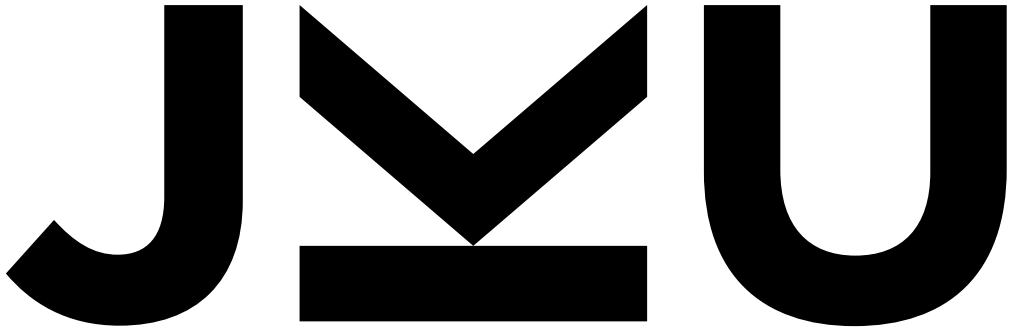
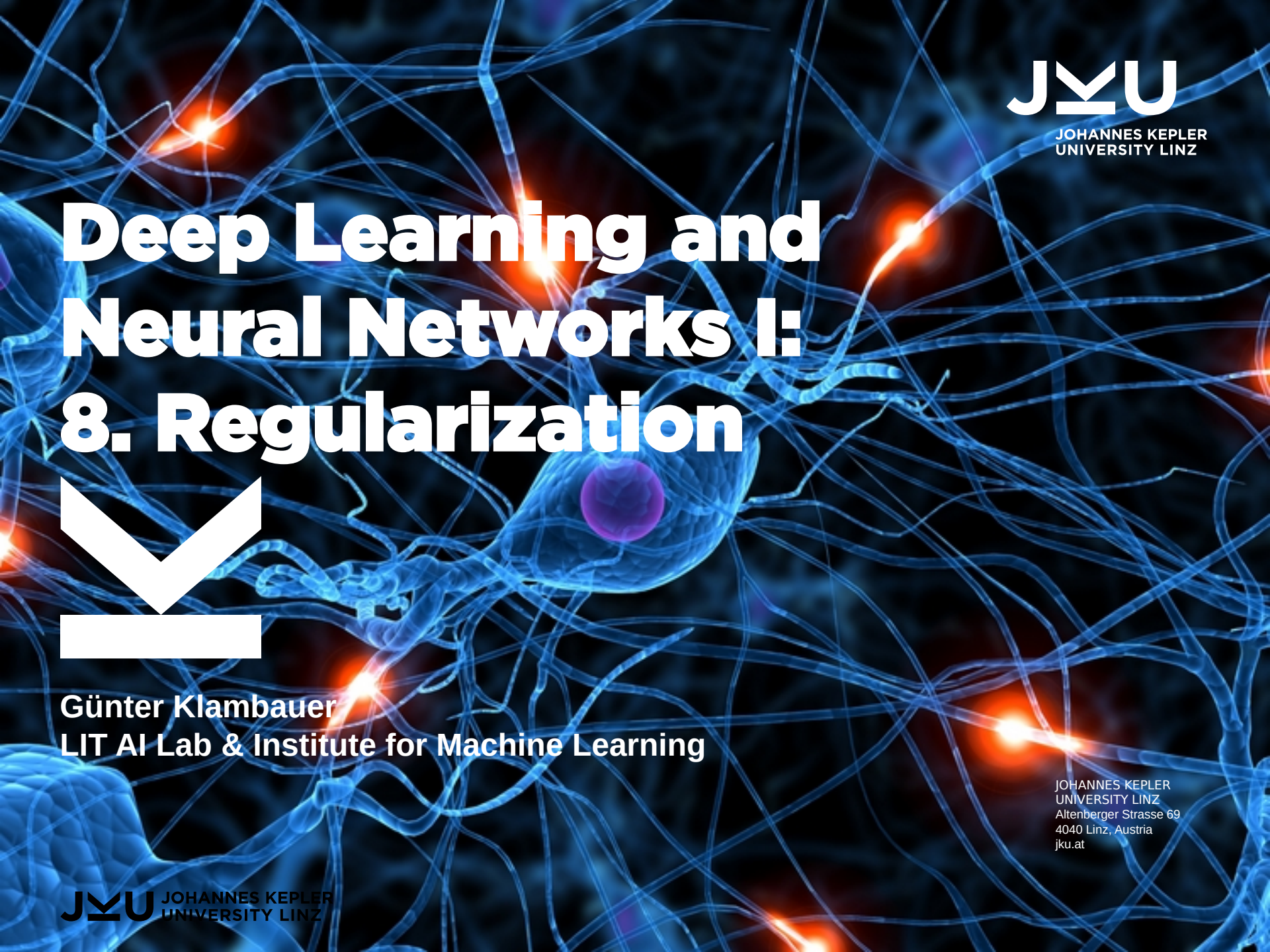


JOHANNES KEPLER
UNIVERSITY LINZ



Deep Learning and Neural Networks I: 8. Regularization

Günter Klambauer
LIT AI Lab & Institute for Machine Learning

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Strasse 69
4040 Linz, Austria
jku.at

Administrative

Overview

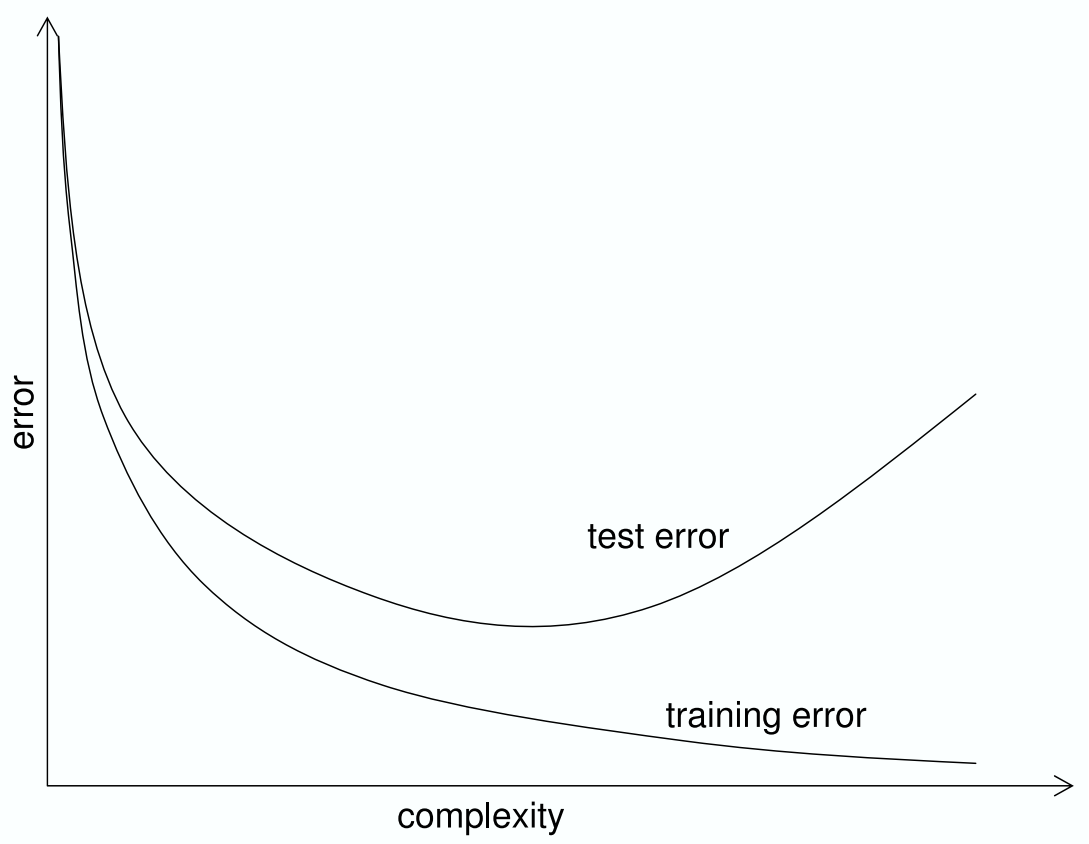
- 8.1 Weight Decay
- 8.2 Neuron Noise and Weight Noise
- 8.3 Weight Sharing
- 8.4 Dropout
- 8.5 Multi-task Learning
- 8.6 Growing
- 8.7 Pruning Methods

Overview

- 8.8 Early Stopping
- 8.9 Flat Minimum Search
- 8.10 Data Augmentation
- 8.11 Self-regularization by Stochastic Gradient Descent

