Why does my network fail to learn?

Monthly DESY ML/DL seminar

30.09.2019

Philipp Heuser (DESY-IT)

HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

Outlook ML seminars@DESY

28.10. Kai Krajsek (FZJ): **Hyperparamter & Hyperparameter Optimisation**

25.11. Matthias Heinrich (Uni Lübeck): **Multimodal Learning**

27.01. Alexandr Ignatenko (DESY): **Yolo&Co Object Recognition**

NN Dirk Krücker: **Graph Networks**

NN Dirk Krücker: **Introduction to deep learning using Keras**

Mailing List

Mailinglist for ML activities at DESY / Announcements of ML related events.

desy-ml@desy.de

Please sign up! Future seminar announcements will be done on this list only!

https://lists.desy.de/

Review ML seminars@DESY

Introduction to DESY ML infrastructure (Frank Schlünzen)

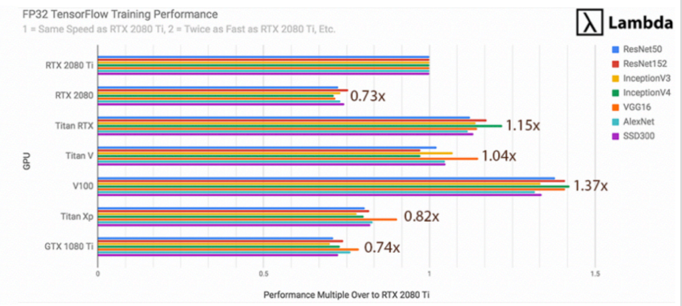
GPU Resources in the Maxwell Cluster

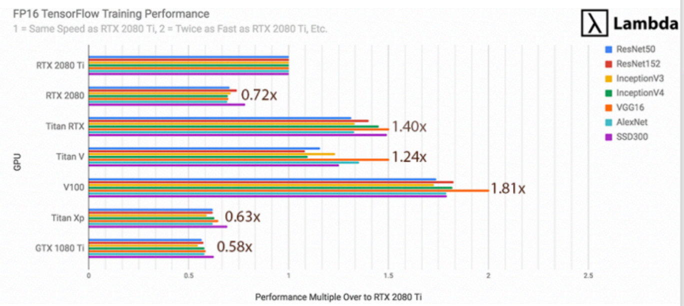
Compute Nodes

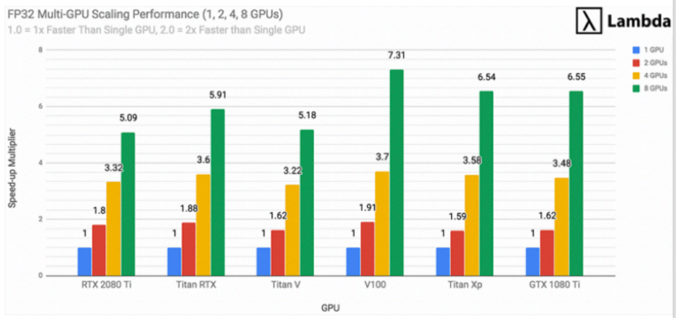
	GPU/node	# hosts	FP64 TFlops	GPU Memory	CPU Memory
K20 (*2)	2	6		6GB	256GB
K40 (*1)	1	5		12GB	512GB
K40 (*2)	2	1		12GB	256GB
P100 (*1)	1	59	5.3	16GB	256-768GB
P100 (*2)	2	6	5.3	16GB	512GB
P100 (*4)	4	1	5.3	16GB	512GB
V100 (*1)	1	10	7.8	32GB	384GB
V100 (*2)	2	3	7.8	32GB	384GB
V100 (*6)	6	1	7.8	32GB	1.500GB
	114	92			

Interactive Nodes

	GPU/node	# hosts	FP64 TFlops	GPU Memory	CPU Memory
Quadro M6000	2	2		12GB	256GB
Quadro M6000	4	3		12GB	256GB
GTX 1080 Ti	2	2	0.3	11GB	384GB
GTX 2080 Ti	2	4	0.4	11GB	384GB
GTX 2080 Ti	4	2	0.4	11GB	384GB
	36	13			





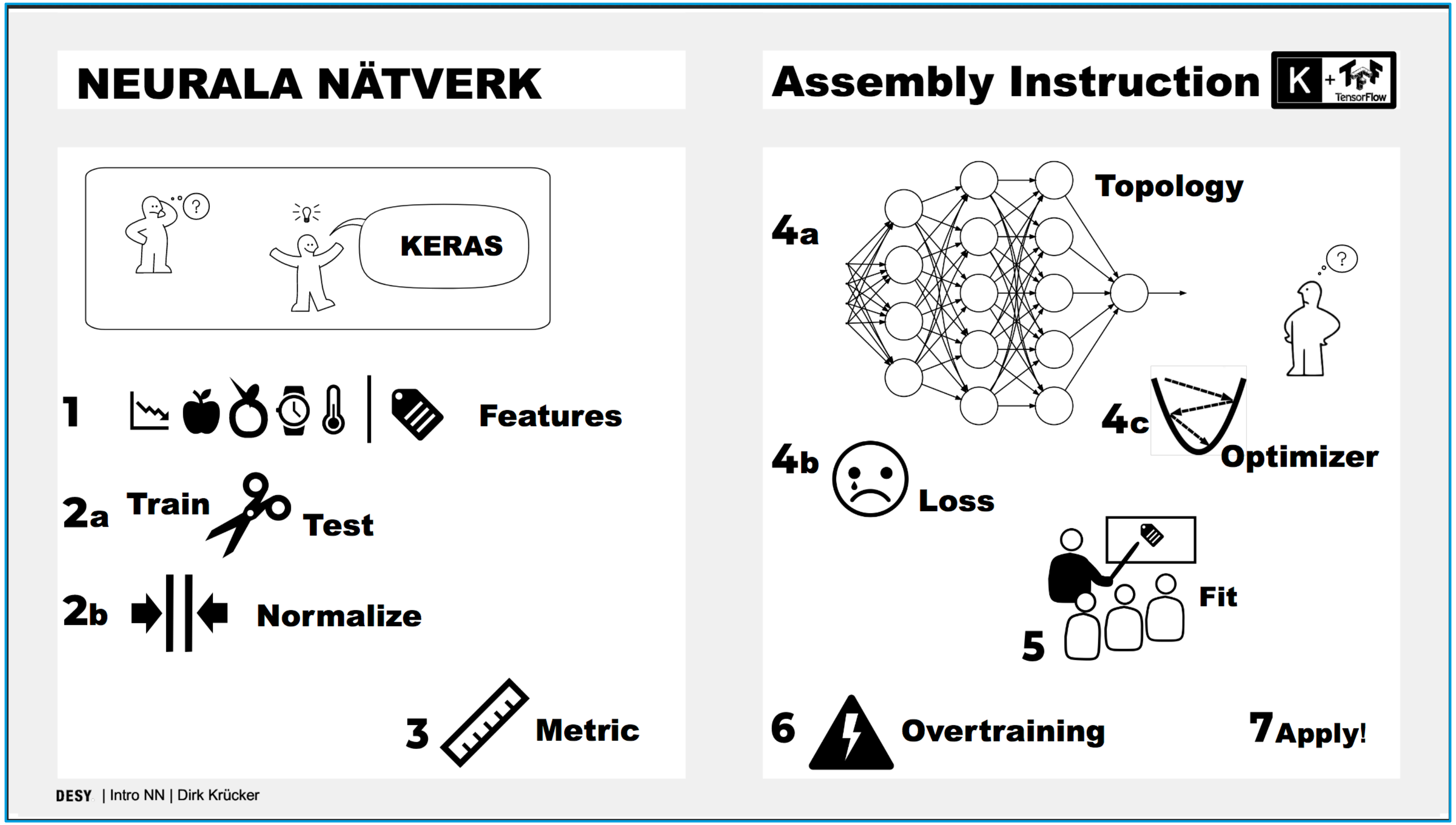


Sources:
https://lambdalabs.com/blog/2080-ti-deep-learning-benchmarks/
https://www.microway.com/knowledge-center-articles/comparison-of-nvidia-geforce-gpus-and-nvidia-tesla-gpus/

https://confluence.desy.de/display/IS/Maxwell
https://indico.desy.de/indico/event/22189/material/slides/

