Deep Learning

LM Computer Science, Data Science, Cybersecurity

2nd semester - 6 CFU

References:

Deep Learning Book (main book)

Mitchell (machine learining concepts)

Bishop (machine learning)

Regularization for deep learning (chapter 7)

Regularization

- Any modification we make to a learning algorithm intended to **reduce its generalization error but not (much) its training error**
- Deep neural networks tend to represent very complex functions -> **overfitting** (variance in the estimator)
- **Regularization**: trades bias with variance
  - Example of Learning bias: favour simpler hypothesis

Constrained optimization problem

- Krush-Kuhn-Tucker approach (generalized Laplacian)
- Problem: minimize $f(\boldsymbol{x})$, such that

$$\forall_i g^{(i)}(\boldsymbol{x}) = 0, \;\; \forall_j h^{(j)}(\boldsymbol{x}) \leq 0$$

Define a Generalized Lagrangian:

$$\min_{x} \max_{\lambda} \max_{\alpha, \alpha \geq 0} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha})$$

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = f(x) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) + \sum_j \alpha_j h^{(j)}(\boldsymbol{x})$$

Under some conditions, generalized Laplacian has the same objective function value and set of optimal points as the original problem

- We can solve a constrained problem using unconstrained optimization

Parameter norm penalties

- Oldest regularization strategies (adopted for linear/logistic regression)

- Limit the capacity of the models, adding a parameter **norm penalty** to the **objective** function

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

Mini-batch of examples

Weight norm

- Larger α results in more regularization

- Different norms favour different solutions

- In general only weights are regularized (bias terms tend to be easy to learn)

Parameter norm penalties

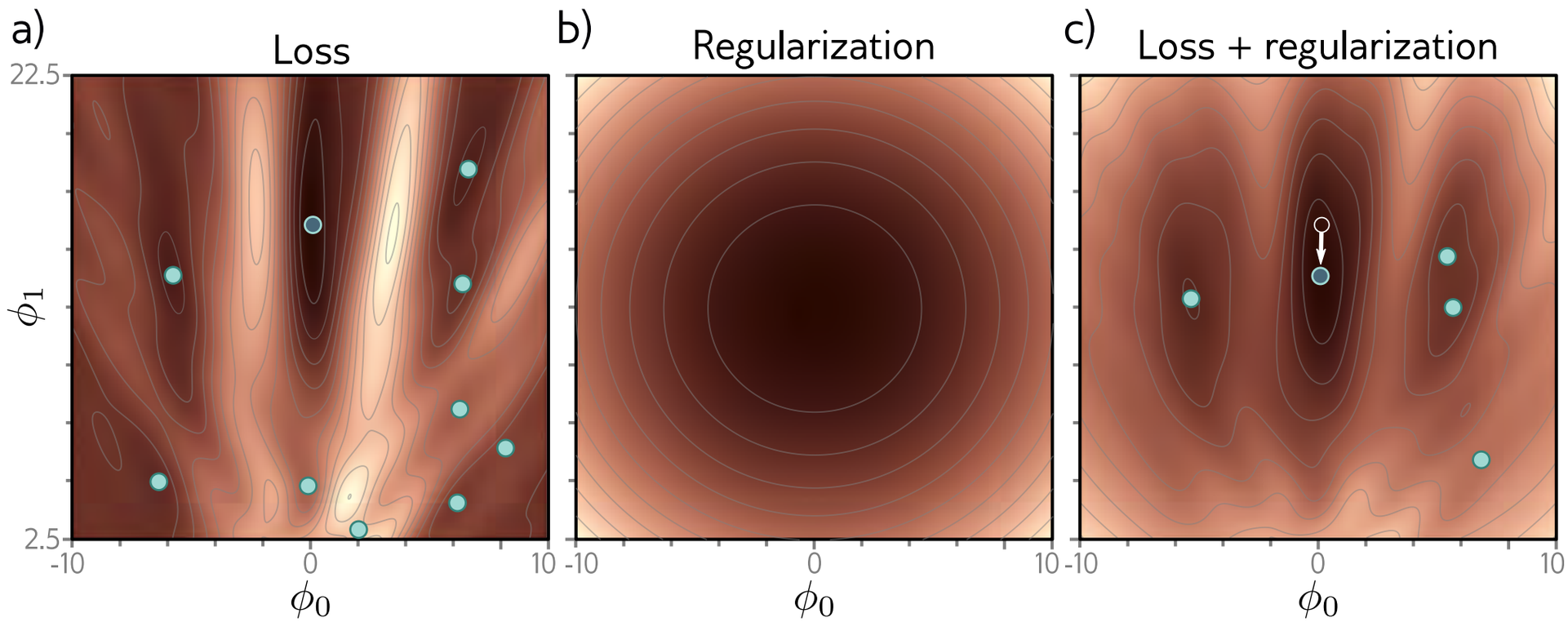


Figure 9.1 Explicit regularization. a) Loss function for Gabor model (see section 6.1.2). Cyan circles represent local minima, green circle represents the global minimum. b) The regularization term favors parameters close to the center of the plot by adding an increasing penalty as we move away from this point. c) The final loss function is the sum of the original loss function plus the regularization term. This surface has fewer local minima, and the global minimum has moved to a different position (arrow shows change).

Weight Decay (L^2 norm)

- Drives the weights close to the origin

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|w\|_2{}^2$$

Probabilistic interpretation: from ML to MAP with Gaussian prior on the weights

- Also known as ridge regression or Tikhonov regularization
- Assume no bias, i.e. $\boldsymbol{\theta} = \boldsymbol{w}$, and strength of reg. α

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

- With gradient $\nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}}J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$
- A single SGD step with learning rate ϵ would be

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon(\alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}}J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}))$$
$$\boldsymbol{w} \leftarrow (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon\nabla_{\boldsymbol{w}}J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

  - The weights get shrinked on each step: weight decay

Ordinary Least squares

- Regularization is necessary in some ill-posed ML problems
  - When we have less examples than features
    - Or more in general when the solution is not unique
- Consider linear regression

$$\boldsymbol{X}\boldsymbol{w} = \boldsymbol{y}$$

- With MSE loss: $J(\boldsymbol{w}) = \frac{1}{n}\|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|^2$

Least squares closed-form solution

- Gradient

$$\nabla_{\boldsymbol{w}} J = \frac{1}{n} 2(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})^T \boldsymbol{X} = \frac{1}{n} 2(\boldsymbol{w}^T \boldsymbol{X}^T \boldsymbol{X} - \boldsymbol{y}^T \boldsymbol{X})$$

