Neural networks: Training and regularization

Lecture 17 of “Mathematics and AI”

Outline

1. Anatomy of a supervised learner
2. Training

   Backpropagation
3. Regularization

   SGD, drop-out, pruning, architecture constraints
4. Convolutional neural networks

Anatomy of a supervised learner

Supervised learning (recap)

Hello Machine …

Let me show you some queries …

Let me tell you the correct answers to those queries …

Find the pattern!

Here are some queries that you haven't seen before.

Let me check how well you can answer those based on the pattern that you learned.

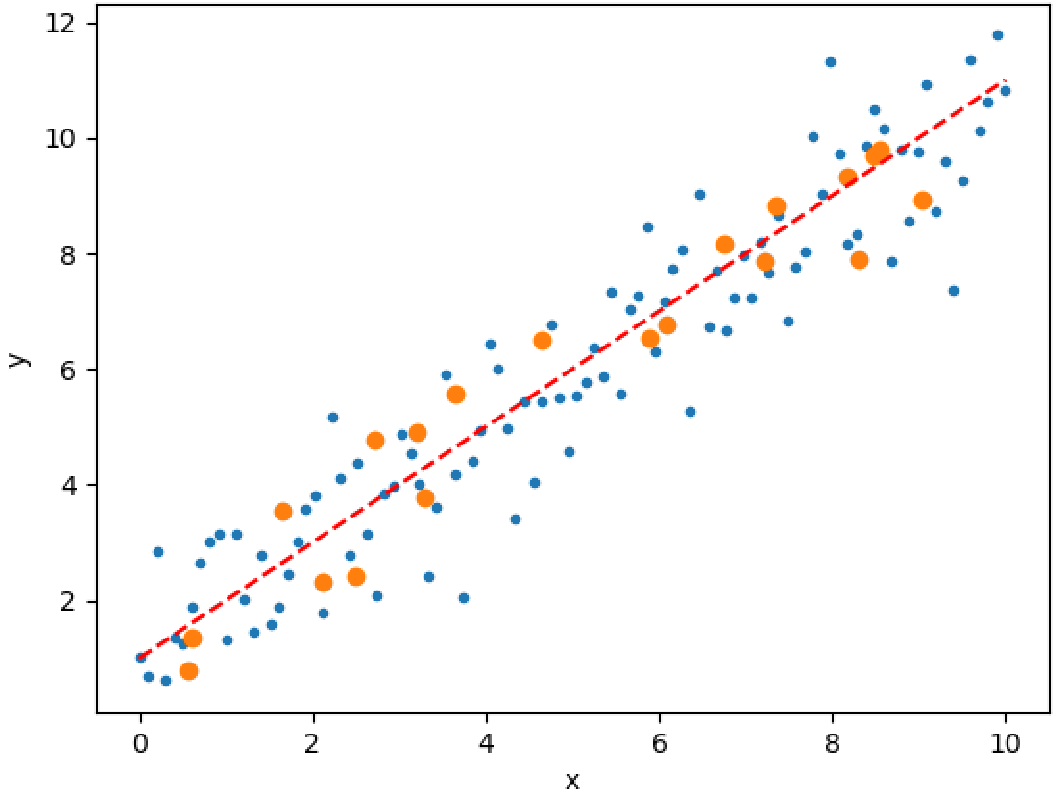
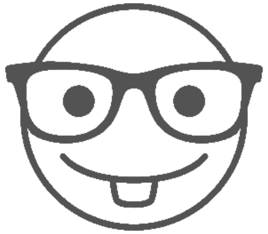


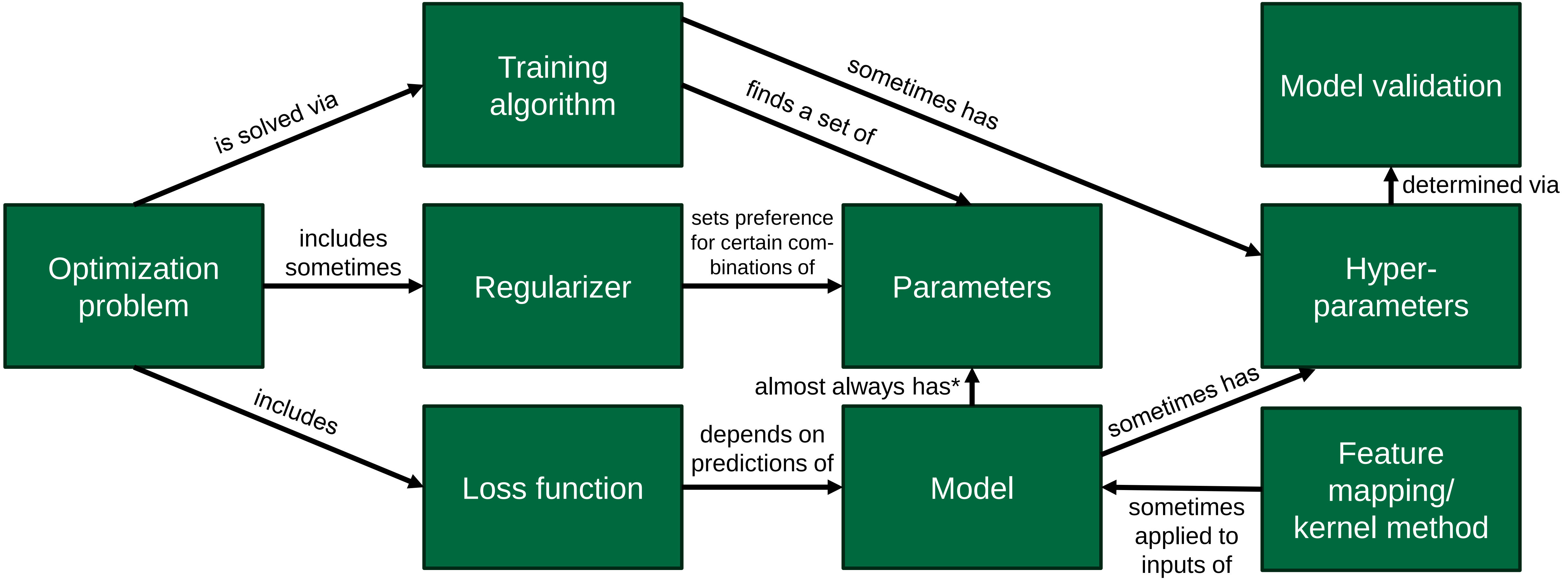
Image source: Photo by Dr. Schwarze of Dr. Schwarze's cat Emmy

Supervised learning (recap)



