Basic Deep Learning

Leander Thiele, leander.thiele@ipmu.jp

The plan for this class…

- Deep Learning is currently more an art than a science
- Practice makes perfect: the hands-on exercises will familiarize you with the basic deep learning workflow
- Let’s have a conversation! Your input will be as important for your colleagues as whatever I have to say

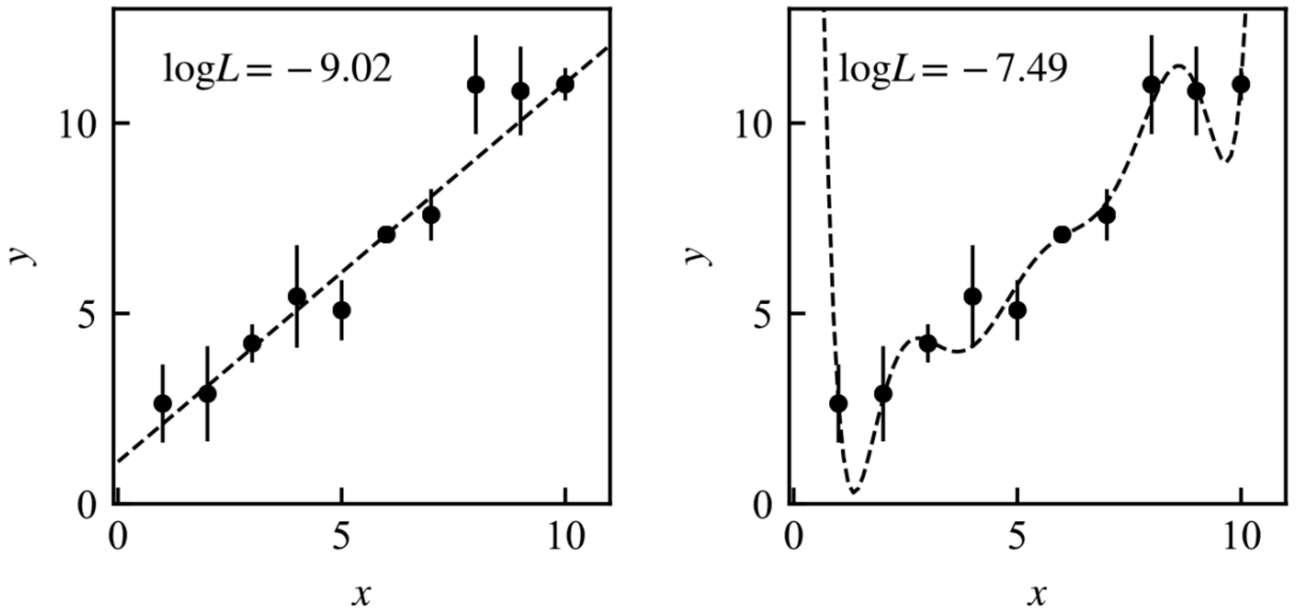
The plan for this class…

- understand the utility of deep learning
- multi-layer perceptron
- training via stochastic gradient descent
- supervised learning: regression & classification
- how to train deep neural nets *systematically*

Overfit and model selection

- Overfit = a statistical model describes random noise
  - The likelihood of a model often increases with the number of model parameters.
  - In high-dimensional problems, evaluating a model solely based on likelihood becomes inappropriate.
- Model selection:
  - If the model is too simple → it cannot adequately explain the data.
  - If the model is too complex → overfitting occurs.
  - It's essential to choose a model with the right level of complexity. = model selection
- Two approaches: Information criteria & cross-validation.

Overfit =
Low generalization error



When do we want/need to use Deep Learning in Science?

What we learned yesterday…

Kernel functions

$$\Sigma = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{pmatrix}$$

- Replacing covariances to the kernel functions: $\sigma_{ij} = k(x_i, x_j)$
- RBF (Gaussian) kernel
  - The distance between two points $\mathbf{x}_i, \mathbf{x}_j$ ~ their correlation.
  - Powerful for capturing smooth and continuous relationships in the data.
  - θ 2 = length-scale
- Estimating a few kernel parameters from N data points → Gaussian Process Regression

RBF kernel $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \theta_1 \exp\left\{-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2}{\theta_2}\right\}$

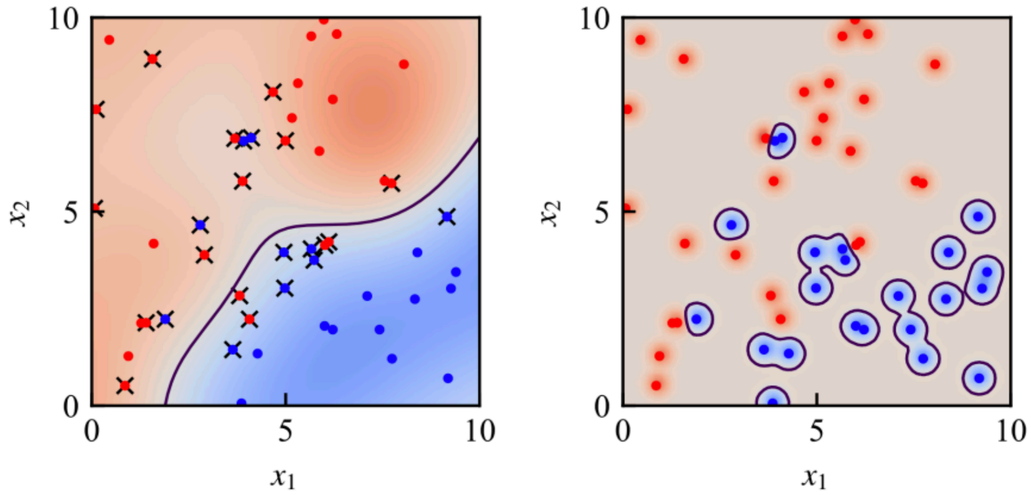
Exponential kernel $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \theta_1 \exp\left\{-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2}{\theta_2}\right\}$

Periodic kernel $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \theta_1 \exp\left\{-\theta_2 \cos\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2}{\theta_3}\right\}$