- Closed-form solution equating the derivative to zero

$$\frac{1}{n} 2(\boldsymbol{w}^T \boldsymbol{X}^T \boldsymbol{X} - \boldsymbol{y}^T \boldsymbol{X}) = \boldsymbol{0}$$

$$\boldsymbol{w}^T \boldsymbol{X}^T \boldsymbol{X} = \boldsymbol{y}^T \boldsymbol{X}$$

$$\boldsymbol{w}^T = \boldsymbol{y}^T \boldsymbol{X} (\boldsymbol{X}^T \boldsymbol{X})^{-1}$$

$$\boldsymbol{w} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

- $(\boldsymbol{X}^T \boldsymbol{X})^{-1}$ should be invertible!
  - Square and full-rank (rank n)
  - $\boldsymbol{X}^T \boldsymbol{X}$ is not invertible when the number of variables exceeds the number of data points

Rank: the maximum number of linearly independent rows that can be chosen from the matrix

Regularized Least Squares(L^2 norm)

With (M)SE loss and l2: $J(\boldsymbol{w}) = \frac{1}{n}\|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|^2 + \alpha\|\boldsymbol{w}\|^2$

- Gradient

$$\nabla_{\boldsymbol{w}} J = 2(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})^T\boldsymbol{X} + 2\alpha\boldsymbol{w}^T = 2(\boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{X} - \boldsymbol{y}^T\boldsymbol{X} + \alpha\boldsymbol{w}^T)$$

- Closed-form solution equating the derivative to zero

$$2(\boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{X} - \boldsymbol{y}^T\boldsymbol{X} + \alpha\boldsymbol{w}^T) = 0$$
$$\boldsymbol{w}^T\boldsymbol{X}^T\boldsymbol{X} + \alpha\boldsymbol{w}^T = \boldsymbol{y}^T\boldsymbol{X}$$
$$\boldsymbol{w}^T(\boldsymbol{X}^T\boldsymbol{X} + \alpha\boldsymbol{I}) = \boldsymbol{y}^T\boldsymbol{X}$$
$$\boldsymbol{w}^T = \boldsymbol{y}^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X} + \alpha\boldsymbol{I})^{-1}$$
$$\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X} + \alpha\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

- Full rank! Always invertible!

Moore-Penrose pseudo-inverse

$$\boldsymbol{X}^+ = \lim_{\alpha \to 0}(\boldsymbol{X}^T\boldsymbol{X} + \alpha\boldsymbol{I})^{-1}\boldsymbol{X}^T$$

L^1 regularization

$$\Omega(\boldsymbol{\theta}) = \|w\|_1$$

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha\|w\|_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

- With gradient $\nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha\, sign(\boldsymbol{w}) + \nabla_{\boldsymbol{w}}J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$
- Where $sign(\cdot)$ is applied element-wise
- Forces solutions to be sparse
  - i.e. performs feature selection

Why does L^1 reg. enforce sparsity?

- Consider linear regression

$$\boldsymbol{X}\boldsymbol{w} = y$$

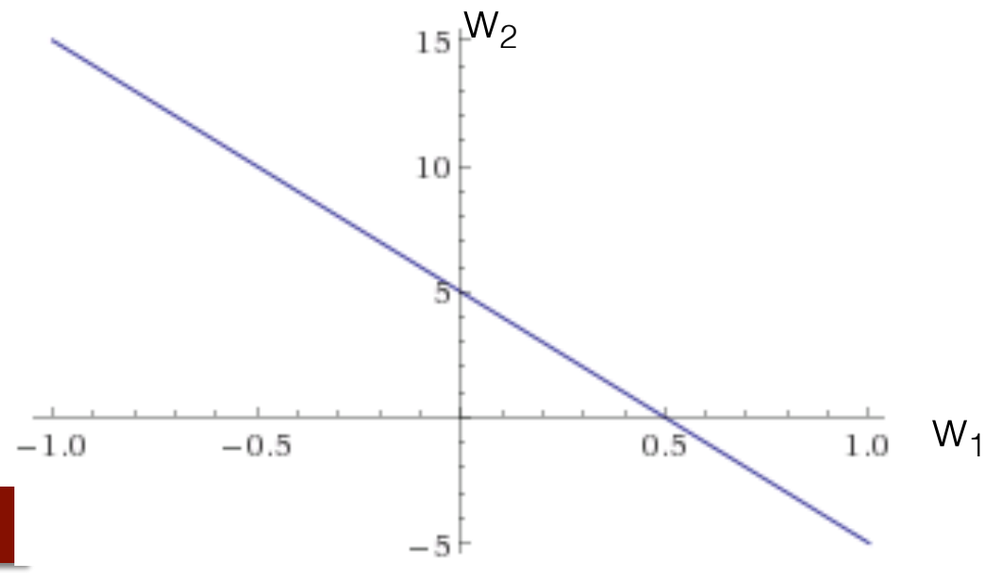
$$\begin{bmatrix} x_1^{(1)} & x_2^{(1)} \\ x_1^{(2)} & x_2^{(2)} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

That is ill-defined, e.g. a single ex. in the training set

$$\begin{bmatrix} 10 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = 5$$

We have infinitely many solutions satisfying

$$w_2 = 5 - 10w_1$$

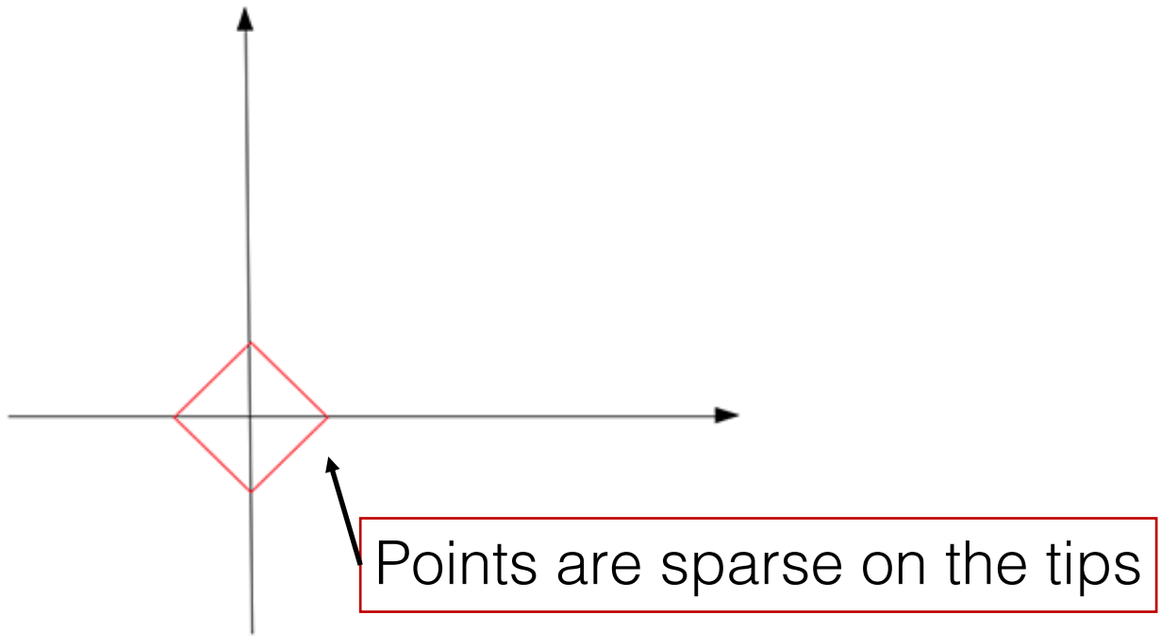


Why does L^1 reg. enforce sparsity?

