Fitting neural networks, tricks of the trade and MLOps

Ole Winther

Bioinformatics Centre, Department of Biology
University of Copenhagen (UCph)

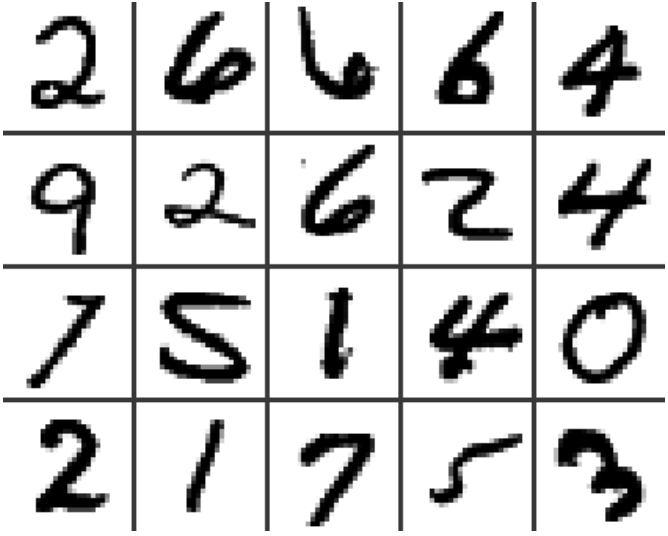
Dept for Applied Mathematics and Computer Science
Technical University of Denmark (DTU)

November 5, 2025

Objectives of week 2 lectures

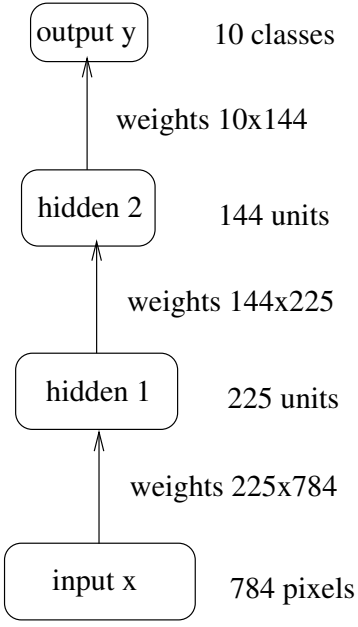
- Part 0 Recap of the feedforward architecture back-propagation
- Part 1 Tricks of the trade
- Part 2 MLOps (=machine learning operations)

Example: MNIST handwritten digits



Train a network to classify 28×28 images.
Data: 60000 input images $\mathbf{x}_n$ and labels $\mathbf{t}_n$, $n = 1, \ldots, 60000$.
Example model gives around 1.2% test error.

Example Network



$$\mathbf{y} = \text{softmax}(\mathbf{W}^{(3)}\mathbf{h}^{(2)} + \mathbf{b}^{(3)})$$

$$\mathbf{h}^{(2)} = \text{relu}(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)})$$

$$\mathbf{h}^{(1)} = \text{relu}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

$$\text{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

$$\text{relu}(z) = \max(0, z)$$

Training criterion

- Find parameters

$$\boldsymbol{\theta} = \{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}_{l=1,\ldots,3}$$

that minimize expected negative log-likelihood:

$$C = -\sum_{n=1}^{N} \log P(\mathbf{t}_n | \mathbf{x}_n, \boldsymbol{\theta}).$$

- Learning becomes optimization.

Classification - one hot encoding and cross-entropy

- MNIST, output labels: $0, 1, \ldots, 9$.
- Convenient to use a sparse one hot encoding:

$$
\begin{aligned}
0 &\to \mathbf{t} = (1,0,0,0,0,0,0,0,0,0)^T \\
1 &\to \mathbf{t} = (0,1,0,0,0,0,0,0,0,0)^T \\
2 &\to \mathbf{t} = (0,0,1,0,0,0,0,0,0,0)^T \\
&\ldots. \\
9 &\to \mathbf{t} = (0,0,0,0,0,0,0,0,0,1)^T
\end{aligned}
$$

- Output

$$\mathbf{y} = \text{softmax}(\mathbf{W}^{(3)}\mathbf{h}^{(2)} + \mathbf{b}^{(3)})$$

  interpreted as class(-conditional) probability.
- Cross-entropy cost - sum over data and label

$$C = -\sum_{n=1}^{N}\sum_{k=1}^{K} y_{nk} \log t_{nk}$$

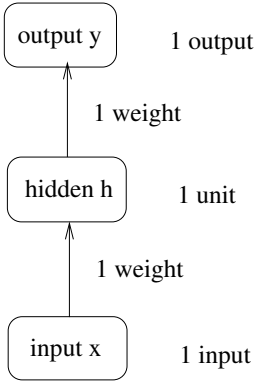
Gradient descent

- Simple algorithm for minimizing the training criterion C.
- Gradient $\mathbf{g} = \nabla_{\boldsymbol{\theta}} C(\theta) = \begin{pmatrix} \frac{\partial C}{\partial \theta_1} \\ \vdots \\ \frac{\partial C}{\partial \theta_m} \end{pmatrix}$
- $\dim(\theta) = m$.
- Iterate $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k \mathbf{g}_k$
- Notation: iteration k, stepsize (or learning rate) η_k

Gradient descent

- Simple algorithm for minimizing the training criterion C.
- Gradient $\mathbf{g} = \nabla_{\boldsymbol{\theta}} C(\theta) = \begin{pmatrix} \frac{\partial C}{\partial \theta_1} \\ \vdots \\ \frac{\partial C}{\partial \theta_m} \end{pmatrix}$
- $\dim(\theta) = m$.
- Iterate $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k \mathbf{g}_k$
- Notation: iteration k, stepsize (or learning rate) η_k
- We find the $\mathbf{g}$ efficiently with backpropation

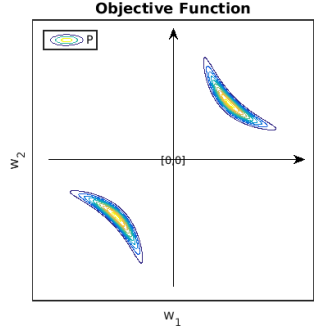
Tiny Example



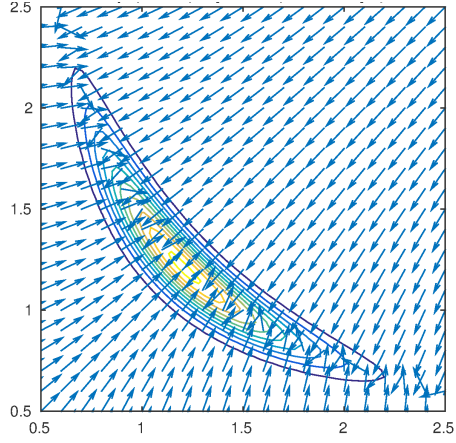
- LIkelihood $p(t|x) = \mathcal{N}(t|y, 1)$
- Model

$$y = w_2 h$$
$$h = w_1 x$$

- "Data set": $\{x = 1, t = 1.5\}$
- Add some weight decay.
- $C = (w_1 w_2 - 1.5)^2 + 0.04(w_1^2 + w_2^2)$

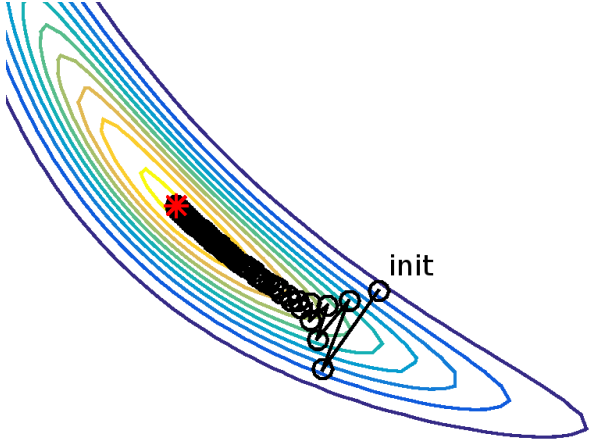


Gradient $\mathbf{g} = \nabla_\theta C(\theta) = \begin{pmatrix} \frac{\partial C}{\partial \theta_1} \\ \vdots \\ \frac{\partial C}{\partial \theta_m} \end{pmatrix}$



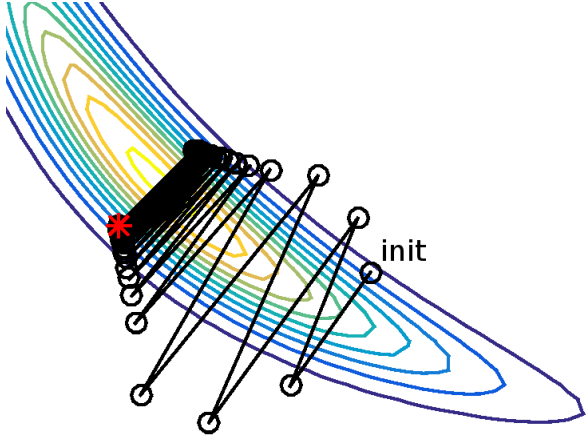
Gradient descent, $\eta_k = 0.25$ ($\rightarrow$ too slow)

