CSE 575
Statistical Machine Learning

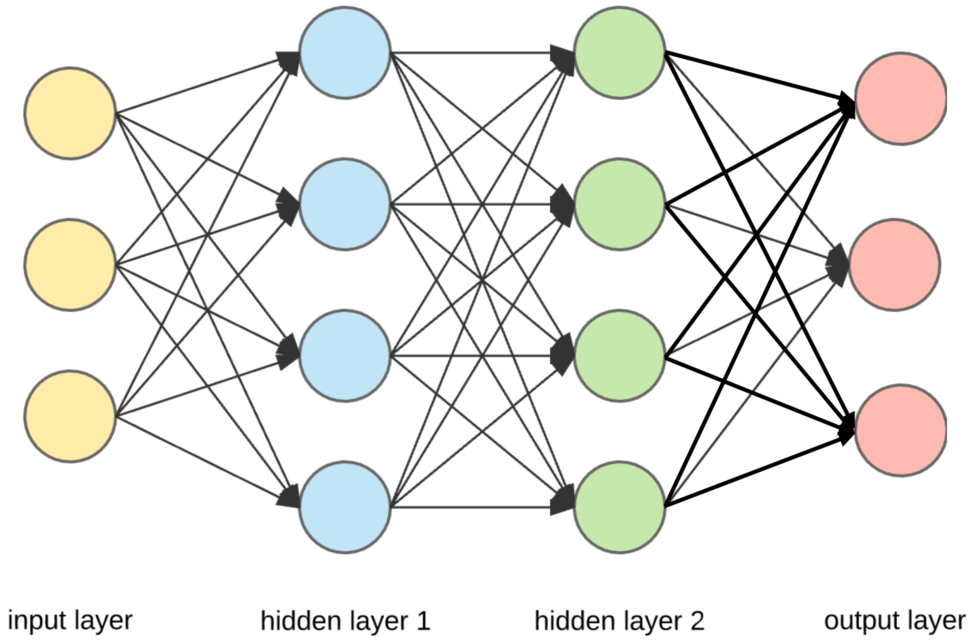
Lecture 14

YooJung Choi

Fall 2022

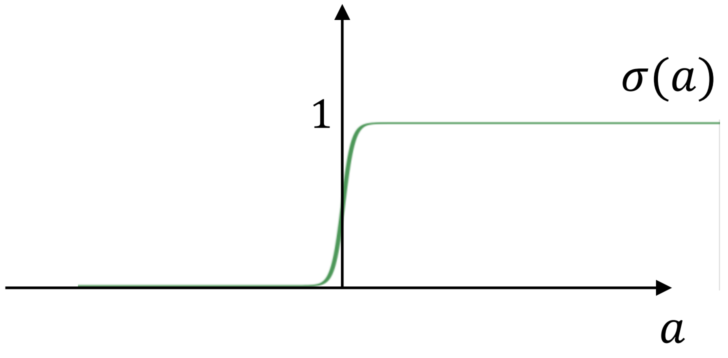
Deep learning: choices

- Output layer & error/loss function
  - Often determined by the task & data (regression, binary classification, ...)
- Hidden layer activation functions
- Network architecture
- Improving training
  - Optimization techniques,
  - Input standardization,
  - Also influences choice of activation functions



Hidden layer activation functions

- So far, we considered the sigmoid activation for hidden layer units



$$\sigma'(a) = \sigma(a)(1 - \sigma(a)) \leq (0.5)(0.5) = 0.25$$

$$\begin{aligned}
\frac{\partial E}{\partial W_{..}^{[1]}} &= \delta_{..}^{[1]} \cdot z_{..}^{[0]} \\
&= \left(\sigma'\left(a_{..}^{[1]}\right) \sum_k \delta_k^{[2]} W_{..}^{[2]}\right) \cdot z_{..}^{[0]} \\
&= \left(\sigma'\left(a_{..}^{[1]}\right) \sum_k \left(\sigma'\left(a_{..}^{[2]}\right) \sum_k \delta_k^{[3]} W_{..}^{[3]}\right) W_{..}^{[2]}\right) \cdot z_{..}^{[0]}
\end{aligned}$$

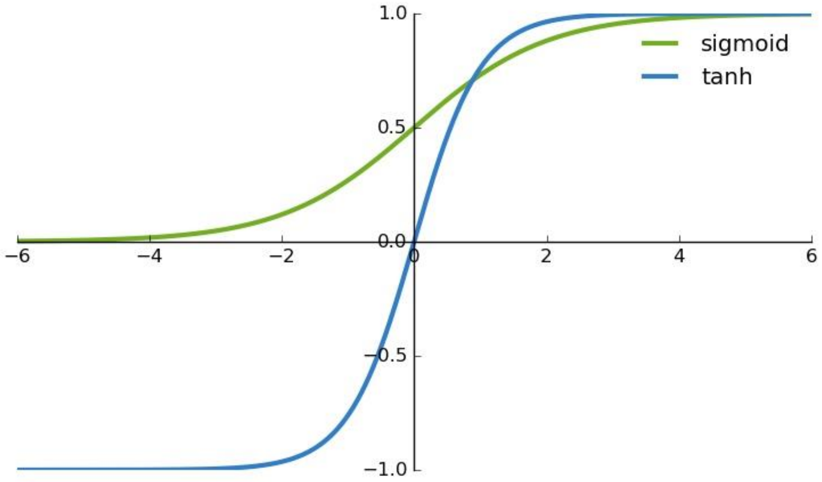
Vanishing gradient problem

- The deeper the network, the smaller the gradients become in the early layers
- i.e. harder to train the early layers (gradient descent will make very small updates)

Hidden layer activation functions

- Hyperbolic tangent: $$\tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)}$$

$$\tanh'(a) = 1 - (\tanh(a))^2$$

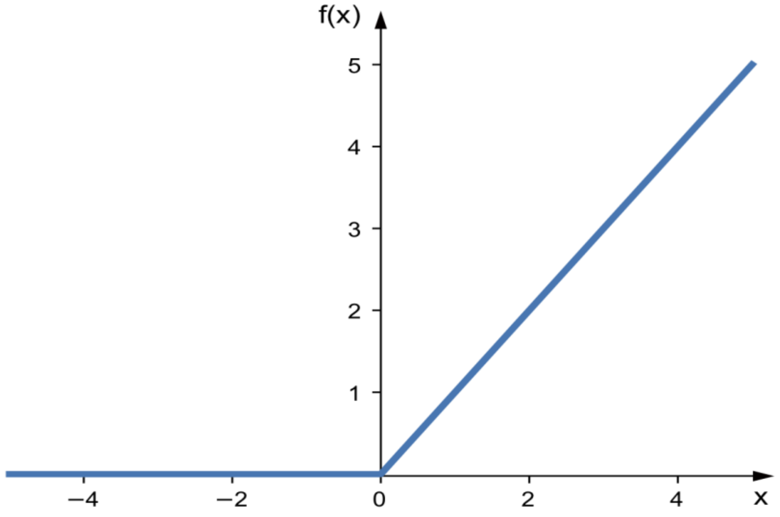


- Rectified linear unit (ReLU)

$$\mathrm{ReLU}(a) = \begin{cases} 0 & \text{if } a \leq 0 \\ a & \text{if } a > 0 \end{cases} = \max\{0, a\}$$

Most widely used activation function

What happens if the a is always negative for some neuron?
"dead ReLU" : no parameter update happens



Hidden layer activation functions

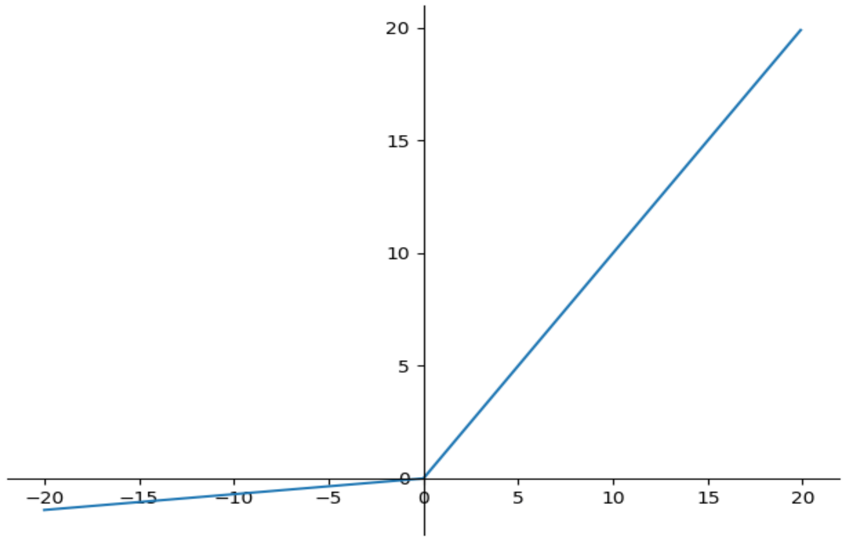
- Leaky Rectified linear unit (LReLU)

$$\mathrm{LReLU}(a) = \begin{cases} \alpha \cdot a & \text{if } a \leq 0 \\ a & \text{if } a > 0 \end{cases} = \max\{\alpha \cdot a, a\}$$

- Often $\alpha = 0.01$ or similar

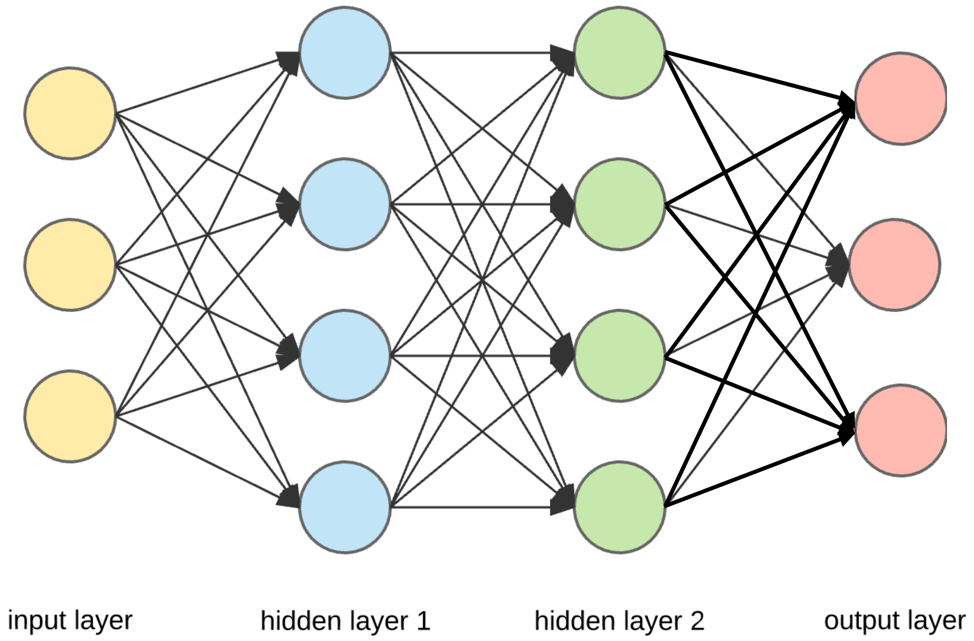
- If α is randomly sampled during training,

