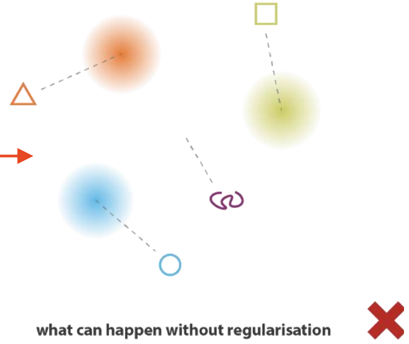
  - trained by reconstruction loss
  - problem: can't easily get valid embeddings for generating new outputs
    - parts of embedding space might be unused – will generate weird stuff
    - no easy interpretation of embeddings – no idea what the model will generate
- **Denoising autoencoder** – extension:
  - add noise to inputs, train to generate clean outputs
  - use in multi-task learning, representations for use in downstream tasks

https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

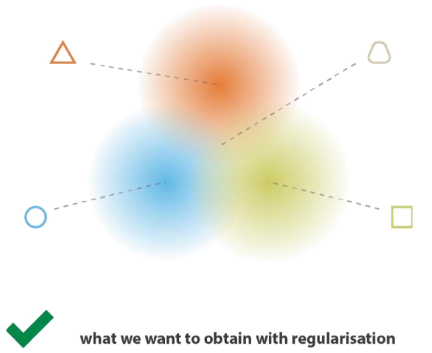
Variational Autoencoders

- Making the encoding latent space more useful
  - using **Gaussians** – continuous space by design
  - encoding input into vectors of means μ & std. deviations σ
  - sampling encodings from $N(\mu, \sigma)$ for generation
    - samples vary a bit even for the same input
    - decoder learns to be more robust
- model can degenerate into normal AE ($\sigma \rightarrow 0$)
    - we need to encourage some σ, smoothness, overlap ($\mu \sim 0$)
    - add **2nd loss: KL divergence** from $N(0,1)$
    - VAE learns a trade-off between
      using unit Gaussians & reconstructing inputs
- **Reparameterization trick:** bypass sampling when training
  - $z \sim \mu + \sigma \cdot \mathcal{N}(0,1)$, then differentiate w. r. t. μ and σ

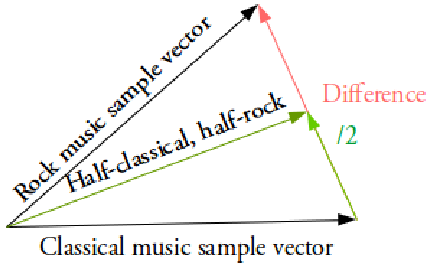




what can happen without regularisation



what we want to obtain with regularisation



https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf
https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73
http://kvfrans.com/variational-autoencoders-explained/
https://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/

Discrete VAE: Gumbel-Softmax

(Jang et al., 2017)
https://arxiv.org/abs/1611.01144

- **VAE for discrete/categorial distributions**
  - this makes the latent space better interpretable
  - similar reparameterization trick to ensure differentiability
- **Gumbel-max trick**:
  - categorial distribution π with probabilities π_i
  - sampling from π: $z = \mathrm{onehot}(\arg\max_i (\log \pi_i + g_i))$
- Swap argmax for softmax with temperature τ
  - differs from π if $\tau > 0$, but may be close to it
  - approx. sample of the true distribution
  - fully differentiable
  - g_i bypassed in differentiation, same as $\mathcal{N}(0,1)$ in Gaussian sampling

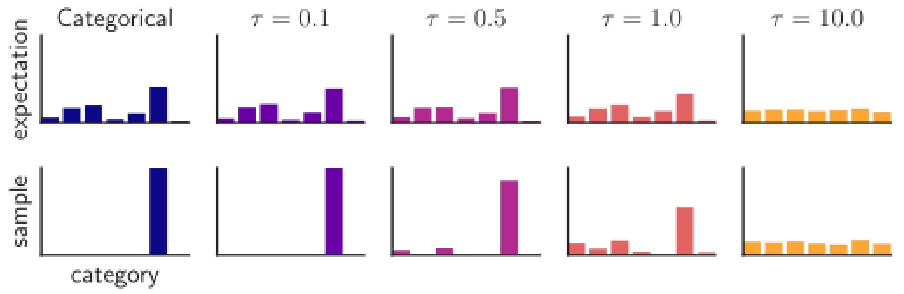
Gumbel noise:

$$g_i = -\log(-\log(\mathrm{Uniform}(0,1)))$$

$$y_i = \frac{\exp\left(\frac{\log(\pi_i) + g_i}{\tau}\right)}{\sum_{j=1}^{N} \exp\left(\frac{\log(\pi_j) + g_j}{\tau}\right)}$$