Risk as a Function of the Complexity

$$R \leq R_{\text{emp}} + \text{complexity}$$



Principle Regularization Approaches

Typical **regularization terms** are:

- Smoothing terms, which control the curvature and higher order derivatives of the function represented by the network
- Complexity terms, which control number of units and weights or their precision

Types:

- Regularization during training
- Regularization after training

8.1 Weight Decay

The empirical risk R_{emp} is extended by a complexitx term $\Omega(w)$:

$$R_{\text{reg}}(\boldsymbol{w}) \;=\; R_{\text{emp}}(\boldsymbol{w}) \;+\; \lambda\Omega(\boldsymbol{w})$$

Gradient descent:

$$\nabla_{\boldsymbol{w}} R_{\text{reg}}(\boldsymbol{w}) \;=\; \nabla_{\boldsymbol{w}} R_{\text{emp}}(\boldsymbol{w}) \;+\; \lambda\, \nabla_{\boldsymbol{w}}\Omega(\boldsymbol{w})$$

L2 weight decay:

$$R_{\text{reg}}(\boldsymbol{w}) \;=\; R_{\text{emp}}(\boldsymbol{w}) \;+\; \frac{\lambda}{2}\, \|\boldsymbol{w}\|_2^2$$

8.1 Weight Decay

Interpretation of L2 weight decay

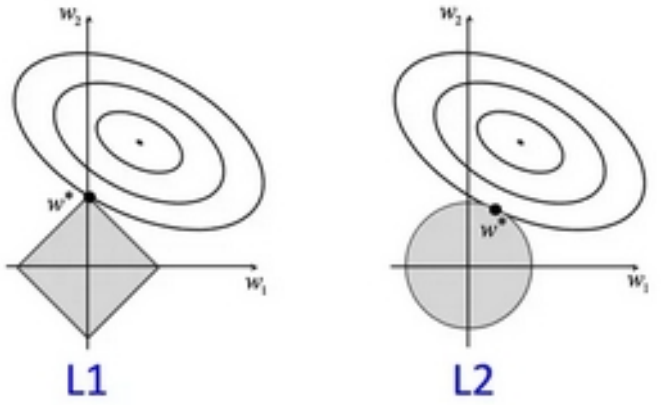
If we neglect the first data-driven term and only consider the complexity term, the gradient would be $\lambda \boldsymbol{w}$.

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta\lambda\boldsymbol{w}^{(t)}$$
$$\boldsymbol{w}^{(t)} = (1 - \eta\lambda)^t \ \boldsymbol{w}^{(0)}$$

The weights decay to zero, if $0 < \eta\lambda < 1$.

8.1 Weight Decay

L1 weight decay: $R_{\text{reg}}(\boldsymbol{w}) = R_{\text{emp}}(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_1$



Further variants:

- $\Omega(\boldsymbol{w}) = \log\left(1 + \boldsymbol{w}^T\boldsymbol{w}\right)$
- $\Omega(\boldsymbol{w}) = \sum_i \frac{w_i^2/w_0^2}{1 + w_i^2/w_0^2}$, w_0 : threshold

8.1 Weight Decay: Interpretation (second part)

Assuming a quadratic approximation of the error around a local minimum w_0 :

$$R(\boldsymbol{w}) \approx R(\boldsymbol{w}_0) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}_0)^T \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}_0)$$

Minimum: $\boldsymbol{w} = \boldsymbol{w}_0$

Adding a L2 regularization term, the minimum moves to a point $\tilde{\boldsymbol{w}}$,where: $\lambda\tilde{\boldsymbol{w}} + \boldsymbol{H}(\tilde{\boldsymbol{w}} - \boldsymbol{w}_0) = \boldsymbol{0}$. We then solve for $\hat{\boldsymbol{w}}$.
We can now use the *eigendecomposition* $\boldsymbol{H} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$:

$$\tilde{\boldsymbol{w}} = (\boldsymbol{H} + \lambda\boldsymbol{I})^{-1}\boldsymbol{H}\boldsymbol{w}_0$$

$$\tilde{\boldsymbol{w}} = \boldsymbol{Q}(\boldsymbol{\Lambda} + \lambda\boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^T\boldsymbol{w}_0$$

Thus eigenvalues are scaled according to $\frac{\lambda_i}{\lambda_i + \lambda}$.

8.2 Neuron Noise and Weight Noise

To regularize and to avoid that the network extracts individual characteristics of few training examples noise can be added to:

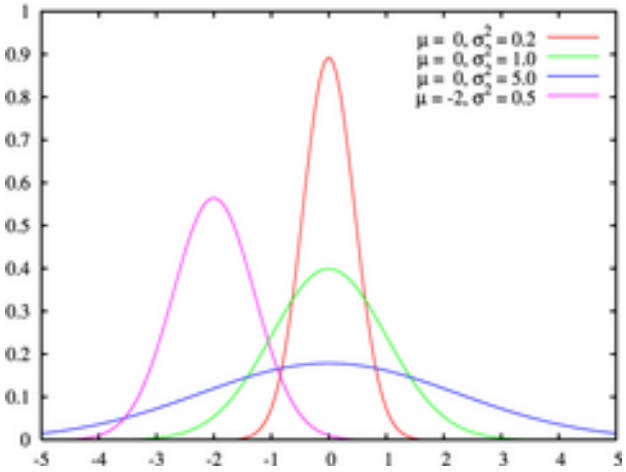
- the training examples,
- the network components.

Properties:

- Closely related to L2 regularization (if noise amplitude is small)
- involves second derivatives of the network function, and so evaluation of the gradients of this error with respect to network weights will be computationally demanding.

Bishop, C. M. (1993). Curvature-driven smoothing: a learning algorithm for feedforward networks. IEEE Transactions on Neural Networks, 4(5), 882-884.
Bishop, C. M. (1995). Neural networks for pattern recognition. Oxford university press. [Section 9.3]

8.3 Weight Sharing



Nowlan and Hinton propose that the weights have special preferred values and should be grouped: *soft weight sharing*

Each group can be modeled by a Gaussian:

$$\mathcal{N}(w;\sigma_j,\mu_j) = \frac{1}{(2\pi)^{1/2}\,\sigma_j}\exp\left(-\frac{1}{2\sigma_j^2}(w-\mu_j)^2\right).$$

Each group j has a prior size $p(j) = \alpha_j$ which estimates what percentage of weights will belong to this group.

Likelihood of a weight w : $$p(w) = \sum_j \alpha_j \mathcal{N}(w;\sigma_j,\mu_j)$$

$$p(w \mid j) = \frac{\alpha_j \mathcal{N}(w;\sigma_j,\mu_j)}{\sum_k \alpha_k \mathcal{N}(w;\sigma_k,\mu_k)}$$

8.3 Weight Sharing

$$R(\boldsymbol{w}) = R_{\text{emp}}(\boldsymbol{w}) - \lambda \sum_i \log p(w_i)$$

$$= R_{\text{emp}}(\boldsymbol{w}) - \lambda \sum_i \log \left(\sum_j \alpha_j \mathcal{N}(w_i; \sigma_j, \mu_j) \right)$$

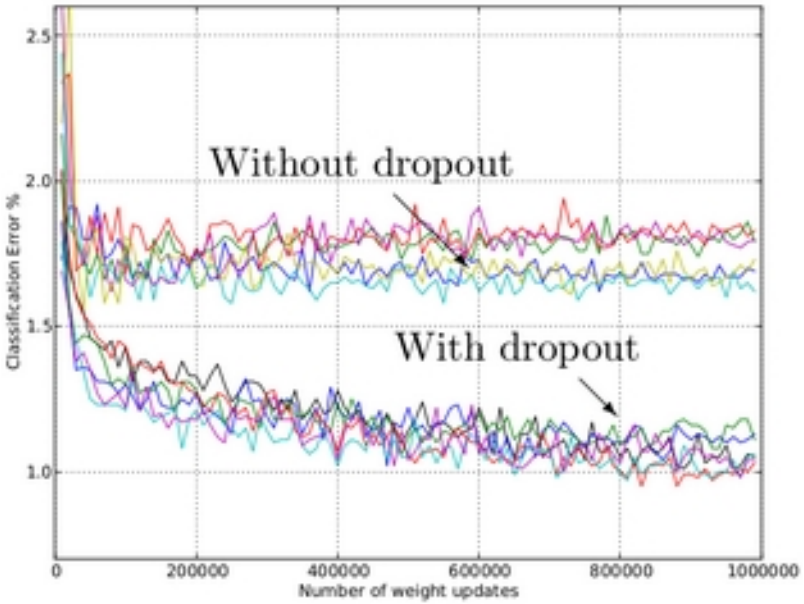
$$\frac{\partial R(\boldsymbol{w})}{\partial w_i} = \frac{\partial R_{\text{emp}}(\boldsymbol{w})}{\partial w_i} + \lambda \sum_j p(w_i \mid j) \frac{(w_i - \mu_j)}{\sigma_j^2}$$