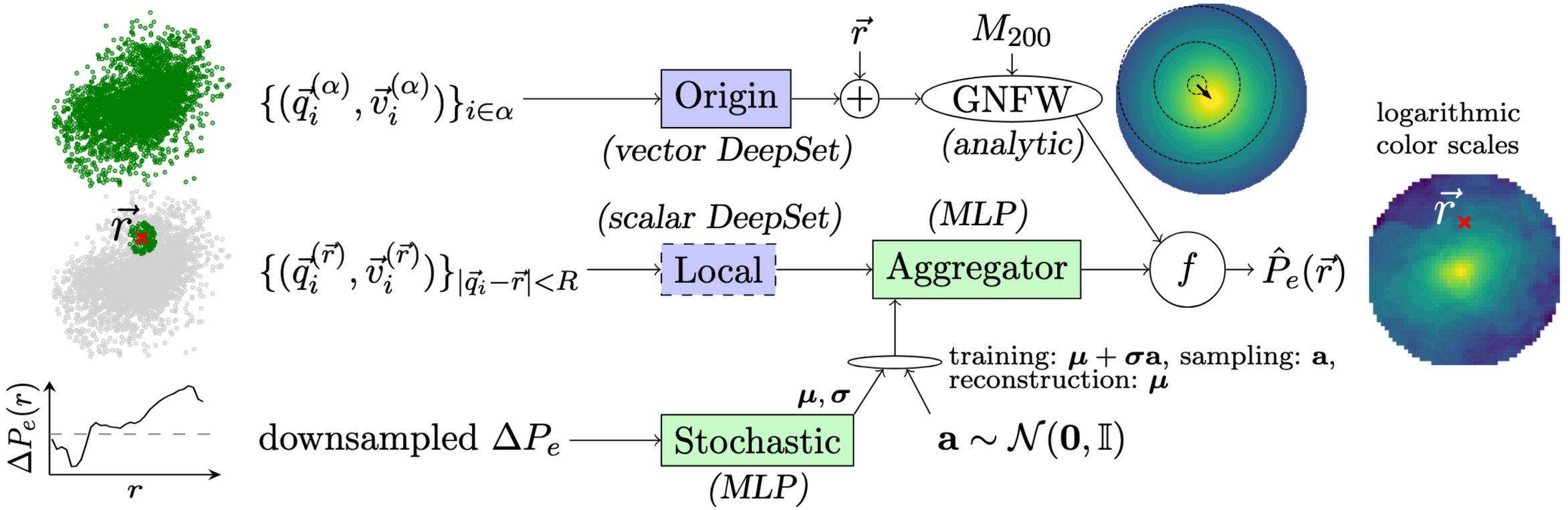
Linear kernel $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \theta_1 \boldsymbol{x}_i^T \boldsymbol{x}_j$

Matern kernel $K(x, x') = \frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu}|x-x'|}{\rho}\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu}|x-x'|}{\rho}\right)$

K ν is the modified Bessel function of the second kind



- Use RBF kernel
- Determine regularization parameter C and kernel parameter σ by evaluating with ROC-AUC through cross-validation
- Left Figure: Best model
- Right Figure: Overfitted model
  - Kernel variance is small, and C is large.



Sometimes it is just super useful to be able to insert a maximally expressive function, subject to constraints.

This does not necessarily mean full going black box.

What, actually, is Deep Learning?

Deep Learning is…

- … hard

Getting everything conceptually right can be quite difficult. And deep learning tends to be unforgiving and difficult to debug.

Being completely in the wrong part of hyperparameter space can happen. With experience it gets easier to get out of there.

Deep Learning is…

- … hard
- … easy

Once everything is correctly set up, and we're approximately right about the hyperparameters, neural nets are actually quite easy to optimize!

Hyperparameters tend to have predictable effects, getting us close to optimum quite fast.

Deep Learning is…

- … hard
- … easy
- … fun!

Setting up an architecture is a creative process. What works and what doesn’t usually corresponds to real physics, so we’re having a learning experience.

Flavours

supervised (today):

- regression
- classification

unsupervised (Thursday) — generative models

AIC（Akaike information criterion）

$$\mathrm{AIC} = -2(\log L(\hat{\boldsymbol{\theta}}) - K) = -2\log L(\hat{\boldsymbol{\theta}}) + 2K$$

- The log-likelihood $\log L(\hat{\theta})$ of the data can become biased when the model is too complex.
- To correct this bias, AIC introduces a penalty term, the number of model parameters K.

AIC（Akaike information criterion）

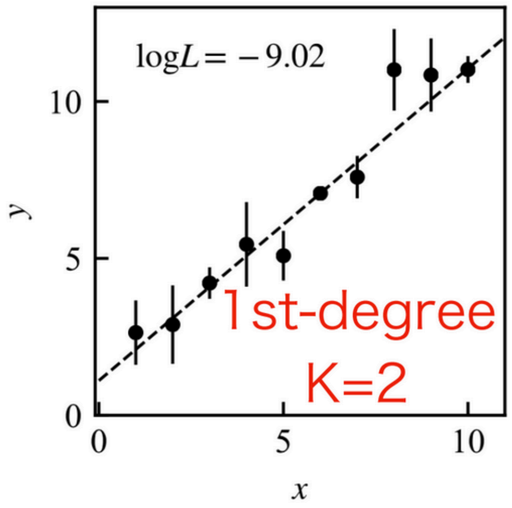
$$\mathrm{AIC} = -2(\log L(\hat{\boldsymbol{\theta}}) - K) = -2\log L(\hat{\boldsymbol{\theta}}) + 2K$$

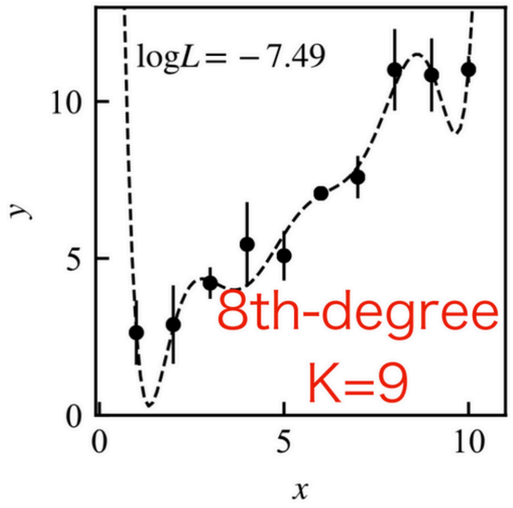
- The log-likelihood $\log L(\hat{\theta})$ of the data can become biased when the model is too complex.
- To correct this bias, AIC introduces a penalty term, the number of model parameters K.

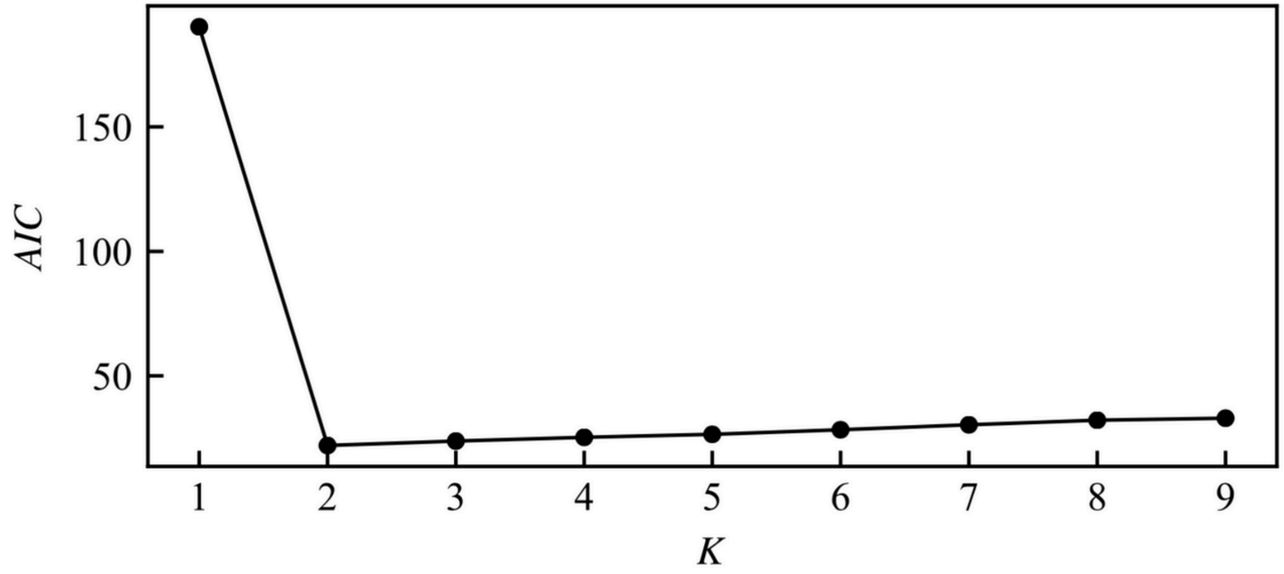
Usually, this is our intuition a Bayesians — “Occam’s razor”.