$\tau \to 0$: more like one-hot

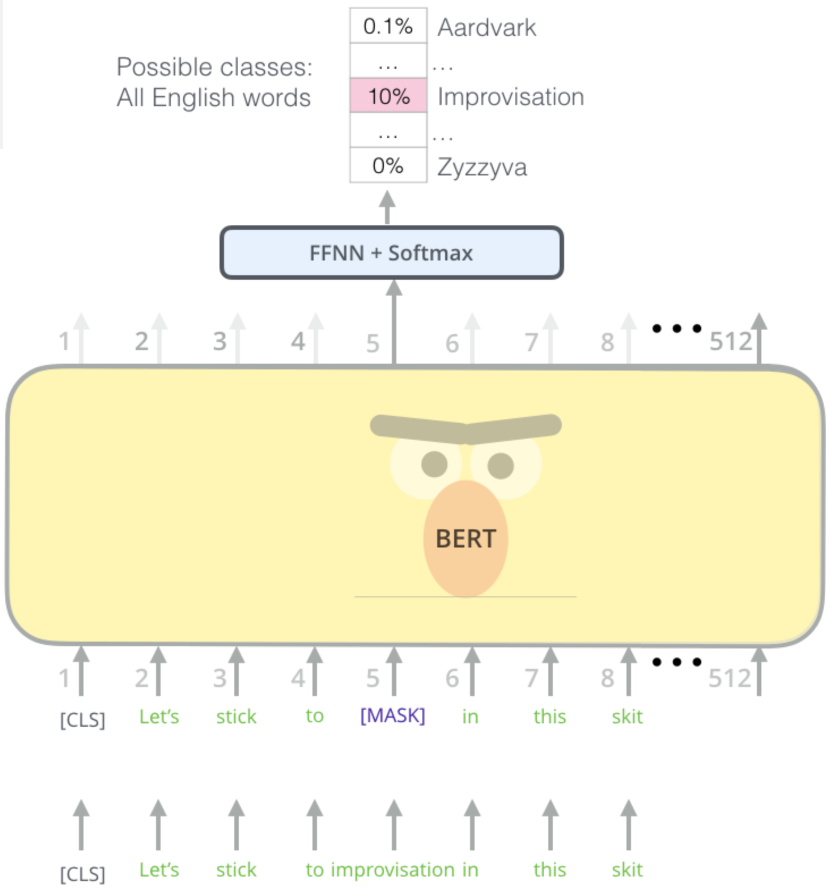
$\tau \to \infty$: more like uniform

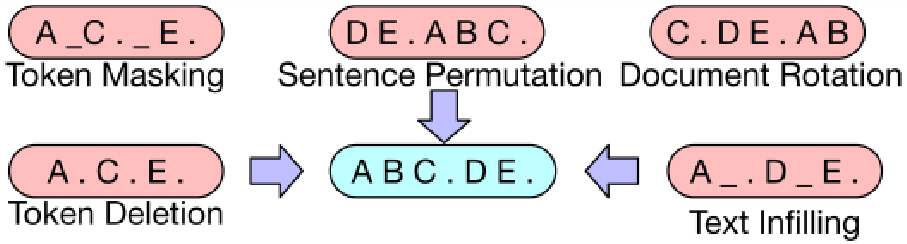


https://anotherdatum.com/gumbel-gan.html

Self-supervised training

- train supervised, but **don't provide labels**
  - use naturally occurring labels
  - create labels automatically somehow
  - use specific tasks that don't require manual labels
- good to train on huge amounts of data
  - language modelling
    - **next-word prediction**
    - **MLM** – masked word prediction (~like word2vec)
    - masked span prediction: MLM for multiple words
  - **autoencoding**: predict your own input
  - denoising: corrupt data & learn to fix them
  - learn from rule-based annotation (not ideal)
- unsupervised, but with supervised approaches