Review ML seminars@DESY

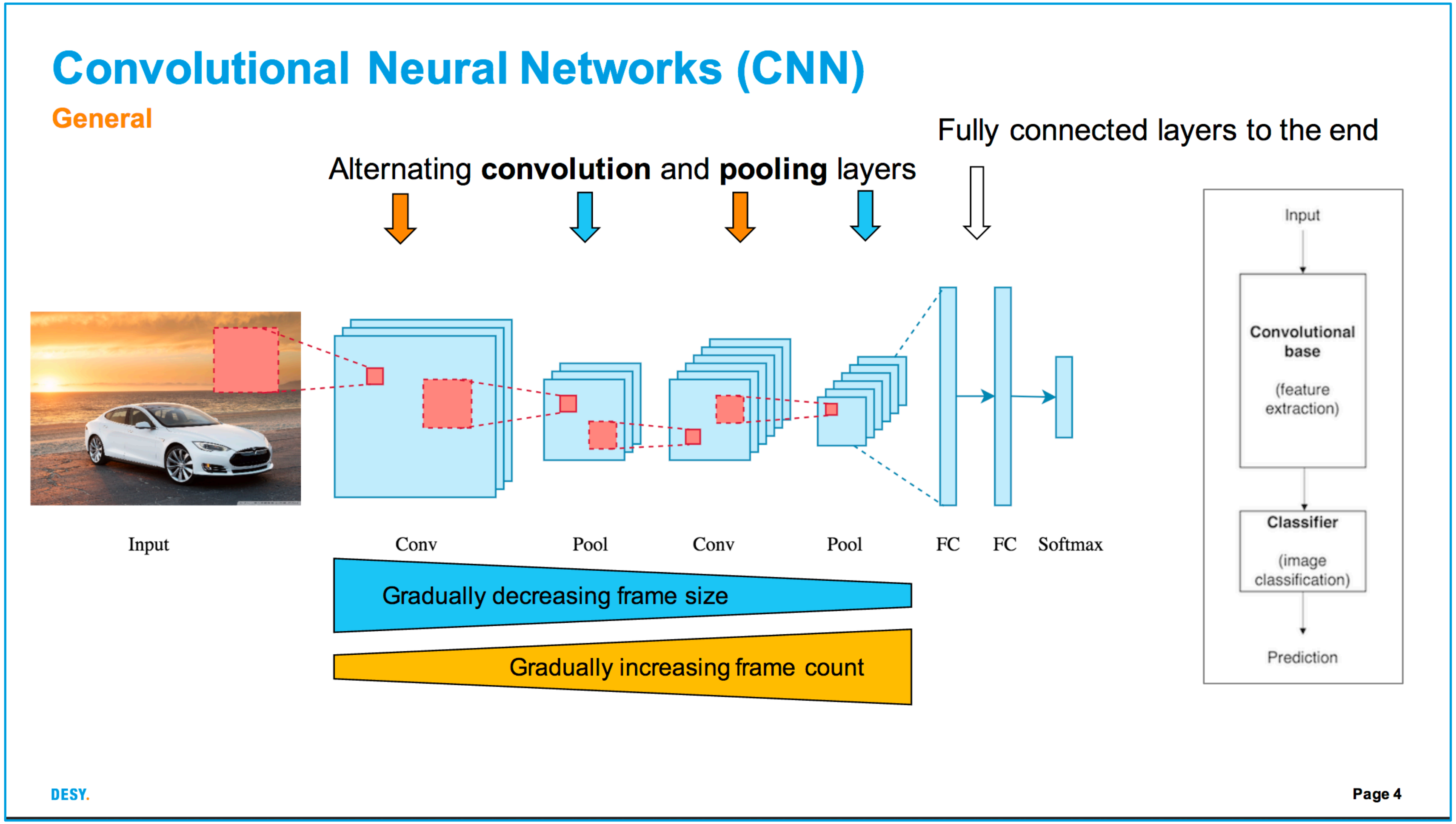
A Gentle Introduction to Neural Networks and Deep Learning (Dirk Krücker)



https://indico.desy.de/indico/event/22190/material/slides/0.pdf

Review ML seminars@DESY

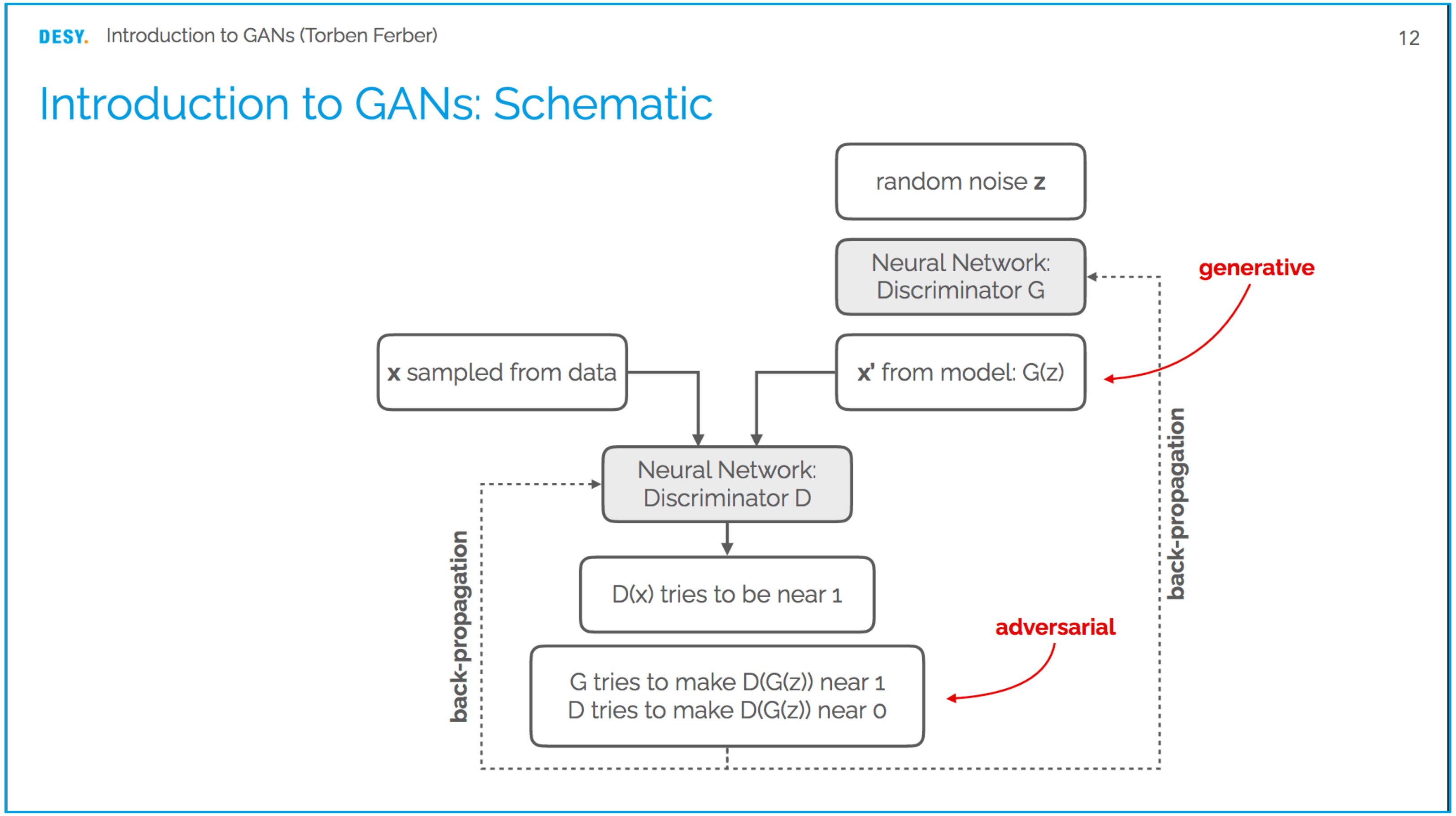
Gentle introduction to Convolutional Neural Networks (Philipp Heuser)



https://indico.desy.de/indico/event/22191/material/slides/0.pdf

Review ML seminars@DESY

Introduction to Generative Adversarial Networks (GANs) (Torben Ferber)



https://indico.desy.de/indico/event/22192/material/slides/0.pdf

9 Reasons why your machine learning project will fail
https://www.kdnuggets.com/2018/07/why-machine-learning-project-fail.html

20 Tips, Tricks and Techniques That You Can Use To Fight Overfitting and Get Better Generalization
https://machinelearningmastery.com/improve-deep-learning-performance/

Improving the Performance of a Neural Network
https://towardsdatascience.com/how-to-increase-the-accuracy-of-a-neural-network-9f5d1c6f407d