Does this intuition apply for deep learning, though?

After all, a neural net is just a very complicated model, right?

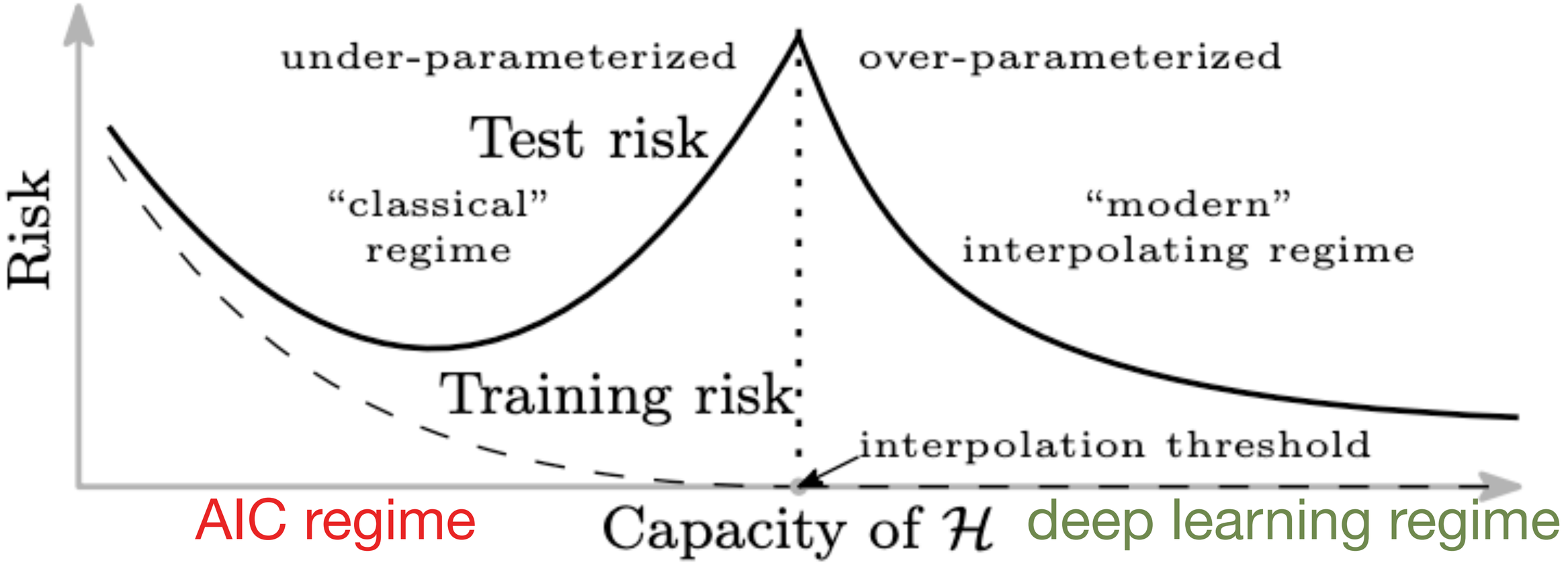






“model capacity”

Overparameterization



- parameter counting is usually not useful when constructing neural nets
- just choose the largest architecture that fits on your hardware
- overfitting is still very much possible, and we need to develop ways to deal with it
- regularization, as introduced yesterday, is key! But in the high-dimensional parameter spaces of neural nets, it can be as much about training/ dynamics as about the formal loss function.

Perceptron

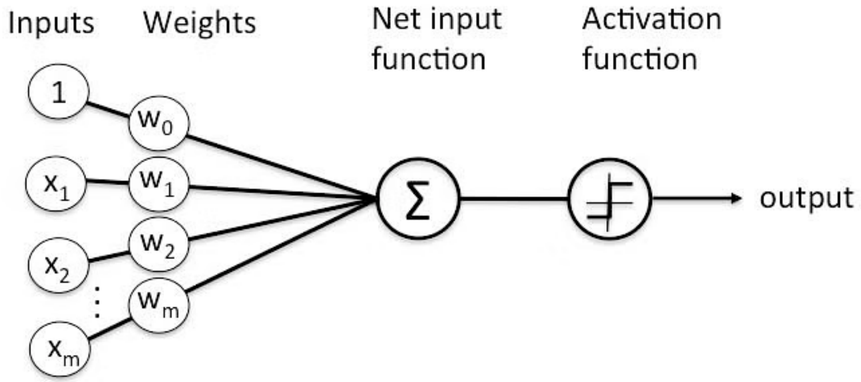
The building block of typical feed-forward neural nets

$$z(x) = f(Wx + b)$$

activation function

weight

bias



Perceptron

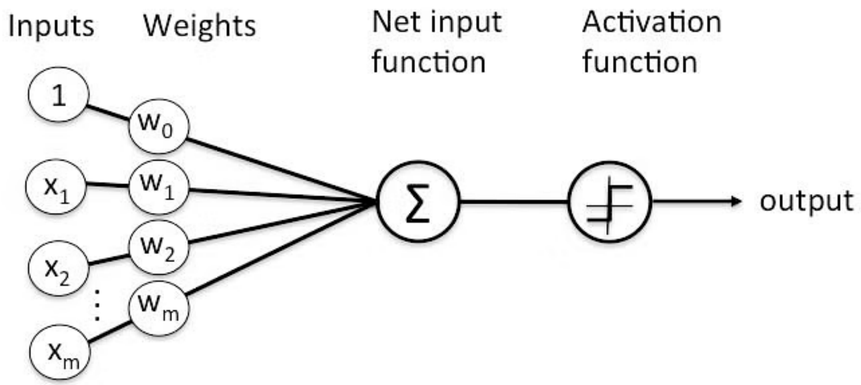
The building block of typical feed-forward neural nets

$$z(x) = f(Wx + b)$$

activation function

weight

bias



Why this specific choice of a non-linear function?