  "randomized" leaky ReLU



Deep learning: choices

- Output layer & error/loss function
  - Often determined by the task & data (regression, binary classification, ...)
- Hidden layer activation functions
- Network architecture
- Improving training
  - Optimization techniques,
  - Input standardization,
  - Also influences choice of activation functions



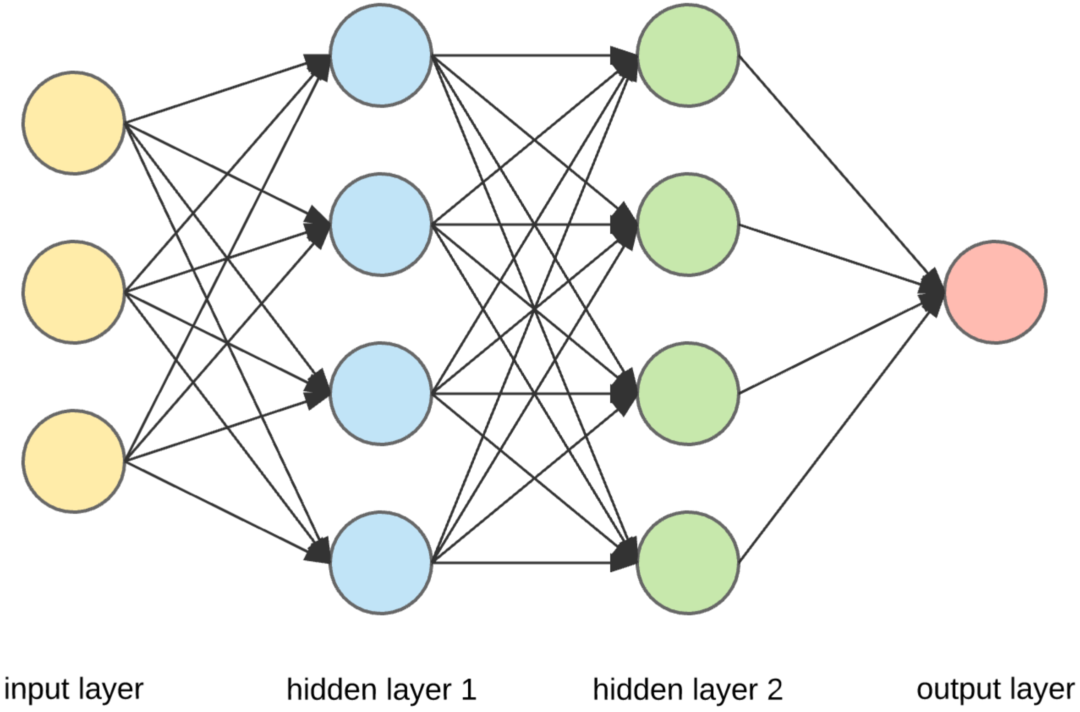
Fully-connected NN for images

An image is simply a matrix of numbers

256x256x3

Dog or not?

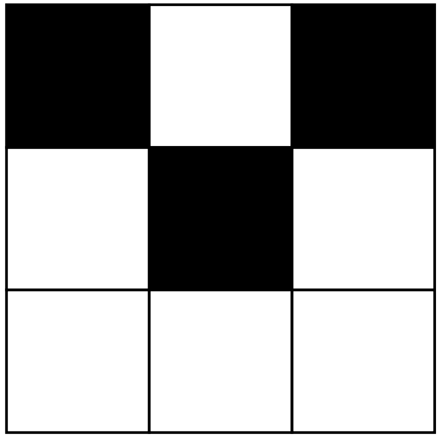
parameters in layer 1:
256x256x3x*width*



P("Dog")

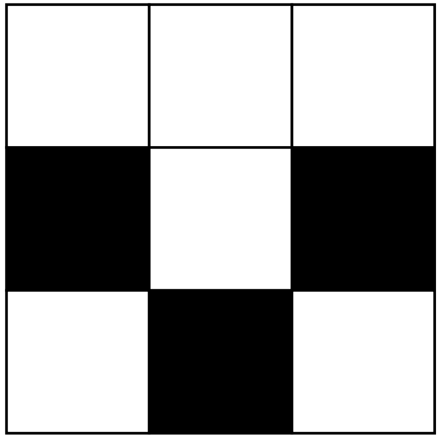
Images as vectors

Suppose we want to recognize a V-shape *anywhere in the image*



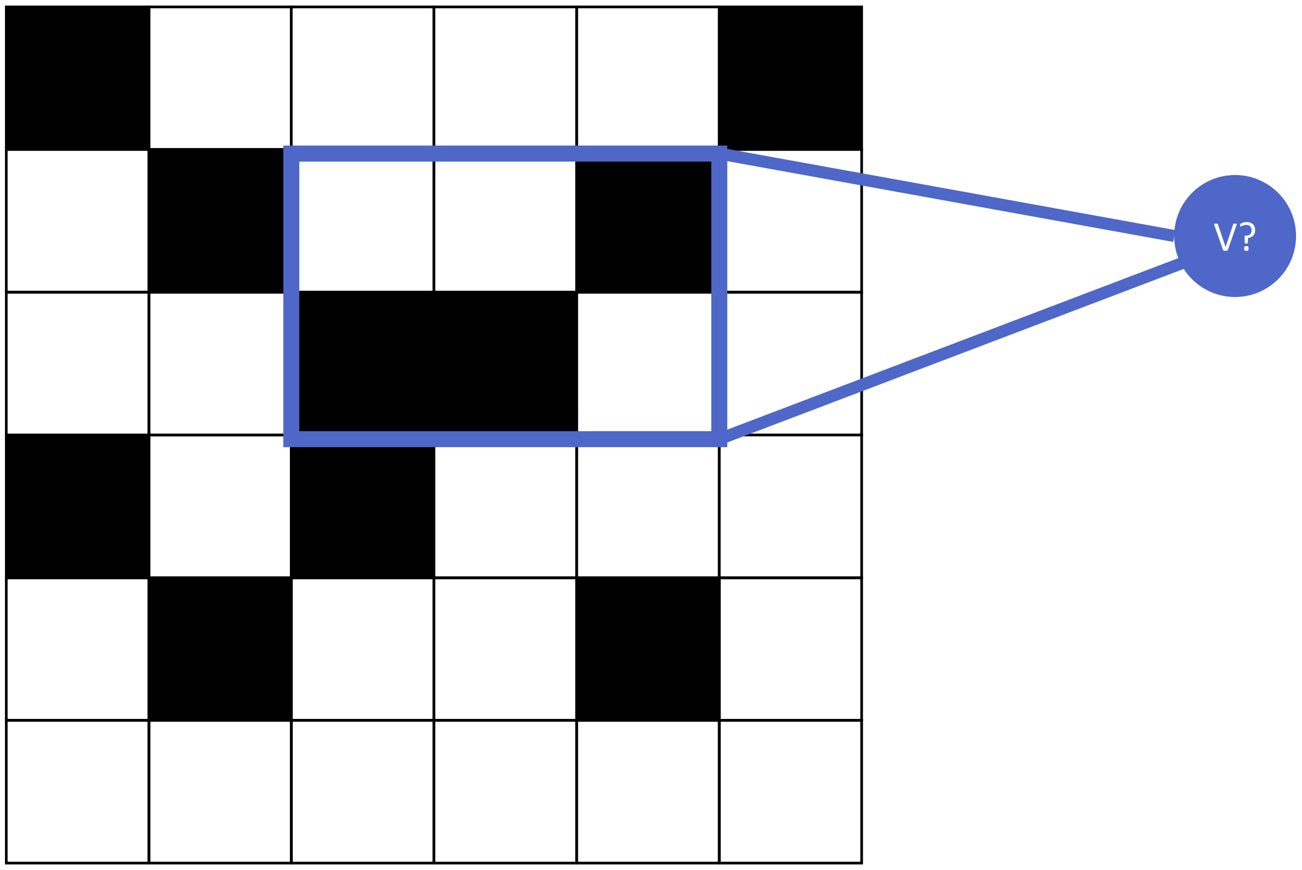
Flattened to a vector

Spatial information is lost!



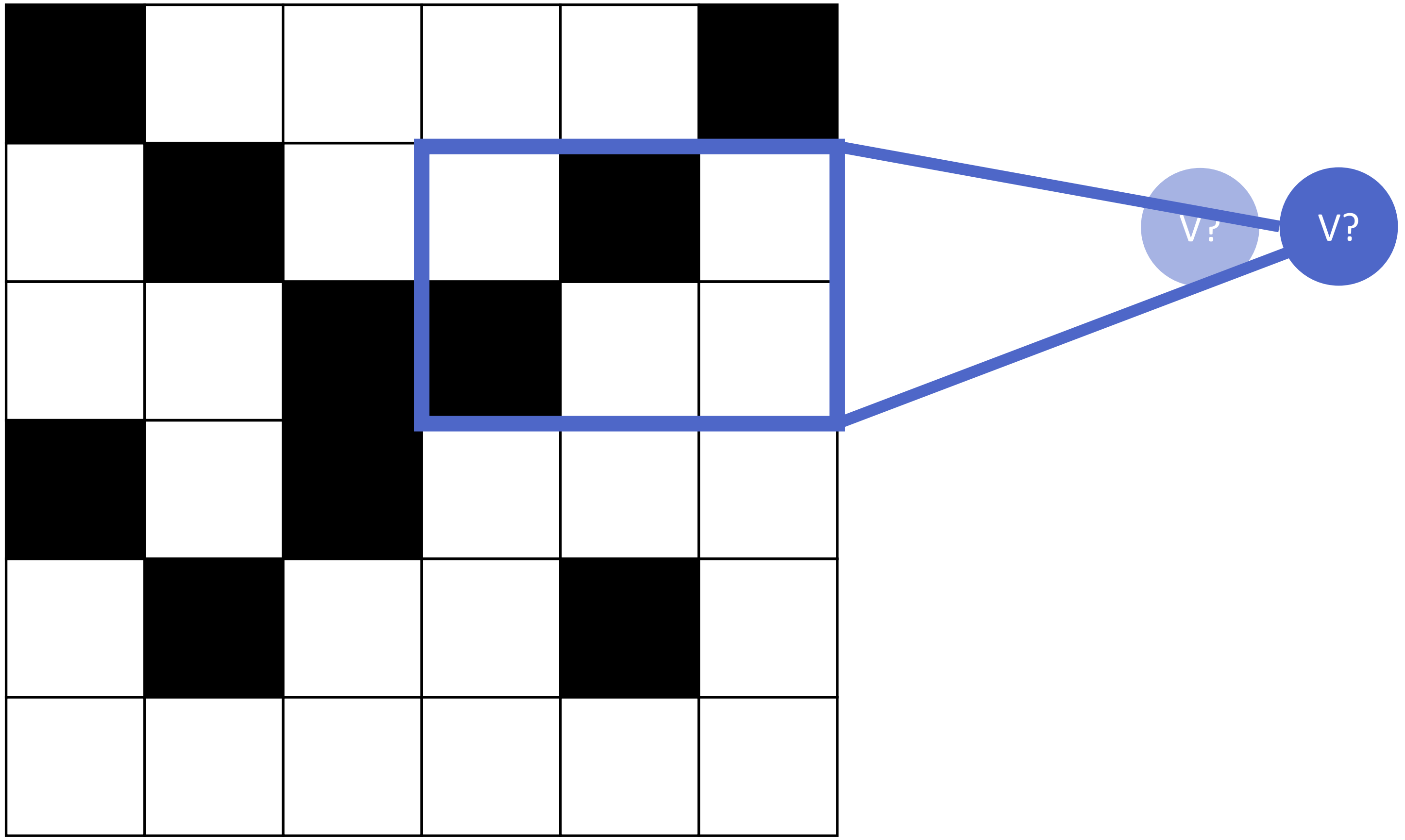
How to use spatial structure?