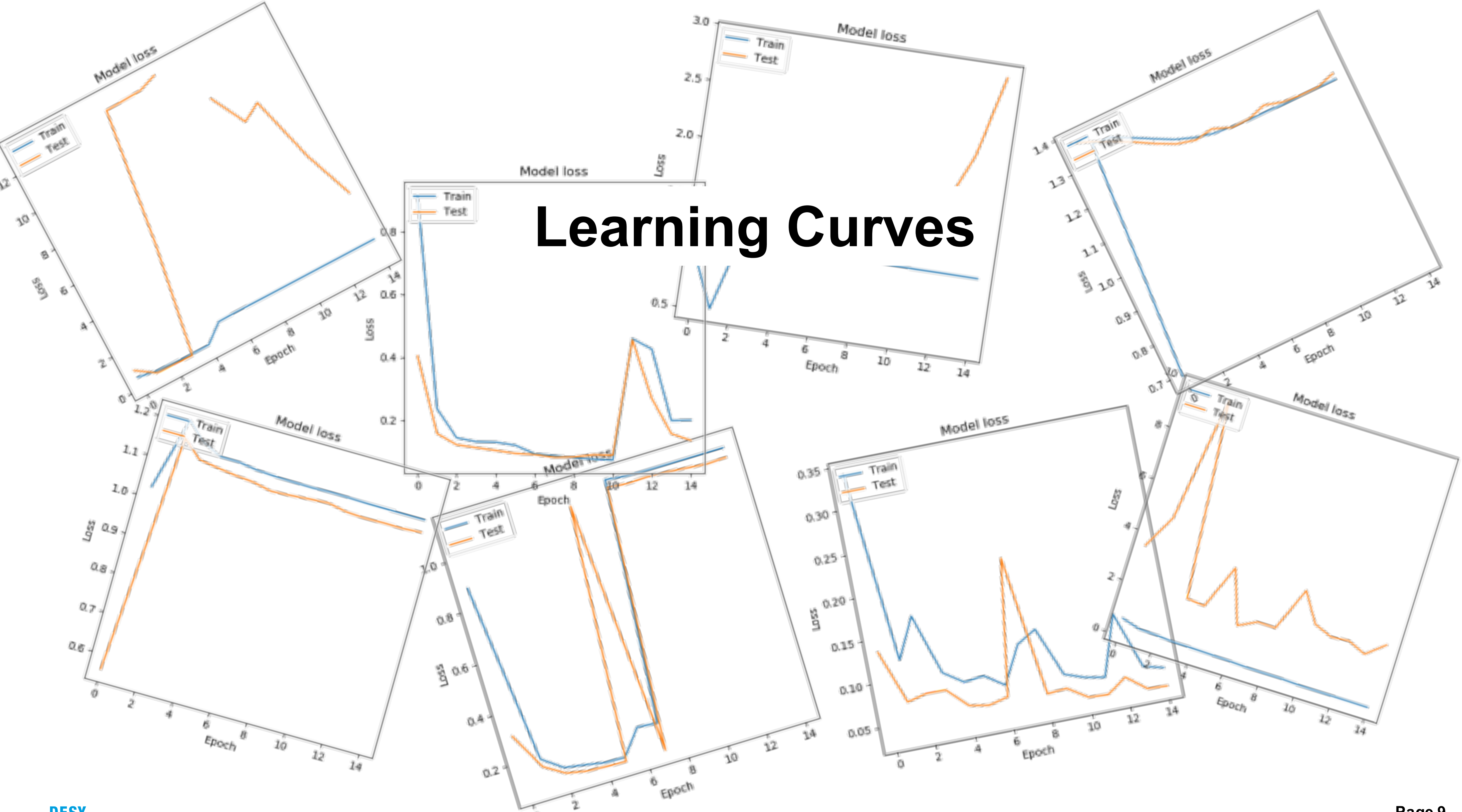
How to debug neural networks. Manual.
https://medium.com/machine-learning-world/how-to-debug-neural-networks-manual-dc2a200f10f2

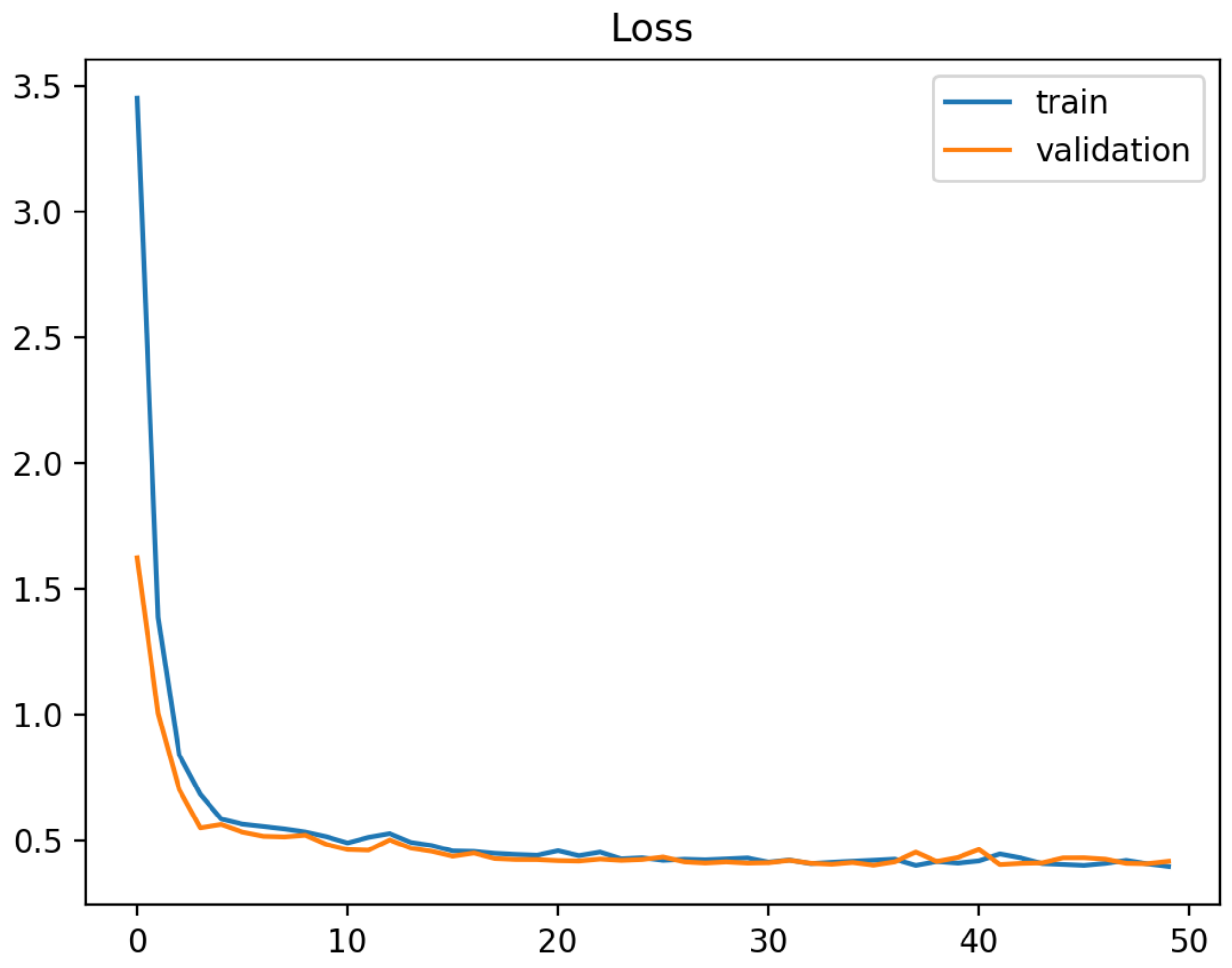
My Neural Network isn't working! What should I do?
http://theorangeduck.com/page/neural-network-not-working

37 Reasons why your Neural Network is not working
https://blog.slavv.com/37-reasons-why-your-neural-network-is-not-working-4020854bd607

How to use Learning Curves to Diagnose Machine Learning Model Performance
https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

Learning Curves





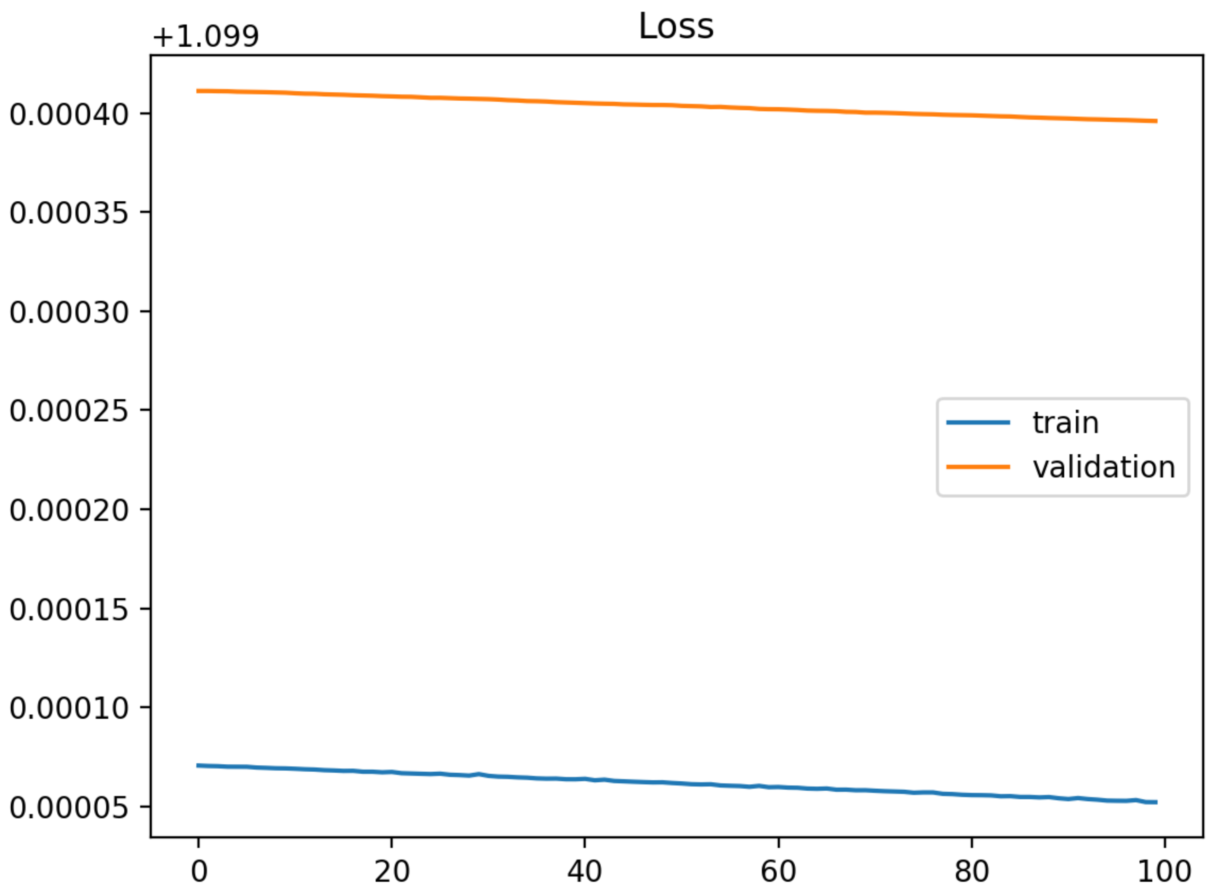
Train Learning Curve:
gives an idea of how well the model is learning.