8.3 Weight Sharing

- Weight sharing is also a main component of CNNs
  - Neurons in the same feature map share their weights
  - One reason for the success of CNNs

8.4 Dropout

- Main idea: Set neuron activations to zero randomly during training
- Standard neural network: $\boldsymbol{a} = f(\boldsymbol{W}\boldsymbol{x})$



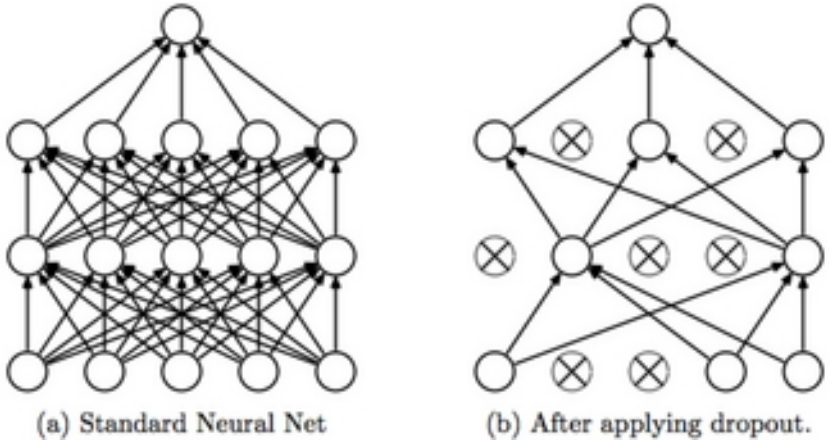
Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958.

8.4 Dropout

- Main idea: Set neuron activations to zero randomly during training
- Neural network with dropout: $\boldsymbol{a} = \boldsymbol{d} \odot f(\boldsymbol{W}\boldsymbol{x})$

  dropout mask $\boldsymbol{d}$:
- Realization of drawing from an I-dimensional Bernoulli RV
- Probability: $p : d_i \sim \mathcal{B}(1, p)$

  $\mathbb{E}(d_i) = p, \mathrm{Var}(d_i) = p(1-p)$



(a) Standard Neural Net

(b) After applying dropout.

- "keep-prob": probability to keep a unit is p

8.4 Dropout: Pseudo-code forward pass

```
def forward_pass(x, w_list, b_list):
    w1, w2, w3 = w_list
    b1, b2, b3 = b_list

    z1 = np.dot(x, w1.T) + b1
    h1 = np.maximum(0, z1)  # ReLU
    h1 *= np.random.binomial(1, 0.5, size=h1.shape)  # 50% dropout

    z2 = np.dot(h1, w2.T) + b2
    h2 = np.maximum(0, z2)
    h2 *= np.random.binomial(1, 0.5, size=h2.shape)  #50% dropout

    z3 = np.dot(h2, w3.T) + b3
    h3 = softmax(z3)
    return (h1, h2, h3)
```

8.4 Dropout: introduced bias

- Dropout should only be applied during training, but not for inference
- Network with large dropped parts (e.g. “training time”):

$$\tilde{s} = \boldsymbol{w}^T(\boldsymbol{d} \odot \boldsymbol{a})$$

- Network without dropped parts (e.g. “test time”):

$$s = \boldsymbol{w}^T \boldsymbol{a}$$

$$\mathbb{E}_{p_{\text{data}},d_i}(\tilde{s}) = \mathbb{E}_{p_{\text{data}},d_i}\left(\sum_i w_i d_i a_i\right) = p\sum_i w_i \,\mathbb{E}_{p_{\text{data}}}(a_i)$$

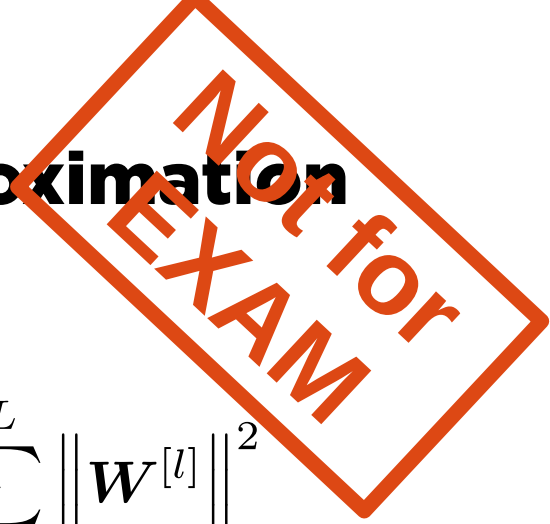
$$\mathbb{E}_{p_{\text{data}}}(s) = \sum_i w_i \,\mathbb{E}_{p_{\text{data}}}(a_i)$$

8.4 Dropout: bias countering strategies

Strategies to counter this bias:

- Weight scaling:
  - After training, weights are rescaled by p to remove the bias
- State scaling:
  - During training, activations are re-scaled by $1/p$
  - Advantage: network can be left as it is; the dropout layer has to have two modes: training and test mode. In training mode, outgoing values have to be rescaled.
- Monte-Carlo-Dropout:
  - Multiple forward-passes with dropout are averaged

8.4 Dropout: DO as Bayesian approximation



- Objective with dropout regularization:

$$R_{\text{Dropout}-\text{reg}}(\boldsymbol{w}) := \frac{1}{N}\sum_{n=1}^{N} L(y_n, \hat{y}_n) + \lambda \sum_{l=1}^{L} \left\| \boldsymbol{W}^{[l]} \right\|^2$$

  where $\hat{y}_n$ are the predictions of a network that applies dropout.

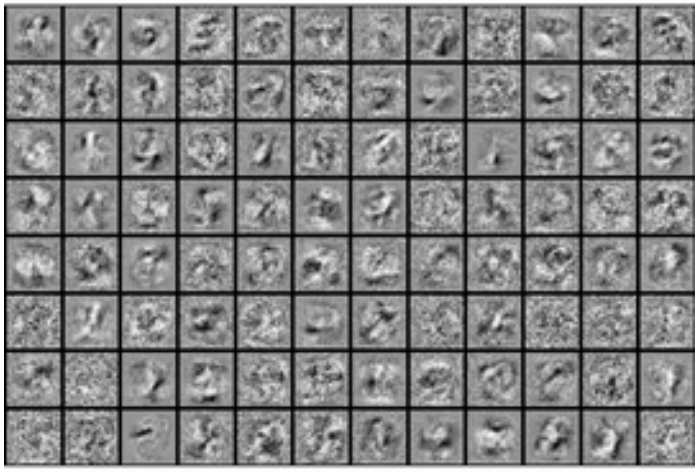
- In a Bayesian setting with distributions over parameters, and Gaussian process approximation, we can obtain:

$$R_{\text{GP}-\text{MC}} \propto \frac{1}{N}\sum_{n=1}^{N} -\frac{1}{\tau} \log p(y_n | \boldsymbol{x}_n, \hat{\boldsymbol{\omega}}_n) + \sum_{l=1}^{L} \frac{p_i l^2}{2\tau N} \|\boldsymbol{M}_i\|^2$$

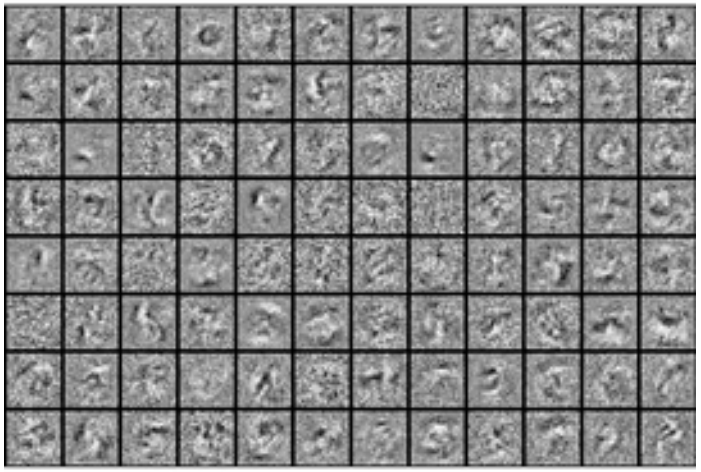
  From which the training objective above can be recovered with suitable choice of τ, l.

Gal, Y., & Ghahramani, Z. (2016, June). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In international conference on machine learning (pp. 1050-1059).

