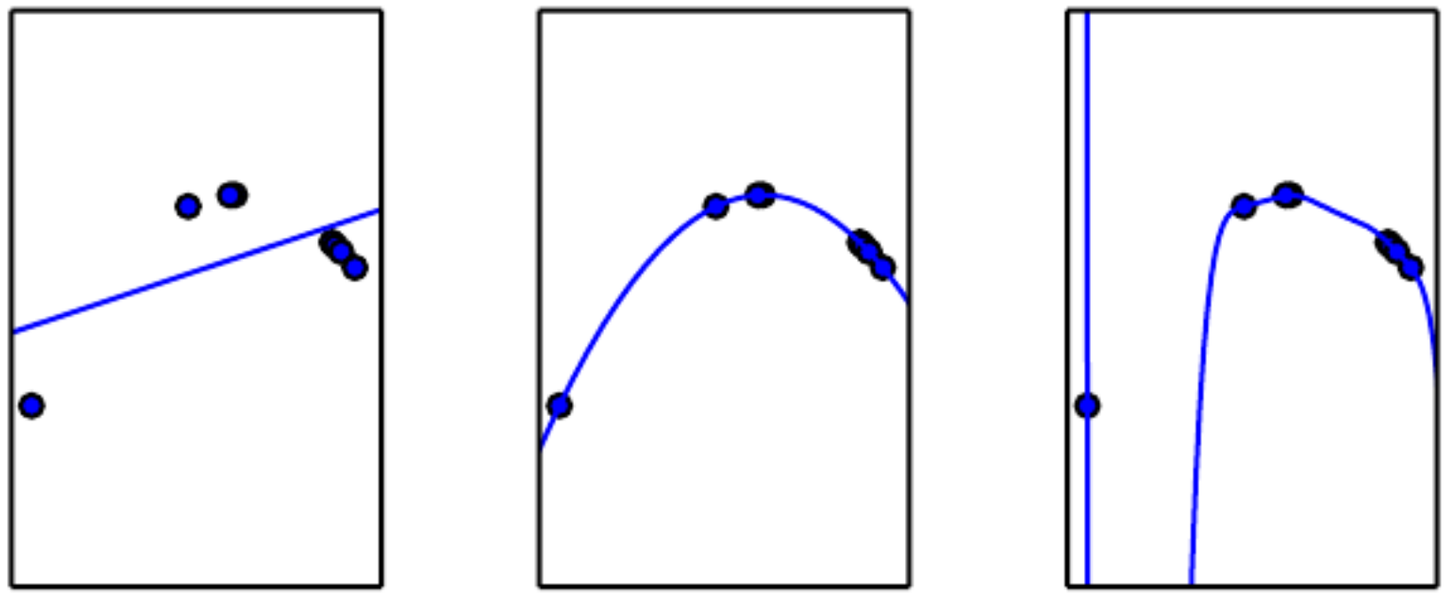
Regularisation

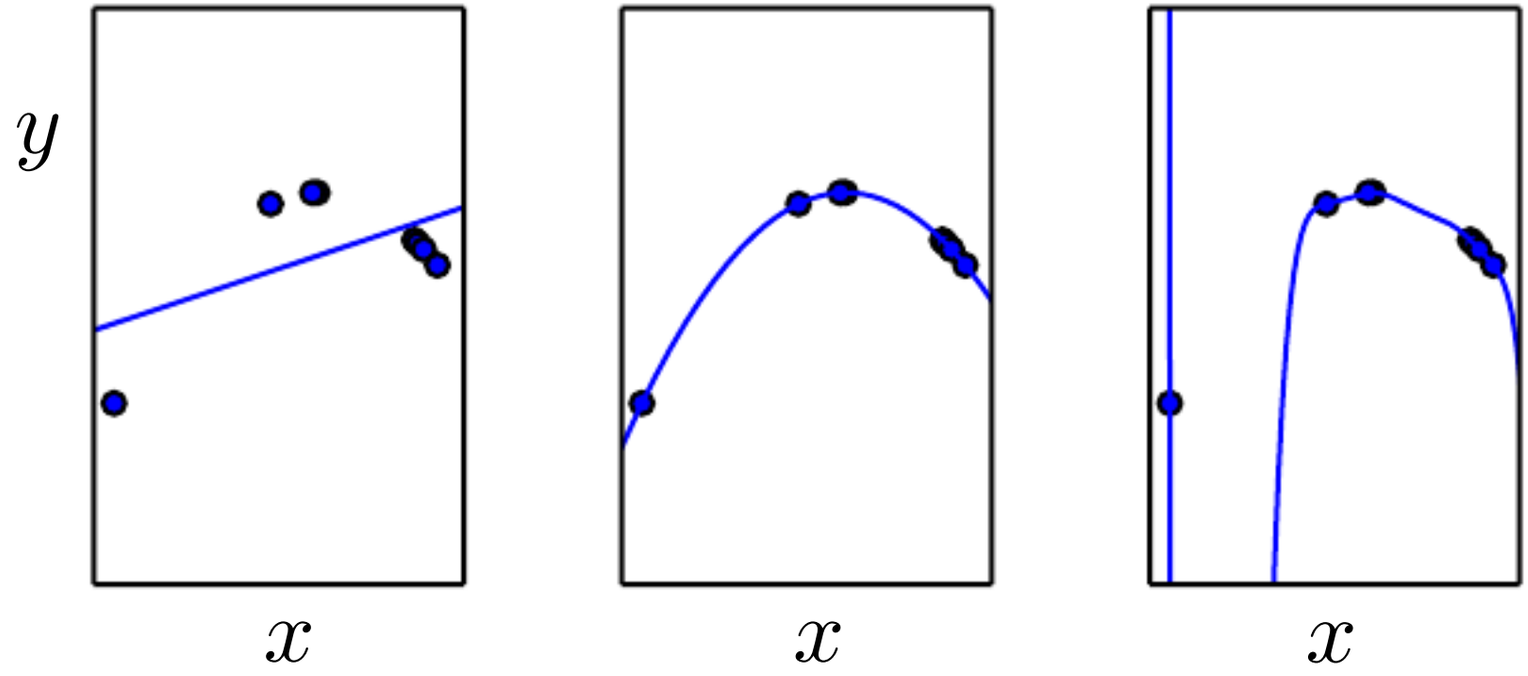
D.M.J. Tax



Regularisation

- Vary the degree p of the regression model:

$$f(x) = w_p x^p + w_{p-1} x^{p-1} + ... + w_1 x + w_0$$

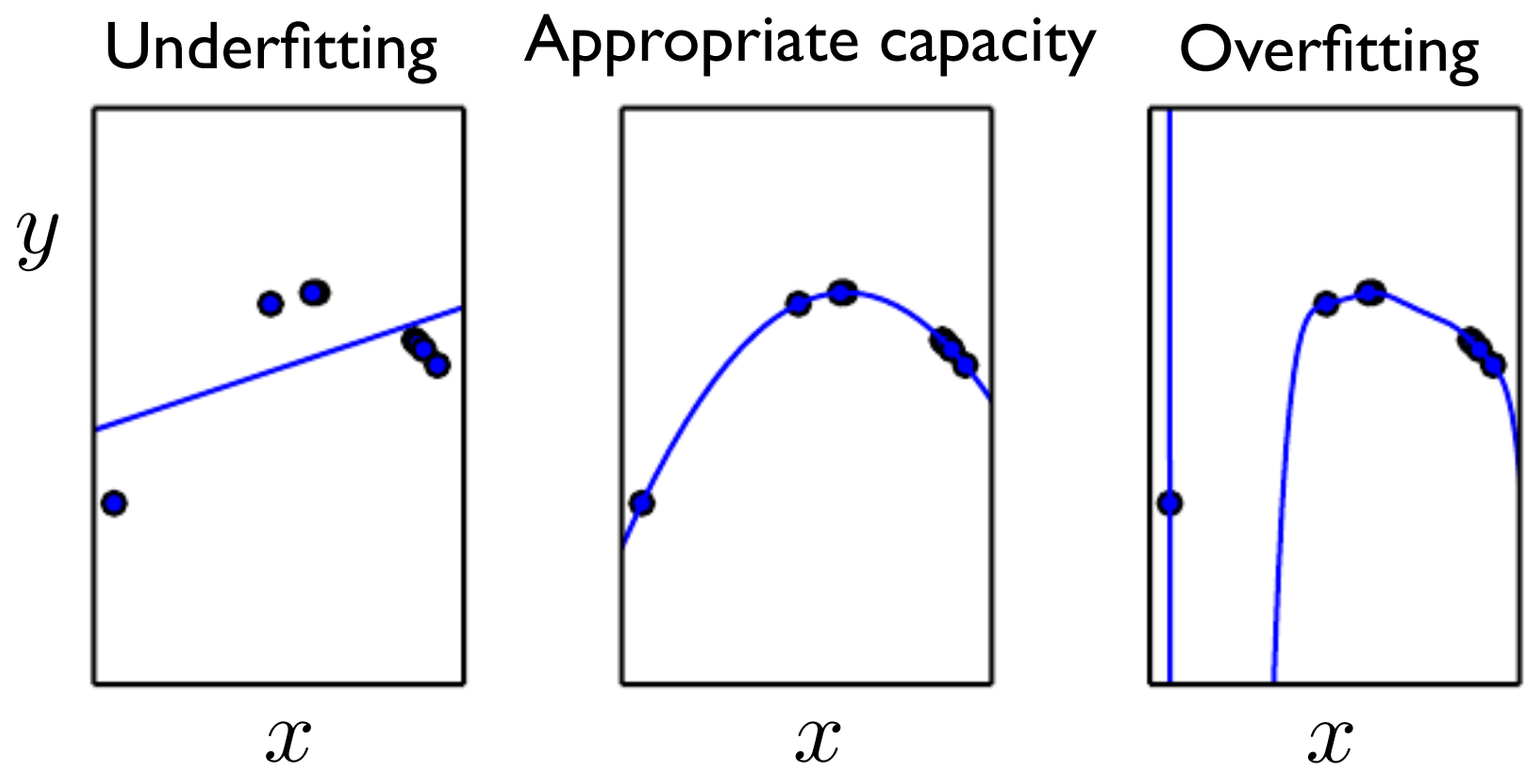


Which one do you prefer?

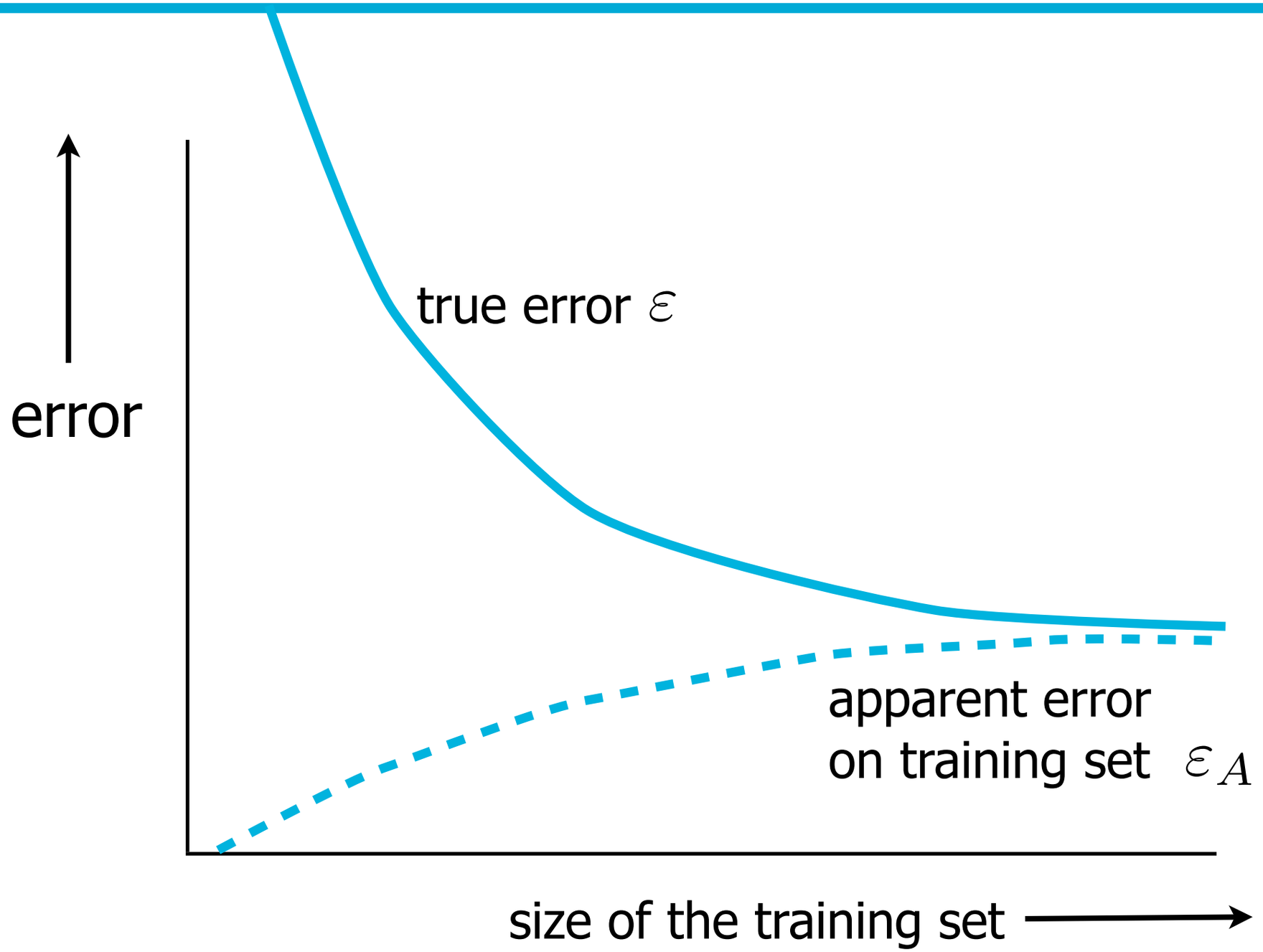
Overfitting

- Vary the degree p of the regression model:

$$f(x) = w_p x^p + w_{p-1} x^{p-1} + \ldots + w_1 x + w_0$$

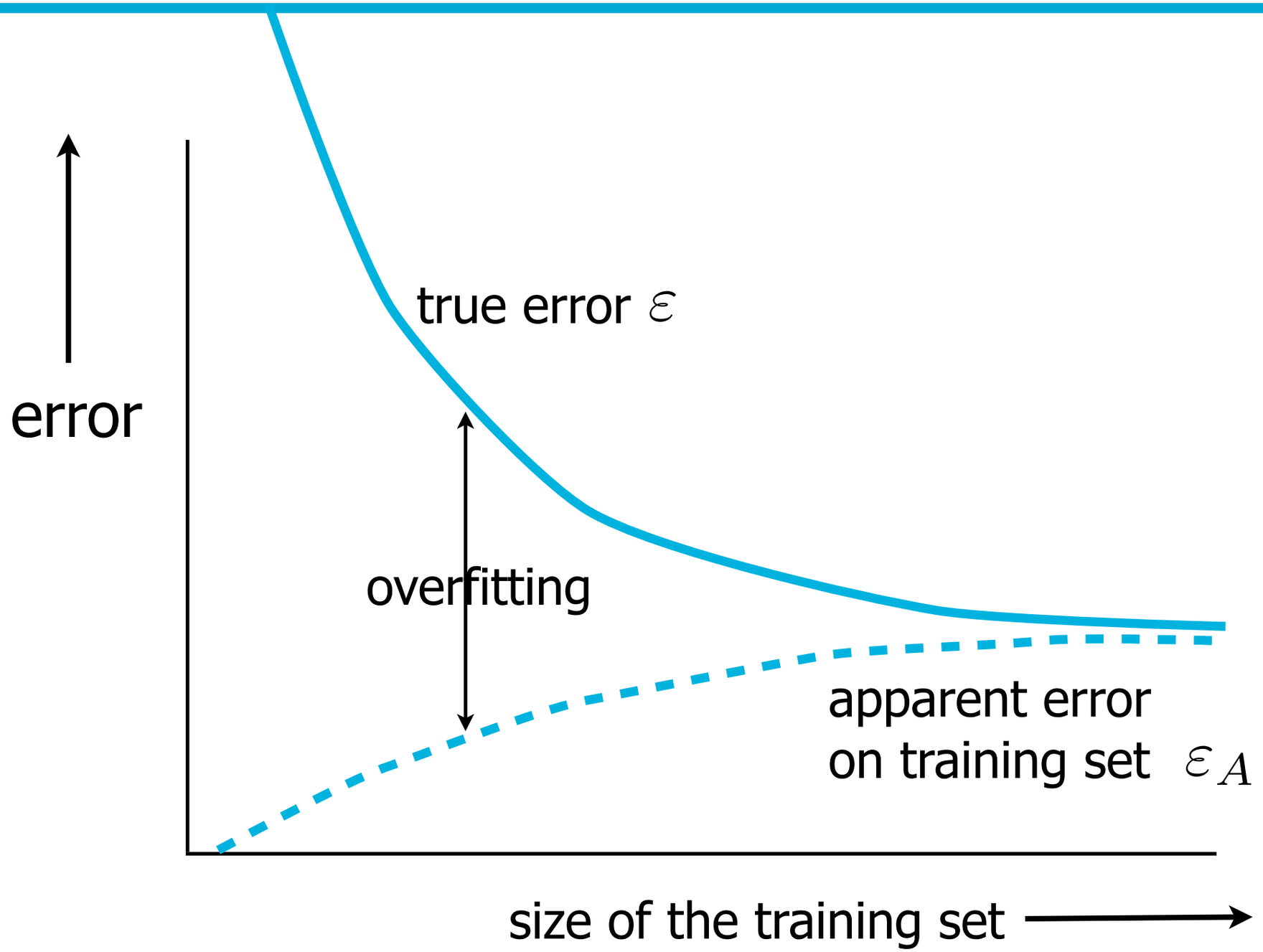


Learning curve



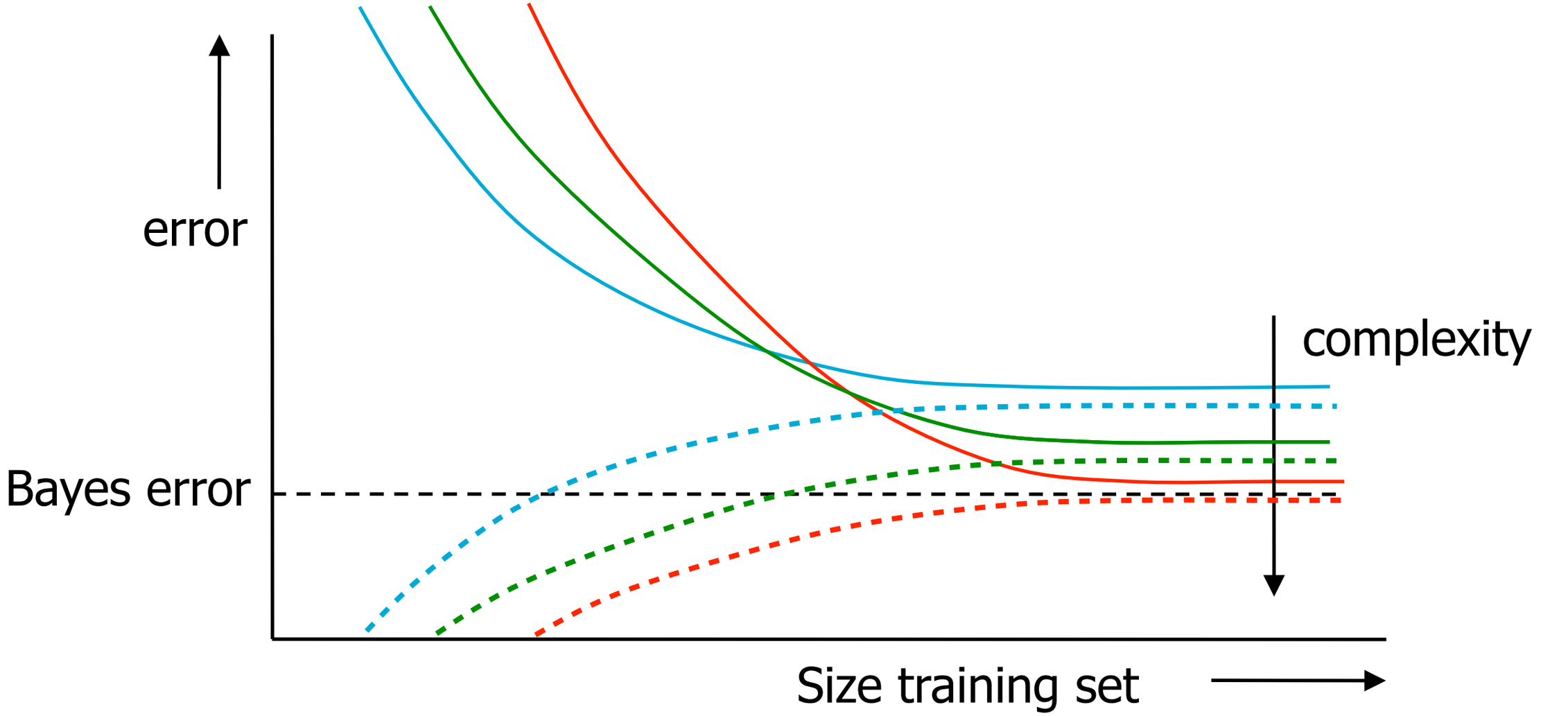
- Note: not the same as training process of a neural net!

Learning curve



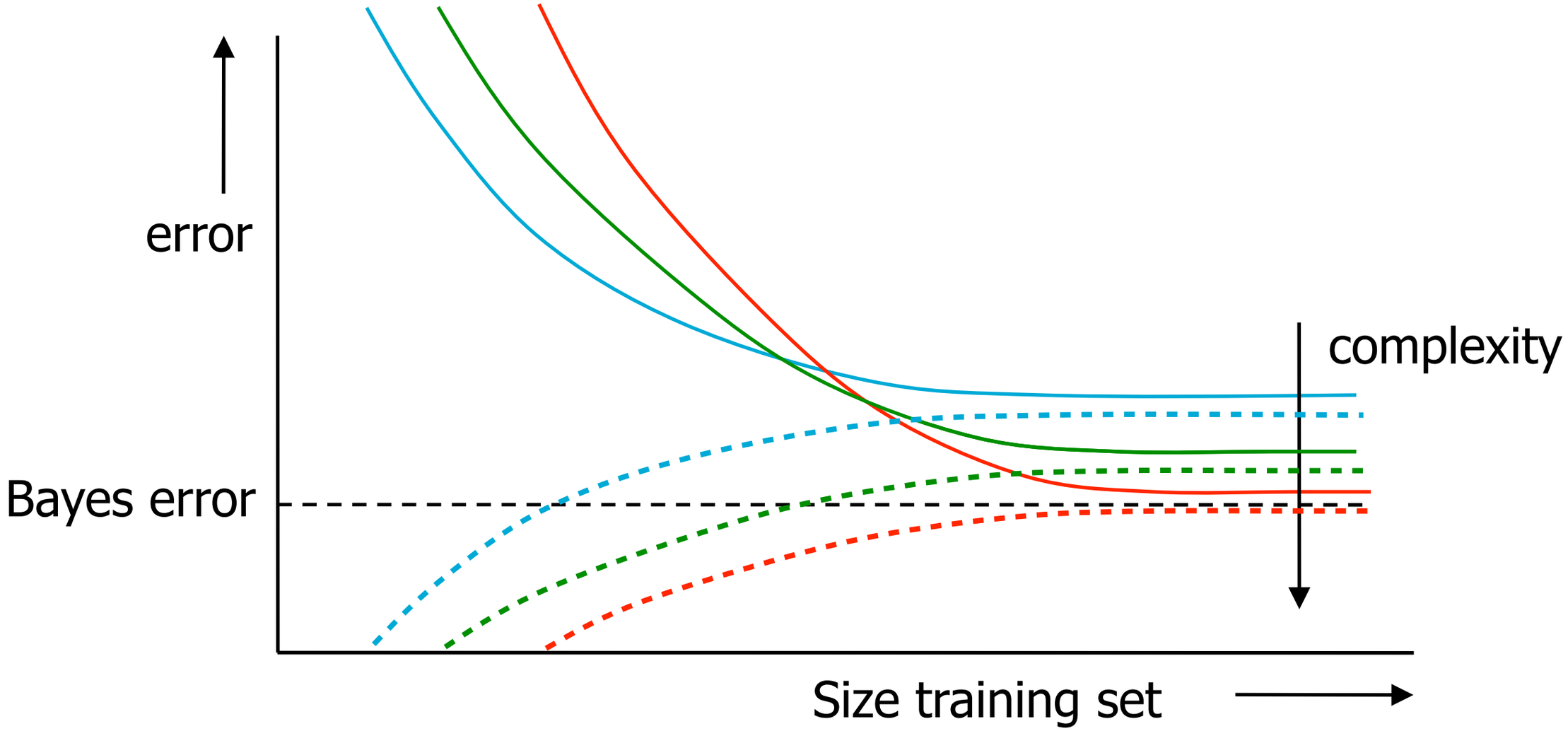
- Small training set: overfitting!

Different model complexity



- Where is underfitting?

Different model complexity



- How does the curve for a very, very, **very** complex model look like?

Beating overfitting

- How to do it?

Beating overfitting

- Use more data! Data augmentation
- Reduce the number of features
- Reduce complexity/flexibility of the model

Beating overfitting

- Use more data! Data augmentation
- Reduce the number of features
- Reduce complexity/flexibility of the model

  - Parameter norm regularisation
  - Early stopping
  - Noise robustness
  - Weight sharing
  - (Dropout)
  - …

Regularisation: discourage overly complex models

Contents

- Intro overfitting, regularisation
- Dataset augmentation
- Generalisation bounds
- Regularisation
- Training a neural network
- Parameter norm penalties, weight decay
- Noise robustness
- Early stopping
- Parameter tying
- Dropout

Dataset augmentatio--

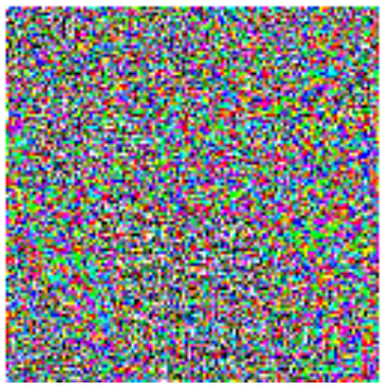
distorted patterns with randomly picked distortion parameters. The distortions were combinations of the following planar affine transformations: horizontal and vertical translations, scaling, squeezing (simultaneous horizontal compression and vertical elongation, or the reverse), and horizontal shearing. Figure 7 shows examples of dis-

(no rotation? why?)

- 'Gradient-based Learning Applied to Document Recognition', Y.LeCun, L.Bottou, Y.Bengio, P.Haffner, 1998
- Only possible when you know the invariances (images)
- Suitable distortions for songs? For video's?

Adverserial data

$+ .007 \times$



$=$

$\boldsymbol{x}$	$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$	$\boldsymbol{x} + \epsilon \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
$y =$"panda"	"nematode"	"gibbon"
w/ 57.7% confidence	w/ 8.2% confidence	w/ 99.3% confidence

- Generate samples that are correctly classified with high confidence
- (Generate samples that are INcorrectly classified with high confidence as counter examples?)

Feature reduction

- See course Pattern Recognition: reduction and extraction
- Currently a bit out of fashion
- Sometimes ‘sneakily’ done in image classification by ‘rescaling to 28x28’

Generalisation bounds

(Only background info)

- Neural network h, input dim.=n, m training objects, with confidence δ holds:

$$\mathrm{er}_P(h) < \widehat{\mathrm{er}}_z^{\gamma}(h) + \sqrt{\frac{c}{m}\left(\frac{B^2(AL)^{\ell(\ell+1)}}{\gamma^{2\ell}}\log n \log^2 m + \log(1/\delta)\right)}$$

- A: sum of (absolute values of) weights
- B: maximum norm of data vectors
- L: some smoothness of the transfer function
- ℓ: number of layers
- γ: margin

Generalisation bounds

(Only background info)

- Neural network h, input dim.=n, m train objects, with confidence δ holds:

$$\mathrm{er}_P(h) < \widehat{\mathrm{er}}_z^{\gamma}(h) + \sqrt{\frac{c}{m}\left(\frac{B^2(AL)^{\ell(\ell+1)}}{\gamma^{2\ell}}\log n \log^2 m + \log(1/\delta)\right)}$$

- Or, for a two-layer network:

!!

$$\text{true error} < \text{train error} + f((\text{norm of weights})^3)$$

Regularisation

- Limit the complexity/flexibility of a complex model

- Optimise a regularised loss

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta}),$$

standard loss

regularisation parameter

regulariser

- 2048

Parameter norm regularisation

- Very standard is L_2 regularisation:

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2} \boldsymbol{w}^\top \boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}),$$

- (And the gradient is…?)
- Results in an update:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon \left(\alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) \right).$$

- Compare: ridge regression, Tikhonov regularisation, weight decay (Hinton 1987)

Parameter norm regularisation

- Alternative: L1 regularisation

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha ||\boldsymbol{w}||_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}),$$

(gradient?)