Idea: neurons in the hidden layer
see “regions” of the input



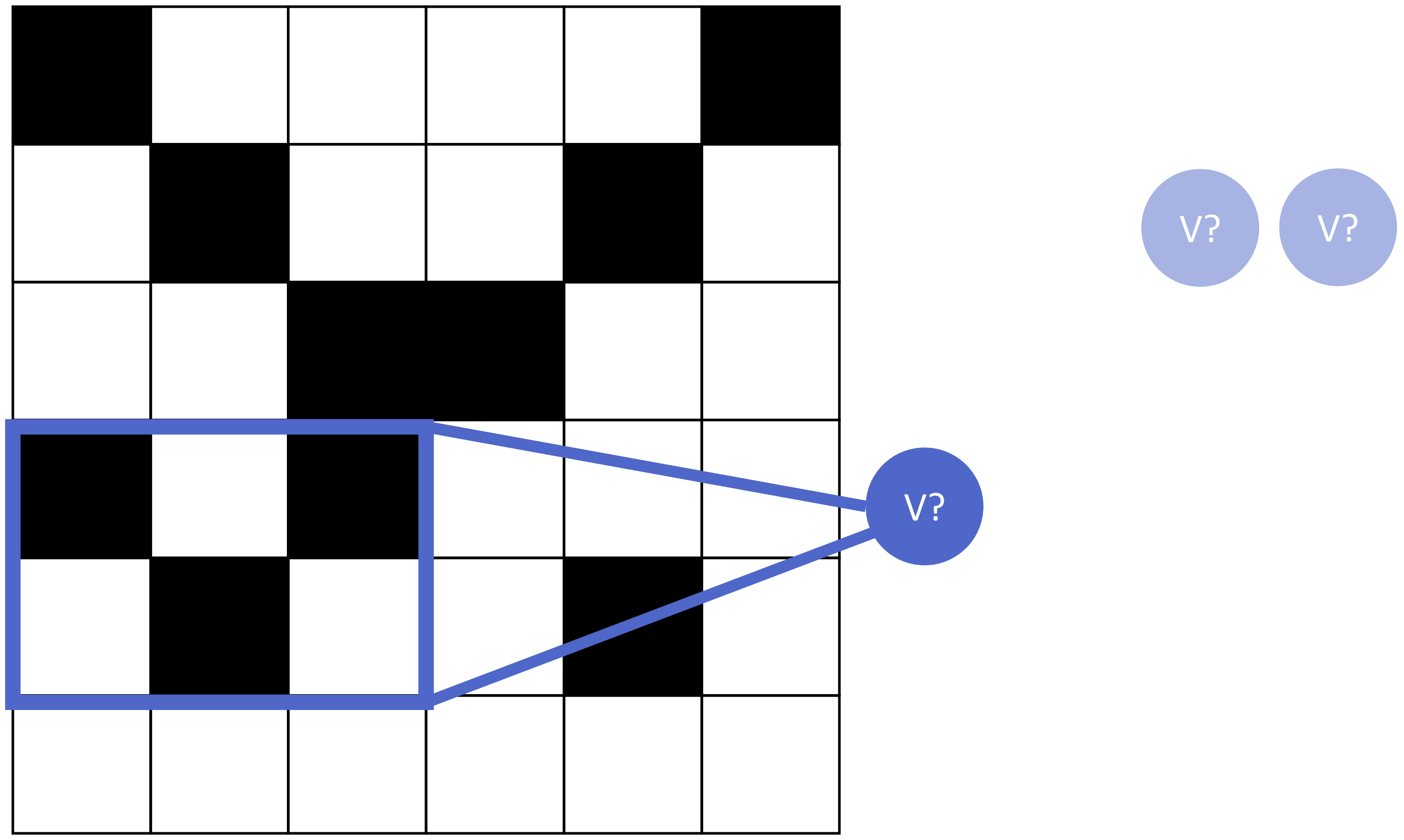
How to use spatial structure?

Idea: neurons in the hidden layer
see “regions” of the input



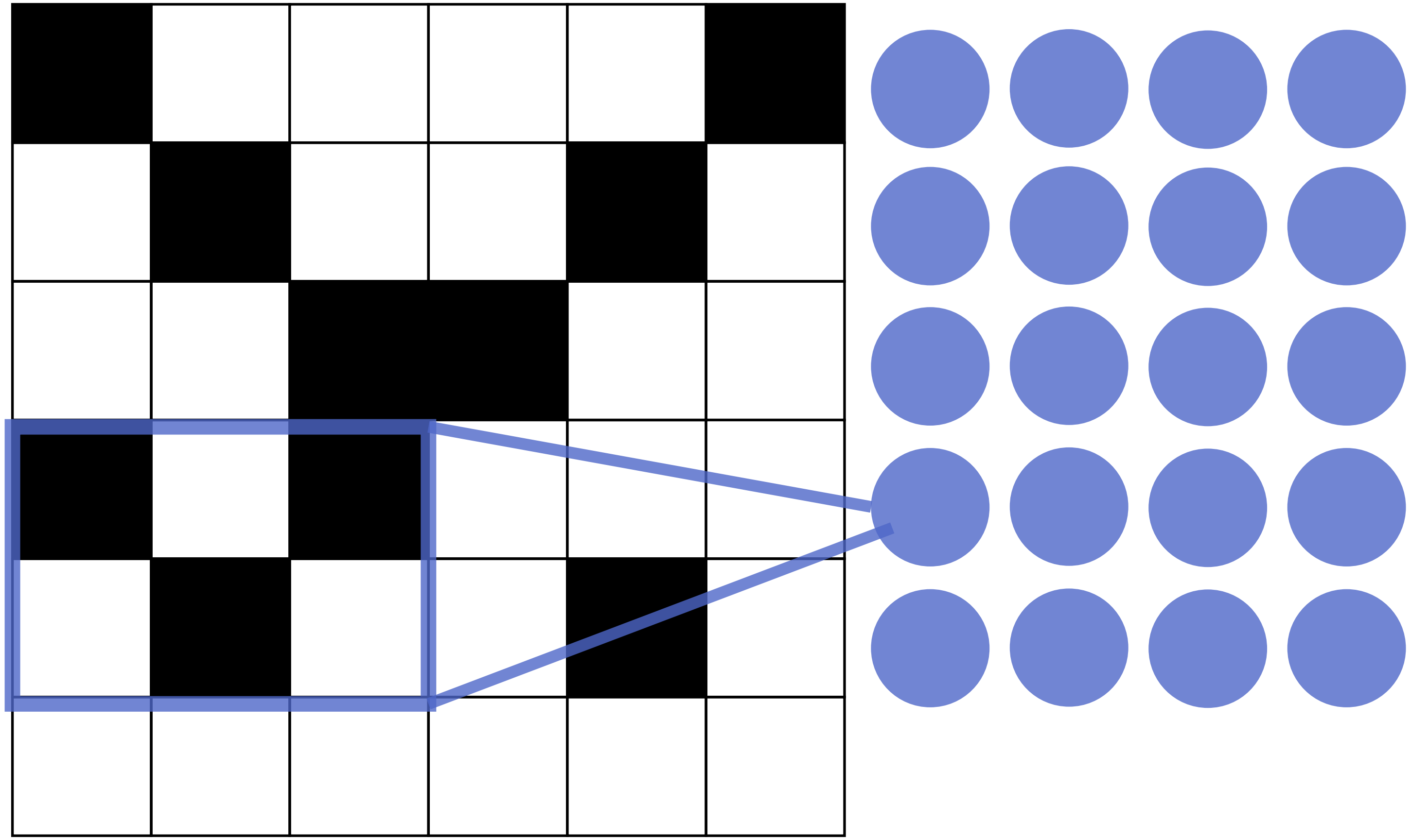
How to use spatial structure?

Idea: neurons in the hidden layer
see “regions” of the input



How to use spatial structure?

