Learning in Neural Networks

Anders Krogh
Center for Health Data Science
University of Copenhagen

Loss function

- We used the **squared error** to quantify the error of a neural network

$$E(w) = \sum_i (f_w(\boldsymbol{x}_i) - t_i)^2$$

- This is also called the **loss function** or cost function
- The choice of loss function depends on the type of problem
- Squared error is often used for regression problems

Regression: predict (continuous) variables *y* from other variables *x*

Maximum likelihood estimation

- Many loss functions correspond to a negative log likelihood:

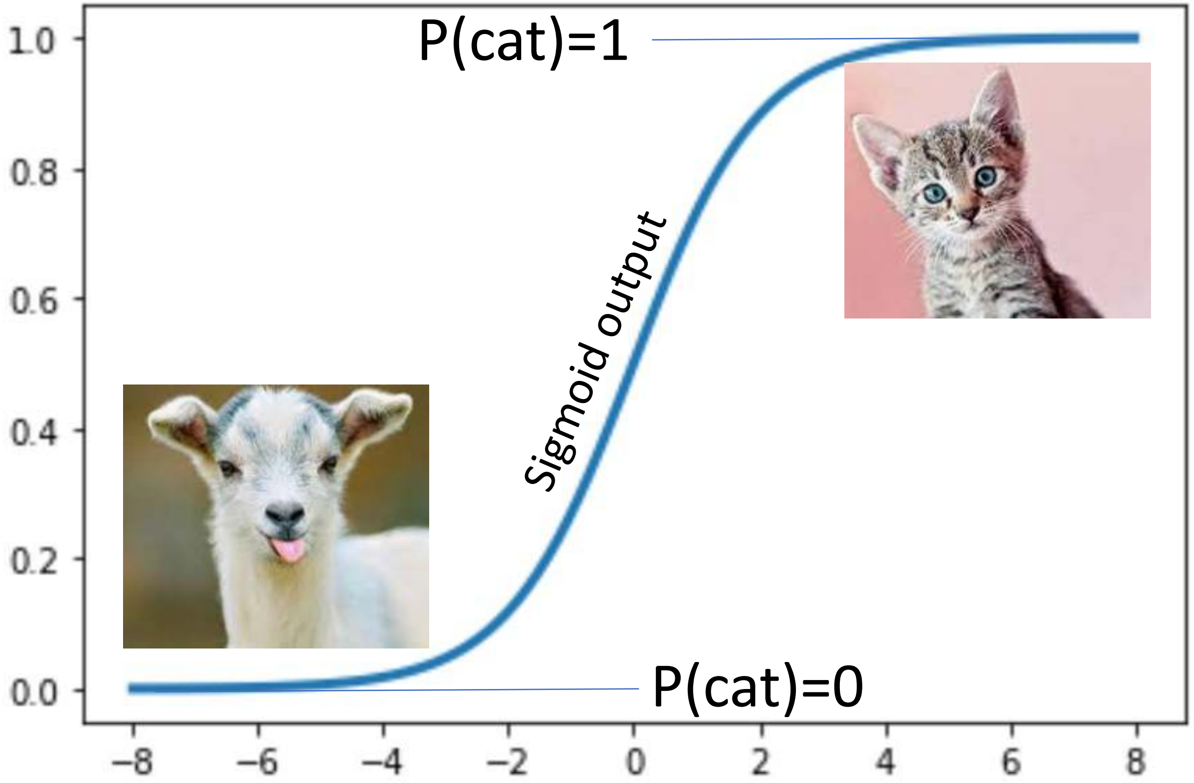
Loss = –log P(data)

- The higher the probability of your data, the better
- Squared loss corresponds to a model with normally distributed errors

Classification

- When there are two classes:
  - Use probabilities
  - Change loss function $E(w)$

- Maximum likelihood leads to the binary cross entropy loss:



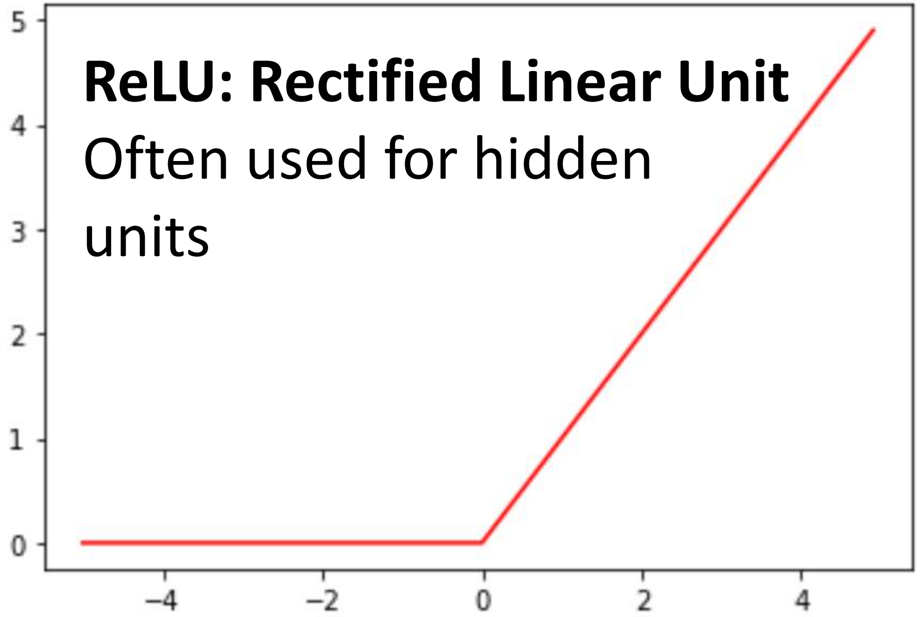
$$E(w) = -\sum_i [t_i \log f_w(\boldsymbol{x}_i) + (1 - t_i) \log (1 - f_w(\boldsymbol{x}_i))]$$