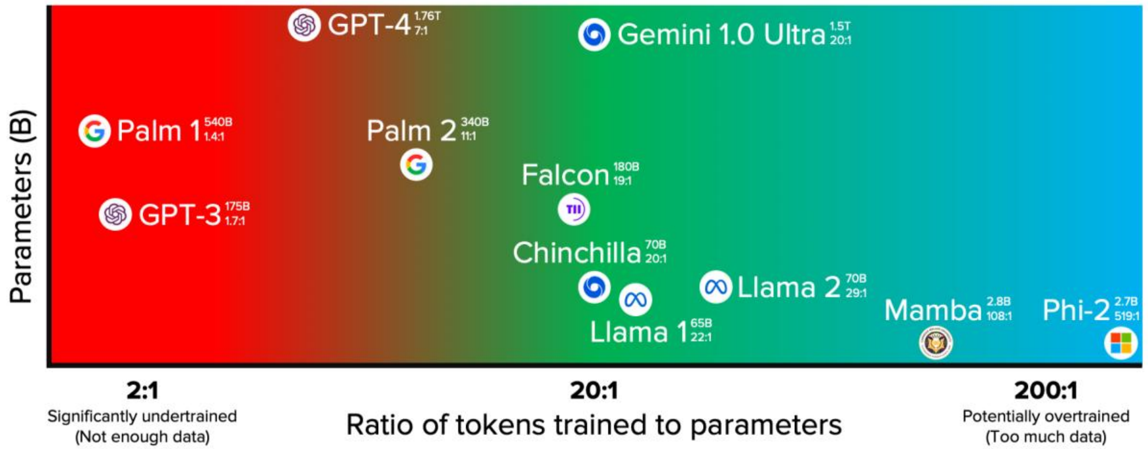
http://jalammar.github.io/illustrated-bert/
https://ai.stackexchange.com/questions/10623/what-is-self-supervised-learning-in-machine-learning

(Lewis et al., 2020) https://aclanthology.org/2020.acl-main.703/

Pretraining & Finetuning: Pretrained LMs

- 2-step training:
  1. **Pretrain** a model on a huge dataset (**self-supervised**, language-based tasks)
  2. **Fine-tune** for your own task on your smaller data (**supervised**)
- ~ pretrained “contextual embeddings” (“better word2vec”, typically Transformer)
- Model capability is all about the data
  - the larger model, the more you need (“Chinchilla scaling laws”)
  - anyway the more (& better quality), the better

https://twitter.com/Thom_Wolf/status/1766783830839406596



https://lifearchitect.ai/chinchilla/
https://www.harmdevries.com/post/model-size-vs-compute-overhead/

Pretrained (Large) Language Models (PLMs/LLMs)

(Zhao et al., 2023)
http://arxiv.org/abs/2303.18223

- **BERT**/**RoBERTa**: Transformer encoder
  - trained by masked language modeling, good for classification
- **GPT-2,** most LLMs (**GPT-3**, **Llama**, **Mistral, Gemma, Phi, Qwen**…): decoder
  - trained by next-word prediction (=language modeling), good for generation / prompting
- **BART, T5** – encoder-decoder (many training tasks, good for generation)
- multilingual: **XLM-RoBERTa**, **mBART**, **mT5**, **Aya**
- many models released plug-and-play
  - **!!** others (GPT-3/ChatGPT/GPT-4, Claude… closed & API-only)

https://huggingface.co/
https://ollama.com/

- PLM vs. LLM distinction a bit vague
  - generally >1B, but more on behavior
  - PLMs: ready to finetune
  - LLMs: ready to prompt (→ →)

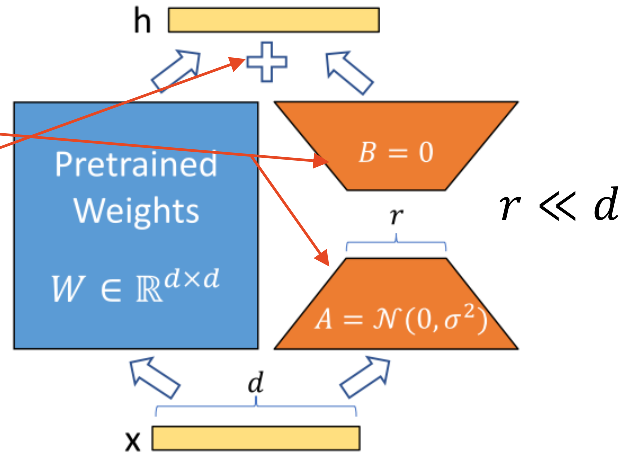
> (((J()(J() 'yoav))))
> @yoavgo
>
> "How large should a model be to qualify as an LLM" is a vacuous question. LLMs are NOT about size, they are about having a set of behaviors that happen to correlate with those exhibited by GPT-3/ ChatGPT, which were large (and not exhibited by GPT2, BERT, T5, which were smaller).