$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k \mathbf{g}_k$, iteration k, stepsize (or learning rate) η_k



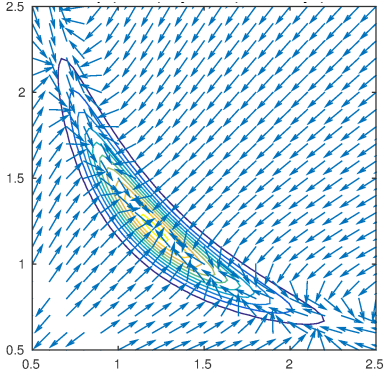
Gradient descent, $\eta_k = 0.35$ ($\rightarrow$ oscillates)

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k \mathbf{g}_k$$



Newton's method, too complex

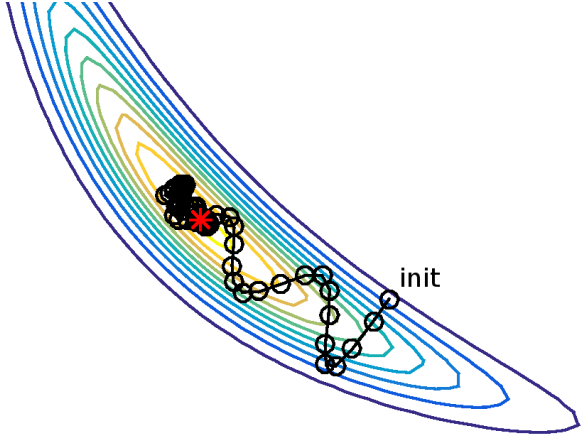
$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mathbf{H}_k^{-1}\mathbf{g}_k, \quad \mathbf{H} = \begin{pmatrix} \frac{\partial^2 C}{\partial\theta_1\partial\theta_1} & \cdots & \frac{\partial^2 C}{\partial\theta_1\partial\theta_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 C}{\partial\theta_m\partial\theta_1} & \cdots & \frac{\partial^2 C}{\partial\theta_m\partial\theta_m} \end{pmatrix}$$



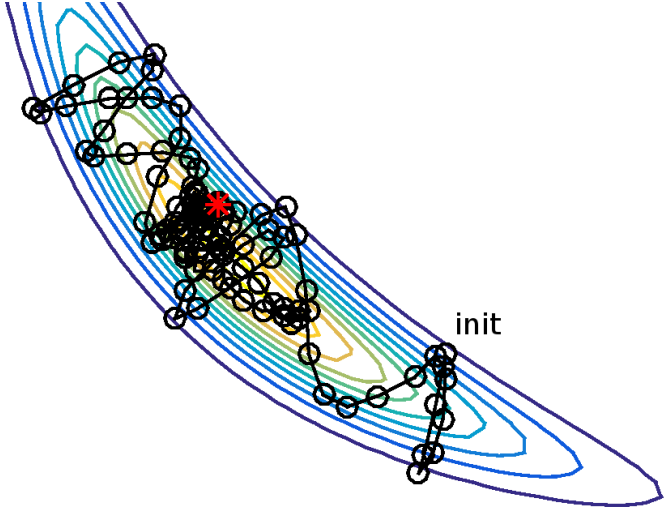
- Less oscillations.
- Points to the wrong direction in places (solvable).
- Computational complexity: $(\#\text{params})^3 = m^3$ (prohibitive).
- There are approximations, but not very popular.

Momentum method (Polyak, 1964)

$$\mathbf{m}_{k+1} = \alpha \mathbf{m}_k - \eta_k \mathbf{g}_k$$
$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \mathbf{m}_{k+1}$$



Momentum method with noisy gradient

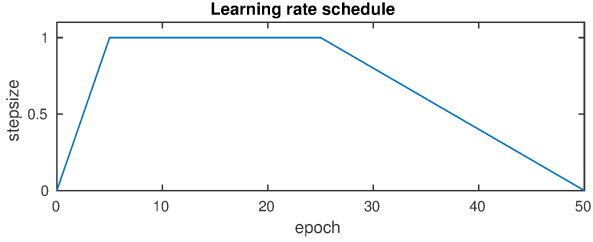


Mini-batch training

- No need to have an accurate estimate of $\mathbf{g}$.
- Use only a small batch of training data at once.
- Leads into many updates per epoch (=seeing data once).
- E.g. 600 updates with 100 samples per epoch in MNIST.

Mini-batch training

- No need to have an accurate estimate of $\mathbf{g}$.
- Use only a small batch of training data at once.
- Leads into many updates per epoch (=seeing data once).
- E.g. 600 updates with 100 samples per epoch in MNIST.
- Important to anneal stepsize η_k towards the end, e.g.



- Adaptation of η_k possible (Adam, Adagrad, Adadelta).

Part 1.1

Tricks of the trade
Faster convergence

Adam algorithm by Kingma and Ba

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Adam algorithm by Kingma and Ba

- $C(\theta)$ is the cost function we want to optimize
- $f_t(\theta)$ stochastic estimate of $C(\theta)$, e.g. training cost on mini-batch at step t.
- Momentum:

$$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1)\nabla_\theta f(\theta_{t-1})$$

- Correct bias from $m_0 = 0$.
- Normalize stepsize with $\sqrt{(\text{estimate of gradient})^2}$ so that

$$|\Delta\theta| \lesssim \alpha$$

- Never take step larger than α
- Take roughly equal steps in all directions, but $\beta_2 > \beta_1$ and $\epsilon > 0$ ensures that we will stop at minimum.
- Demo

Smarter initialization

- Recall the model

$$\begin{aligned}\mathbf{y} &= \text{softmax}(\mathbf{W}^{(3)}\mathbf{h}^{(2)}+\mathbf{b}^{(3)})\\ \mathbf{h}^{(2)} &= \text{relu}(\mathbf{W}^{(2)}\mathbf{h}^{(1)}+\mathbf{b}^{(2)})\\ \mathbf{h}^{(1)} &= \text{relu}(\mathbf{W}^{(1)}\mathbf{x}+\mathbf{b}^{(1)})\end{aligned}$$

- Ignoring softmax and biases, we can write

$$y_i = \sum_{j,k,l} \mathbf{1}\left(h_j^{(2)} > 0\right)\mathbf{1}\left(h_k^{(1)} > 0\right) W_{ij}^{(3)} W_{jk}^{(2)} W_{kl}^{(1)} x_l$$

- Exponential growth/decay of forward signals!

Exponential growth/decay backward

- Given indicators $\mathbf{1}(\cdot)$, model is linear

$$y_i = \sum_{j,k,l} \mathbf{1}\left(h_j^{(2)} > 0\right) \mathbf{1}\left(h_k^{(1)} > 0\right) W_{ij}^{(3)} W_{jk}^{(2)} W_{kl}^{(1)} x_l$$

$$\frac{\partial y_i}{\partial x_l} = \sum_{j,k} \mathbf{1}\left(h_j^{(2)} > 0\right) \mathbf{1}\left(h_k^{(1)} > 0\right) W_{ij}^{(3)} W_{jk}^{(2)} W_{kl}^{(1)}$$

- Exponential growth/decay of gradient, too!
- $\Rightarrow$ Scale of initialization important.

Initialization

- Initialize weights $W_{ij} \sim \sqrt{\frac{4}{n_i+n_j}}\mathcal{N}(0,1)$
- where size of $\mathbf{W}$ is $n_i \times n_j$.

Initialization

- Initialize weights $W_{ij} \sim \sqrt{\frac{4}{n_i+n_j}}\mathcal{N}(0,1)$
- where size of $\mathbf{W}$ is $n_i \times n_j$.
- Outline of derivation:
- Assume independent signals.

Initialization

- Initialize weights $W_{ij} \sim \sqrt{\frac{4}{n_i+n_j}}\mathcal{N}(0,1)$
- where size of $\mathbf{W}$ is $n_i \times n_j$.
- Outline of derivation:
- Assume independent signals.
- Retain variance forward: $\sqrt{\frac{2}{n_j}}$
- Retain variance backward: $\sqrt{\frac{2}{n_i}}$
- (2 is from relu indicators being on half the time.)