- If there are no hidden units, it is the same as logistic regression

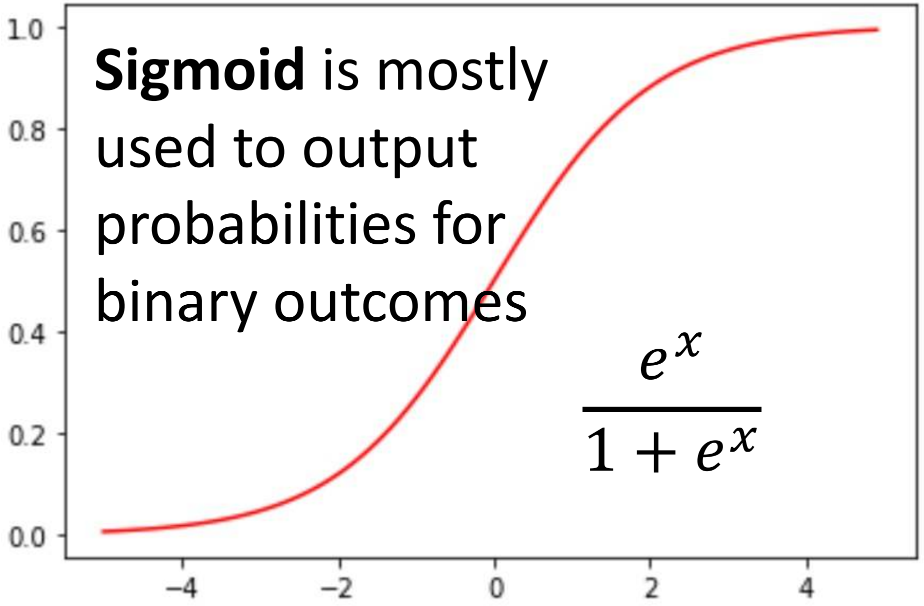
- You can multiple outputs and add the terms

Activation functions

Linear (or no activation)
Used for unbounded continues values (temperature, distance, …)