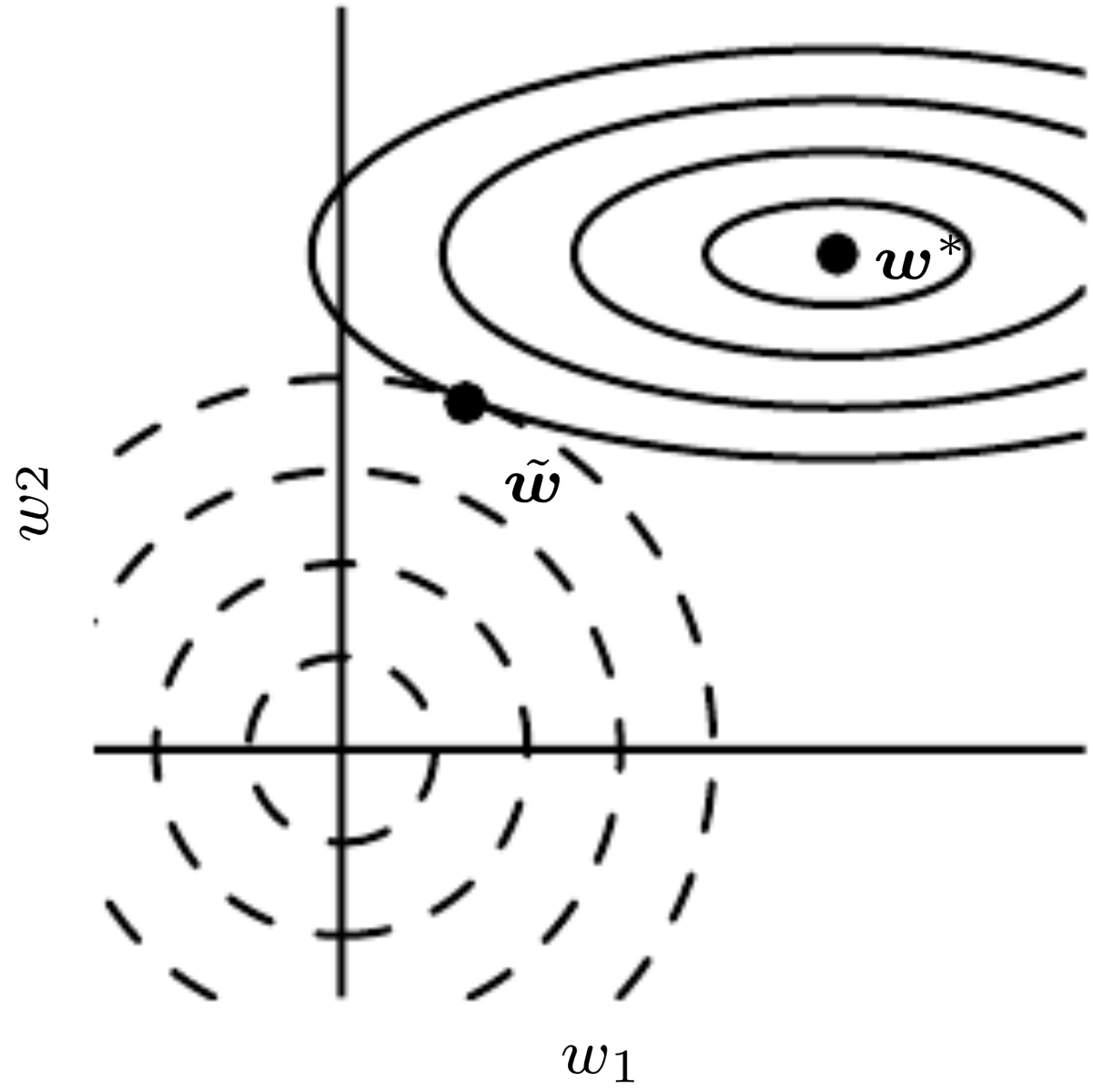
- Reduce the weights to **exactly** 0.
- Performs features selection/neuron removal during training of the network
- Compare: Optimal Brain Damage (LeCun, 1990) (remove weights with small second-order influence on the error)

TU Delft

Visually

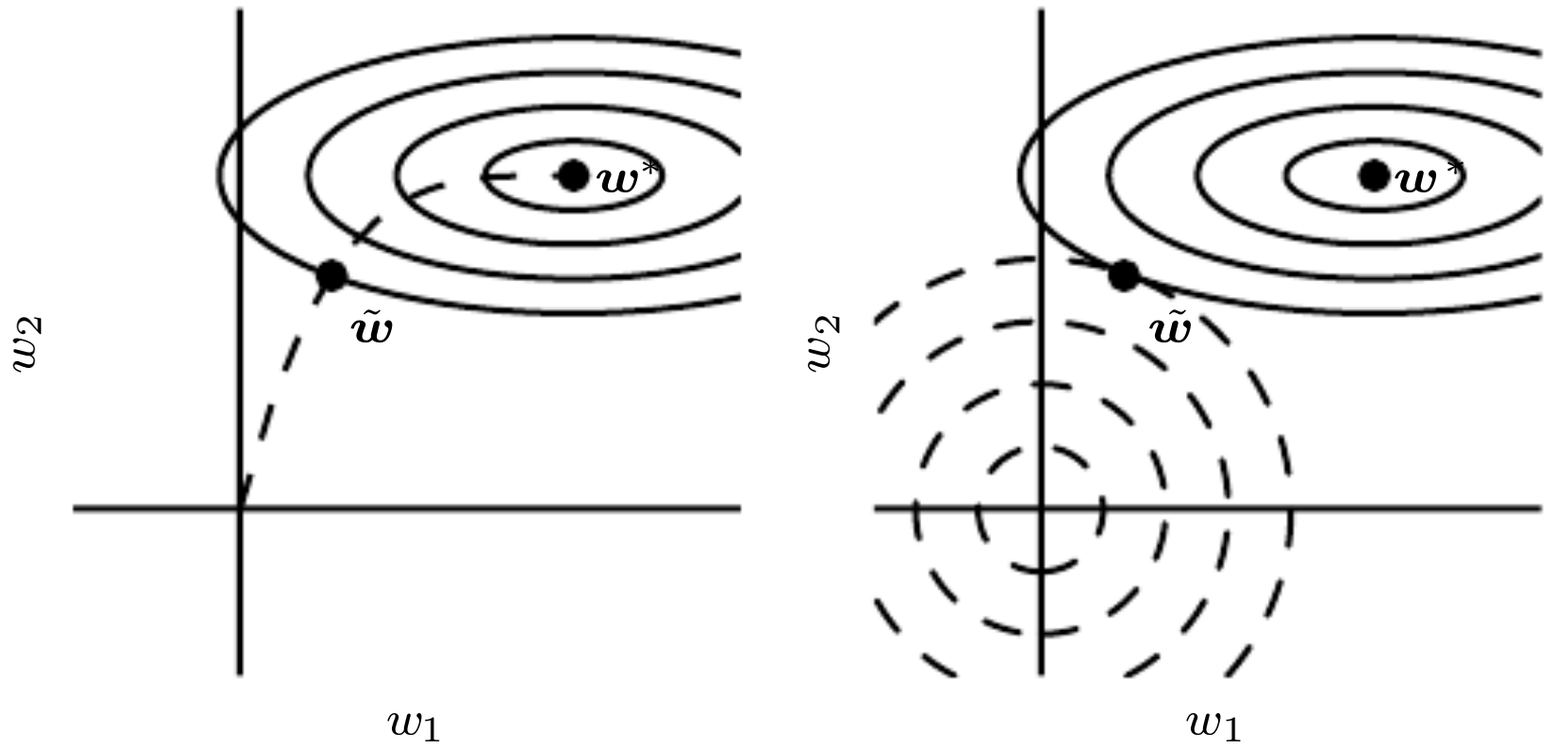
- Weight decay discourages large weights

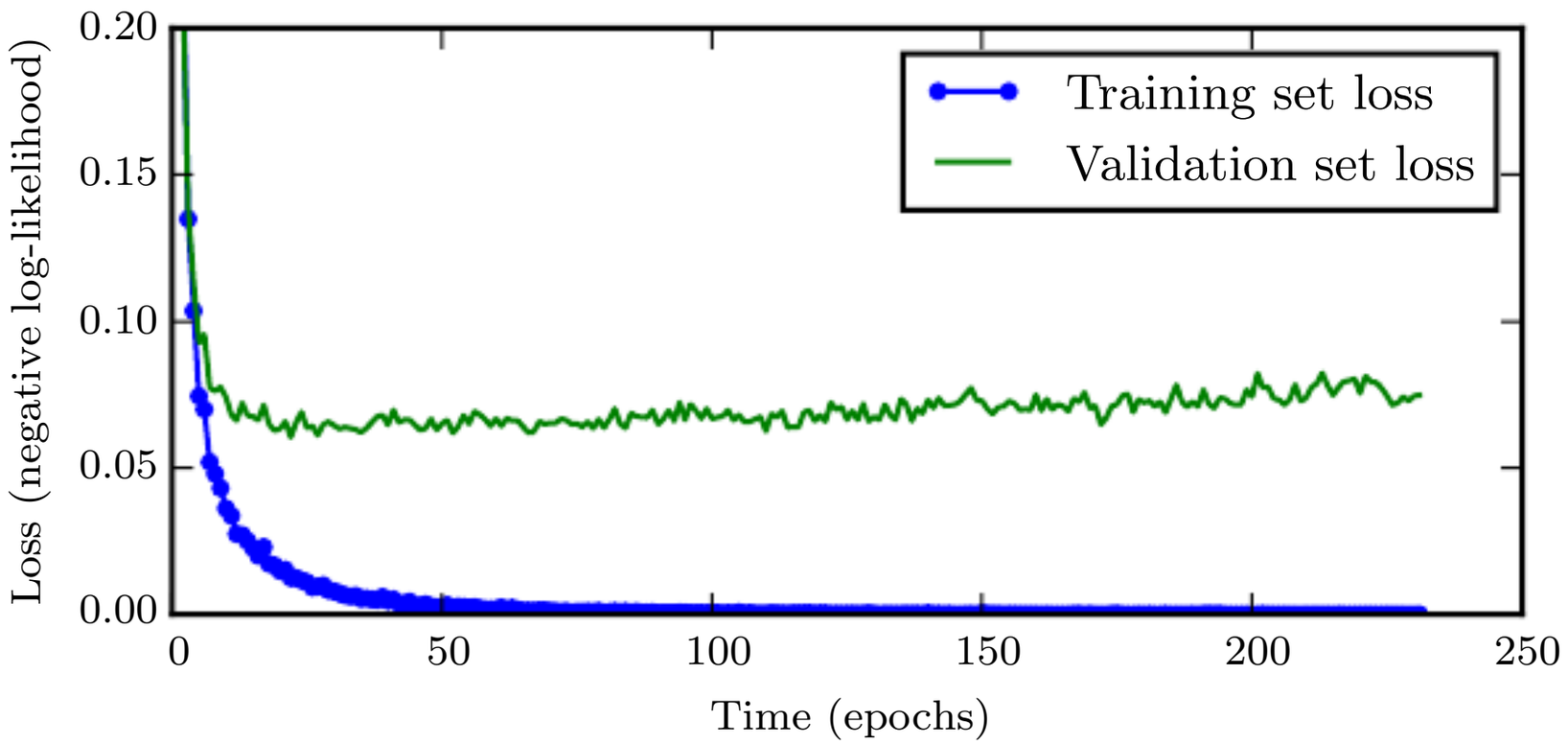
- Instead of finding solution w^* we find $\tilde{w}$



Early stopping

- Stop the gradient updates prematurely
- Instead of the global min $\boldsymbol{w}^*$ we get to the regularised $\tilde{\boldsymbol{w}}$
- If you look at performance on validation set: no need for extra parameters!





Early stopping, initialisation

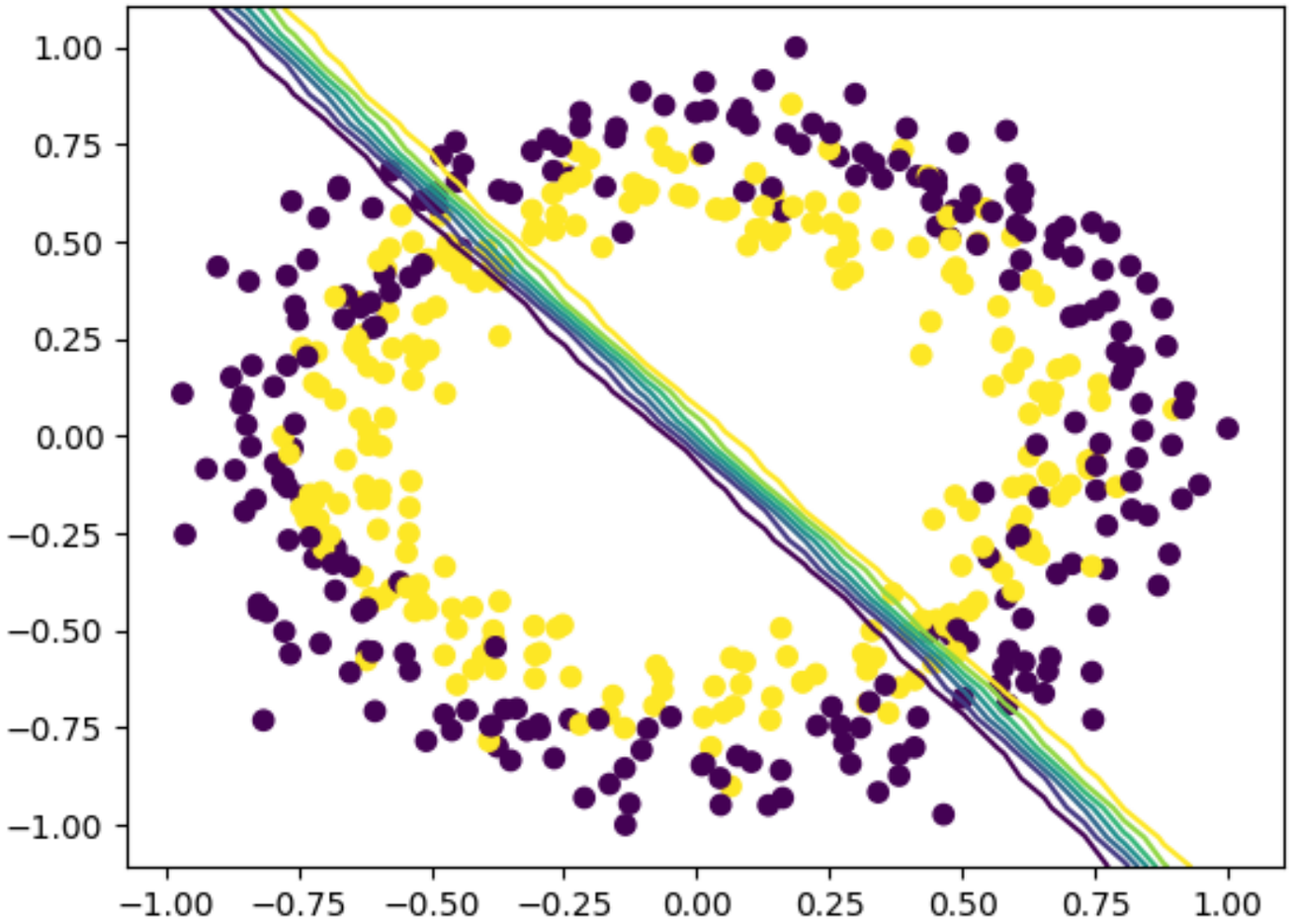
- Early stopping useful when network is correctly initialised
- Starting with large weights is wrong

```
# generate 2d classification dataset
X, y = make_circles(n_samples=1000, noise=0.1, random_state=1)
scaler = MinMaxScaler(feature_range=(-1, 1))
X = scaler.fit_transform(X)
# split into train and test
n_train = 500
trainX, testX = X[:n_train, :], X[n_train:, :]
trainy, testy = y[:n_train], y[n_train:]
# define model
init = RandomUniform(minval=0, maxval=1)
model = Sequential()
model.add(Dense(5, input_dim=2, activation='tanh', kernel_initializer=init))
model.add(Dense(5, activation='tanh', kernel_initializer=init))
model.add(Dense(5, activation='tanh', kernel_initializer=init))
model.add(Dense(5, activation='tanh', kernel_initializer=init))
model.add(Dense(5, activation='tanh', kernel_initializer=init))
model.add(Dense(1, activation='sigmoid', kernel_initializer=init))
# compile model
opt = SGD(lr=0.01, momentum=0.9)
model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
# fit model
history = model.fit(trainX, trainy, validation_data=(testX, testy), epochs=500, verbose=0)
```

