TX00DH43-3001
Introduction to Deep Learning

peter.hjort@metropolia.fi

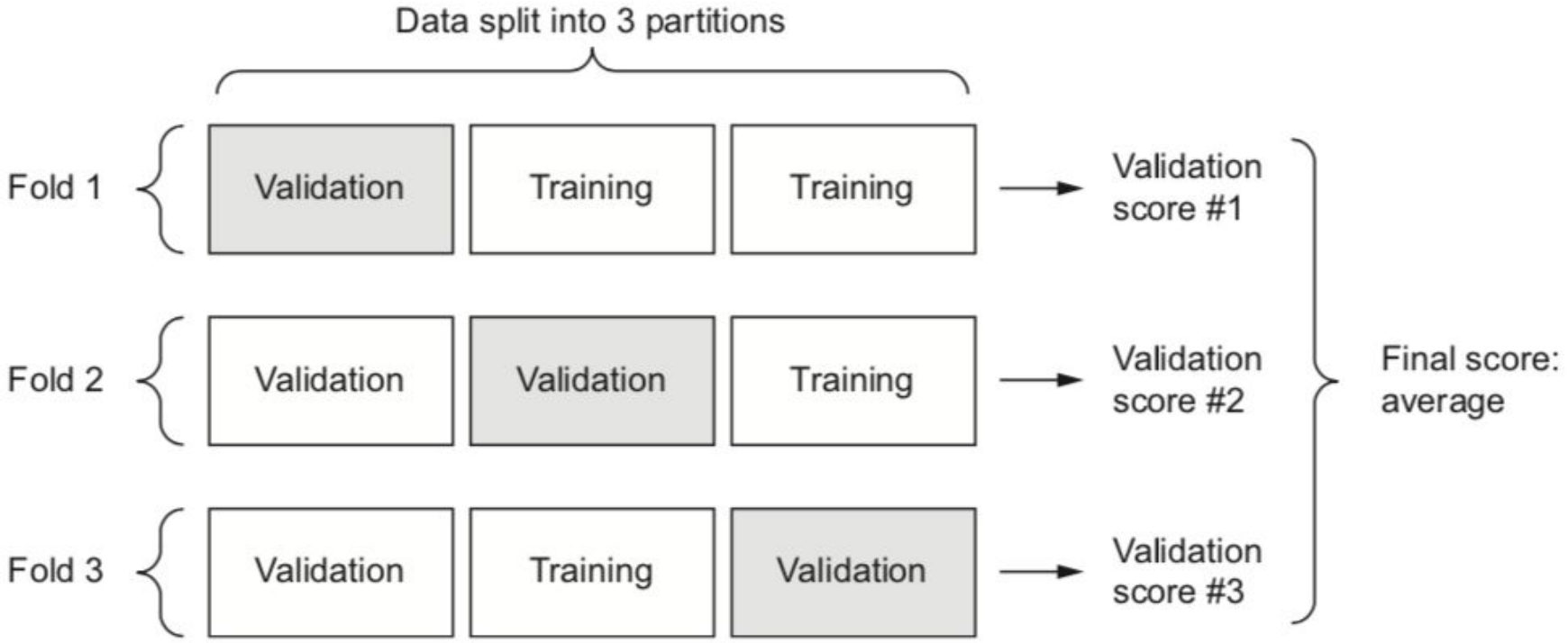
K-fold validation

A way to increase the effective amount of training and validation data (with increased computation cost) → more reliable validation result.

Split union of training **and** validation data into k pieces, use one piece as validation set, and rest k-1 as training set. Cycle through all the data.

Train and validate each combination separately, and compute average over all combinations.

K-fold validation



Model capacity

~ ability of model to fit wide variety of functions.

Hypothesis space: the set of functions to which the solution can belong to. For linear examples this was set of straight lines (or for more dimensions, planes). A larger hypothesis space would be for example polynomials.

Effective capacity: since the optimization algorithm places restrictions on the set of reachable/possible solutions we might not be able to find all functions in hypothesis space.

