Applied Deep Learning

Dr. Philippe Blaettchen
Bayes Business School (formerly Cass)

www.bayes.city.ac.uk

Learning objectives of today

Goals: Understand the difficulties in using neural networks in practice, and the more advanced concepts to overcome these difficulties

- Bias and variance, as well as regularization tools
- The problem of vanishing and exploding gradients, as well as slow learning
- Hyperparameter tuning

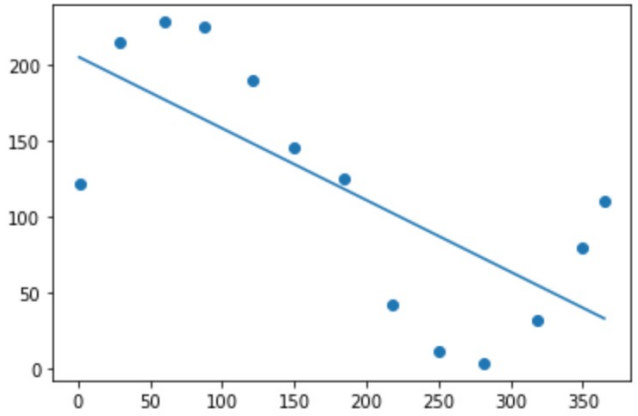
How will we do this?

- We look at the theoretical foundations of both the issues and the solutions
- In class, we will then introduce the TensorFlow code to implement a range of different solutions

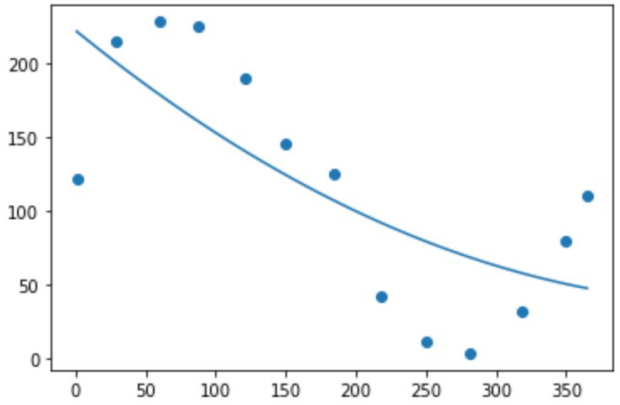
BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Bias-variance trade-off

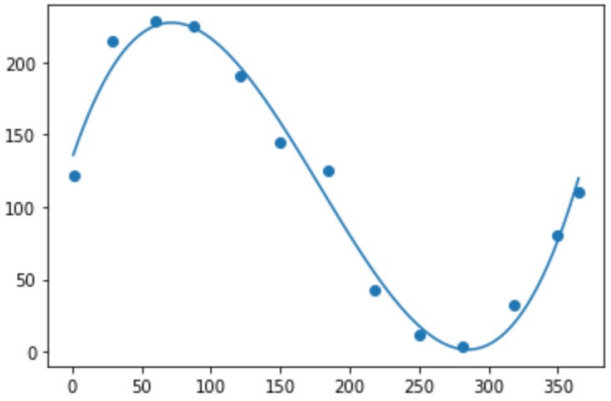
Looking back: predicting sales using polynomial regression and the problem of overfitting