- L1 norm is the sum. of absolute values

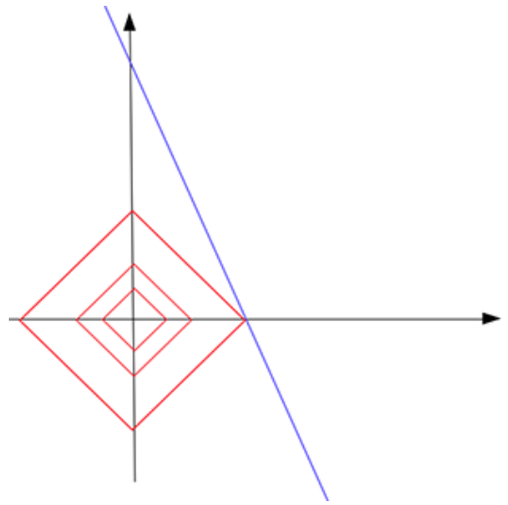
$$\|w\|_1 = |w_1| + |w_2|$$

Let us draw the points with constant l1 norm =c

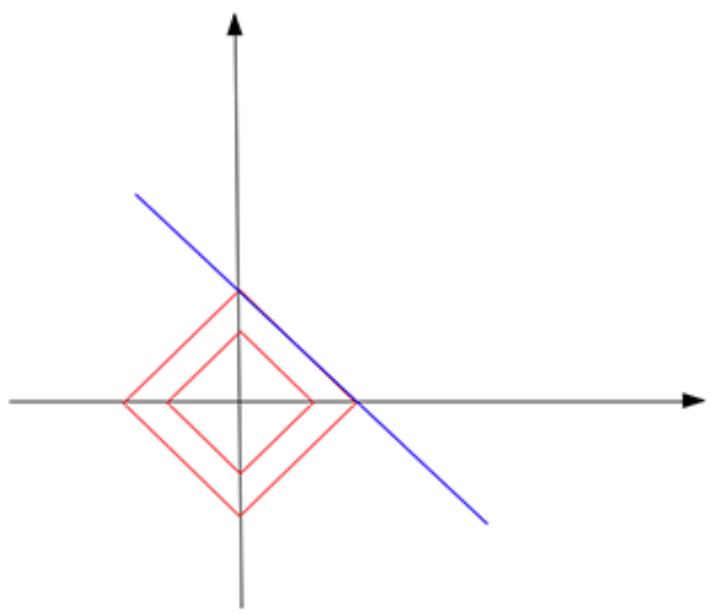


Why does L^1 reg. enforce sparsity?

- The solution with minimum l1 probably will be sparse



- ..not always, but in high-dimensional spaces it is likely



Note: with l2 points with constant norm make a circle

Parameter norm regularization-summary

- Regularization is necessary in some ill-posed ML problems
  - When we have less examples than features
    - Or more in general when the solution is not unique

- Regularization (in particular l2) warrantees that the optimization problem is well-defined
  - Using the Moore-Penrose pseudo-inverse instead of the inverse corresponds to stabilizing underdetermined problems with regularization

$$\boldsymbol{X}^{+} = \lim_{\alpha \to 0} (\boldsymbol{X}^{T}\boldsymbol{X} + \alpha \boldsymbol{I})^{-1}\boldsymbol{X}^{T}$$

- Useful to avoid overfitting

Data augmentation

- Generate “fake” training data
  - Applying transformations to the training instances
  - E.g. object detection (on images), we want the model to be invariant to roto—translations or scaling
- Injecting noise in the input examples
  - E.g. denoising autoencoder
- Noise in the hidden representations → dropout
- Noise in the target, e.g. label smoothing
  - Maximum Likelihood learning may never converge and keeps increasing the weights
  - Replacing hard 0 and 1 targets with $\epsilon/_{k-1}$ and $1-\epsilon$

Semi-supervised learning as regularizer

- Exploits unlabelled examples from $P(\boldsymbol{x})$ and labelled examples from $P(\boldsymbol{x}, y)$ to estimate $P(y \mid \boldsymbol{x})$
- We learn a representation $\mathbf{h} = f(\boldsymbol{x})$ such that similar examples are close in the new space
  - A linear classifier in the new space may achieve better generalization
- E.g. unsupervised generative model modelling $P(\boldsymbol{x})$ (autoencoder) that shares parameters with a discriminative model estimating $P(y \mid \boldsymbol{x})$

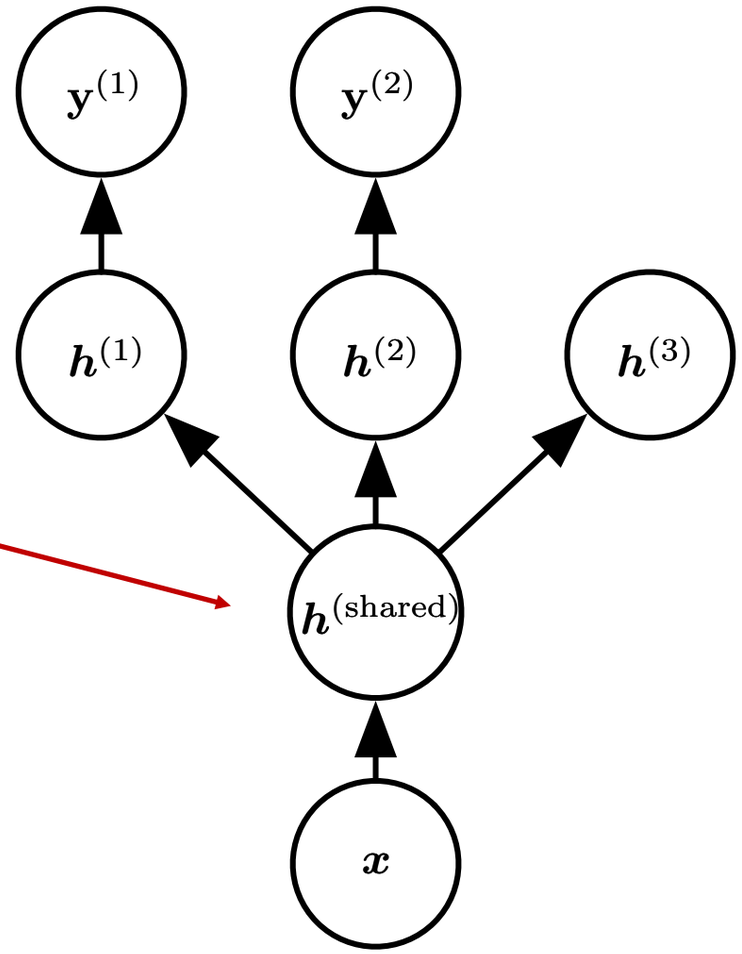
Transfer learning (Pre-training)

- When training data is limited, other datasets can be exploited to improve performance.

