Deep Learning

4DV661

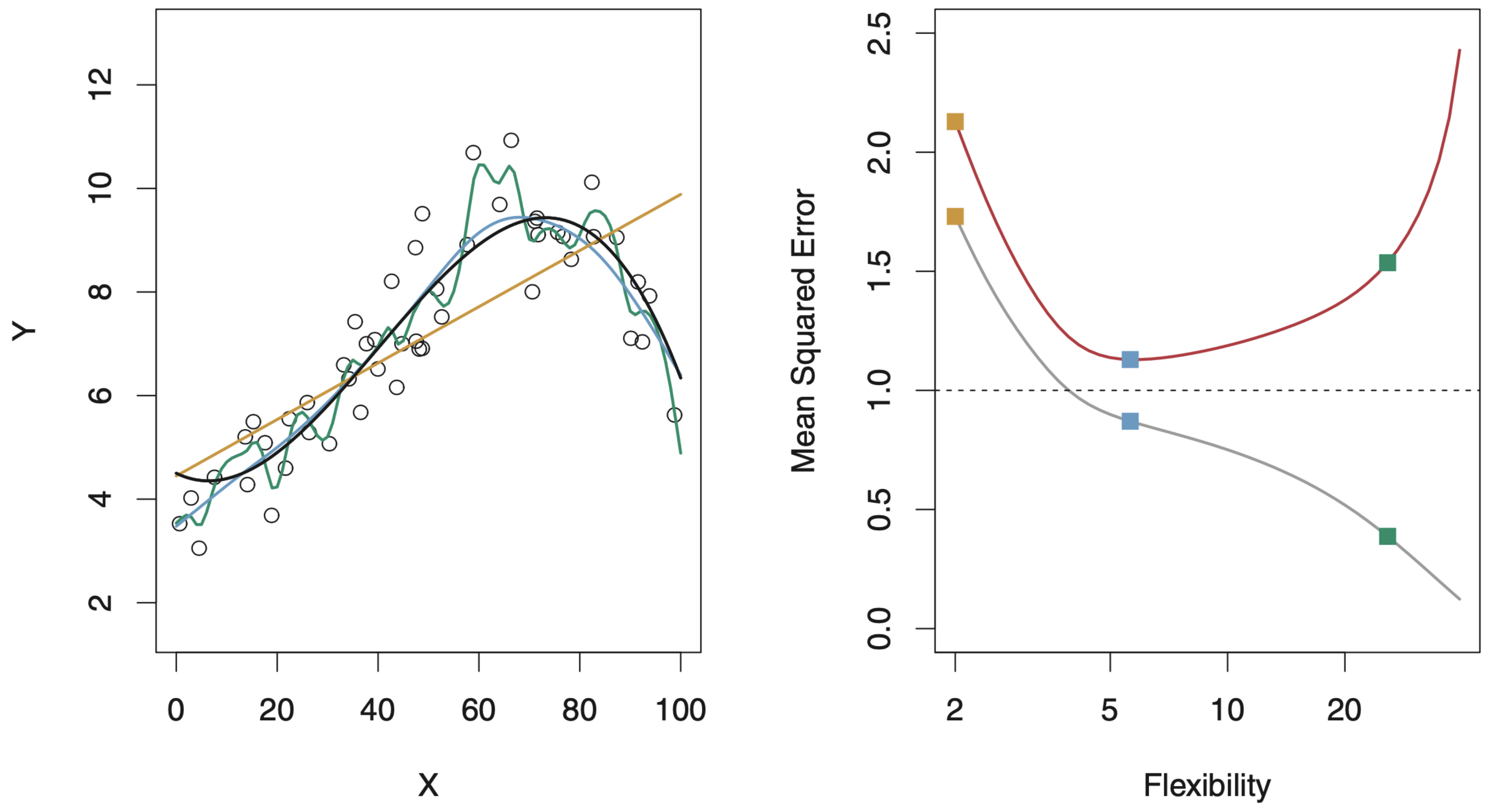
Regularization for Deep Learning

Welf Löwe

Course structure

1. Introduction
2. Applied Math Basics
3. Deep Feedforward Networks
4. Regularization for Deep Learning
5. Optimization for Training Deep Models
6. Convolutional Networks
7. Sequence Modeling: Recurrent and Recursive Nets
8. Practical Methodology
9. Applications