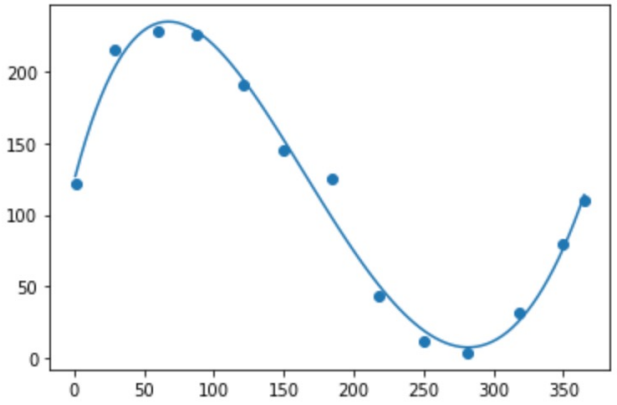
d=1, R^2=0.516



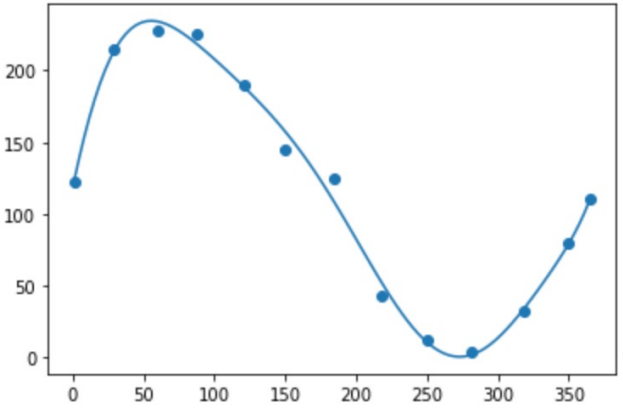
d=2, R^2=0.531



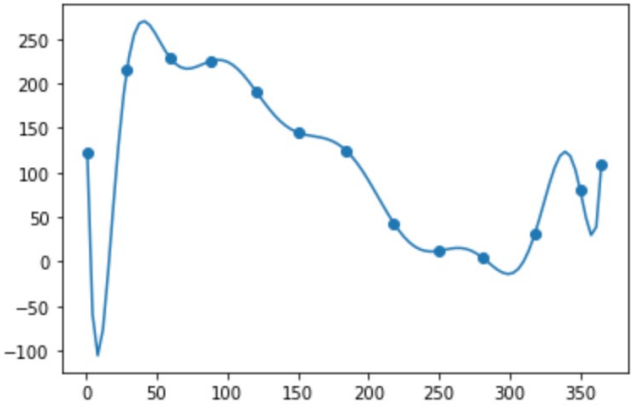
d=3, R^2=0.979



d=5, R^2=0.985



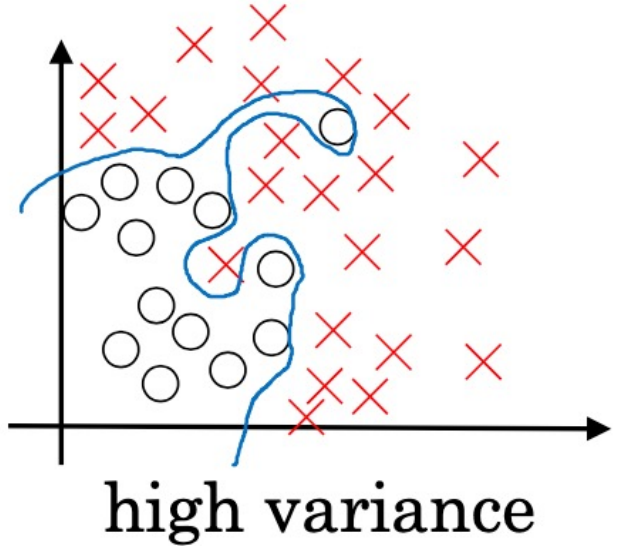
d=8, R^2=0.992



d=12, R^2=1

BAYES BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

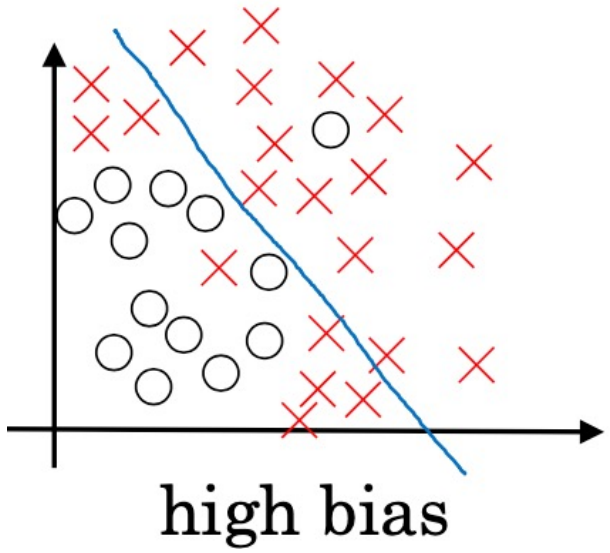
Bias and variance

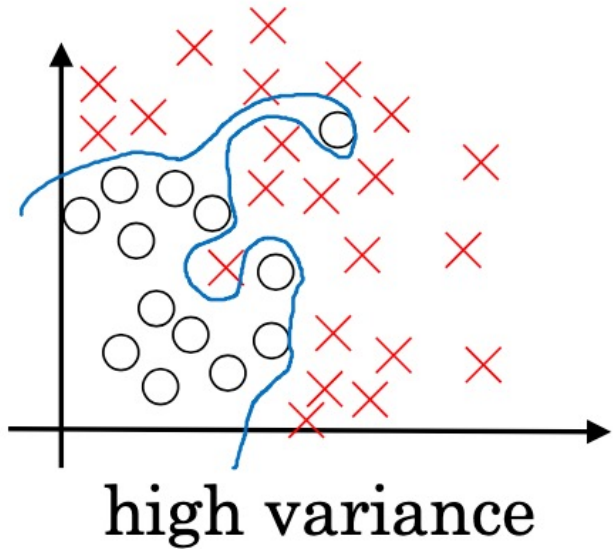


Source: DeeplearningAI

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Bias and variance

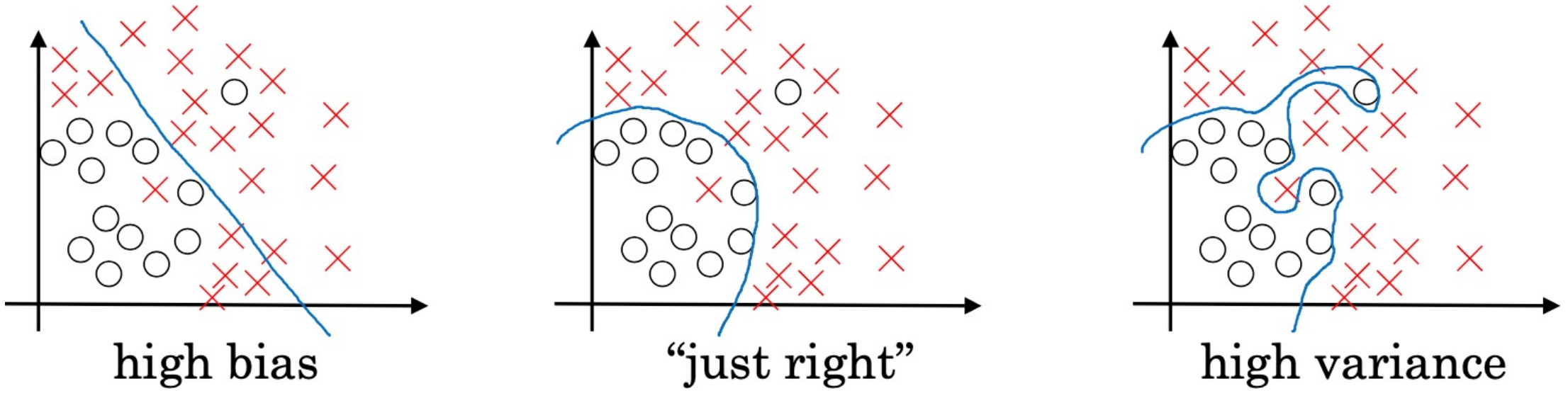




Source: DeeplearningAI

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Bias and variance



Source: DeeplearningAI

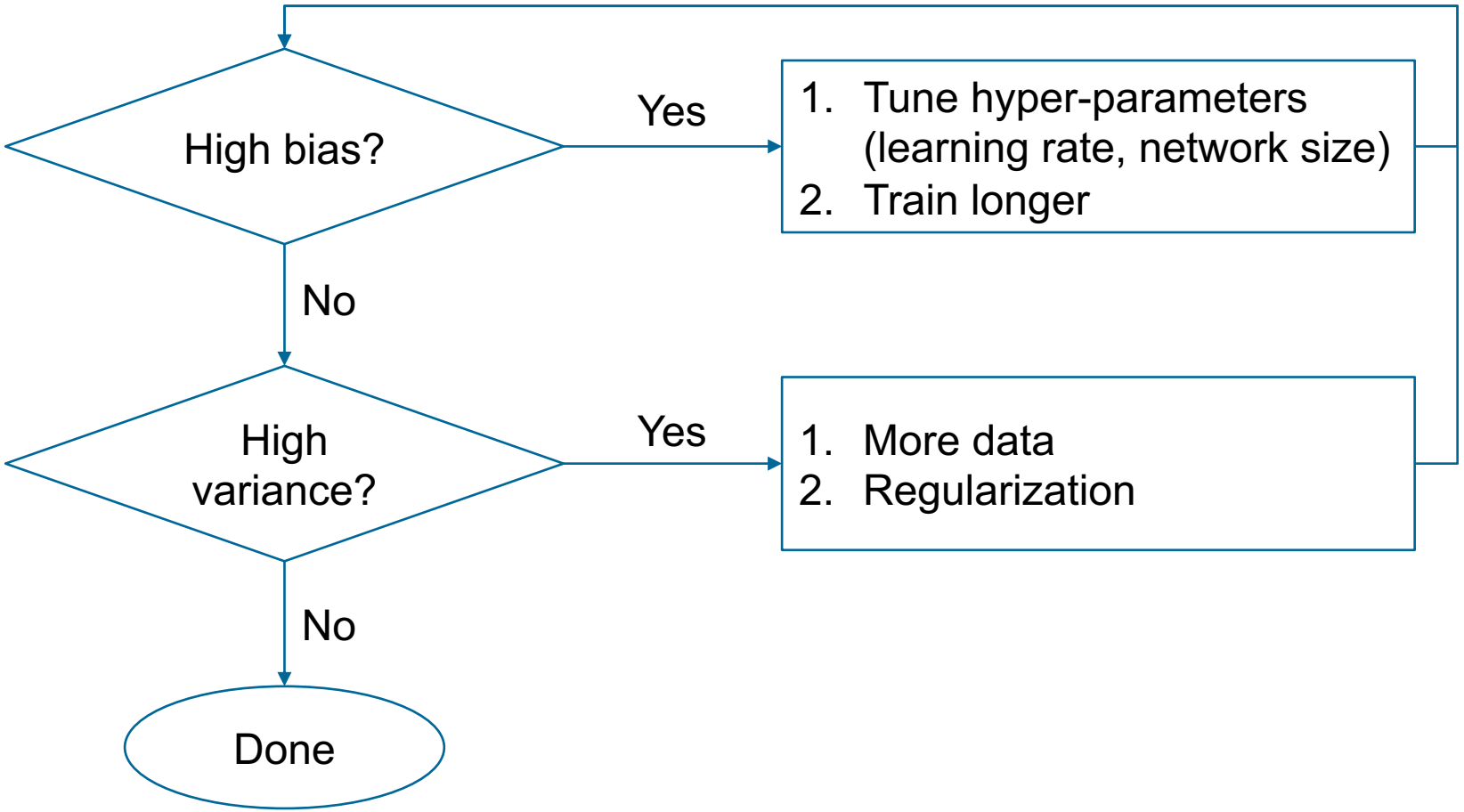
BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Recognizing bias and variance

"Bayes error"	(Avoidable) bias	1%	1%
Error on training set	Variance	2%	10%
Error on validation set		10%	12%
		High variance	High bias

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

How to deal with bias and variance



Source: DeeplearningAI

BAYES BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

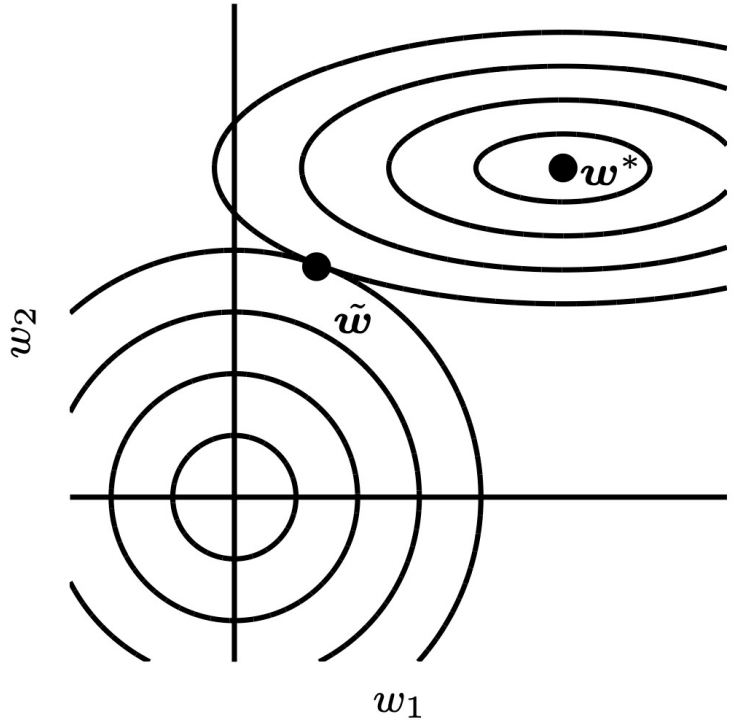
L2-regularization ("Ridge regression")

$$J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L^{(i)} \quad \Longleftrightarrow \quad J_R(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L^{(i)} + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_2 = \frac{1}{n}\sum_{i=1}^{n} L^{(i)} + \frac{\lambda}{2}\sqrt{\theta_1^2 + \theta_2^2 + \cdots + \theta_m^2}$$

- Gradient: $\nabla_\theta J_R(\boldsymbol{\theta}) = \nabla_\theta J(\boldsymbol{\theta}) + \lambda\boldsymbol{\theta}$
- Gradient descent update:

$$\begin{aligned}\boldsymbol{\theta} &:= \boldsymbol{\theta} - \alpha\nabla_\theta J_R \\ &= \boldsymbol{\theta} - \alpha\nabla_\theta J(\boldsymbol{\theta}) - \alpha\lambda\boldsymbol{\theta} \\ &= (1 - \alpha\lambda)\boldsymbol{\theta} - \alpha\nabla_\theta J(\boldsymbol{\theta})\end{aligned}$$

→ "weight decay"



Source: Goodfellow

BAYES BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

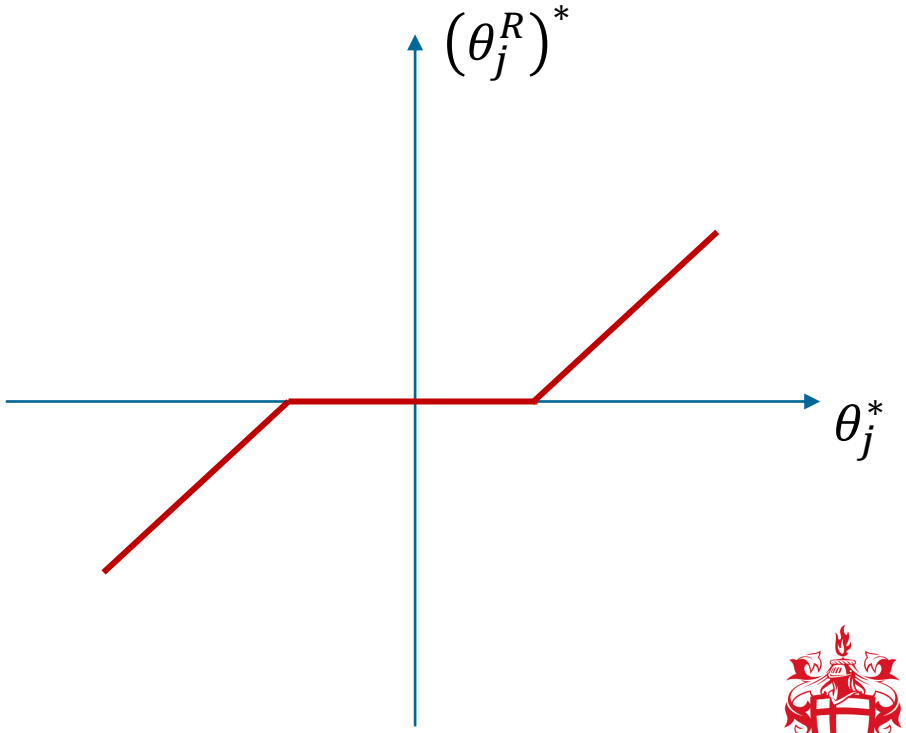
L1-regularization (“Lasso regression”)

$$J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L^{(i)} \quad \Longleftrightarrow \quad J_R(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L^{(i)} + \lambda\|\boldsymbol{\theta}\|_1 = \frac{1}{n}\sum_{i=1}^{n} L^{(i)} + \lambda\sum_{j=1}^{m}|\theta_j|$$