- Transfer learning:
  - the network is trained to perform a related secondary task for which data are more plentiful.
  - The resulting model is subsequently adapted to the original task by removing the last layer and adding one or more layers that produce a suitable output.
    - The main model may be fixed and the new layers trained for the original task, or we may fine-tune the entire model
- The network will build a good internal representation of the data from the secondary task, and this can subsequently be exploited for the original task.
- transfer learning can be viewed as initializing most of the parameters of the final network in a sensible part of the space that is likely to produce a good solution

Multi-Task learning

- Idea: use examples from different (related) tasks.
- Part of the model is shared across different tasks, and is driven to learn a representation that generalizes well
- Assumes shared factors that explain variation in data
  - Task-specific parameters
  - Generic parameters



Self-supervised learning

- When no data from other tasks is available, we can create large amounts of "free" labeled data using self-supervised learning and use this for transfer learning.
- generative self-supervised learning: part of each data example is masked and the secondary task is to predict the missing part
- contrastive self-supervised learning: two versions of each unlabeled example are presented, where one has been distorted in some way. The system is trained to predict which is the original. For example:
  - we might use a corpus of unlabeled images where the secondary task is to identify which version of the image is upside-down
  - we might use a large corpus of text, where the secondary task is to determine whether two sentences followed one another or not in the original text.

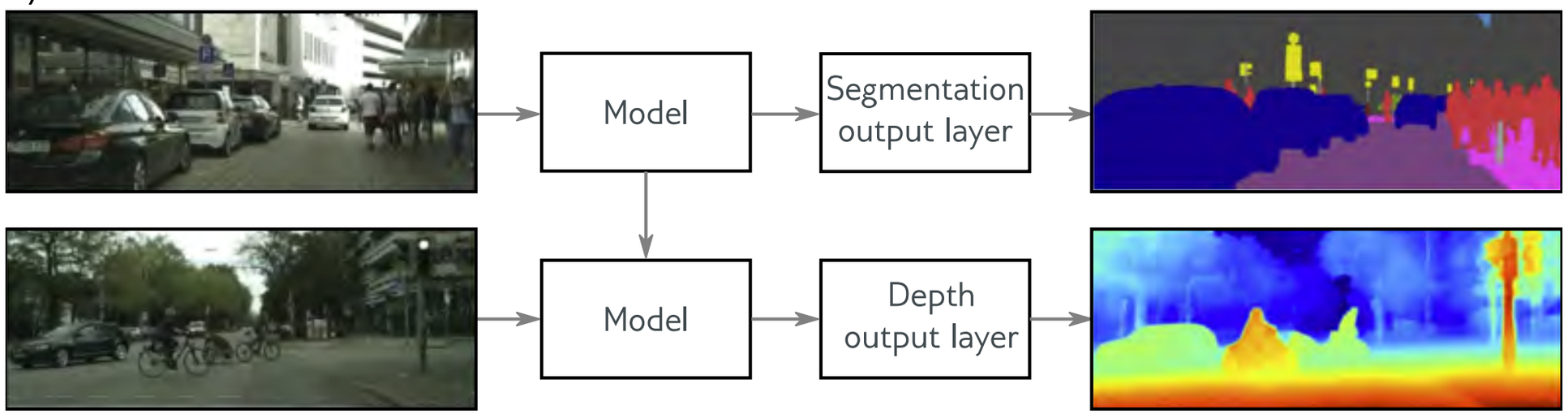
Exercise: Transfer, multi-task, and self-supervised learning.

multi-task learning

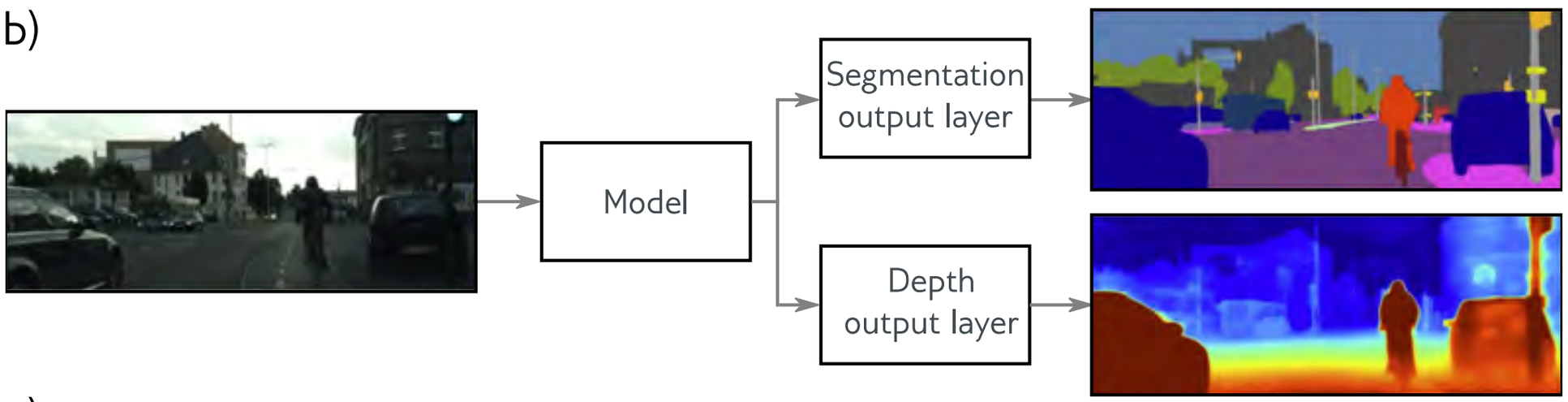
Self-supervised learning

Transfer learning

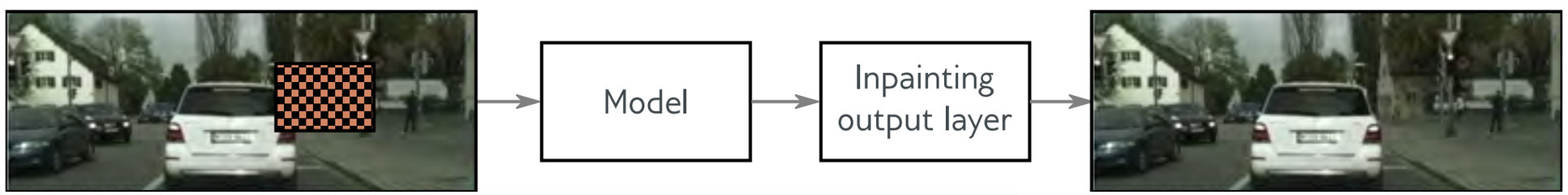
a)



b)