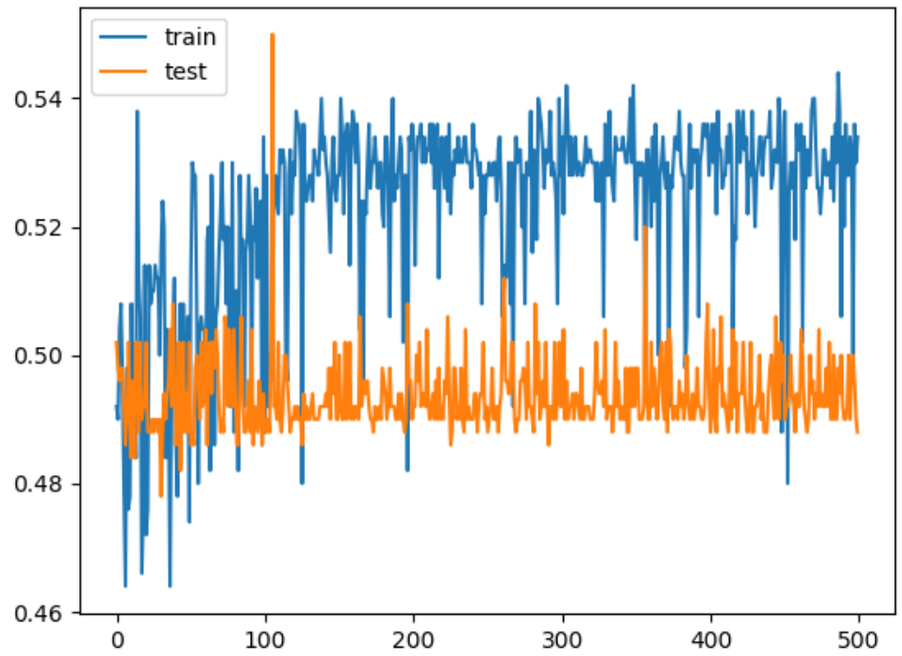
From a blog post:
https://machinelearningmastery.com/how-to-fix-vanishing-gradients-using-the-rectified-linear-activation-function/

Bad initialisation

- Early stopping useful when network is correctly initialised
- Starting with large weights is wrong



performance

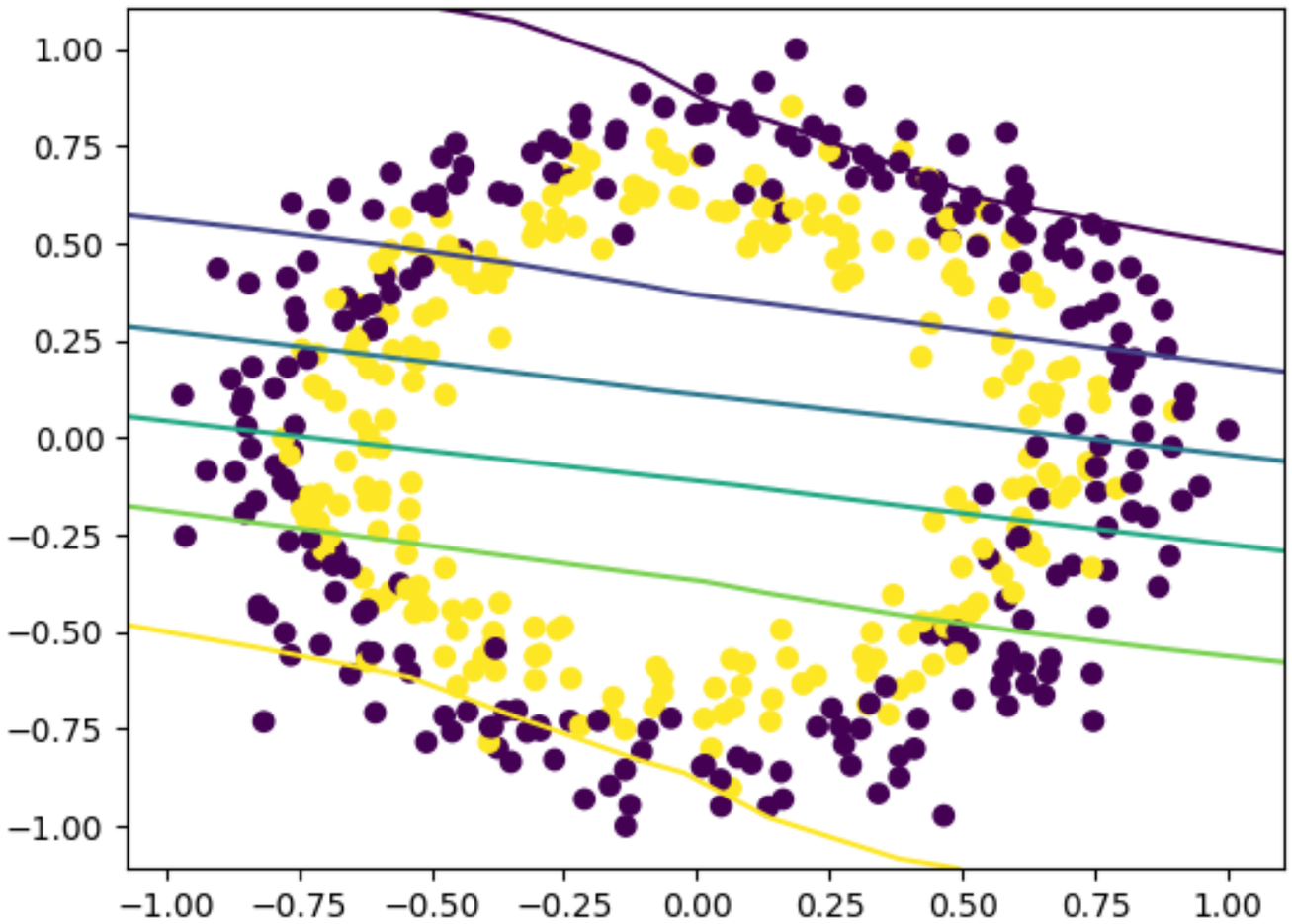


learning epochs

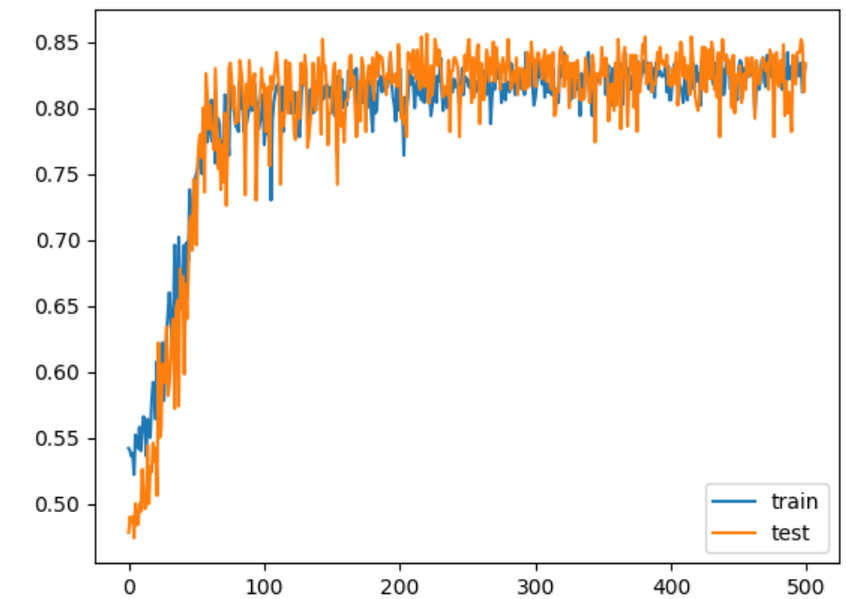
From a blog post:
https://machinelearningmastery.com/how-to-fix-vanishing-gradients-using-the-rectified-linear-activation-function/

Good initialisation

- Early stopping useful when network is correctly initialised
- Starting with small weights is good:



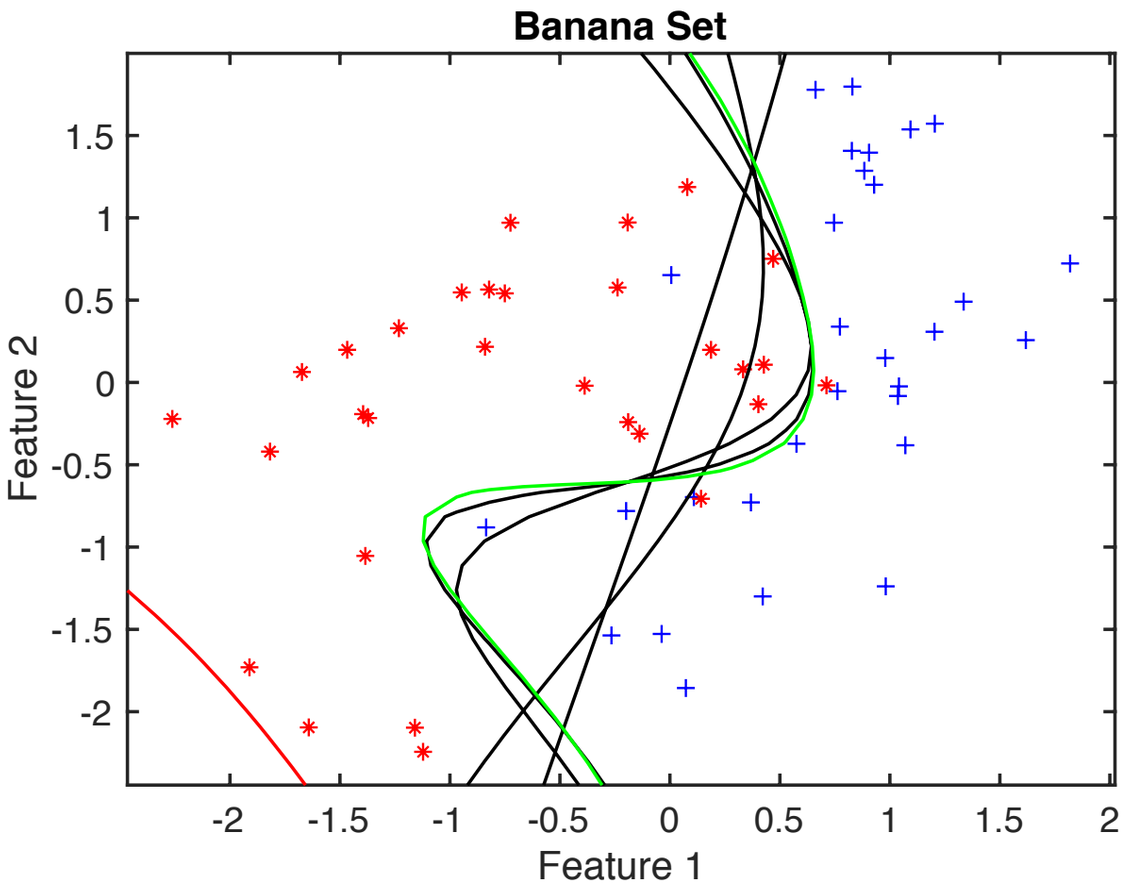
performance



learning epochs

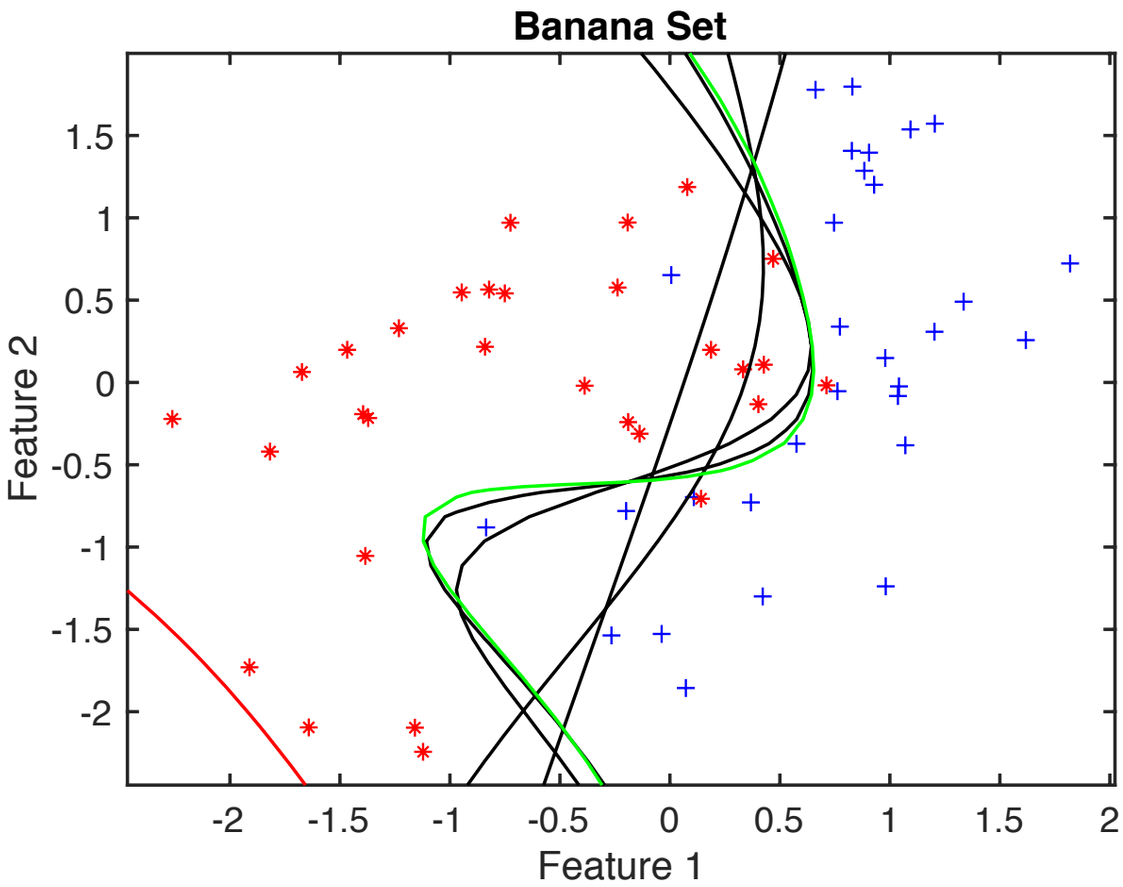
From a blog post:
https://machinelearningmastery.com/how-to-fix-vanishing-gradients-using-the-rectified-linear-activation-function/