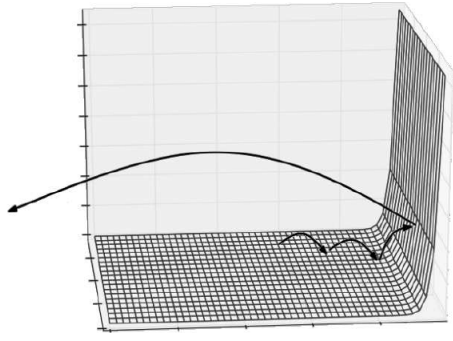
Initialization

- Initialize weights $W_{ij} \sim \sqrt{\frac{4}{n_i+n_j}}\mathcal{N}(0,1)$
- where size of $\mathbf{W}$ is $n_i \times n_j$.
- Outline of derivation:
- Assume independent signals.
- Retain variance forward: $\sqrt{\frac{2}{n_j}}$
- Retain variance backward: $\sqrt{\frac{2}{n_i}}$
- (2 is from relu indicators being on half the time.)
- Strike a balance between the two (Glorot and Bengio, 2010).

Initialization

- Initialize weights $W_{ij} \sim \sqrt{\frac{4}{n_i+n_j}}\mathcal{N}(0,1)$
- where size of $\mathbf{W}$ is $n_i \times n_j$.
- Outline of derivation:
- Assume independent signals.
- Retain variance forward: $\sqrt{\frac{2}{n_j}}$
- Retain variance backward: $\sqrt{\frac{2}{n_i}}$
- (2 is from relu indicators being on half the time.)
- Strike a balance between the two (Glorot and Bengio, 2010).
- (Other ideas, relevant for RNNs: sparse initialization (Martens, 2010), orthogonal initialization (Saxe et al., 2014))

Gradient clipping



- Highly nonlinear model: Gradient update can catapult parameters very far.
- Heuristic: Clip the magnitude of the gradient.
- Figure from (Pascanu, 2014)

Batch normalization (Ioffe and Szegedy, 2015)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

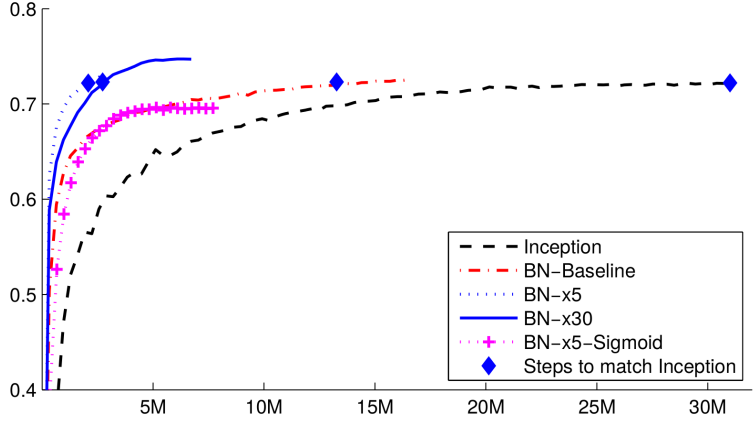
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch normalization (Ioffe and Szegedy, 2015)

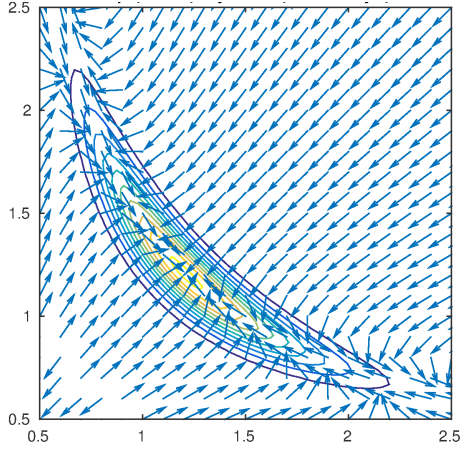
- Improves learning (BN-baseline vs. Inception)
- Allows bigger learning rates (5x and 30x)



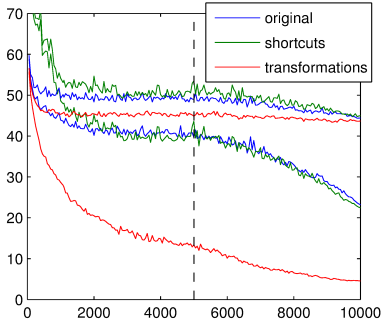
- Other variants exist, e.g. layer normalization

Why does it work? Recall Newton's

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mathbf{H}_k^{-1}\mathbf{g}_k, \quad \mathbf{H} = \begin{pmatrix} \frac{\partial^2 C}{\partial\theta_1\partial\theta_1} & \cdots & \frac{\partial^2 C}{\partial\theta_1\partial\theta_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 C}{\partial\theta_n\partial\theta_1} & \cdots & \frac{\partial^2 C}{\partial\theta_n\partial\theta_n} \end{pmatrix}$$

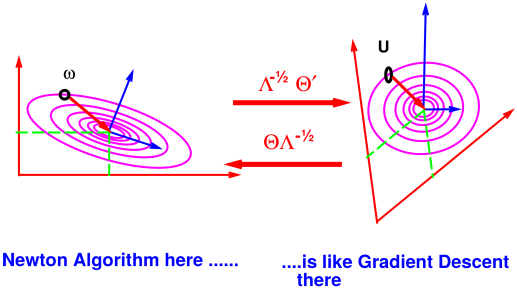


Why does it work? (Raiko et al., 2012)



- Transformations do not change the model, but the optimisation
- Hessian $\mathbf{H}$ is closer to a diagonal
- Traditional gradient is thus closer to Newton's and parameter updates are more independent

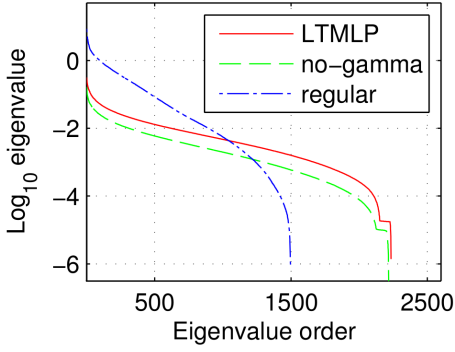
Eigenvalues of the Hessian $\mathbf{H}$ (LeCun et al., 1998)



- Eigenvectors corresponds to (update) directions
- In Newton's method, each direction has its own learning rate: inverse of the eigenvalue
- Some eigenvalues can be negative:
  Newton's method points the wrong way

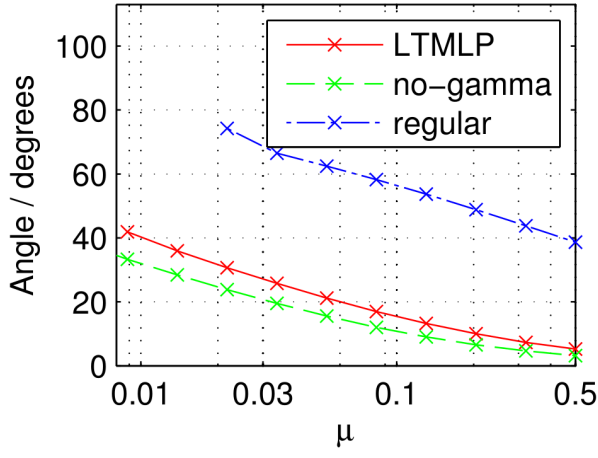
Why does it work? (Vatanen et al., 2013)

- Analysis of eigenvalues of Hessian in 2600-parameter model
- Curvature is much more even with transformations



Why does it work? (Vatanen et al., 2013)

Angle between gradient and second order update:
$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - (\mathbf{H}_k + \mu\mathbf{I})^{-1}\mathbf{g}_k$,



Part 2.2

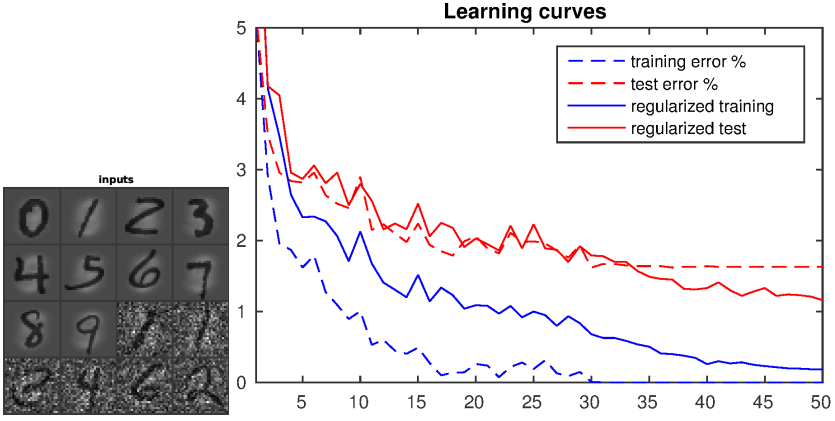
Tricks of the trade

Better generalization through regularization

Goal of Regularization

- Neural networks are very powerful (universal appr.).
- Easy to perform great on the training set (overfitting).
- **Regularization** improves generalization to new data at the expense of increased training error.
- Use held-out validation data to choose hyperparameters (e.g. regularization strength).
- Use held-out test data to evaluate performance.