Under- and overfitting

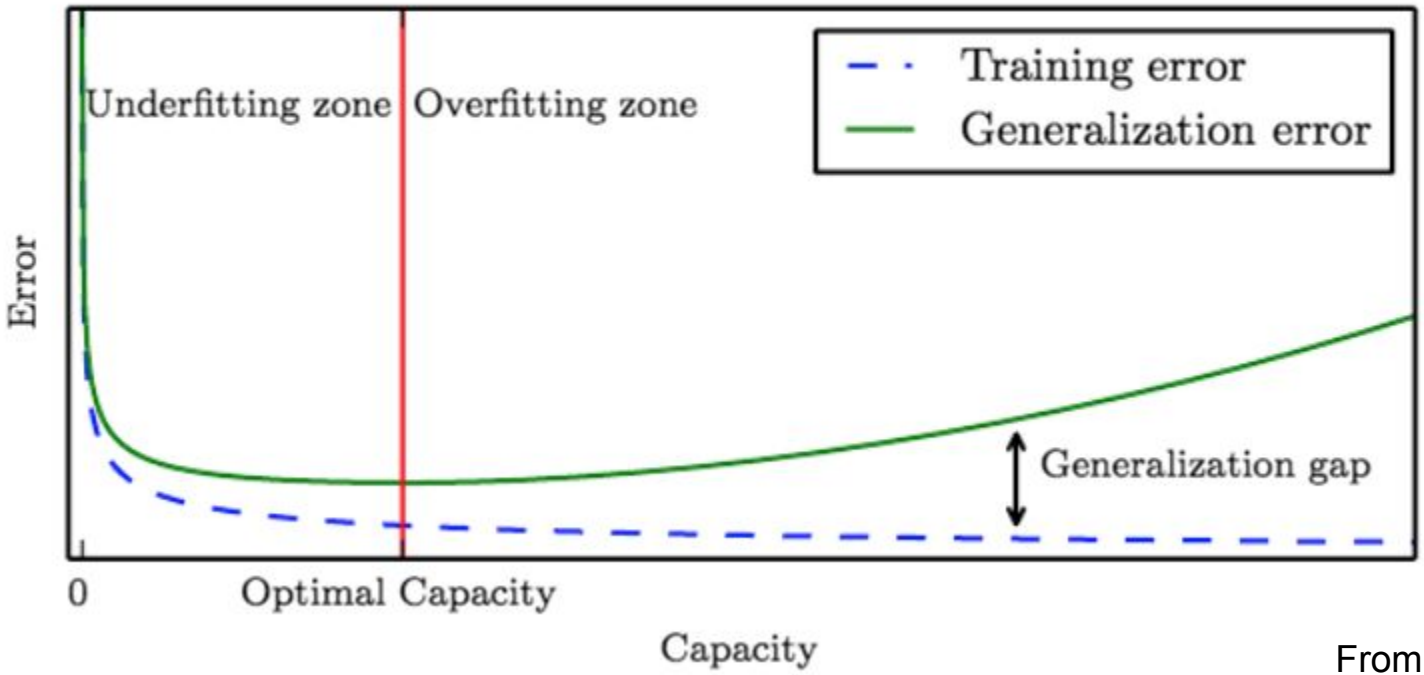
Underfitting: model does not have enough capacity to learn the feature - result mapping. Training accuracy does not come close to baseline.

- Increase model capacity by increasing # of layers / layer width(s) etc.

Overfitting: model specialized too much on training data, and does not **generalize** well: **generalization error** is high. Accuracy on training data is much higher than accuracy on validation data.

- Increase training data amount, decrease model capacity, use regularization or dropout layers, (early stopping)

Capacity / under- and overfitting



From deeplearningbook.org

Managing overfitting - regularization

Penalize the network for too large weights; forces more generalized learning. A good way to reduce overfitting.

Modification of cost function (an example, other alternatives do exist):

$J_{reg}(w) = J(w) + \lambda \cdot \| w \|_2$ (where $\| w \|_2$ is ~ the length of vector w)

Modified cost function will be more reluctant to increase weights because increased weight increases cost function value.

In practise, add regularization parameter to layer specification in Keras.

Managing overfitting - dropout layers

Drop out neurons from a layer with a given probability → create holes in the network. Another good way to reduce overfitting.