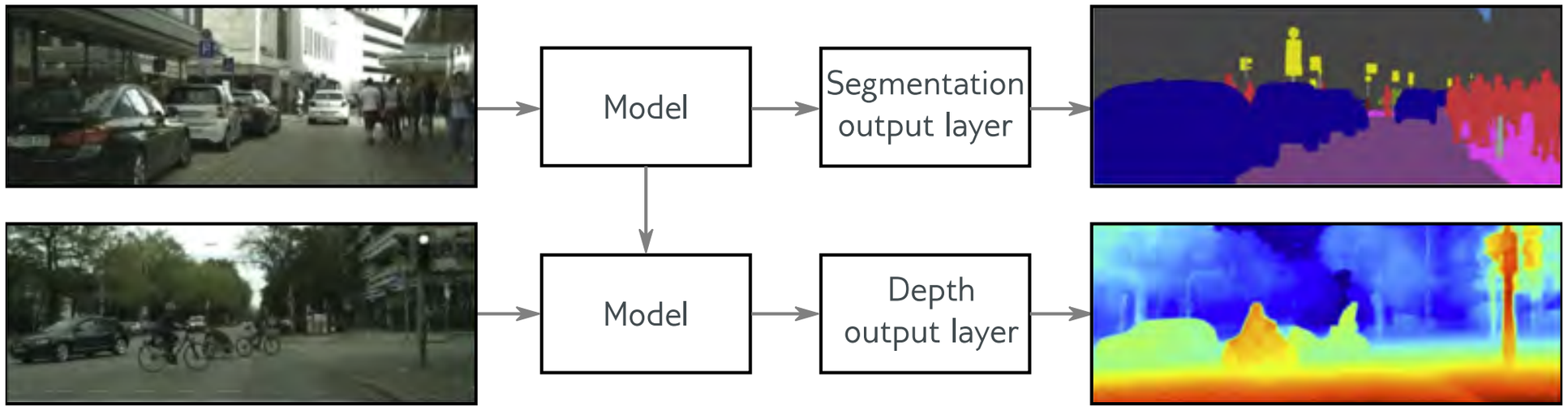


c)



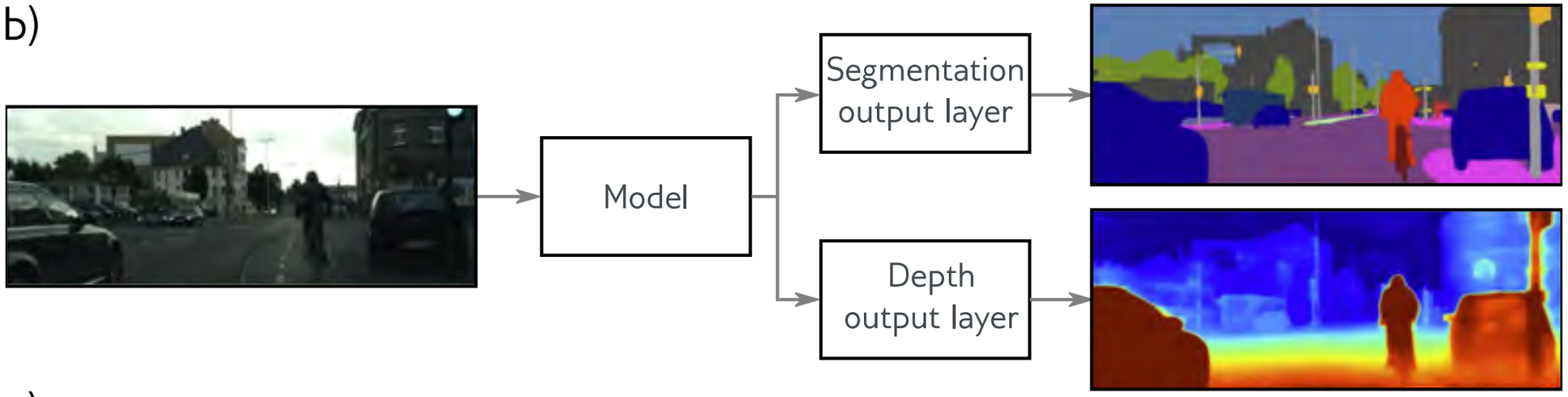
Self-supervised learning

a)



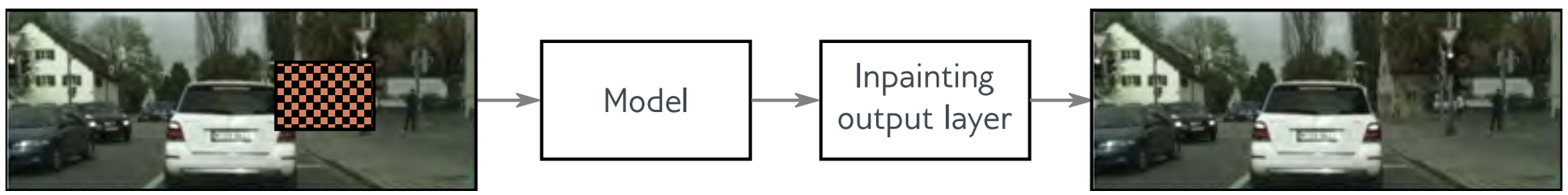
Transfer learning

b)