Problem of over-fitting



The Bias-Variance trade-off

- Variance of $\hat{f}$ refers to the amount by which $\hat{f}$ would change if we estimated it using different training data sets.
- Bias of $\hat{f}$ refers to the error that is introduced by approximating/simplifying a real-life problem.
- For any value x_0 the expected (squared) error is

$$E\left(y_0 - \hat{f}(x_0)\right)^2 = \mathrm{Var}(\hat{f}(x_0)) + [\mathrm{Bias}(\hat{f}(x_0))]^2 + \mathrm{Var}(\epsilon)$$

  - $\mathrm{Var}(\hat{f}(x_0)) = E(\hat{f}(x_0)^2) - E(\hat{f}(x_0))^2$ (definition of Var)
  - $\mathrm{Bias}(\hat{f}(x_0)) = E(\hat{f}(x_0)) - E(f(x_0)) = E(\hat{f}(x_0)) - f(x_0)$ (f is deterministic but unknown)

- More flexible models result in less bias but higher variance and vice versa.

The Bias-Variance trade-off (cont'd)

- The optimum flexibility level corresponds to the minimum test error optimizing the matching of the real-world with the model
- The relative rate of change of variance and bias determines whether the test MSE increases or decreases with model flexibility
  - the squared bias and variance may change at different rates
  - and we don't know the test MSE

- What does that mean for deep learning?

Motivation of Regularization

- (Sufficiently complex) deep learning models have basically no bias (perfect learner)
- This high flexibility leads too a high risk of overfitting (high variance)
- Regularization mitigates this problem by
  - Restricting the model flexibility
  - Restricting the training process
  - Adding new data (augmentation, later in the course)

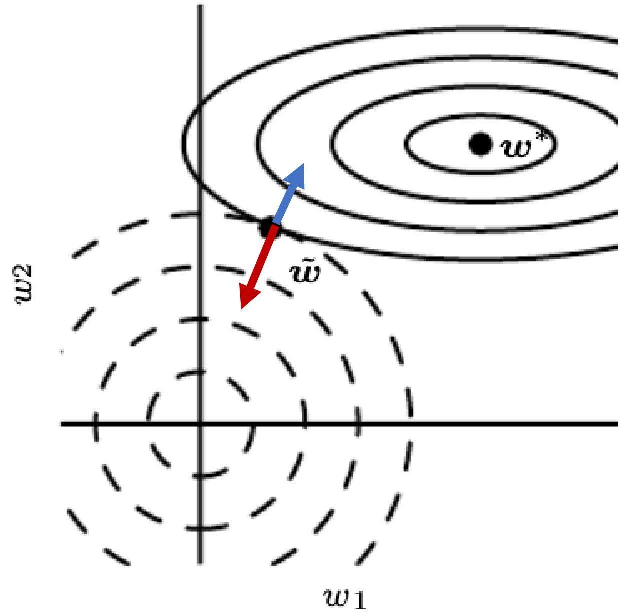
Agenda for today

- Parameter Norm Penalties
- Norm Penalties as Constraints
- Regularization and Under-Constrained Problems
- Dataset Augmentation
- Noise Robustness
- Semi-Supervised Learning
- Multitask Learning
- Early Stopping
- Bagging and Other Ensemble Methods
- Dropout
- Parameter Typing and Parameter Sharing
- Sparse Representations
- Adversarial Training
- Tangent Distance, Tangent Prop and Manifold, Tangent Classifier