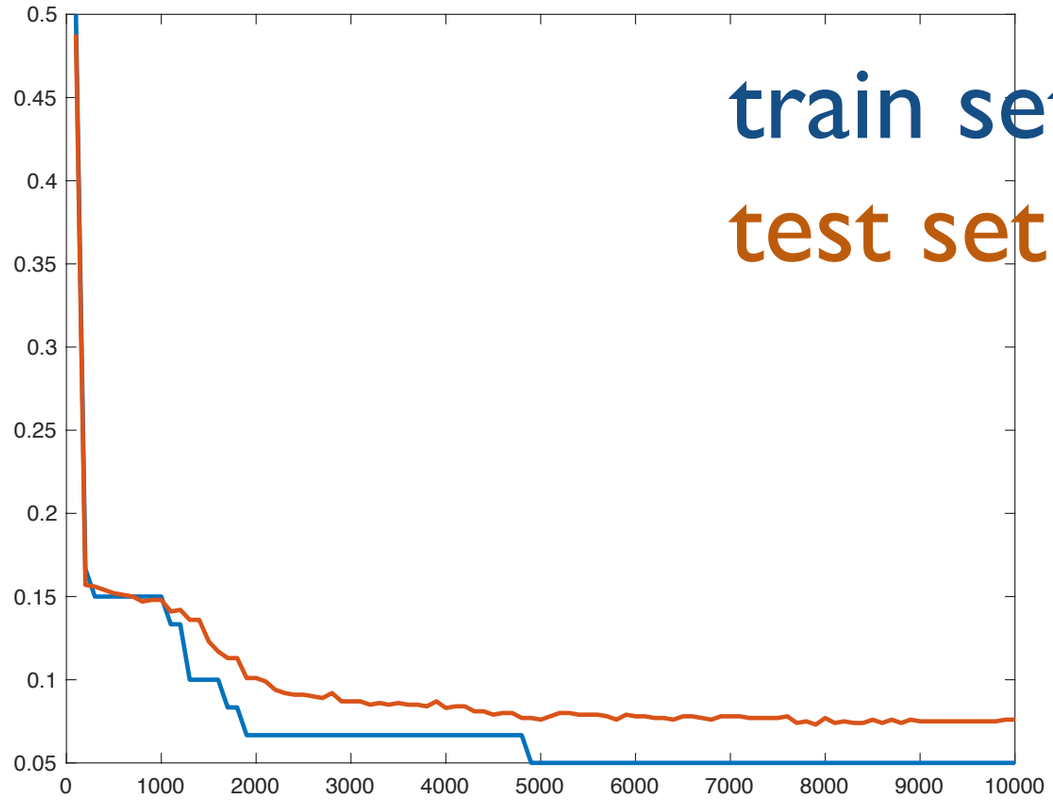
Training a network



- Train 1-layer network on Banana set (30 obj. per class)
- At start: red decision boundary
- At end: green decision boundary

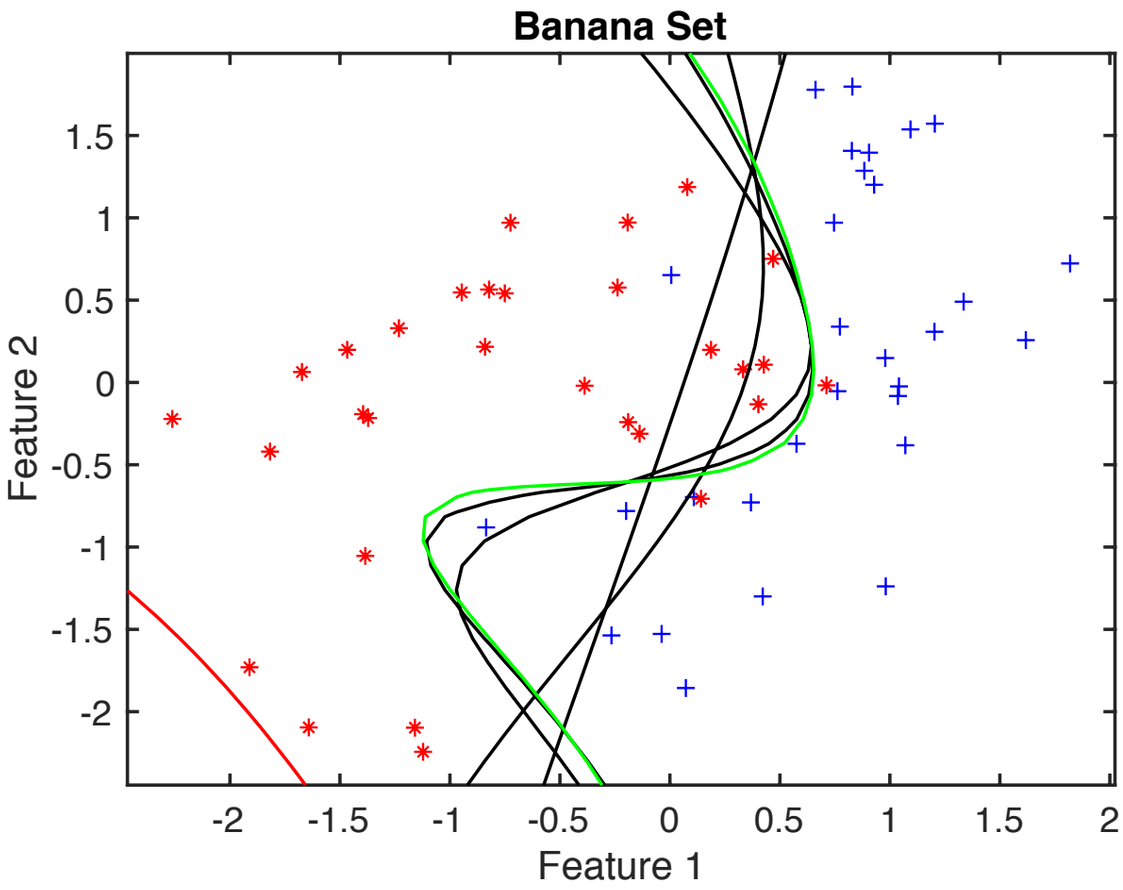
Training a network

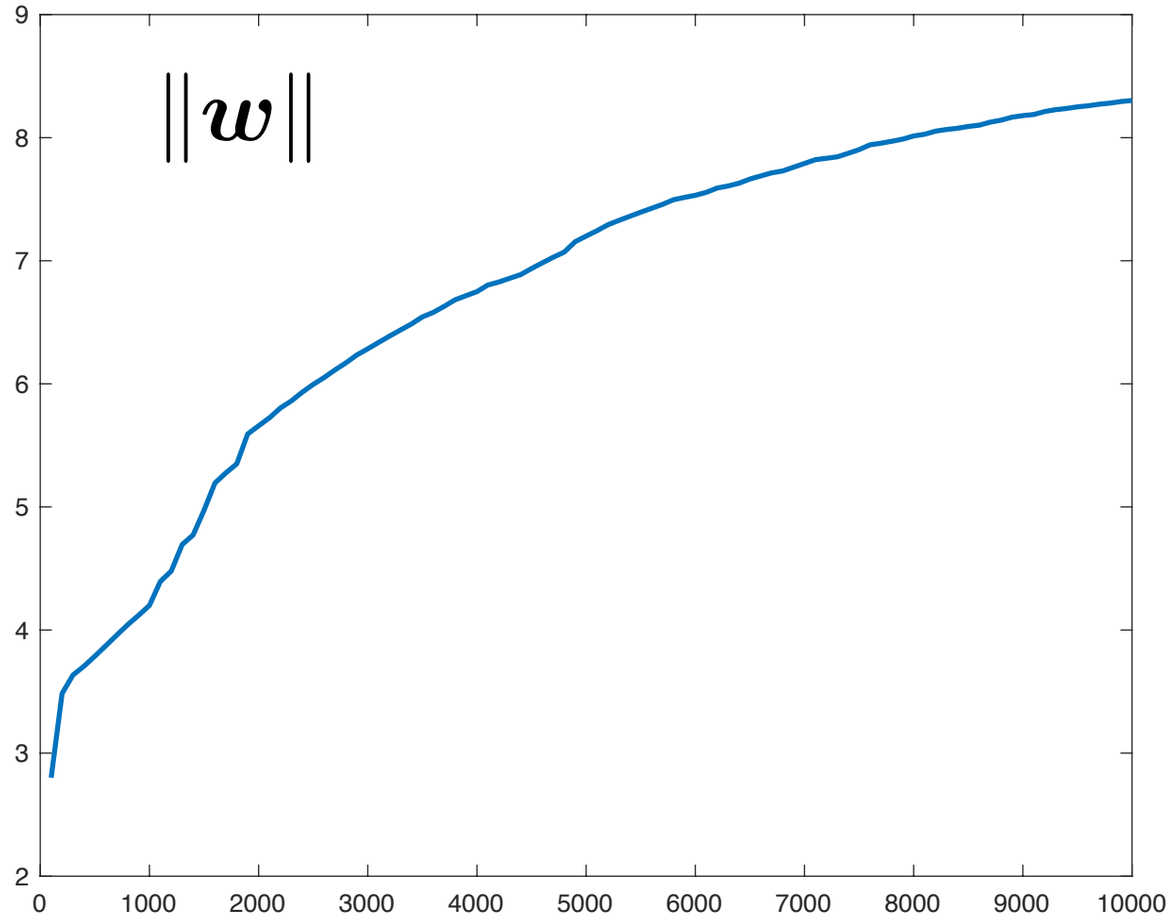




- At start: red decision boundary
- At end: green decision boundary
- Notice plateaus and sudden drops