- Gradient: $\nabla_\theta J_R(\boldsymbol{\theta}) = \nabla_\theta J(\boldsymbol{\theta}) + \lambda\, sign(\boldsymbol{\theta})$
- Gradient descent update:

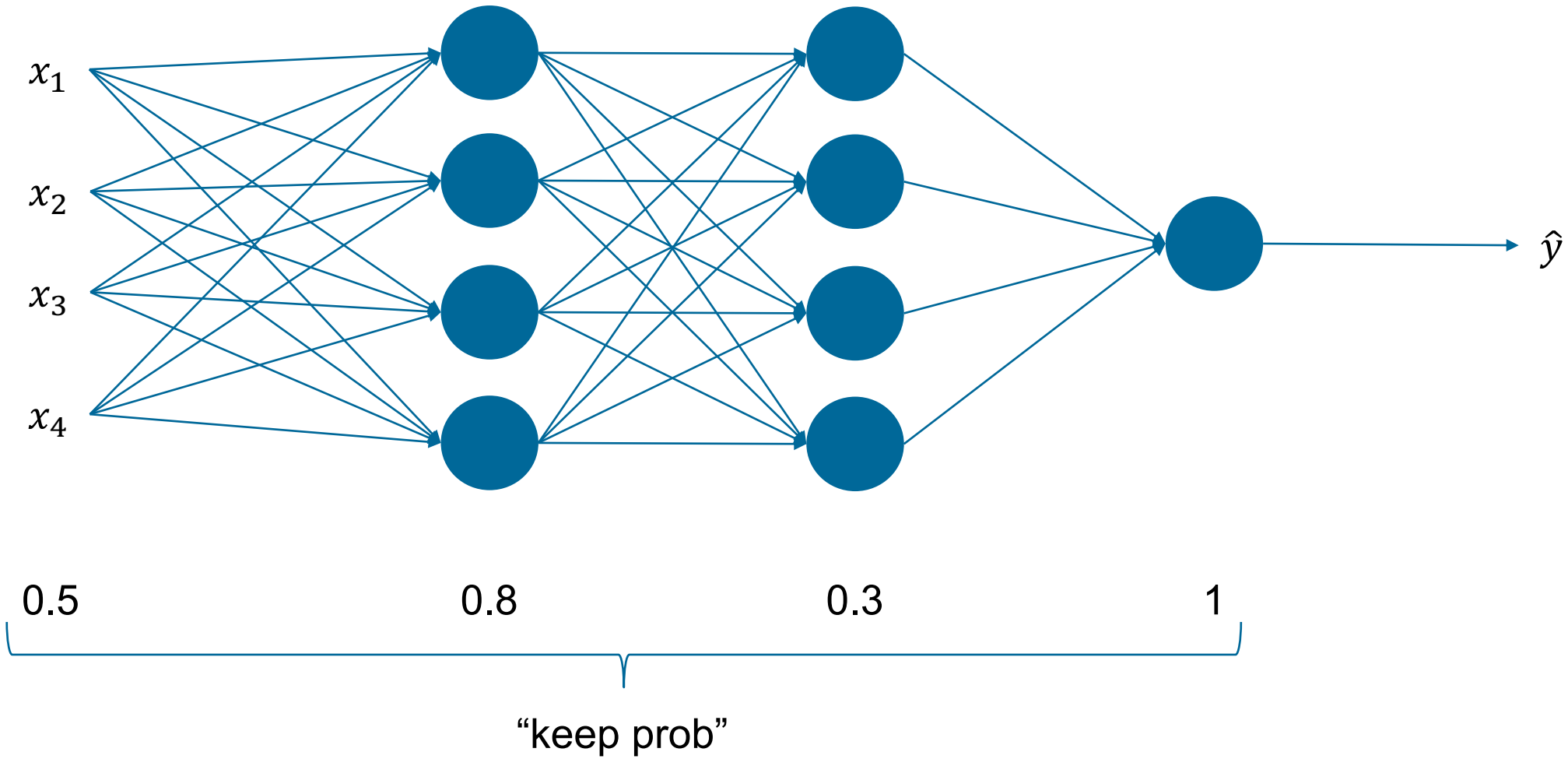
$$\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha\nabla_\theta J_R$$
$$= \boldsymbol{\theta} - \alpha\nabla_\theta J(\boldsymbol{\theta}) - \alpha\lambda\, sign(\boldsymbol{\theta})$$

→ “sparsity”



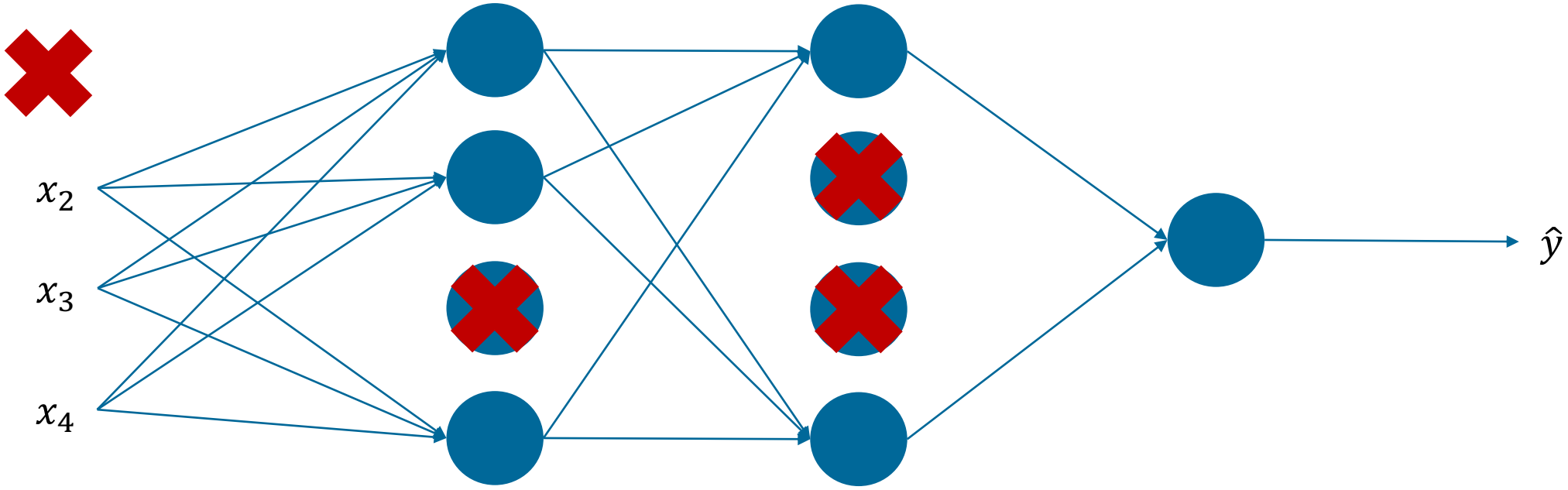
BAYES BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Dropout regularization



BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Dropout regularization

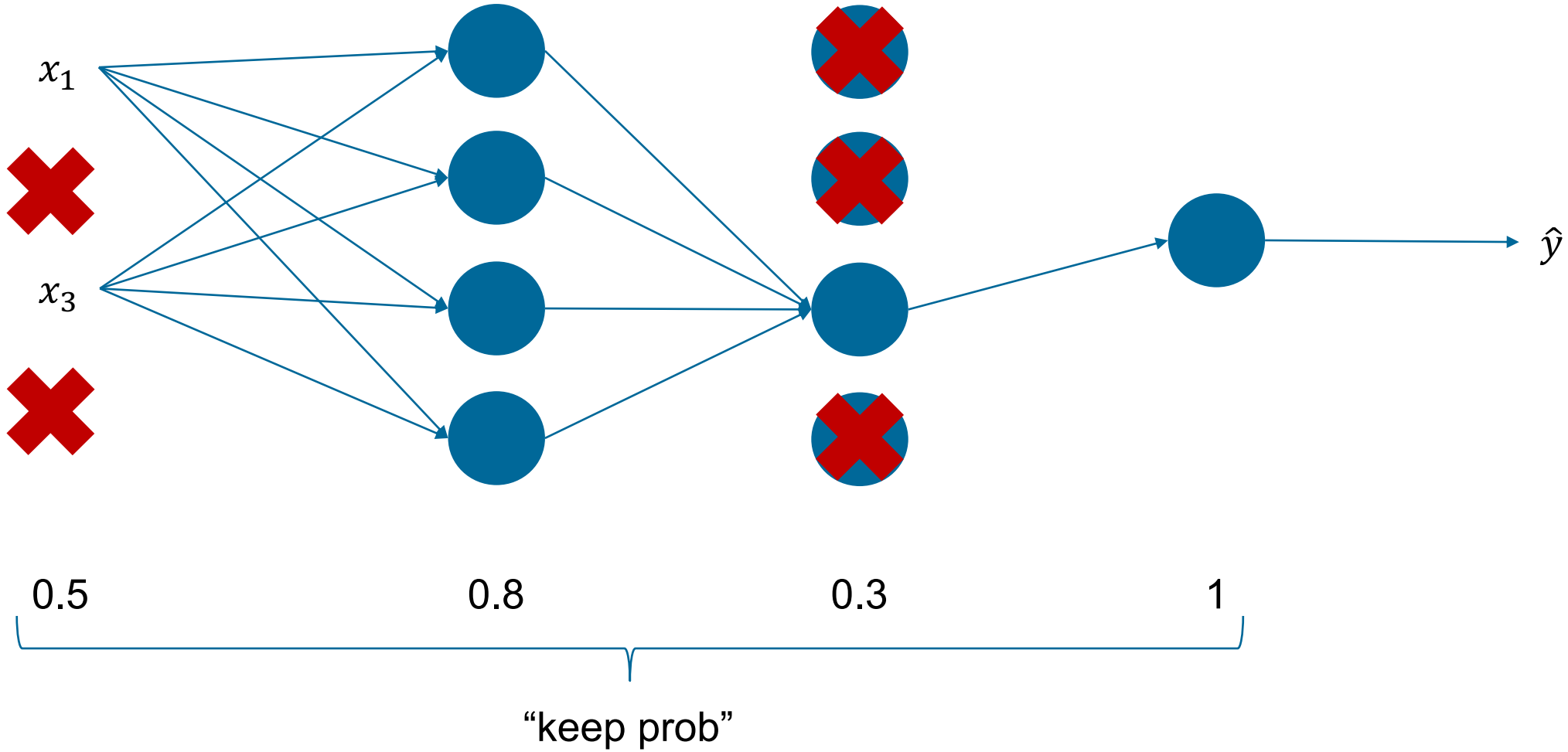


0.5 0.8 0.3 1

"keep prob"

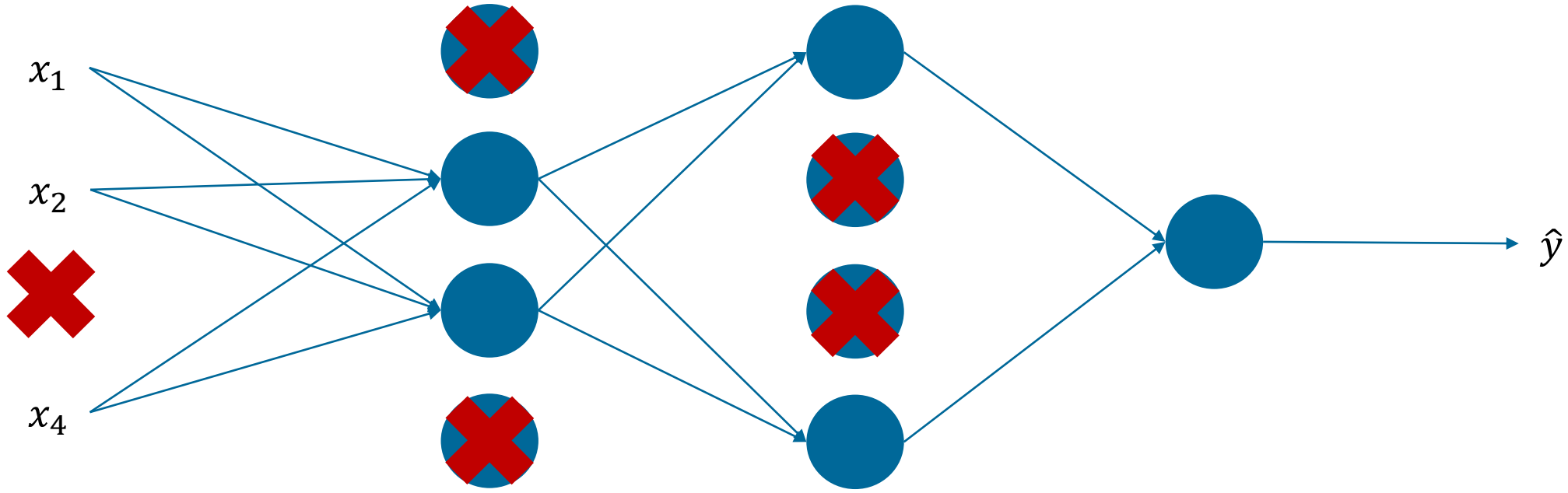
BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Dropout regularization



BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Dropout regularization



0.5 0.8 0.3 1

"keep prob"

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

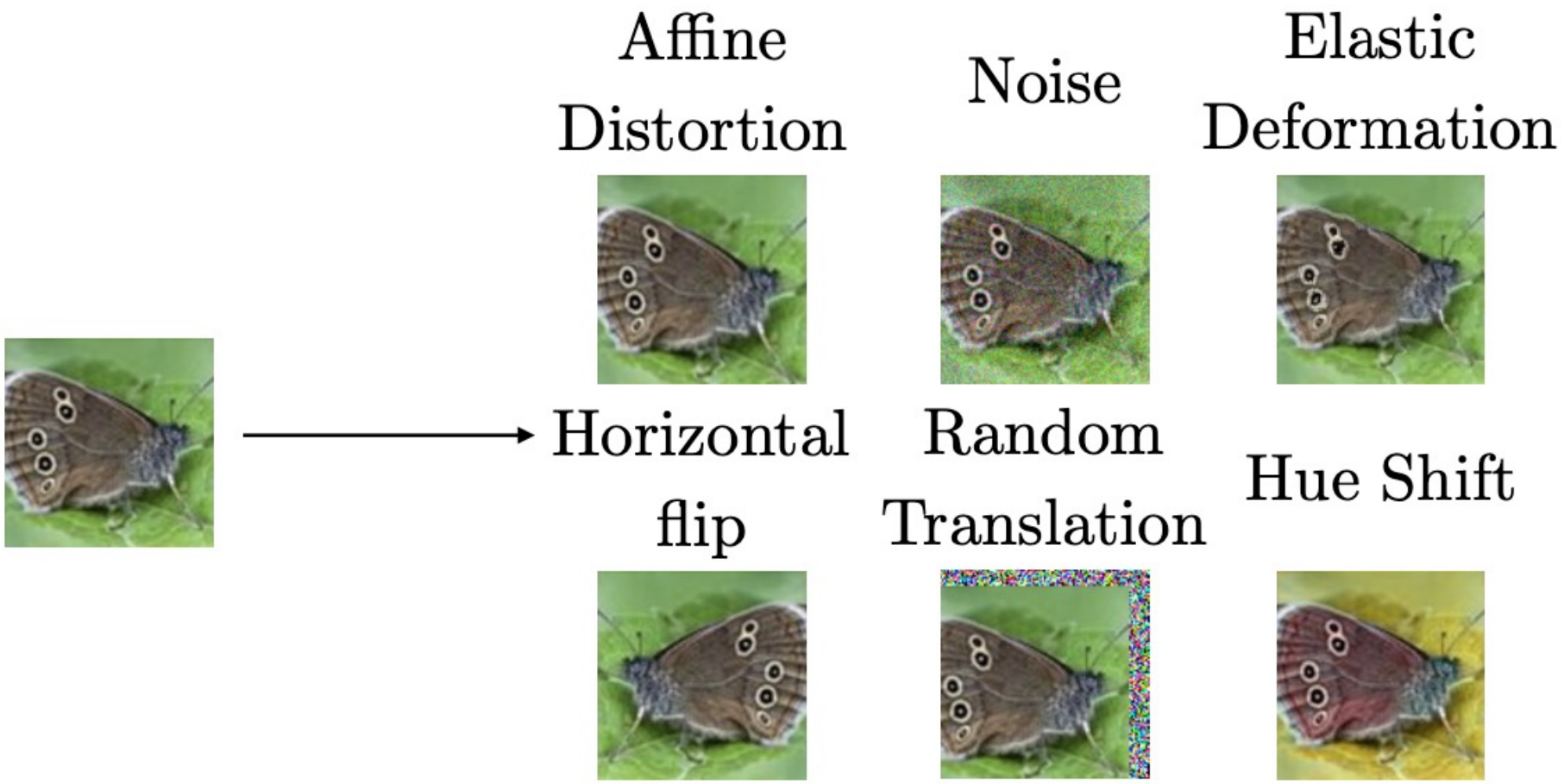
The idea behind dropout regularization

We cannot rely on any one feature (or previous neuron's activation), so the weights need to be spread out.

Effectively, we train multiple smaller networks in an ensemble.