Restricting the model flexibility

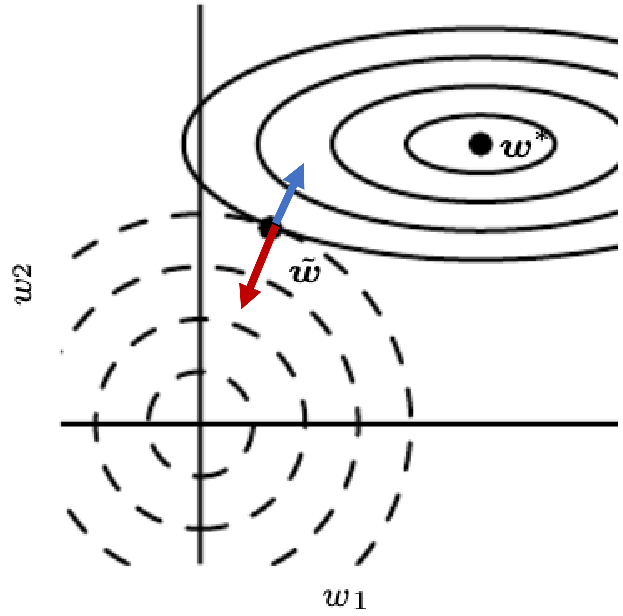
Restricting the training process

Parameter Norm Penalties



- The loss function has two components, i.e., the Original loss plus a Norm penalty function. We denote
  - the Original loss function $J(\boldsymbol{w})$ with $w*$ as the optimum
  - the Norm penalty function $\alpha\Omega(\boldsymbol{w})$ with Ω a norm of the weight vector and α a weight of this penalty
  - In the image, L2 norm, i.e., the squared length of the parameter vector
- Optimum $\tilde{w} \neq w*$ of the sum of both functions, i.e., $J(\boldsymbol{w})+\alpha\Omega(\boldsymbol{w})$ is minimum for $\boldsymbol{w}= \tilde{\boldsymbol{w}}$
  - $\nabla J(\tilde{\boldsymbol{w}}) \neq 0$, $J(\boldsymbol{w})$ gets smaller in the blue direction
  - $\nabla\Omega(\tilde{\boldsymbol{w}}) \neq 0$, $\Omega(\boldsymbol{w})$ gets smaller in the red direction
- Optimum can be found with gradient descent on the sum of both functions, i.e., $J(\boldsymbol{w})+\alpha\Omega(\boldsymbol{w})$
  - Iteratively adjust weights until $\nabla(J(\tilde{\boldsymbol{w}}) + \alpha\Omega(\tilde{\boldsymbol{w}})) \approx 0$

Norm Penalties as Constraints



- For a first try, substitute the penalty function $\alpha\Omega(\boldsymbol{w})$ with an equality constraint $\Omega(\boldsymbol{w}) = c$
  - So, we minimize $J(\boldsymbol{w})$ requiring the for norm $\Omega(\boldsymbol{w}) = c$
  - In the image, each dotted lines corresponds to the L2 norm with different values of c
- For the optimum $\widetilde{\boldsymbol{w}}$, the gradients of loss function $\nabla J(\widetilde{\boldsymbol{w}})$ and the norm function $\Omega(\boldsymbol{w})$ are parallel (both orthogonal to the tangent in $\widetilde{\boldsymbol{w}}$ on the contour of the loss and the norm functions)

$$\nabla J(\widetilde{\boldsymbol{w}}) = \lambda \nabla \Omega(\widetilde{\boldsymbol{w}}) \text{ and } \Omega(\widetilde{\boldsymbol{w}}) = c$$

- Parallel: there is only a scale factor λ of the gradients $\nabla J(\widetilde{\boldsymbol{w}})$ and $\nabla \Omega(\widetilde{\boldsymbol{w}})$ making these two vectors equivalent
  - Called the Lagrangian multiplier λ
  - Can be understood as a new variable to optimize for
- The condition can be formulated with the Lagrangian $\mathcal{L}$ and is gradient $\nabla \mathcal{L}$

$$\mathcal{L}(\boldsymbol{w}, \lambda) = J(\boldsymbol{w}) + \lambda(\Omega(\boldsymbol{w}) - c)$$
$$\nabla \mathcal{L}(\boldsymbol{w}, \lambda) = 0 \text{ and } \Omega(\widetilde{\boldsymbol{w}}) = c$$

- This looks like an unconstrained optimization problem. However, the solution is on saddle points of the Lagrangian, hence, gradient descent does not apply directly
- Still, we can transform the problem such that the solution is a minimum of the transformed problem, e.g., minimize the (squared) magnitude of the gradient of the Lagrangian (≥ 0).