Perceptron

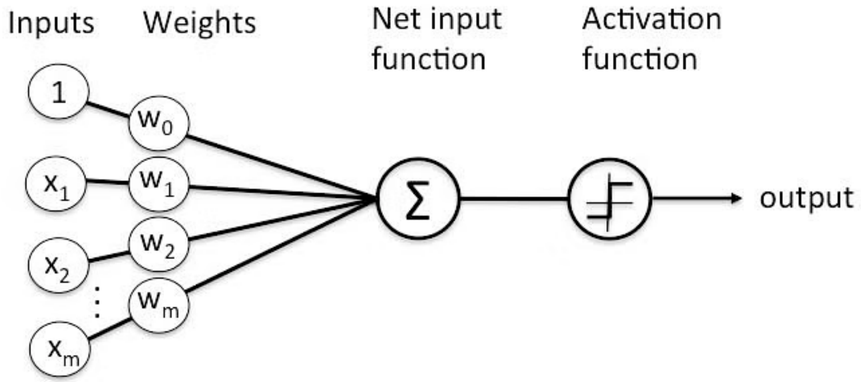
The building block of typical feed-forward neural nets

$$z(x) = f(Wx + b)$$

activation function

weight

bias



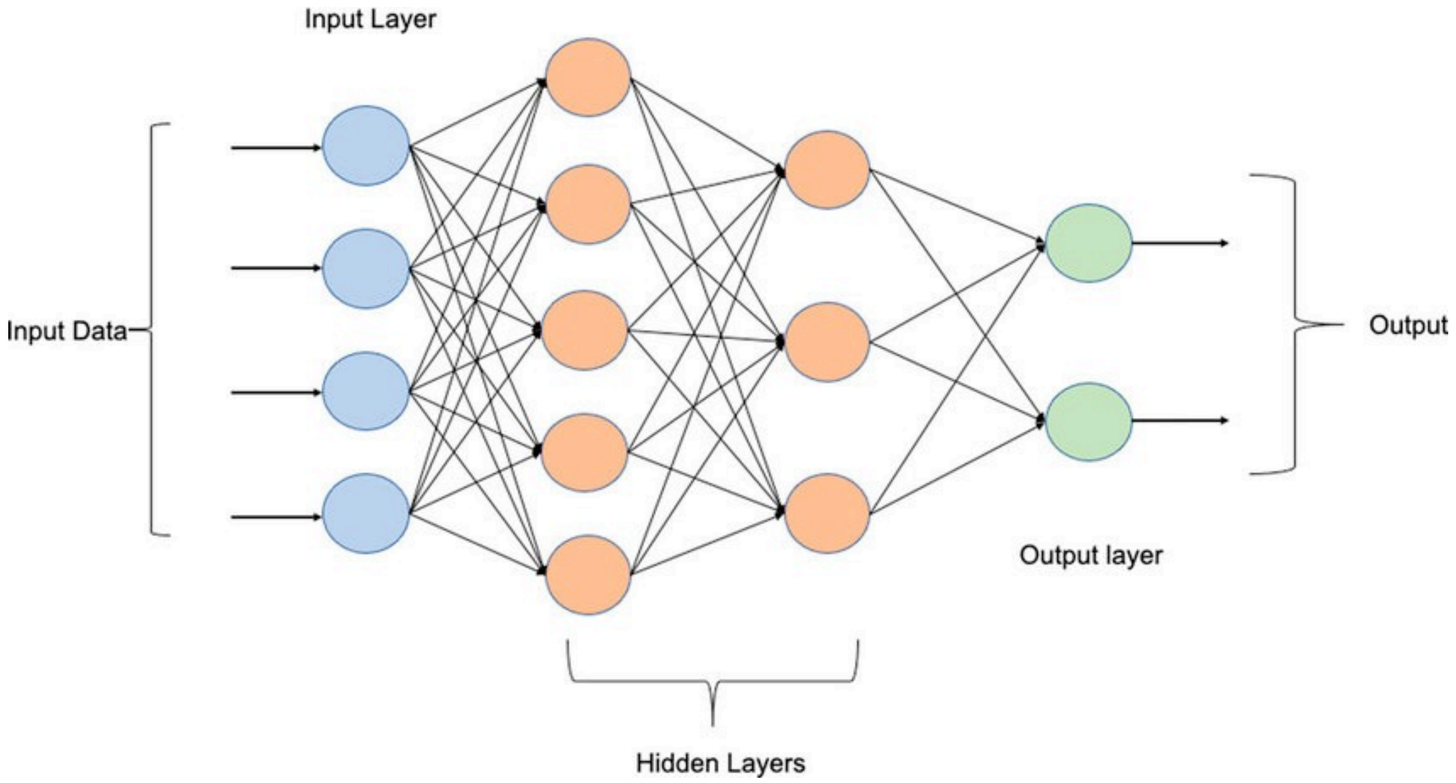
What should we use as activation functions?

Multi-layer perceptron (MLP)

easy: stack multiple perceptrons!

One subtlety: often we want to keep the output on the entire real line.

Just use a Wx+b affine transformation at the end.



Hands-on #1: write an MLP

How to train a neural net?

In general, training a neural net is a non-convex optimization problem.

It is actually often NP-hard to find the global optimum.

How can we hope to train such an object into a useful state?

How to train a neural net?

In general, training a neural net is a non-convex optimization problem.

It is actually often NP-hard to find the global optimum.

How can we hope to train such an object into a useful state?

Empirically, it works pretty well! Probably due to over-parameterization.

How to train a neural net?

- pick a loss function. Needs to be adapted to the problem at hand (what are we trying to accomplish?).
- split data into training/validation/test sets (70/20/10 is a good guide).
- If constraints permit, multiple training/validation splits desirable (K-fold cross-validation)
- train by stochastic gradient descent

Stochastic gradient descent

In the most basic form, SGD iterates over the following:

$$\theta \leftarrow \theta - \text{lr} \nabla_\theta L(\theta)$$

learning rate

global loss function

$$L(\theta) = \sum_{\text{training examples } x} L(\theta, x)$$

Stochastic gradient descent

In the most basic form, SGD iterates over the following:

$$\theta \leftarrow \theta - \text{lr} \nabla_\theta L(\theta)$$

learning rate

global loss function

$$L(\theta) = \sum_{\text{training examples } x} L(\theta, x)$$

In practice, this doesn't work super well, …

Usable gradient descent

- mini-batching
- some form of momentum

How to compute gradients (efficiently)

Modern software, such as pytorch/tensorflow/…, has backpropagation.

This is an efficient way to compute the chain rule.

The math is not so difficult to understand, but not very enlightening and you won’t need it very often.

https://en.wikipedia.org/wiki/Backpropagation

How to compute gradients (efficiently)

Modern software, such as pytorch/tensorflow/…, has backpropagation.

This is an efficient way to compute the chain rule.

The math is not so difficult to understand, but not very enlightening and you won't need it very often.

https://en.wikipedia.org/wiki/Backpropagation

The key point for practitioners is that we need to tell the software which objects require gradients. Typically, network parameters have this set out-of-the-box, but sometimes we need to be explicit.

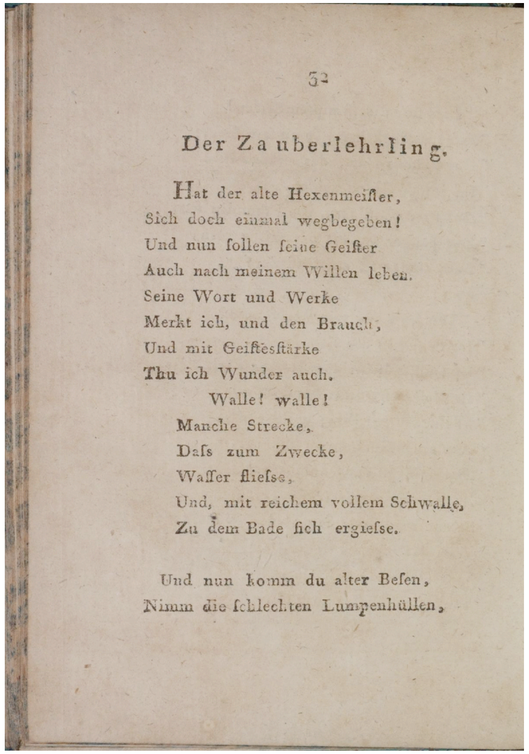
Hands-on #2: train 1-d MLP

Universal approximation theorem

Physicist's statement:

Under reasonable conditions on the input function (i.e., physicist's functions), for any accuracy requirement there exists a multi-layer perceptron (MLP) able to approximate the function with the required accuracy.