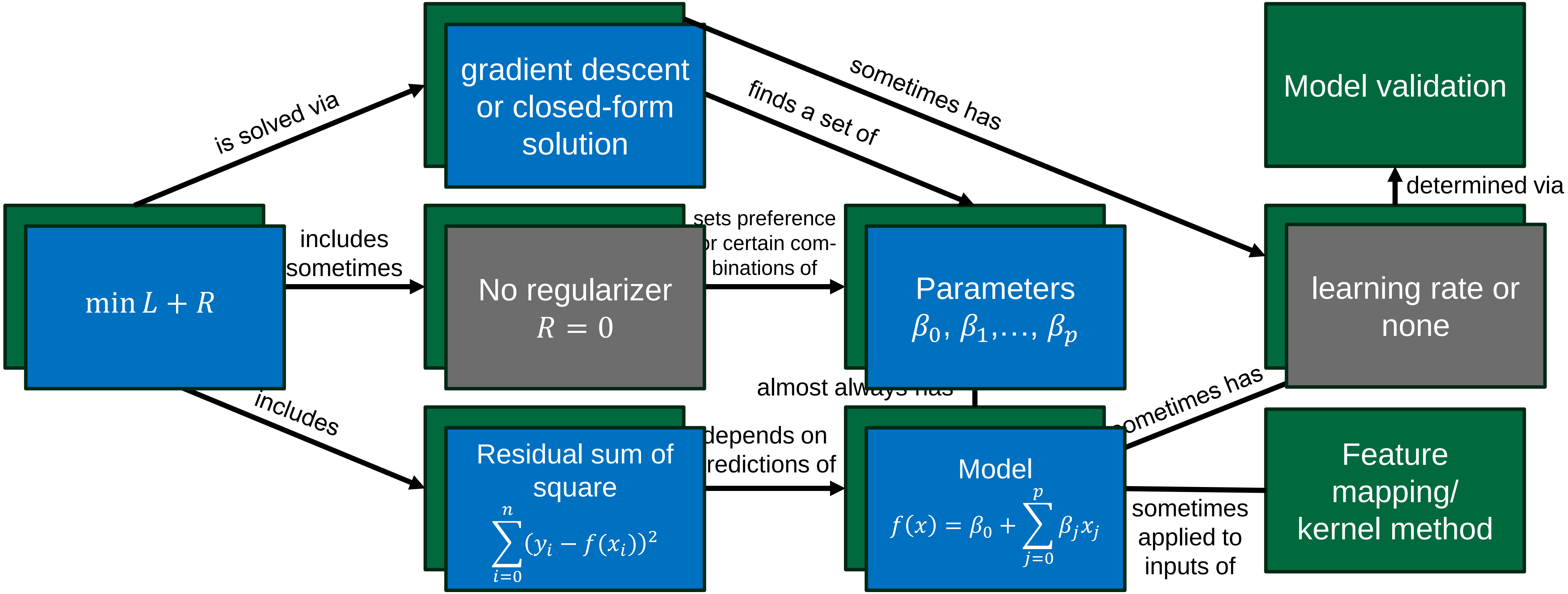
	Statistics	Machine learning
Hello Machine …		
Let me show you some queries … Let me tell you the correct answers to those queries …	Sample	Training set
Find the pattern!	Fit the model Quality of fit (within sample)	Train a model Training accuracy
Here are some queries that you haven't seen before.	Out-of-sample prediction	Test set
Let me check how well you can answer those based on the pattern that you learned.	Out-of-sample quality of fit	Test accuracy