Norm Penalties as Constraints (cont'd)

- We have transformed an equality-constrained minimization problem in an unconstrained optimization problem, but
- For regularization, we face an inequality constraint: $\Omega(\boldsymbol{w}) \leq c$
- This can easily be transformed into an equality constraint using a slack variable s (adding a new variable to optimize for)

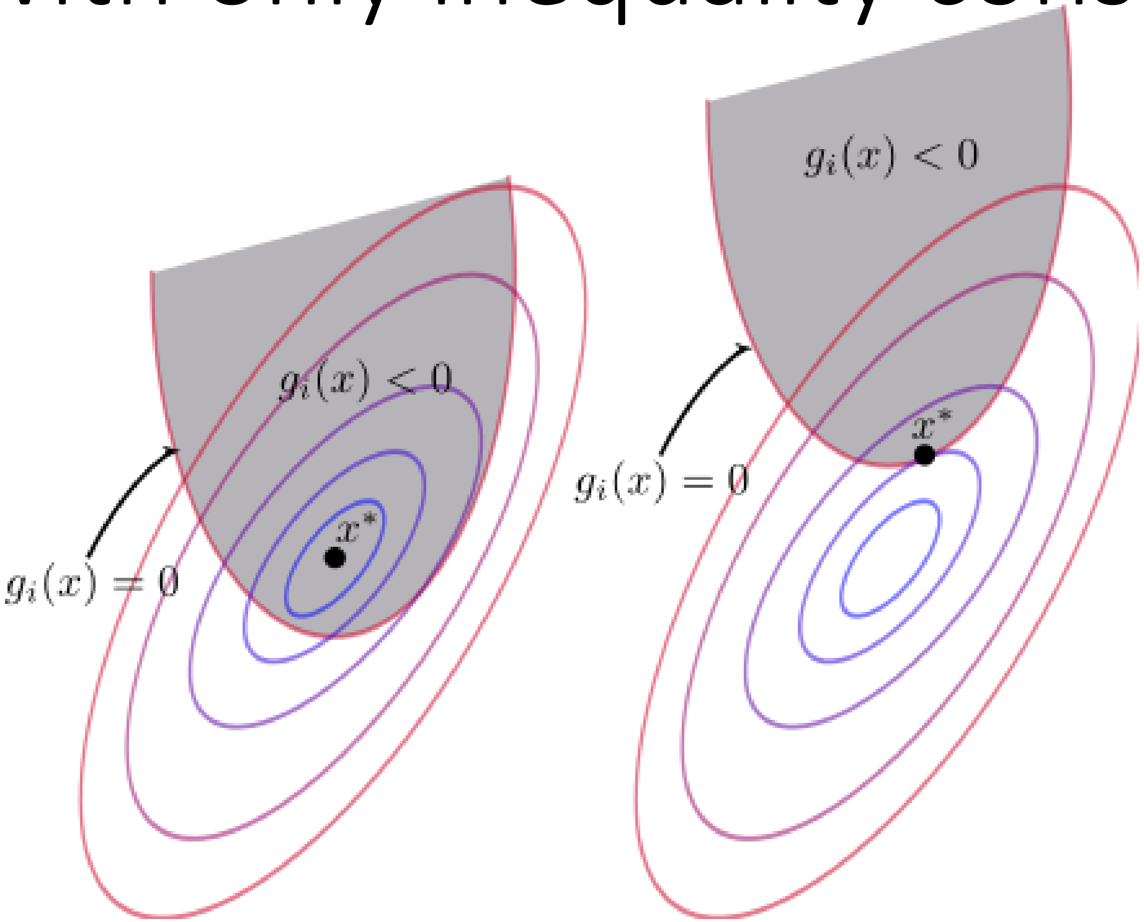
$$\Omega(\boldsymbol{w}) + s^2 = c.$$

- Changes the Lagrangian and the condition:

$$\mathcal{L}(\boldsymbol{w}, \lambda, s) = J(\boldsymbol{w}) + \lambda(\Omega(\boldsymbol{w}) + s^2 - c)$$
$$\nabla\, \mathcal{L}(\boldsymbol{w}, \lambda, s) = 0 \text{ and } \Omega(\boldsymbol{w}) + s^2 = c$$