There is some dark magic…



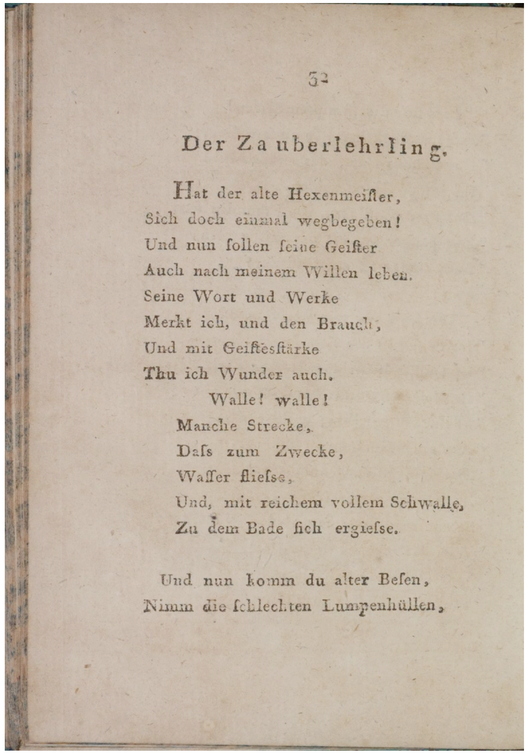
32

Der Zauberlehrling.

Hat der alte Hexenmeiſter,
Sich doch einmal wegbegeben!
Und nun ſollen ſeine Geiſter
Auch nach meinem Willen leben.
Seine Wort und Werke
Merkt ich, und den Brauch;
Und mit Geiſtesſtärke
Thu ich Wunder auch.
Walle! walle!
Manche Strecke,
Daſs zum Zwecke,
Waſſer flieſse,
Und, mit reichem vollem Schwalle,
Zu dem Bade ſich ergieſse.

Und nun komm du alter Beſen,
Nimm die ſchlechten Lumpenhüllen,

There is some dark magic…



32

Der Zauberlehrling.

Hat der alte Hexenmeiſter,
Sich doch einmal wegbegeben!
Und nun ſollen ſeine Geiſter
Auch nach meinem Willen leben.
Seine Wort und Werke
Merkt ich, und den Brauch;
Und mit Geiſtesſtärke
Thu ich Wunder auch.
Walle! walle!
Manche Strecke,
Daſs zum Zwecke,
Waſſer flieſse,
Und, mit reichem vollem Schwalle,
Zu dem Bade ſich ergieſse.

Und nun komm du alter Beſen,
Nimm die ſchlechten Lumpenhüllen,

Let’s learn it together and from each other!

Let’s learn it together and from each other!

Each of the 8 hack groups investigate one topic!

Slides linked in the Github. Find your group’s slide and fill it with information about the specific topic.

Please answer the following: what’s the motivation/ problem, what’s the intuition, when would you use which technique? If you can find original/seminal papers, please link them. Visualizations are great, too!

At the end, please be prepared to present your group’s slide to the other participants.

Deep learning workflow

1. formulate the problem. What do you want to learn? Does your problem have randomness (suggests generative model)? What is the performance metric you actually care about and how can you map this to a loss function?

Deep learning workflow

1. formulate the problem. What do you want to learn? Does your problem have randomness (suggests generative model)? What is the performance metric you actually care about and how can you map this to a loss function?

2. pick architecture. It needs to be expressive enough and fit on the hardware. Easy debugging: take a small subset of training data and make sure you can fit it perfectly

Deep learning workflow

1. formulate the problem. What do you want to learn? Does your problem have randomness (suggests generative model)? What is the performance metric you actually care about and how can you map this to a loss function?

2. pick architecture. It needs to be expressive enough and fit on the hardware. Easy debugging: take a small subset of training data and make sure you can fit it perfectly

3. pick a batch size B: such that #samples/unit time almost maximum, but not larger. Make sure your compute budget allows dozens of experiments seeing enough samples. You'll usually run more experiments than anticipated.

Deep learning workflow

1. formulate the problem. What do you want to learn? Does your problem have randomness (suggests generative model)? What is the performance metric you actually care about and how can you map this to a loss function?

2. pick architecture. It needs to be expressive enough and fit on the hardware. Easy debugging: take a small subset of training data and make sure you can fit it perfectly

3. pick a batch size B: such that #samples/unit time almost maximum, but not larger. Make sure your compute budget allows dozens of experiments seeing enough samples. You'll usually run more experiments than anticipated.

4. scan learning rate (most important hyper-parameter). Learning rate is strongly degenerate with batch size. That's why we fix batch size and don't touch it.

Deep learning workflow

1. formulate the problem. What do you want to learn? Does your problem have randomness (suggests generative model)? What is the performance metric you actually care about and how can you map this to a loss function?

2. pick architecture. It needs to be expressive enough and fit on the hardware. Easy debugging: take a small subset of training data and make sure you can fit it perfectly

3. pick a batch size B: such that #samples/unit time almost maximum, but not larger. Make sure your compute budget allows dozens of experiments seeing enough samples. You'll usually run more experiments than anticipated.

4. scan learning rate (most important hyper-parameter). Learning rate is strongly degenerate with batch size. That's why we fix batch size and don't touch it.

5. [layernorm/batchnorm sometimes useful to prevent divergences]

Deep learning workflow

1. formulate the problem. What do you want to learn? Does your problem have randomness (suggests generative model)? What is the performance metric you actually care about and how can you map this to a loss function?

2. pick architecture. It needs to be expressive enough and fit on the hardware. Easy debugging: take a small subset of training data and make sure you can fit it perfectly

3. pick a batch size B: such that #samples/unit time almost maximum, but not larger. Make sure your compute budget allows dozens of experiments seeing enough samples. You'll usually run more experiments than anticipated.

4. scan learning rate (most important hyper-parameter). Learning rate is strongly degenerate with batch size. That's why we fix batch size and don't touch it.