A machine-learning model for supervised learning

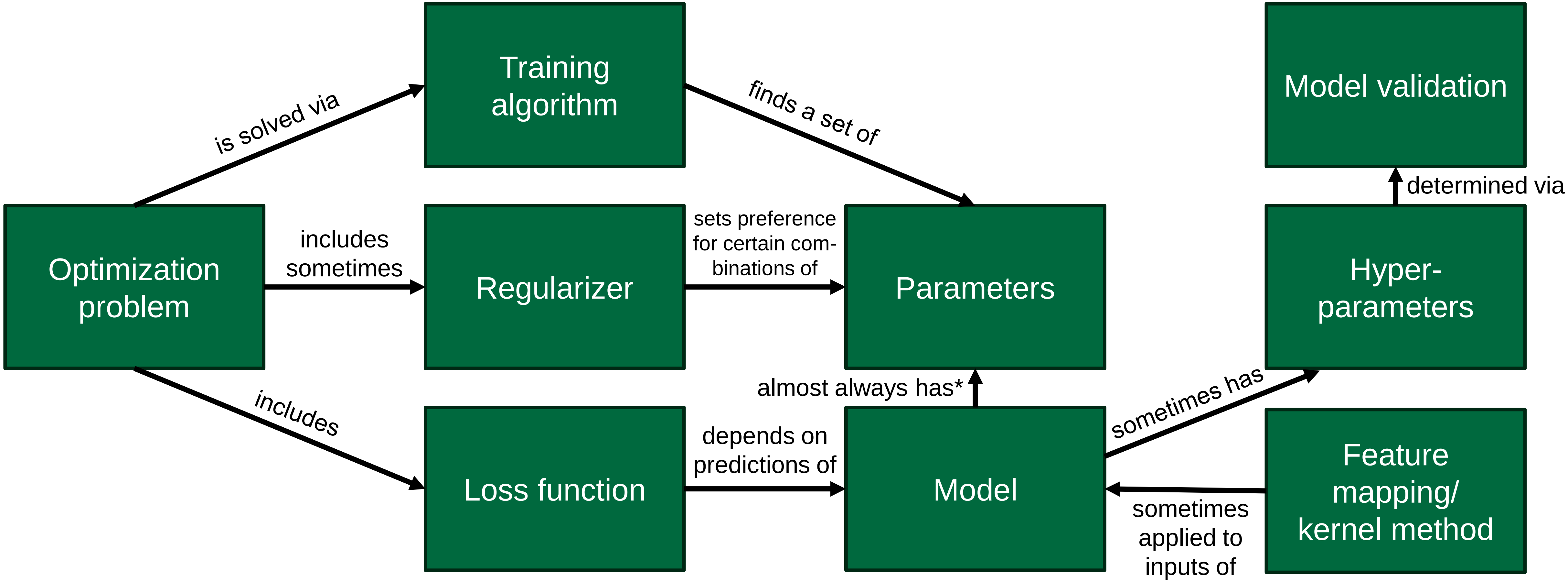


*parameter-free models don't require

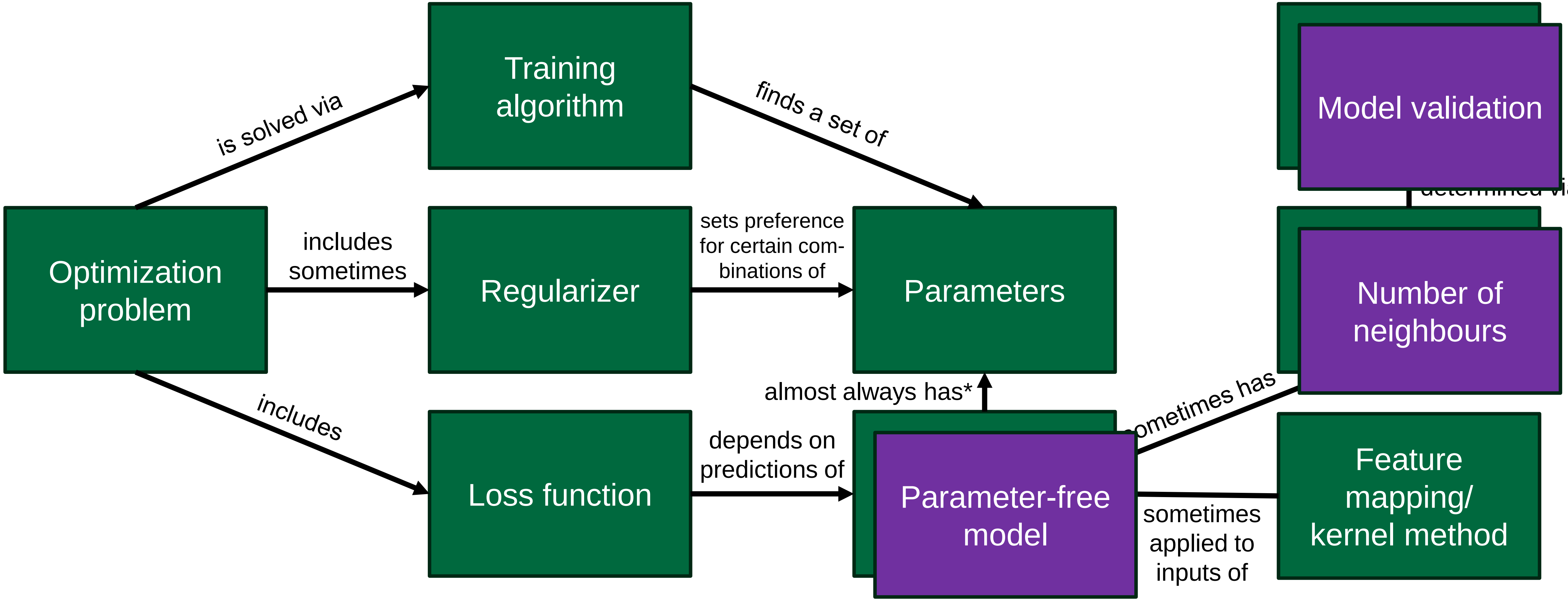
Example 1: OLS linear regression



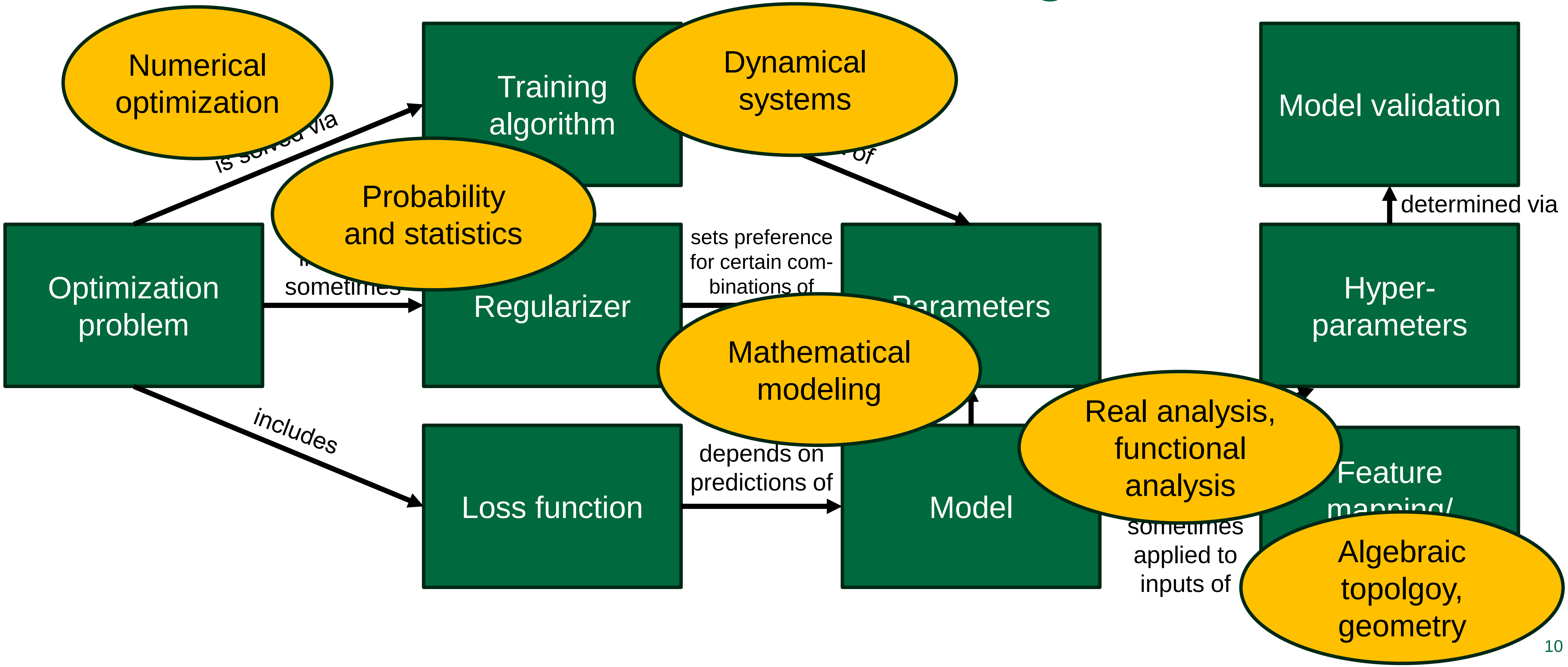
Example 2:


Training algorithm
Model validation
Optimization problem
Regularizer
Parameters
Hyper-parameters
Loss function
Model
Feature mapping/ kernel method
is solved via
finds a set of
determined via
includes sometimes
sets preference for certain com-binations of
includes
almost always has*
sometimes has
depends on predictions of
sometimes applied to inputs of

Parameter-free approaches (e.g., KNN)



Mathematics and machine learning



Training neural networks

Backpropagation

- How should a discrepancy between y and $\hat{y}$ affect the weights $w_{ij}^{(k)}$?
- Update rule for weights: SGD + “backpropagation”
- Backpropagation: use chain rule to determine updates in ealier layers

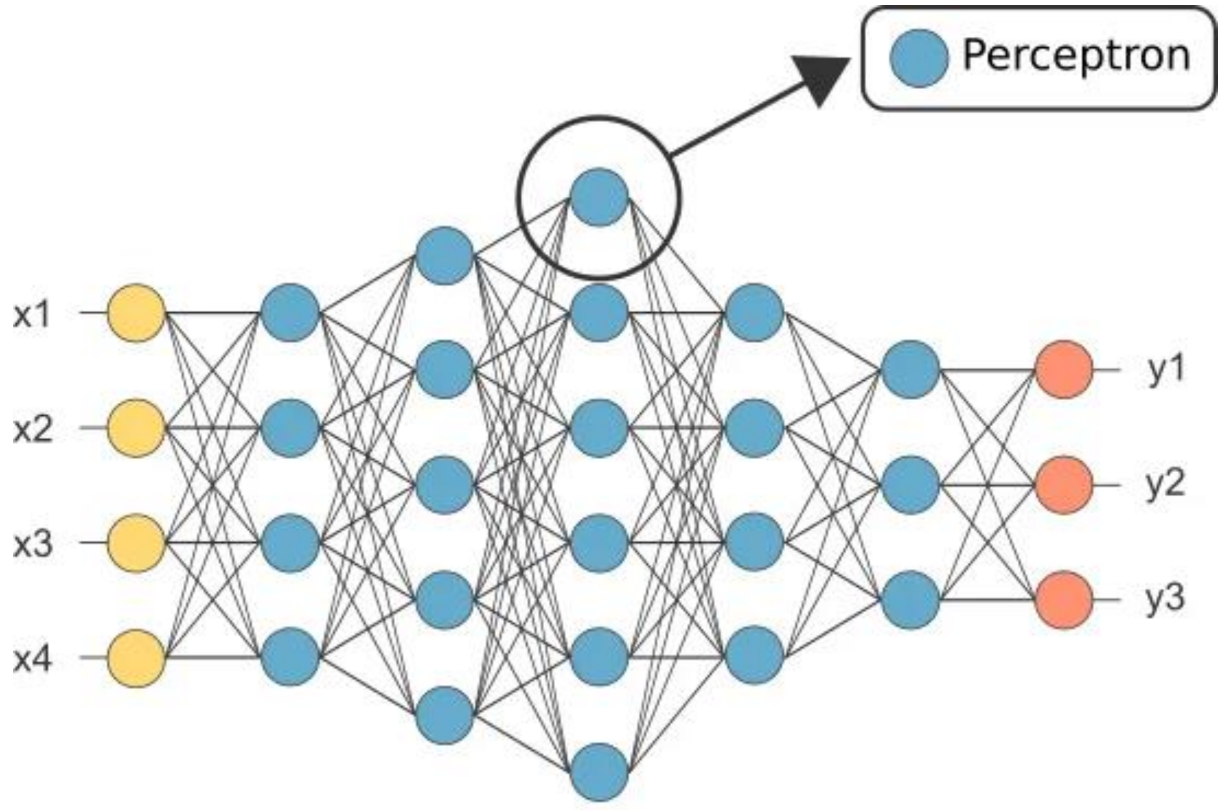


Image source: https://medium.com/@b.terryjack/introduction-to-deep-learning-feed-forward-neural-networks-ffnns-a-k-a-c688d83a309d

Backpropagation in equations

- Loss function $L(\text{model parameters, training data})$
- Parameters: $W^{(1)}, W^{(2)}, \ldots, W^{(k)}, \vec{b}^{(1)}, \vec{b}^{(2)}, \ldots, \vec{b}^{(k)}$
- Training data: $\vec{x}_1, y_1, \ldots, \vec{x}_n, y_n$
- Example: OLS loss for FNN

$$L = \sum_{i=1}^{n} \left(y_i - f_k(W^{(k)}, \vec{b}^{(k)}, \vec{x}_i^{(k)}) \right)^2$$