- Solution by reduction to an unconstraint case and an equality constraint case disregarding the slack with the Karush–Kuhn–Tucker (KKT) approach

KKT cases with only inequality constraints $g(x)$



3a) Solution is equal to the unconstraint case (constraint is inactive)

3b) Solution is equal to the equality constraint case (constraint is active)

https://en.wikipedia.org/wiki/Karush%E2%80%93Kuhn%E2%80%93Tucker_conditions

Norm regularization in Tensorflow

Practical hints:

- Given a regression equation y=Wx+b, where x is the input, W the weights matrix and b the bias vector.
  - Kernel regularizer tries to reduce the weights W (excluding bias).
  - Bias regularizer tries to reduce the biases b.
  - Activity regularizer tries to reduce the layer's output y, thus, it will adjust the weights and the bias so Wx+b is constrained.
- If you have no idea about the function that you wish to model, only use the Kernel Regularizer
  - since a large enough network can still model your function even if the regularizations on the weights are big.
- If you want the output function to pass through (or have an intercept closer to) the origin, use the Bias Regularizer.
- If you want the output to be smaller (or closer to 0), use the Activity Regularizer.

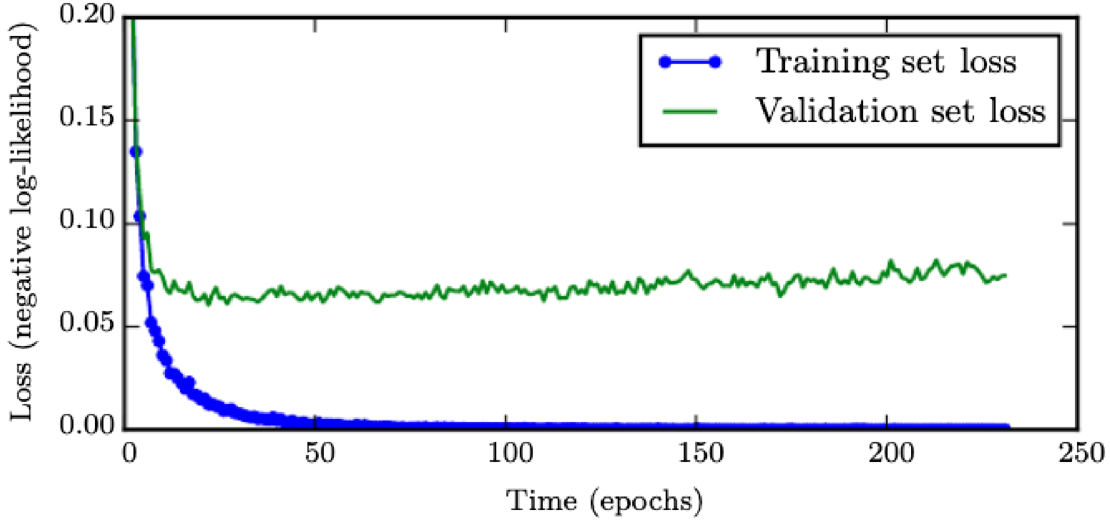
Norm Regularization in Tensorflow (cont'd)

- $L1$ versus $L2$ loss (not to be confused with the networks loss function).
  - $L2$ loss is defined as $|w|_2 = \sqrt{\sum(w_i)^2}$
  - $L1$ loss is defined as $|w|_1 = \sum w_i$.
- The gradient of
  - $L2$ will be: $2w$
    $L1$ will be: $sign(w)$
- In Norm Regularization, for each gradient update with a learning rate of ε and a weight decay of α, using $L2$ loss, the weights will be subtracted by $\varepsilon\alpha \cdot w$, while in $L1$ loss they will be subtracted by $\varepsilon\alpha \cdot sign(w)$.
- The effect of $L2$ loss on the weights is a reduction of large components in the weight matrix while $L1$ loss will make the weights matrix sparse, with many zero values.
  - The same applies on the bias and output respectively using the bias and activity regularizer.