multi-task learning

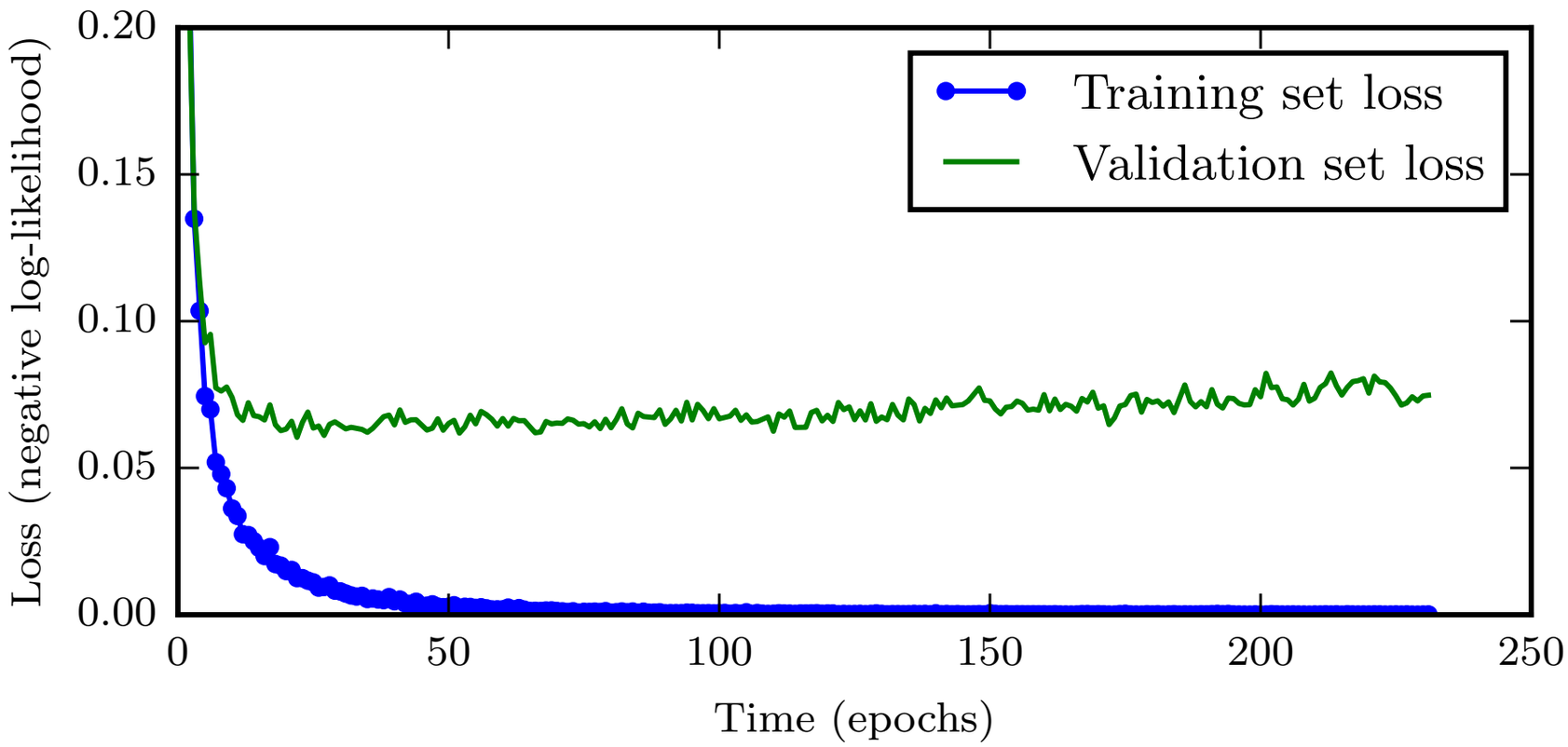
c)



Self-supervised learning

Early Stopping

- Large models overfit the task
- Idea: return the model with lowest validation loss
  - Stop training if no improvement for N epochs
  - Regularization: limits the parameter space to a neighbourhood of the initial parameter values



Parameter Tying and sharing

- Belief that some parameters should be close to one another
- E.g. two models A and B solve similar tasks
  - regularization term penalizing distant parameters

$$\Omega\left(w^{(A)}w^{(B)}\right) = \left\|w^{(A)} - w^{(B)}\right\|_2^2$$

- Parameter sharing (seen with Multi-task learning)
  - Especially effective in Convolutional Neural Networks
  - Sharing of parameters across multiple image locations
  - Example of incorporating domain knowledge in network architecture

Sparse representations

- We have seen L^1 regularization to induce **weight** sparsity

- We can similarly induce **representation sparsity**

$$\Omega(\boldsymbol{h}) = \|\boldsymbol{h}\|_1 = \sum_i |h_i|$$

  - not just L^1, we can use any penalty on representations

- A hard constraint can be posed on the norm of $\boldsymbol{h}$

  - Orthogonal Matching Pursuit finds a sparse input repr.

$$\underset{\boldsymbol{h}, \|\boldsymbol{h}\|_0 < k}{arg\min} \|\boldsymbol{x} - \boldsymbol{W}\boldsymbol{h}\|^2$$

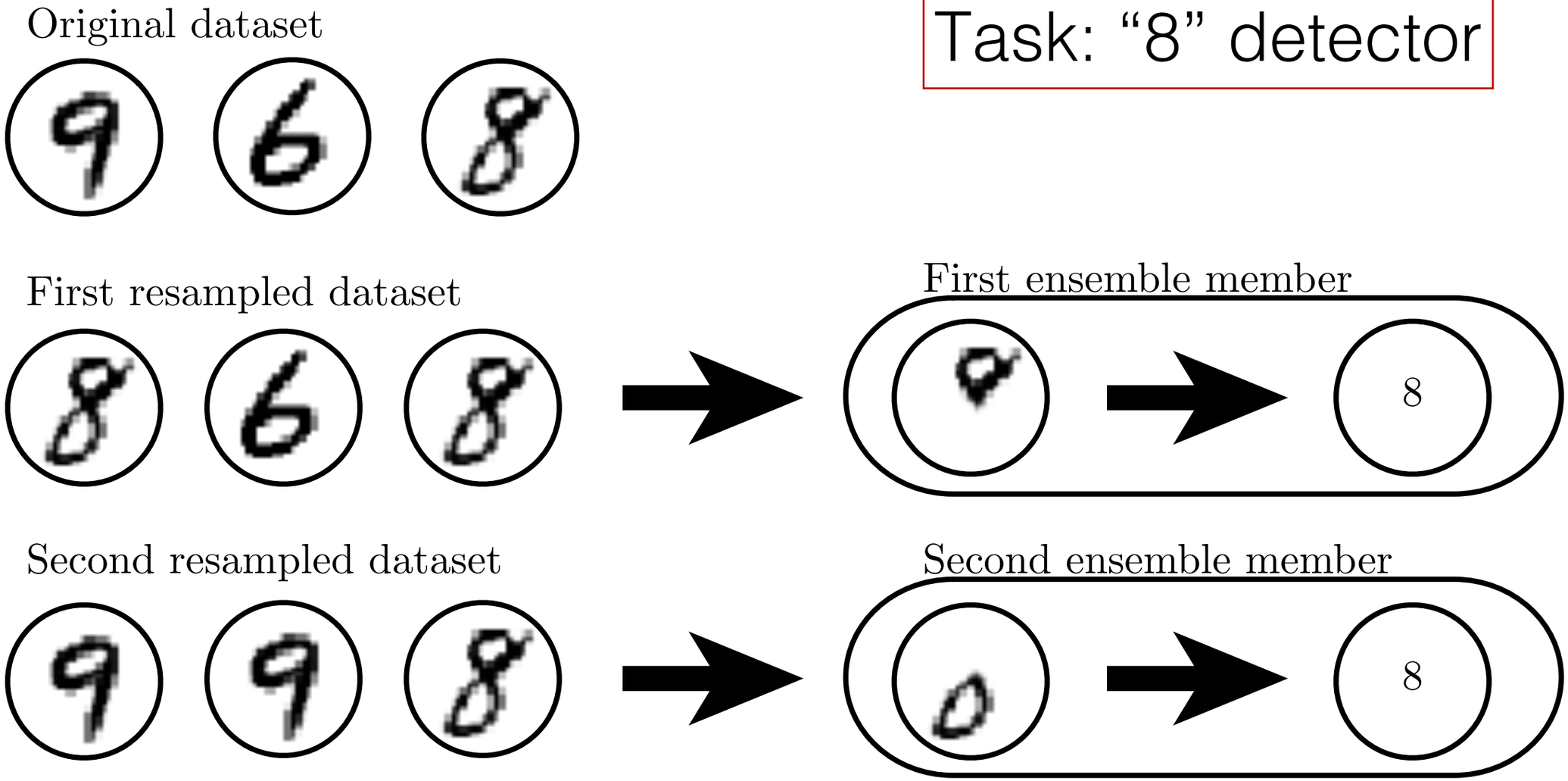
  - At most k non-zero entries in $\boldsymbol{h}$. Can be solved efficiently when $\boldsymbol{W}$ is orthogonal.