Forces network to learn in more “wholesome” way.

In practise introduce a dropout layer to model with a parameter specifying the probability of dropping out a connection.

Note: weights must be rebalanced when some neurons are shut down; Keras does this automatically.

Another note: dropout only happens during training.

Regularization and dropout in Keras

```
model.add(keras.layers.Dense(50, activation='relu',
kernel_regularizer=keras.regularizers.l2(0.01)))
```

L_2 regularizer (square root of sum of squares) with λ = 0.01.

```
model.add(keras.layers.Dense(50,
activation='relu'))
keras.layers.Dropout(0.3)
model.add(keras.layers.Dense(25,
activation='relu'))
```

Drop 3/10 of input units, ie. cut connection.

Forward propagation

Compute output for a given input with current weights in the network. Note that when network is created, it will have random (and close to 0) weights assigned, and bias terms typically = 0.

When the model is used (as part of real application), forward propagation is performed every time a new prediction is needed. (If the model is large this can be computationally expensive). A reason to prefer small models, or a reason to have more processing power in application environment.

Backward (or back-)propagation

Given the result of forward propagation phase, compute loss function value from the result (y_pred) and true result value (y).

Walk through the network in backward direction (from output towards input) and update weights (slightly) based on the loss value. How to decide the amount of correction for each weight?

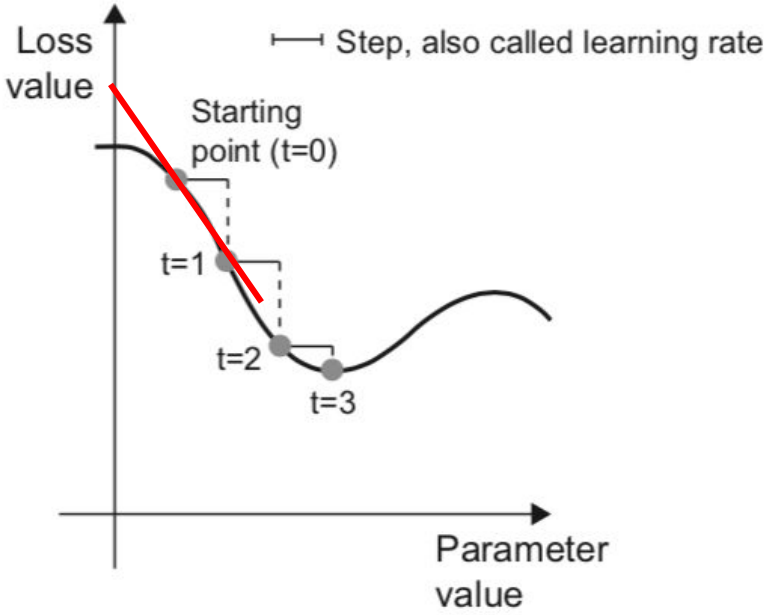
Backpropagation and gradient

Use the **gradient** (~ multi-dimensional derivative) of the cost function to decide corrections to weights. The components of gradient are multiplied with **learning rate** (which is a controllable parameter) and subtracted from current weight values.

Calculation of correction terms proceeds from the output towards the input, using chain rule of derivatives in the process to “allocate” error correction to each weight in each layer. This will lead to navigating towards a minimum of the loss function. Thus the name **gradient descent**.

(Those interested in the math of this, please take a look at for example: https://theclevermachine.wordpress.com/2014/09/06/derivation-error-backpropagation-gradient-descent-for-neural-networks/)