Validation Learning Curve:
gives an idea of how well the model is generalising.

A plot of learning curves shows a good fit if:
The plot of training loss decreases to a point of **stability**.
The plot of validation loss decreases to a point of stability and has a **small gap** with the training loss.

https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

Underfitting

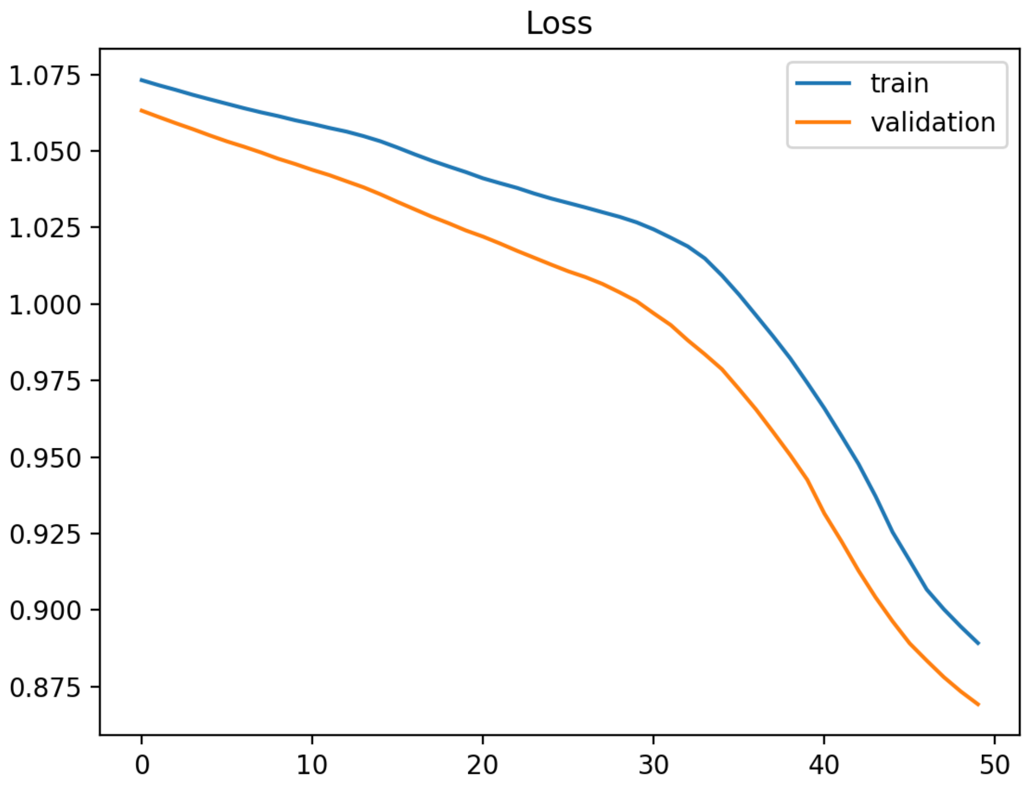


Underfitting refers to a model that cannot learn the training dataset.

The training loss may show **a flat line** or noisy values of relatively high loss
The model was **unable to learn** the training dataset at all.
This is common when the model does **not have a suitable capacity** for the complexity of the dataset.

https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

Underfitting

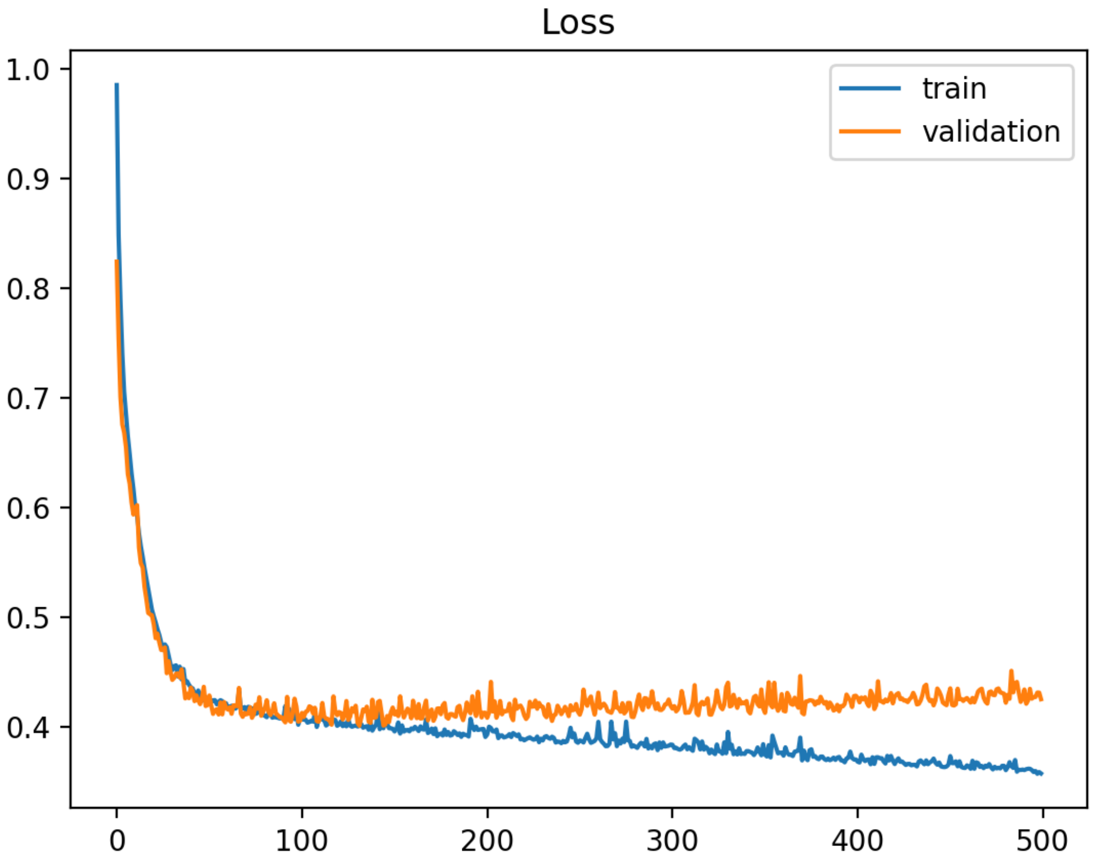


Underfit can also be identified by a training loss that is decreasing and **continues to decrease at the end** of the plot.

This indicates that the model is **capable of further learning** and possible further improvements and that the training process was halted prematurely.

https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

Overfitting



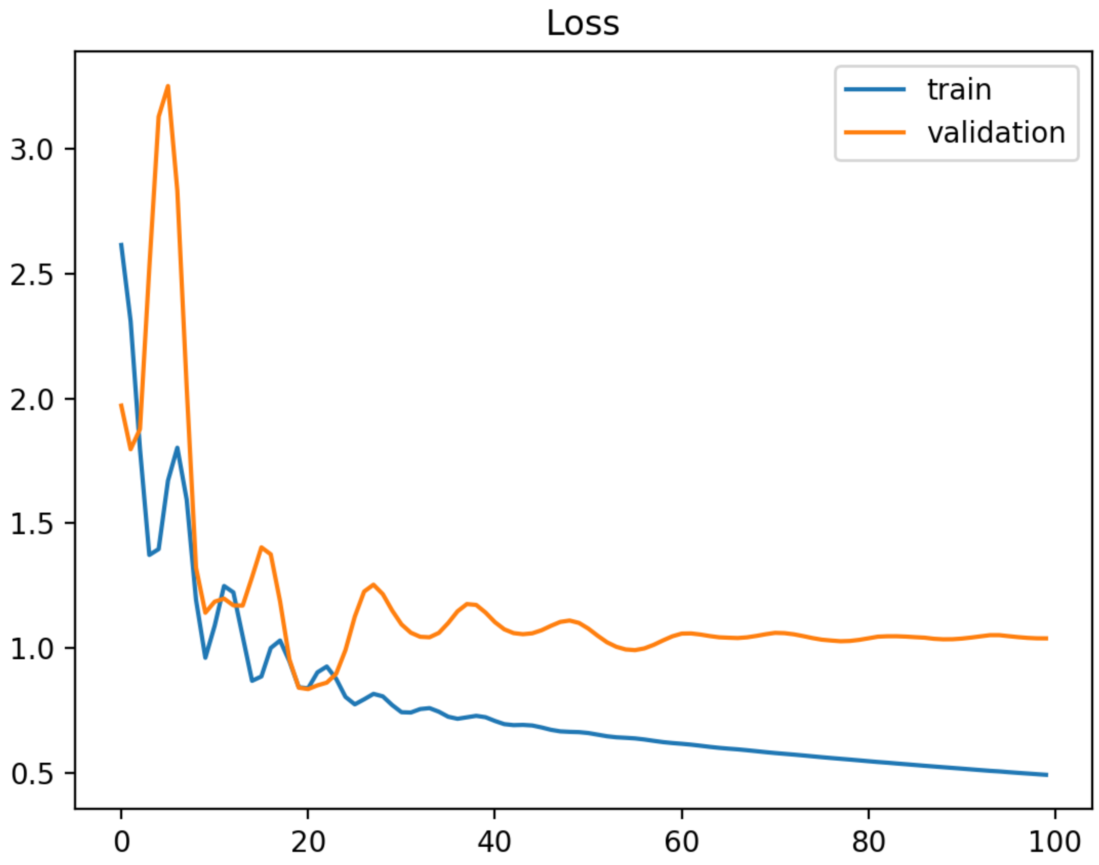
A model has **learned the training dataset too well** (including statistical noise or random fluctuations)

The more specialized the model becomes to training data, the **less well it is able to generalize** to new data.