Idea: neurons in the hidden layer see “regions” of the input

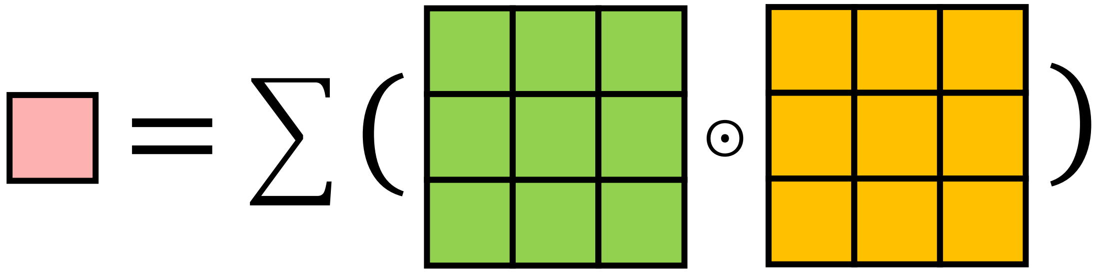


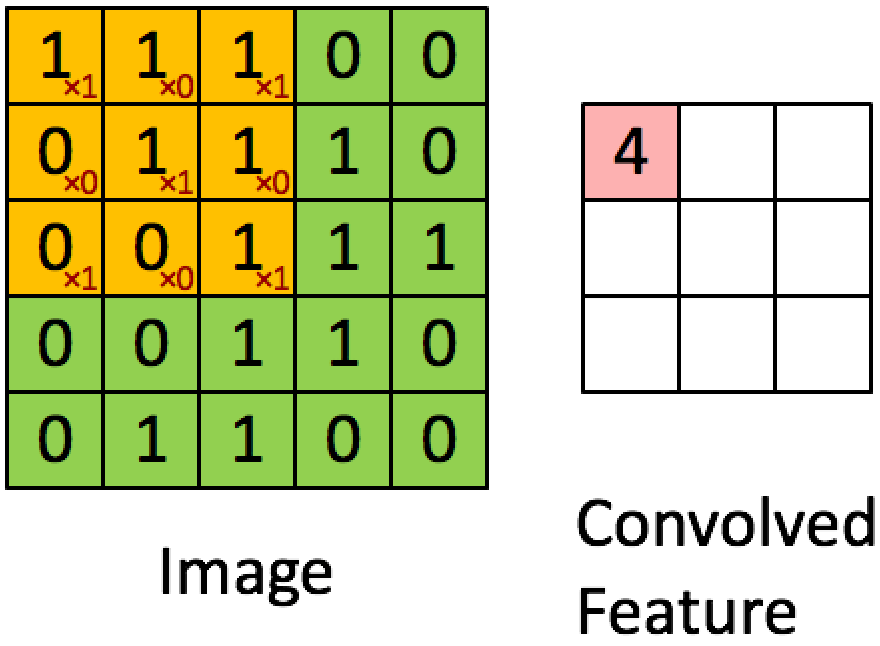
A sliding window (“filter”) applied to each “patch” of input to get a neuron output

e.g. each neuron outputs whether there’s a V shape in the corresponding patch

Convolution

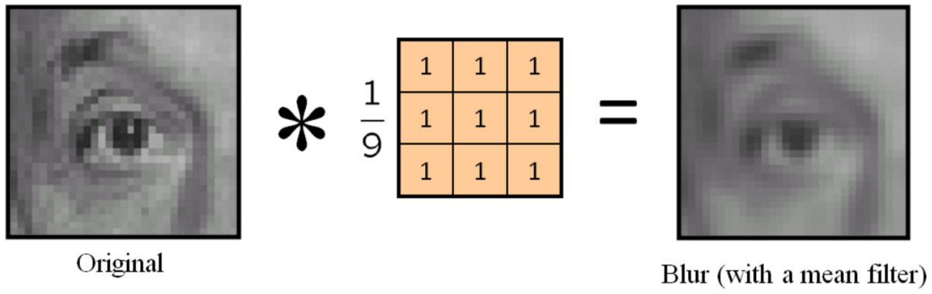
Element-wise product then sum

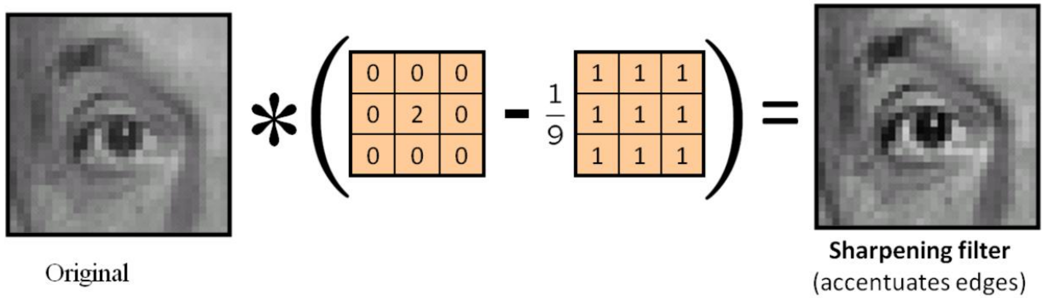


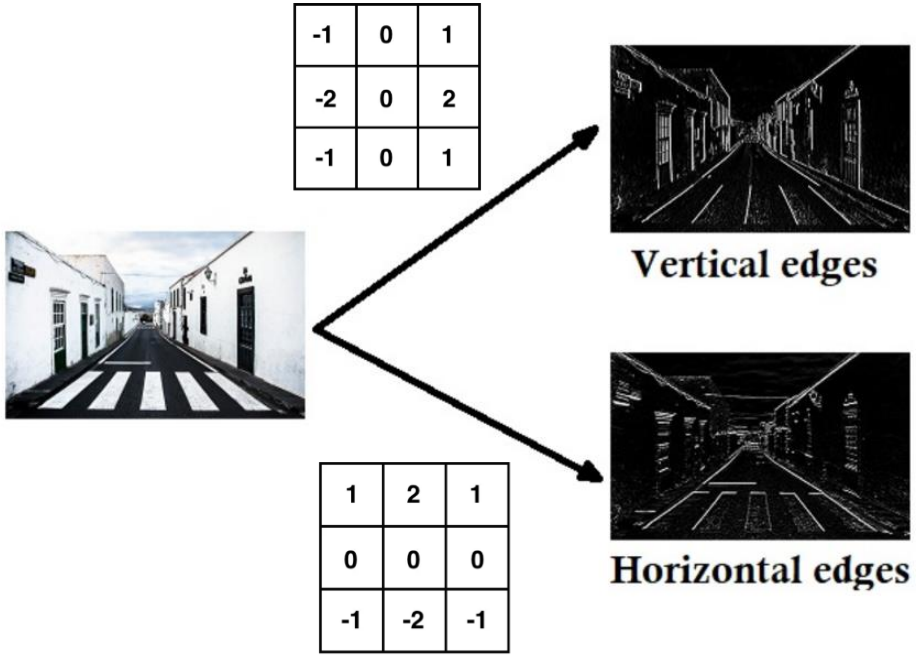


- Each filter (matrix of parameters) extracts *local features*
- Outcome: neurons that see different patches of the image & share the same parameters (*filter*)
- Use multiple filters to extract different features

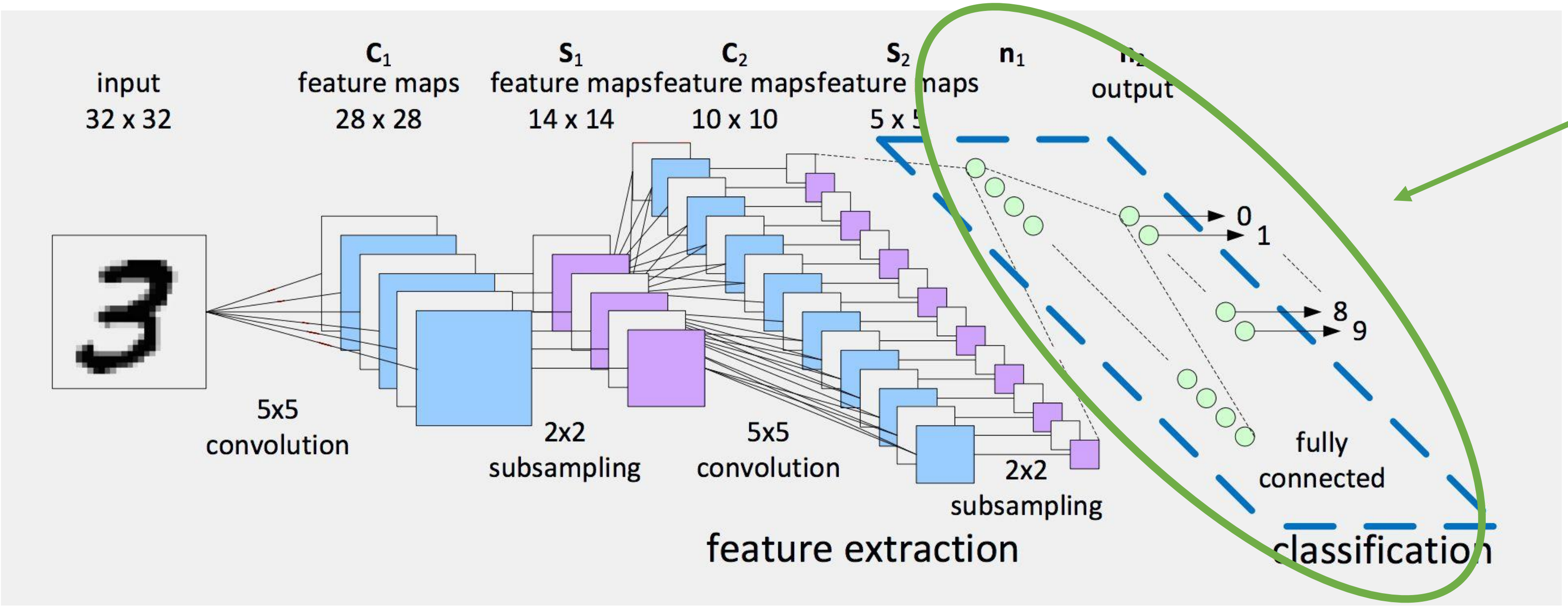
Feature maps







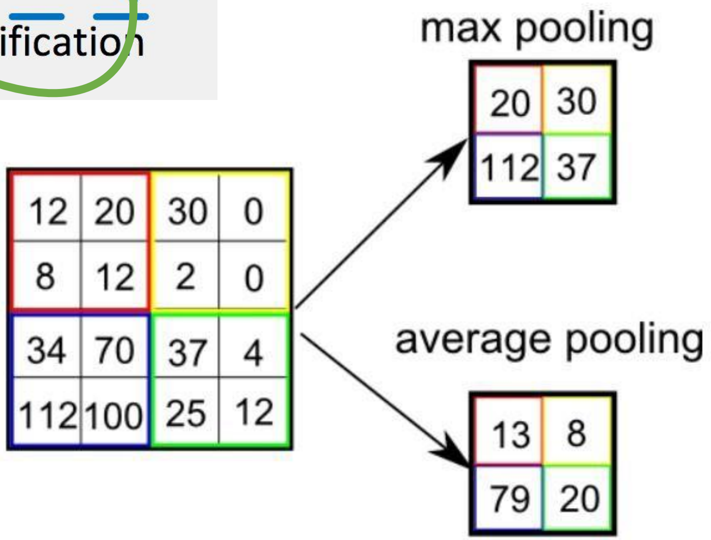
Convolutional neural networks



Can be swapped out for different downstream tasks
(object detection, image segmentation, etc.)