Backpropagation in equations

- Example: OLS loss for FNN

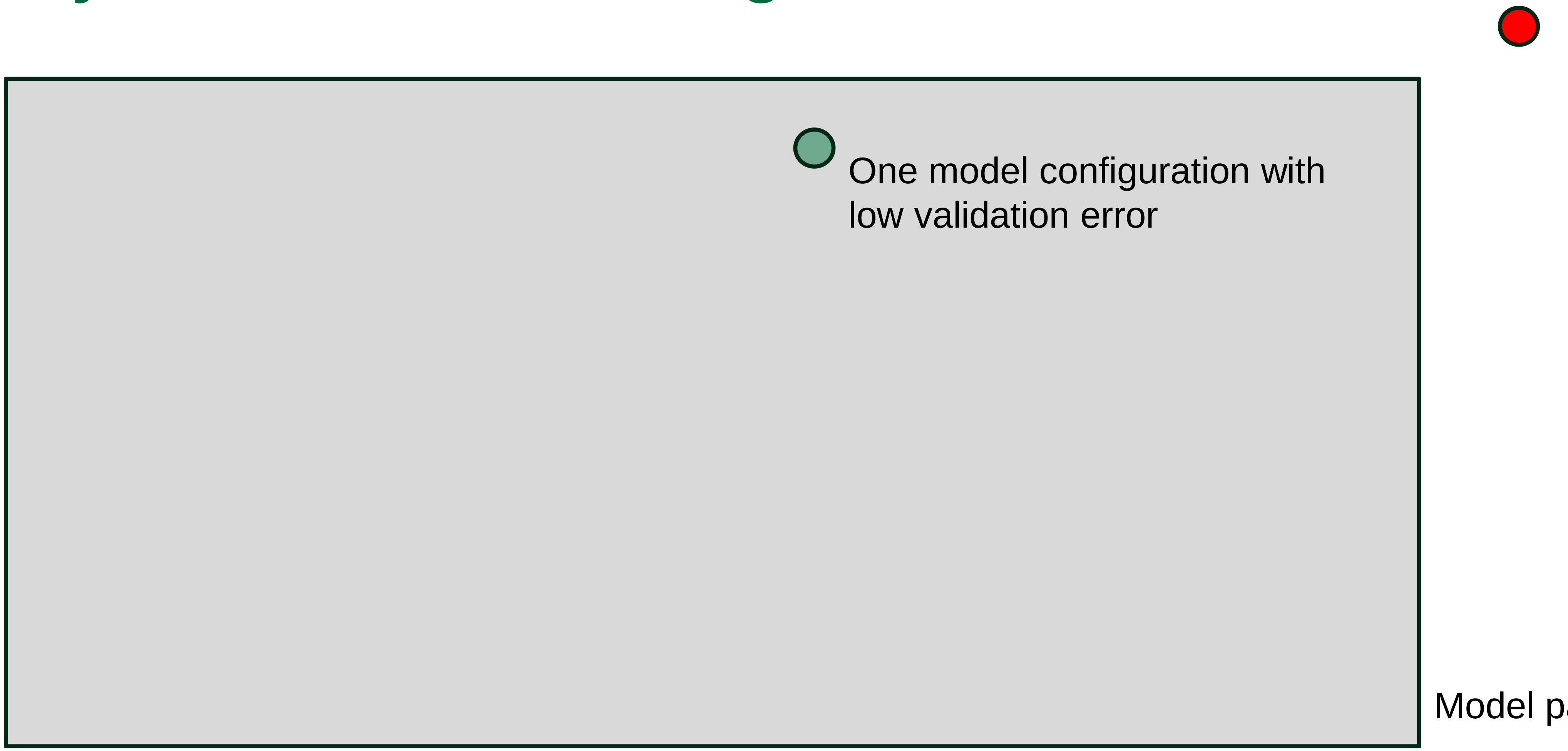
$$L = \sum_{i=1}^{n} \left(y_i - f_k(W^{(k)}, \vec{b}^{(k)}, \vec{x}_i^{(k)}) \right)^2$$

$$L = \sum_{i=1}^{n} \left(y_i - f_k(W^{(k)}, \vec{b}^{(k)}, f_{k-1}(W^{(k-1)}, \vec{b}^{(k-1)}, \vec{x}_i^{(k-1)})) \right)^2$$

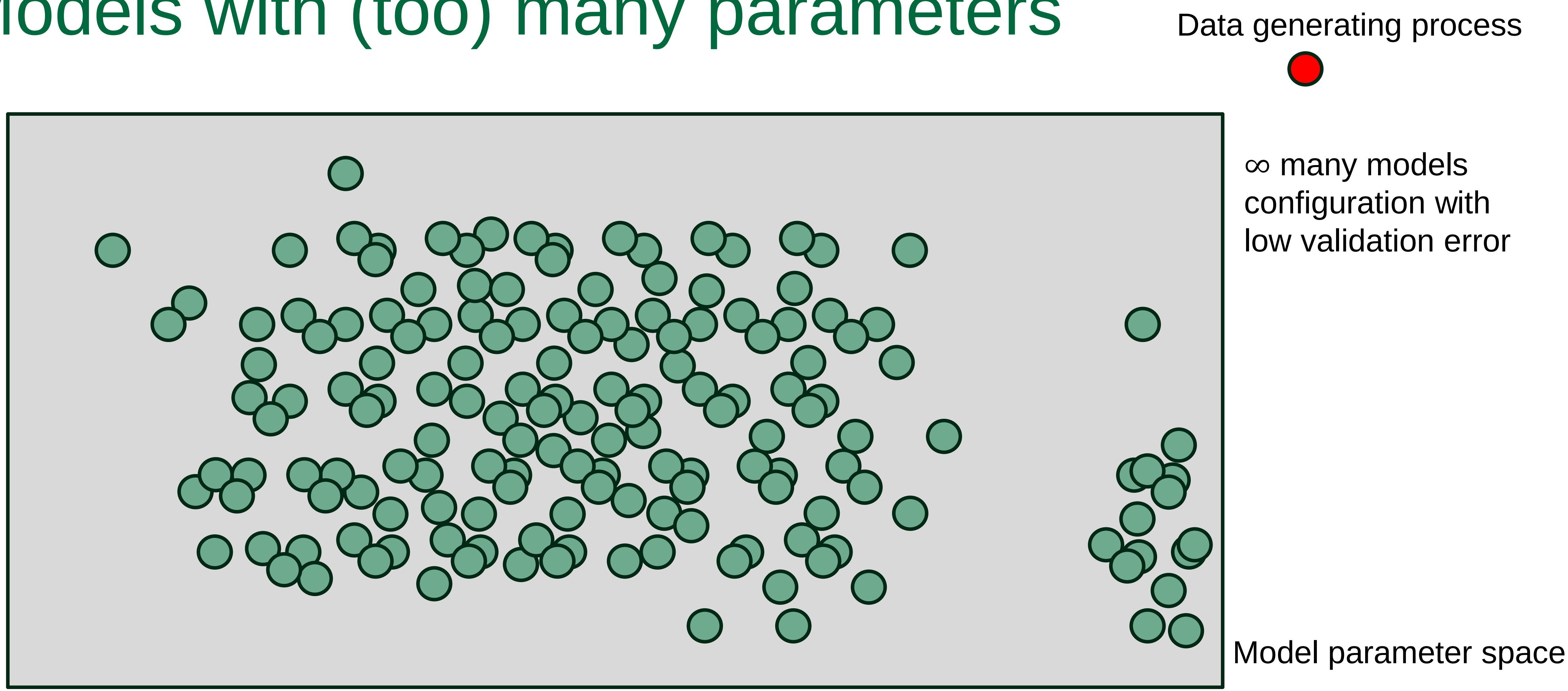
$$L = \sum_{i=1}^{n} \left(y_i - f_k(W^{(k)}, \vec{b}^{(k)}, f_{k-1}(W^{(k-1)}, \vec{b}^{(k-1)}, f_{k-2}(W^{(k-2)}, \vec{b}^{(k-2)}, \vec{x}_i^{(k-2)}))) \right)^2$$

Regularization

Why do we want to regularize?