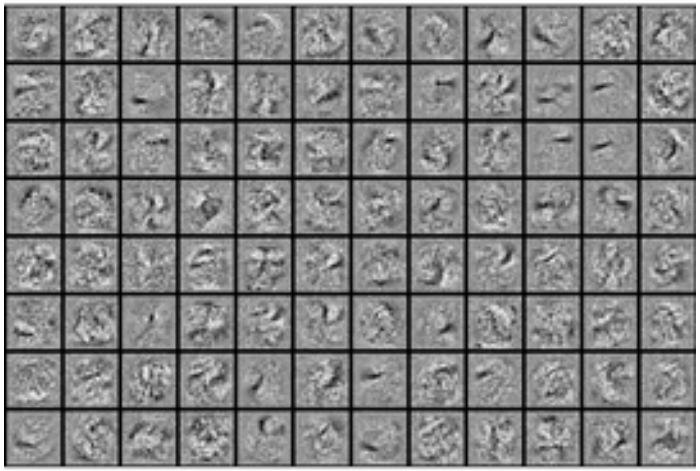
8.4 Dropout: Effects of dropout



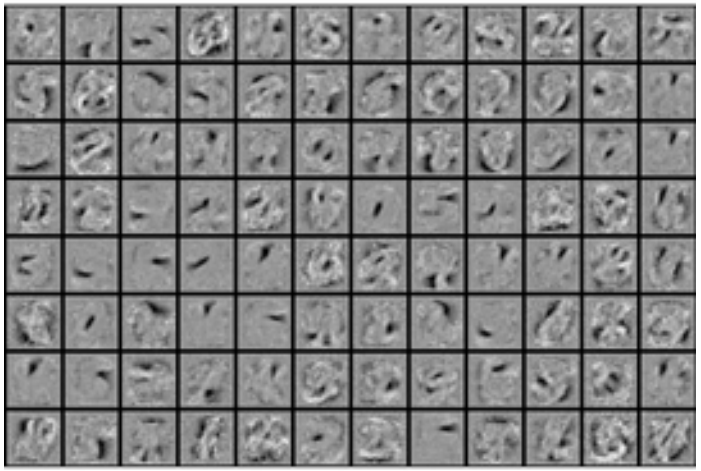
a) Sigmoid activations



a) ReLU activations

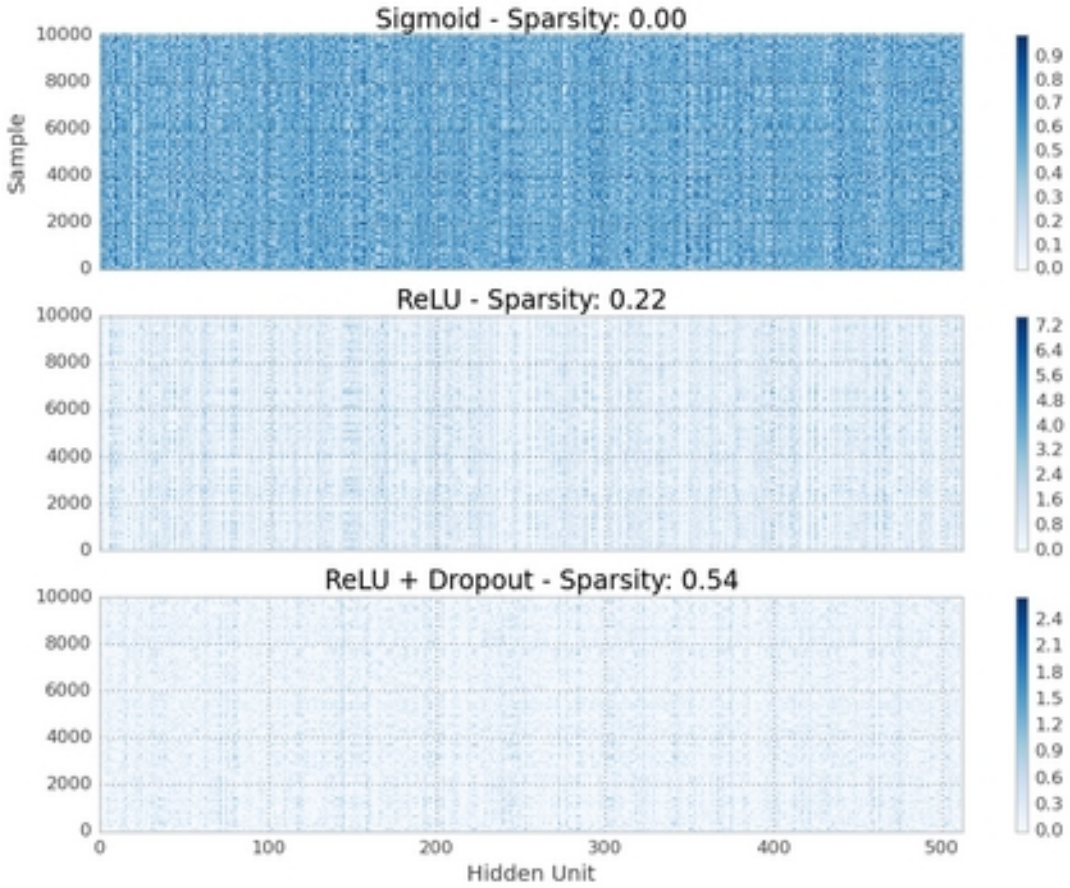


c) ReLU activations, DO



d) ReLU activations, DO, input DO

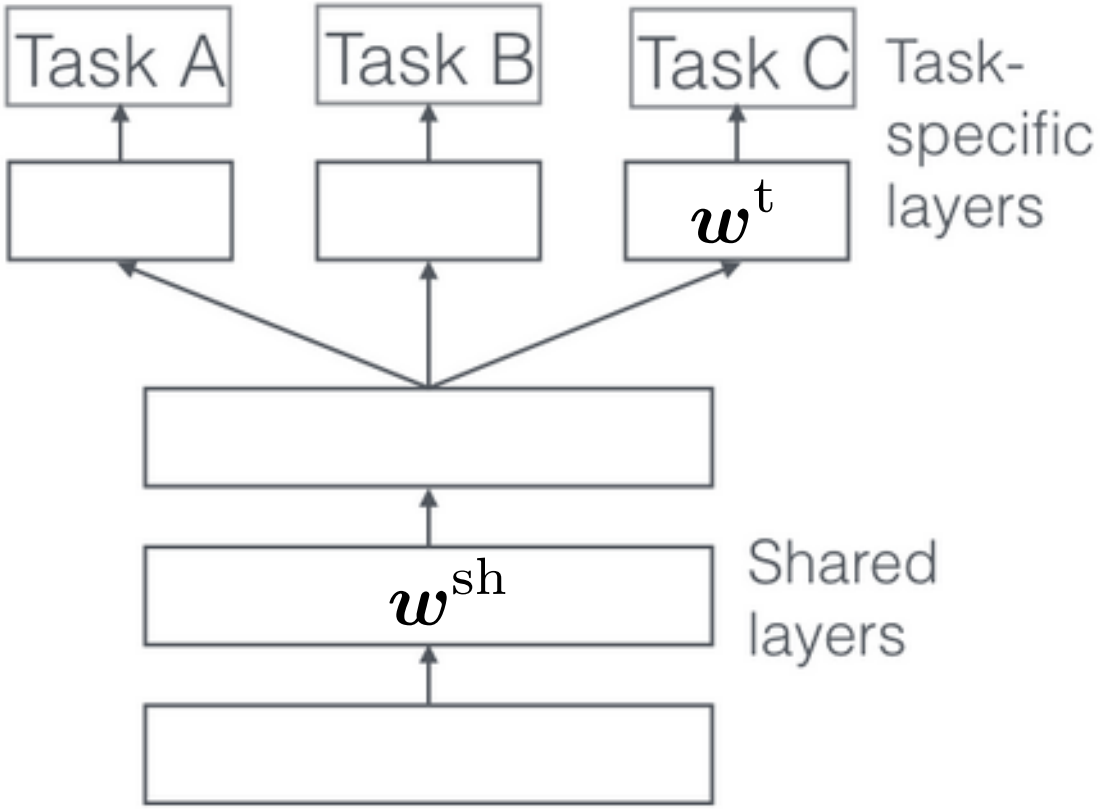
8.4 Dropout: Effects of dropout on sparsity



8.5 Multi-task Learning

- Main idea: Solve multiple prediction tasks with a single network with multiple output units
- Multi-task Learning is an approach to inductive transfer that improves generalization by using the domain information contained in the training signals of related tasks as an inductive bias. It does this by learning tasks in parallel while using a shared representation; what is learned for each task can help other tasks be learned better. (Caruana, 1997)
- Multi-task setting: $g(\boldsymbol{x}, \boldsymbol{w}^{\mathrm{sh}}, \boldsymbol{w}^{\mathrm{t}}) : \mathcal{X} \mapsto \mathcal{Y}^t$
- For each task there can be a task specific loss function $L^t(\cdot, \cdot)$.