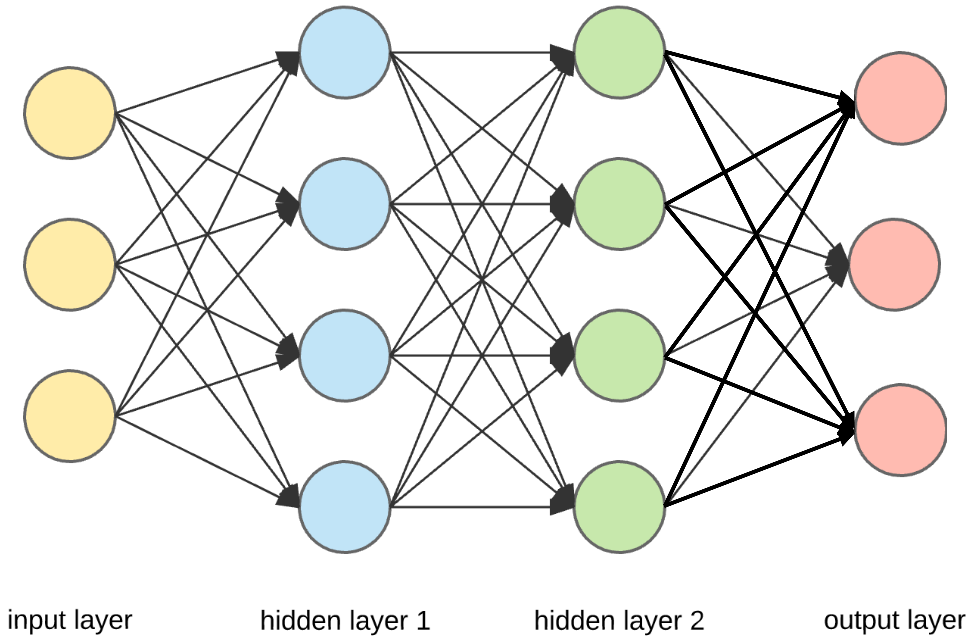
Convolution + non-linearity (often ReLU) + *pooling*

Down-sampling / dimensionality reduction



Deep learning: choices

- Output layer & error/loss function
  - Often determined by the task & data (regression, binary classification, ...)
- Hidden layer activation functions
- Network architecture
- Improving training
  - Optimization techniques,
  - Input standardization,
  - Also influences choice of activation functions



Optimization

Gradient descent

$$\boldsymbol{W}^{(new)} \leftarrow \boldsymbol{W}^{(old)} - \eta \sum_{n=1}^{N} \nabla E_n(\boldsymbol{W})$$

+ Each step informed by the entire data
- Slow: computing gradient is expensive for large data

Stochastic gradient descent

$$\boldsymbol{W}^{(new)} \leftarrow \boldsymbol{W}^{(old)} - \eta \nabla E_n(\boldsymbol{W})$$

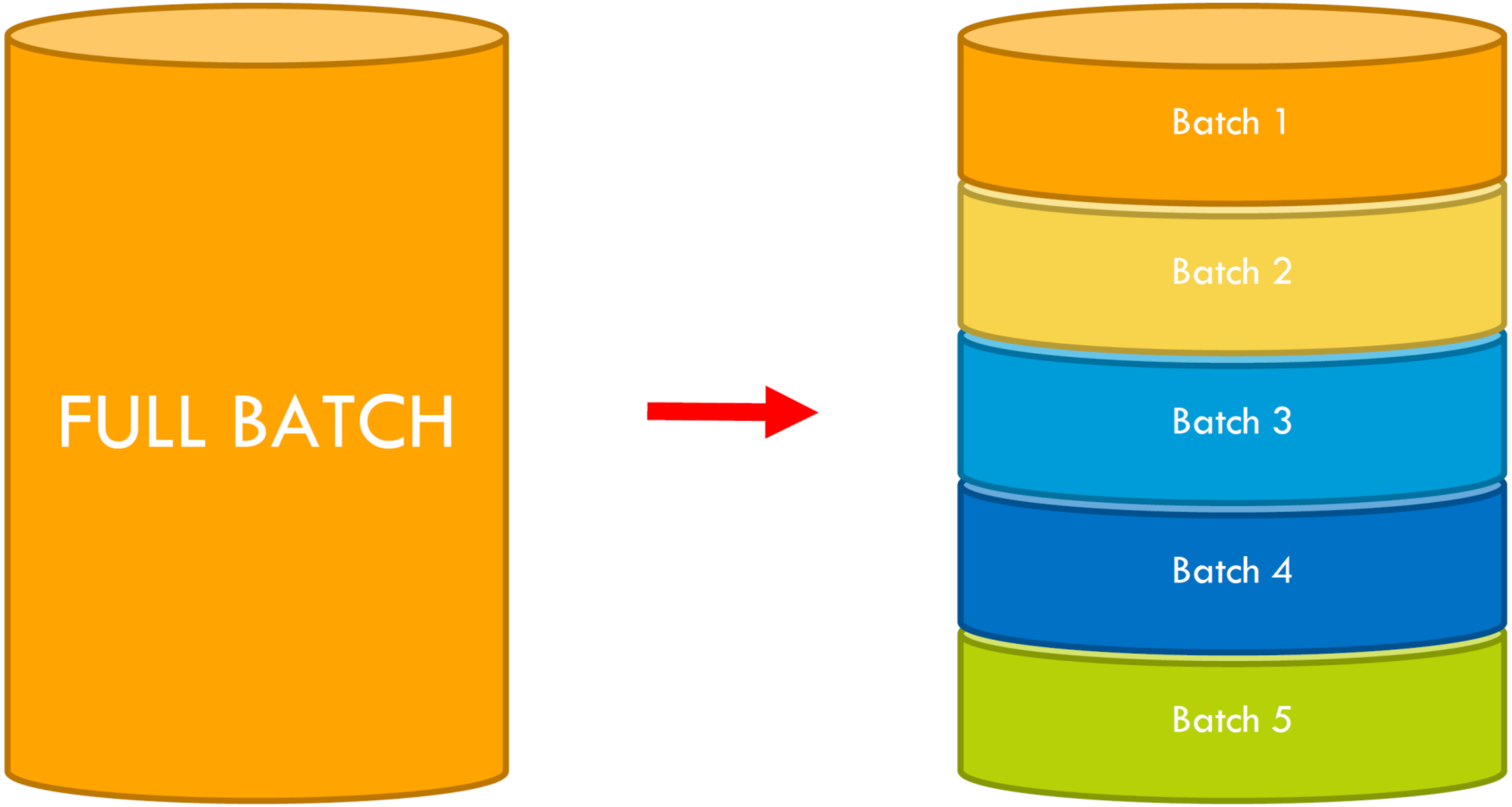
- Steps are “less informed”
- Take more steps
+ Each step is faster
+ A form of regularization

Mini-batch gradient descent

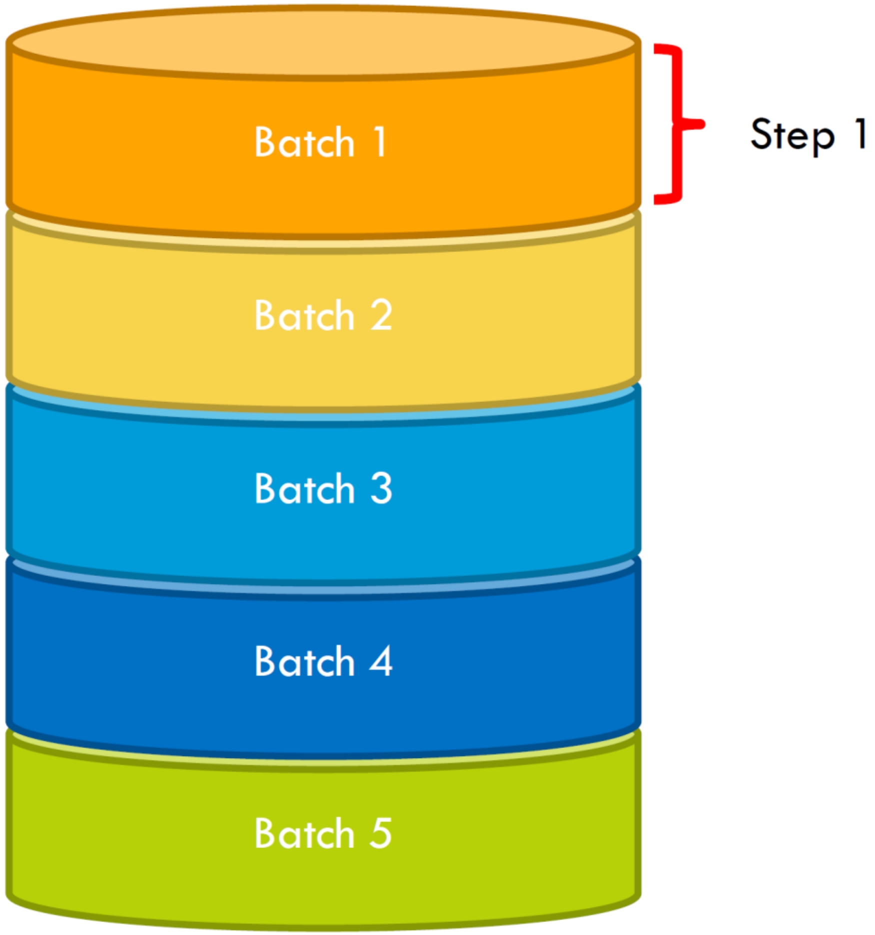
$$\boldsymbol{W}^{(new)} \leftarrow \boldsymbol{W}^{(old)} - \eta \sum_{m} \nabla E_m(\boldsymbol{W})$$