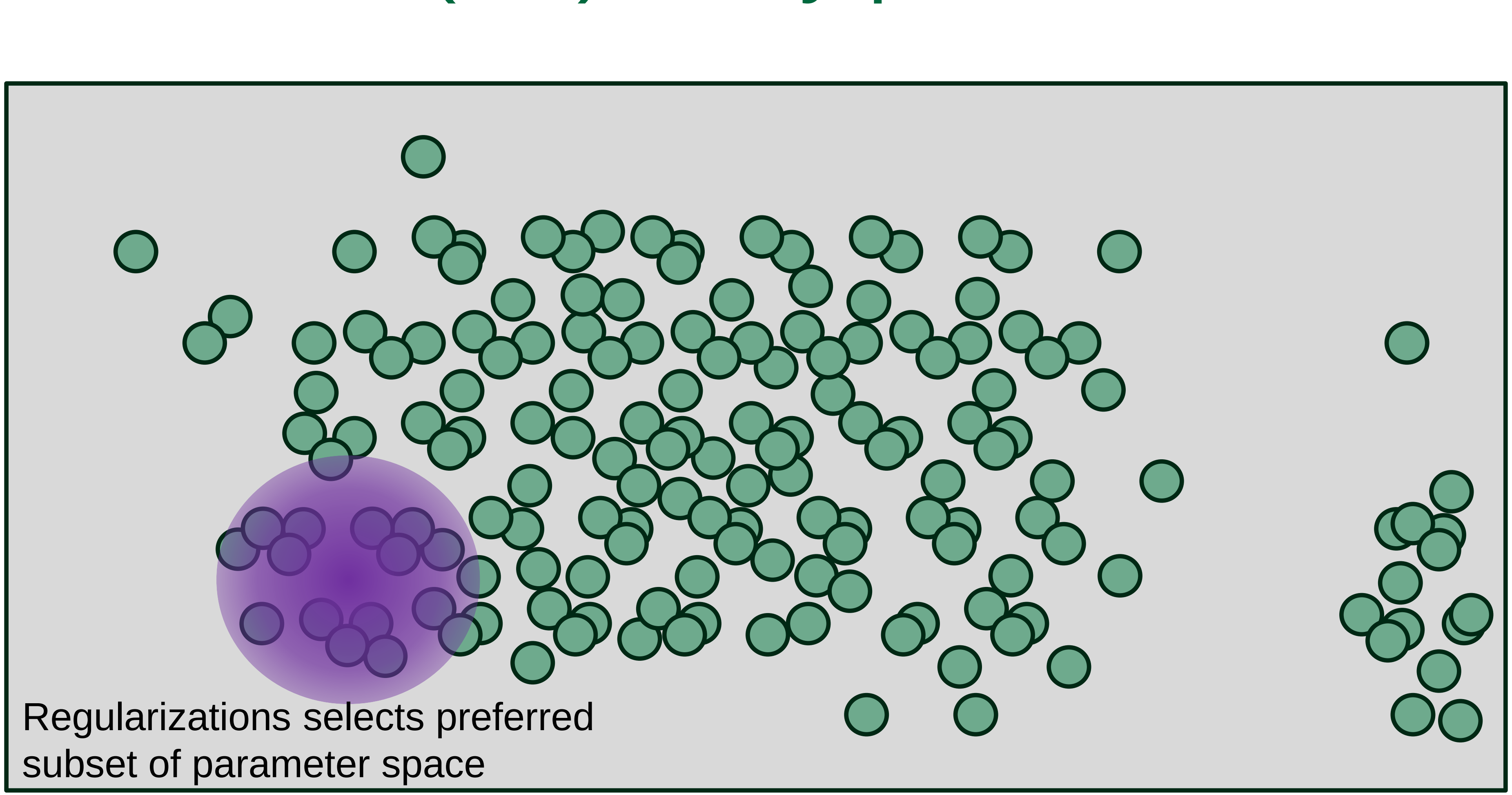


Models with (too) many parameters



Models with (too) many parameters

Data generating process



∞ many models configuration with low validation error

Model parameter space

Regularizers

- Regularizers in loss function:
  - Ridge penalty
  - Lasso penalty
- Regularizers in training algorithm (SGD):
  - Minibatch
  - momentum

Specific regularization methods for neural networks

- Early stopping
- Drop-out learning
- Weight pruning
- Architecture constraints

Early stopping

- Stop training before 0 training error is reached

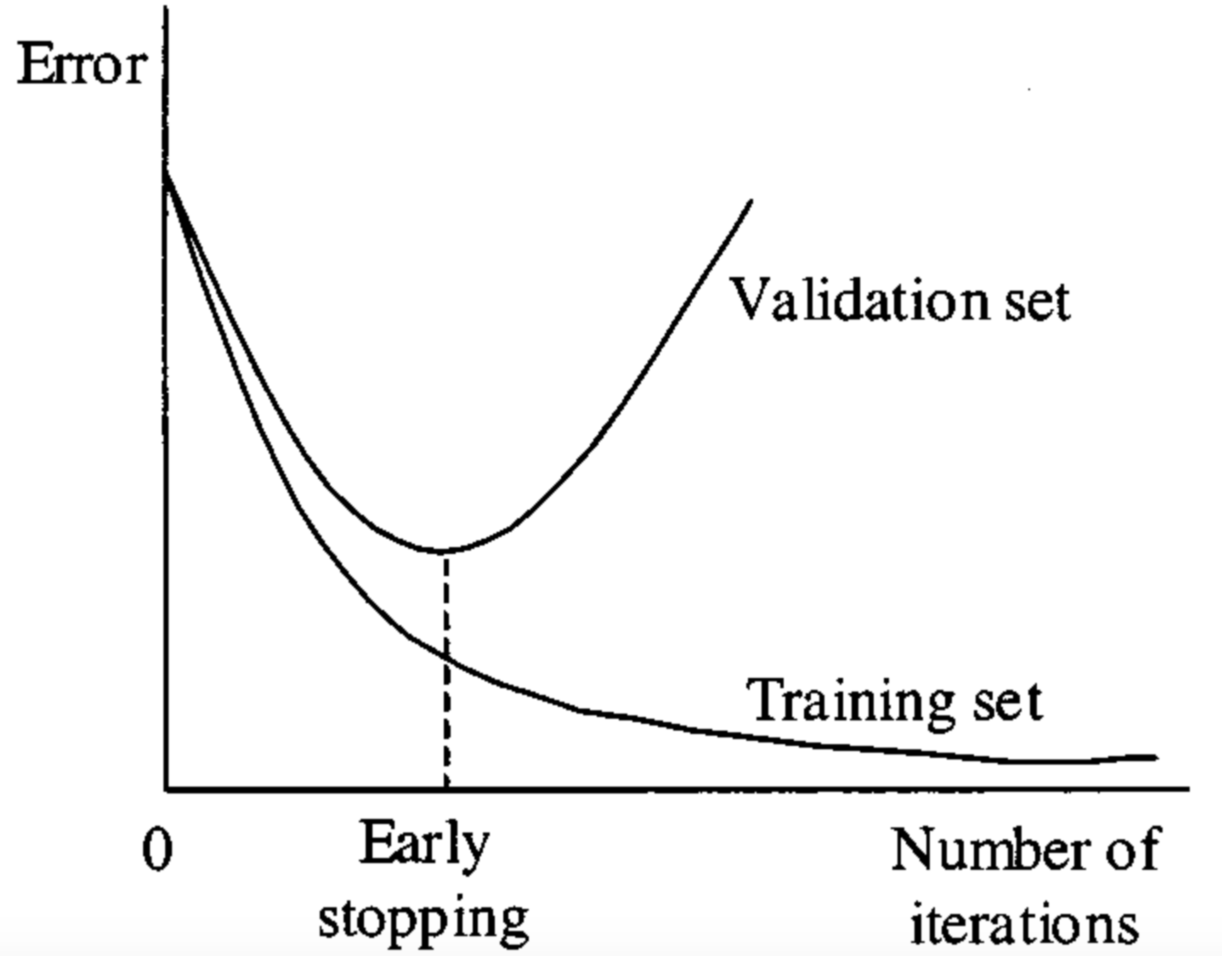


Image source: https://paperswithcode.com/method/early-stopping

Drop-out learning

- Ignore a randomly chosen set of nodes and their associated weights in each training instance of SGD