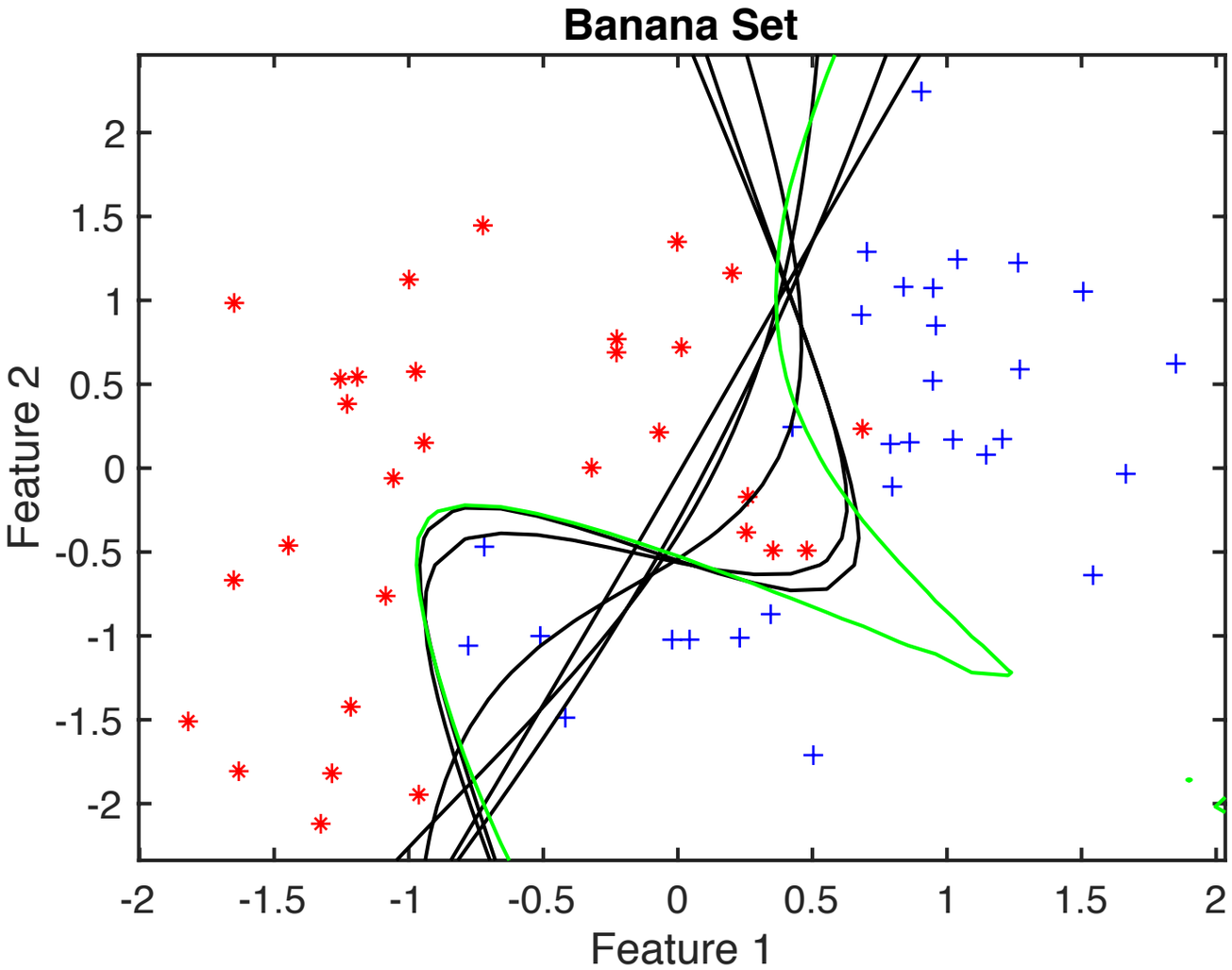
Training a network

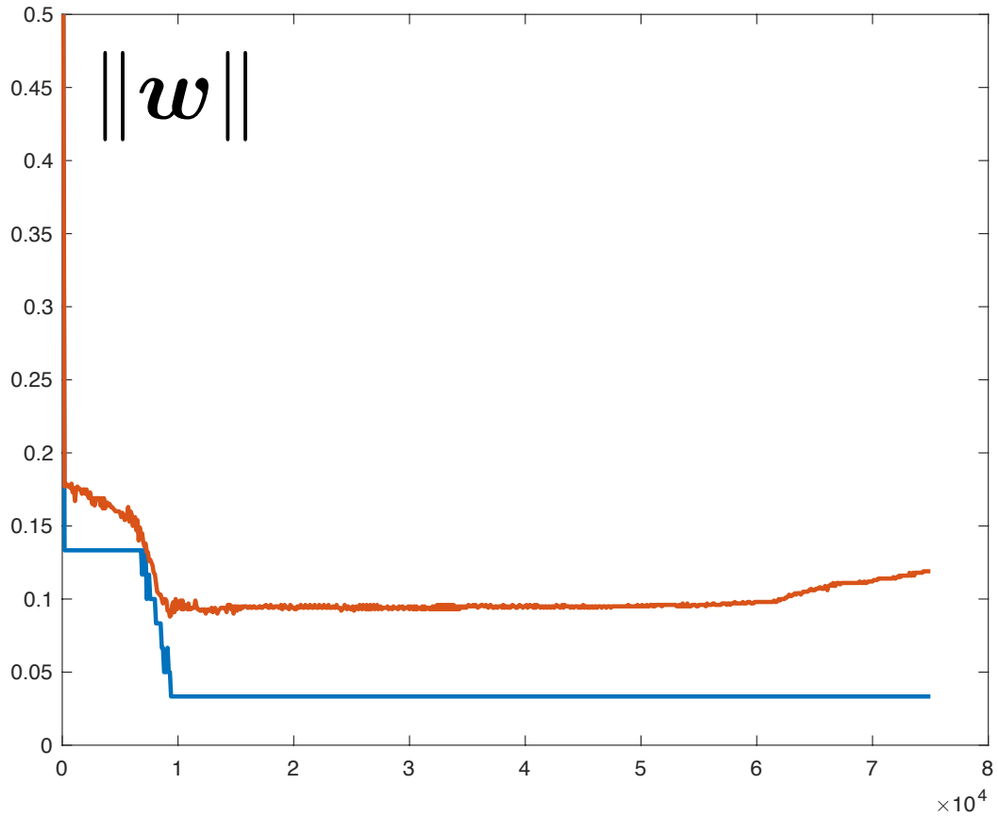




- Notice how the norm of the weights increases!

Training a network (overfitting)

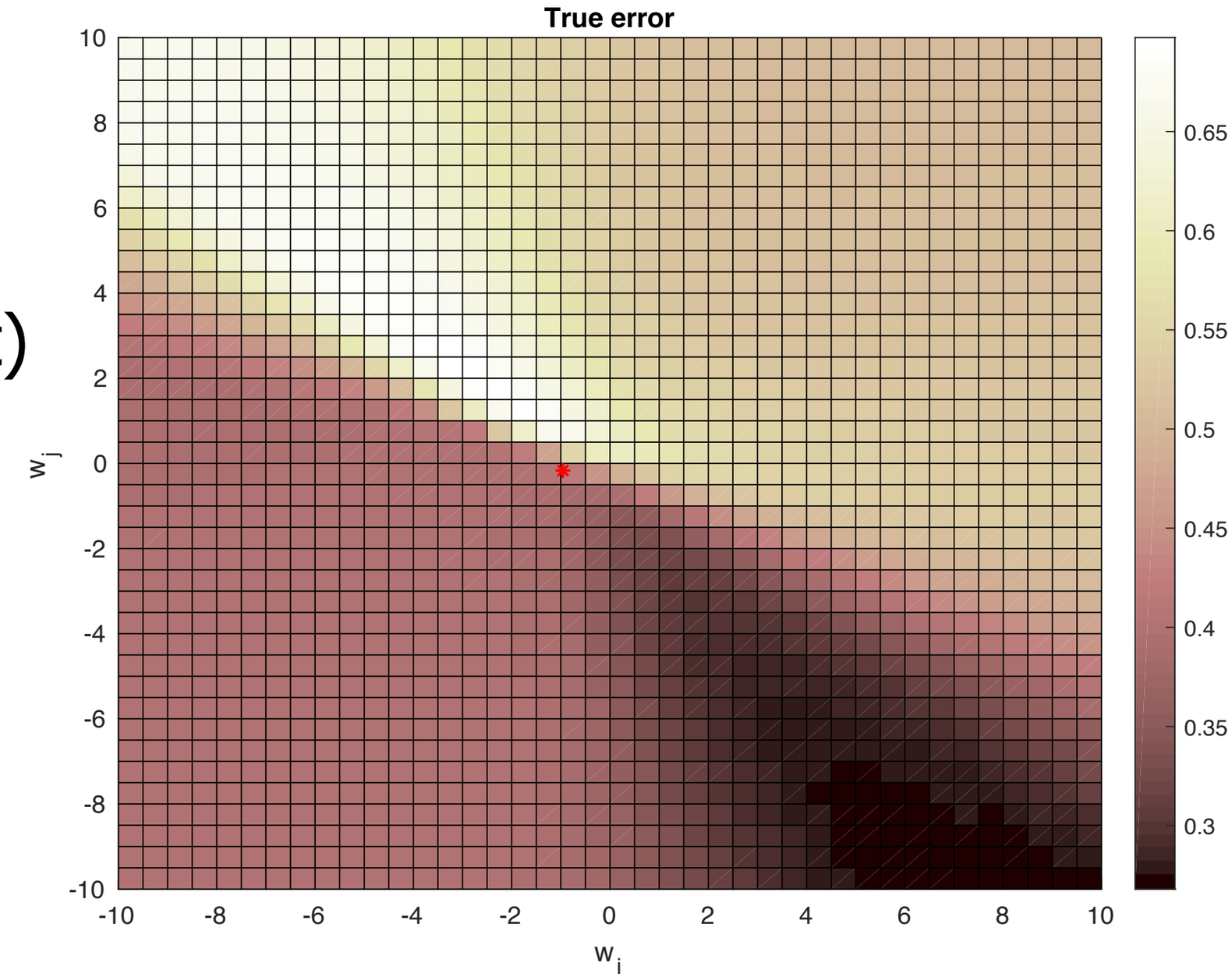




- Another training set
- Notice the overfitting…

Weight space

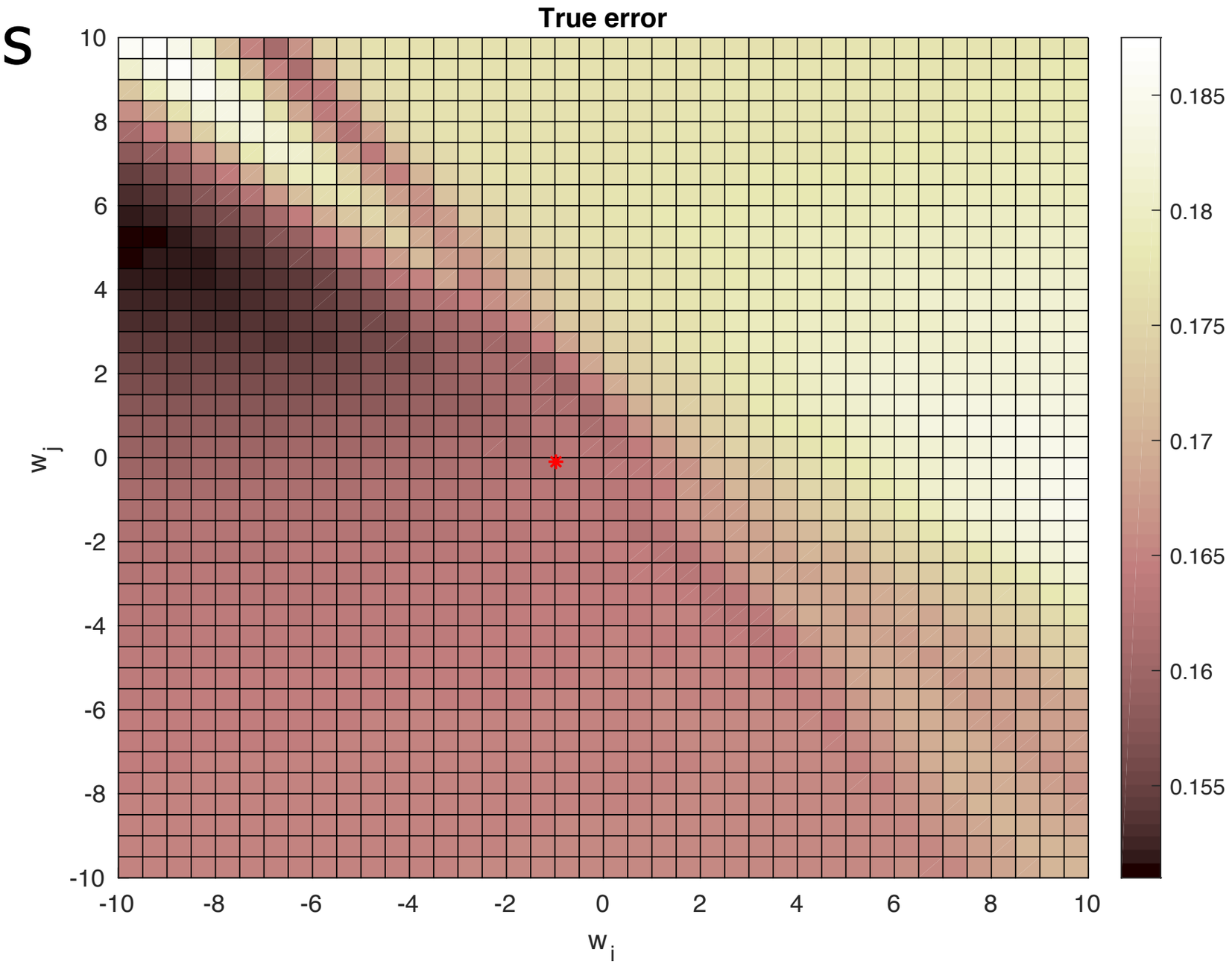
- Take two weights from the network
- Vary their values
- Compute the (test) error
- At initialisation:
- (notice the color bar)



Weight space

- After 100 iterations

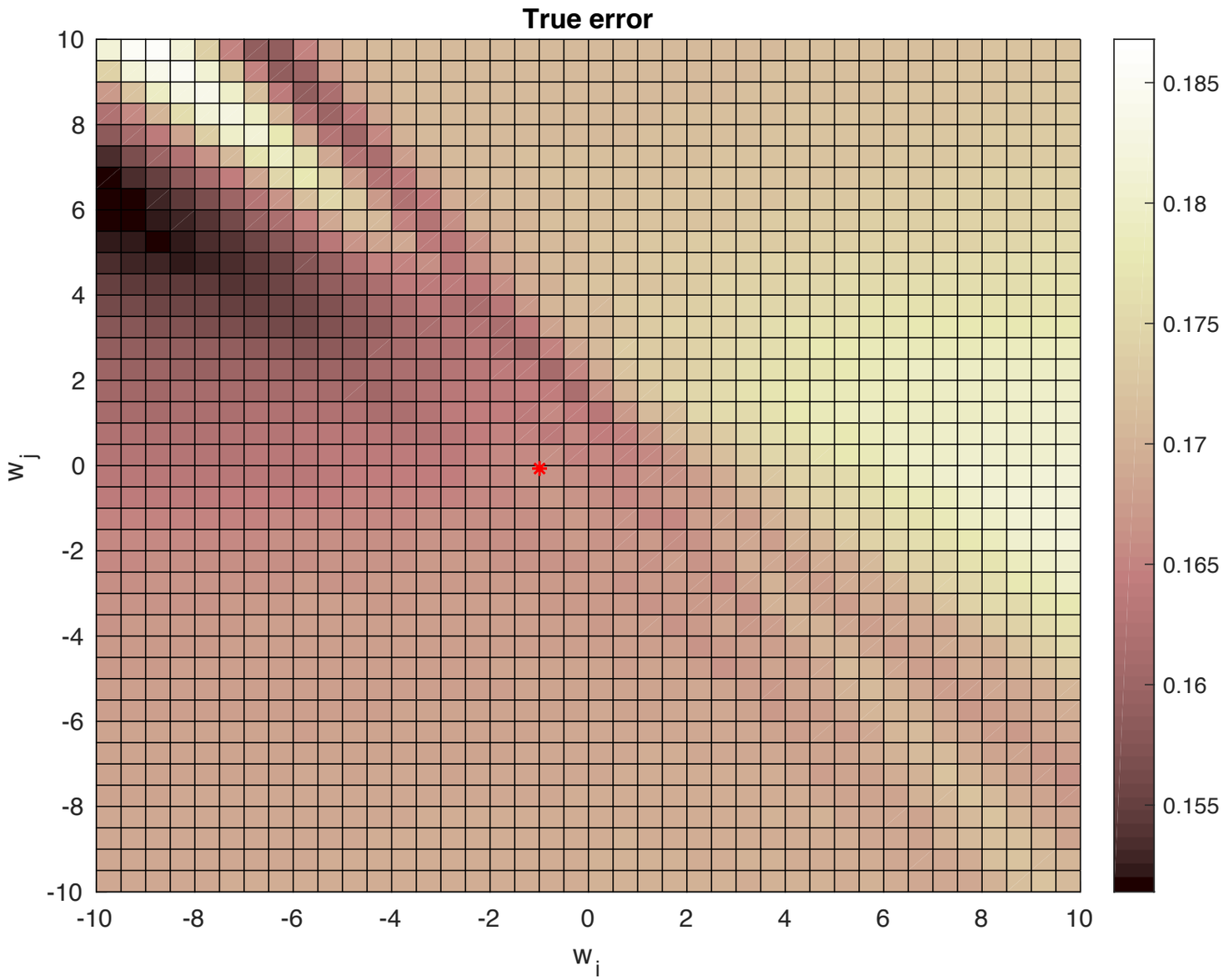
- Why has the surface changed?