8.5 Multi-task learning



8.5 Multi-task Learning

- With the usual empirical risk minimization principle the multi-task objective has following form:

$$\min_{\boldsymbol{w}^{\mathrm{sh}},\boldsymbol{w}^{1},\ldots,\boldsymbol{w}^{\mathrm{t}},\ldots,\boldsymbol{w}^{\mathrm{T}}} \quad \sum_{t=1}^{T} \alpha^{t} R^{t}_{\mathrm{emp}}(\boldsymbol{w}^{\mathrm{sh}}, \boldsymbol{w}^{\mathrm{t}})$$

- Assuming we are mainly interested in task $t = 1$, one can use all other tasks as auxiliary tasks.
- The auxiliary tasks act as regularization for task $t = 1$ via their influence on $\boldsymbol{w}^{\mathrm{sh}}$.
- Weighting of tasks can be learned or hyperparameters

8.6 Growing

Main idea:

- Start with a small network
- Add new units in a step-wise way or new weights which amount to increasing the complexity

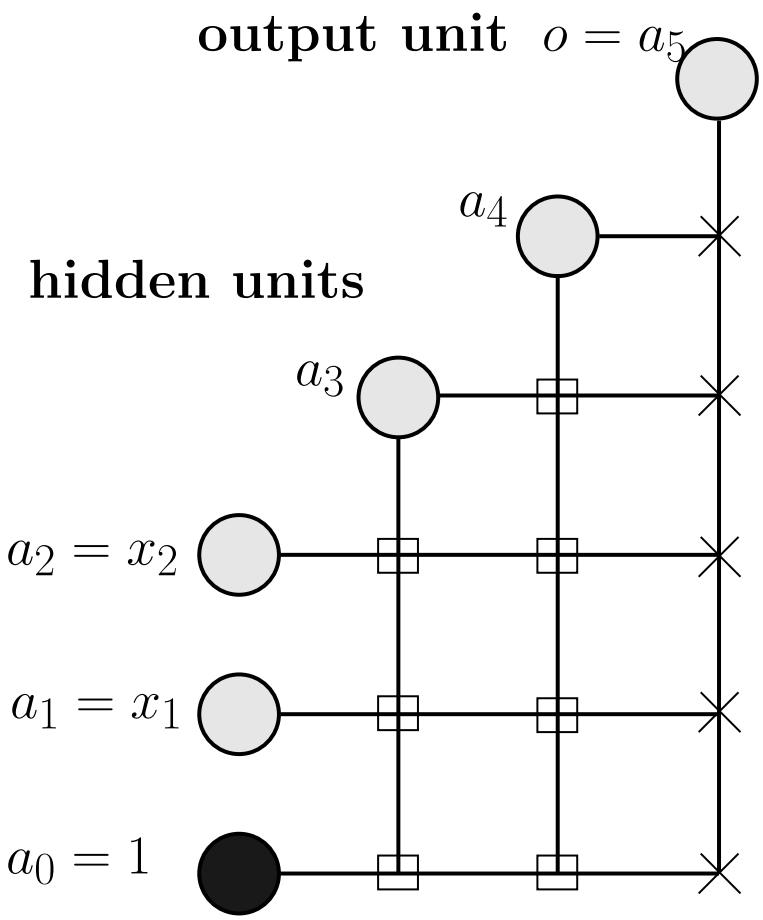
8.6 Growing – Cascade Correlation

- Start: Single layer network
- Add sequentially new hidden neurons
- These new hidden neurons have incoming connections from all existing input neurons and hidden neurons.
- These connections are adjusted such that the **correlation between the new unit's activation and the output errors is maximal**.
- Units that correlate with the error signal can potentially help the whole network to decrease the overall error.

8.6 Growing – Cascade Correlation

Architecture of the network:

- Squares are weights which are trained.
- Crosses represent weights which are retrained only after the addition of a hidden unit.



8.6 Growing – Cascade Correlation

Objective to maximize (correlation with error):

$$C_\xi = \sum_{k=1}^{K} \left| \sum_{n=1}^{N} \left(a_\xi^n - \bar{a_\xi}\right) \left(\epsilon_k^n - \bar{\epsilon}_k\right) \right|$$

$$\epsilon_k^n = \left(g(\boldsymbol{x}^n; \boldsymbol{w}) - y_k^n\right)$$

The new hidden unit ξ gets input from all input neurons and hidden neurons that currently exits:

$$s_\xi = \sum_{j=1}^{J} w_{\xi j} a_j$$

8.6 Growing – Cascade Correlation

Derivative of C_ξ with respect to the weights $w_{\xi i}$:

$$\frac{\partial C_\xi}{\partial w_{\xi j}} = \pm \sum_{k=1}^{K} \sum_{n=1}^{N} (\epsilon_k^n - \bar{\epsilon}_k) \; f'(s_\xi) \; a_j^n$$

The training of cascade-correlation networks is very fast because

- only the incoming connections to the new hidden units have to be trained.
- the hidden to output weights are then found by training methods for single layer networks

8.7 Pruning Methods

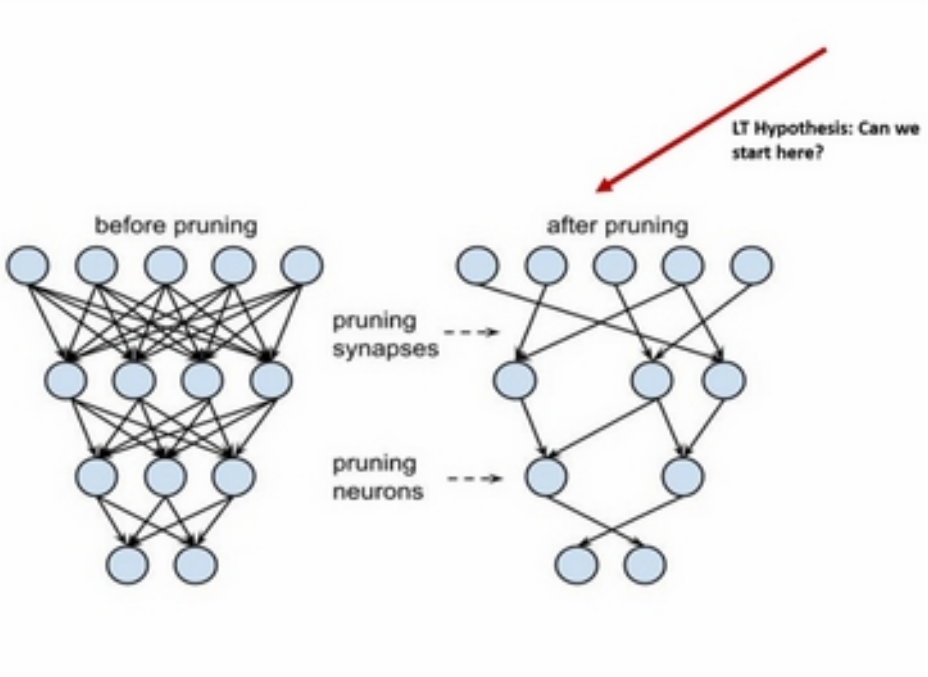
Main idea:

- First train neural network until a local minimum of the training error is found
- Then the complexity of the trained network is reduced by removing weights (setting them to zero) from the network.
- Simple approach: **Pruning away low weights**
  - Remove p% of lowest weights in each layer.
  - Often results in improved accuracy
  - Smaller networks, faster calculation; e.g. for NN on mobile devices

8.7 Pruning Methods: Lottery ticket hypothesis

- **Observation:**
  - 90% of a trained network pruned away with simple technique → accuracy remains high
  - The 10% remaining connections re-set to initial weights → accuracy still high

- **“lottery ticket hypothesis":**
  - dense, randomly-initialized, feed-forward networks contain subnetworks ("winning tickets") that - when trained in isolation - reach test accuracy comparable to the original network in a similar number of iterations.
  - The winning tickets “have won the initialization lottery”: their connections have initial weights that make training particularly effective.



Frankle, J., & Carbin, M. (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635.

