Deep Learning

Lecture 7 – Practical methodology, batch normalization, and learning from multiple tasks

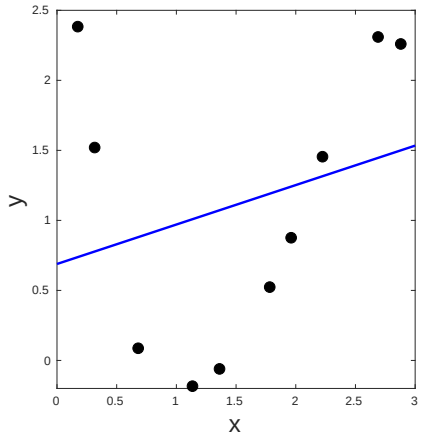
UPPSALA
UNIVERSITET

Niklas Wahlström
Division of Systems and Control
Department of Information Technology
Uppsala University

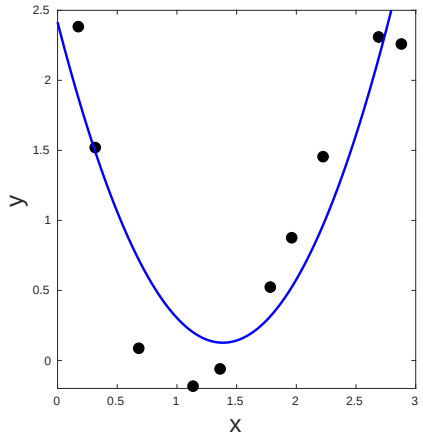
Email: niklas.wahlstrom@it.uu.se

Summary Lecture 6 (I/IV)
Bias and Variance

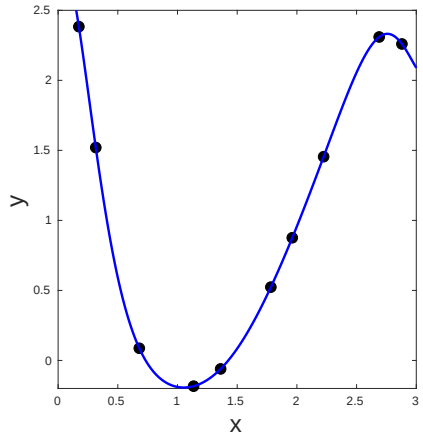
Training dataset 1



Model 1
Low model complexity



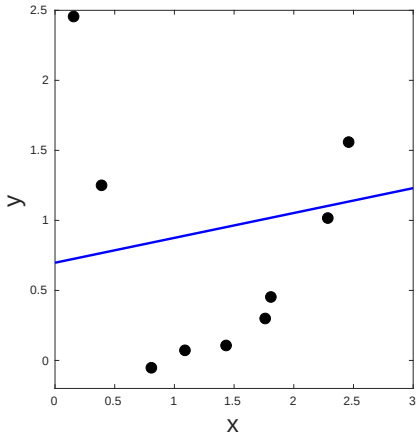
Model 2



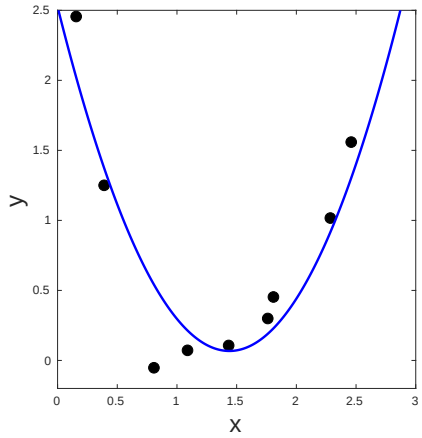
Model 3
High model complexity

Summary Lecture 6 (I/IV)
Bias and Variance

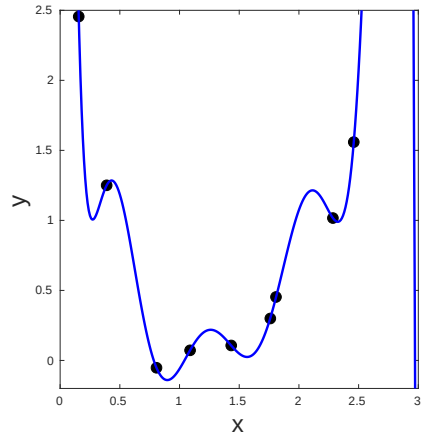
Training dataset 2



Model 1
Low model complexity



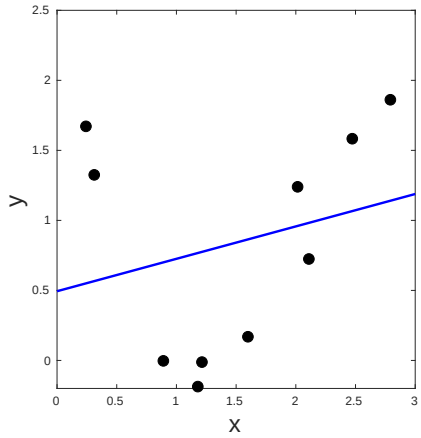
Model 2



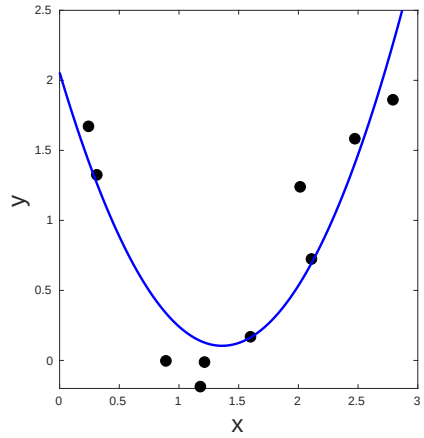
Model 3
High model complexity

Summary Lecture 6 (I/IV)
Bias and Variance

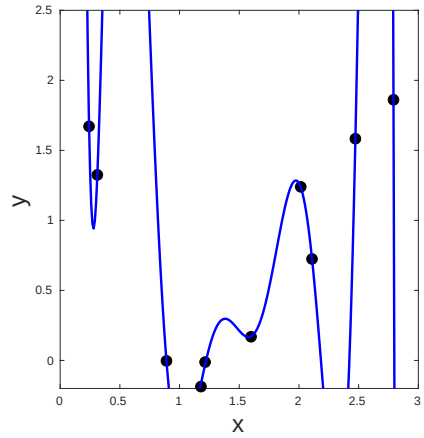
Training dataset 3



Model 1
Low model complexity



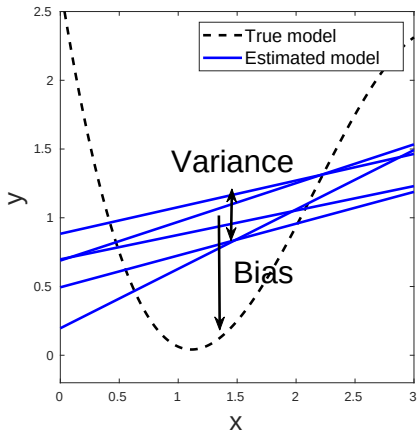
Model 2

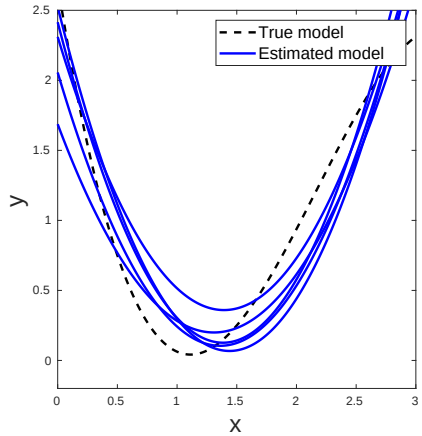


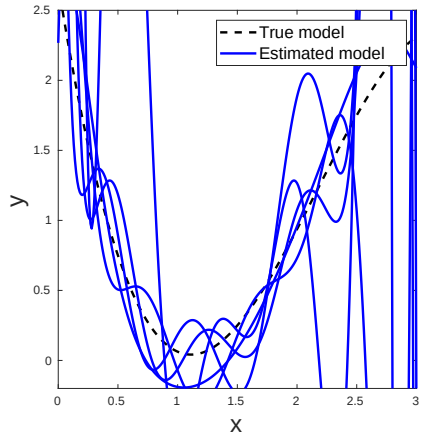
Model 3
High model complexity

Summary Lecture 6 (I/IV)
Bias and Variance

Models from all training datasets