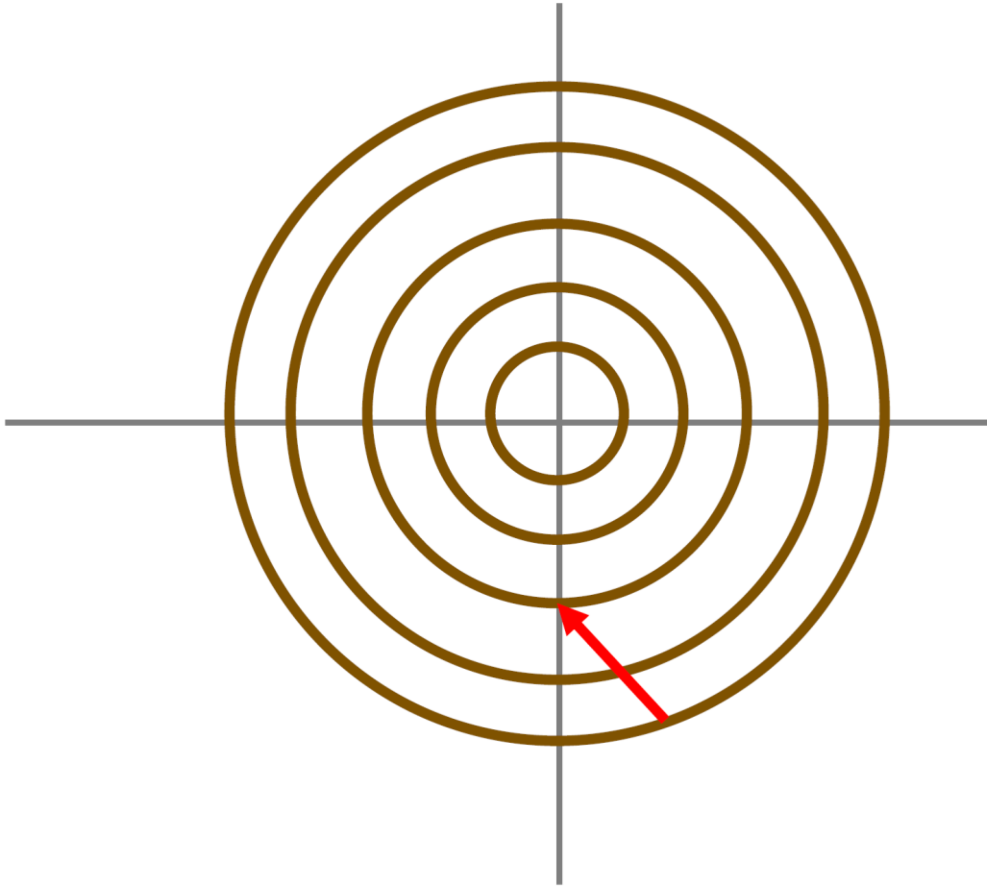
Use a small subset of data at each step

Training in action

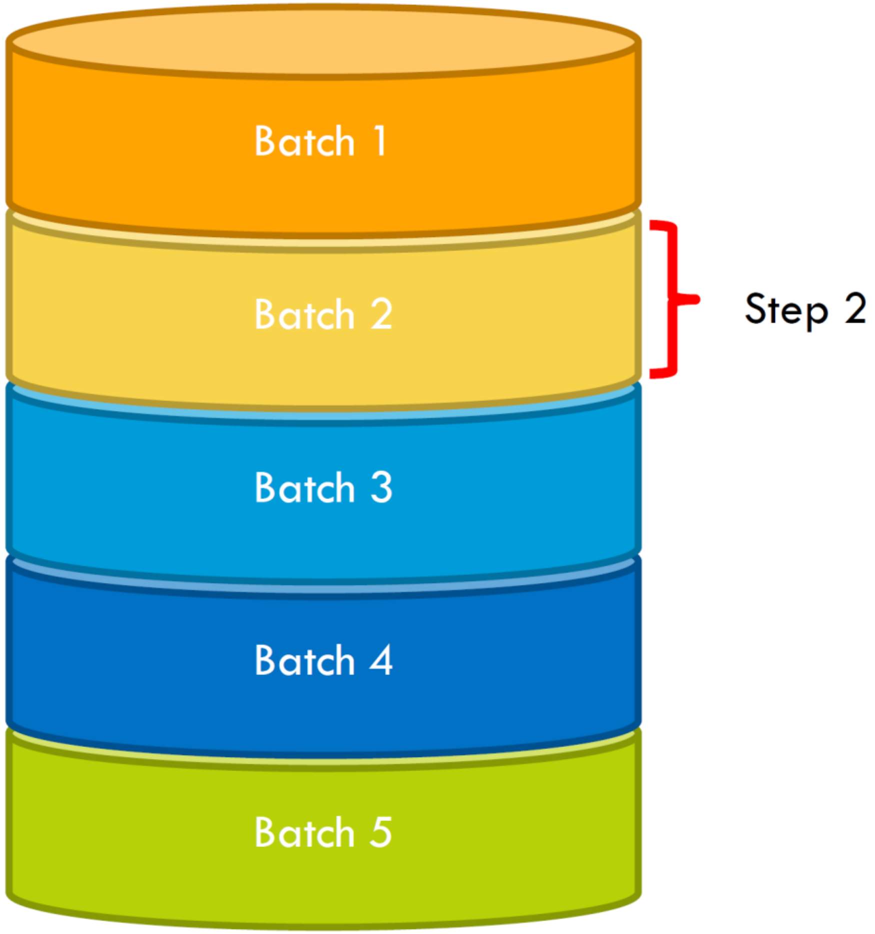


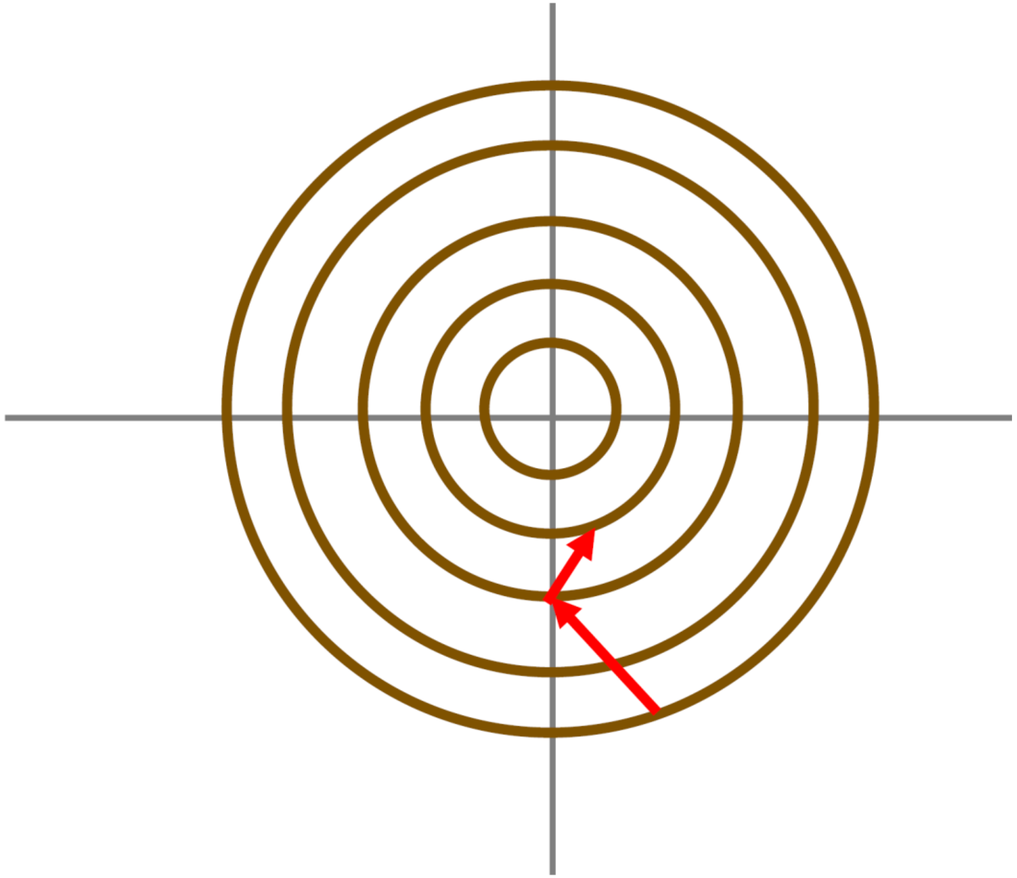
Training in action



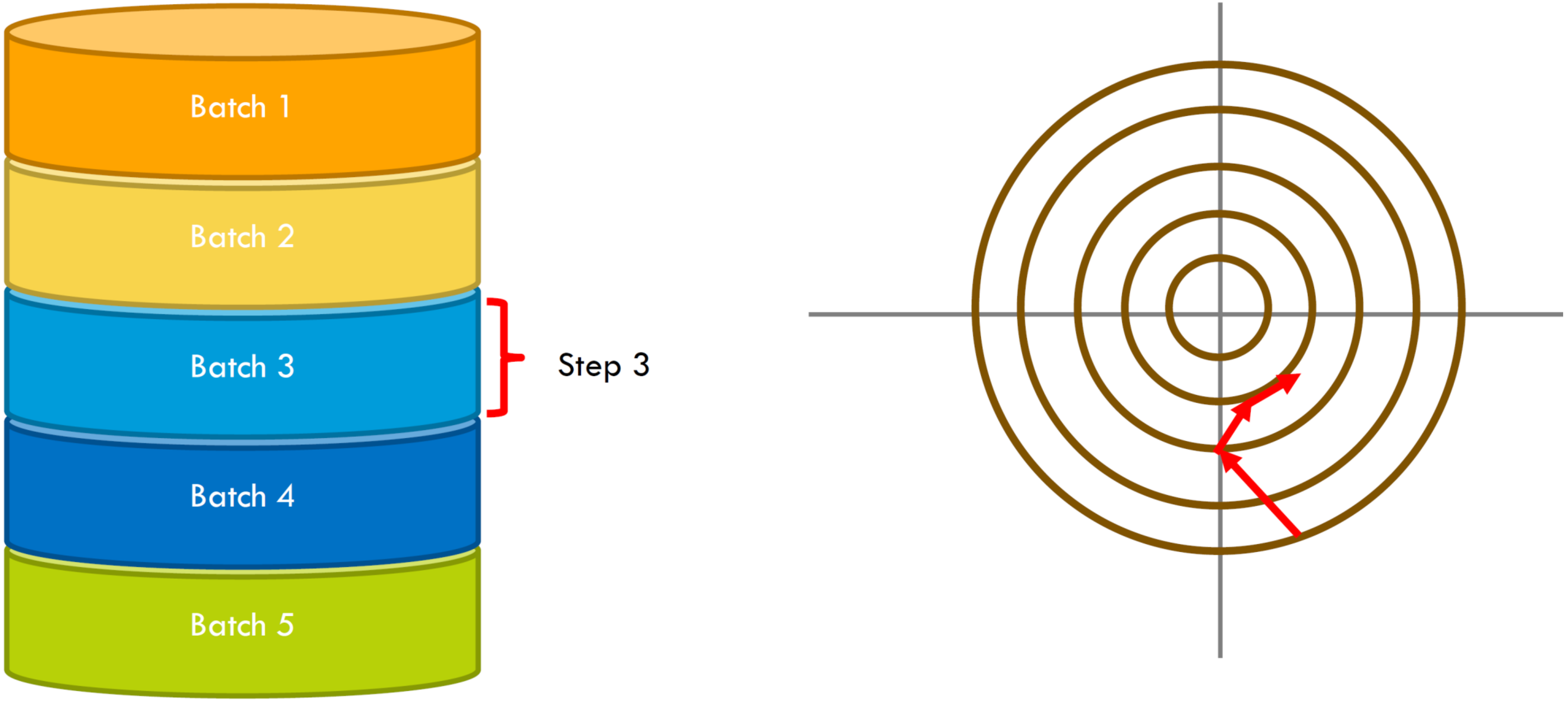


Training in action