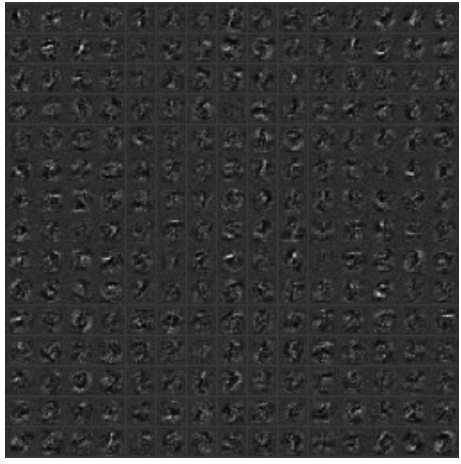
Example



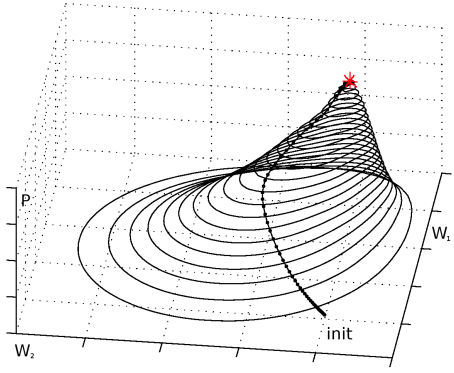
Without regularization training error goes to zero and learning stops.
With noise regularization, test error keeps dropping.

Expressivity demo: Training first layer only

No regularization, training $\mathbf{W}^{(1)}$ and $\mathbf{b}^{(1)}$ only.
0.2% error on training set, 2% error on test set.



What is overfitting?



Posterior probability mass matters
Center of gravity $\neq$ maximum

Probability theory states how we should make predictions (of $\mathbf{y}_{\text{test}}$) using a model with unknowns $\boldsymbol{\theta}$ and data $\mathbf{X} = \{\mathbf{x}_{\text{train}}, \mathbf{y}_{\text{train}}, \mathbf{x}_{\text{test}}\}$:

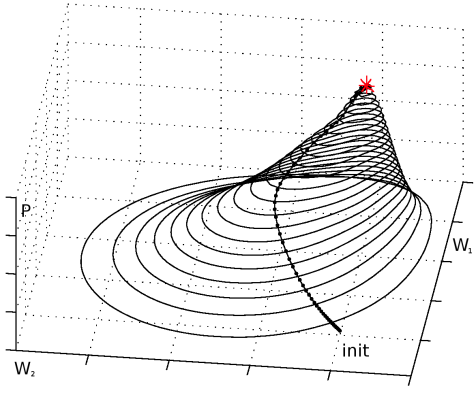
$$P(\mathbf{y}_{\text{test}} \mid \mathbf{X}) = \int P(\mathbf{y}_{\text{test}}, \boldsymbol{\theta} \mid \mathbf{X}) \mathrm{d}\boldsymbol{\theta}$$
$$= \int P(\mathbf{y}_{\text{test}} \mid \boldsymbol{\theta}, \mathbf{X}) P(\boldsymbol{\theta} \mid \mathbf{X}) \mathrm{d}\boldsymbol{\theta}.$$

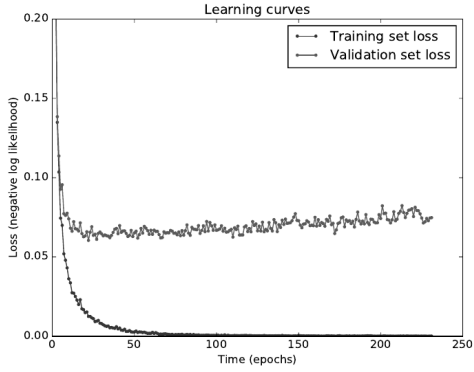
Probability of observing $\mathbf{y}_{\text{test}}$ can be acquired by summing or integrating over all different explanations $\boldsymbol{\theta}$. The term $P(\mathbf{y}_{\text{test}} | \boldsymbol{\theta}, \mathbf{X})$ is the probability of $\mathbf{y}_{\text{test}}$ given a particular explanation $\boldsymbol{\theta}$ and it is weighted with the probability of the explanation $P(\boldsymbol{\theta} | \mathbf{X})$. However, such computation is intractable. If we want to choose a single $\boldsymbol{\theta}$ to represent all the probability mass, it is better not to overfit to the highest probability peak, but to find a good representative of the mass.

Regularization methods

- Limited size of network
- Early stopping
- Weight decay
- Data augmentation
- Injecting noise
- Parameter sharing (e.g. convolutional)
- Sparse representations
- Ensemble methods
- Auxiliary tasks (e.g. unsupervised)
- Probabilistic treatment (e.g. variational methods)
- Adversarial training, . . .

Early stopping



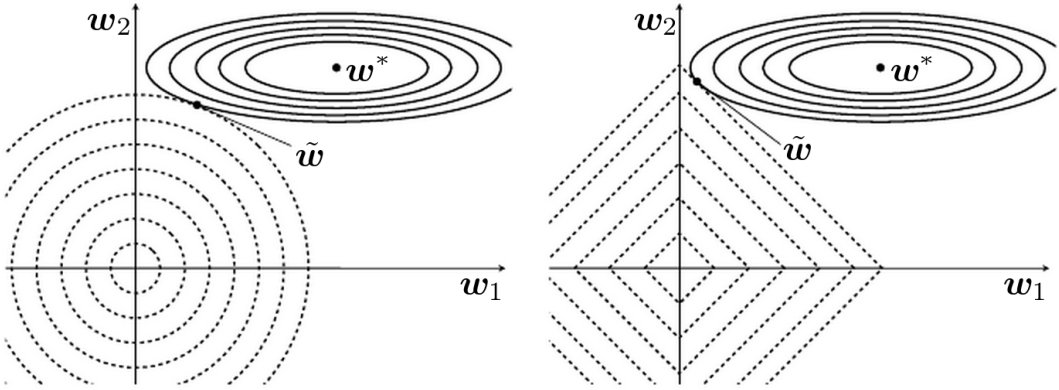


- Monitor validation performance during training
- Stop when it starts to deteriorate
- With other regularization, it might never start
- Keeps solution close to the initialization

Weight decay (Tikhonov, 1943)

- Add a penalty term to the training cost $C = \cdots + \Omega(\boldsymbol{\theta})$
  Note: only a function of parameters $\boldsymbol{\theta}$, not data.
- L^2 regularization: $\Omega(\boldsymbol{\theta}) = \frac{\lambda}{2}\|\boldsymbol{\theta}\|^2$
  hyperparameter λ for strength.
  Gradient: $\frac{\partial \Omega(\boldsymbol{\theta})}{\partial \theta_i} = \lambda \theta_i$.
- L^1 regularization: $\Omega(\boldsymbol{\theta}) = \lambda/2\|\boldsymbol{\theta}\|_1$
  Gradient: $\frac{\partial \Omega(\boldsymbol{\theta})}{\partial \theta_i} = \lambda \text{sign}(\theta_i)$.
  Induces sparsity: Often many params become zero.
- Max-norm: Constrain row vectors $\mathbf{w}_i$ of weight matrices to $\|\mathbf{w}_i\|^2 \leq c$.

Weight decay



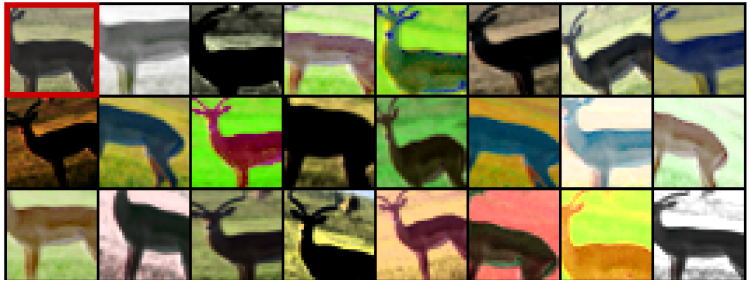
- L2 (left) and L1 (right).
- w^* unregularised solution, $\tilde{w}$ regularised solution.
- Note: L1 pushes small parameters more towards zero - sparsity!

How to set hyperparameter λ?