Demo

- Notebooks https://github.com/WelfLowe/Public-ML-Notebooks/
- Check: Regularization of linear regression with norms and constraints
- For equality constraint optimization, check: Constraint Optimization (this is FYI about the approach; it is not a regularization method)

Early stopping



- Check training and validation loss under training
- Continue minimizing the training loss over the epochs
- However, keep the (parameters of the) model with the lowest validation loss
- Can be interpreted as hyper-parameter (number of epochs) learning

Early stopping (cont’d)

- Requires a validation set, which means some training data is not shown to the optimizer for deriving (gradients of) the model (parameters).
  - Yet the final model is not independent of validation data
  - Therefore, distinguish validation from test data
  - Use cross-validation and -testing
- Needs additional data, processing and memory recourses
- Otherwise, its an unobtrusive form of regularization
  - No negative effect
  - No changes to the optimization goal
  - Possibly used together with other regularization strategies

Bagging/Ensemble methods – idea

- Bagging (bootstrap aggregation): construct k different datasets
  - Each bag contains the same number of data point,
  - constructed by sampling with repetition from the original dataset,
    - not necessary a fair sample
    - you might, e.g., choose to oversample the minority class
  - with high probability, each bag is missing some of the examples (1/3 is OOB)
- Ensemble: construct k different models
  - train k different models separately,
  - different models will usually not make all the same errors on the test set
- Model aggregation: models' mode (classification) or mean (regression) sets the output for test examples
  - Stacking: aggregation by yet another ML model

Effect of Bagging/Ensemble methods

- Each of the k models (for the k bags) makes an error ϵ_i,
- Assume ϵ_i are drawn from a zero-mean multivariate normal distribution with
  - variances $Var[\epsilon_i]$ =$\mathrm{E}[\epsilon_i^2] = \frac{1}{k}\sum \epsilon_i^2 = v$ and covariances $Cov[\epsilon_i \epsilon_j] = \mathrm{E}[\epsilon_i \epsilon_j] = \frac{1}{k}\sum \epsilon_i \epsilon_j = c$.
- The expected (squared) error of the ensemble is $\frac{1}{k}\sum \epsilon_i$ and the expectation of the squared error is

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k^2}\mathbb{E}\left[\sum_i \left(\epsilon_i^2 + \sum_{j\neq i} \epsilon_i \epsilon_j\right)\right]$$
$$= \frac{1}{k}v + \frac{k-1}{k}c.$$

- Perfectly correlated errors between the models (c=v): Expectation of $MSE = v$, i.e., no effect.
- Perfectly uncorrelated errors between the models (c=0): Expectation of $MSE = \frac{1}{k}v$, i.e., the error is inverse proportional to the number of models.
- Truth somewhere in between

Proof hints

- Total variance:

$$\sum_{i=1}^{k}\sum_{j=1}^{k}\mathrm{Cov}(e_i,e_j)=\sum_{i=1}^{k}\mathrm{Var}(e_i)+\sum_{i\neq j}\mathrm{Cov}(e_i,e_j)=k\sigma^2+k(k-1)c.$$