5. [layernorm/batchnorm sometimes useful to prevent divergences]

6. once it trains reasonably well, can start with secondary hyperparameters: architecture details, optimizer secondaries, regularization, …

Loss curve interpretation

typical problems:

- explosion, with and without recovery
- predicting the mean
- slow convergence, underfitting
- overfitting

Hyperparameter search

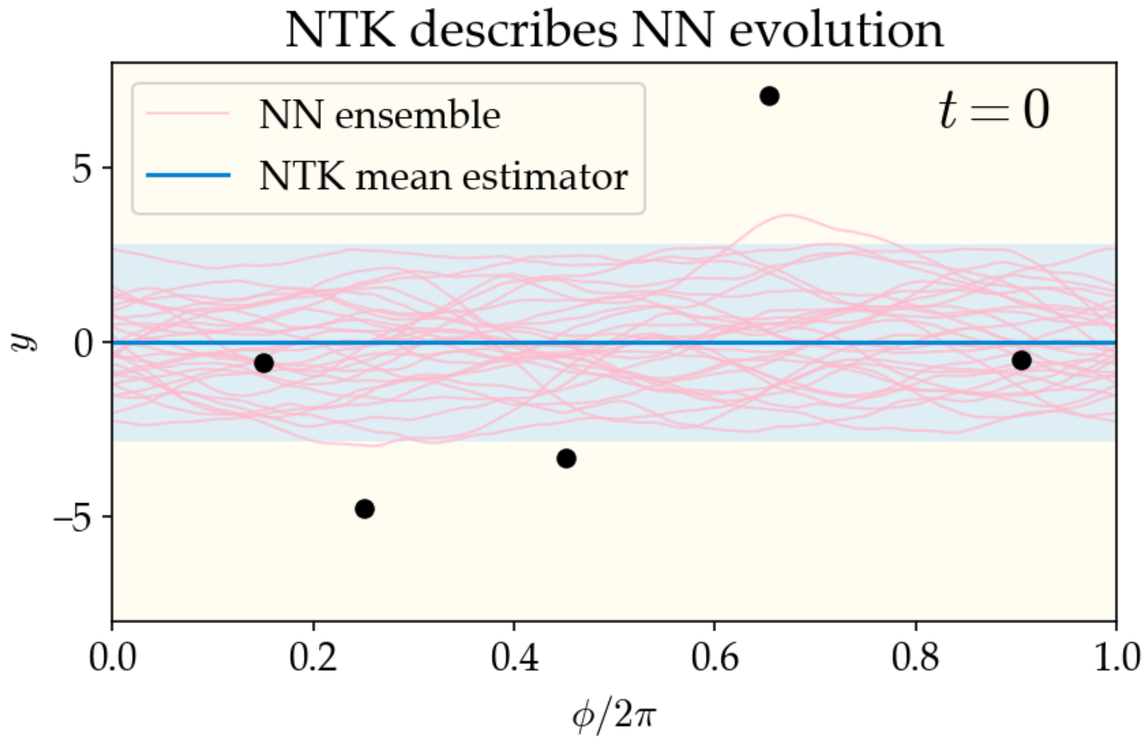
Automated tools for Bayesian Optimization such as optuna can make it easier. They do not replace the human experimenter!

Do not grid scan hyperparameters (except when there is only one). Usually they have vastly different levels of importance.

Hands-on #3: classify MNIST

Understanding deep learning

- neural tangent kernel: useful to study infinite width neural nets -> ensemble of infinite-width neural nets converges to Gaussian process



Understanding deep learning

- neural tangent kernel: useful to study infinite width neural nets -> ensemble of infinite-width neural nets converges to Gaussian process

For practical large-scale problems, the neural tangent kernel seems to be more of theoretical interest. It cannot, as of yet, replicate the performance of real-world neural nets.

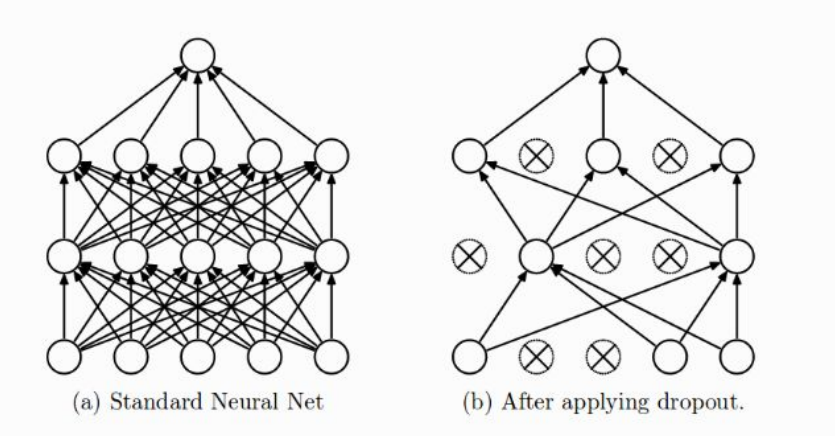
But we'll see what the future holds…

Understanding deep learning

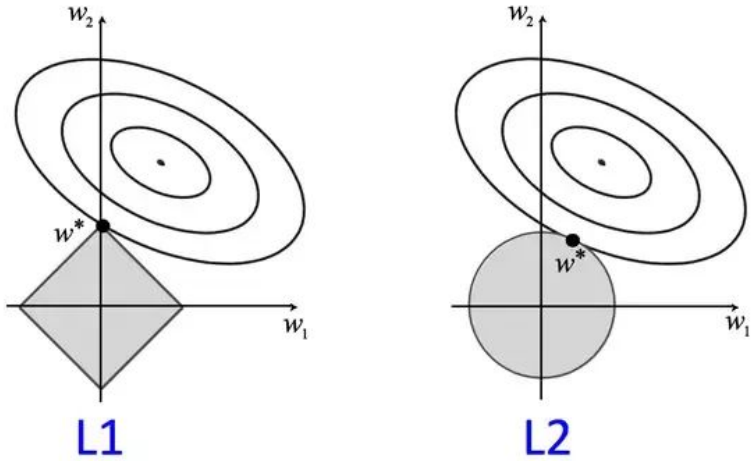
- thermodynamics: connection between neural nets and critical phenomena

A) Dropout, L1/L2/… regularization

Motivation : Regularization -> improve model generalization and prevent overfitting

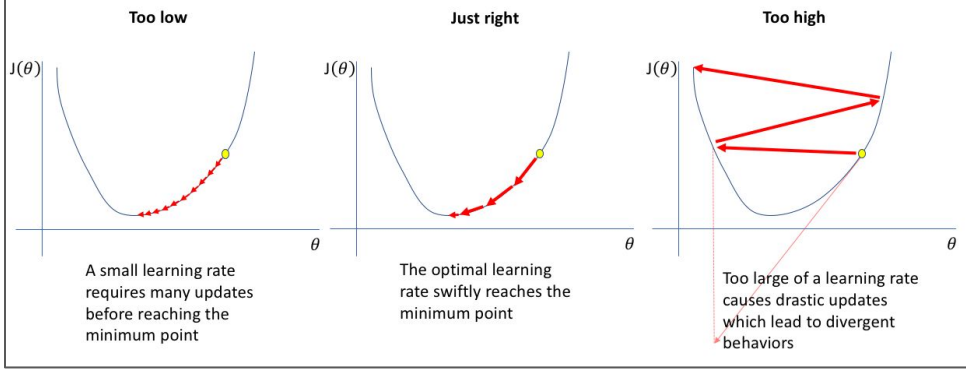


link



https://miro.medium.com/v2/resize:fit:602/0*IgQLm-qC3bBLMHc5.png

B) learning rate scheduling



Motivation: Hyperparameter— scales the magnitude of the network's weight updates in order to minimize the network's loss function

```
new_weight = existing_weight — learning_rate * gradient
```

Intuition: Avoid local minima, improve speed of convergence, avoid overfitting

When to use which technique: if data benefit from changing the learning rate for example, a time series data which can benefit from smaller LR with late time training
Step decay: where the learning rate is reduced by some percentage after a set number of training epochs.