Gradient descent in 1-dimensional case



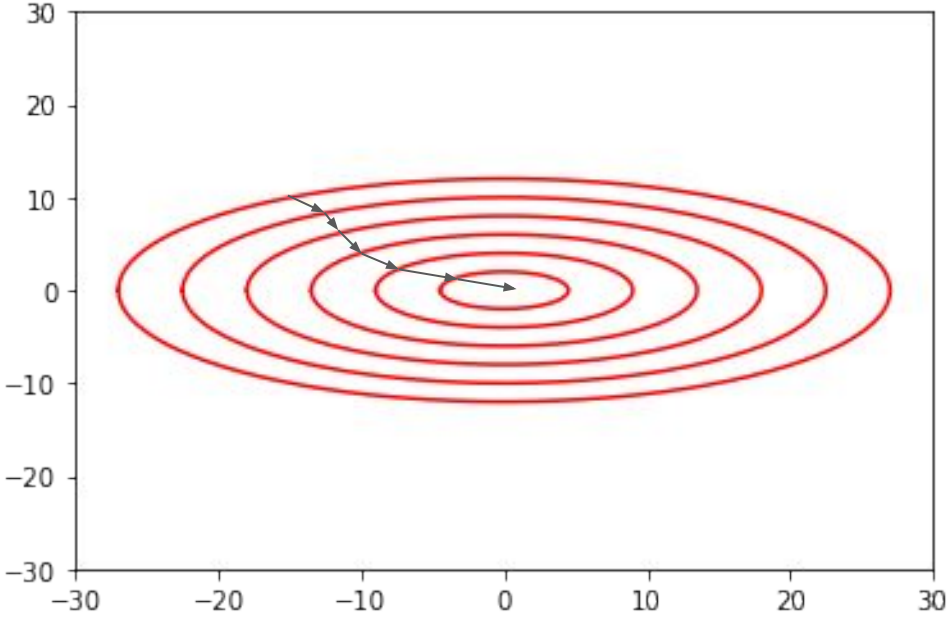
At parameter point w, compute derivative for loss function
dJ/dw = slope of tangent for J.

Next value for w_{next} = w - α · dJ/dw (α is the learning rate)

Slope of tangent < 0 → move towards higher w

Slope of tangent > 0 → move towards lower w

Gradient descent in 2-dimensional case



Look at the cost function diagram “from above” - lines represent equal cost function values. Most smooth descent when the diagram looks like a circle; the more squeezed the ellipse (or in general more complicated surface) becomes the more erratic and varying jinxes the gradient descent makes - vanishing and exploding gradients.

Note on cost function minimum for large ANNs

In the simple cases of linear regression and logistic regression the cost function is convex → when we find minimum, it is guaranteed to global minimum

For large ANNs with non-linear activation functions the cost function will have many (even uncountably many) **local minima**. Their value, however, is *believed* to be sufficiently **close to global minimum** → finding only a local minimum is ok. (This remains an active area of research, however).

A phenomena for higher-dimensional cost functions is that they tend to have **saddle areas** where partial derivatives are 0 but which are far from global minimum. Empirical evidence suggests, however, that gradient descent in practise manages to escape these areas, for example by utilizing higher derivatives in the process.

Optimization algorithms

Stochastic gradient descent (in Keras `sgd`): compute loss for mini-batch of **random** samples, compute gradient (wtr. weights), additionally, may use adaptive learning rate to try to ensure convergence.

Some (very popular) variations of sgd:

RMSProp (Root Mean Square prop(agation); in Keras `rmsprop`): adapt learning rate separately for each parameter and adjust each by dividing with running average

Adam (adaptive moment estimation; in Keras `adam`): adaptive and maintains per-parameter momentum to avoid getting stuck in local optima. Probably the best default choice.

Note: many of the algorithm implementations include ideas from other algorithms in their implementation.

Feature engineering

Preprocess the data samples in a way that allows the network to perform better.

But this was supposed to be needed for shallow models only, deep networks can learn representations themselves, right?

Yes. However, with good feature engineering one can:

- Solve the problem with simpler deep learning solution
- Solve the problem with less data

How to select/design network architecture?

"Non sunt multiplicanda entia sine necessitate"

François Chollet 
@fchollet

Following

The ML research community has long been driven by the need to publish, which results in a stark, sometimes ridiculous bias towards complexity. Remember to ask: "can we do this with k-means and logistic regression?"