8.7 Pruning Low-Influence Weights

- A Taylor expansion $R(\boldsymbol{w})$ of the empirical error around the local minimum $\boldsymbol{w}^*$ is made.
- The gradient vanishes in the minimum $\nabla_{\boldsymbol{w}} R(\boldsymbol{w}^*) \; = \; \mathbf{0}$, therefore we obtain with $\Delta \boldsymbol{w} \; = \; (\boldsymbol{w} \; - \; \boldsymbol{w}^*)$:

$$R(\boldsymbol{w}) \; = \; R(\boldsymbol{w}^*) \; + \; \frac{1}{2} \Delta \boldsymbol{w}^T \boldsymbol{H}(\boldsymbol{w}^*) \; \Delta \boldsymbol{w} \; + \; O\left((\Delta \boldsymbol{w})^3 \right)$$

- To delete the weight w_i we have to ensure:

$$\boldsymbol{e}_i^T \; \Delta \boldsymbol{w} \; + \; w_i \; = \; 0$$

8.7 Pruning: “Optimal Brain Damage” (OBD)

- The error increase is largely determined by (neglecting cross-terms):

$$\sum_i H_{ii} \ (\Delta w_i)^2$$

- If we want to remove a weight then this term should be as small as possible. At the minimum the Hessian is positive definite, which means that for all $\boldsymbol{v}$

$$\boldsymbol{v}^T \boldsymbol{H} \ \boldsymbol{v} \ \geq \ 0 \ .$$

- Therefore: $\boldsymbol{e}_i^T \boldsymbol{H} \ \boldsymbol{e}_i \ = \ H_{ii} \ \geq \ 0$
- Therefore we choose $\Delta \boldsymbol{w} \ = \ - \ w_i \ \boldsymbol{e}_i .$
- Minimal error increase: $\min_i H_{ii} \ w_i^2$
- Removed weight: w_k with $k \ = \ \arg\min_i H_{ii} \ w_i^2$

8.7 Pruning: “Optimal Brain Surgeon” (OBS)

- The Taylor expansion and the constraint of removing weight i can be stated as a quadratic optimization problem

$$\min_{\Delta \boldsymbol{w}} \quad \frac{1}{2} \Delta \boldsymbol{w}^T \boldsymbol{H} \; \Delta \boldsymbol{w}$$

$$\text{s.t.} \quad \boldsymbol{e}_i^T \; \Delta \boldsymbol{w} \; + \; w_i \; = \; 0$$

- The **Lagrangian** is

$$L \; = \; \frac{1}{2} \; \Delta \boldsymbol{w}^T \boldsymbol{H} \; \Delta \boldsymbol{w} \; + \; \alpha \left(\boldsymbol{e}_i^T \; \Delta \boldsymbol{w} \; + \; w_i \right).$$

- The derivative has to be zero

$$\frac{\partial L}{\partial \Delta \boldsymbol{w}} \; = \; \boldsymbol{H} \; \Delta \boldsymbol{w} \; + \; \alpha \; \boldsymbol{e}_i \; = \mathbf{0}.$$

- So:

$$\Delta \boldsymbol{w} \; = \; - \; \alpha \; \boldsymbol{H}^{-1} \; \boldsymbol{e}_i$$

8.7 Pruning: “Optimal Brain Surgeon” (OBS)

Further we have

$$w_i \;=\; -\,\boldsymbol{e}_i^T\;\Delta\boldsymbol{w} \;=\; \alpha\;\boldsymbol{e}_i^T\boldsymbol{H}^{-1}\;\boldsymbol{e}_i \;=\; \alpha\;\boldsymbol{H}_{ii}^{-1},$$

which gives

$$\alpha \;=\; \frac{w_i}{\boldsymbol{H}_{ii}^{-1}}$$

and results in

$$\Delta\boldsymbol{w} \;=\; -\;\frac{w_i}{\boldsymbol{H}_{ii}^{-1}}\;\boldsymbol{H}^{-1}\;\boldsymbol{e}_i\;.$$

The objective becomes (using the above expression):

$$\frac{1}{2}\;\Delta\boldsymbol{w}^T\boldsymbol{H}\;\Delta\boldsymbol{w} \;=\; \frac{1}{2}\frac{w_i^2}{\boldsymbol{H}_{ii}^{-1}}.$$

8.7 Pruning: “Optimal Brain Surgeon” (OBS)

The criterion for selecting a weight to remove is the

$$\frac{1}{2}\frac{w_i^2}{\boldsymbol{H}_{ii}^{-1}}$$

and the correction is done by

$$\Delta \boldsymbol{w} \;=\; -\;\frac{w_i}{\boldsymbol{H}_{ii}^{-1}}\;\boldsymbol{H}^{-1}\;\boldsymbol{e}_i\;.$$

8.7 Pruning: “Optimal Brain Surgeon” (OBS)

Heuristics to improve OBS:

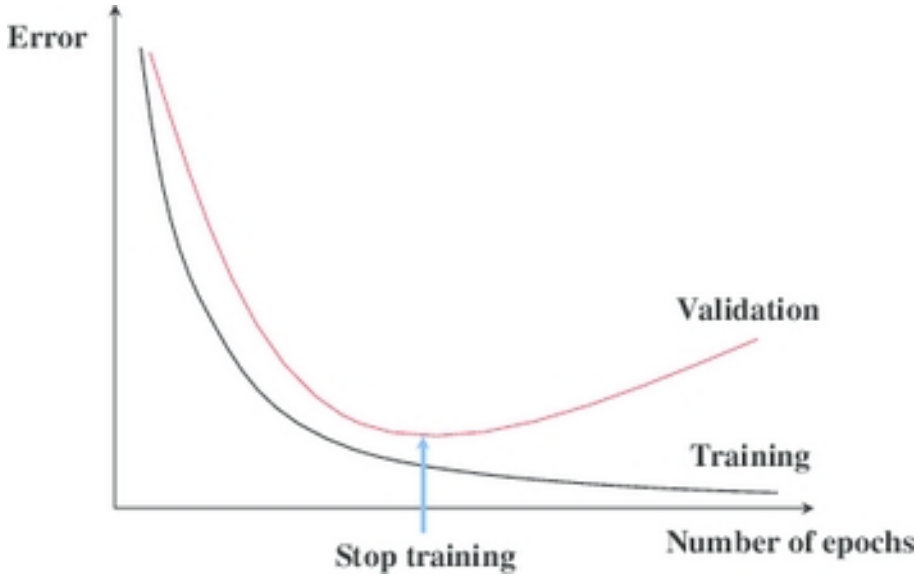
- Checking for the first order derivative $\nabla_{\boldsymbol{w}} R(\boldsymbol{w}^*) \; = \mathbf{0}$
- Retraining after weight deletion
- Checking for $\left\| \boldsymbol{I} \; - \; \boldsymbol{H}^{-1} \; \boldsymbol{H} \right\| \; > \; \beta$ to see the quality of the approximation of the inverse Hessian
- Check for weight changes larger than γ (e.g. $\gamma = 0.5$)

8.8 Early Stopping

Main idea:

Stop learning before the minimum is reached

- During learning the network will first extract the most dominant rules, that is the most important structures in the data.
- Later in the learning procedure characteristics are found which apply only to a few or just one example.



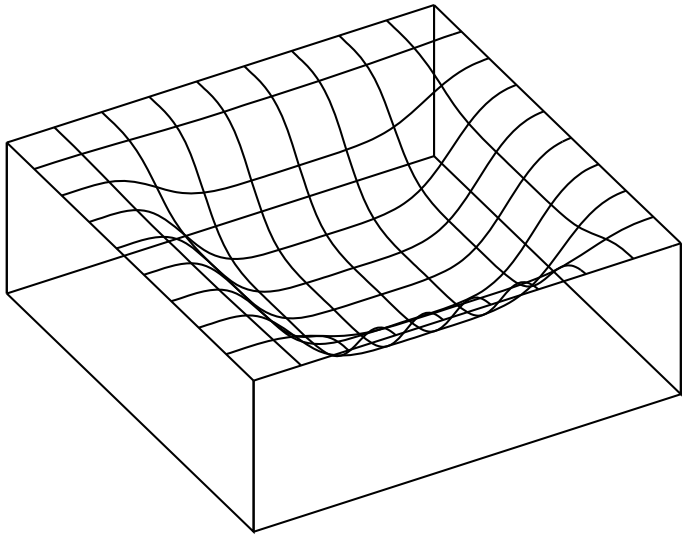
Drawbacks:

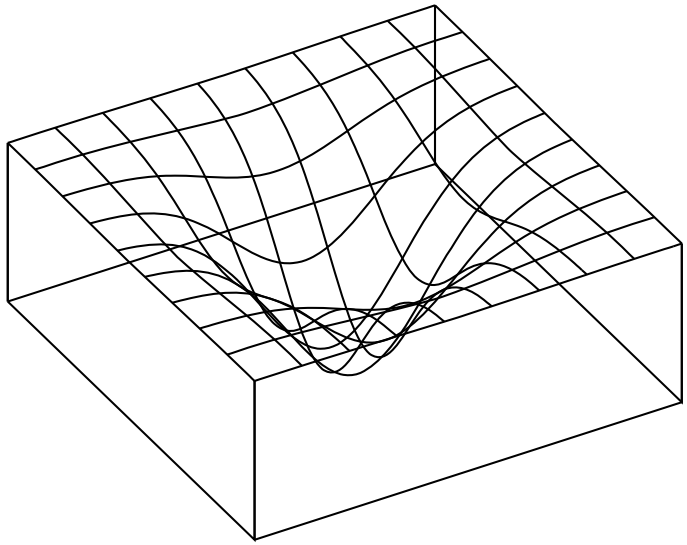
- Variability of early-stopping parameter is often high
- Strategy for using the validation set needed

8.9 Flat Minimum Search

Main idea:

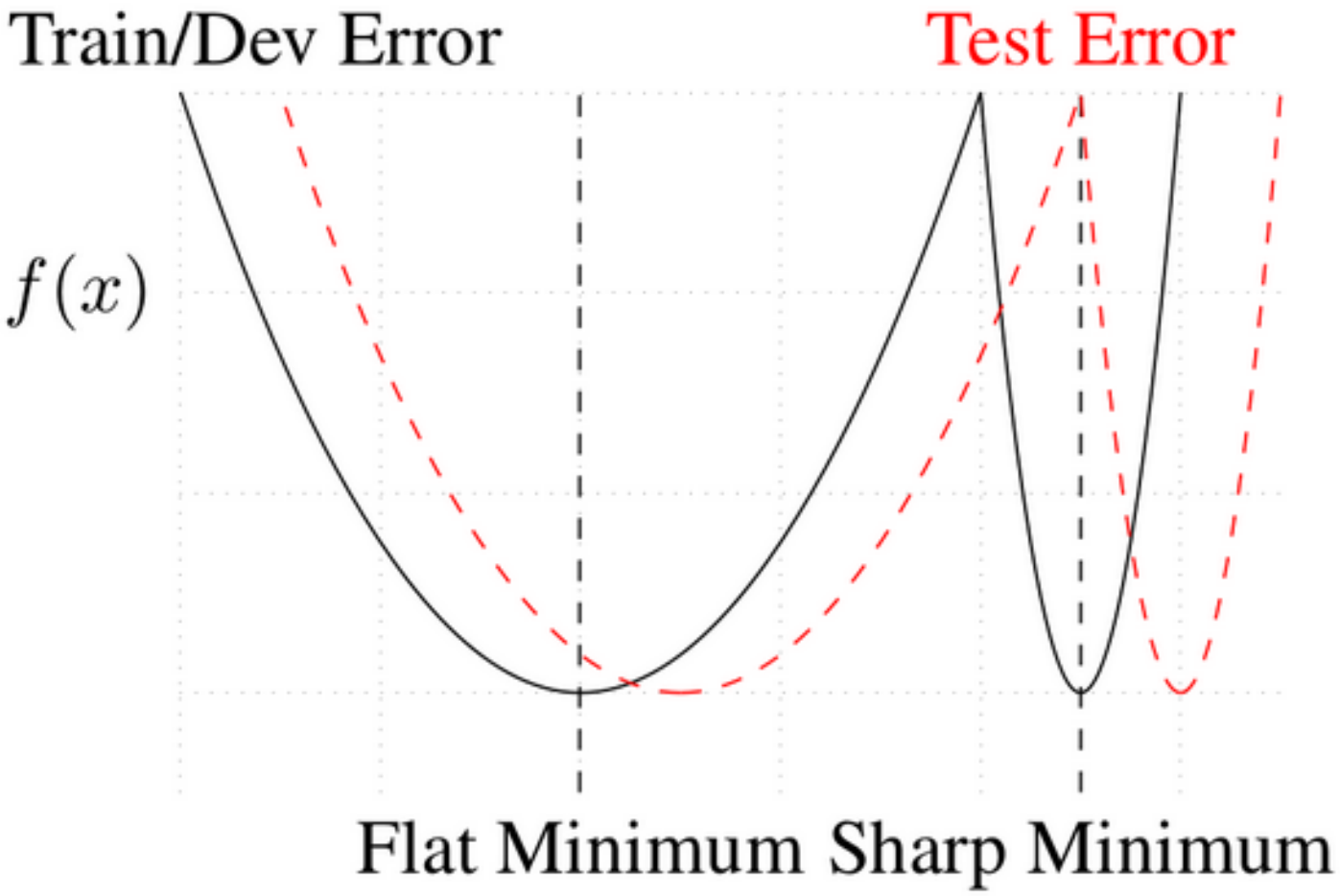
FMS searches for large regions in the weight space where the network function does not change but the empirical risk is small.





Hochreiter, S., & Schmidhuber, J. (1997). Flat minima. Neural Computation, 9(1), 1-42.

8.9 Flat Minimum Search

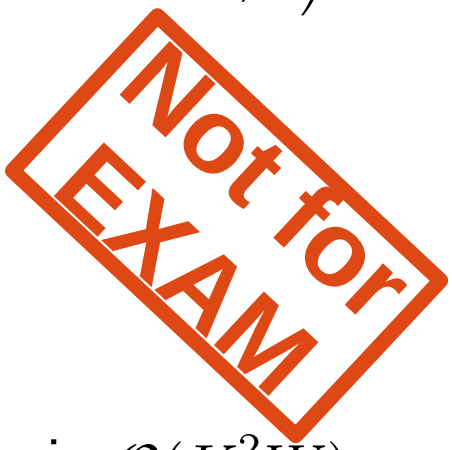


8.9 Flat Minimum Search

Flat minimum search adds an additional error term $\Omega(\boldsymbol{w})$ to the objective function $R(\boldsymbol{w})$. This error term $\Omega(\boldsymbol{w})$ describes the local flatness and has to be minimized.

$$\Omega(\boldsymbol{w}) = \frac{1}{2}\left(-W\log\epsilon + \sum_{i,j,l}\log\sum_{k=1}^{K}\left(\frac{\partial\hat{y}_k}{\partial w_{ij}^{[l]}}\right)^2 + W\log\sum_{k=1}^{K}\left(\sum_{i,j,l}\frac{\left|\frac{\partial\hat{y}_k}{\partial w_{ij}^{[l]}}\right|}{\sqrt{\sum_{k=1}^{K}(\frac{\partial\hat{y}_k}{\partial w_{ij}^{[l]}})^2}}\right)^2\right)$$

$$\frac{\partial\Omega(\boldsymbol{w})}{\partial w_{uv}} = \sum_{k=1}^{K}\sum_{i,j,l}\frac{\partial\Omega(\boldsymbol{w})}{\partial\left(\frac{\partial\hat{y}_k}{\partial w_{ij}^{[l]}}\right)}\frac{\partial^2\hat{y}_k}{\partial w_{ij}^{[l]}\,\partial w_{uv}^{[l]}}$$



- $\boldsymbol{H}^k$ is the Hessian of output unit k . The complexity is $\mathcal{O}(K^2W)$

8.10 Data Augmentation

- Main idea: Increase the size of the training set
- Empirical error without data augmentation:

$$R_{\text{emp}}(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} L(y^n, g(\boldsymbol{x}^n; \boldsymbol{w}))$$

- Empirical error with data augmentation:

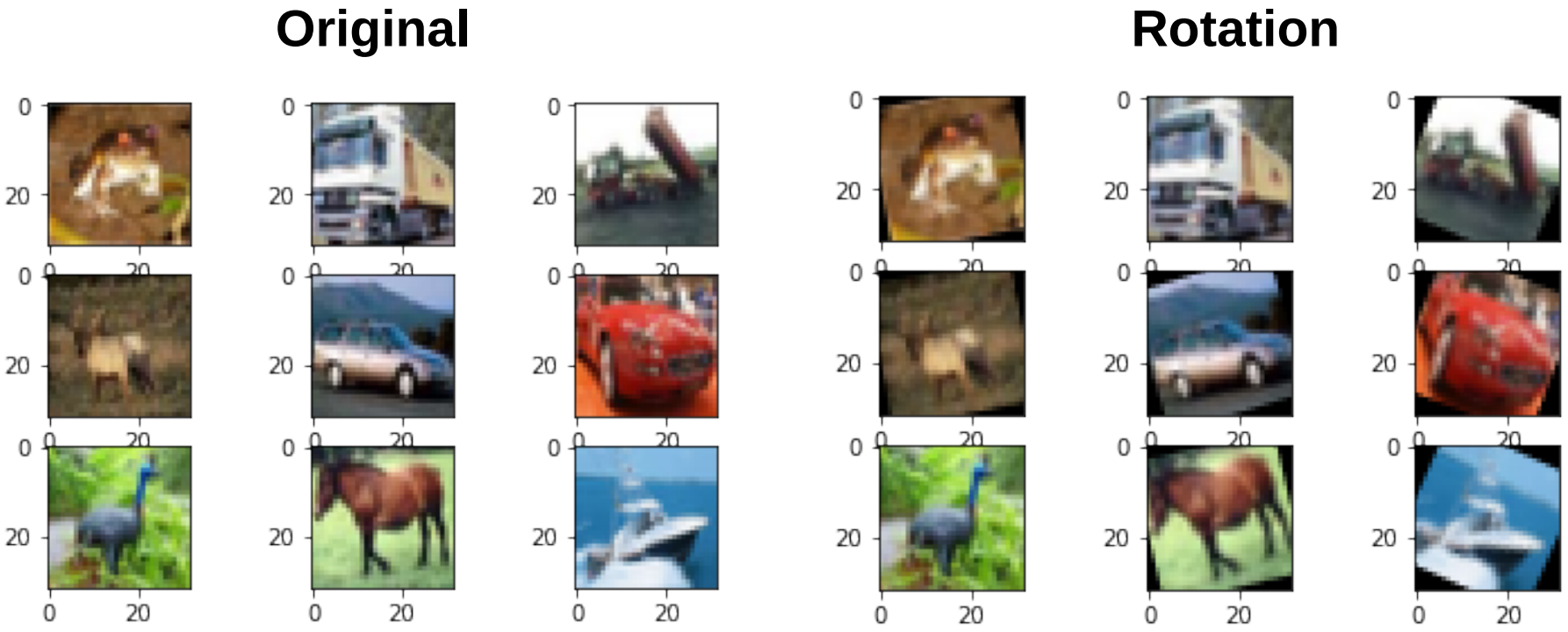
$$R_{\text{reg}}(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} L(y^n, g(\boldsymbol{x}^n; \boldsymbol{w})) + \underbrace{\frac{1}{L} \sum_{l=1}^{L} L(y^l, g(\boldsymbol{x}^l; \boldsymbol{w}))}_{\text{error on augmented data}}$$

8.10 Data Augmentation

Strategies for image recognition tasks:

- Rotation
- Translation
- Cropping
- Brightness noise
- Shearing

8.10 Data Augmentation



8.10 Data Augmentation

Original

Translation

8.10 Data Augmentation

Original

Cropping

8.10 Data Augmentation

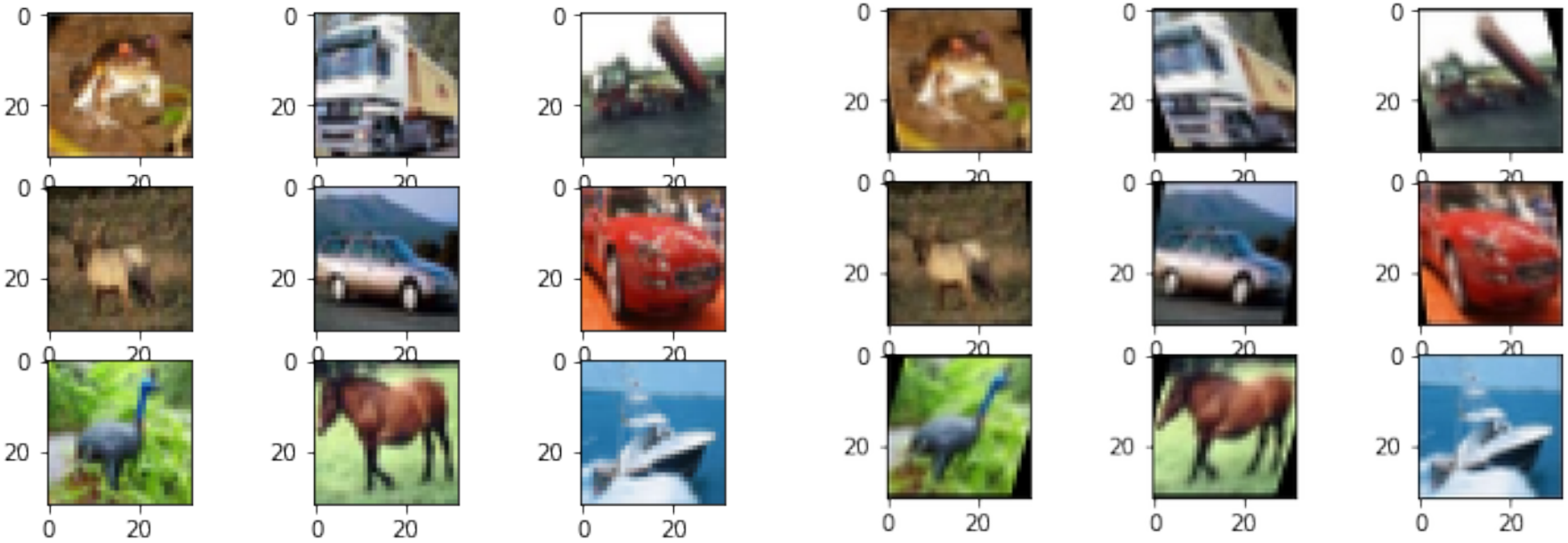
Original

Brightness noise

8.10 Data Augmentation

Original

Shearing



8.11 Self-regularization by SGD

- SGD minimizes an average potential over the posterior distribution of weights along with an entropic regularization term
- The so called Fokker-Planck equations describe the evolution of the system and can be used to provide an expression of the steady-state-distribution ρ^{ss}.
- This steady-state distribution over the weights estimated by the SGD is given by:

$$\rho^{\mathrm{ss}} = \mathrm{argmin}_{\rho}\, \mathbb{E}_{\boldsymbol{w}\sim\rho}\left(\Phi(\boldsymbol{w})\right) - \frac{\eta}{2b} H(\rho)$$

H : Entropy of the system

$\Phi(\boldsymbol{w})$: Original empirical error function

b : Batch size

η : Learning rate

8.11 Self-regularization by SGD

Consequences:

- SGD does not find minima of the original objective function.
- The deviation scales linearly with $\beta^{-1} = \eta/(2b)$.
- Large learning rates and small batch sizes have a regularization effect on learning since $\beta^{-1} = \frac{\eta}{2b}$.
- The learning rate should linearly scale with batch size.

Chaudhari, P., & Soatto, S. (2018, February). Stochastic gradient descent performs variational inference, converges to limit cycles for deep networks. In 2018 Information Theory and Applications Workshop (ITA) (pp. 1-10). IEEE.

Example for exam questions (1/2)

As described in the lecture, for single-layer neural networks with quadratic loss, we have the following objective function $R(b) = \frac{1}{2} \sum_{n=1}^{N} (y^n - (\mathbf{w}^T \mathbf{x}^n + b))^2$, where we made the bias parameter b explicit. Which bias parameter optimizes the objective function assuming fixed $\mathbf{w}$?

Select one:

- a. $b = \frac{1}{2N} \sum_{n=1}^{N} (y^n - (\mathbf{w}^T \mathbf{x}^n))$
- b. $b = \frac{1}{N} \sum_{n=1}^{N} (y^n - (\mathbf{w}^T \mathbf{x}^n))$
- c. $b = \sum_{n=1}^{N} (y^n - (\mathbf{w}^T \mathbf{x}^n))$
- d. $b = \frac{2}{N} \sum_{n=1}^{N} (y^n - (\mathbf{w}^T \mathbf{x}^n))$
- e. $b = \frac{1}{N} \sum_{n=1}^{N} (x^n - (\mathbf{w}^T \mathbf{y}^n))$

Example for exam questions (2/2)

For training a machine learning model $g(\mathbf{x}; \mathbf{w})$ with stochastic gradient descent, the important quantity to obtain updates is the gradient of the loss function $\mathcal{L}$ with respect to the inputs $\mathbf{x}$: $\nabla_x \mathcal{L}(y, g(\mathbf{x}; \mathbf{w}))$, True or False?

Select one:

- a. False, the important quantity is the gradient of the weights w.r.t. the inputs
- b. True, the weights of the model are updated with the gradient of the loss function w.r.t. the input
- c. True, the gradient w.r.t. the inputs determine the direction in which to go on the loss function to minimize the loss
- d. False, the important quantity is $\nabla_w \mathcal{L}(y, g(\mathbf{x}; \mathbf{w}))$
- e. False, the update is the loss scaled by the learning rate