(controversial! see discussion 🍿) https://x.com/yoavgo/status/1828383882317549765

Parameter-efficient Finetuning

(Lialin et al., 2023) http://arxiv.org/abs/2303.15647
(Sabry & Belz, 2023) http://arxiv.org/abs/2304.12410

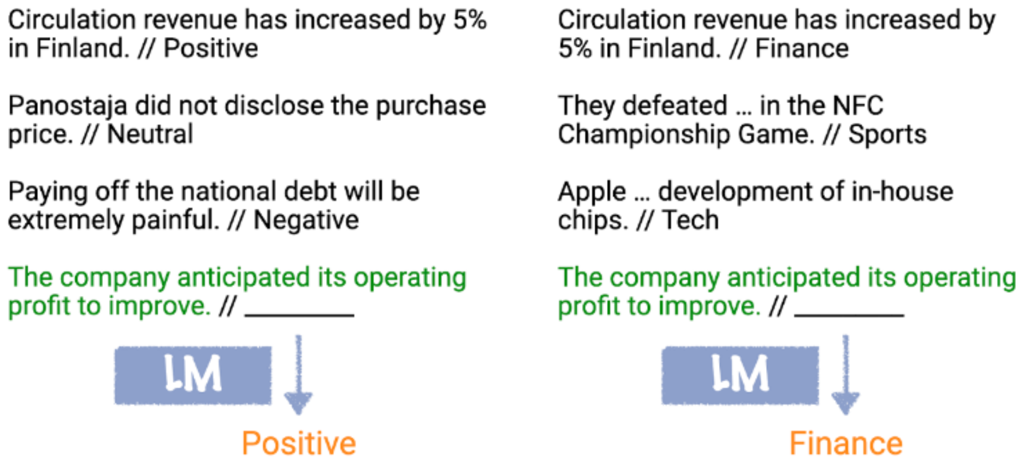
- Finetuning large models: don’t update all parameters
  - faster, less memory-hungry (fewer gradients/momentums etc.)
  - trains faster
  - less prone to overfitting (~ regularization)
- Add few parameters & only update these
  - **Adapters** – small feed-forward networks after/on top of each layer
  - **Soft prompts** – tune a few special embeddings & use them in a prompt
  - **LoRA** (low-rank adaptation):
    - updates = 2 decomposition matrixes A, B (parallel to each layer)
    - update = multiplication AB
    - $2 \times r \times d$ is much smaller than full weights (d^2)
    - update is added to original weights on the fly
  - **QLoRA** – LoRA + quantized 4/8-bit computation
    - to fit large models onto a small GPU



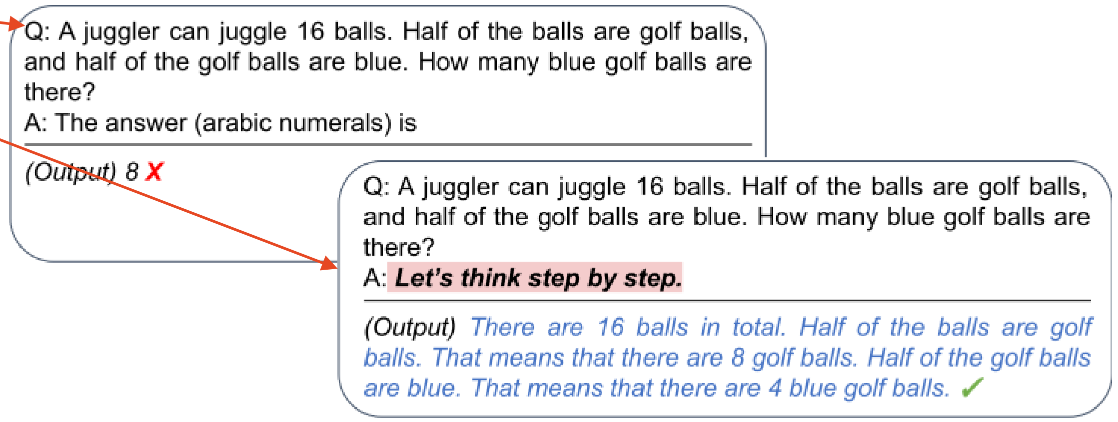
(Houlsby et al., 2019) http://proceedings.mlr.press/v97/houlsby19a.html
(Lester et al., 2021) https://aclanthology.org/2021.emnlp-main.243
(Hu et al., 2021) http://arxiv.org/abs/2106.09685
(Dettmers et al., 2023) http://arxiv.org/abs/2305.14314