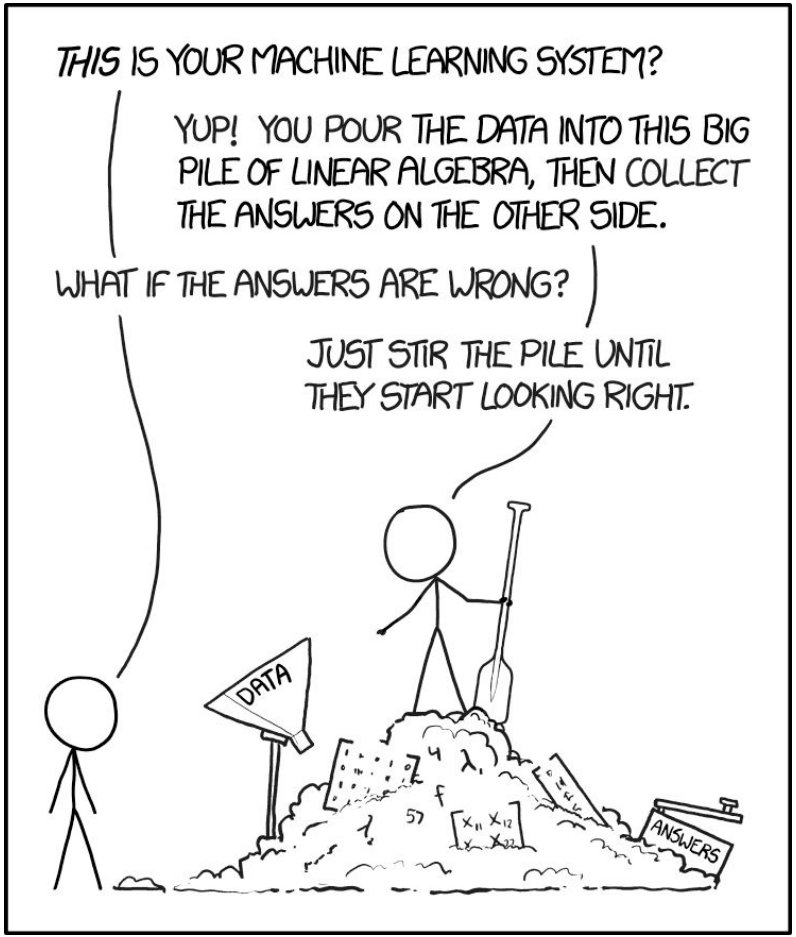
- Use the simplest architecture that gives desired results
- Start with simple network, move to more complex to achieve sufficient training accuracy, even overfitting.
- Bottlenecks are probably a bad idea (but not always)
- Fix overfitting with regularization / dropout / simpler network
- … copy overall architecture from an existing network that solves a close problem
- (Pay attention to the quality of training data)

Why isn’t there a good cookbook for deep learning?

There are some success stories, and successful architectures for solving some problems. There are some principles to follow (select loss function based on problem nature, move from overfitting model to one that generalizes more, etc).

But there is no overall theory. (There are pieces of that).

Read http://www.argmin.net/2018/01/25/optics/ for comparison of optics and deep learning (and a bit more).



https://xkcd.com/1838/

Reading list for exam 5.2

session03.pdf

Chollett: 3.6.4, 4.2, 4.3.2, 4.4, 4.5

Exercise: Boston housing with k-fold validation

Boston housing dataset is relatively small for a deep model. Use k-fold validation, and see if you achieve better results than without it. (Check Chollet chapter 3.6.4 for k-fold validation implementation).

Exercise: regularization vs. dropout fashion-MNIST

Create (or reuse, remember reference) fashion-MNIST model that **overfits**. Study the behaviour when you use a) regularization and b) dropout layers to reduce overfit.

Exercise*: AlphaGo Zero and it's critique

Read the article "Mastering the game of Go without human knowledge" published in Nature (access via https://deepmind.com/research/publications/mastering-game-go-without-human-knowledge/) and critical comment on it at https://medium.com/@GaryMarcus/in-defense-of-skepticism-about-deep-learning-6e8bfd5ae0f1. Write an essay of 300-400 words based on these writings. In the essay, pick your side and defend your point of view.

You will get 0-5 points that will 1) be added to your 3rd worst exam score, or if exam points for 3rd worst exam reach 10, 2) rest to your exercise score, or if you have full points there, 3) rest to your reflection score.

Note: **this is not a mandatory exercise**.