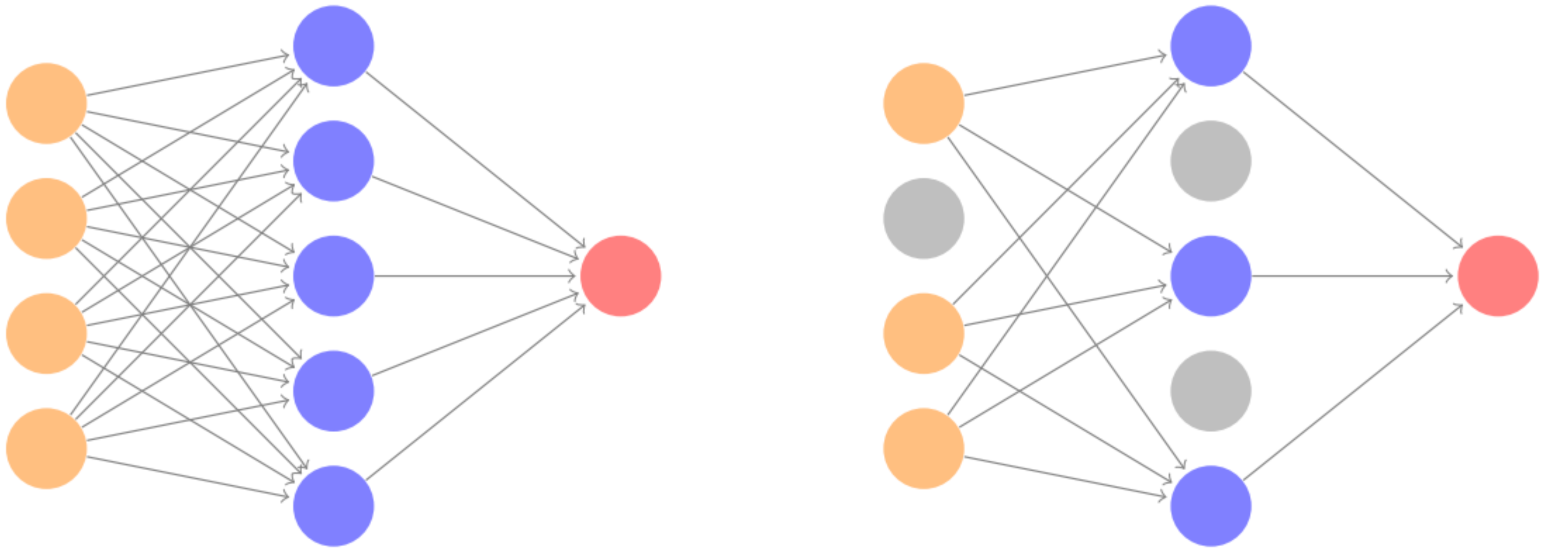
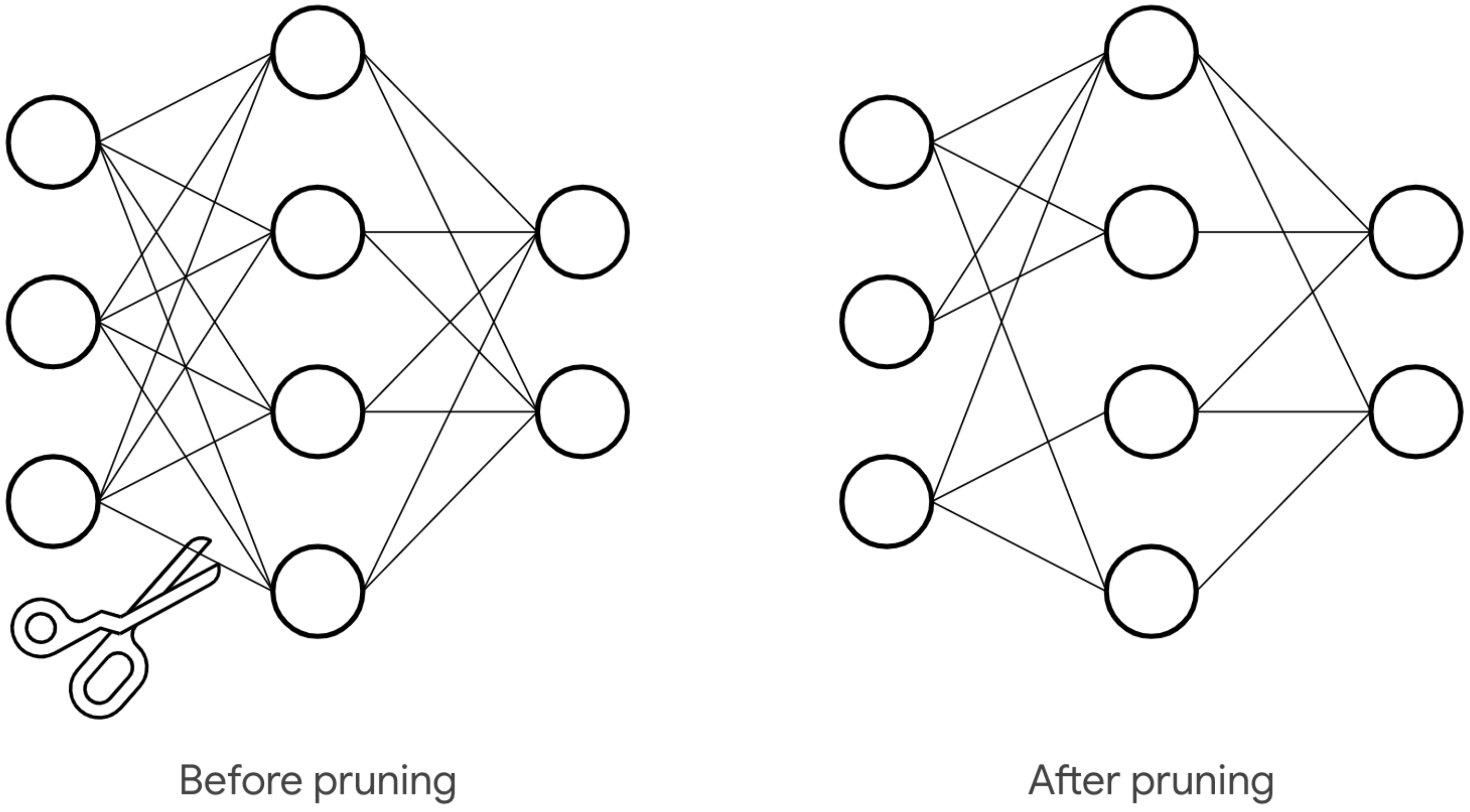


FIGURE 10.19. *Dropout Learning. Left: a fully connected network. Right: network with dropout in the input and hidden layer. The nodes in grey are selected at random, and ignored in an instance of training.*

Image source: ISLP

Weight pruning

- Set smallest* weights to zero after one* epoch



*other variations also exist

Image source: https://blog.tensorflow.org/2019/05/tf-model-optimization-toolkit-pruning-API.html

Architecture constraints

- Set some weights to zero before training and keep them fixed
- Force some weights to have identical values
- Example: Convolutional neural networks