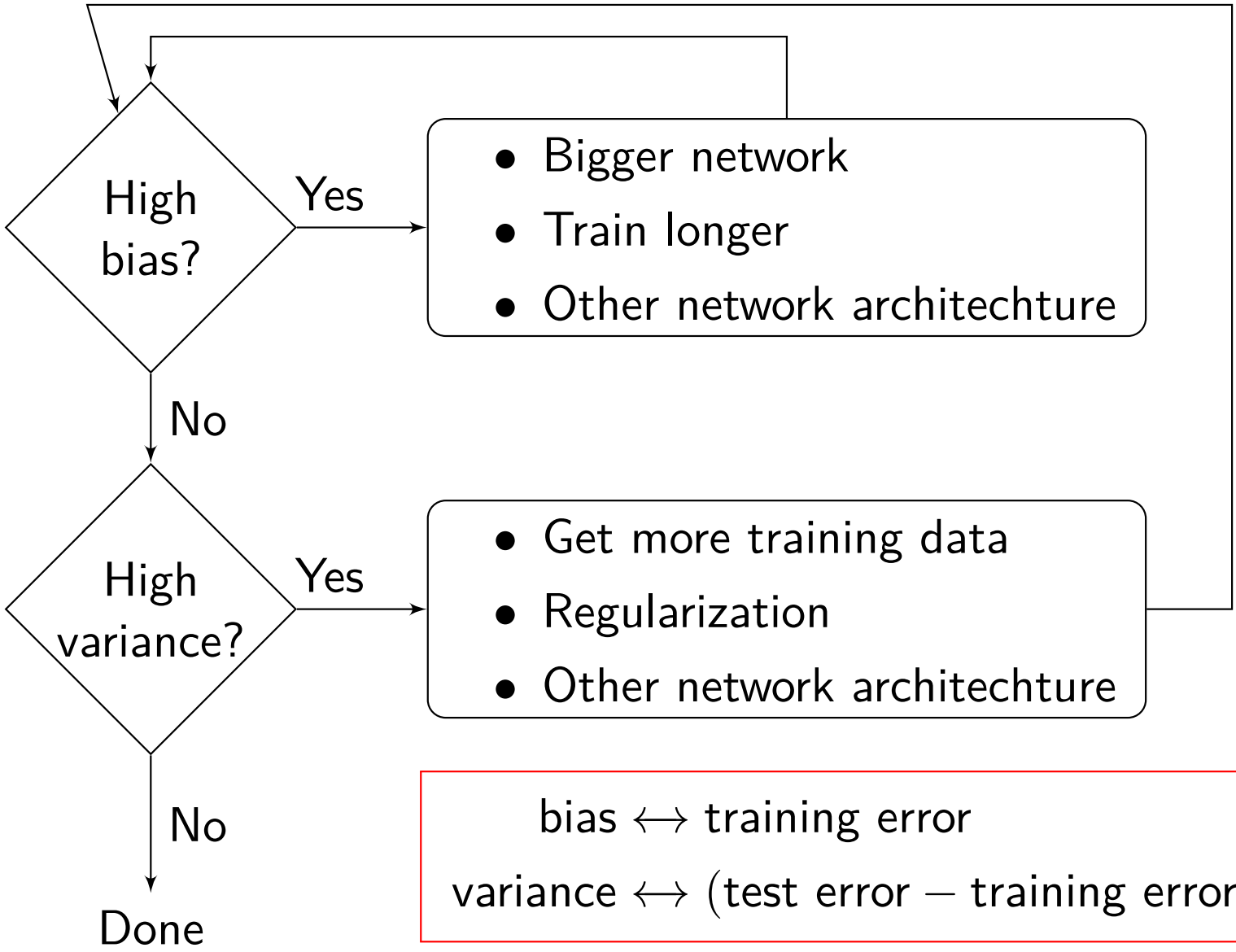






Model 1	Model 2	Model 3
Low model complexity		High model complexity
High bias		High variance
Low variance	"Just right"	Low bias
Underfitting		Overfitting

Summary Lecture 6 (II/IV)
Basic recipe for ML



$$\text{bias} \longleftrightarrow \text{training error}$$

$$\text{variance} \longleftrightarrow (\text{test error} - \text{training error})$$

Summary Lecture 6 (III/IV)
Regularization methods

Weight decay

Regularize using a penalty on the Euclidean norm

$$\tilde{J}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \frac{\lambda}{2n}\|\boldsymbol{\theta}\|_2^2.$$

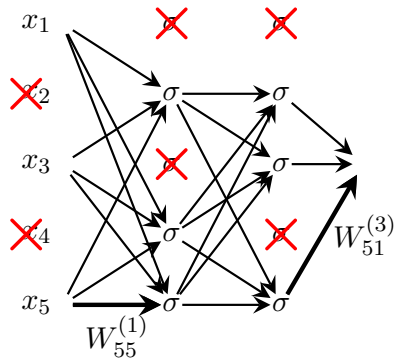
Ensamble methods

Train many high-varaince models and average

$$\hat{f}(x) = \sum_{i=1}^{B} \frac{\hat{f}_i(x)}{B}$$

Dropout

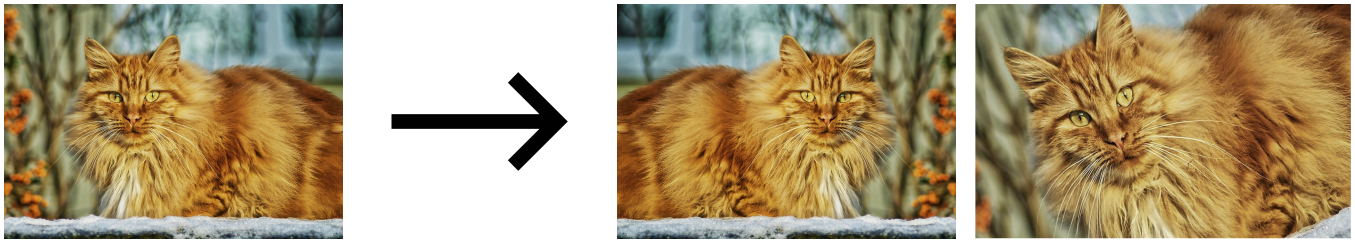
During training randomly drop units. Can be interpreted as an ensemble method



Summary Lecture 6 (IV/IV)
Regularization methods

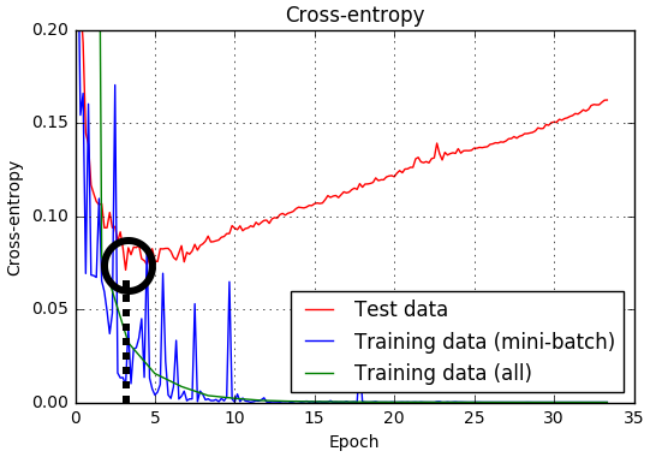
Data augmentation

Augment the training data by applying transformation (for images for example flipping, tilting, zooming etc).



Early stopping

Stop training when test loss (or error) is at its minimum



Outline - Lecture 7

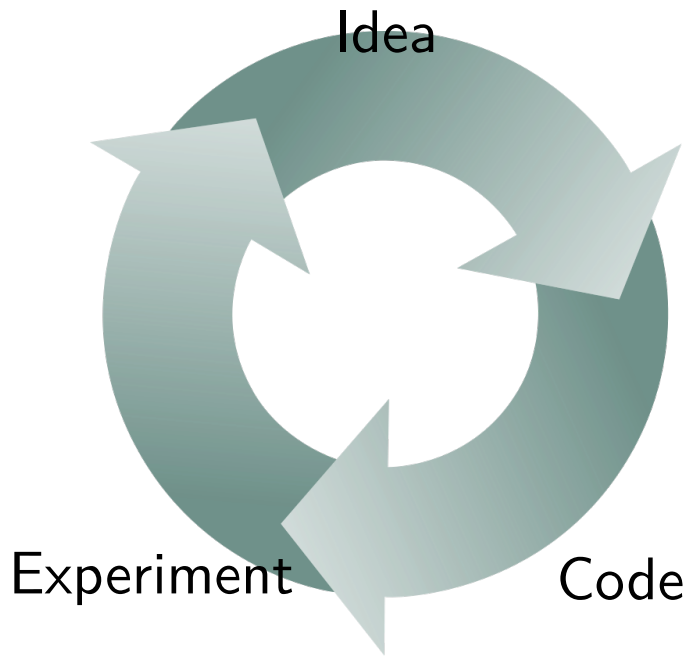
Outline:

1. Summary Lecture 6
2. Practical methodology
3. Input normalization and batch normalization
4. Learning from multiple tasks

Applied deep learning is an iterative process

We have encountered quite some different model choices

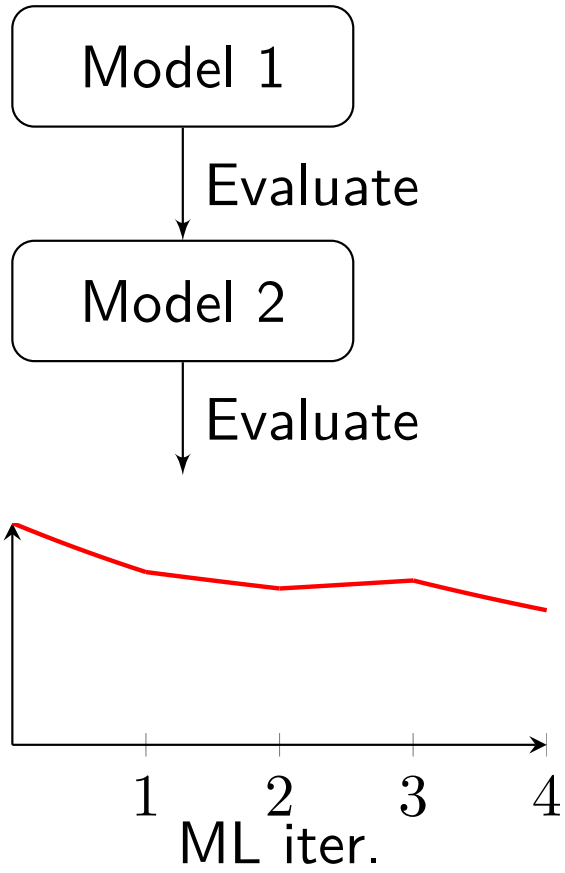
- Number of layers
- Number of hidden units
- Kernel size (for CNN)
- Learning rate
- Regularization parameters
- Activation function



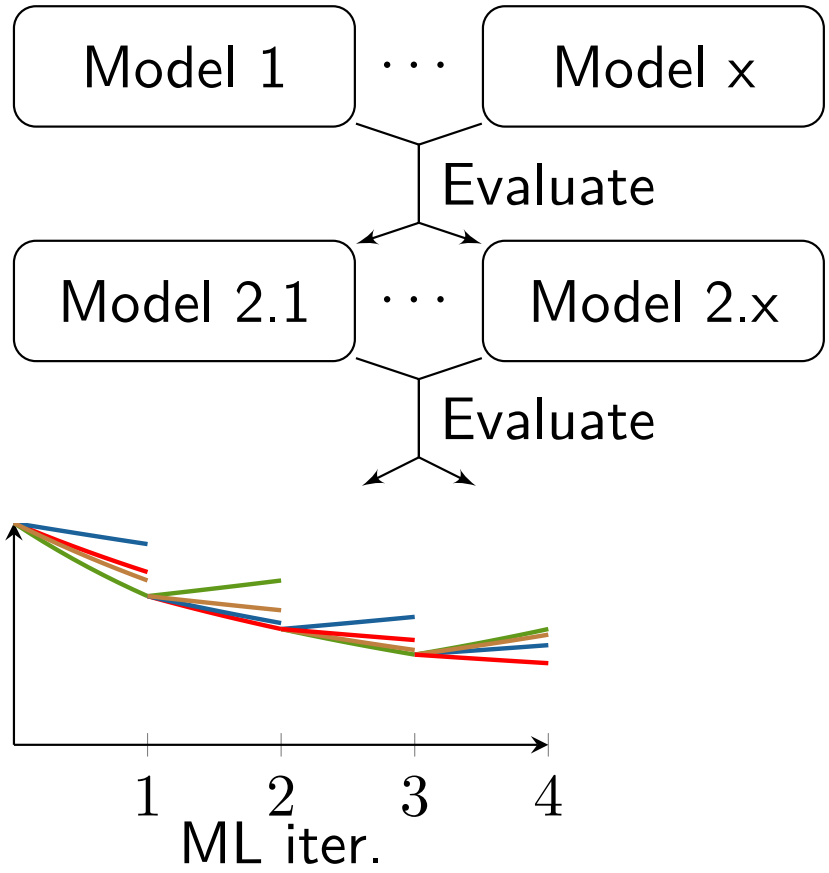
We choose hyperparameters which gives the best performance on data not used during training (test data).

Finding hyperparameters in practize

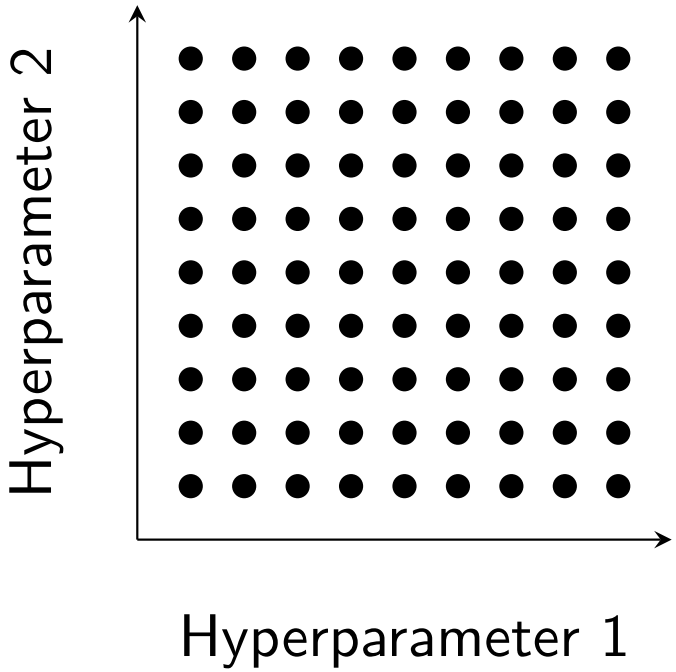
Babysitting one model
(Try one set of hyper-parameters at a time)



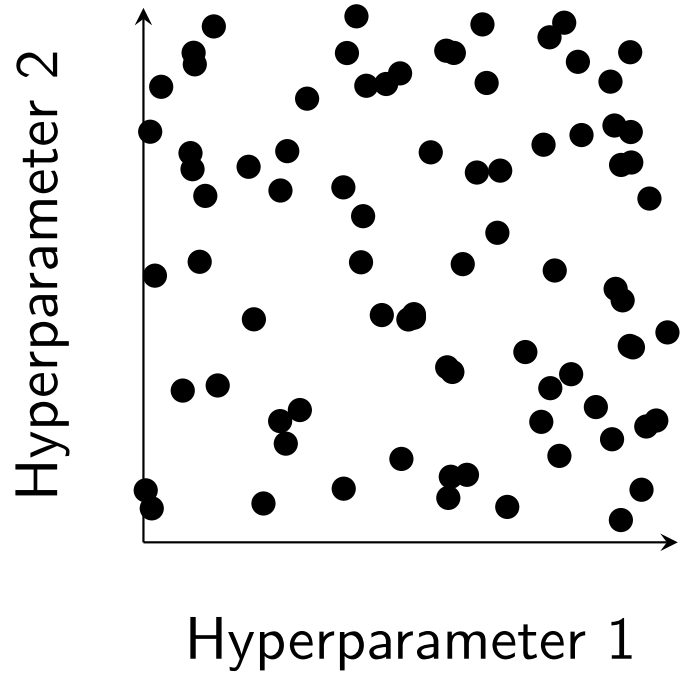
Many models in parallel
(Try multiple sets of hyperparameters at a time)



Hyper-parameter search



If you have few hyperparameters, you can use a grid.