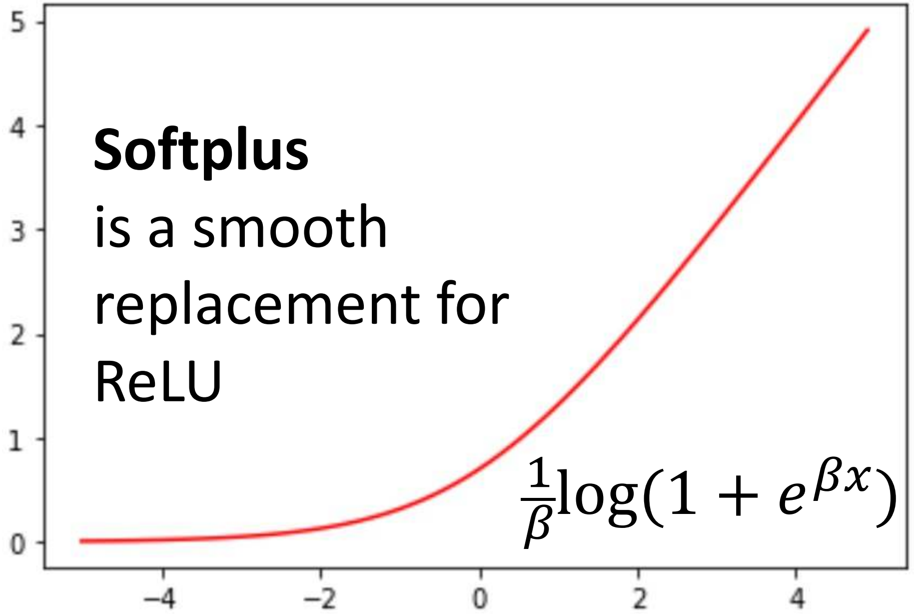


ReLU: Rectified Linear Unit
Often used for hidden units



Sigmoid is mostly used to output probabilities for binary outcomes

$$\frac{e^x}{1+e^x}$$

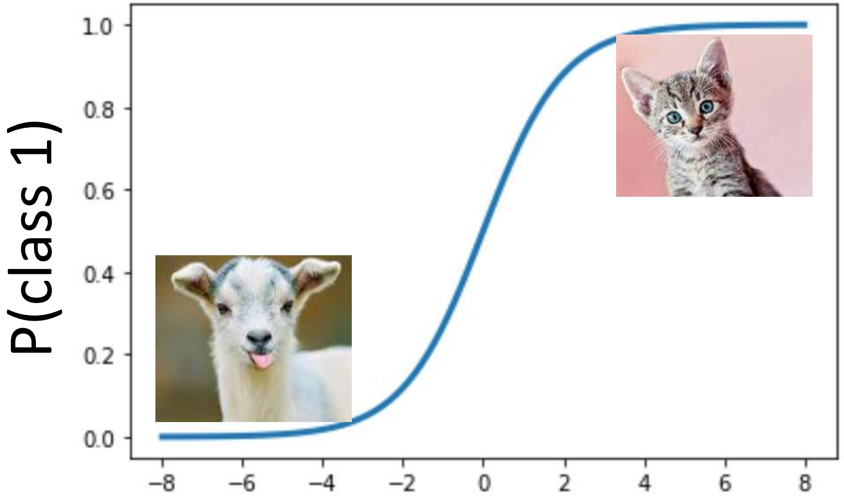


Softplus
is a smooth replacement for ReLU

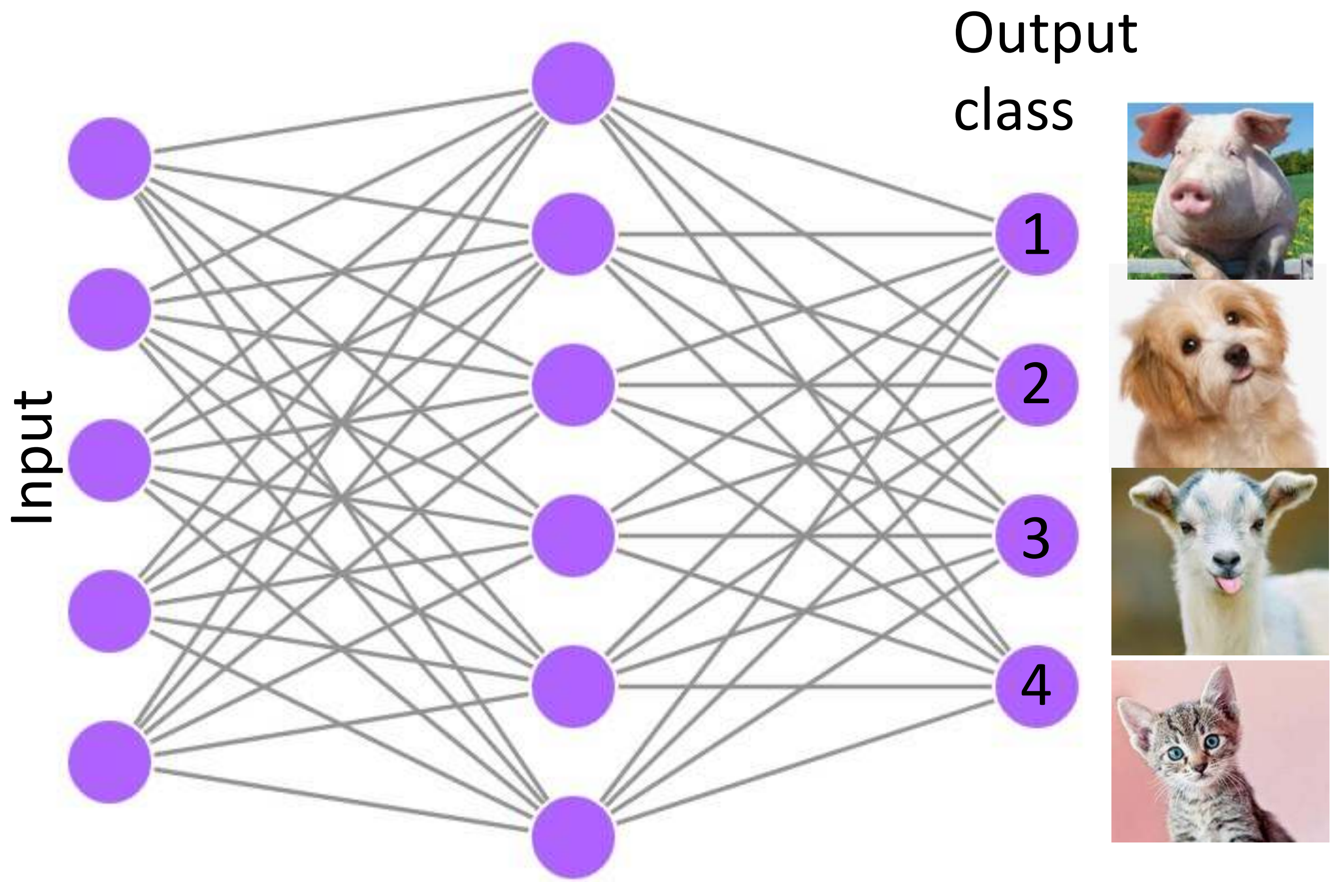
$$\frac{1}{\beta}\log(1+e^{\beta x})$$

What if we have multiple classes?