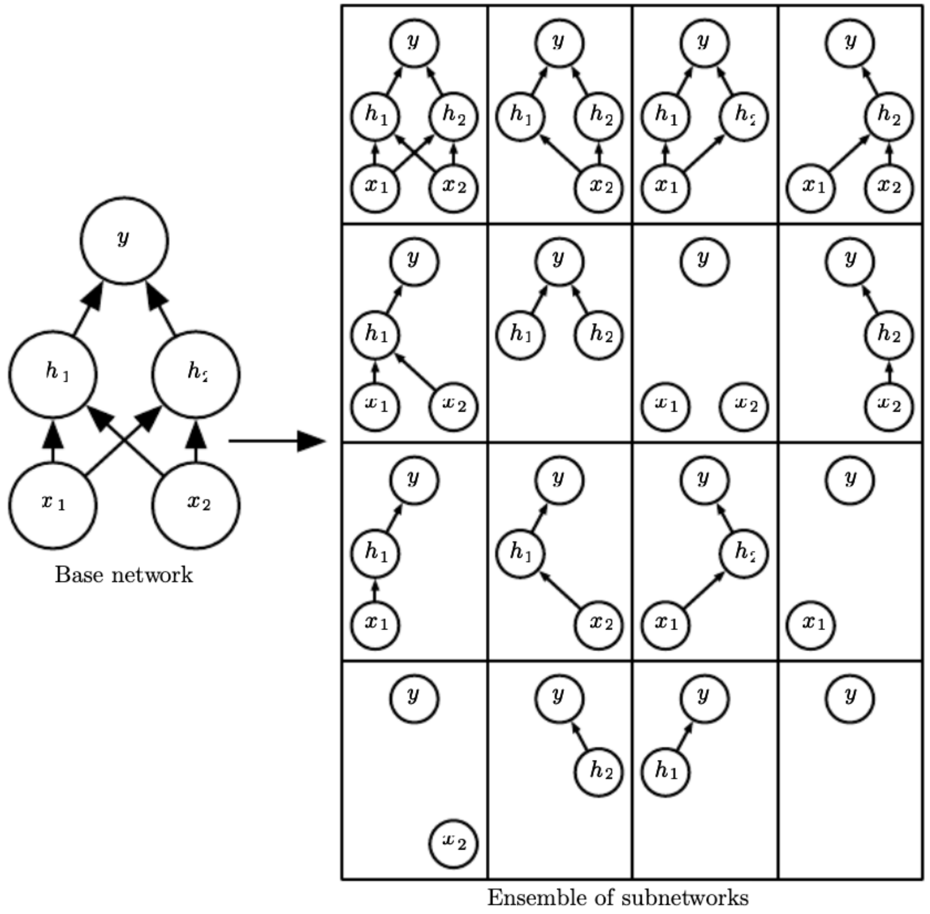
- Simplification of the expected squared error:

$$\left(\sum_i a_i\right)^2=\sum_i a_i^2+2\sum_{i<j}a_ia_j$$

Bagging in Deep Learning

- Bootstrap: choose mini batches, i.e., samples from the original dataset, in each epoch
  - As the name suggests: mini batches are smaller than the full training set
- Aggregation: take the minimum weights from the current epoch as initialization for the next epoch (optimized with a new mini batch)
  - However, models are not learned independently
  - Resemble boosting in this respect

Dropout



- Can be interpreted as a special ensemble method
- Mask vector μ specifies which units to include
- Cost function $J(\theta, \mu)$ of the cost of the model defined by parameters θ and mask μ.
- Dropout training minimize the expectation $\mathrm{E}_{\mu}[J(\theta, \mu)]$.

Dropout Analogy to Ensemble

- Ensemble accumulates the votes from all its members
- Assume that (each) model i outputs a probability distribution
- Ensemble averages these distributions

$$\frac{1}{k}\sum_{i=1}^{k} p^{(i)}(y \mid \boldsymbol{x})$$

- In the case of dropout, $p(\mu)$ is probability of mask μ at training time

$$\sum_{\boldsymbol{\mu}} p(\boldsymbol{\mu}) p(y \mid \boldsymbol{x}, \boldsymbol{\mu})$$

Assignment 4

- Read Chapter 8 (54 pages): Optimization for Training Deep Models
- Check out, understand, and reimplement the Jupyter notebook: “Regularization of linear regression with norms and constraints.pdf” (in MyMoodle). Implement regularization with the
  - norm penalty using the L1 or L2 norm and
  - norm inequality constraint using the L1 or L2 norm and
- All implementation in Python and explain what you have done.

- Deadline: 2025-04-29 (before the next lecture)