LLMs: Prompting = In-context Learning

- No model finetuning, just show a few examples in the input (=prompt)
- pretrained LMs can do various tasks, given the right prompt
  - they've seen many tasks in training data
  - only works with the larger LMs (>1B)
- adjusting prompts often helps
  - "**prompt engineering**"
  - **zero-shot** (no examples) vs. **few-shot**
  - **chain-of-thought** prompting: "let's think step by step"
  - adding / rephrasing **instructions** (see → →)



http://ai.stanford.edu/blog/understanding-incontext/



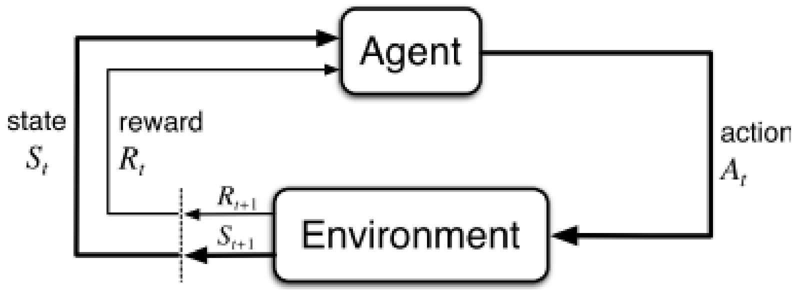
https://lilianweng.github.io/posts/2023-03-15-prompt-engineering/

(Liu et al., 2023) https://arxiv.org/abs/2107.13586

Reinforcement Learning

- Learning from **weaker supervision**
  - only get feedback once in a while, not for every output
  - good for globally optimizing sequence generation
    - you know if the whole sequence is good
    - you don't know if step X is good
  - sequence = e.g. sentence, dialogue
- Framing the problem as **states & actions & rewards**
  - "robot moving in space", but works for dialogue too
  - state = generation so far (sentence, dialogue state)
  - action = one generation output (word, system dialogue act)
  - defining rewards might be an issue
- Training: **maximizing long-term reward**
  - via state/action values (Q function)
  - directly – optimizing policy

your model

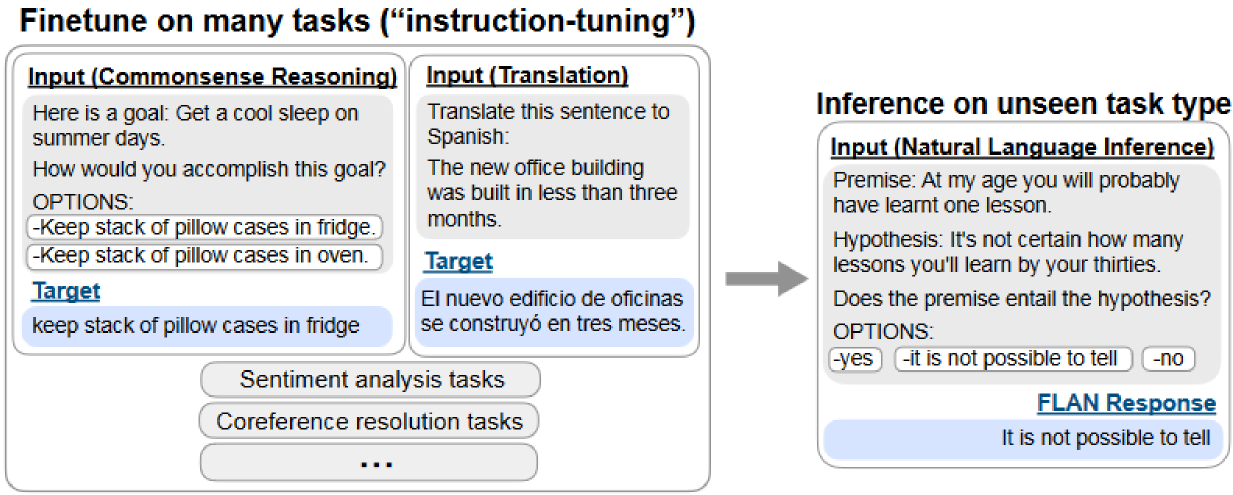


(Sutton & Barto, 2018)