Sigmoid for two classes



P(class 0) = 1–P(class 1)

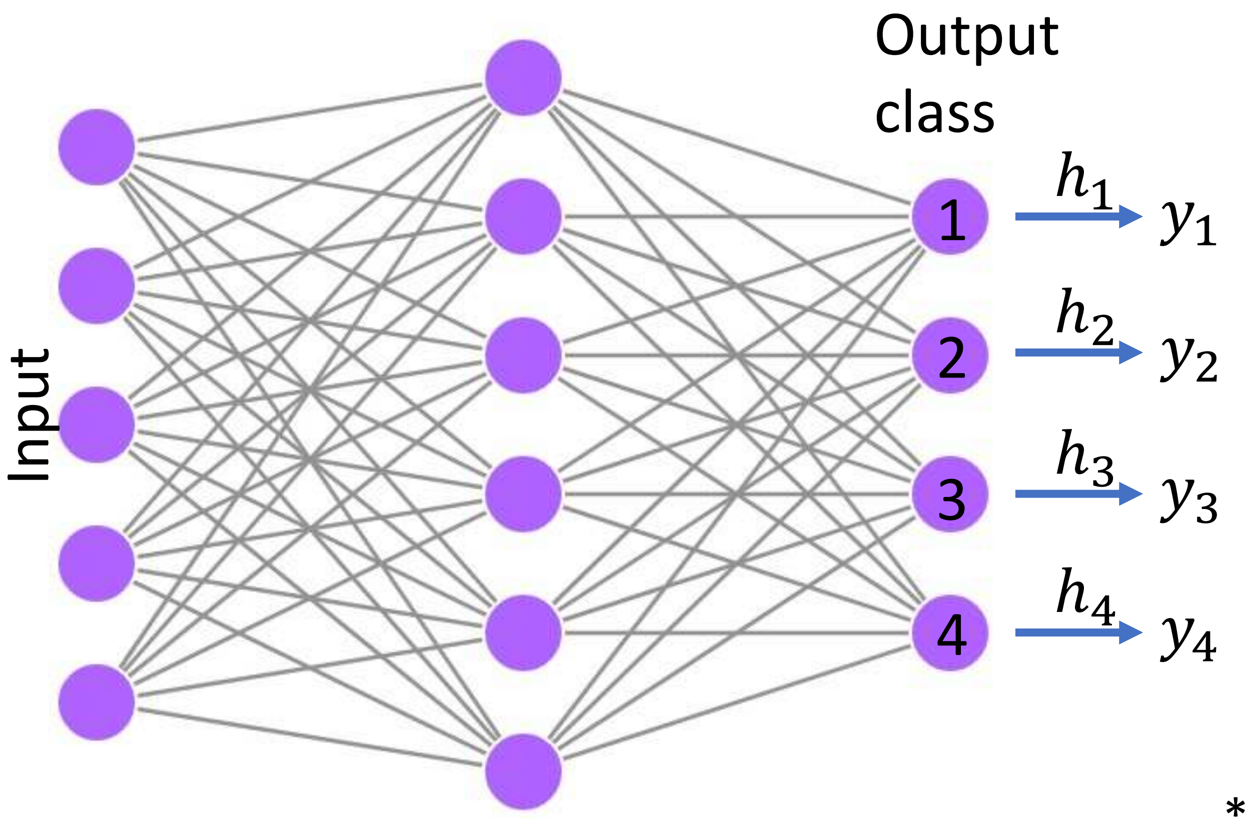


We would like the network to output the probability of each class

Neural net drawing made at http://alexlenail.me/NN-SVG/index.html

Multiple classes: Softmax

Use the softmax function to ensure probabilities sum to one



Weighted sum for the last layer is called h_i for class i

Output for class i:

$$y_i = \mathrm{softmax}(h_i) = \frac{e^{h_i}}{\sum_{j=1}^{n} e^{h_j}}$$

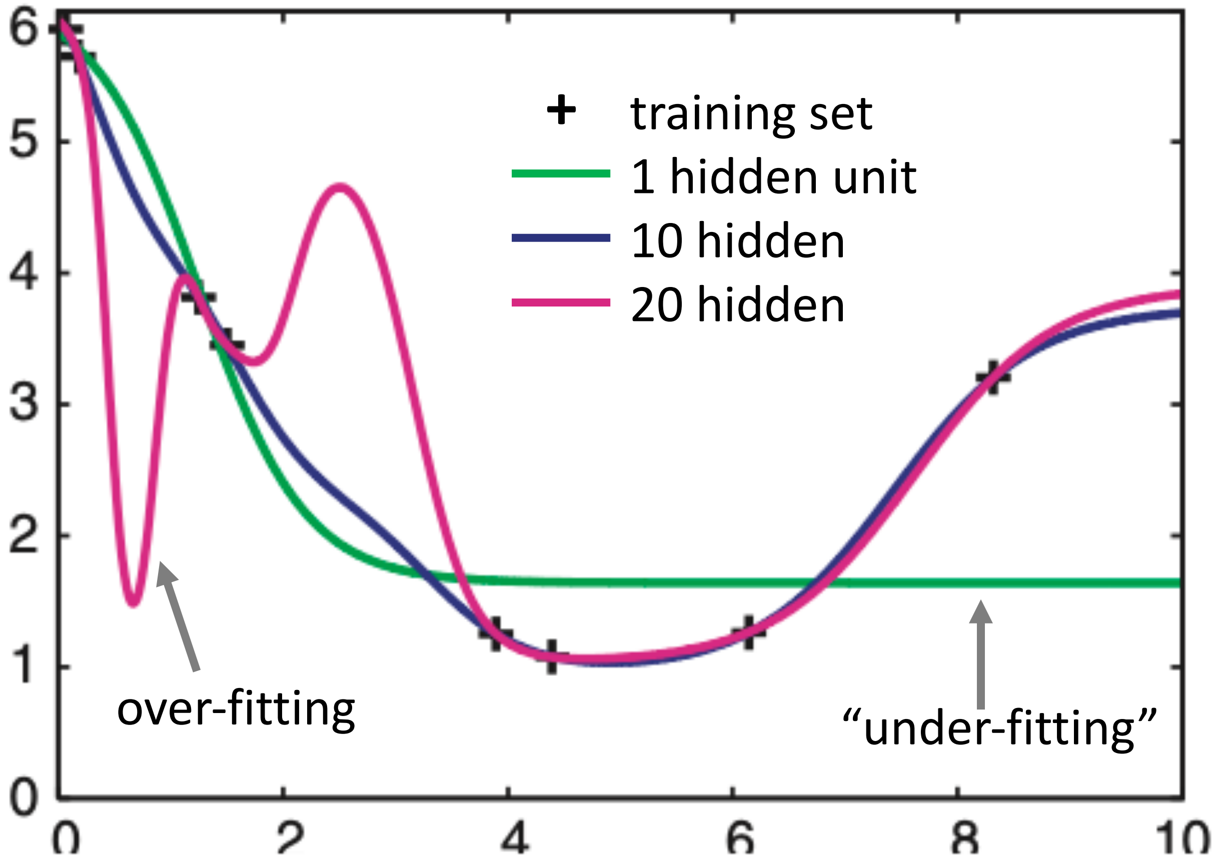
- **Loss function**: cross entropy *
- In pytorch, the softmax is built into the loss function: it takes h_i instead of y_i.

*) Cross entropy loss is similar to binary case.

$E(w) = -\sum_i t_i \log y_i$ where target t_i is 1 or 0

Over-fitting and generalization

- Many parameters and few training data leads to over-fitting
- If the network over-fits, it cannot generalize
- To generalize means to be able to predict on unseen (test) data