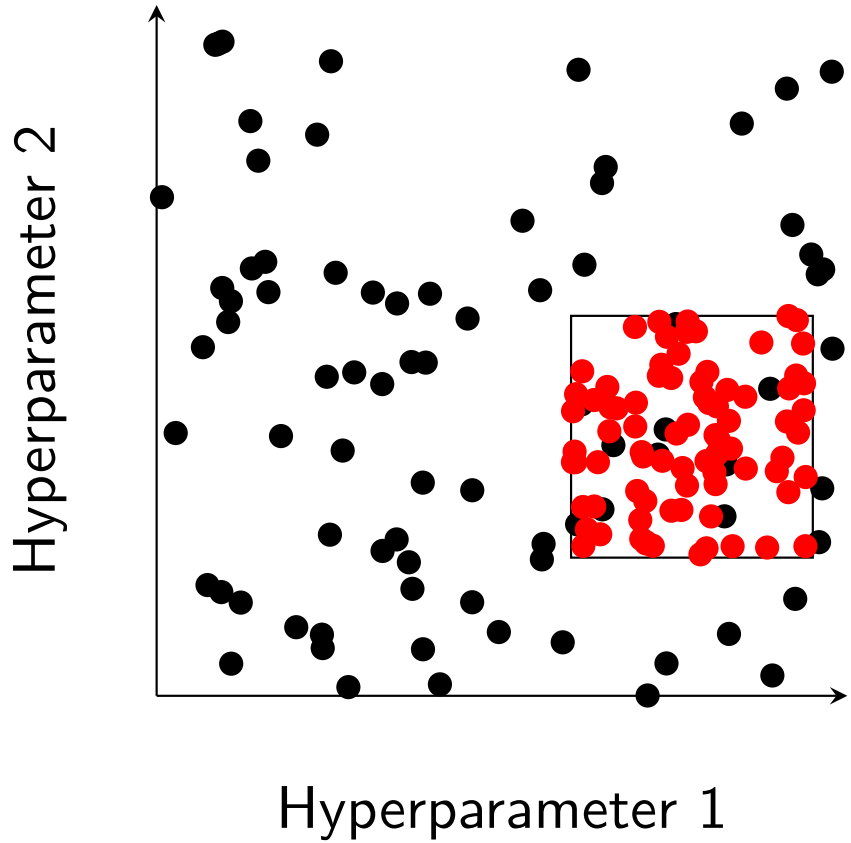


If you have many hyperparameters, pick points at random.

Note: Number of grid points increases exponentially with number of hyperparamters

Refine the grid

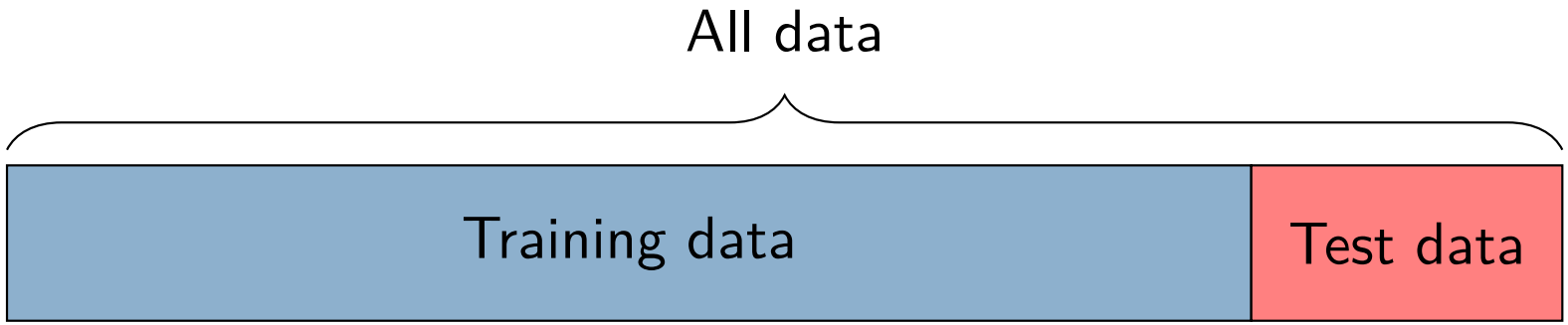
Refine the search region based on your evaluation.



Optimizing hyperparameters based on your test dataset now starts becoming problematic. Why?

Lecture 2: Training/test data split

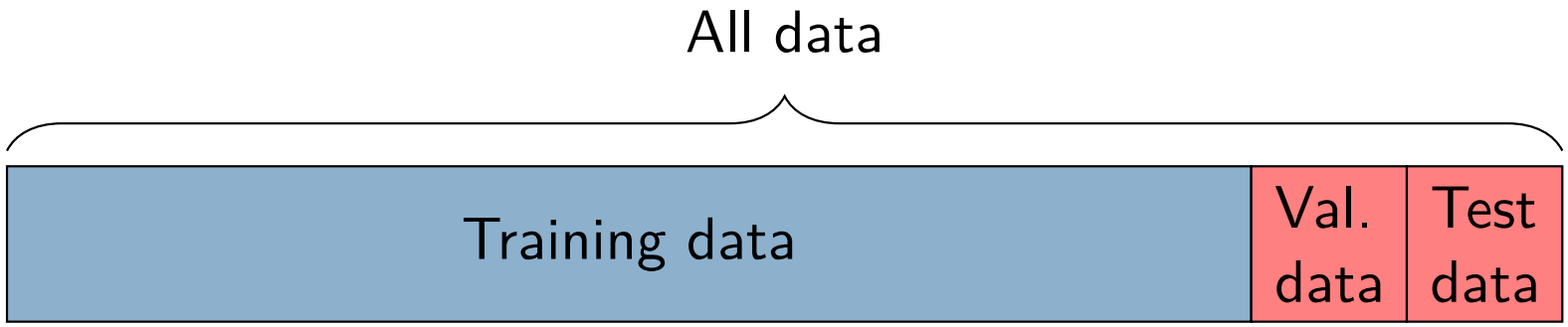
In Lecture 2 we we explined the training/test data split



- **Training data**: Used for **training** the model
- **Test data**: Used for **evaulating** the model

Problem: If you optimize hyperparameters based on your test data performance, you might start to adapt to your test data!
Solution: Split your available data into three parts.

Training/validation/test data split

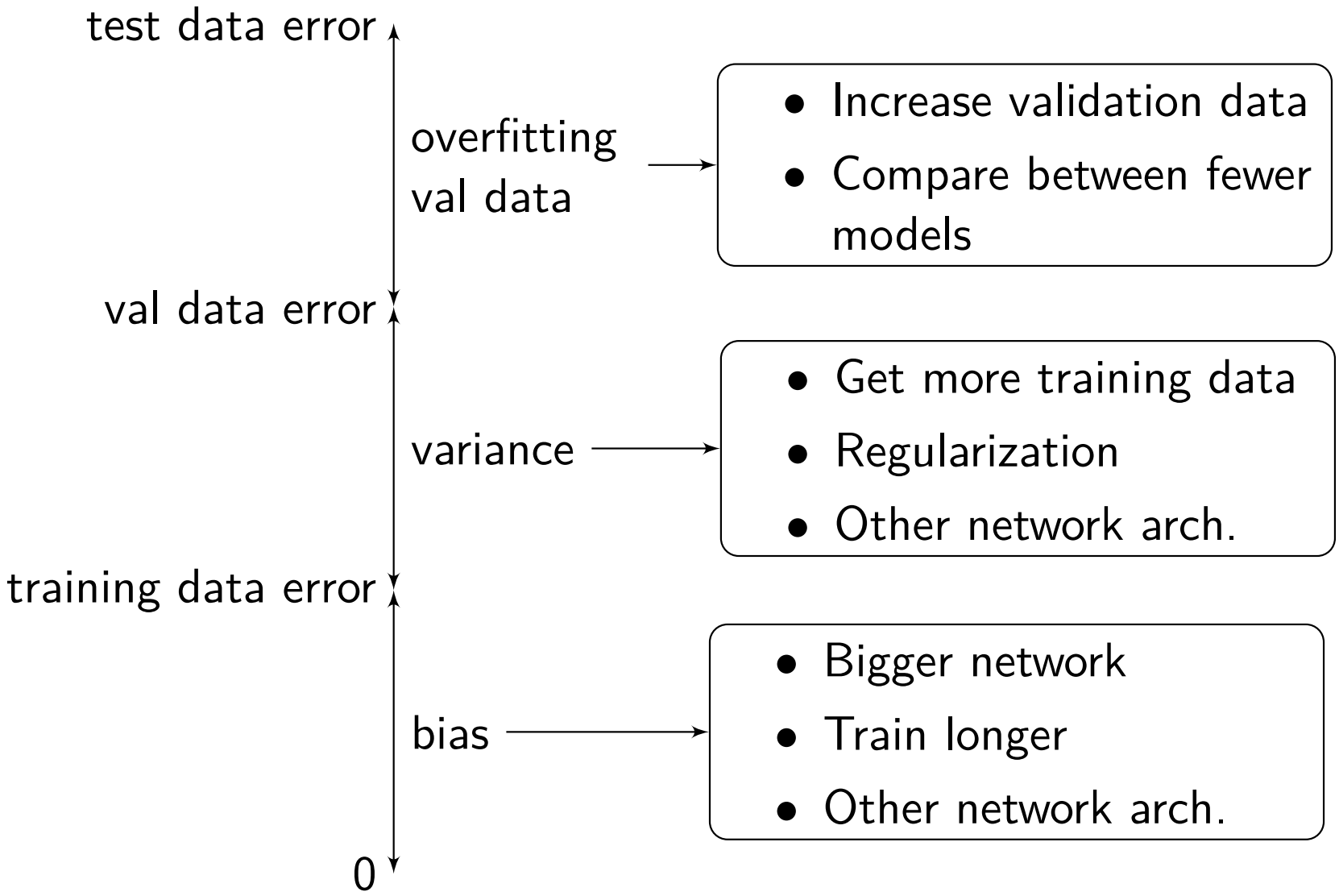


Split your data into:

- **Training data**: Used for **training** the model
- **Validation data**: Used for **chosing between models**for **optimizing hyperparameters**
- **Test data**: Used for **evaulating** the model

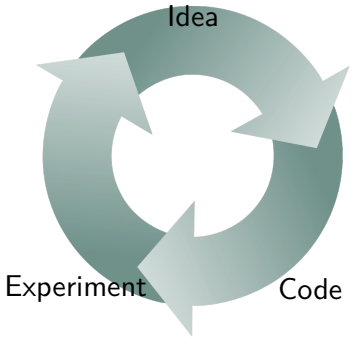
Note: Validation data and test data should come from the same distribution!

Training/validation/test data split
Bias and variance



Single number evaluation metric

Example: Classification of skin cancer tumors (based in images)
$y \in \{\text{malignant}, \text{benign}\}$



	Precision	Recall
Model A	93%	85%
Model B	86%	95%

Precision: Of examples classified as malignant, what % are actually malignant?
Recall: What % of malignant examples are actually classified as malignant?
F1-score: "Avarage" of P and R $\frac{2}{\frac{1}{P}+\frac{1}{R}}$.
What a good evaluation metrics is depends on your problem.

Well-defined validation/test set + single number evaluation metrics defines your problem speeds up your search!

Changing evaluation metrics (I/II)

Metric: Classification error (1- classification accuracy)
Model A: 5 % error
Model B: 8 % error
Problem: Model A seems to do better but you prefer model B since it is doing better on malignant melanoma tumors which you are especially interested in.
Solution:

- Change your metrics

$$\frac{1}{\sum_{i=1}^{n} w_i} \sum_{i=1}^{n} w_i \mathbb{I}(\hat{y}_i \neq y_i), \; w_i = \begin{cases} 10 & \text{if malignant melanoma} \\ 1 & \text{otherwise} \end{cases}$$

- and/or your validation/test dataset distribution (by including more malignant melanoma examples).

Changing evaluation metrics (II/II)

- How do we make sure that we train our model towards this metric?
- If we have reweighed the metric, we can do the same for our training objective.

Metric

$$\frac{1}{\sum_{i=1}^{n} w_i} \sum_{i=1}^{n} w_i \mathbb{I}(\hat{y}_i \neq y_i)$$

Cost function

$$J = \frac{1}{\sum_{i=1}^{n} w_i} \sum_{i=1}^{n} w_i L(\hat{y}_i \neq y_i)$$

Conclusion:

1. Define your problem by choosing metric and validation/test dataset.
2. Adapt your learning algorithm to do well on your metric.

Mismatched training and val/test data (I/II)

Task Detecting deers based on images from webcam.
We scroll the web for more training data.

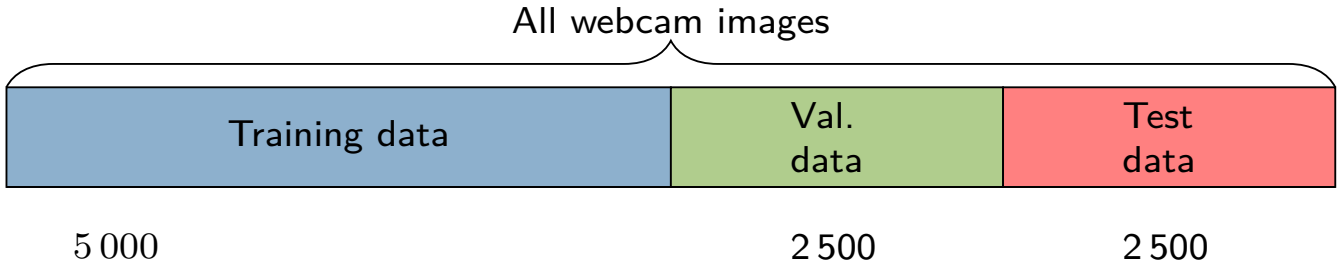


Data from webpages $\approx 200\,000$



Data from webcam $\approx 10\,000$

Option 1: Use only webcam images



Advantage: We only train, validate and evaluate on webcam images which is also what we want our model to perform well on.
Disadvantage: We have quite few data points and don't exploit the potential performance increase the webpage images could give us.

Mismatched training and val/test data (I/II)

Task Detecting deers based on images from webcam.
We scroll the web for more training data.

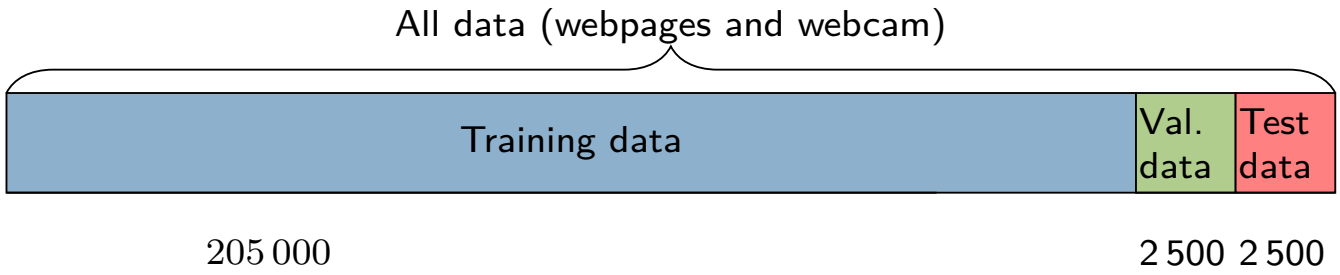


Data from webpages $\approx 200\,000$



Data from webcam $\approx 10\,000$

Option 2: Randomly shuffle your data and split it up



Advantage: We have a lot more training data in comparison to option 1.

Disadvantage: We evaluate our model mainly on webpage images whereas we want our model to perform well on webcam images.

Mismatched training and val/test data (I/II)

Task Detecting deers based on images from webcam.
We scroll the web for more training data.