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

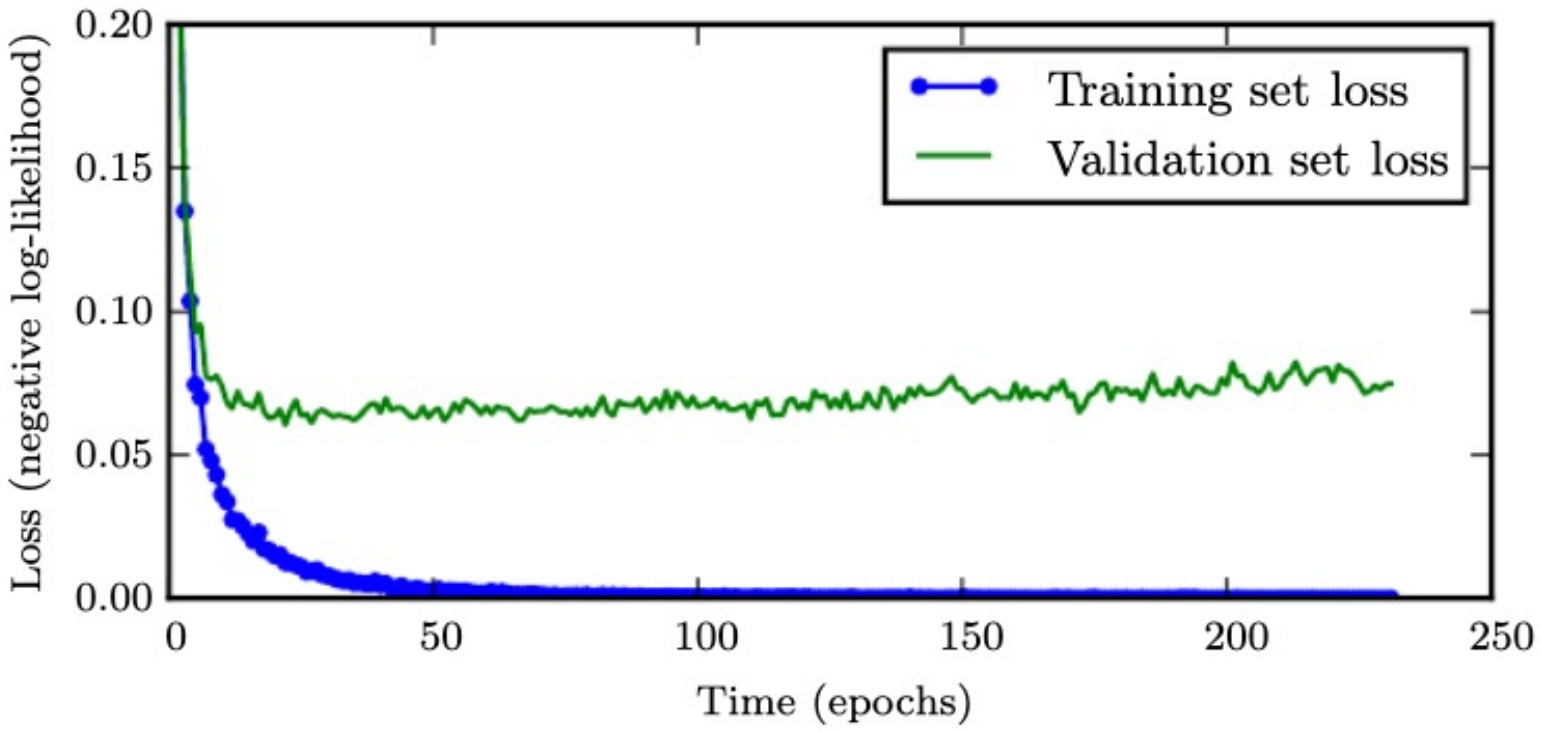
Dataset augmentation



Source: Goodfellow

BAYES BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Early stopping



Source: Goodfellow

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Improving learning

Other possible issues we might come across in training

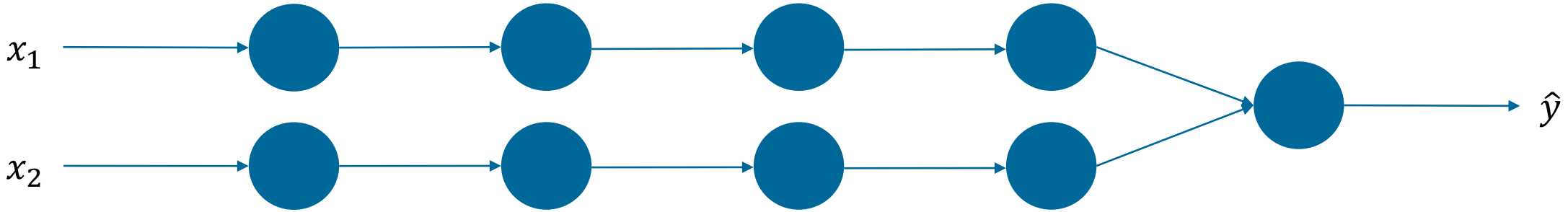
- Vanishing and exploding gradients
  - → Initialization
  - → Activation functions
  - → Data normalization
  - → Batch normalization

- Very slow training
  - → Mini-batch gradient descent
  - → Momentum optimization, RMSprop, Adam
  - → Reusing pretrained layers

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Improving learning – vanishing and exploding gradients

Why gradients vanish and explode



For simplicity, assume a linear activation function, as well as $b = 0$. We then have that

$$\nabla_{W^{[l]}} J = \left(A^{[l-1]}\right)^T \left(W^{[l+1]} \dots W^{[L]}\right)^T$$

$$= \left(A^{[l-1]}\right)^T \begin{pmatrix} w_{1,1}^{[l+1]} w_{1,1}^{[l+2]} \dots w_{1,1}^{[L]} & 0 \\ 0 & w_{2,2}^{[l+1]} w_{2,2}^{[l+2]} \dots w_{2,2}^{[L]} \end{pmatrix}^T$$

If we assume the weights are w everywhere, we have

$$= \left(A^{[l-1]}\right)^T \begin{pmatrix} w^{L-l} & 0 \\ 0 & w^{L-l} \end{pmatrix}^T$$

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Possible counter: the “right” initialization

- Glorot-initialization (assuming logistic sigmoid, tanh, softmax activation):
  - variance of inputs ≈ variance of outputs
  - variance of gradients before layer ≈ variance of gradients after layer

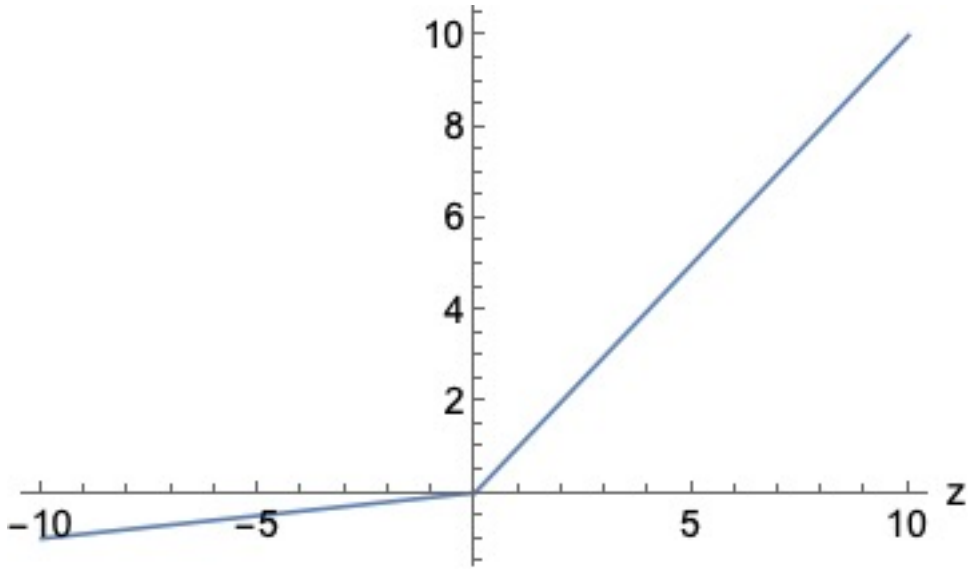
  - Idea: given a layer with in inputs and out neurons, distribute weights either
    - normally, with mean 0 and variance $\frac{1}{\frac{in+out}{2}}$, or (kernel_initializer=“glorot_normal”)
    - uniformly between $\left[-\sqrt{\frac{3}{\frac{in+out}{2}}}, \sqrt{\frac{3}{\frac{in+out}{2}}}\right]$ (default)

- He-initialization (assuming ReLU and its variants):
  - Idea: given a layer with in inputs and out neurons, distribute weights either
    - normally, with mean 0 and variance $\frac{2}{in}$, or (kernel_initializer=“he_normal”)
    - uniformly between $\left[-\sqrt{\frac{6}{in}}, \sqrt{\frac{6}{in}}\right]$ (kernel_initializer=“he_uniform”)

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

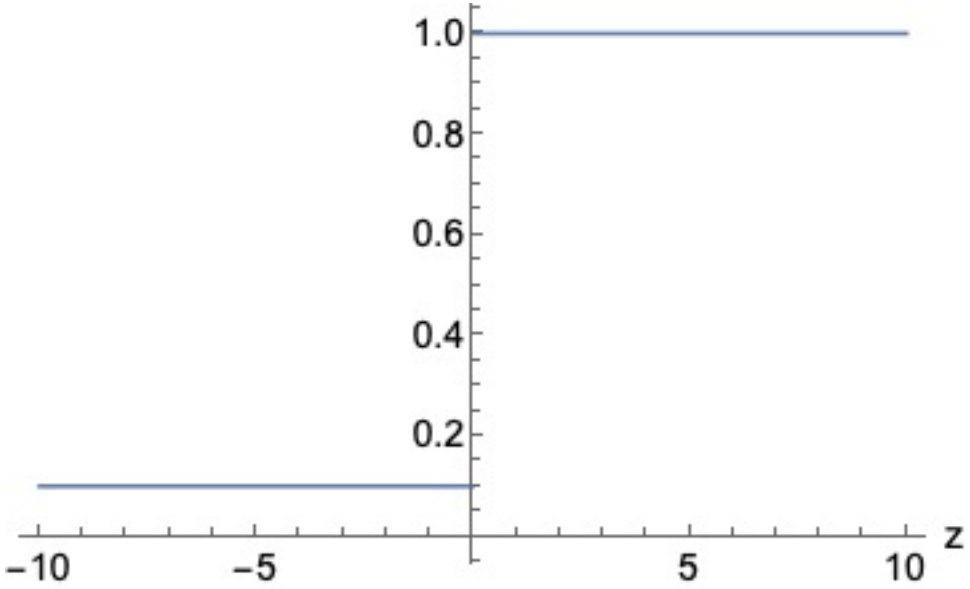
Another possible counter: non-saturating activation functions

Leaky ReLU



$$f(z) = \max\{0.1z, z\}$$

"Derivative"