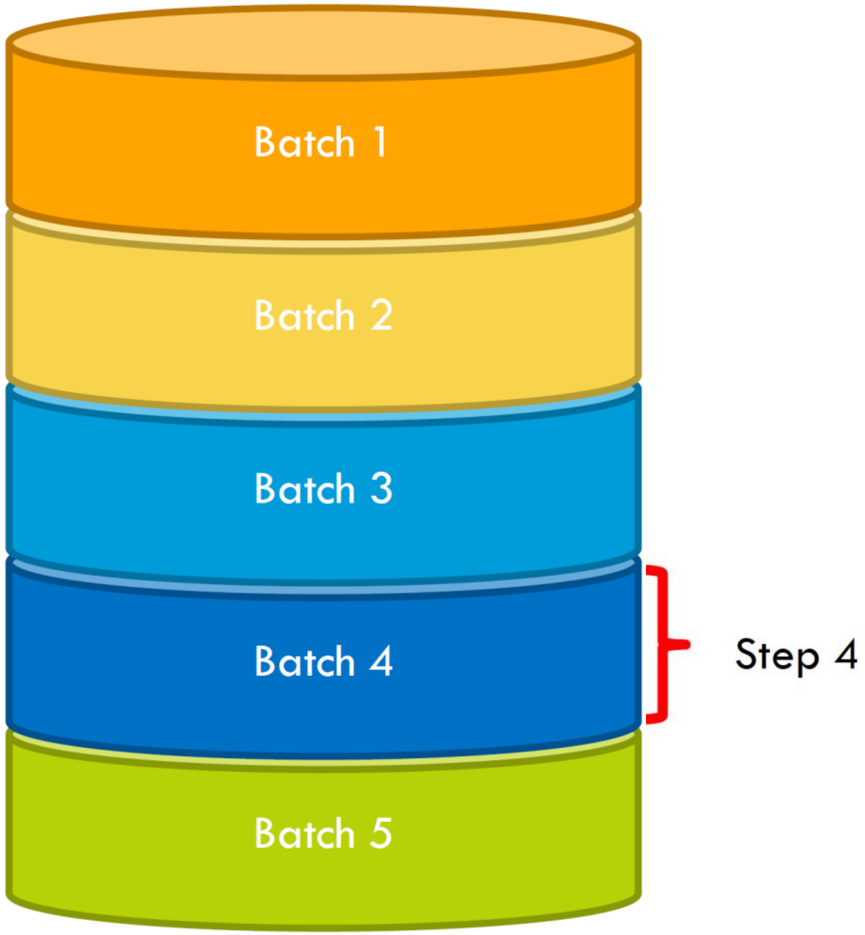


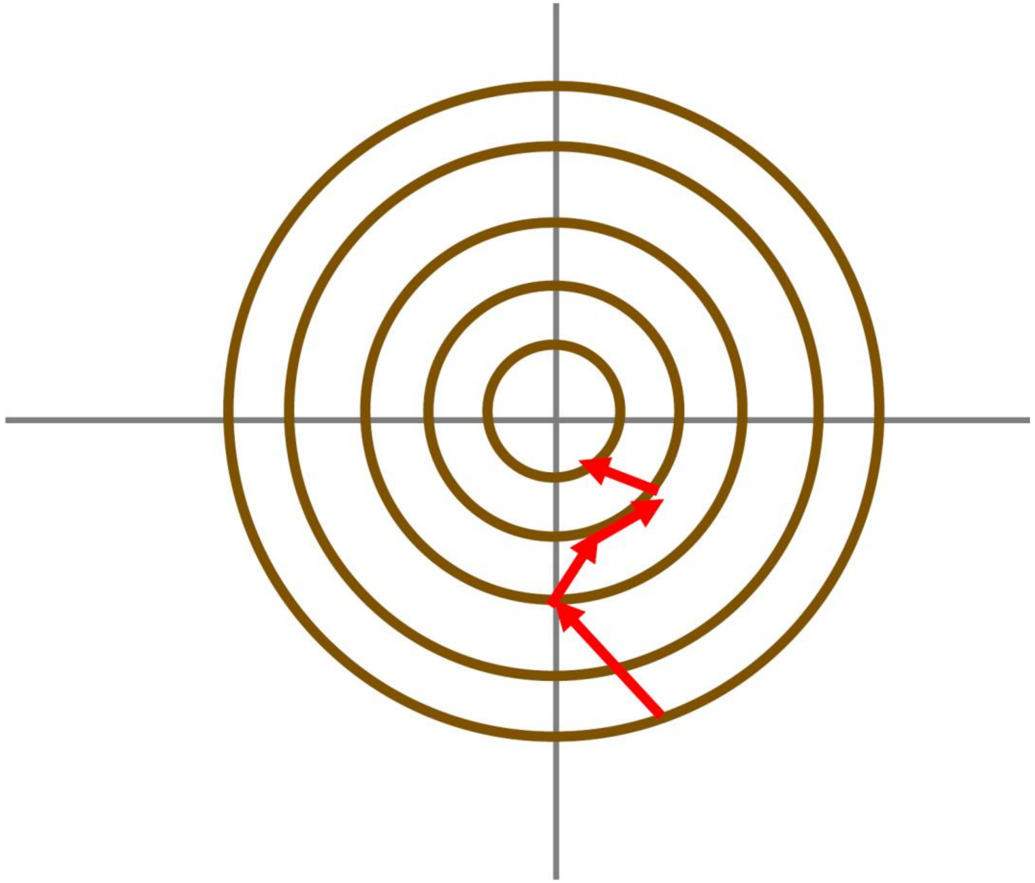


Training in action

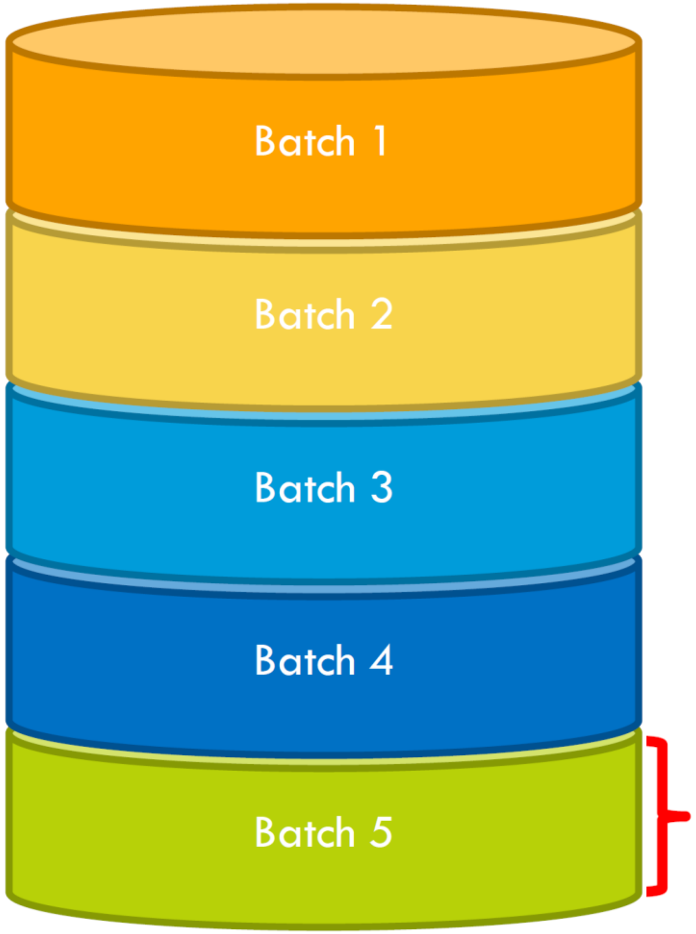


Training in action

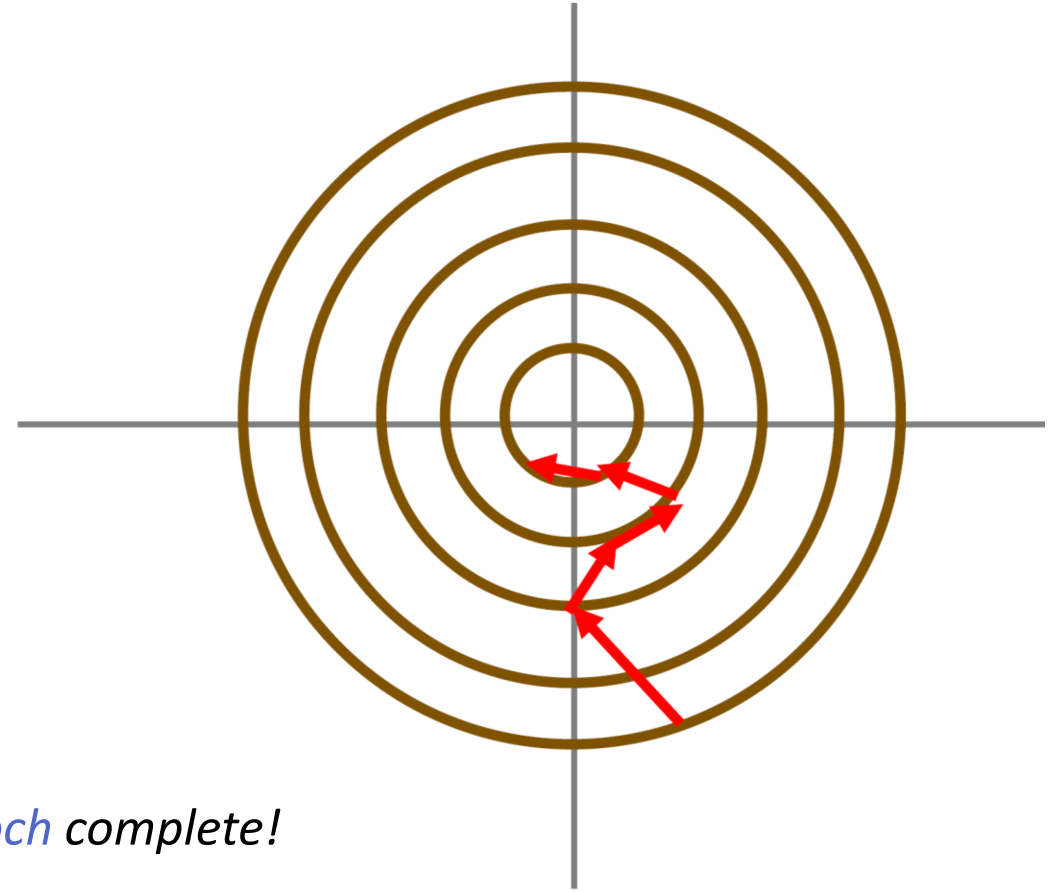




Training in action

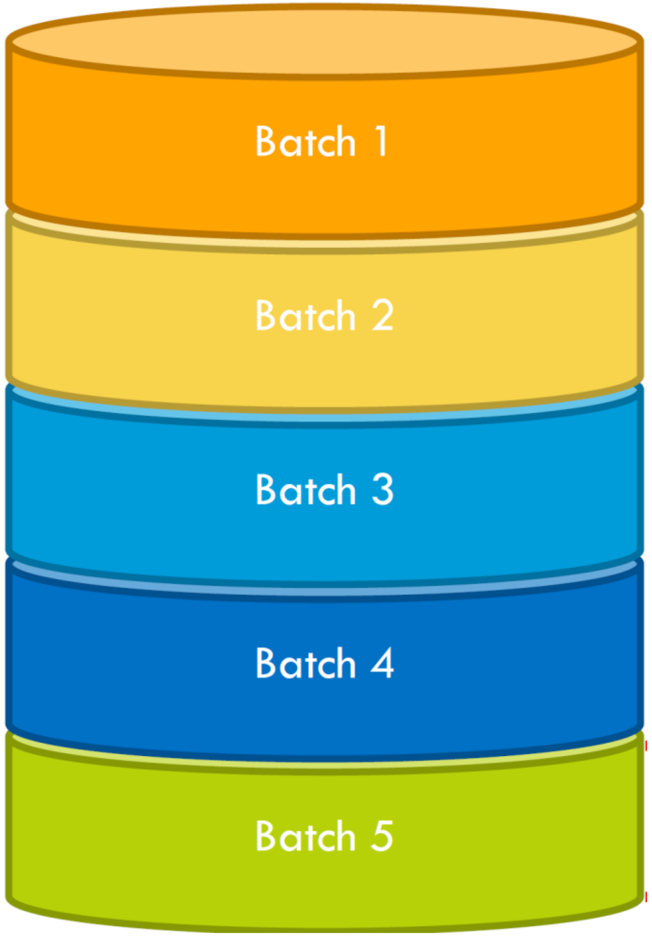


Step 5



1 *epoch* complete!