Weight space

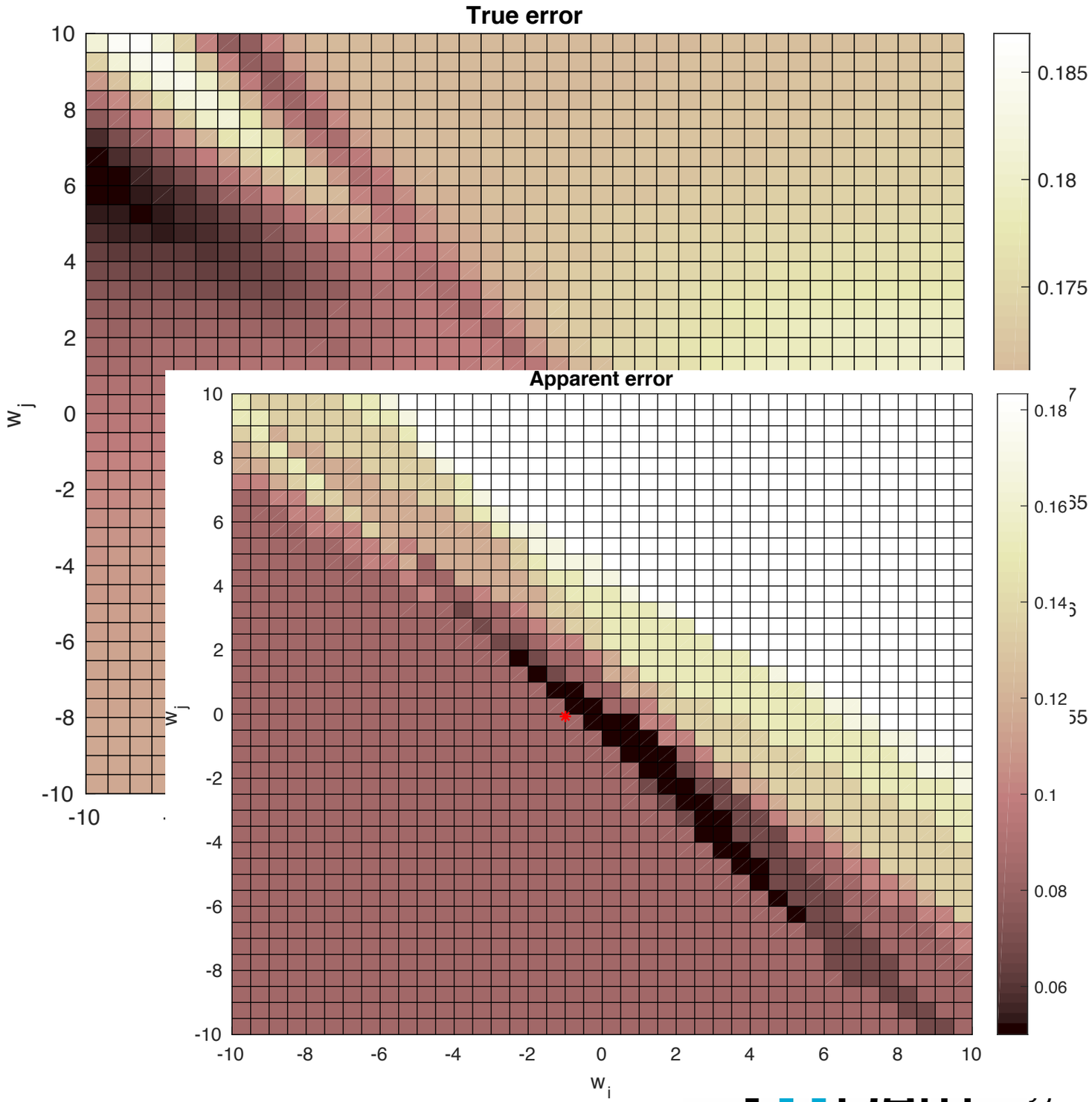
- After 400 iterations:

- More iterations does not do a lot? Why?



Weight space

- After 400 iterations:
- More iterations does not do a lot? Why?
- Look at the apparent error!



Noise robustness

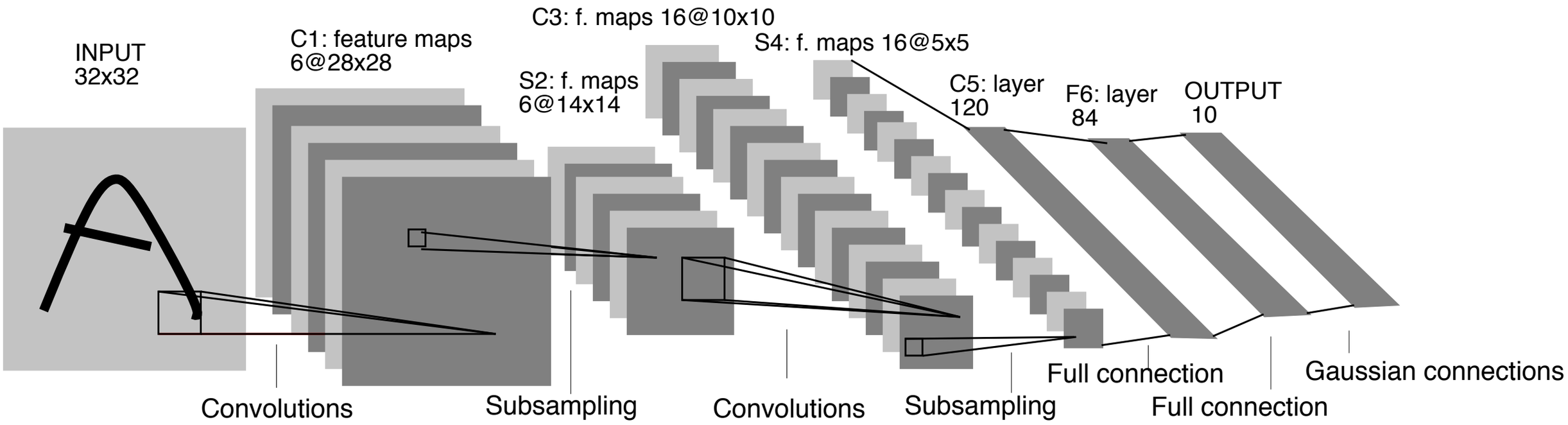
- Not only variations in **input** data, also noisy variations of the **weights** and **outputs**

- Noise added to input can be considered data augmentation

- Noise added to weights encourages stability of the model
- Noise with infinitesimal variance is equivalent to… weight norm regularisation

- Noise added to the output means label smoothing

Parameter sharing/tying



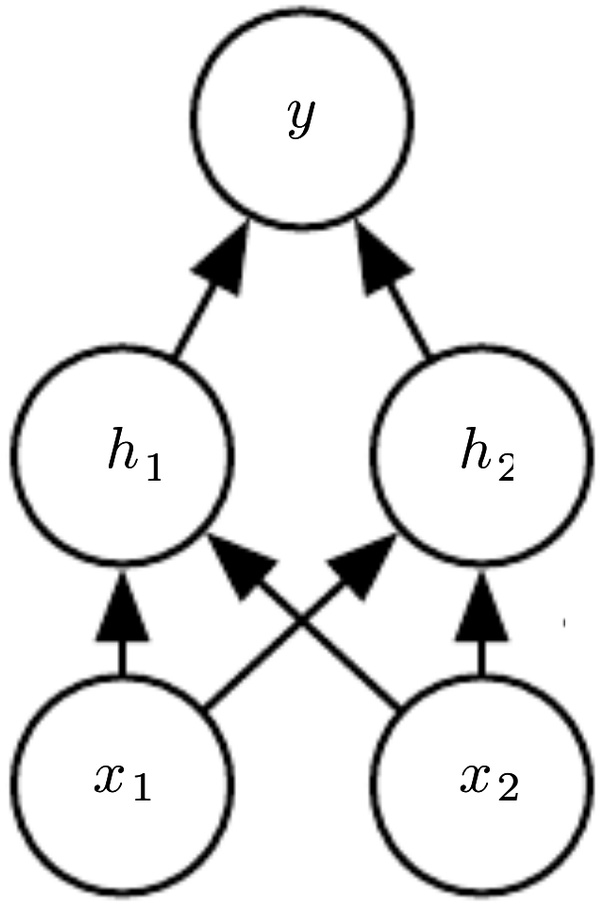
- (huge) reduction of weights

Dropout (1)

- Combination of weight decay and noise injection
- (I'm not fond on the 'bagging' reasoning)

- Each each training step: randomly select a fraction 'p' of the nodes, and make them inactive (set to 0)

- Perform standard backprop on remaining network



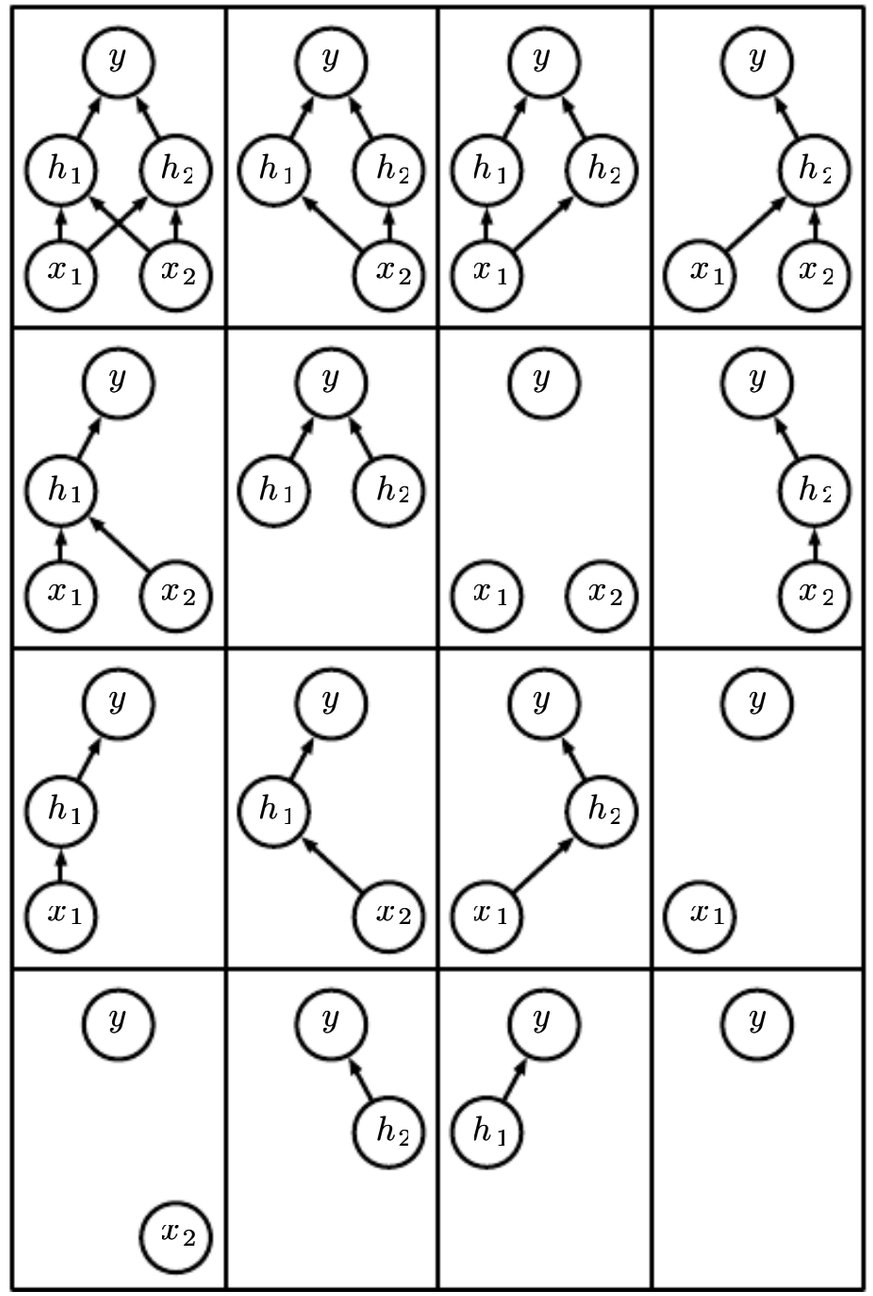
Base network

Dropout

- Combination of weight decay and noise injection
- (I'm not fond on the 'bagging' reasoning)

- Each each training step: randomly select a fraction 'p' of the nodes, and make them inactive (set to 0)

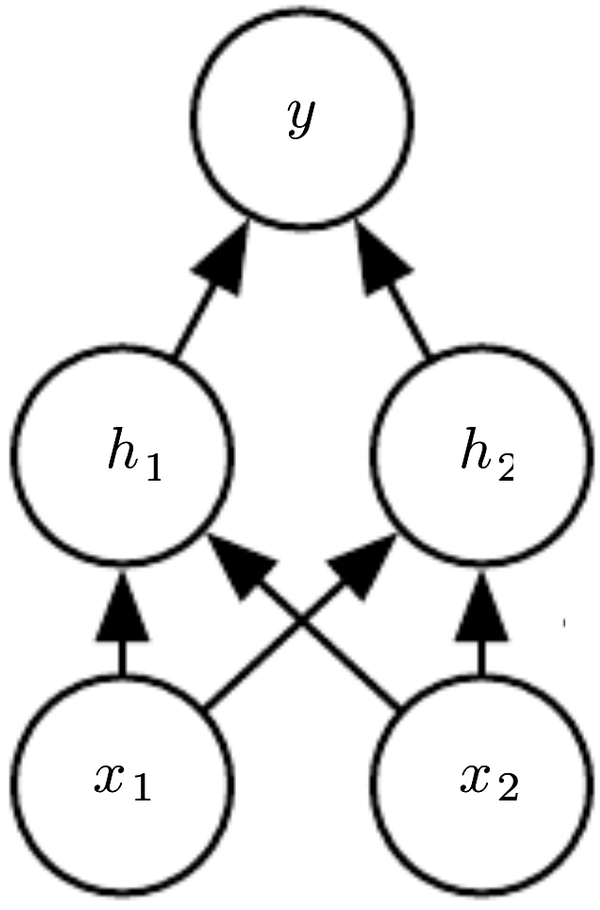
- Perform standard backprop on remaining network
- Average outcome over some/all realisations



Dropout (2)

- Combination of weight decay and noise injection
- (I'm not fond on the 'bagging' reasoning)

- Each each training step: randomly select a fraction 'p' of the nodes, and make them inactive (set to 0)
- Rescale other weights with p
- Perform standard backprop on remaining network