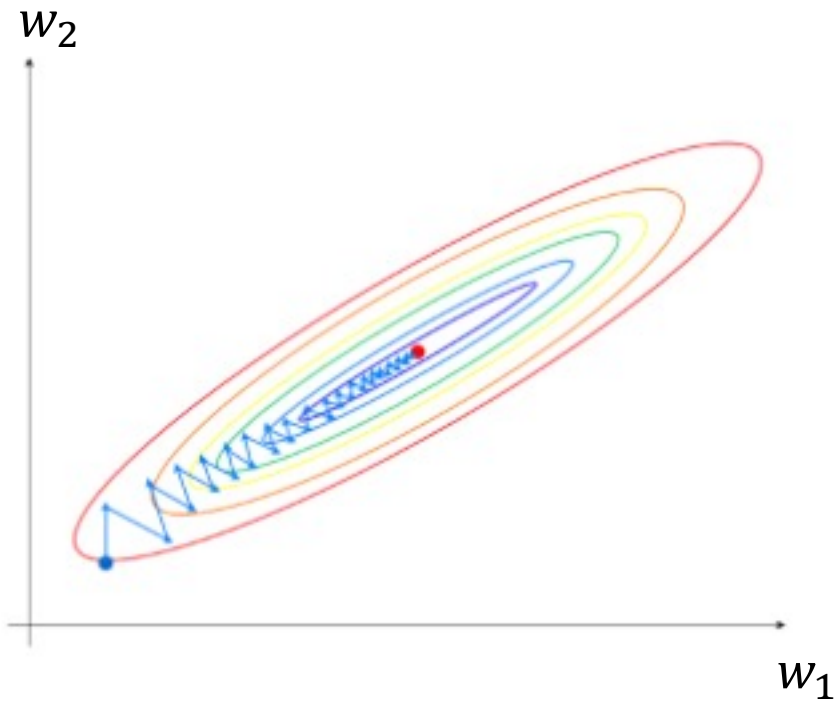
$$f'(z) = \begin{cases} 0.1, if\ z < 0 \\ 1, if\ z > 0 \end{cases}$$

Consider also: ELU

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Finally: normalizing the inputs

E.g., $x_1 \in (0,100)$
$x_2 \in (0,1)$

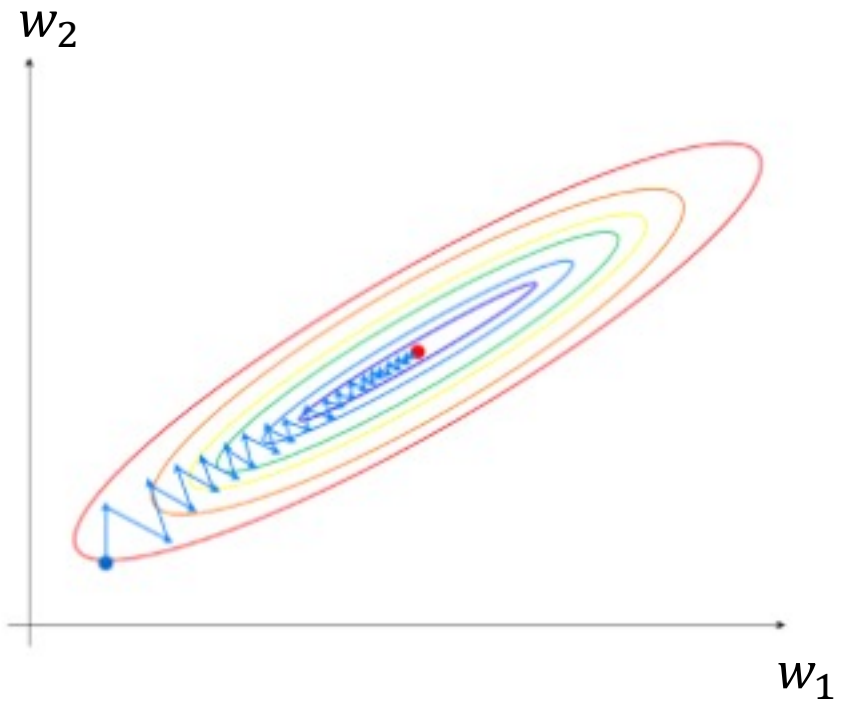


Source: Erdem

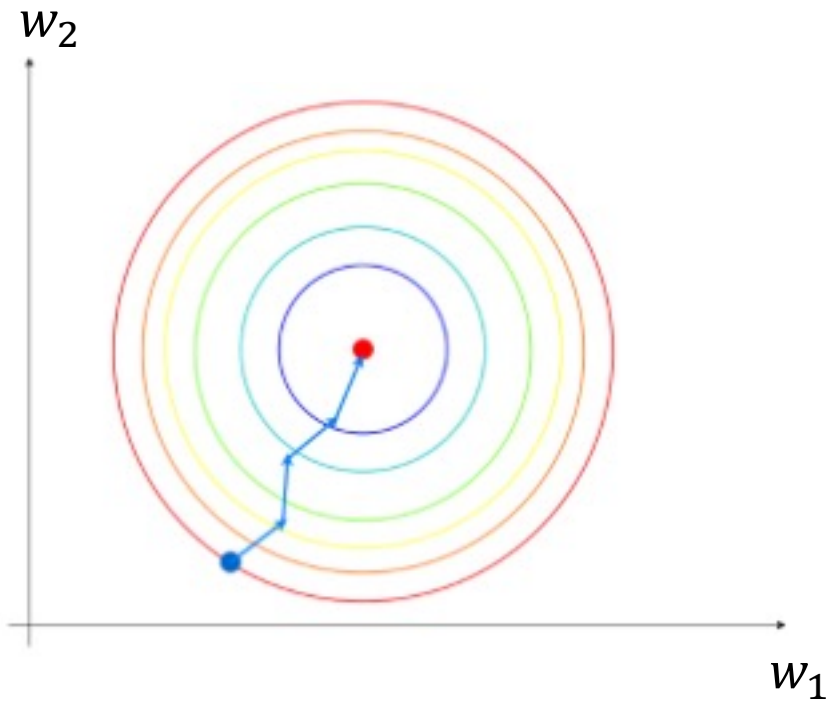
BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Finally: normalizing the inputs

E.g., $x_1 \in (0,100)$
$x_2 \in (0,1)$



E.g., $x_1 \in (0,1)$
$x_2 \in (0,1)$



Source: Erdem

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Normalizing the inputs also for deeper layers: batch normalization

- Add an operation before/after activation function. For each input:
  - Standardize it (zero-centering, and division by standard deviation)
  - Then, scale it by a parameter γ and add a shift β (we **learn** these parameters)

- When we use the neural network to compute predictions, we don't necessarily have means and standard deviations, however (or the new observations might not be independent, …)
  → also keep track of a running mean μ and variance σ (we keep track of a moving average, but we are not learning, so these are **non-trainable** parameters)

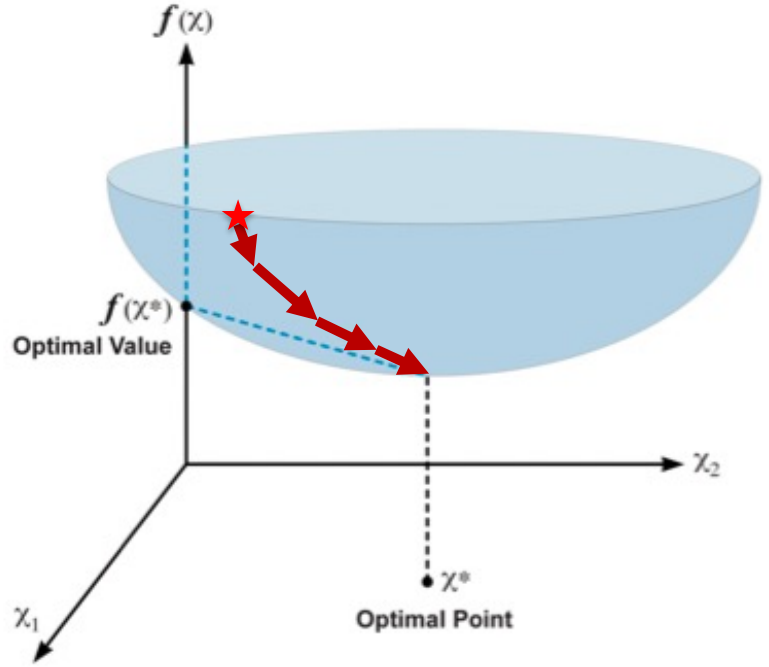
- Overall, for each layer that is batch-normalized, we have $4 \times neurons$ additional parameters

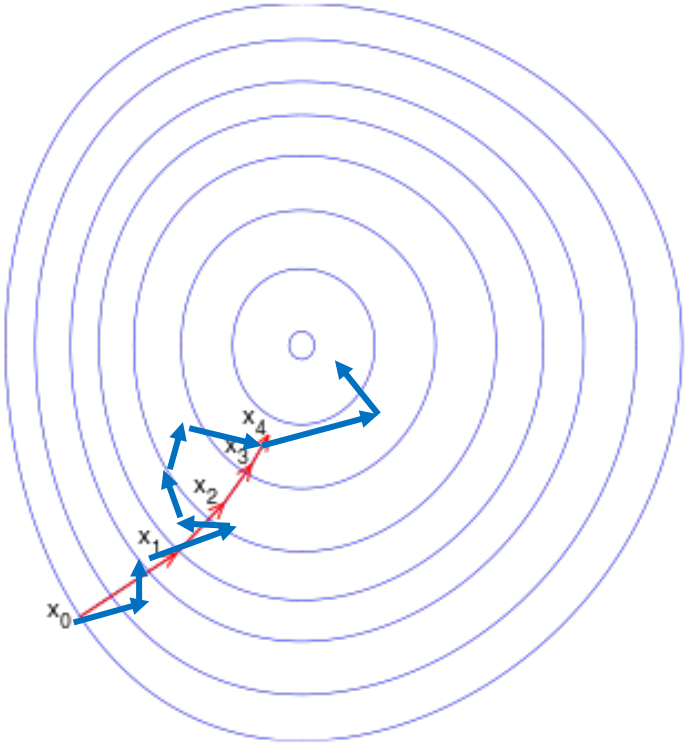
- Batch normalization tends to make the network less sensitive to the initialization, and also helps to regularize it, but adds to the runtime