From A Krogh (2008) Nat. Biotech. 26, p. 195

Over-fitting and generalization

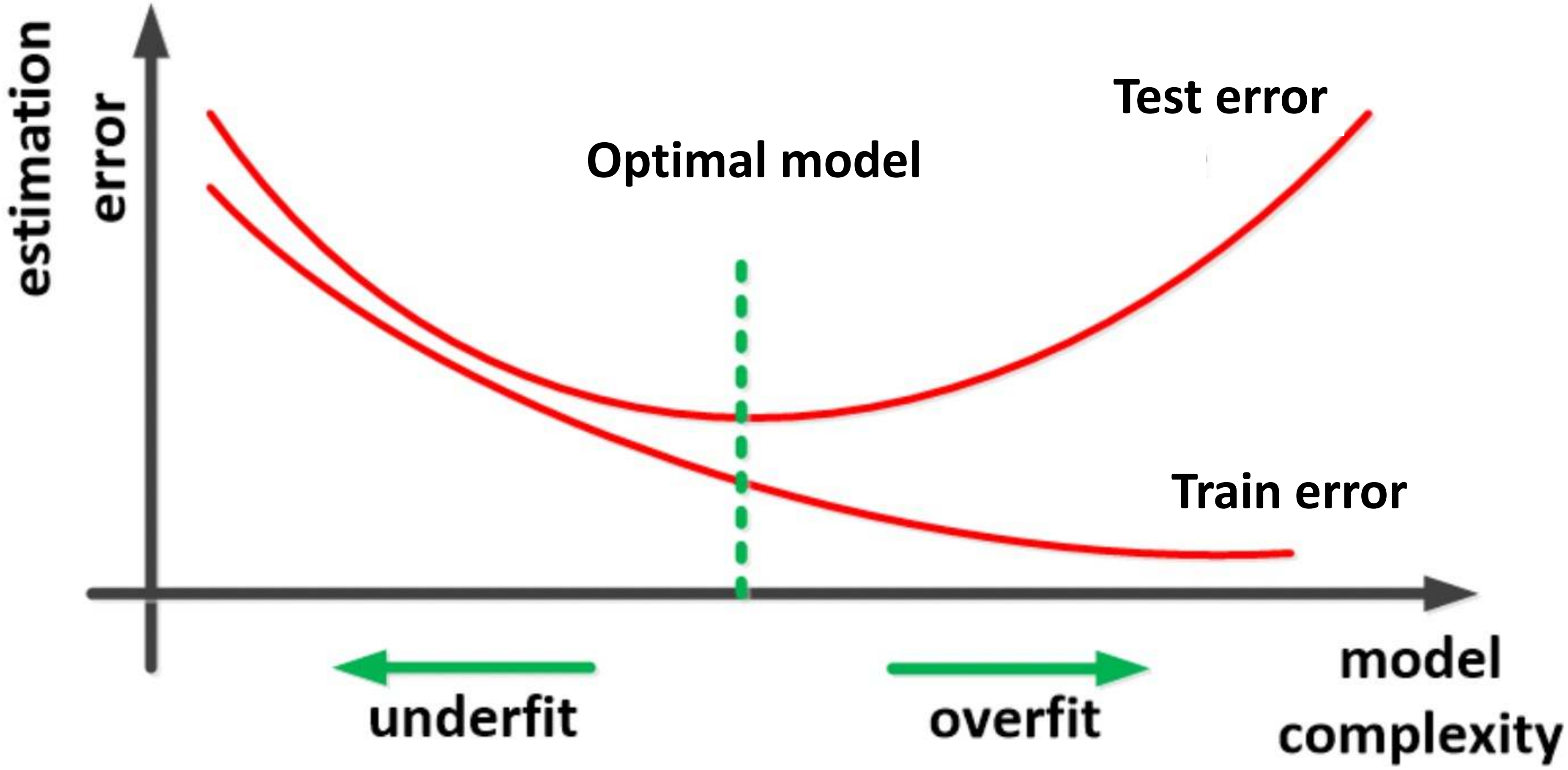


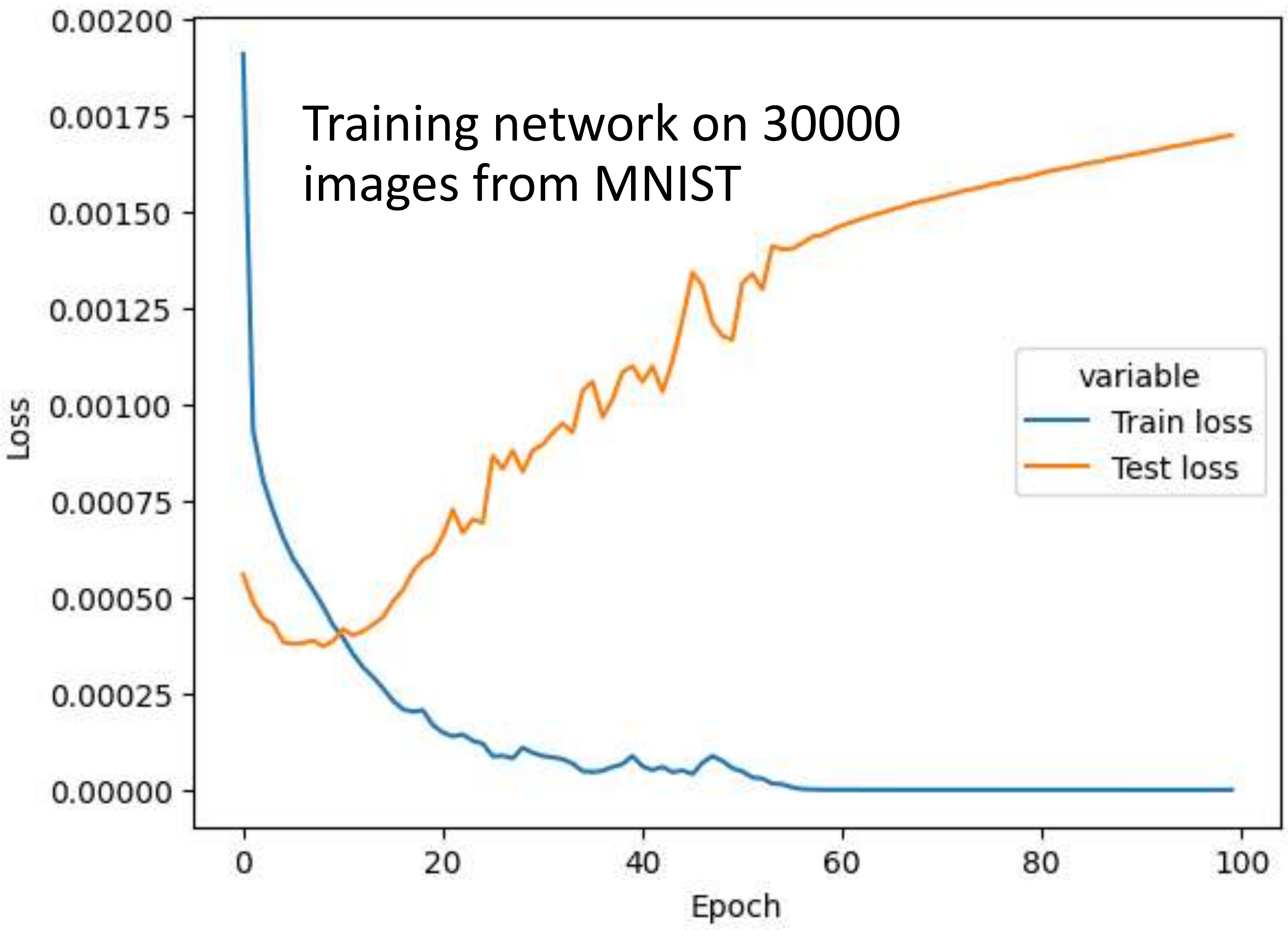
Figure adapted from Ghojogh & Crowley (2019). https://arxiv.org/abs/1905.12787