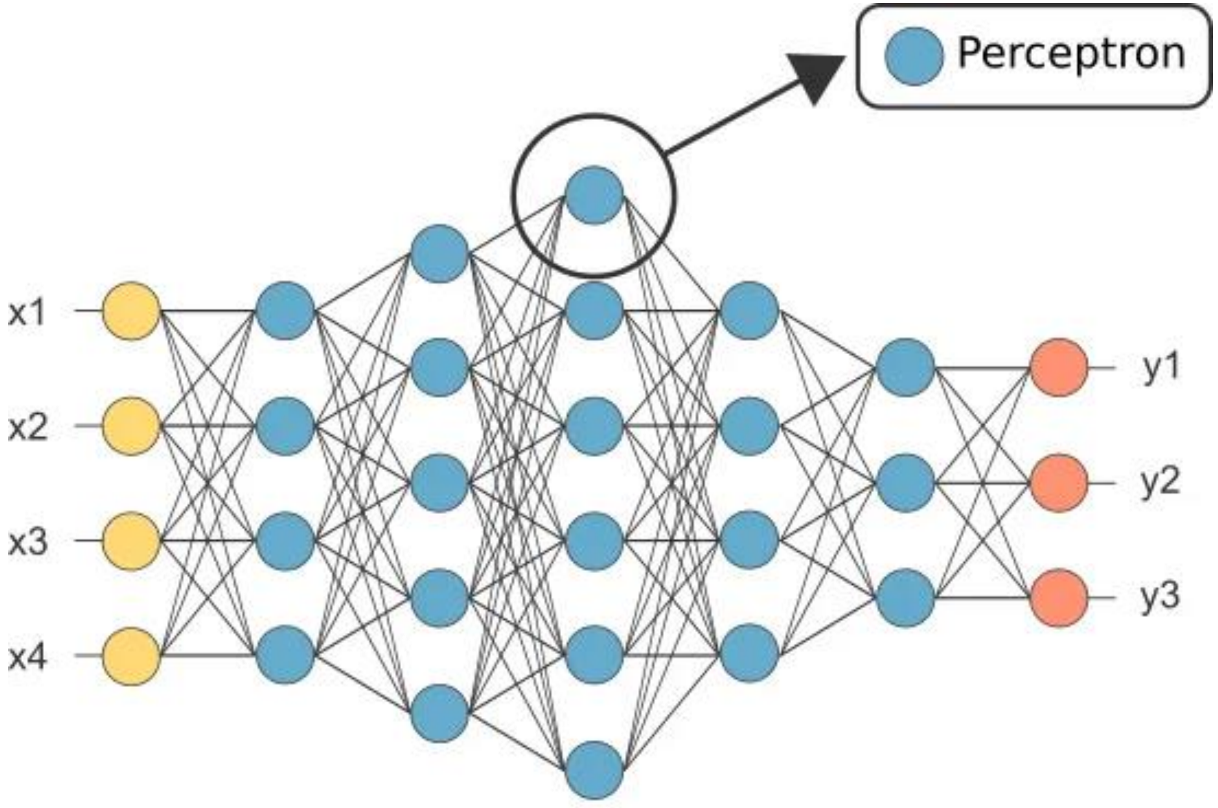


Image source: https://medium.com/@b.terryjack/introduction-to-deep-learning-feed-forward-neural-networks-ffnns-a-k-a-c688d83a309d

Convolutional neural networks

Convolutional neural networks (CNN)

- Used for image classification
- Computer vision: convolutions with small filters/ kernels

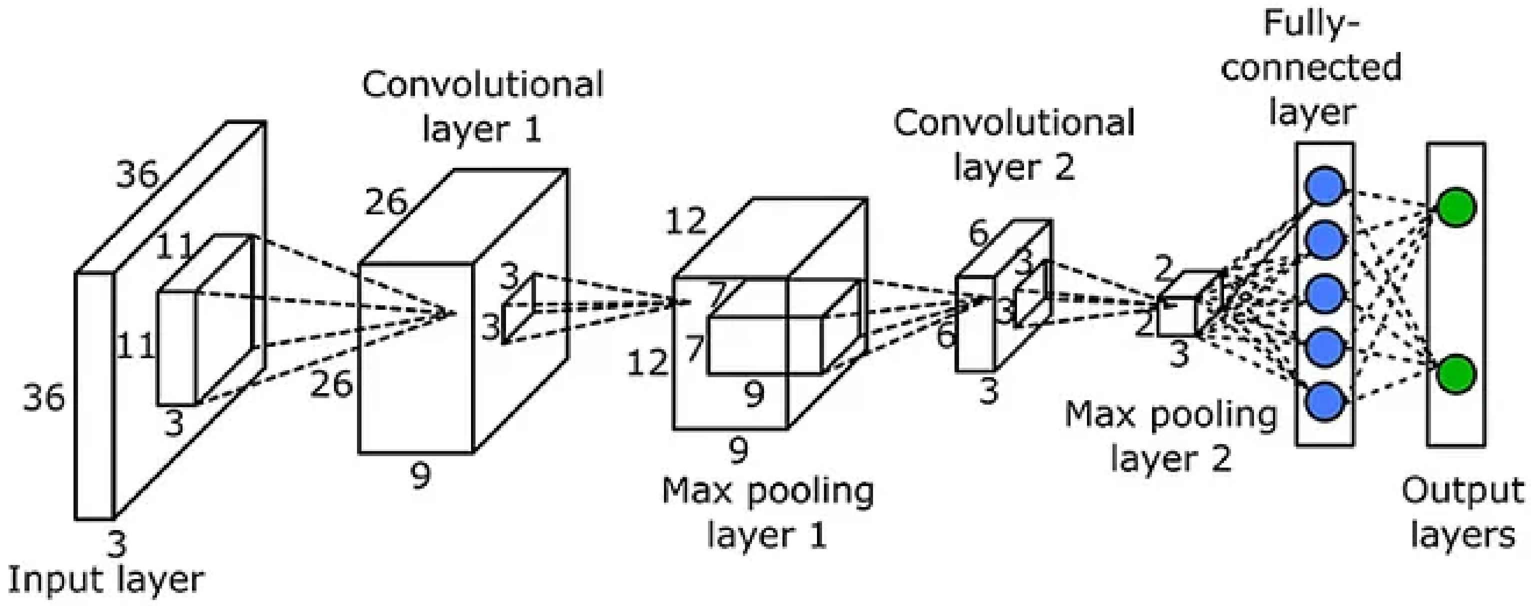
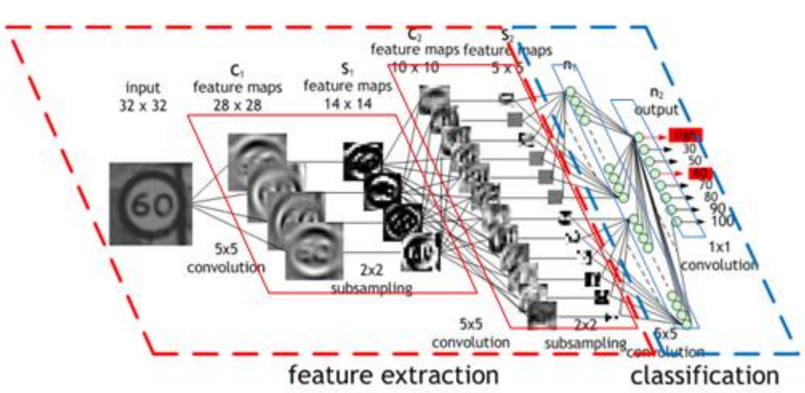


Image source: https://nafizshahriar.medium.com/what-is-convolutional-neural-network-cnn-deep-learning-b3921bdd82d5