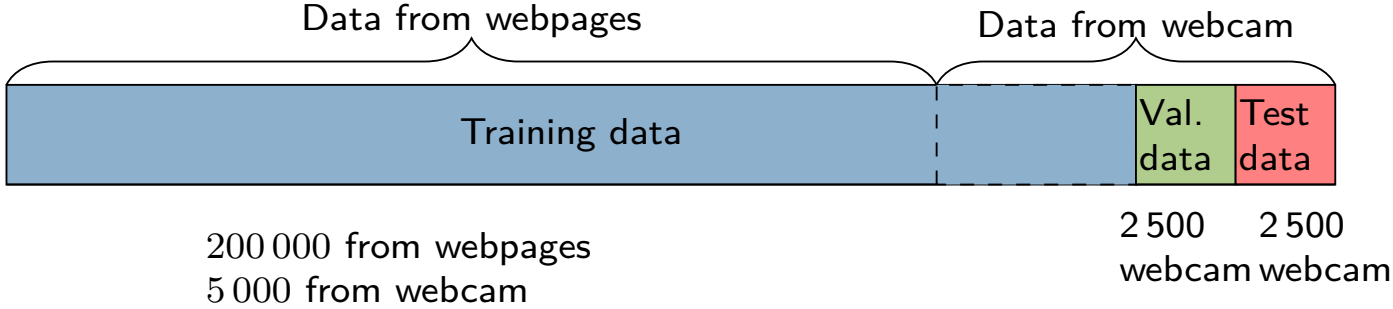


Data from webpages $\approx 200\,000$



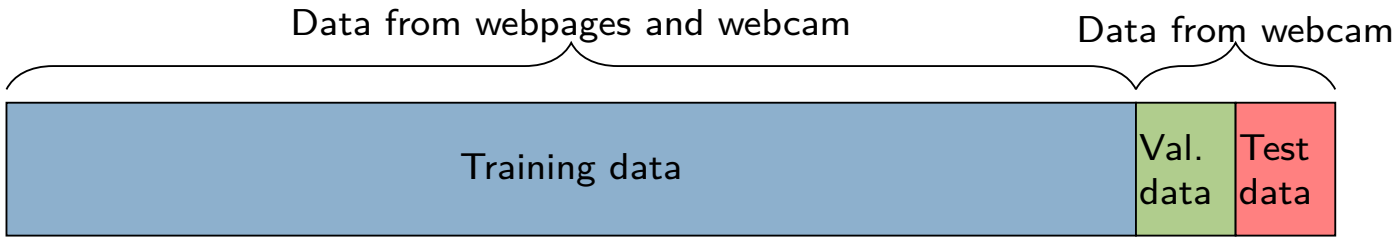
Data from webcam $\approx 10\,000$

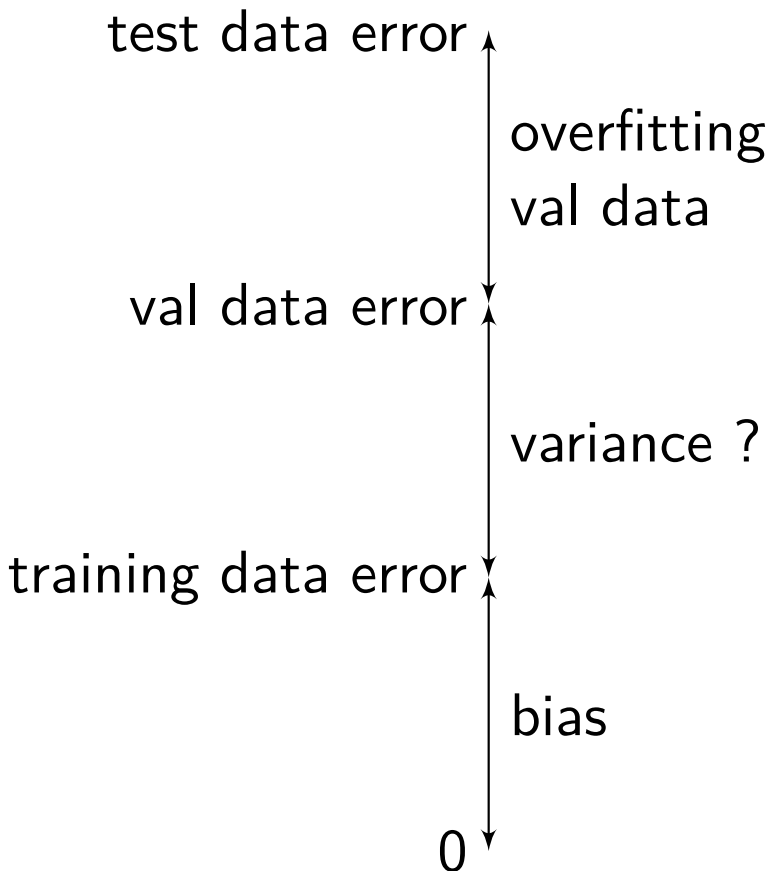
Option 3: Webcam images in val/test set and some in training set.



Advantage: We have more training data than option 1 and we do evaluate our model on webcam images in contrast to option 2.
Disadvantage: Training data does no longer have the same distribution as validation/test data.

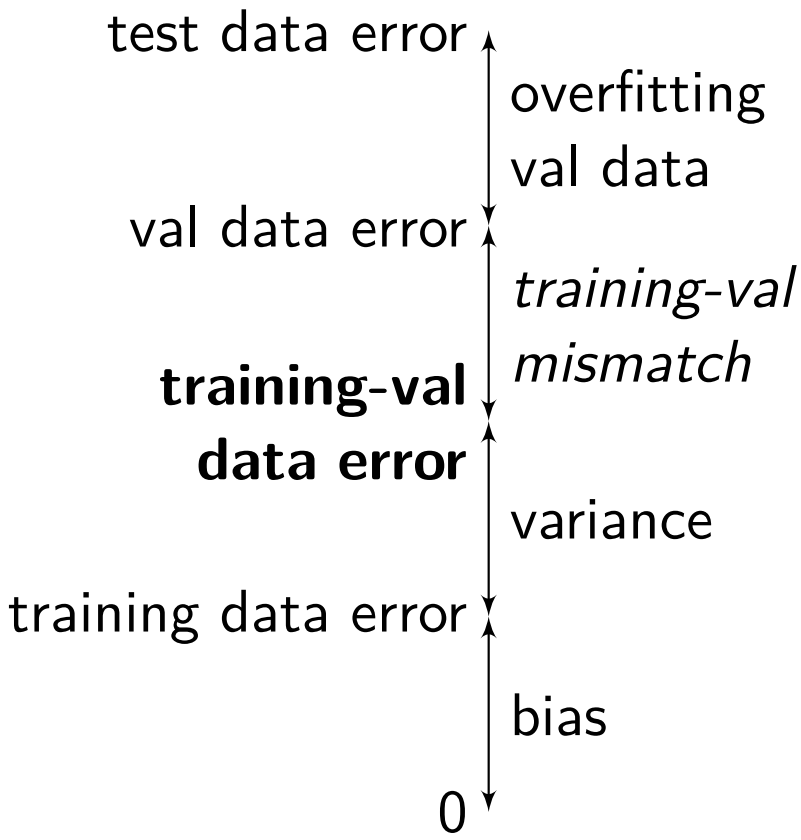
Mismatched training - val/test data (II/II)
Bias and variance





- **Task** Decompose performance in bias and variance.
- **Problem** Training and validation data no longer come from same distribution.

Mismatched training - val/test data (II/II)
Bias and variance



Solution

- From your training data create a separate **training-val data**
- Training-val data should have the *same* distribution as the training data.
- You do *not* train on the training-val data, only on training data.

Addressing training - val/test data mismatch

- **Analyze** the differences between training and val/test data
- Collect more training data of the **same type** as val/test data.
- Create **artificial training data examples** which resembles the val/test data.
- Give webcam images **higher importance** in the training cost function.

Outline - Lecture 7

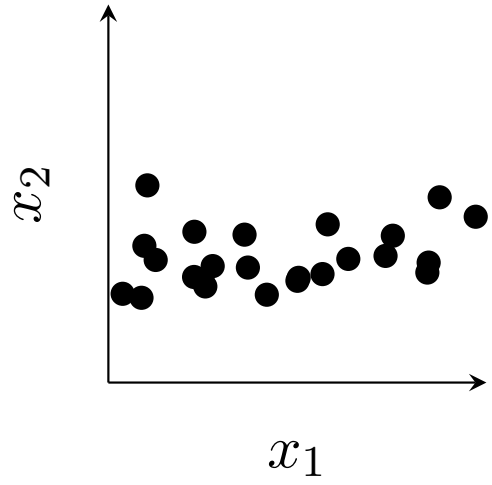
Outline:

1. Summary Lecture 6
2. Practical methodology
3. Input normalization and batch normalization
4. Learning from multiple tasks

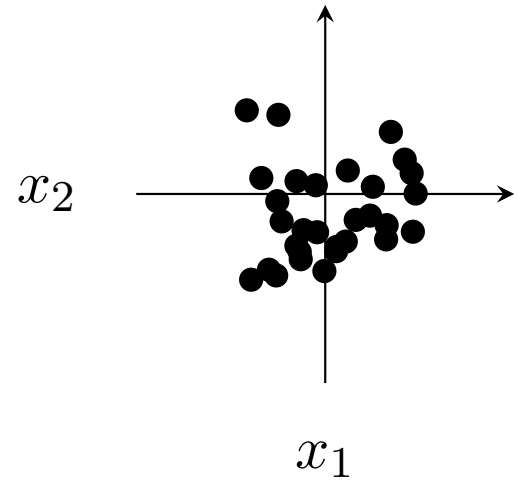
Normalizing training inputs

Consider problem with two inputs x_1 and x_2.

Unnormalized



Normalized



1. Compute mean and variance of training data.

$$\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_{ij}, \qquad \sigma_j^2 = \frac{1}{n}\sum_{i=1}^{n}(x_{ij}-\mu_j)^2$$

2. Normalize $\bar{x}_{ij} = \frac{x_{ij}-\mu_j}{\sigma_j}$

Use also μ_j and σ_j^2 to normalize validation/test data.

Why normalizing inputs?

- If inputs x_1 and x_2 are unnormalized, the cost function will also be "unnormalized".
- This leads to slower training.