This often occurs if the model **has more capacity** than is required for the problem.

https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

Unrepresentative Train Dataset



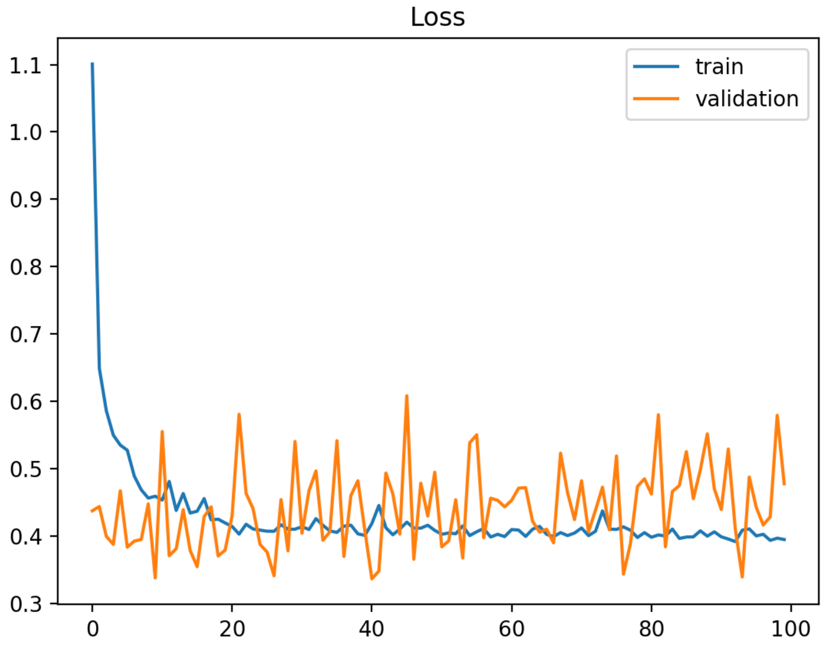
An unrepresentative training dataset means that the training dataset does not provide sufficient information to learn the problem, relative to the validation dataset used to evaluate it.

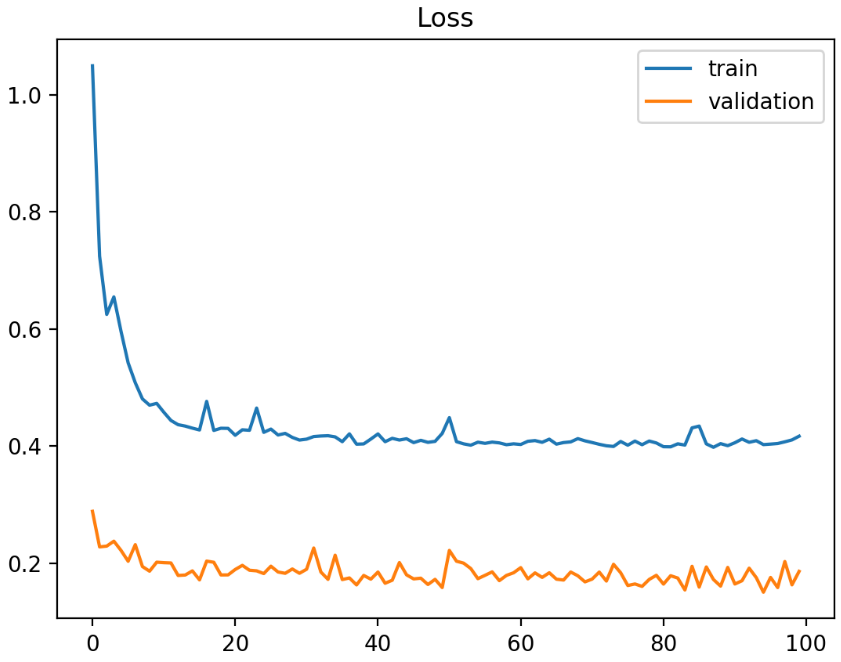
e.g. training dataset has too few examples

Both curves show improvement, but a **large gap remains** between curves.

https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

Unrepresentative Validation Dataset

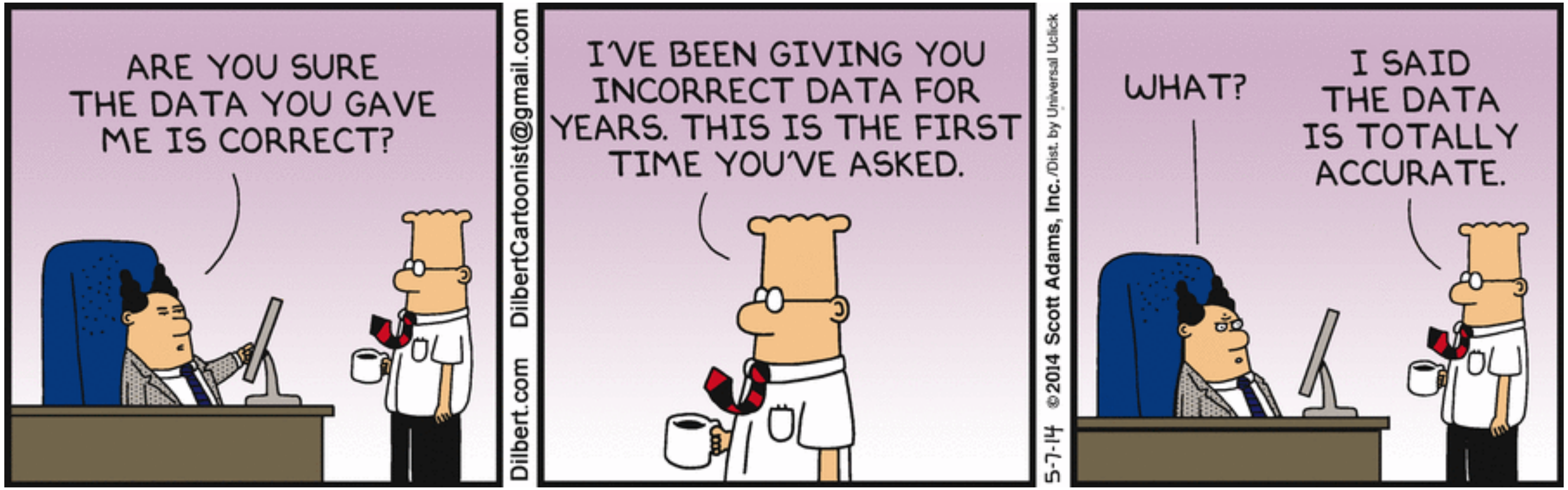




An unrepresentative validation dataset means that the validation dataset does not provide sufficient information to evaluate the ability of the model to generalize.
e.g. validation dataset has too few examples
validation loss curve with noisy movements around the training loss.

Or validation loss is lower than the training loss. This indicates that the validation dataset may be easier for the model to predict than the training dataset.

Dataset issues



Check your input data

- Correct data & label **path**?
- Images with **all zeros**?
- Using the **same image/batch** over and over?

Check the data loader/generator

- the code that passes the input to the net might be broken.

Make sure input & output match

- Do input samples have the **correct labels?**
- Are input and ground truth data **shuffled** the same way?
- **Same pre-processing** steps for in and truth?

Try random input

- Pass random numbers/images/labels instead of actual data and check.

Shuffle the dataset

- If your dataset has a particular order to it (ordered by label) this could negatively impact the learning.
- Make sure you are shuffling input and labels together.
- Make sure your batches don't contain a single label

Reduce class imbalance

- Are there a 1000 class A images for every class B image?
- Balance loss function or try other class imbalance approaches.

Do you have enough training examples?

- If you are training a net from scratch, you probably need lots of data.
- For image classification you probably need 1000 images per class or more.

Is the relationship between input and output too random?

- Maybe the input is not sufficiently related to the output.
- There isn't an universal way to detect this as it depends on the nature of the data.

Is there too much noise in the dataset?

- There might be **too many bad labels** that the network couldn't learn.
- Check input samples manually and see **if labels match**.
- **The cutoff point is up for debate**, as a paper got above 50% accuracy on MNIST using 50% corrupted labels.

Have enough relevant data!

- (biomedical) data may not be available due to sensitivity of data / data protection rules etc.
- You should not have too much data, e.g. **inclusion of useless data does not help.**
- Human intelligence needed to select the right data, use only relevant data
- Data is often **human labelled, thus error-prone**.
- Using faulty data or dirty data can lead to making bad predictions

Dataset problems

Try to overfit your model with small dataset
If your loss doesn’t go down, then your problem is deeper.

Use iterative logic in solving problem
Try to build the simplest network that solve your problem and then move step by step to global problem.