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Improving learning – slow training

Gradient descent versus stochastic gradient descent





1. Decide a "learning rate" α
2. Start with some parameters $\boldsymbol{\theta}$
3. For a certain number of iterations
   - Compute $J(\boldsymbol{\theta})$
   - Compute $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \left(\frac{1}{n} \sum_{i=1}^{n} L^{(i)} \right)$
   - Let $\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

1. Decide a "learning rate" α
2. Start with some parameters $\boldsymbol{\theta}$
3. For a certain number of iterations
   - Compute $J(\boldsymbol{\theta})$
   - Compute $\nabla_{\boldsymbol{\theta}} L^{(i)}$ for a random (i)
   - Let $\boldsymbol{\theta} := \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} L^{(i)}$

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Mini-batch gradient descent

- Core idea: don’t take the derivative over all observations, but over a bit more than just one

- Trading off between normal gradient descent (“batch gradient descent”) and stochastic gradient descent
  - Batch gradient descent: too slow per iteration
  - Stochastic gradient descent: loses benefits of vectorization

- For small training sets: batch gradient descent
  - Otherwise, use typical batch sizes such as 64, 128, 256, 512, 1024 …

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Gradient descent with momentum

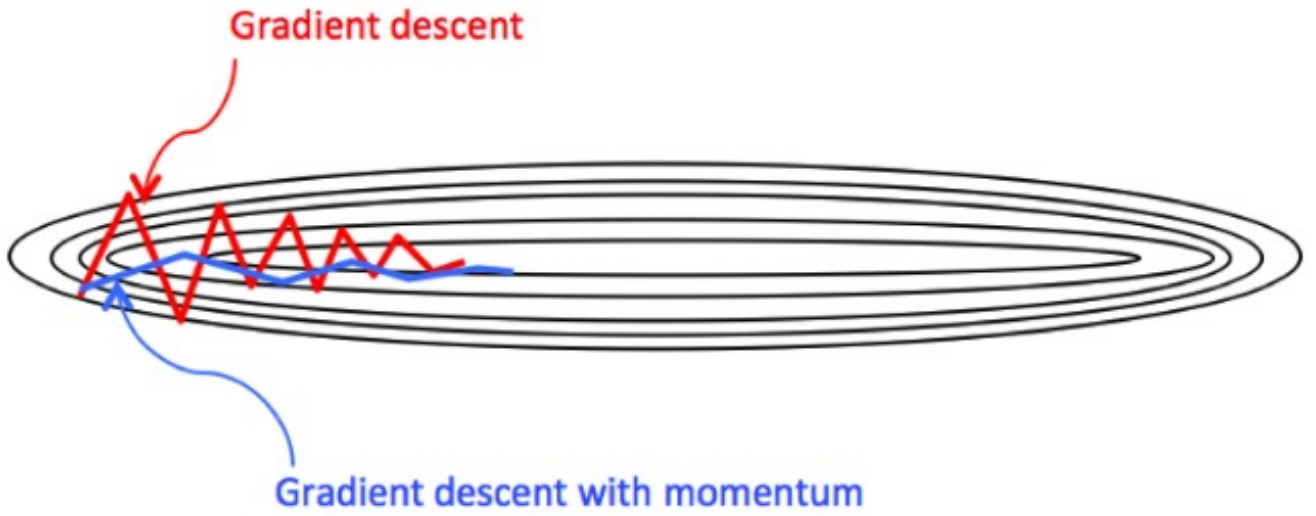
- In gradient descent, we take small, regular steps down a slope
- But if you think of a ball rolling down a slope, it will start slowly, but build up speed (and, thus, momentum) → the "steps" depend not just on the current slope, but on the slope so far!

Gradient descent:

$$\boldsymbol{\theta} \coloneqq \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

Momentum optimization:

$$\boldsymbol{m} \coloneqq \beta \boldsymbol{m} - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$
$$\boldsymbol{\theta} \coloneqq \boldsymbol{\theta} + \boldsymbol{m}$$



- We can also escape faster from plateaus

Source: Trehan

BAYES BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

RMSprop (“Root mean square prop”)

- We normalize the gradient (using the moving average of the square of the gradients)
  - If there is a direction with a lot of oscillation, we penalize the update in this direction
  - If there is a direction with little oscillation, we help along the update in this direction

$$s := \beta s + (1 - \beta)\left(\frac{\partial J}{\partial \theta}\right)^2$$

$$\theta := \theta - \alpha \frac{\frac{\partial J}{\partial \theta}}{\sqrt{s + \varepsilon}}$$

- Used to be the standard algorithm to use until Adam

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Adam ("Adaptive moment estimation")

- Combining momentum and RMSprop

$$m \coloneqq \beta_1 m - (1 - \beta_1)\frac{\partial J}{\partial \theta}$$

$$s \coloneqq \beta_2 s + (1 - \beta_2)\left(\frac{\partial J}{\partial \theta}\right)^2$$

$$\hat{m} \coloneqq \frac{m}{1 - \beta_1^t}$$

$$\hat{s} \coloneqq \frac{s}{1 - \beta_2^t}$$

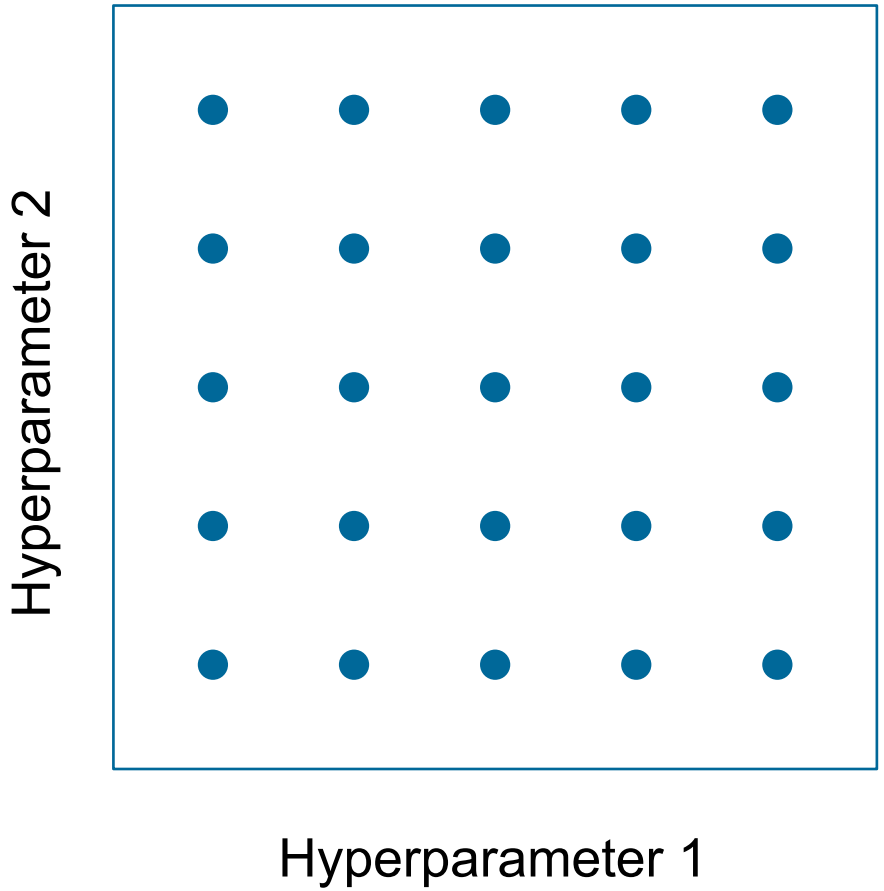
$$\theta \coloneqq \theta + \alpha \frac{\hat{m}}{\sqrt{\hat{s} + \varepsilon}}$$

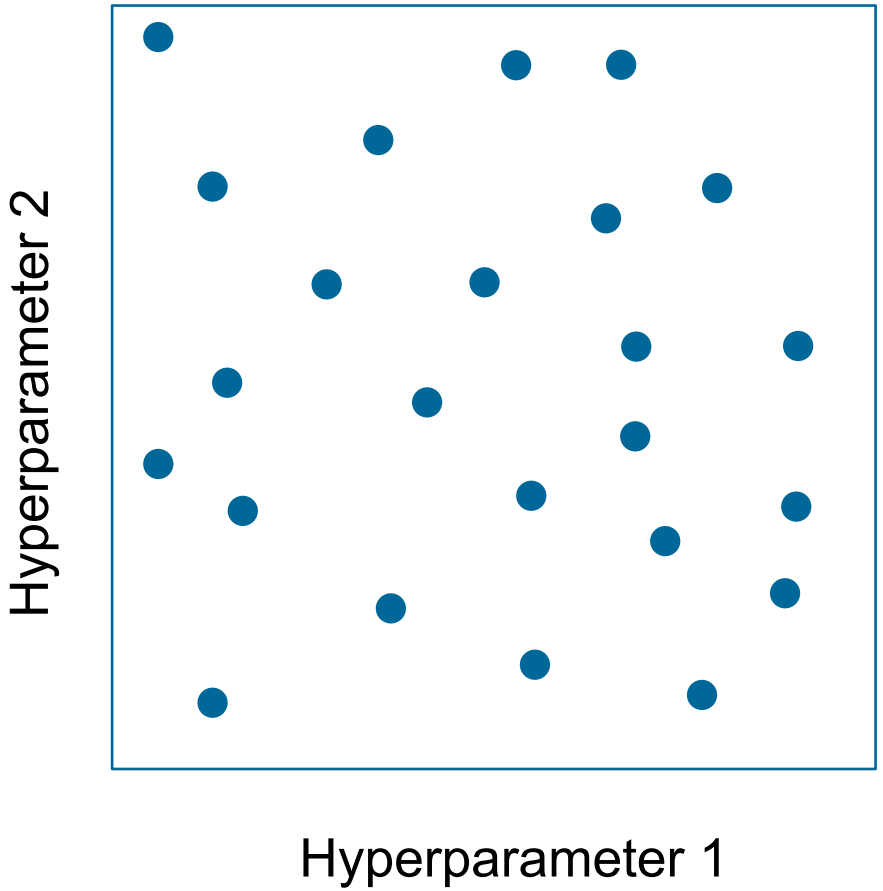
- Like RMSprop, the algorithm is adaptive → less tuning of the learning rate α is required