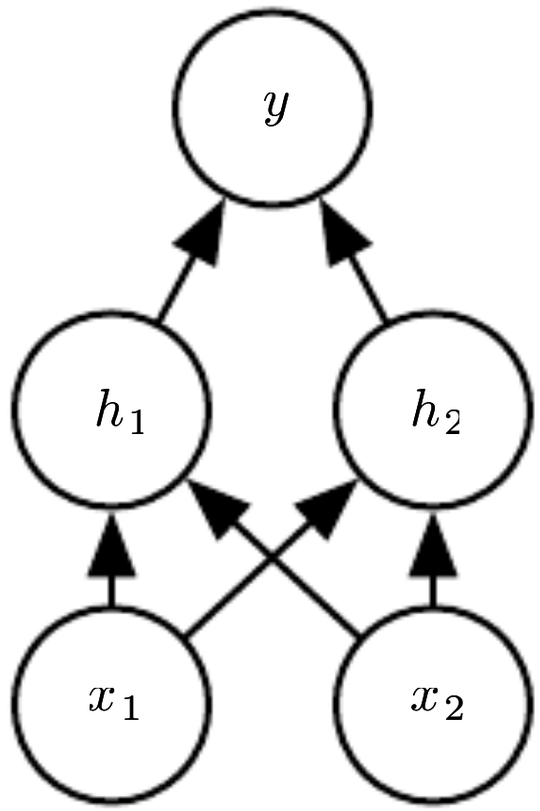


Base network

Dropout

- At evaluation time:

- simulate a few (20) networks with dropped out nodes, average their outcomes

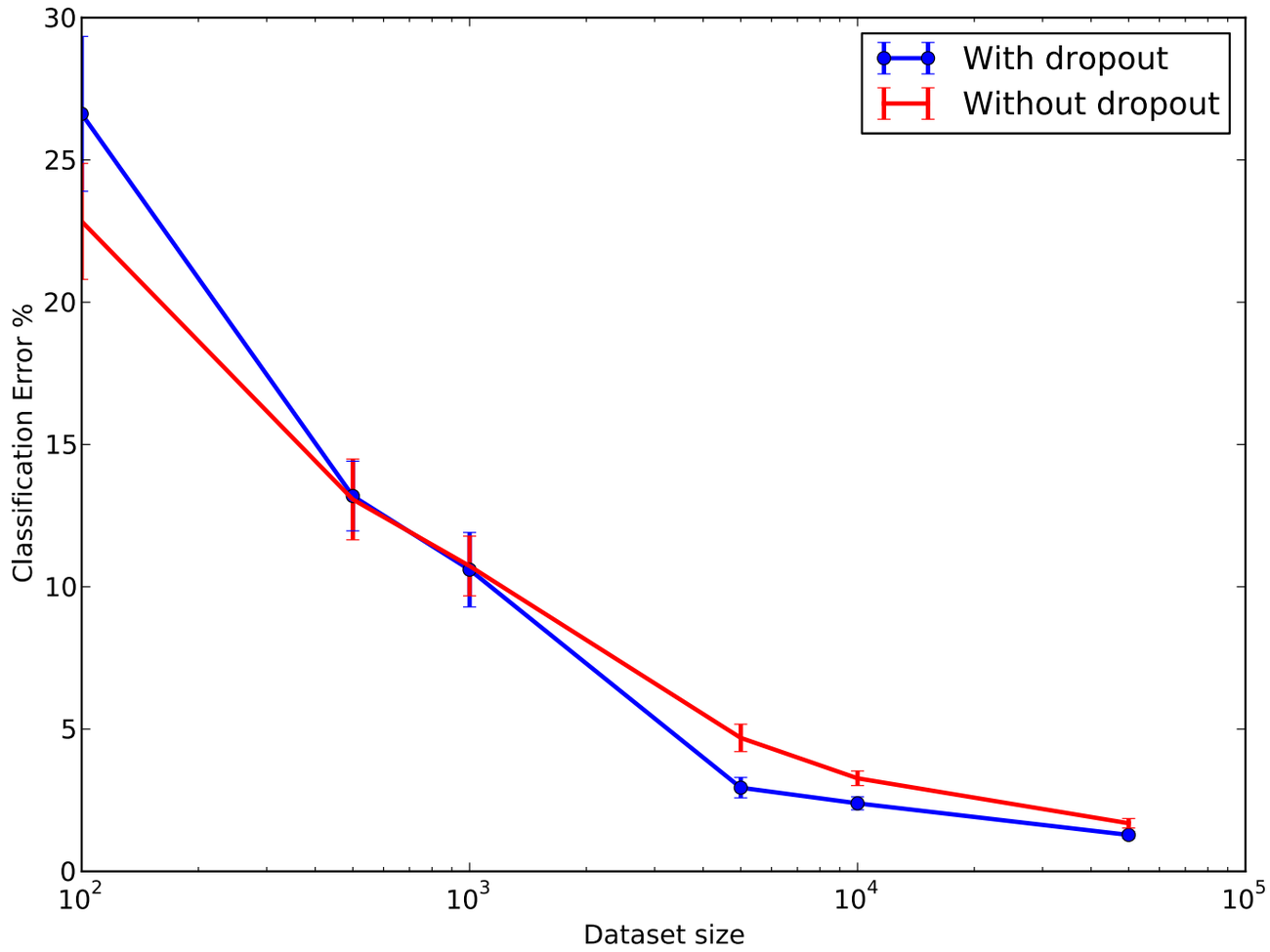


- do not drop nodes, but rescale the weights:

$$\begin{aligned}\mathbf{w}_{new} &= \mathbf{w}_{org} P[\text{no dropout}] + \mathbf{0} P[\text{dropout}] \\ &= \mathbf{w}_{org}(1-p)\end{aligned}$$

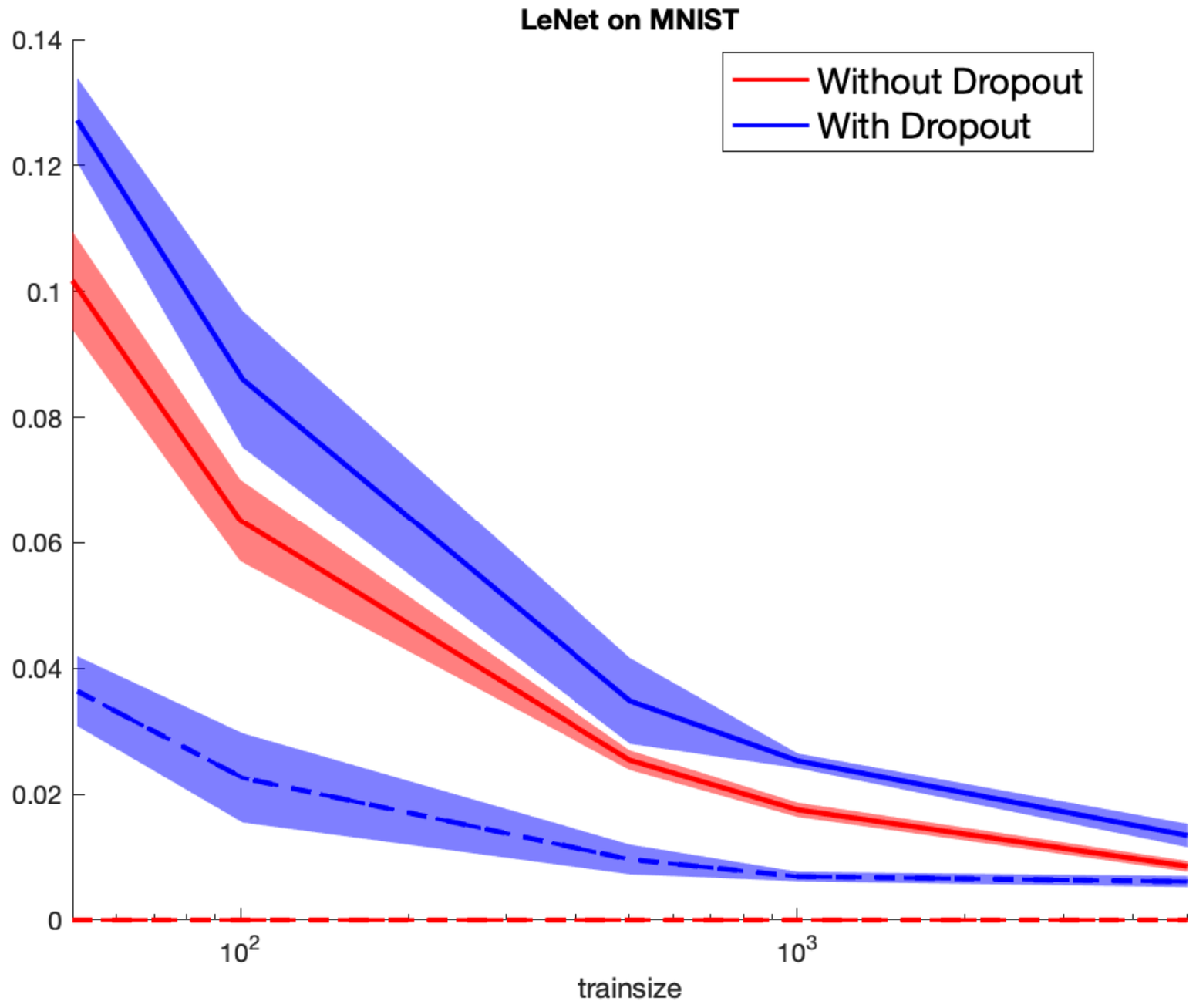
Is it really true?

- In the paper of Srivastava et al, JMLR 2014 this result is presented:



- It’s not regularizing, it is **increasing** the complexity!

My experience with Dropout



- I cannot make dropout to work at all

Other peoples experience

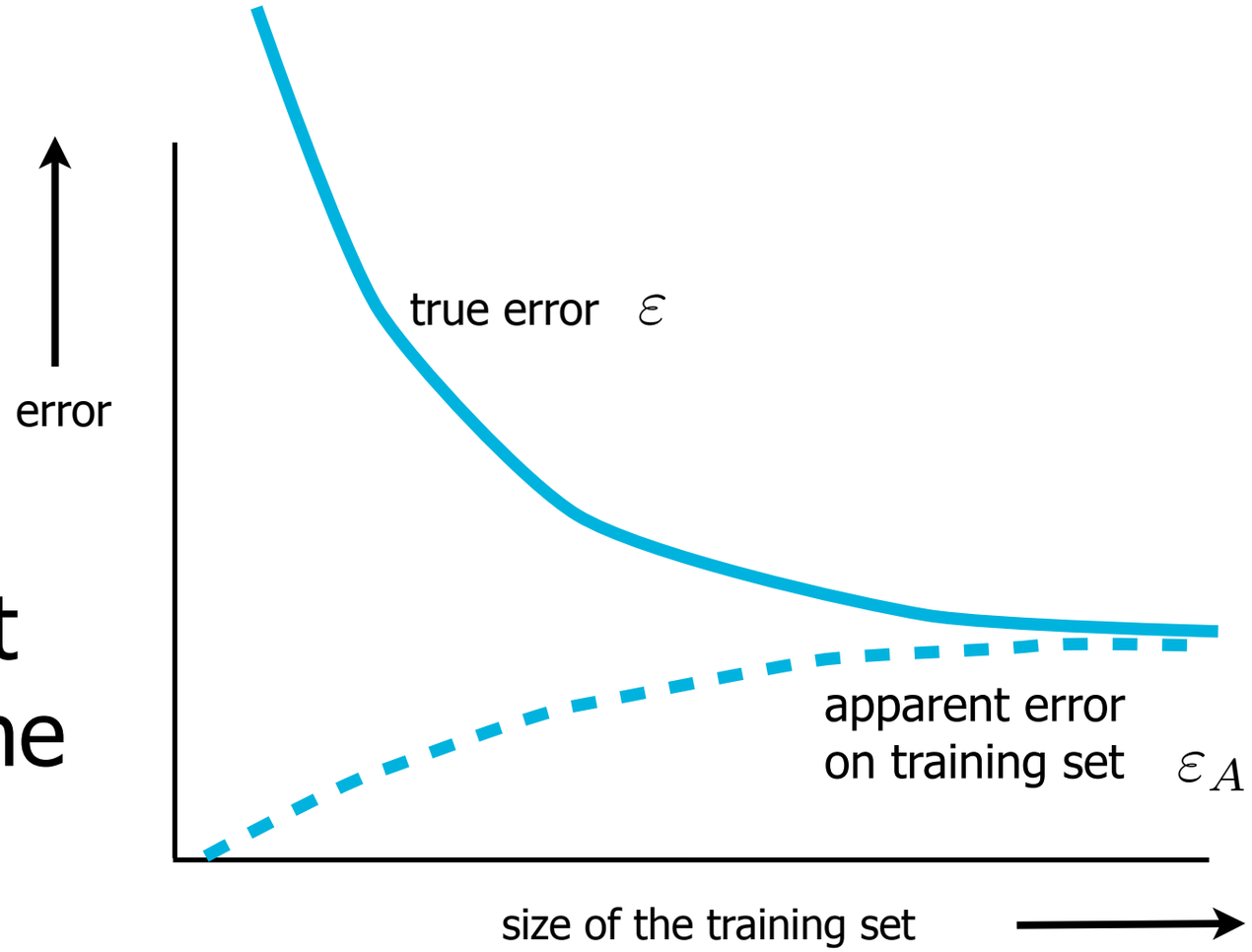
- Some proof that it works for ‘wide’ networks, not for ‘slim and deep networks’
- Not very suitable for convolutional networks
- Alternatives include SpatialDropout, probabilistic weighted pooling, max-drop, cutout

- Need for future research
  (why is weight decay not enough?)

TU Delft

Returning to the main point

- When do you need regularisation? When you overfit…
- When do you overfit? …
- Learning curve!



- Unfortunately, it is expensive
- And, you need to repeat it a few times (to find the stability)…

Warning!

- Try to avoid looking too much at the test set
- Each time you overfit yourself!
  It influences **your** choices for hyperparameters

- Look at the test set, only when you want to optimise on a specific dataset
  But you **cannot** conclude that your estimated true error is the error you will get on a new dataset

- Beware of what is claimed in the literature. Not always true

Questions to think about…

- Do you need regularisation when you have infinite amount of training data?
- How can you know that you have to regularise less?
- The goal of the optimisation of a network is to reach the global optimum?
- Are there other regularisation strategies?
- How much is the number of weights reduced by weight sharing (see pg. 26)?
- Do you have to change the regularisation if you change your loss function?
- Pg.16: if the bound says $\|w\|_1^3$, why not use $\|w\|_1^3$?

Contents

- Intro overfitting, regularisation
- Dataset augmentation
- Generalisation bounds
- Regularisation
- Training a neural network
  - Parameter norm penalties, weight decay
  - Noise robustness
  - Early stopping
  - Parameter tying
  - (Dropout)

Questions/remarks?

- ?