Training in action

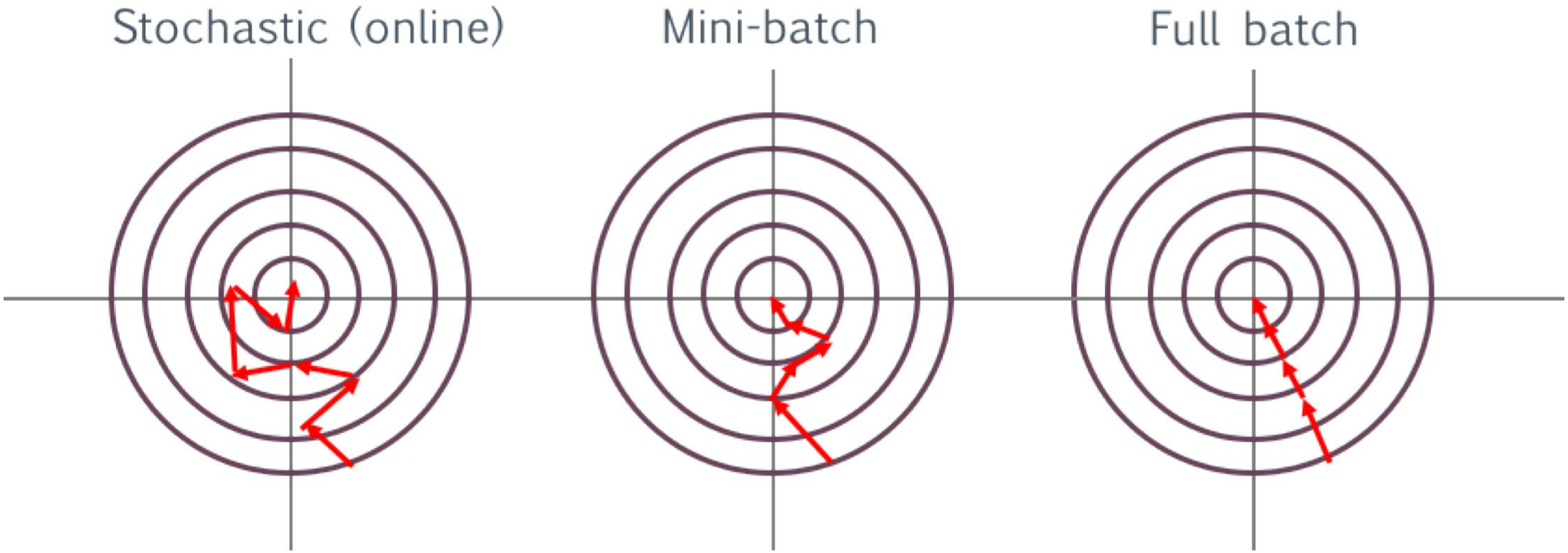


Epoch: a single cycle of training using all the training data

- Full-batch gradient descent: 1 epoch = 1 step using the entire dataset
- Mini-batch gradient descent: 1 epoch = N/(batch-size) steps, each step using a subset of size “batch-size”
- Stochastic gradient descent: 1 epoch = N steps, each step using a single training example

Next epoch: reshuffle the data and repeat

Optimization trade-off



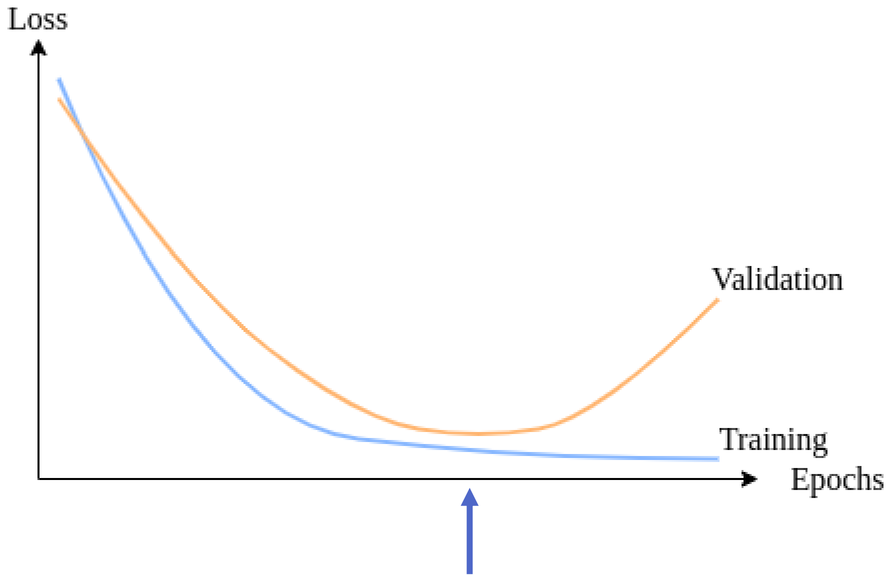
May also help escape out of local minima

How to choose the number of epochs?

Too small -> underfitting
Too large -> unnecessary updates after convergence, even overfitting

Early stopping

1. Split the data into training / validation / test data
2. Set the number of possible epochs as a large number and start training
3. After each epoch, evaluate the model on the validation set
4. If generalization error starts to increase, stop training



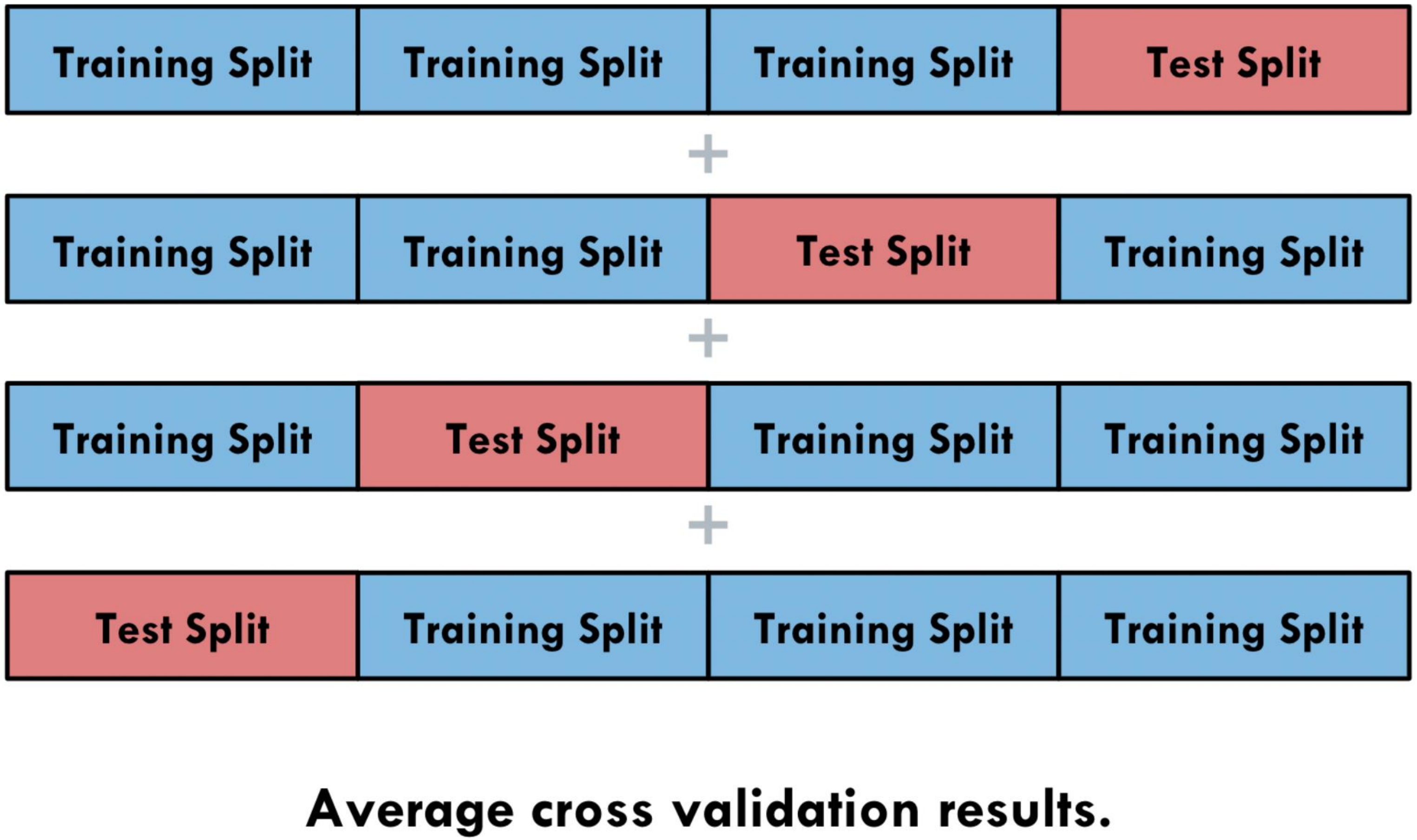
Hyperparameter tuning / model selection

Hyperparameters: define the model architecture & training procedure

- *Not model parameters*, which are trained from data to optimize an error function
- E.g. number of hidden layers, width of hidden layers, learning rate, batch size, shape & stride of convolution filters, …
- features (e.g. polynomial degree), regularization parameter, slack penalty for soft-margin SVM, …

1. Determine the hyperparameters and candidate values for each
2. For every possible combination of values (*grid search*) or randomly selected values (*random search*), train a model using those hyperparameters
3. Select the best one based on average cross-validation scores
4. Evaluate the model on test set using the selection from (3)

K-fold cross validation



Train and evaluate