- In general, difficult to set strength of regularization
- Split data into training, validation, and test sets
- Choose a number of settings λ, train separately
- Use validation performance to pick the best λ
- (Retrain using both training and validation sets)
- Evaluate final performance on test data
- Ongoing work on adjusting hyperparameters on the fly (Luketina et al., 2016)

Data augmentation

Image from (Dosovitskiy et al., 2014)



Augmented data by image-specific transformations.
E.g. cropping just 2 pixels gets you 9 times the data! Infinite MNIST:
http://leon.bottou.org/projects/infimnist

Parameter sharing

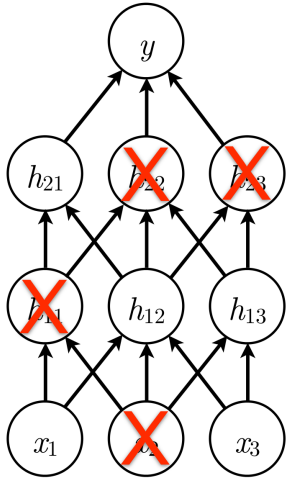
- Force sets of parameters to be equal
- Reduces the number of (unique) parameters
- Should reflect properties of data (inductive bias)
- Important in convolutional networks (CNNs), Transformers and (unrolled) recurrent neural networks (RNNs)

Ensemble methods

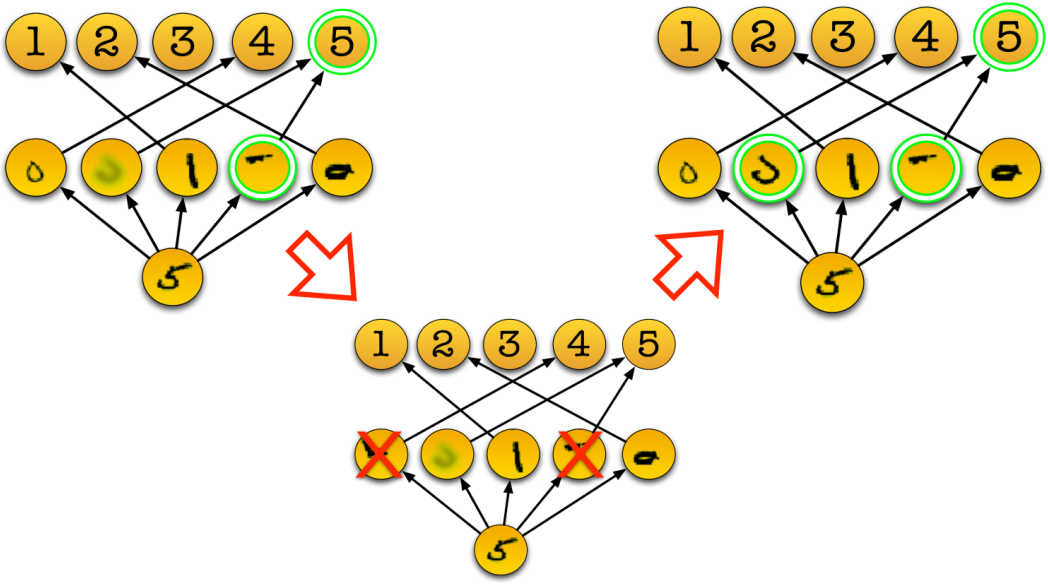
- Train several models and take average of their outputs
  Instead of one point representing $P(\boldsymbol{\theta}|\mathbf{X})$, use several
- Also known as *bagging* or *model averaging*
- It helps to make individual models different by
  - varying models or algorithms
  - varying hyperparameters
  - varying data (dropping examples or dimensions)
  - varying random seed
- It is possible to train a single final model to mimick the performance of the ensemble, for test-time computational efficiency (Hinton et al., 2015)

Dropout (Srivastava et al., 2014)

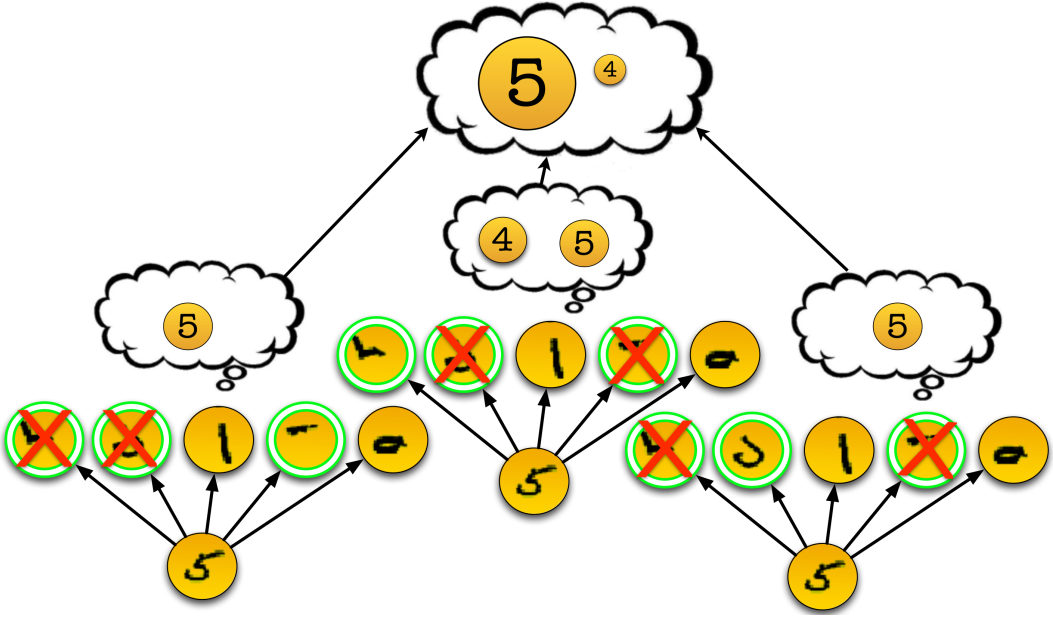
- Each time we present data example $\mathbf{x}$, randomly delete each hidden node with 0.5 probability
- Can be seen as *injecting noise* or as *ensemble*:
  - Multiplicative binary noise
  - Training an ensemble of $2^{|h|}$ networks with weight sharing
- At test time, use all nodes but divide weights by 2



Dropout training

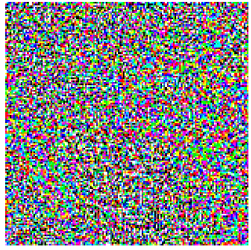


Dropout as bagging

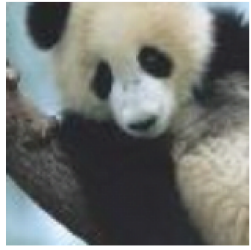


Adversarial training (Szegedy et al., 2014)

$+ .007 \times$



$=$



$\boldsymbol{x}$

$y =$ “panda”

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

$\boldsymbol{x} +$
$\epsilon \, \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
“gibbon”

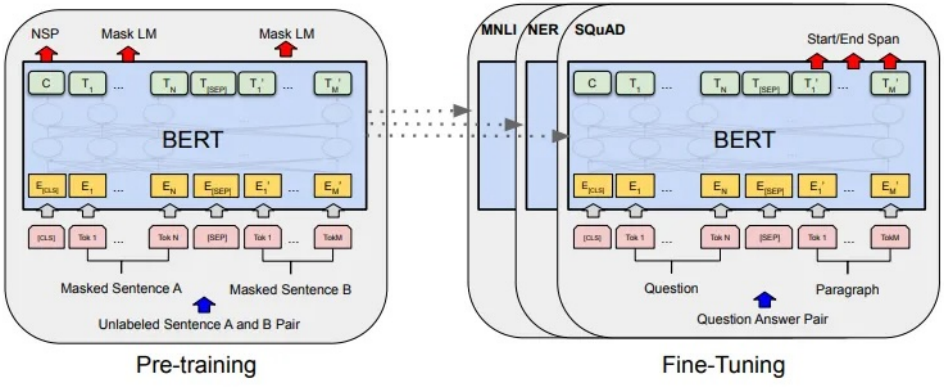
- Search for an input $\mathbf{x}'$ near a datapoint $\mathbf{x}$ that would have very different output $\mathbf{y}'$ from $\mathbf{y}$
- Adversaries can be found surprisingly close!
- Miyato et al. (2016) build a very effective regulariser

Foundation models - Pretrain + finetune paradigm

- Multi-task learning: *Parameter sharing* between 2+ tasks

Foundation models - Pretrain + finetune paradigm

- Multi-task learning: *Parameter sharing* between 2+ tasks
- Foundation models may be viewed as multi-task version that:
  - Train self-supervised task with plenty of data, e.g. next token prediction from large quantities of scraped text
  - Fine-tune on small dataset with labeled data to solve specific task or on curated data get alignment with desired behavior