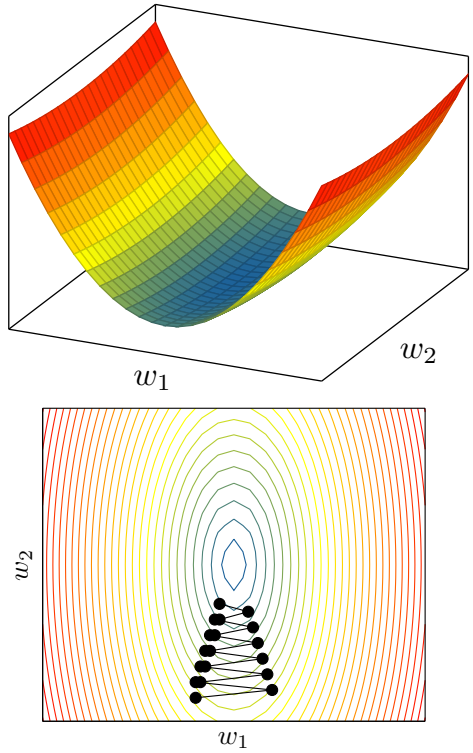
Model

$$\hat{y} = w_1 x_1 + w_2 x_2$$

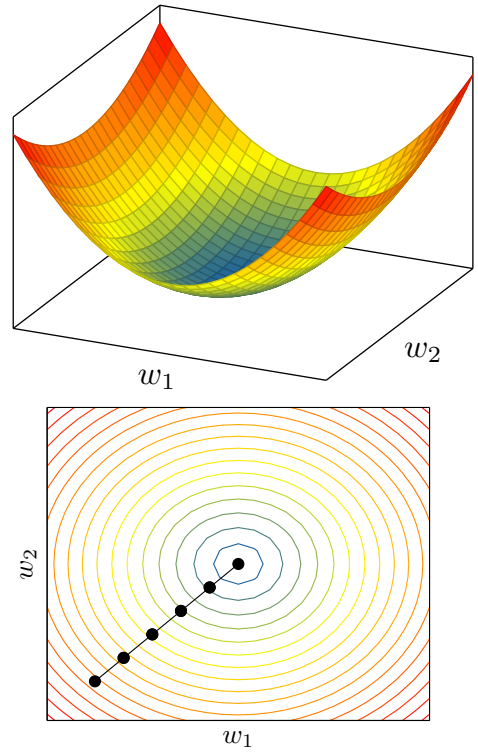
Cost function

$$J = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Unnormalized



Normalized



Compare with pre-course assignment: Very difficult to find a good learning rate without normalizing the inputs.

Batch normalization

- **Batch normalization** is a technique where you normalize the input of each layer in the network during training. Ioffe, Sergey, and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." International Conference on Machine Learning. 2015.
- Has since its introduction 2015 been one of the most efficient techniques for training deep neural networks.
- Can use **higher learning rate** without getting problem with vanishing or exploding gradients.
- Has a slight **regularizing effect** (reduces the need for dropout for example)

Batch normalization on each mini-batch

For every mini-batch, do the following:

- Compute **mean** and **variance** for every unit j in all layers l

$$\mu_j^{(l)} = \frac{1}{n_{\text{batch}}} \sum_{i=1}^{n_{\text{batch}}} z_{ij}^{(l)}$$

$$(\sigma_j^{(l)})^2 = \frac{1}{n_{\text{batch}}} \sum_{i=1}^{n} (z_{ij}^{(l)} - \mu_j^{(l)})^2$$

$z_{ij}^{(l)}$ is the hidden unit *before* the activation

- **Normalize** every unit j in all layers l

$$\bar{z}_{ij} = \frac{z_{ij}^{(l)} - \mu_j^{(l)}}{\sqrt{(\sigma_j^{(l)})^2 + \epsilon}}$$

ϵ used for numerical stability

- **Scale** and **shift** every unit

$$\tilde{z}_{ij}^{(l)} = \gamma_j^{(l)} \bar{z}_{ij}^{(l)} + \beta_j^{(l)}$$

Batch normalization - gradient descent

Forward propagation

$\mathbf{h}^0 = \mathbf{x}$

For layer $l = 1$ to L

1. Apply linear transformation $\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{h}^{(l-1)}$
2. Normalize $\bar{\mathbf{z}}^{(l)} = \frac{\mathbf{z}^{(l)} - \boldsymbol{\mu}^{(l)}}{\sqrt{\boldsymbol{\sigma} + \epsilon}}$
3. Scale and shift $\tilde{\mathbf{z}}^{(l)} = \boldsymbol{\gamma}^{(l)}\bar{\mathbf{z}}^{(l)} + \boldsymbol{\beta}^{(l)}$.
4. Apply nonlinear transformation $\mathbf{h}^{(l)} = g(\tilde{\mathbf{z}}^{(l)})$.

No $\mathbf{b}^{(l)}$ needed. Redundant due to $\boldsymbol{\beta}^{(l)}$

Backward propagation

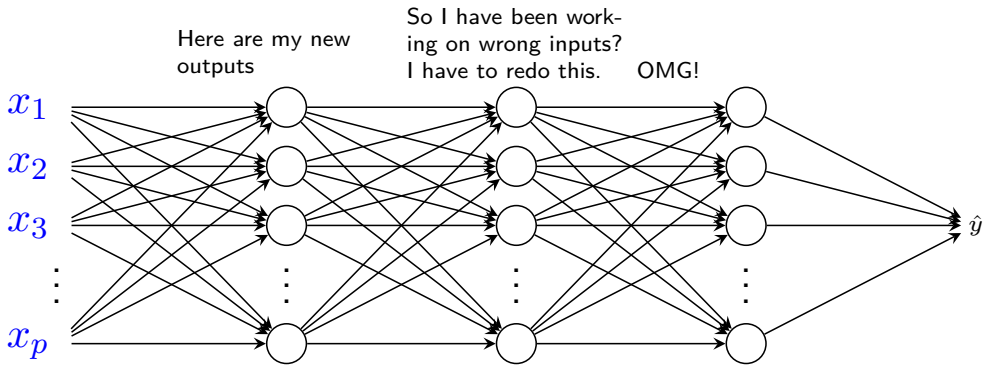
Do backward propagation to get gradients with respect to all $\{\mathbf{W}^{(l)}, \boldsymbol{\beta}^{(l)}, \boldsymbol{\gamma}^{(l)}\}_{l=1}^{L}$.

Update all parameters

Update all parameters $\mathbf{W}^{(l)}, \boldsymbol{\beta}^{(l)}, \boldsymbol{\gamma}^{(l)}{}_{l=1}^{L}$

Why does batch normalization work?

- Not yet fully understood why batch normalization works.
- Motivation of the original paper was based on **internal covariante shift**



- More recent (and widely accepted) explanation is that it makes the **loss landscape more smooth** and easier to optimize.

Santurkar, Shibani, et al. "How does batch normalization help optimization?." Advances in Neural Information Processing Systems. 2018.

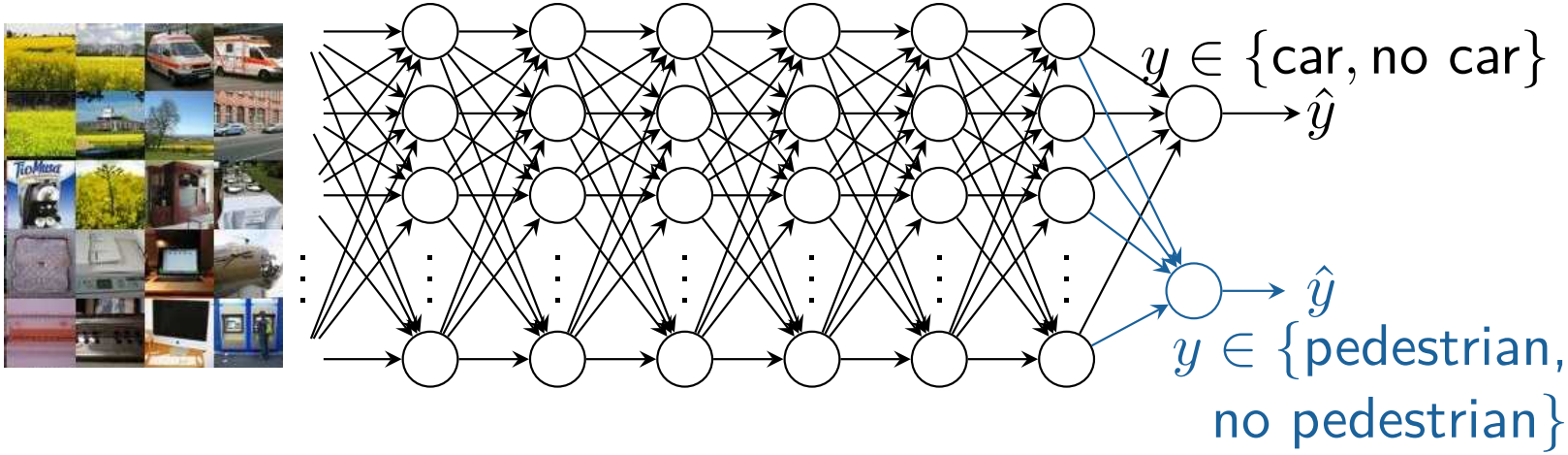
Outline - Lecture 7

Outline:

1. Summary Lecture 6
2. Practical methodology
3. Input normalization and batch normalization
4. Learning from multiple tasks

Multitask learning

In **multitask learning** we train a network for solving multiple tasks.