Feature extraction



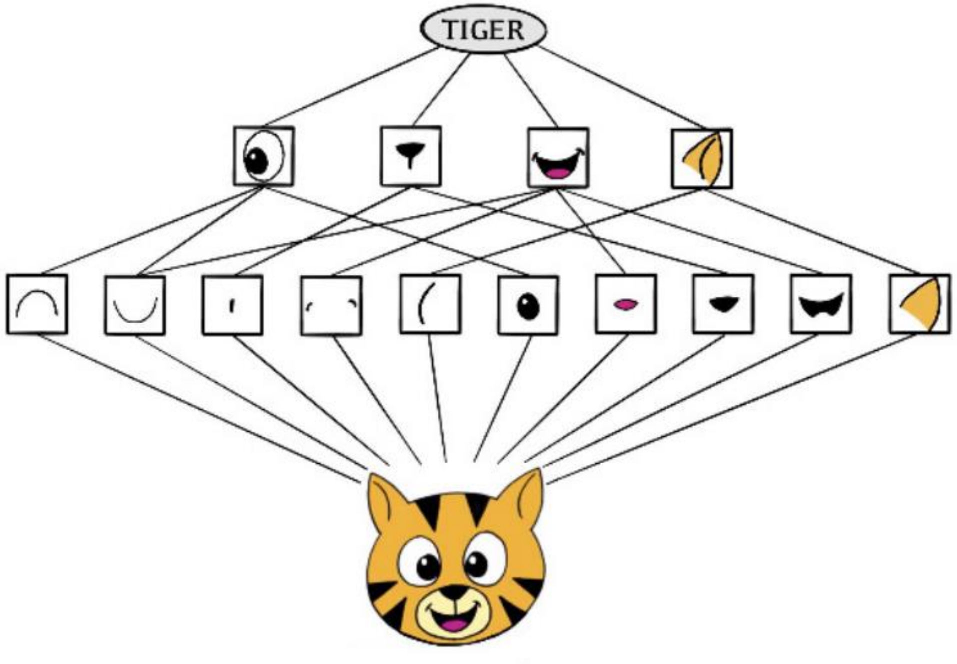


Image source: ISLP, https://developer.nvidia.com/discover/convolutional-neural-network

Convolution with filters

$$\text{Original Image} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \end{bmatrix}.$$

Now consider a 2×2 filter of the form

$$\text{Convolution Filter} = \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}.$$

When we *convolve* the image with the filter, we get the result[7]

$$\text{Convolved Image} = \begin{bmatrix} a\alpha + b\beta + d\gamma + e\delta & b\alpha + c\beta + e\gamma + f\delta \\ d\alpha + e\beta + g\gamma + h\delta & e\alpha + f\beta + h\gamma + i\delta \\ g\alpha + h\beta + j\gamma + k\delta & h\alpha + i\beta + k\gamma + l\delta \end{bmatrix}.$$

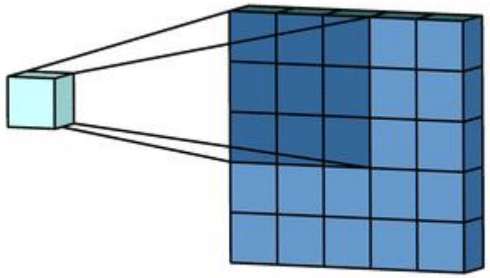
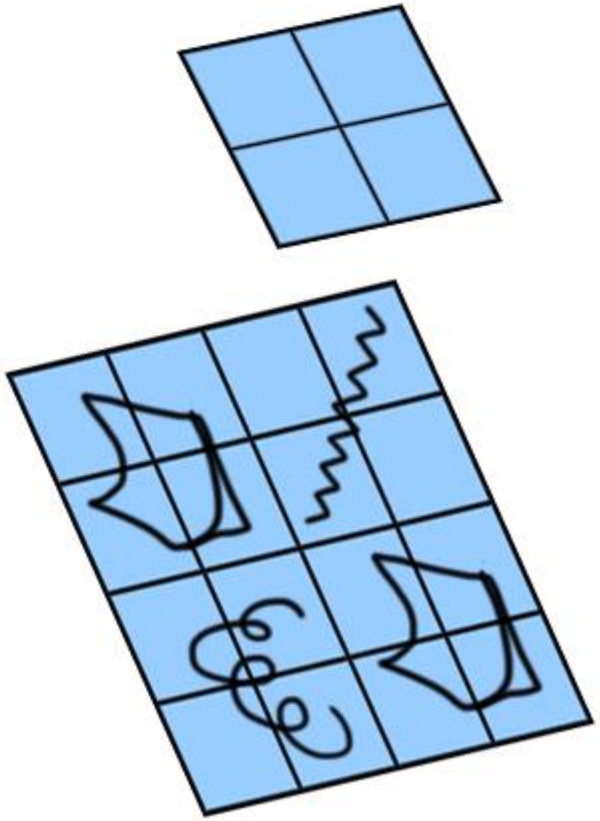


Image source: ISLP, https://medium.com/@abhishekjainindore24/all-about-convolutions-kernels-features-in-cnn-c656616390a1

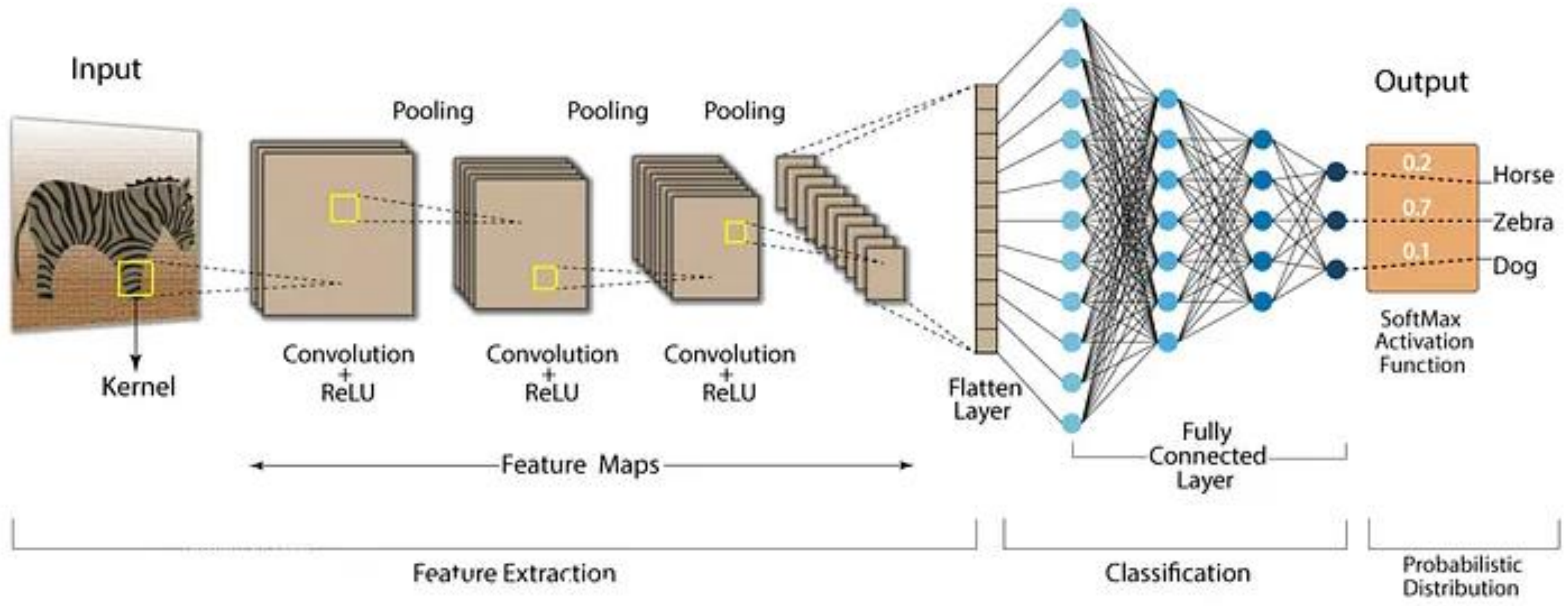
Convolution layers

Original	Gaussian Blur	Sharpen	Edge Detection
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$
			



Pooling layers

- Reduce image size in hidden layers



Example

https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html