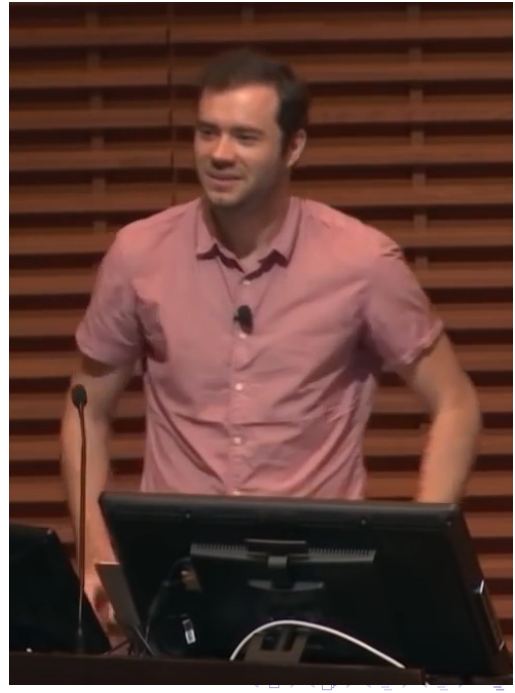
Devlin et al, BERT, 2018

Part 2.3

Tricks of the trade
A recipe for training neural networks

Andrej Karpathy blog post

- Andrej Karpathy blog post
- Part A - Become one with the data
- Part B - Simple baselines
- Part C - Overfit, regularize, tune and tune some more
- Quiz - for you to take



Part A

Not knowing your data is a recipe for failure

A Recipe for Training Neural Networks

- Large gap between
  here is how a convolutional layer works
  and
  our convnet achieves state of the art results

A Recipe for Training Neural Networks

- Large gap between
  here is how a convolutional layer works
  and
  our convnet achieves state of the art results
- Not talk about common errors but rather
- about how one can avoid making these errors altogether (or fix them very fast)

A Recipe for Training Neural Networks

- Large gap between
  here is how a convolutional layer works
  and
  our convnet achieves state of the art results
- Not talk about common errors but rather
- about how one can avoid making these errors altogether (or fix them very fast)
- Neural net training is a leaky abstraction

```
>>> your_data = # plug your awesome dataset here
>>> model = SuperCrossValidator(SuperDuper.fit, your_data, ResNet50, SGDOptimizer)
# conquer world here
```

- If you insist on using the technology without understanding how it works you are likely to fail.

A Recipe for Training Neural Networks

- Large gap between
  here is how a convolutional layer works
  and
  our convnet achieves state of the art results
- Not talk about common errors but rather
- about how one can avoid making these errors altogether (or fix them very fast)
- Neural net training is a leaky abstraction

```
>>> your_data = # plug your awesome dataset here
>>> model = SuperCrossValidator(SuperDuper.fit, your_data, ResNet50, SGDOptimizer)
# conquer world here
```

- If you insist on using the technology without understanding how it works you are likely to fail.
- Neural net training fails silently
- a “fast and furious” approach to training neural networks does not work (in 2020)

Become one with the data

- Spend copious amount of time (measured in units of hours) scanning through thousands of examples,
- understanding their distribution and looking for patterns.
- Might find: duplicate examples, corrupted images / labels.
- Look for data imbalances and biases.

Become one with the data

- Spend copious amount of time (measured in units of hours) scanning through thousands of examples,
- understanding their distribution and looking for patterns.
- Might find: duplicate examples, corrupted images / labels.
- Look for data imbalances and biases.
- Think about your process for classifying the data

Become one with the data

- Spend copious amount of time (measured in units of hours) scanning through thousands of examples,
- understanding their distribution and looking for patterns.
- Might find: duplicate examples, corrupted images / labels.
- Look for data imbalances and biases.
- Think about your process for classifying the data
- Are very local features enough or do we need global context?

Become one with the data

- Spend copious amount of time (measured in units of hours) scanning through thousands of examples,
- understanding their distribution and looking for patterns.
- Might find: duplicate examples, corrupted images / labels.
- Look for data imbalances and biases.
- Think about your process for classifying the data
- Are very local features enough or do we need global context?
- How much variation is there and what form does it take?

Become one with the data

- Spend copious amount of time (measured in units of hours) scanning through thousands of examples,
- understanding their distribution and looking for patterns.
- Might find: duplicate examples, corrupted images / labels.
- Look for data imbalances and biases.
- Think about your process for classifying the data
- Are very local features enough or do we need global context?
- How much variation is there and what form does it take?
- What variation is spurious and could be preprocessed out?

Become one with the data

- Spend copious amount of time (measured in units of hours) scanning through thousands of examples,
- understanding their distribution and looking for patterns.
- Might find: duplicate examples, corrupted images / labels.
- Look for data imbalances and biases.
- Think about your process for classifying the data
- Are very local features enough or do we need global context?
- How much variation is there and what form does it take?
- What variation is spurious and could be preprocessed out?
- Does spatial position matter or do we want to average pool it out?

Become one with the data

- Spend copious amount of time (measured in units of hours) scanning through thousands of examples,
- understanding their distribution and looking for patterns.
- Might find: duplicate examples, corrupted images / labels.
- Look for data imbalances and biases.
- Think about your process for classifying the data
- Are very local features enough or do we need global context?
- How much variation is there and what form does it take?
- What variation is spurious and could be preprocessed out?
- Does spatial position matter or do we want to average pool it out?
- How much does detail matter and how far could we afford to downsample the images?

Become one with the data

- Spend copious amount of time (measured in units of hours) scanning through thousands of examples,
- understanding their distribution and looking for patterns.
- Might find: duplicate examples, corrupted images / labels.
- Look for data imbalances and biases.
- Think about your process for classifying the data
- Are very local features enough or do we need global context?
- How much variation is there and what form does it take?
- What variation is spurious and could be preprocessed out?
- Does spatial position matter or do we want to average pool it out?
- How much does detail matter and how far could we afford to downsample the images?
- How noisy are the labels?

Become one with the data - now using the model

- When you have a qualitative sense then write some simple code to search/filter/sort by, e.g.
- type of label, size of annotations, number of annotations, etc. and
- visualize their distributions and the outliers along any axis.
- The outliers especially almost always uncover some bugs in data quality or preprocessing.

Become one with the data - now using the model

- When you have a qualitative sense then write some simple code to search/filter/sort by, e.g.
- type of label, size of annotations, number of annotations, etc. and
- visualize their distributions and the outliers along any axis.
- The outliers especially almost always uncover some bugs in data quality or preprocessing.
- Eventually you can use trained model as a compressed/compiled version of your dataset
- Look at network (mis)predictions and understand where they might be coming from
- If model is not consistent w data then something is off

Part B
Simple baselines

Set up the end-to-end training/evaluation skeleton + get dumb baselines

- Use simple model, linear model or tiny conv net
- Things we do: train it, visualize the losses, accuracy, model predictions, and perform ablations

Set up the end-to-end training/evaluation skeleton + get dumb baselines

- Use simple model, linear model or tiny conv net
- Things we do: train it, visualize the losses, accuracy, model predictions, and perform ablations
- Tips and tricks:
  - fix random seed.
  - simplify. No regularization, data augmentation, etc.
  - add significant digits to your eval. Loss on entire validation set
  - verify loss at init. E.g cross entropy is $\log C$, $C =$ number of classes
  - for MNIST $C = 10$ and $\log C = 2.303$.
  - init well. E.g. Glorot.
  - human baseline.

Set up the end-to-end training/evaluation skeleton + get dumb baselines

- Tips and tricks continued:
  - input-indepent baseline. Set all inputs to zero and train.
  - overfit one batch.
  - verify decreasing training loss. Increase model capacity little by little.
  - visualize data just before the net
  - visualize prediction dynamics. Prediction on fixed test batch.
  - use backprop to chart dependencies. Gradients give you information about what depends on what in your network, which can be useful for debugging. E.g. set loss to output for ith example in mini-batch and calculate gradient with respect to all inputs X in the mini-batch.
  - generalize a special case. E.g. start implementation with functions with loops and later vectorize

Use backprop to chart dependencies - details

1. Create a multi-batch input (x = torch.rand([4, 3, 224, 224])
2. Set your input to be differentiable (x.requires_grad = True)
3. Set your model into evaluation mode (model.eval()) to avoid batch norm
4. Run a forward pass (out = model(x))
5. Define the loss as depending on one of the inputs (for instance: loss = out[2].sum())
6. Run a backprop (loss.backward)
7. Verify that only x[2] has non-null gradients: assert (x.grad[i] == 0.).all() for i != 2 and (x.grad[2] != 0).any()

Credit: Blogpost comment by @Elow2709.

Part C

Overfit, regularize and tune in that order

Scale up while overfitting

- get more data.
- data augment.
- creative augmentation.

Scale up while overfitting

- get more data.
- data augment.
- creative augmentation.
- use pretrained network (for images ImageNet and NLP BERT)
- stick with supervised learning (= don't do unsupervised yourself)

Scale up while overfitting

- get more data.
- data augment.
- creative augmentation.
- use pretrained network (for images ImageNet and NLP BERT)
- stick with supervised learning (= don't do unsupervised yourself)
- smaller input dimensionality.
- smaller model size. E.g. pooling can save a lot parameters.
- decrease the batch size. (= more regularization)

Scale up while overfitting

- get more data.
- data augment.
- creative augmentation.
- use pretrained network (for images ImageNet and NLP BERT)
- stick with supervised learning (= don't do unsupervised yourself)
- smaller input dimensionality.
- smaller model size. E.g. pooling can save a lot parameters.
- decrease the batch size. (= more regularization)
- drop. Add dropout. Use dropout2d (spatial dropout) for ConvNets.
- weight decay. Increase the weight decay penalty.
- early stopping.