Papers: Not sure: http://arxiv.org/pdf/1206.5533v2

C) advanced optimizers: Adam, …

- Motivation: Try to minimise a loss function without getting stuck in local minima
- Intuition: Move against the direction of the gradient
- Problem: Real loss surface is very complicated
- Technique: Many optimizers have been suggested
- Papers: https://arxiv.org/abs/1910.05446 for empirical comparisons of optimizers
- Visualization/plots: [movie]

SGD(H_t, η_t)

$\theta_{t+1} = \theta_t - \eta_t \nabla \ell(\theta_t)$

Momentum(H_t, η_t, γ)

$v_0 = 0$

$v_{t+1} = \gamma v_t + \nabla \ell(\theta_t)$

$\theta_{t+1} = \theta_t - \eta_t v_{t+1}$

Nesterov(H_t, η_t, γ)

$v_0 = 0$

$v_{t+1} = \gamma v_t + \nabla \ell(\theta_t)$

$\theta_{t+1} = \theta_t - \eta_t \left(\gamma v_{t+1} + \nabla \ell(\theta_t)\right)$

RMSProp$(H_t, \eta_t, \gamma, \rho, \epsilon)$

$v_0 = 1, m_0 = 0$

$v_{t+1} = \rho v_t + (1-\rho)\nabla \ell(\theta_t)^2$

$m_{t+1} = \gamma m_t + \frac{\eta_t}{\sqrt{v_{t+1} + \epsilon}} \nabla \ell(\theta_t)$

$\theta_{t+1} = \theta_t - m_{t+1}$

Adam$(H_t, \alpha_t, \beta_1, \beta_2, \epsilon)$

$m_0 = 0, v_0 = 0$

$m_{t+1} = \beta_1 m_t + (1-\beta_1)\nabla \ell(\theta_t)$

$v_{t+1} = \beta_2 v_t + (1-\beta_2)\nabla \ell(\theta_t)^2$

$b_{t+1} = \frac{\sqrt{1-\beta_2^{t+1}}}{1-\beta_1^{t+1}}$

$\theta_{t+1} = \theta_t - \alpha_t \frac{m_{t+1}}{\sqrt{v_{t+1}} + \epsilon} b_{t+1}$

NAdam$(H_t, \alpha_t, \beta_1, \beta_2, \epsilon)$

$m_0 = 0, v_0 = 0$

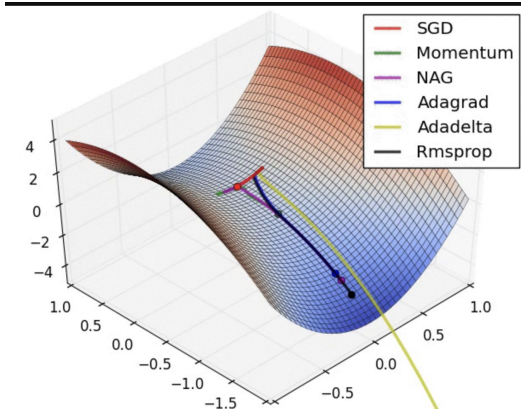
$m_{t+1} = \beta_1 m_t + (1-\beta_1)\nabla \ell(\theta_t)$

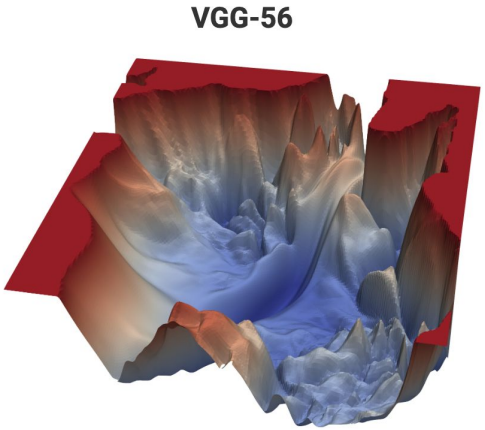
$v_{t+1} = \beta_2 v_t + (1-\beta_2)\nabla \ell(\theta_t)^2$

$b_{t+1} = \frac{\sqrt{1-\beta_2^{t+1}}}{1-\beta_1^{t+1}}$

$\theta_{t+1} = \theta_t - \alpha_t \frac{\beta_1 m_{t+1} + (1-\beta_1)\nabla \ell(\theta_t)}{\sqrt{v_{t+1}} + \epsilon} b_{t+1}$

Loss surface





D) Layer and Batch normalization

Layer normalization

Motivation: speed up the convergence

Problem: In a given layer, each neuron processes different inputs. These inputs can vary widely, especially during training.

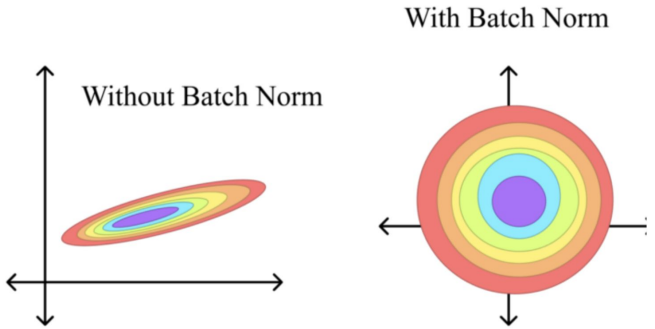
LN computes the mean and standard deviation across all features for a given layer.

Intuition: LN is better than BN. Because of less consumption of memory

When to use LN works on individual samples and is well-suited for sequences with varying lengths; recurrent neural networks (RNNs) and transformer-based architectures

Batch normalization (BN):
normalizes the input across all features for a single batch.
Motivation: control the varying magnitudes of variables, stabilizing training and improving convergence.

Papers: Ioffe and Szegedy, 2015



E) Parameter initialization

- ## Motivation:
  Initializing the parameters of a deep neural network is an important step in the training process, as it can have a significant impact on the convergence and performance of the model
  To avoid locally optimal solution (bad init → 'exploding or vanishing gradients' → optimization failure)

- ## Techniques:
  Zero Initialization, Random Initialization Xavier Initialization (Glorot Initialization),
  He Initialization, Orthogonal Initialization, Uniform Initialization

- ## Links to the original paper:
  Xavier Initialization, https://arxiv.org/abs/1006.5202
  He Initialization, https://arxiv.org/abs/1502.01852
  LeCun Initialization, http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

F) Residual connections

Motivation: Traditional CNN meets a problem that with the depth of the network increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly [1].

Advantage: Training deeper network
Addressing degradation problem

Technique: artificially adding an Identity transformation in the network

When to use: if one want to use deeper networks to address complicated problems but do not want to lose the simple mapping information

Degradation Problem

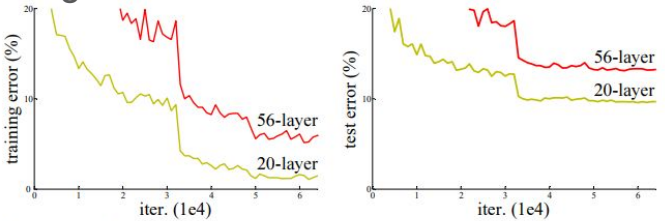
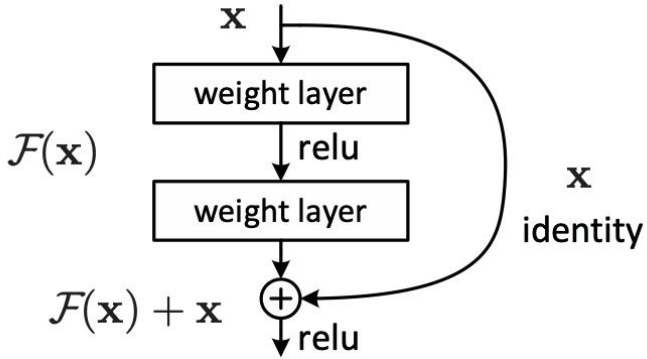


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.