Over-fitting

If the test error (loss) is larger than the training error, there is over-fitting

A typical sign of over-fitting:

Test error starts to grow while training error is still decreasing



The famous MNIST dataset

Image from https://en.wikipedia.org/wiki/MNIST_database

Dealing with over-fitting

The network size can be decreased if it over-fits (e.g. fewer hidden units)

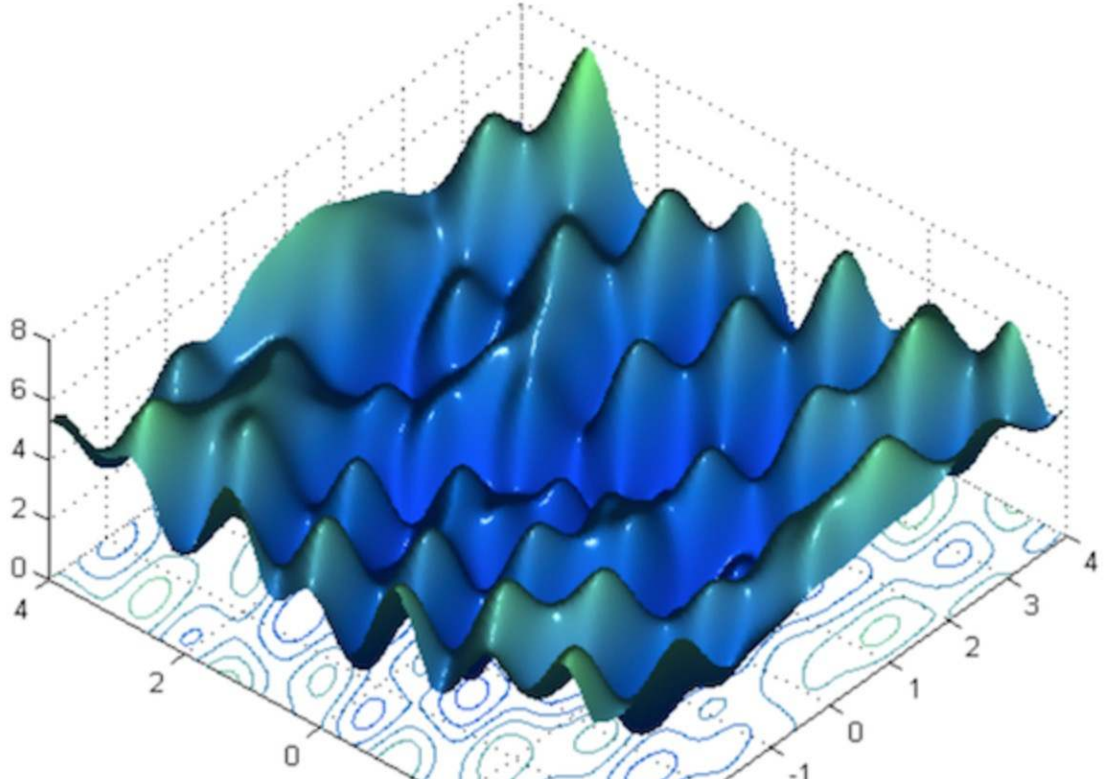
A weight decay* can mitigate over-fitting (other similar regularization techniques exists)

Early stopping: Chose the network with lowest validation error

Drop-out: A method that randomly removes hidden units. Increases robustness

Weight decay: a term λw is subtracted from a weight w in each iteration. λ is normally small, 10^{-2} to 10^{-6}

Local minima of the loss



For complex networks the loss has multiple local minima

Plain gradient descent does not work well

Stochastic gradient descent use "mini batches"