Scale up while overfitting

- get more data.
- data augment.
- creative augmentation.
- use pretrained network (for images ImageNet and NLP BERT)
- stick with supervised learning (= don't do unsupervised yourself)
- smaller input dimensionality.
- smaller model size. E.g. pooling can save a lot parameters.
- decrease the batch size. (= more regularization)
- drop. Add dropout. Use dropout2d (spatial dropout) for ConvNets.
- weight decay. Increase the weight decay penalty.
- early stopping.
- try a larger model.
- (visualize first layer weights -conv net)

Tune

- random over grid search. Focus on parameters that make a difference
- hyper-parameter optimization. Bayesian optimization

Squeeze out the juice

- ensembles.
- leave it training

Squeeze out the juice

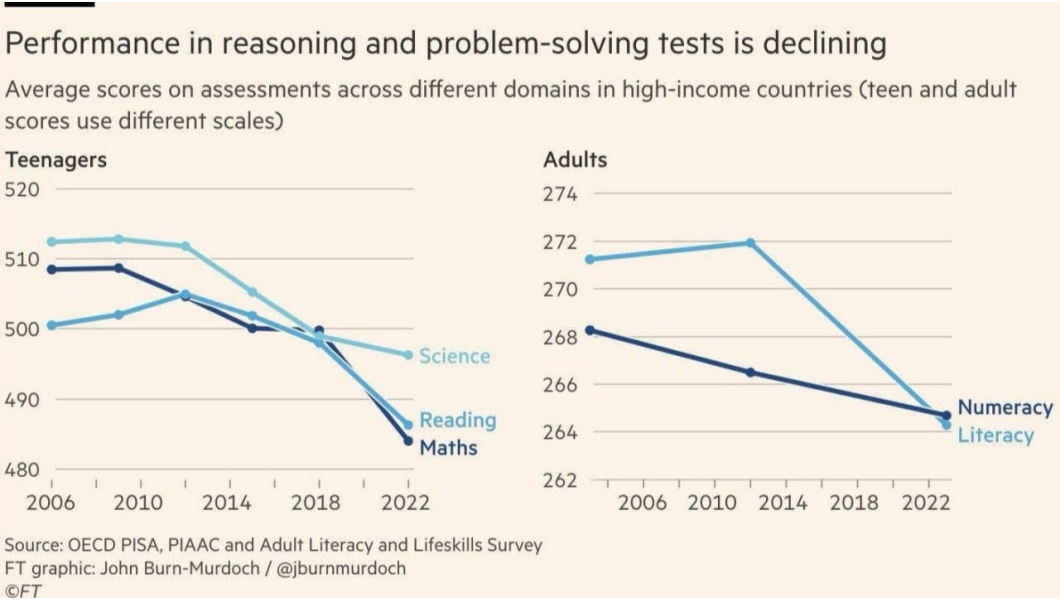
- ensembles.
- leave it training
- If you make until here: You are a master!

Quiz

- Recap: why do we split our data into training, validation and tests sets?
- Data: What does Andrei Karpathy mean when he says that the model is compressed/compiled version of your dataset?
- Data: If we accept that why is it then a good idea to spend time really understanding the data?
- Data: What else might we find when we explore the data?
- Data: List a number of statistics we can compute on the data before starting modelling it.
- Baselines: What components should our training set-up have?
- Baselines: Implement chart dependies in Pytorch for one of the models we have use so far. Does it work as it should?
- Scaling up: How do see that a model is overfitting?
- Scaling up: Why should we overfit in this phase?

Part 3
MLOps

Elephants in the room I - Past peak intelligence



John Burn-Murdoch, Financial Times, 2025

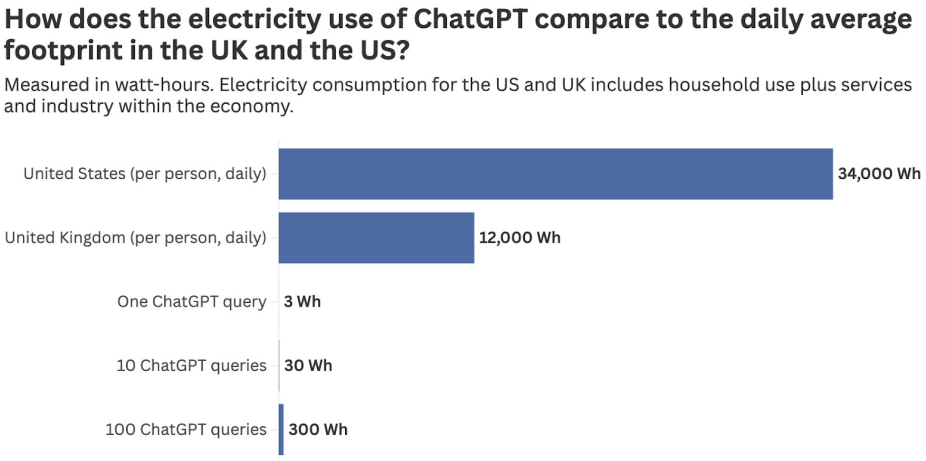
The doctor will see you now

Benjamin Popokh, medical student U Texas Southwestern New Yorker interview:

> On a recent rotation, his professors asked his class to work through a case using A.I. tools such as ChatGPT and OpenEvidence, an increasingly popular medical L.L.M. that provides free access to health-care professionals. Each chatbot correctly diagnosed a blood clot in the lungs. “There was no control group,” Popokh said, meaning that none of the students worked through the case unassisted. For a time, Popokh found himself using A.I. after virtually every patient encounter. “I started to feel dirty presenting my thoughts to attending physicians, knowing they were actually the A.I.’s thoughts,” he told me. One day, as he left the hospital, he had an unsettling realization: he hadn’t thought about a single patient independently that day.

Derek Thomson, End of thinking, 2025

Elephants in the room II - Toasting the planet



Hannah Ritchie, What's the carbon footprint of using ChatGPT?, blogpost, 2025

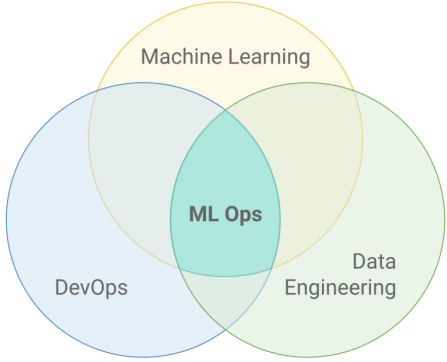
Andy Masley, Using ChatGPT is not bad for the environment, blogpost, 2025 ← water consumption, etc.

Machine learning operations (MLOps)

- Definition: MLOps is a set of practices that combines Machine Learning, DevOps and data engineering. MLOps aims to deploy and maintain ML systems in production reliably and efficiently.

Machine learning operations (MLOps)

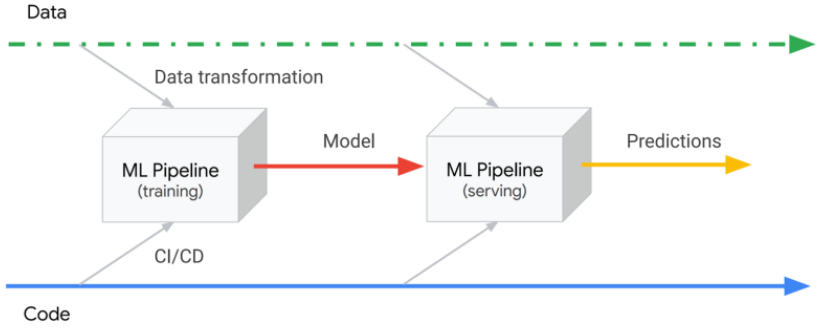
- Definition: MLOps is a set of practices that combines Machine Learning, DevOps and data engineering. MLOps aims to deploy and maintain ML systems in production reliably and efficiently.



- DevOps = DevOps integrates and automates the work of software development (Dev) and IT operations (Ops) Wiki
- Data engineering = Data engineering refers to the building of systems to enable the collection and usage of data Wiki

MLOps continued

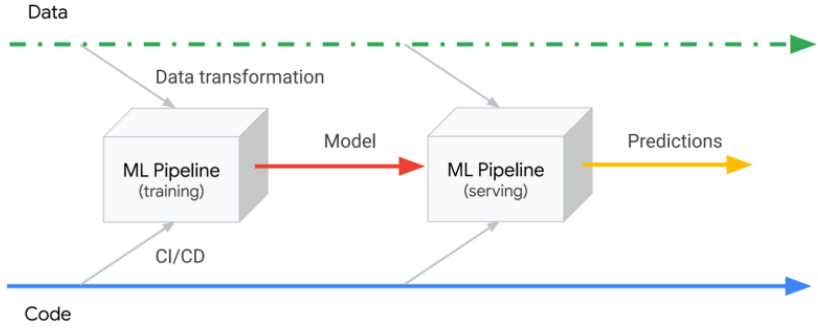
- MLOps so challenging that many companies ultimately never put their data science projects in production



- Continuous integration and continuous deployment (CI/CD) in MLOps context means retraining and redeploying models when new data and better model practices arrive.
- Better pretrained models (open/closed source APIs) $\Rightarrow$ more ML in products, but fewer train their own models.