some definition of rewards

Instruction Tuning / RL from Human Feedback

(Wei et al., 2022) https://arxiv.org/abs/2109.01652



- Finetune for use with prompting
  - "in-domain" for what it's used later
  - Use **instructions** (task description) **+ solution** in prompts
  - Many different tasks
  - Specific datasets available
  - Some LLMs released as base ("foundation") & instruction-tuned versions
- RL improvements on top of this (~InstructGPT/ChatGPT):

  (Ouyang et al., 2022) http://arxiv.org/abs/2203.02155 https://openai.com/blog/chatgpt

  1) generate lots of outputs for instructions
  2) have humans rate them
  3) learn a rating model (some kind of other LM: instruction + solution → score)
  4) use rating model score as reward in RL
  - main point: **reward is global** (not token-by-token) – RL-free alternatives exist

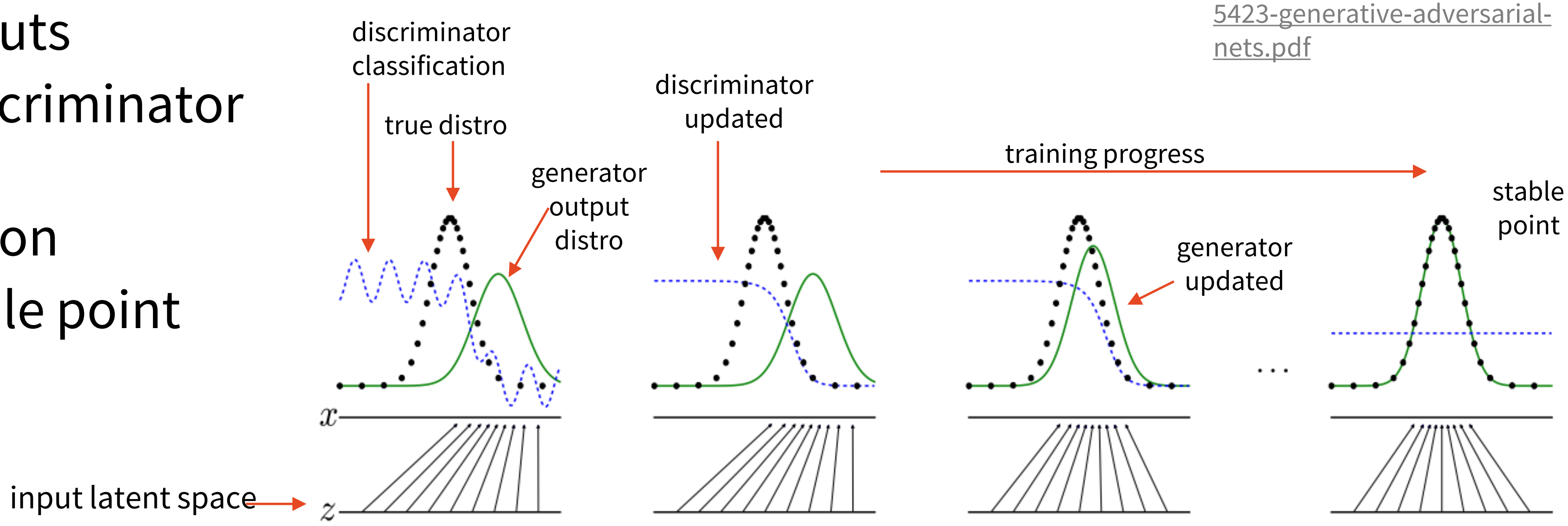
https://nlpnewsletter.substack.com/p/instruction-tuning-vol-1

(Rafailov et al., 2023) http://arxiv.org/abs/2305.18290

Adversarial Learning / Generative Adversarial Nets

- Training generative models to generate **believable** outputs
  - to do so, they necessarily get a better grasp on the distribution
- Getting loss from a 2nd model:
  - **discriminator $\boldsymbol{D}$** – "adversary" classifying real vs. generated samples
  - **generator $\boldsymbol{G}$** – trained to fool the discriminator
    - the best chance to fool the discriminator is to generate likely outputs
- Training iteratively (EM style)
  - generate some outputs
  - classify + update discriminator
  - update generator based on classification
  - this will reach a stable point