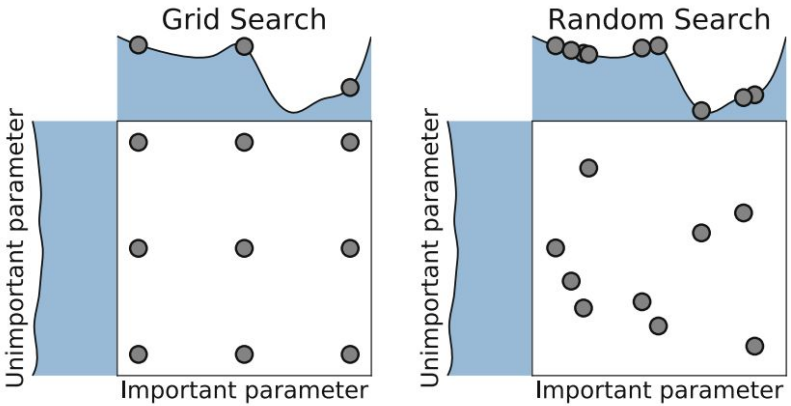
[1] Original paper: Deep Residual Learning for Image Recognition(Kaiming He, et al, 2015)
https://doi.org/10.48550/arXiv.1512.03385

G) Automated hyperparameter optimization

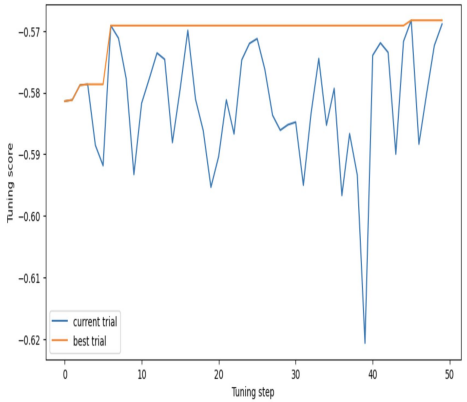
- Examples of hyperparameter : learning rate, batch size, model architecture, etc
- Motivation: Choosing hyperparameter is not always straightforward (i.e., choosing the largest number of hyperparameters compatible is not optimal for certain cases).
- Problem: Some hyperparameters have a more complex interaction with models.

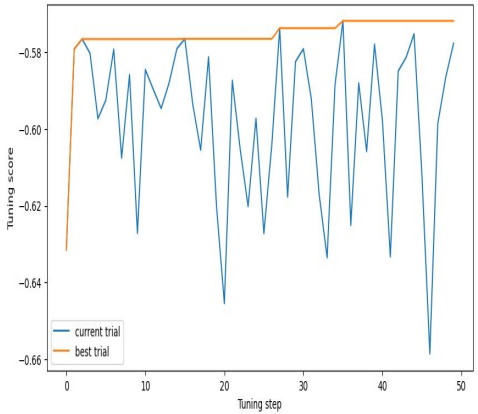
From tensorflow document..

Common Method:



From textbook “Automated Machine Learning”

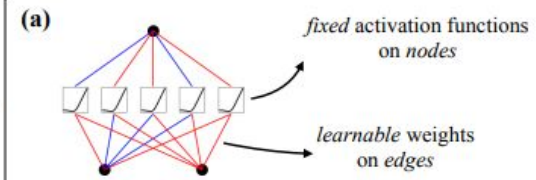
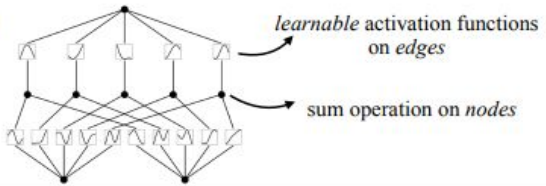
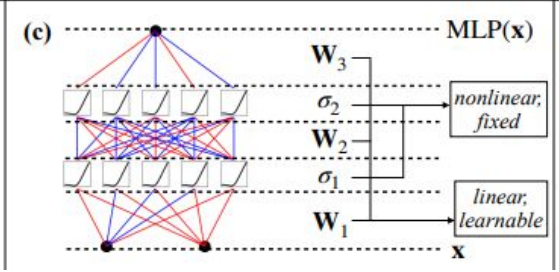
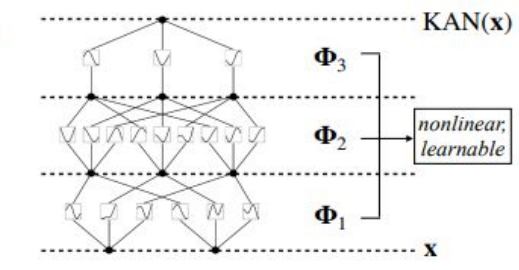




TF Decision Forests model with hyper-parameter tuning using
Left: hyper-parameters to optimize defined **manually**, Test accuracy=0.8722.
Right: set **automatically**, Test accuracy=0.8741,
(The score is effectively minus the log loss. The best trial so far has a score of "-0.40")

https://doi.org/10.1007/978-3-030-05318-5_1

H) Kolmogorov-Arnold networks (KANs)

Model	Multi-Layer Perceptron (MLP)	Kolmogorov-Arnold Network (KAN)
Theorem	Universal Approximation Theorem	Kolmogorov-Arnold Representation Theorem
Formula (Shallow)	$f(\mathbf{x}) \approx \sum_{i=1}^{N(\epsilon)} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$	$f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right)$
Model (Shallow)		
Formula (Deep)	$\mathrm{MLP}(\mathbf{x}) = (\mathbf{W}_3 \circ \sigma_2 \circ \mathbf{W}_2 \circ \sigma_1 \circ \mathbf{W}_1)(\mathbf{x})$	$\mathrm{KAN}(\mathbf{x}) = (\mathbf{\Phi}_3 \circ \mathbf{\Phi}_2 \circ \mathbf{\Phi}_1)(\mathbf{x})$
Model (Deep)		

Liu, Ziming, et al. "Kan: Kolmogorov-arnold networks." *arXiv preprint arXiv:2404.19756* (2024). https://arxiv.org/abs/2404.19756

- **Motivation** :
  Kolmogorov-Arnold representation theorem [6, 7, 8]
- **Intuition** :
  Alternatives to Multi-Layer Perceptrons (MLPs)
- **Specific features**:
  KANs have learnable activation functions on edges. ("weights")
  (no linear weights)
  Every weight parameter is replaced by a univariate function parameterized as a spline.
  KANs have better interpretability than MLPs in some cases.
- **Advantage** :
  Better accuracy in data fitting and PDE solving.