MLOps continued

- MLOps so challenging that many companies ultimately never put their data science projects in production



- Continuous integration and continuous deployment (CI/CD) in MLOps context means retraining and redeploying models when new data and better model practices arrive.
- Better pretrained models (open/closed source APIs) $\Rightarrow$ more ML in products, but fewer train their own models.
- Learning MLOps practices increases likelihood of success!
- Further reading: blogpost with details and paper with definitions.

Last pieces of advice

- Always plot training and validation metrics in the same plot - underfitting is more common than you would think
- Plot more than one metric, e.g. in classification plot both loss and classification error

Last pieces of advice

- Always plot training and validation metrics in the same plot - underfitting is more common than you would think
- Plot more than one metric, e.g. in classification plot both loss and classification error
- Try to enable GPU

```
[ ] # GPU
    device = 'cuda:0' if torch.cuda.is_available() else 'cpu'
    print('GPU State:', device)
```

Last pieces of advice

- Always plot training and validation metrics in the same plot - underfitting is more common than you would think
- Plot more than one metric, e.g. in classification plot both loss and classification error
- Try to enable GPU

```
[ ]  # GPU
     device = 'cuda:0' if torch.cuda.is_available() else 'cpu'
     print('GPU State:', device)
```

- Much better to stand on the shoulders of someone's GitHub repo than starting a project from scratch
- Reproducing results from paper exactly is hard even with authors' code

Last pieces of advice

- Always plot training and validation metrics in the same plot - underfitting is more common than you would think
- Plot more than one metric, e.g. in classification plot both loss and classification error
- Try to enable GPU

```
[ ] # GPU
    device = 'cuda:0' if torch.cuda.is_available() else 'cpu'
    print('GPU State:', device)
```

- Much better to stand on the shoulders of someone's GitHub repo than starting a project from scratch
- Reproducing results from paper exactly is hard even with authors' code
- Running large models (1B params+) requires securing not only GPU RAM but also intermediate storage
- Getting access to sufficient GPU compute can be tricky - start early looking around.

References - Fitting neural networks

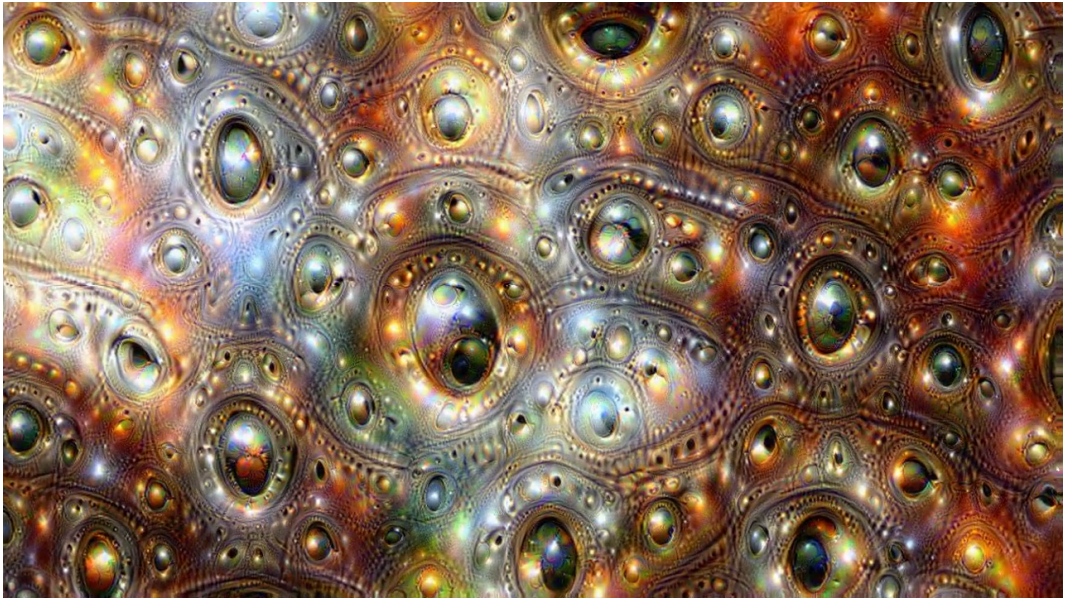
- S. Linnainmaa. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Master's Thesis (in Finnish), Univ. Helsinki, 1970.
- Polyak, B.T. Some methods of speeding up the convergence of iteration methods. USSR Computational Mathematics and Mathematical Physics, 4(5):1–17, 1964.
- Kingma, D. and Ba, J. (2015). ADAM: A method for stochastic optimization. In the International Conference on Learning Representations (ICLR), San Diego. arXiv:1412.6980.
- Dauphin, Pascanu, Gulcehre, Cho, Ganguli, and Bengio. Identifying and attacking the saddle point problem in high dimensional non-convex optimization. In NIPS 2014.
- Choromanska, Henaff, Mathieu, Ben Arous, and LeCun. The Loss Surface of Multilayer Nets. In AISTATS 2015.
- Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." International conference on artificial intelligence and statistics. 2010.
- Martens, J. Deep learning via Hessian-free optimization. In ICML, 2010.
- Saxe, Andrew M., James L. McClelland, and Surya Ganguli. "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks." In ICLR 2014.
- Pascanu, R., "On Recurrent and Deep Neural Networks", PhD thesis, University of Montreal, 2014.
- Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." Proc. of ICML, 2015.
- T. Raiko, H. Valpola, and Y. LeCun. Deep Learning Made Easier by Linear Transformations in Perceptrons. Proc. of AISTATS, 2012.
- Y. LeCun, L. Bottou, G. Orr and K. Muller: Efficient BackProp, in Orr, G. and Muller K. (Eds), Neural Networks: Tricks of the trade, Springer, 1998.
- T. Vatanen, T. Raiko, H. Valpola, and Y. LeCun. Pushing Stochastic Gradient towards Second-Order Methods - Backpropagation Learning with Transformations in Nonlinearities. In Proc. ICONIP, 2013.

References - Convergence

- Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." International conference on artificial intelligence and statistics. 2010.
- Martens, J. Deep learning via Hessian-free optimization. In ICML, 2010.
- Saxe, Andrew M., James L. McClelland, and Surya Ganguli. "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks." In ICLR 2014.
- Pascanu, R., "On Recurrent and Deep Neural Networks", PhD thesis, University of Montreal, 2014.
- Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." Proc. of ICML, 2015.
- T. Raiko, H. Valpola, and Y. LeCun. Deep Learning Made Easier by Linear Transformations in Perceptrons. Proc. of AISTATS, 2012.
- Y. LeCun, L. Bottou, G. Orr and K. Muller: Efficient BackProp, in Orr, G. and Muller K. (Eds), Neural Networks: Tricks of the trade, Springer, 1998.
- T. Vatanen, T. Raiko, H. Valpola, and Y. LeCun. Pushing Stochastic Gradient towards Second-Order Methods - Backpropagation Learning with Transformations in Nonlinearities. In Proc. ICONIP, 2013.

References - Regularisation

- Tikhonov, Andrey Nikolayevich (1943). On the stability of inverse problems (in Russian). Doklady Akademii Nauk SSSR 39 (5): 195–198.
- J. Luketina, M. Berglund, and T. Raiko (2016). Scalable Gradient-Based Tuning of Continuous Regularization Hyperparameters. To be presented at the ICLR workshop track. Preprint available as arXiv:1511.06727 [cs.LG].
- Dosovitskiy, A., Springenberg, J. T., Riedmiller, M., & Brox, T. (2014). Discriminative unsupervised feature learning with convolutional neural networks. In Advances in Neural Information Processing Systems (pp. 766-774).
- Sietsma, J. and Dow, R. J. (1991). Creating artificial neural networks that generalize. Neural networks, 4(1), 67–79.
- Rifai, S., Vincent, P., Muller, X., Glorot, X., & Bengio, Y. Contractive auto-encoders: Explicit invariance during feature extraction. In ICML, pp. 833-840, 2011.
- Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network." NIPS 2014 Deep Learning Workshop.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), 1929-1958.
- Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. Science, 313(5786), 504-507.
- Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, Rob Fergus. Intriguing properties of neural networks. International Conference on Learning Representations (2014)
- Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, Shin Ishii. Distributional Smoothing with Virtual Adversarial Training. Under review as a conference paper at ICLR 2016.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, ACL, 2018



Thanks!
Ole Winther