- The gradient is calculated over a random sample of a certain size – the batch size
- For each cycle through the training set (epoch), the network is updated many times instead of just one
- Because of the randomness it can better escape local minima and has turned out to be much more efficient

There are many other "tricks". Many of these are combined in the popular optimizer called Adam

Figure copied from: https://towardsdatascience.com/neural-network-optimization-7ca72d4db3e0

The optimizer

- In plain stochastic gradient descent (torch.optim.SGD) you need to set parameters (learning rate and momentum)
- The Adam optimizer is usually a better choice
  - It automatically adapts the learning rate and momentum in clever ways
  - It is based on SGD and uses mini-batches
  - you can set a weight decay

Example of code using the Adam optimizer:

```
optimizer = torch.optim.Adam(nn.parameters())
for epoch in range(nepochs):
  for x,t in train_loader:
    optimizer.zero_grad()
    y = nn(x)
    loss = lossfunc(y,t)
    loss.backward()
    optimizer.step()
```

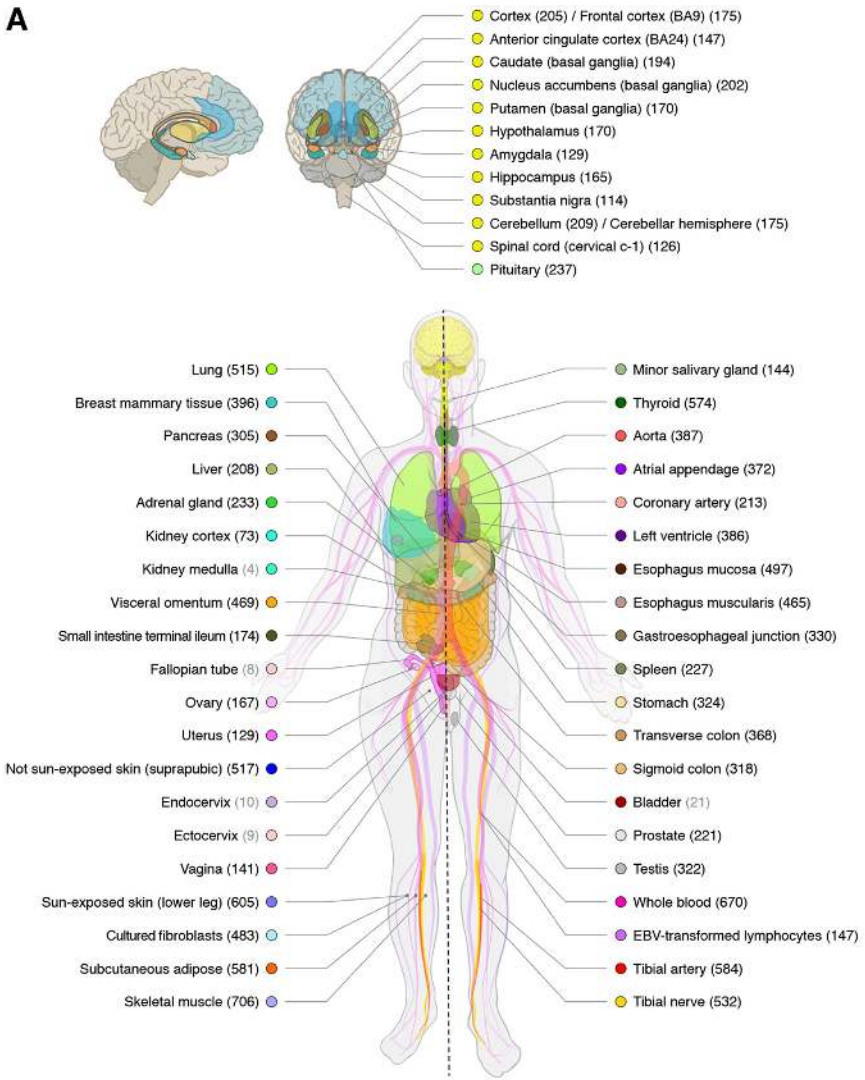
You can set parameters in Adam, such as

- learning rate (e.g. “lr=1.e-4”)
- “weight_decay=1.e-5”

All the choices you have to make …

- There are many parameters you can vary in a Neural Network.
- It is a good idea to make an initial “grid search” where you systematically test performance by varying
  - the number of hidden layers and their size
  - other parameters one by one
- This is sometimes done on a reduced data set and or with quite few iterations
- There are also packages that can help with optimizing parameters

Exercise with gene expression data



- RNA-seq is a great use-case for Machine Learning algorithms:
  - High dimensional data
  - Many cellular pathways are highly correlated
  - Tissue specific
  - Big datasets available (GTex)
- The exercise:
  - Can a neural network capture the information encoded on gene expression and detect tissues?
  - In the exercise we will build a neural network that learns the gene expression profile and is able to guess the tissue of origin

Figure adapted from the Gtex Consortium (2020). https://pubmed.ncbi.nlm.nih.gov/32913098/