Use balanced datasets
Your training data should have same number of inputs for each class.

Network capacity vs dataset size
Your dataset should be enough for network to learn.
small dataset and big network: will stop learning
big dataset and small network: you will see jumping of loss, network capacity can’t store so much information.

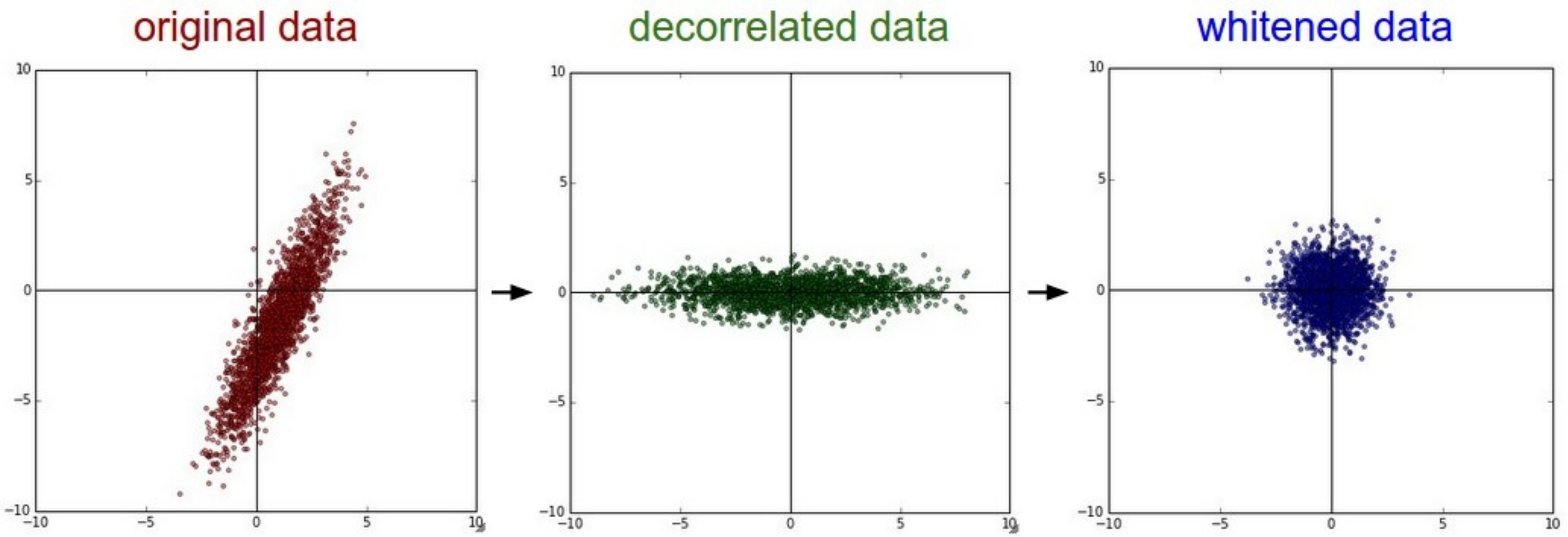
Become one with the data

- **Begin by thoroughly inspecting your data.**
- understand their distribution and look for patterns.
- Remove duplicates/corrupt images
- Look for **data imbalances** and biases.
- Visualize it.

- **Think about your data: How does you (your brain) classify the data?**
- How much variation is there and what form does it take?
- What variation is spurious and could be preprocessed out?
- Does spatial position matter or do we want to average pool it out?
- How much does detail matter and how far could we downsample the images?
- How noisy are the labels?

Knowing you data enables you to to look at your network (mis)predictions and understand where they might be coming from.

Dataset preprocessing



data scaling - data centring - data standardisation - data normalisation

Neural Networks make only a few basic assumptions about the data they take as input

But:

- the **space** the data lies in is somewhat **continuous**
- that a point between two data points is "a mix" of these two data points
- two nearby data points are "similar" things.

Having big discontinuities in the data space, or large clusters of separated data which represent the same thing, is going to make the learning task much more difficult.

Normalisation/Standardisation

Normalisation typically means rescaling the values into a range of [0,1].
Standardisation typically means rescales data to have a mean of 0 and a standard deviation of 1 (unit variance).

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Minmax scaling [0-1]

$$z = \frac{x - \mu}{\sigma}$$

Z-score standardization
[mean 0 & unit variance]

Most parts of a neural network assume that data is normalised/standardised.

This assumptions appears everywhere in deep learning, from **weight initialization**, to **activation functions**, to the **optimization algorithms** which train the network.

Do not use normalisation blindly!

- The scale of features in the neural network will also govern their *importance*.

- Feature in output with large scale will generate a larger error compared to other features.
- Large scale features in the input will dominate the network and cause larger changes downstream.
- **For this reason it isn't always enough to use the automatic normalization.**

Do not use normalisation blindly!

- Is the range of a **feature small/large because it is an (un-)important feature** (**don't re-scale**), or because it has some **small/large unit** in comparison to other features (**re-scale**)?

- Be careful with features that have such a small range that their standard deviation becomes (close to) zero - these will produce instabilities or NaNs if you normalize them.

Do not use normalisation blindly!

https://towardsdatascience.com/normalization-vs-standardization-quantitative-analysis-a91e8a79cebf

model scaler	CART	KNN	LDA	LR	MLP	NB	RF	SVM
	0.735	0.753	0.699	0.754	0.778	0.657	0.71	0.608
MaxAbsScaler	0.735	0.808	0.699	0.778	0.767	0.657	0.71	0.705
MinMaxScaler	0.735	**0.813**	0.699	0.778	0.767	0.657	0.71	0.711
Normalizer	0.716	0.765	0.692	0.699	0.724	0.662	**0.723**	0.524
PowerTransformer-Yeo-Johnson	0.729	0.813	**0.747**	0.76	**0.839**	**0.752**	0.71	0.814
QuantileTransformer-Normal	0.735	0.783	0.694	0.718	0.808	0.752	0.717	0.832
QuantileTransformer-Uniform	**0.74**	0.789	0.742	**0.808**	0.808	0.752	0.717	0.772
RobustScaler	0.735	0.76	0.699	0.735	0.808	0.657	0.71	0.776
StandardScaler	0.735	0.796	0.699	0.742	0.82	0.657	0.71	**0.849**

Data Transformation

Is there some **simple transformation** to ensure that data points which represent things *we* know **are similar always get similar numerical representation**?

Is there a **local coordinate system** you can represent your data in that makes things more natural - perhaps a better colour space - a different format?

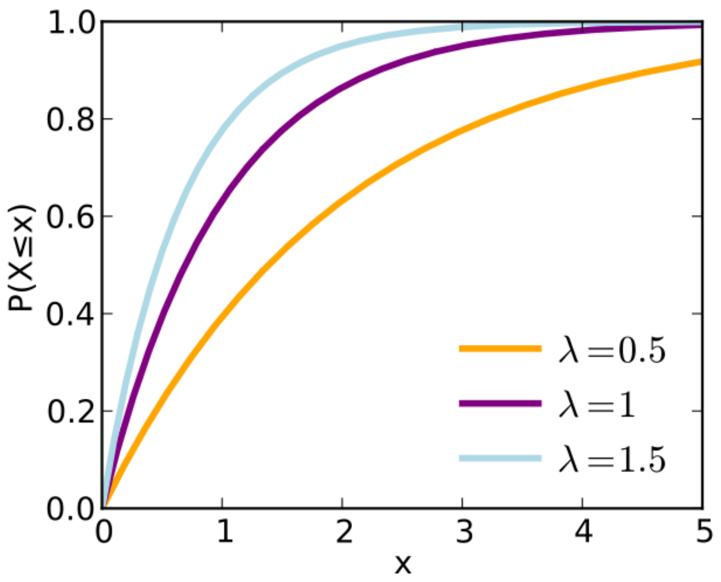
Are different features on the same scale?

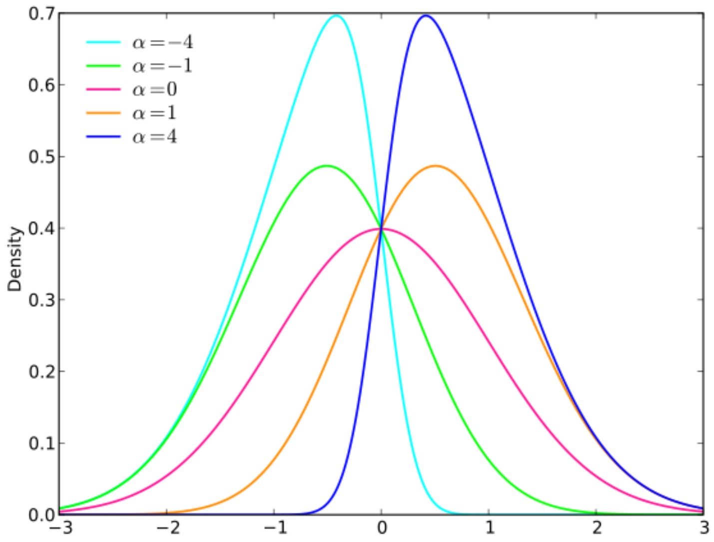
Data Transformation

Does a column look like:

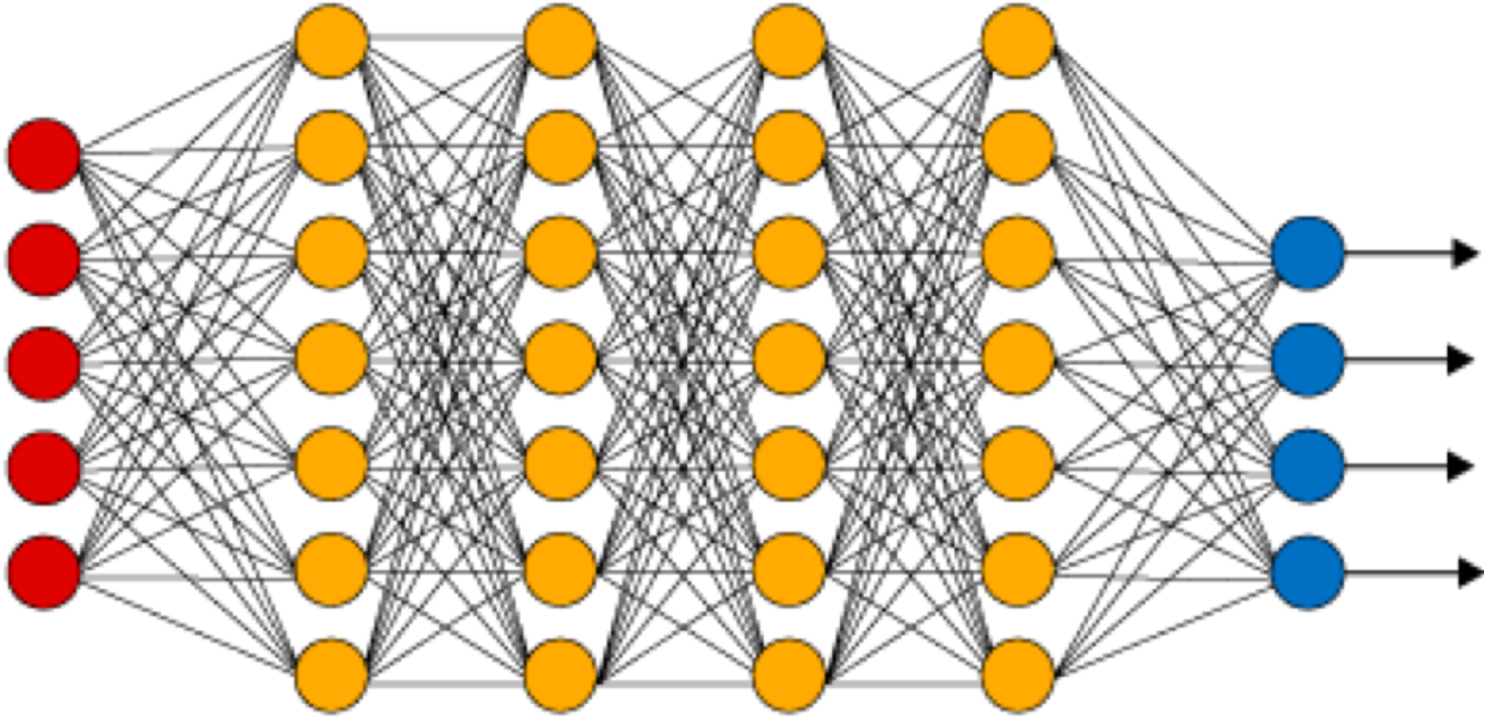
- a **skewed Gaussian**, consider adjusting the skew with a **Box-Cox transform.**
- an **exponential distribution**, consider a **log transform.**
- it has some features, but they are being clobbered by something obvious, try **squaring, or square-rooting.**

- Can you make a feature discrete or binned in some way to better emphasize some feature.





Network/Training issues



Learning Rate

The amount the weights updated during training is the "*learning rate*."

A learning rate of **0.1**, means that weights are updated **10%** of the weight error each time.

"The learning rate is perhaps the most important hyperparameter. If you have time to tune only one hyperparameter, tune the learning rate."

Learning Rate — Choosing an optimum learning rate is important as it decides whether your network converges to the global minima or not.

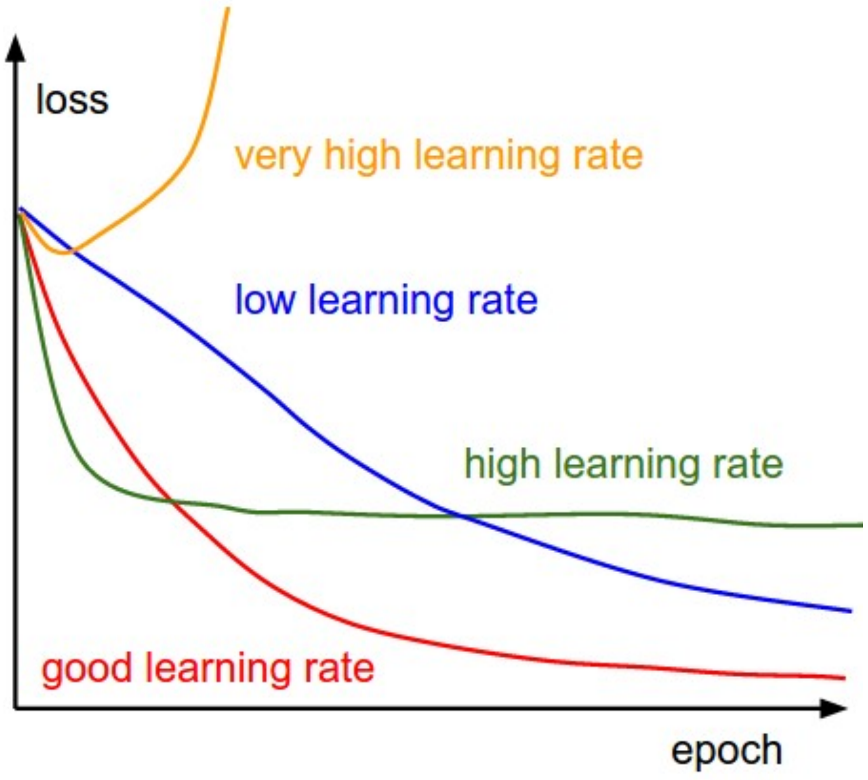
Learning Rate

High learning rate:

- hardly gets you to the global minima (good chance of overshooting it)

Small learning rate:

- might converge to the global minimum but it takes a huge amount of time.
- makes the network susceptible to getting stuck in local minimum.



How to find best learning rate?

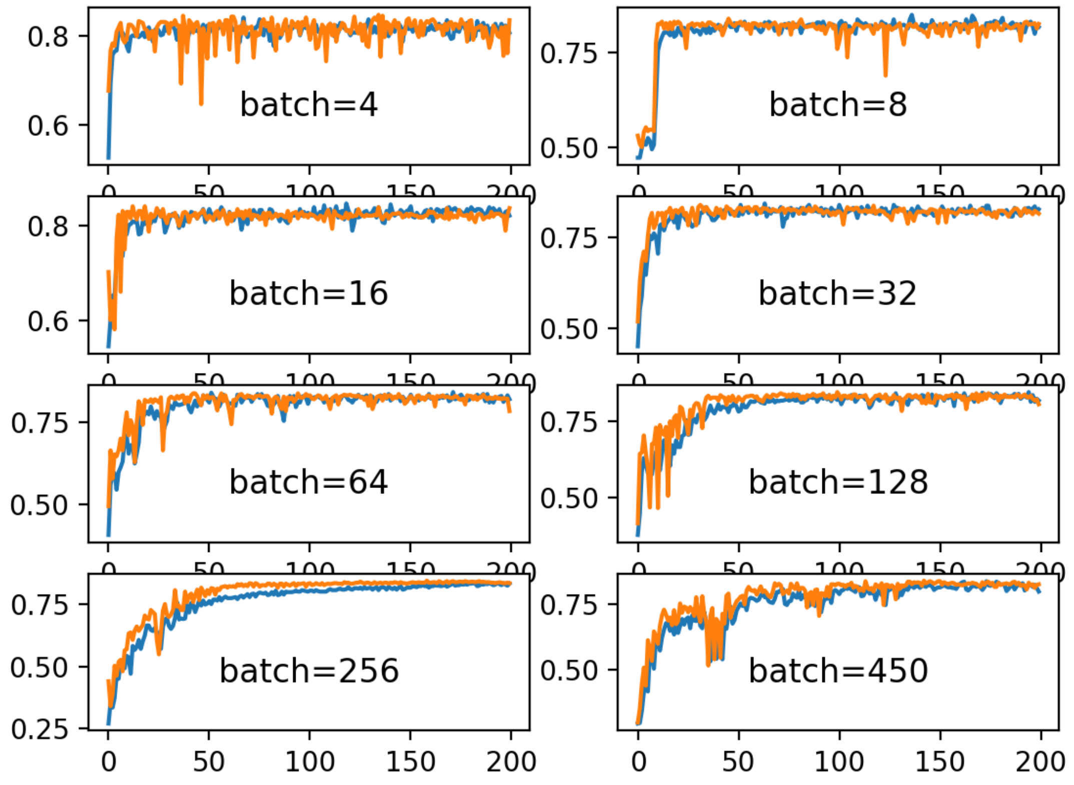
Turn gradient clipping off. Find the highest value for the learning rate which doesn't make the error explode during training. Set the learning rate one order of magnitude lower than this - this is probably pretty close to the optimal learning rate.