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Hyperparameter tuning

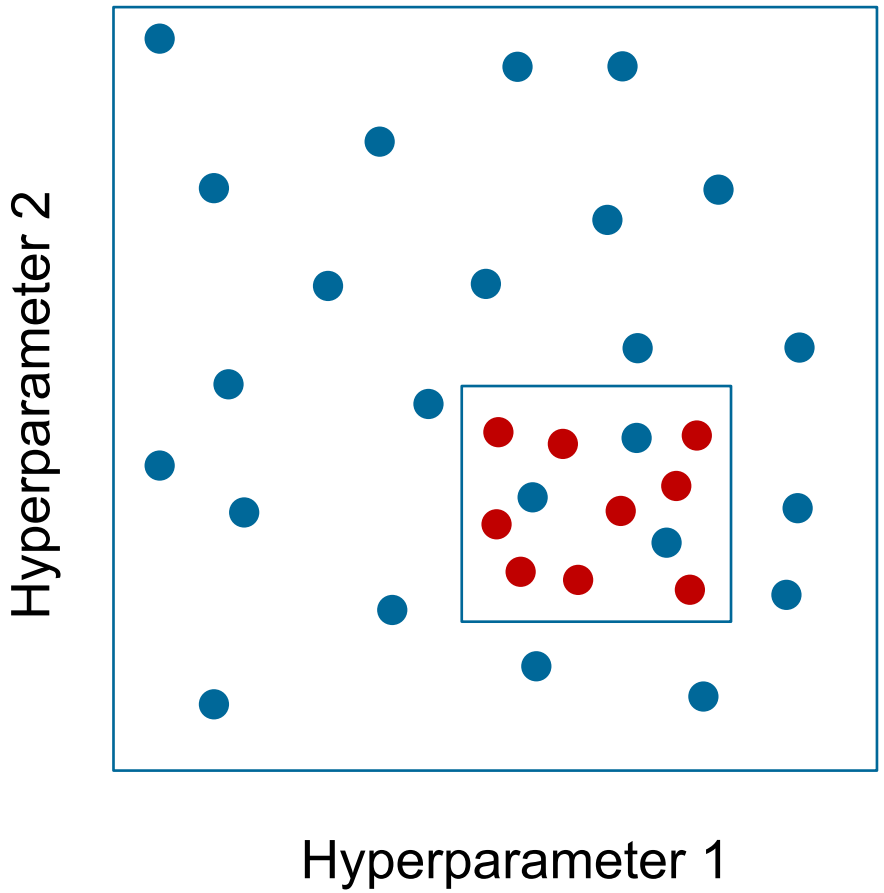
Hyperparameter tuning process rule 1: Choose random combinations





BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Hyperparameter tuning process rule 2: Go from coarse to fine



BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Hyperparameter tuning process rule 3: Pick the right scale

- Say you want to set hyperparameter α in the range $0.001, \ldots, 1$
- You can try out your model 5 times

- The naïve option: uniform distribution between 0.001 and 1
  → $\alpha = np.random.rand(0.001,1)$

- The smarter option: logarithmic spacing
  → $r = -3 \times np.random.rand(0,1)$
  → $\alpha = 10^r$

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Hyperparameter tuning process rule 4: Use purpose-built libraries

- Hyperopt
- Keras Tuner
- Scikit-Optimize
- …

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Hyperparameter tuning process rule 5: Prioritize

A typical (but no way always optimal) prioritization:

1. Learning rate
2. Mini-batch size
3. Number of hidden units (mostly, the same number per layer works just fine – with some exceptions, such as a larger first hidden layer)
4. Number of hidden layers (usually, start with just a few hidden layers, unless you are dealing with complex tasks such as image classification. But then, you usually don't train your own model)
5. Learning rate decay
6. Other algorithm parameters (but the defaults often work fine)

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

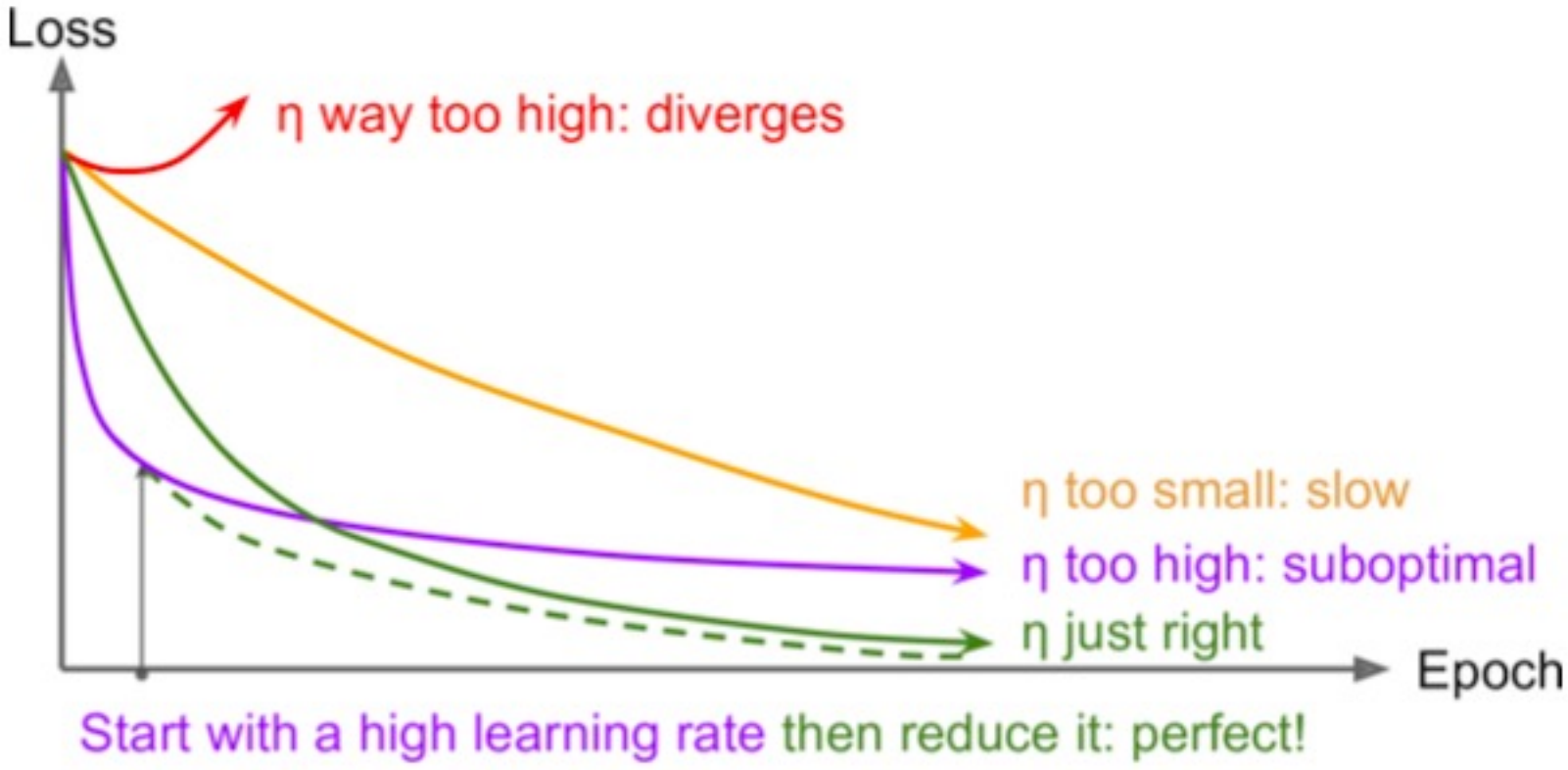
Learning rate scheduling and decay

- Generally, can find a good learning rate by training a bit, then increasing the rate (on a log-scale), then training a bit, …, until the loss goes up again.
  - A decent learning rate is slightly less than that

- Better: adjusting the learning rate during training (with a “learning schedule”)

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Learning rate scheduling and decay



Source: Géron

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Typical learning rate schedules

Power scheduling

- $\alpha_t = \frac{\alpha_0}{1+\frac{t}{s}}$, where t is the epoch and s is the number of epochs until we reach $\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \ldots$
- Can also take a power of the denominator
- Very easy to implement directly within the tf.keras optimization routine

Exponential scheduling

- $\alpha_t = \alpha_0 0.1^{t/s}$, so now we indicate with s the number of epochs until we reach $0.1, 0.01, 0.001, \ldots$
- To use in TensorFlow, need to implement a callback

1cycle scheduling

- Start to increase from α_0 to α_1 (linearly for the first half), then decrease it (again, linearly)

…

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON


See you in class!

Sources

- Bhaskhar, 2021, Introduction to Deep Learning: https://cs229.stanford.edu/syllabus.html
- DeepLearning.AI, n.d.: deeplearning.ai
- Erdem, 2020, DengAI — Data preprocessing: https://towardsdatascience.com/dengai-data-preprocessing-28fc541c9470
- Géron, 2019, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow
- Goodfellow, Bengio, Courville, 2016, The Deep Learning Book: http://www.deeplearningbook.org
- Liang, 2016, Introduction to Deep Learning: https://www.cs.princeton.edu/courses/archive/spring16/cos495/
- Trehan, 2020, Gradient Descent Explained: https://towardsdatascience.com/gradient-descent-explained-9b953fc0d2c