Ensemble Methods

- Idea: train different models
  - Each model votes on the output
  - Different models will (hopefully) not make the same mistakes
- Bagging (bootstrap aggregating)
  - Increase bias and reduce variance compared to single models
    - Construct k different datasets
      - Same number of examples as the original dataset
    - Sampling with replacement
      - Approx. $1/3$ of the examples will be repeated if sampling n elements
    - A model is trained for each dataset. Differences in training set results in differences in the resulting models
- Boosting: vice-versa, builds ensemble with higher capacity

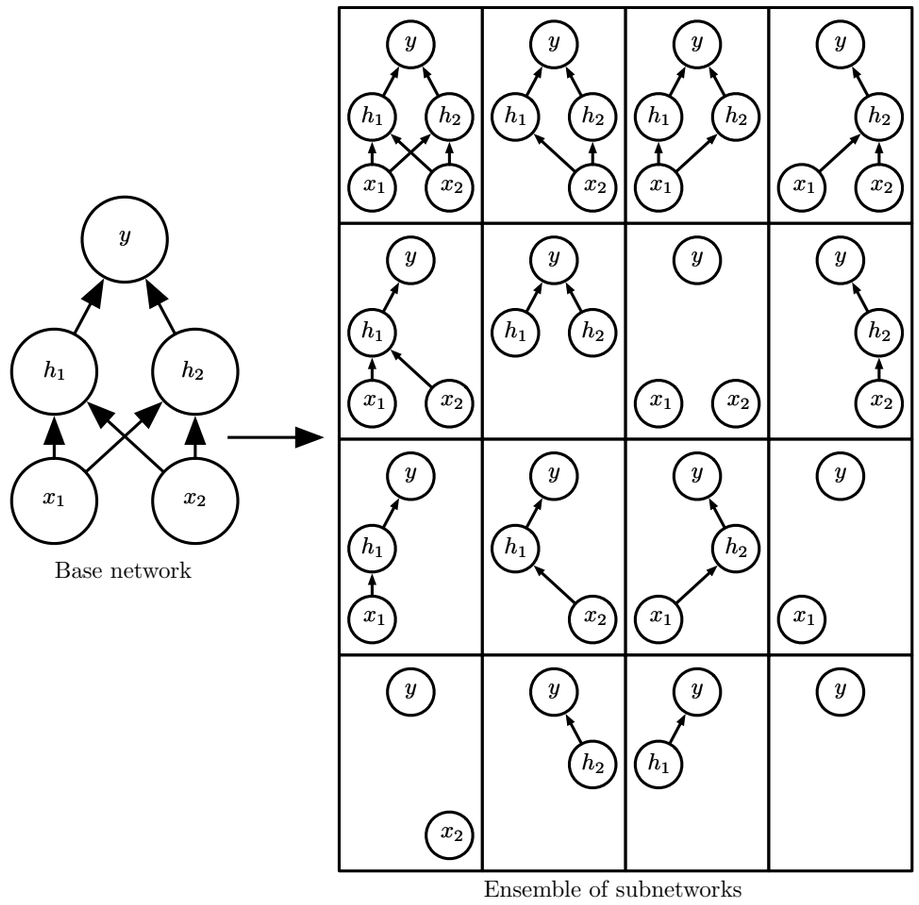
Bagging

Task: “8” detector

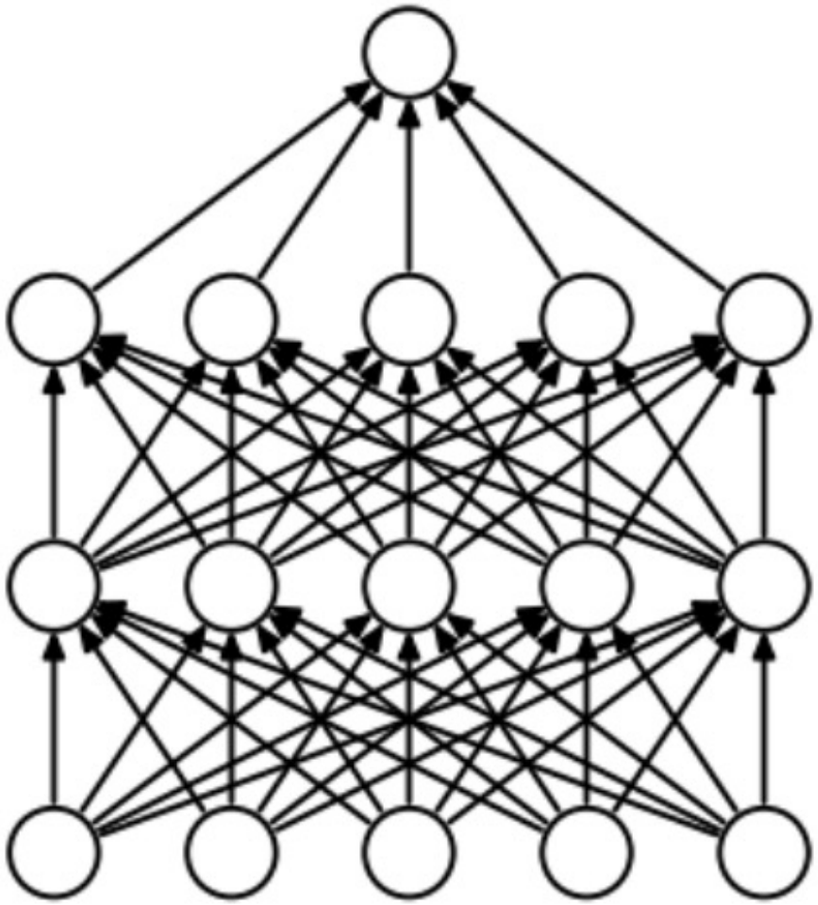


Dropout

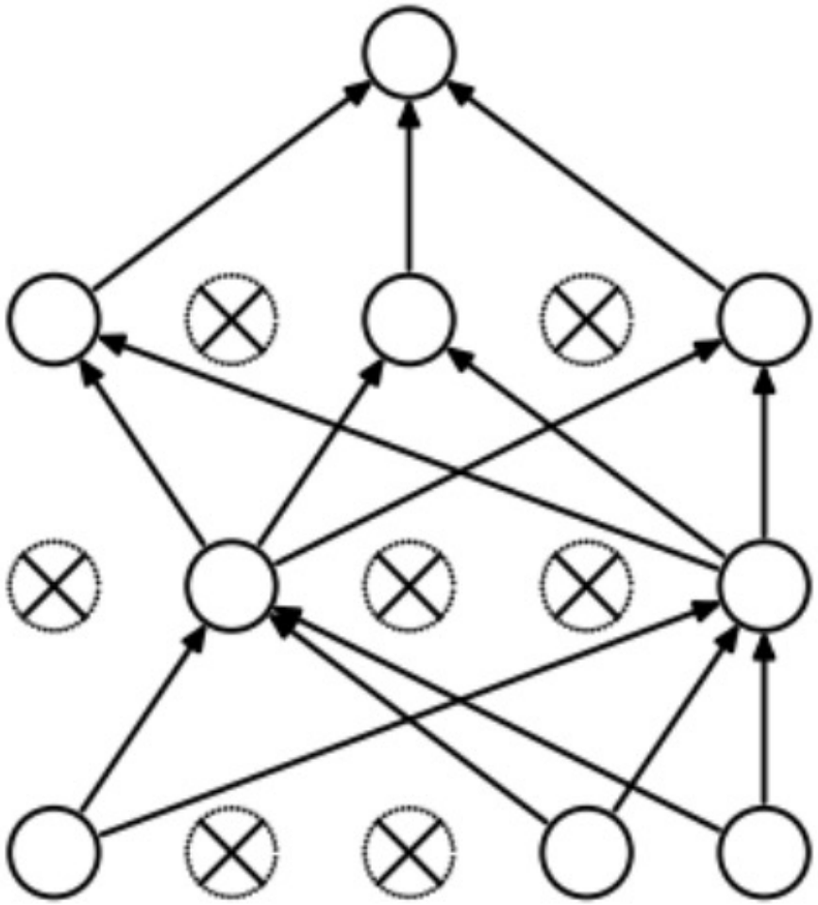
- An efficient way of performing bagging on neural networks
  - Approximates bagging on exponentially many NNs
  - Obtained removing hidden/input units
  - Efficient: just multiply the output of a unit by 0



Dropout



(a) Standard Neural Net



(b) After applying dropout.

Dropout

- Minibatch-based learning algorithm
  - Same idea of bagging, with different classifiers obtained removing neurons from the network (shared parameters)
  - Binary mask $\boldsymbol{\mu}$ to apply to all input and hidden units
  - Probability of including a unit is a hyper-parameter
  - Training objective $\mathbb{E}_{\boldsymbol{\mu}} J(\boldsymbol{\theta}, \boldsymbol{\mu})$
    - Unbiased estimate sampling $\boldsymbol{\mu}$ for each training step
  - Prediction: mean over all models (all masks)
  - If we use *geometric* mean, we can approximate the ensemble prediction with just one forward pass
    - **Weight scaling inference rule**: Multiplying the weights going out from a unit by the probability of including that unit

Weight scaling for dropout

- Linear classifier (softmax regression) n input variables

$$P(\mathrm{y} = y|\boldsymbol{x}) = softmax(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b})_y$$

$$softmax(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

- Sub-models: element-wise multiplication with binary mask