Batch Size

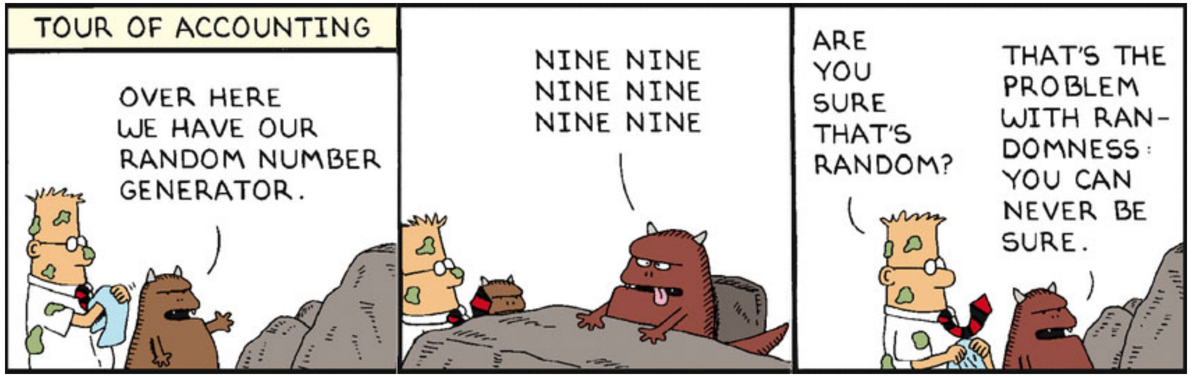
Using too large a batch size can have a negative effect on the accuracy of your network during training since it reduces the stochasticity of the gradient descent.

Find the minimum batch size with which you can tolerate the training time.
optimal use of the GPU parallelism batch size might not be the best for accuracy

Don't be scared to start with a very small batch size such as 16, 8, or even 1.



Initialisation



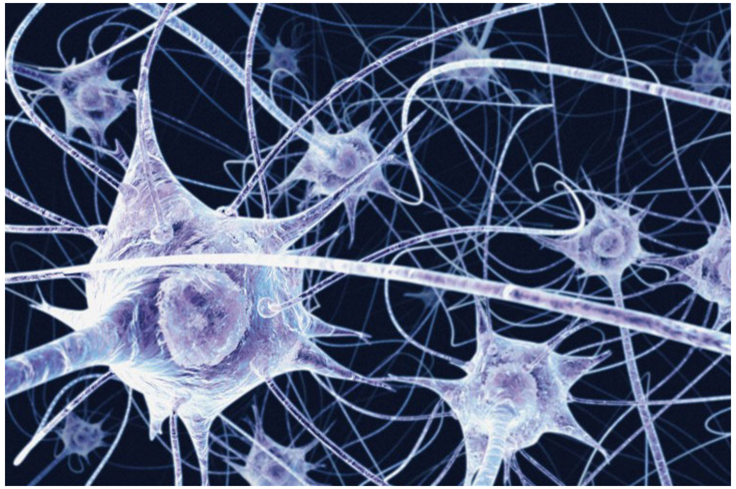
Many components in the NN assume a correct or standardized weight initialization

The 'he', 'lecun' or 'xavier' weight initializations are all popular choices which should work.

For small networks it's enough to use some Gaussian distribution initializers around 1e-2–1e-3.

For deep networks this will not help, because your weights will be multiplied with each other many times that will lead to very small numbers which will almost kill gradients on back-propagation step.

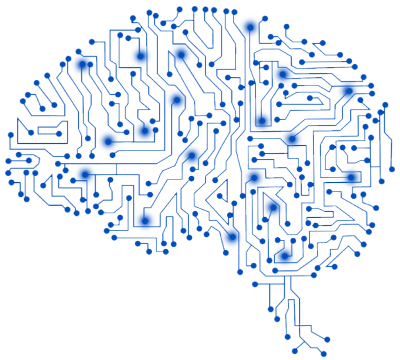
Network contains Bad Gradients



Deep networks using ReLU activation functions can suffer from so called **"dead neurons" caused by bad gradients**. This can negatively affect the performance of the network, or in some cases make it completely impossible to train.

If you find that your training error does not change from epoch to epoch it may be that **all your neurons have died. Try other activation function such as leaky ReLUs or ELUs .**

You Used a Network that was too Deep



Deeper is better right? Well not always ...

Deeper is usually better when we are trying to squeeze 1% more accuracy out
If your little network is failing to learn anything then your 100 layer network is going to fail just as badly if not worse.

Start with 3 to 8 layers. Start experimenting with deeper networks only when you already have things working well and are starting to investigate how to increase the accuracy.

Starting small also means that training your network will be faster, inference will be faster, and iterating on different designs and setups will be faster. **Initially, all of these things will have a much bigger impact on your accuracy than simply stacking a few more layers.**

Number of Hidden Units

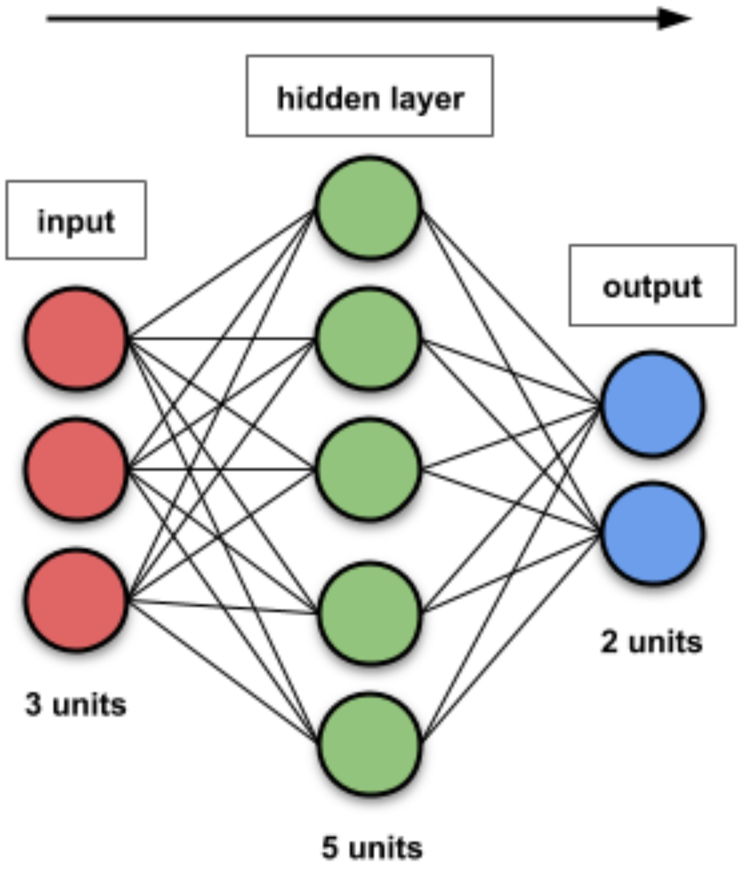
Too many or too few hidden units can make your network difficult to train.

Too few units

- it may not have the capacity to express the task required

Too many

- it may become slow and unwieldy to train with residual noise that is hard to remove.



Number of Hidden Units

Start with something between the number of input neuros and output neurons.

consider roughly what you think may be ***the fewest number of real values required*** **to express the information** scale this number up a bit. (to allow for dropout, to use a more redundant representation)

- for classification use **five to ten times the number of classes** as a good initial guess
- for regression use **two to three times the number of input or output variables**.

(number of hidden units often has quite a small impact on neural network performance when compared to other factors; overestimating the number of hidden units will make training slower)

Regularisation

Regularisation (dropout, noise, or *some* form of stochastic process) is an important aspect of training neural networks.

Even if you have vastly more data than parameters, or over-fitting does not matter, it is helpful to add dropout or some other form of noise.

The most basic way to regularize a neural network is to add dropout before each linear layer (convolutional or dense) in your network.

Regularization isn't just about controlling over-fitting. By introducing some stochastic process into the training procedure you are in some sense **"smoothing" out the cost landscape**. This can speed up training, **help deal with noise or outliers in the data**, and prevent extreme weight configurations of the network.

Summary

If your network refuses too learn….

Check your data

- dataset
- preprocessing (normalisation, standardisation, transformation, scaling)

Adjust

- learning rate
- batchsize
- initialisation

9 Reasons why your machine learning project will fail
https://www.kdnuggets.com/2018/07/why-machine-learning-project-fail.html

20 Tips, Tricks and Techniques That You Can Use To Fight Overfitting and Get Better Generalization
https://machinelearningmastery.com/improve-deep-learning-performance/

Improving the Performance of a Neural Network
https://towardsdatascience.com/how-to-increase-the-accuracy-of-a-neural-network-9f5d1c6f407d

How to debug neural networks. Manual.
https://medium.com/machine-learning-world/how-to-debug-neural-networks-manual-dc2a200f10f2

My Neural Network isn't working! What should I do?
http://theorangeduck.com/page/neural-network-not-working

37 Reasons why your Neural Network is not working
https://blog.slavv.com/37-reasons-why-your-neural-network-is-not-working-4020854bd607

How to use Learning Curves to Diagnose Machine Learning Model Performance
https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

Mailing List

Mailinglist for ML activities at DESY / Announcements of ML related events.

desy-ml@desy.de

Please sign up! Future seminar announcements will be done on this list only!

https://lists.desy.de/