(Goodfellow et al, 2014)
http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

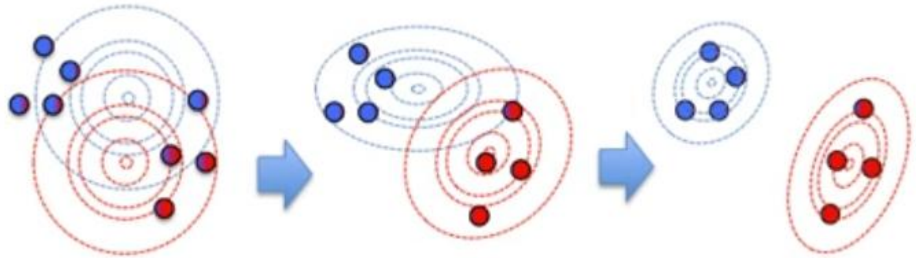


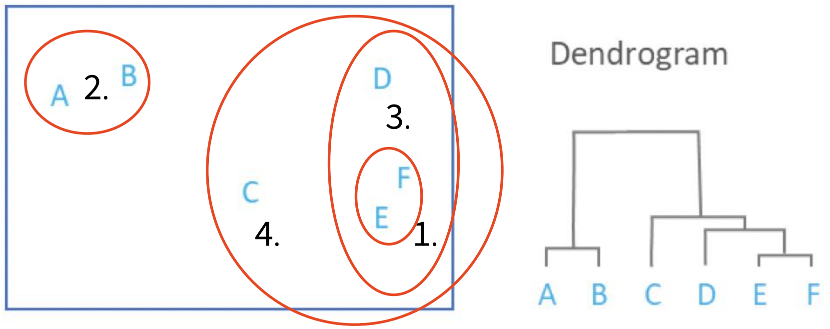
Clustering

https://en.wikipedia.org/wiki/K-means_clustering
https://www.displayr.com/what-is-hierarchical-clustering/
https://towardsdatascience.com/gaussian-mixture-models-d13a5e915c8e

- Unsupervised, finding similarities in data
- basic algorithms
  - **k-means**: assign into k clusters randomly, iterate:
    - compute means (centroids)
    - reassign to nearest centroid
  - **Gaussian mixture**: similar, but soft & variance
    - clusters = multivariate Gaussian distributions
    - estimating probabilities of belonging to each cluster
    - cluster mean/variance based on data weighted by probabilities
  - **hierarchical** (bottom up):
    start with one cluster per instance, iterate:
    - merge 2 closest clusters
    - end when you have k clusters / distance is too big
  - hierarchical top-down (reversed ⬏)
- distance metrics & features decide what ends up together

https://www.youtube.com/watch?v=9YA2t78Ha68





Summary

- Supervised training
  - cost function
  - stochastic **gradient descent** – minibatches
  - backpropagation
  - **learning rate** tricks – optimizers (Adam), schedulers
  - regularization: dropout, multi-task training
- Self-supervised learning (~kinda unsupervised)
  - autoencoders, denoising, variational autoencoders
  - (masked) language models
- PLMs/LLMs: **pretraining & finetuning, prompting, instruction tuning**
- Reinforcement learning (more to come later)
- Unsupervised: GANs, clustering

Thanks

Contact us:

https://ufaldsg.slack.com/
{odusek,hudecek,kasner}@ufal.mff.cuni.cz
Zoom/Skype/Troja

Labs in 10 mins

Next Thu 10:40

Get the slides here:

http://ufal.cz/npfl099

References/Further:

Goodfellow et al. (2016): Deep Learning, MIT Press (http://www.deeplearningbook.org)
Kim et al. (2018): Tutorial on Deep Latent Variable Models of Natural Language (http://arxiv.org/abs/1812.06834)
Milan Straka's Deep Learning slides: http://ufal.mff.cuni.cz/courses/npfl114/1819-summer

Neural nets tutorials:

- https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/#0
- https://minitorch.github.io/index.html
- https://objax.readthedocs.io/en/latest/

GPT in Excel: https://www.youtube.com/watch?v=FyeN5tXMnJ8