$$P(\mathrm{y} = y|\boldsymbol{x}; \boldsymbol{d}) = softmax(\boldsymbol{W}^T(\boldsymbol{x} \odot \boldsymbol{d}) + \boldsymbol{b})_y$$

- Geometric mean of predictions

$$\tilde{P}_{ensemble}(\mathrm{y} = y|\boldsymbol{x}) = \sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} P(\mathrm{y} = y|\boldsymbol{x}; \boldsymbol{d})}$$

- Geom. Mean is not a probability: renormalize it

$$P_{ensemble}(\mathrm{y} = y|\boldsymbol{x}) = \frac{\tilde{P}_{ensemble}(\mathrm{y} = y|\boldsymbol{x})}{\sum_{y'} \tilde{P}_{ensemble}(\mathrm{y} = y'|\boldsymbol{x})}$$

Weight scaling for dropout

$$\tilde{P}_{ensemble}(\mathrm{y} = y|\boldsymbol{x}) = \sqrt[2^n]{\prod_{\boldsymbol{d}\in\{0,1\}^n} softmax(\boldsymbol{W}^T(\boldsymbol{x}\odot\boldsymbol{d}) + \boldsymbol{b})_y}$$

$$= \sqrt[2^n]{\prod_{\boldsymbol{d}\in\{0,1\}^n} \frac{exp(\boldsymbol{W}^T_{y,:}(\boldsymbol{x}\odot\boldsymbol{d}) + b_y)}{\sum_{\boldsymbol{y'}} exp(\boldsymbol{W}^T_{y',:}(\boldsymbol{x}\odot\boldsymbol{d}) + b_{y'})}}$$

Since $\tilde{P}$ will be normalized, we can ignore the denominator (it is the same for all y)

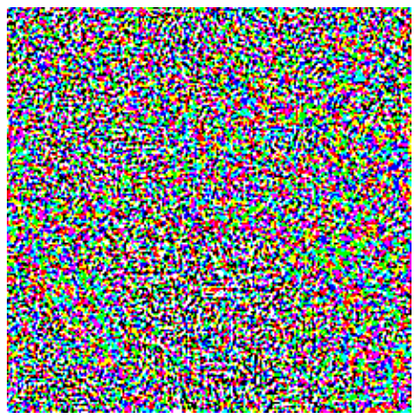
$$\tilde{P}_{ensemble}(\mathrm{y} = y|\boldsymbol{x}) \propto \sqrt[2^n]{\prod_{\boldsymbol{d}\in\{0,1\}^n} exp(\boldsymbol{W}^T_{y,:}(\boldsymbol{x}\odot\boldsymbol{d}) + b_y)}$$

$$= \exp\left(\frac{1}{2^n}\sum_{\boldsymbol{d}\in\{0,1\}^n} \boldsymbol{W}^T_{y,:}(\boldsymbol{x}\odot\boldsymbol{d}) + b_y\right) = \exp\left(\boldsymbol{W}^T_{y,:}E_{\boldsymbol{d}}[(\boldsymbol{x}\odot\boldsymbol{d})] + b_y\right) = \exp\left(\frac{1}{2}\boldsymbol{W}^T_{y,:}\boldsymbol{x} + b_y\right)$$

the final normalization gives the softmax with scaled weights

Adversarial training

$+ .007 \times$



$=$

$\boldsymbol{x}$

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

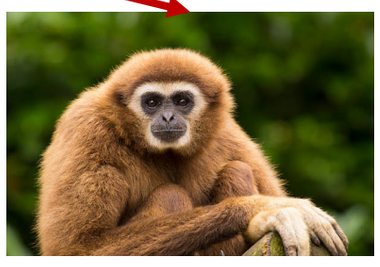
$\boldsymbol{x} +$
$\epsilon \, \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

$y =$“panda”
w/ 57.7%
confidence

“nematode”
w/ 8.2%
confidence

“gibbon”
w/ 99.3 %
confidence

?



- Training on adversarial examples is mostly intended to improve security but can sometimes provide generic regularization.

- Examples of learning curves: what to expect when applying regularization?

END