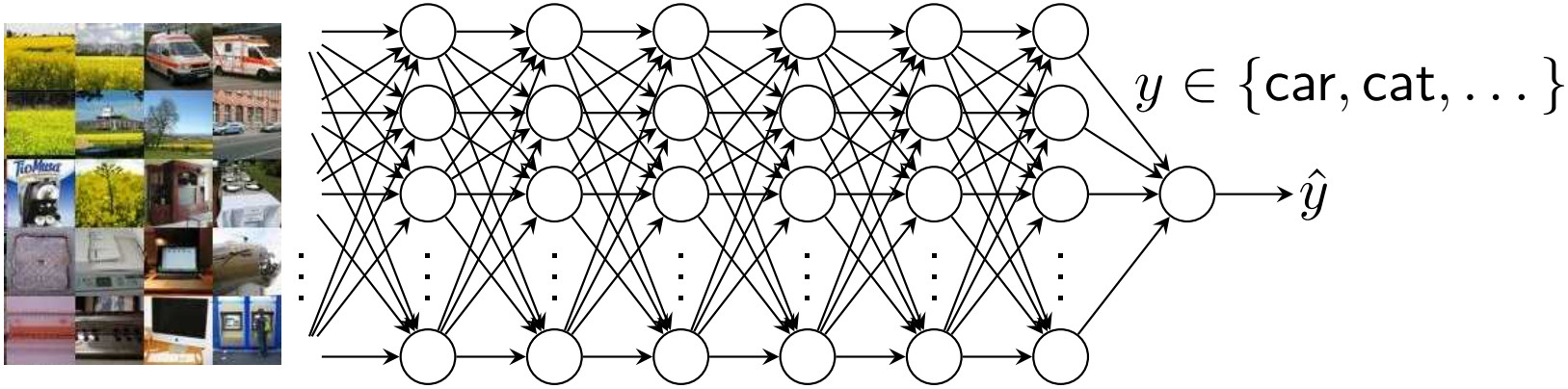


- Different tasks benefit from having **shared lower-level features**
- Can be seen as a **regularization** (in comparison to training the task separately)
- We already have seen examples of multi-task learning in Lecture 5 (Segmentation, object detection for example)

Transfer learning

In **transfer learning** we reuse models that have been trained on a different task than the one we are interested in.

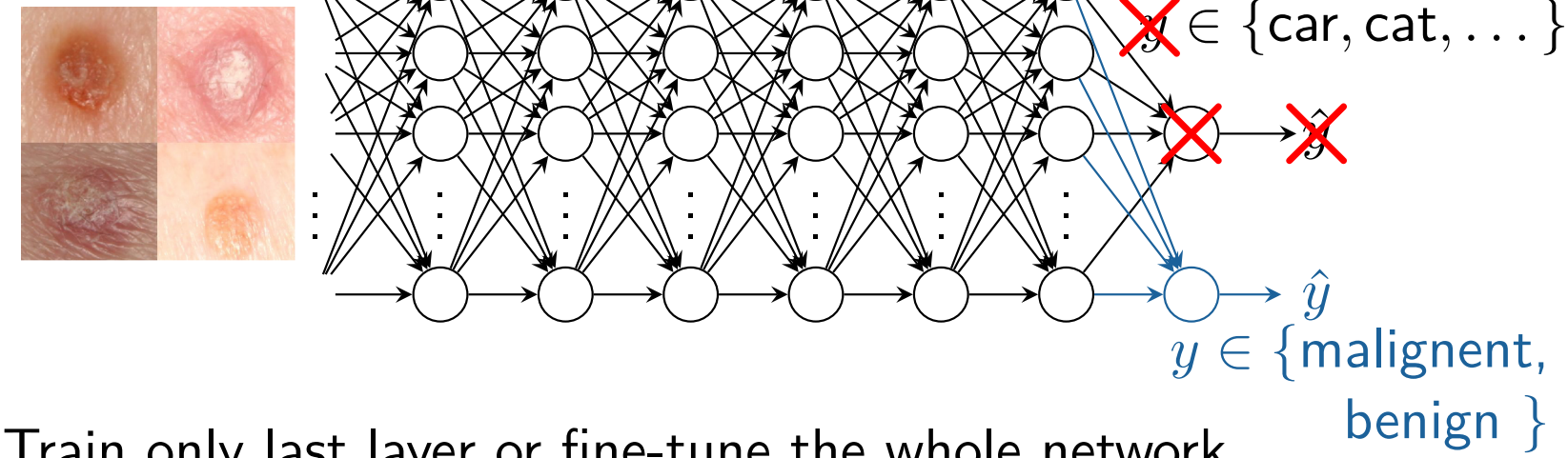
Task A: Image recognition $\approx 10\,000\,000$ images



Transfer learning

In **transfer learning** we reuse models that have been trained on a different task than the one we are interested in.

Task B: Skin cancer detection $\approx 100\,000$ images



Train only last layer or fine-tune the whole network.

Intuition Low level features from Task A helpful for solving Task B

Transfer learning applicable when:

- Task A and Task B have same input $\mathbf{x}$.
- We have a lot more data for task A than for task B.

Self-supervised learning

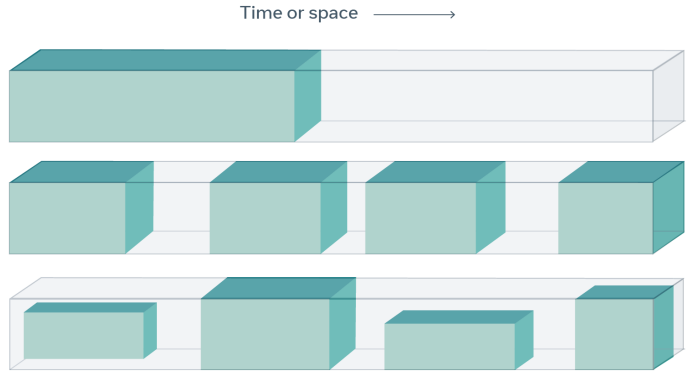
- Transfer learning requires a *labeled* dataset to transfer from.
- In **self-supervised learning** we generate supervised tasks from *unlabeled* data and pretrain on these supervised tasks.

Example:

Data: *This is a cat* $\xrightarrow{\text{Generate supervised task}}$ Input x: *This is _ cat*, Output y: *a*

Different strategies how to generate supervised tasks

1. Predict future
2. Predict part of sequence
3. Predict part of frame and sequence



LeCun, Yann, and Misra, Ishan "Self-supervised learning: The dark matter of intelligence", 2021.

UPPSALA UNIVERSITET

Reminder: Optional project

- After the course you are encouraged to carry out a project
- Preferably 1-4 students in each team
- Awarded 3hp extra
- Conducted during summer
- If you are looking for colleagues to do the project with, I added a discussion forum in Studium for this purpose.

Dates:
June 14 Project proposals
June 16 Project proposal presentations
August 24 Final reports (negotiable)
August 26 Final project presentations (negotiable)

A few concepts to summarize lecture 7

Validation data: Dataset used for choosing between models and for optimizing hyper-parameters

Hyperparameter: Parameters set prior training. We can optimize hyperparameters by picking the ones that gives best performance on validation data.

Evaluation metric: The metric you use to evaluate your performance. The metric together with your validation/test dataset defines your problem.

Batch normalization: A technique where you normalize the input of each layer in the network during training.

Multitask learning: Where you train a network to solve multiple tasks

Transfer learning: Where you fine-tune a network trained on a different task than the one that you are interested in.

Self-supervised learning: Same as transfer learning but where the tasks your transfer from have been generated from unlabeled data.