Intro to ML II
SLAC Summer Institute

Lukas Heinrich, TUM
Supervised Learning
What we need

We have the general ingredients for learning

We need to now formulate the actual objectives for the tasks we’re interested in

Let’s start with supervised learning
Latent Concepts

We interpret the real world data we perceive / measure to be a realization of an “underlying concept”

We observe the data but the concept is “latent”

**latent**  | ˈleɪt(ə)nt |
---|---|
adjective
(of a quality or state) existing but not yet developed or manifest; hidden or concealed: they have a huge reserve of latent talent.
Latent Concepts

We interpret the real world data we perceive / measure to be a realization of an “underlying concept”

concept: “cat”

realization: pixel values in image
Latent Concepts

We interpret the real world data we perceive / measure to be a realization of an “underlying concept” (or “label”)

concept: “true weight”

realization: reading on the scale
In statistical learning, we assume that concept $z$ and realization $x$ are linked through a conditional probability:

$$x \sim p(x | z)$$

$z = \text{cat}$

**Latent Concepts**

many realizations  

true value
In statistical learning, we assume that concept $z$ and realization $x$ are linked through a conditional probability:

$$x \sim p(x \mid z)$$

$z = 2.50$ kg

Latent Concepts

2.53 kg  2.51 kg
2.56 kg  2.47 kg

many realizations  true value
Inference

Classic Goal in Statistics: try to find out (“infer”) the latent values given the observed values, i.e. the data $x$

Bayes’ Theorem

$$p(z \mid x) = \frac{p(x \mid z)p(z)}{p(x)}$$
We name inference based on the type of the latent variable

Inference by another name

“cat”

$z \in \{z_0, z_1 \ldots z_n\}$

finite set = “Classification”

$z \in \mathbb{R}$

real values: “Regression”
To do **standard statistics**, we’d need to know what the true data-generating process is $p(x \mid z), p(z)$, but we don’t!

We want:

$$p(\text{animal} \mid \text{image})$$

But we don’t even have:

$$p(\text{image} \mid \text{animal}) = ?$$

$$p(\text{animal}) = ?$$
Apply “learning as search”. If we don’t know \( p(z \mid x) \) maybe we can approximate it?

Look for the best candidate family of candidate distributions \( q_{\phi}(z \mid x) \)

“Variational Inference”
To define a family of distributions, we can use

- well-known densities (e.g. Gaussian, …)
- compute parameters as functions of data: $f_\phi(x)$

$\rightarrow q_\phi(z \mid x) = q(z \mid \theta = f_\phi(x))$
Examples

**Distribution Type**

- **Gaussian** \( \mathcal{N}(z | \mu, \sigma) \)
- **Bernoulli** \( \text{Bern}(z | \theta) \)
- **Categorical** \( \text{Cat}(z | \{p_1 \ldots p_n\}) \)

**Parameters as function of data**

**Mean**

\[
\mu = f_\phi(x) \quad \log \sigma = g_\phi(x)
\]

**Variance**

**Probability of \( z=1 \)**

\[
\{p_i\}, \text{ s.t. } \sum_i p_i = 1
\]
Soft Perceptron was our first example

\[ q_{w,b}(z \mid x) = \text{Ber}(z \mid \sigma(wx + b)) \]

iso-contours

\[ w \cdot x + b = \text{const} . \]
Our goal is to approximate $p(z \mid x)$. Intuitively, we some notion of **distance between distributions** $d(p, q_\phi)$

Learning as minimization of that distance

$\phi^* = \text{argmin}_\phi d(p, q_\phi)$
Distributions are extended objects, not single points. Distances between Distributions

\[ D_{KL}(p \| q) = \int dx \, p(x) \log \frac{p(x)}{q(x)} \]

A common choice: 

“KL Distance”

same distance in means but which pair is “closer”?
So a **Natural Objective**: get good inference performance across all the possible data we might encounter.

I.e. minimize:

\[
L(\phi) = \mathbb{E}_{p(x)}D_{KL}(p(z|x) \mid \mid q_\phi(z|x))
\]

**A natural objective**

**So a Natural Objective**: get good inference performance across all the possible data we might encounter.

I.e. minimize:

\[
L(\phi) = \mathbb{E}_{p(x)}D_{KL}(p(z|x) \mid \mid q_\phi(z|x))
\]
A natural objective

Ok, but it seems like to compute the objective we already need to know the answer?

\[ L(\phi) = \mathbb{E}_{p(x)} D_{KL}(p(z \mid x) \mid \mid q_{\phi}(z \mid x)) \]

\[ = \mathbb{E}_{p(x)} \mathbb{E}_{p(z \mid x)} \log \frac{p(z \mid x)}{q_{\phi}(z \mid x)} \]
A natural objective

Amazingly it all drops out! In the end, we get something we can estimate purely from data pairs \((x_i, z_i)\) - which we have

\[
L(\phi) = \mathbb{E}_x \mathbb{E}_{p(z|x)} \log \frac{p(z|x)}{q_\phi(z|x)} = - \mathbb{E}_{p(x,z)} \log q_\phi(z|x)
\]

This is called the “Cross-Entropy” Loss and is most used for supervised learning
Alternative Names

CE sometimes hides under another name / formula:

**Gaussians:** “Mean Squared Error” (MSE)

\[
\mathbb{E}_{p(x,z)}(z - \mu_\phi(x))^2
\]

**Binary Classification:** “Binary Cross Entropy” (BCE)

\[
\mathbb{E}_{p(x,z)}[z \log \theta_\phi(x) + (1 - z)\log(1 - \theta_\phi(x))]
\]
Output Activations

For UFA the type of non-linearity was irrelevant, now we need to at least be careful with the output activation.

**Regression**

\[ \mu_\phi(x) \in \mathbb{R} \]

No activation!

**Binary Classification**

\[ \theta(x) \in [0,1] \]

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

**Multi-class Classification**

\[ p_i(x) \geq 0 \text{ s.t. } \sum p_i = 1 \]

\[ \text{softmax}(x) = \frac{e^{x_i}}{\sum_i e^{x_i}} \]
Deep Learning
Inductive Bias & Friends
You might be wondering

So far, we’ve not mentioned deep learning at all. Do we need it at all if we have universal function approximators?

You might also heard about “neural network” architectures (Convolutional Nets, Recurrent Nets, Graph Nets)

Where is everybody?
Two issues with shallow networks

Shallow Networks are great (universal, even!), but there are two issues:

• to *actually* model complex functions you need a lot of neurons, and i.e. parameters

• sometimes we know quite a bit about our target function, should we really search in a *universal space*?
Deep Learning

A lot of classic machine learning was done on highly preprocessed data ("engineered features") didn’t require (and couldn’t afford) very complex hypothesis spaces
Deep Learning

More ambitious: why rely on human-provided features? Can we learn the features from the raw data?
Deep Learning Considerations

To pull this off you will need

- **much more complex functions** for e.g. pixels → cat → bigger hypothesis sets

- sufficient amount of data to be able to afford them
  → remember bias variance tradeoff

- … or an affective way too constrain the search space
  → inductive bias
Growing Neural Networks

We could’ve grown neural networks not only by making them wider, but also deeper.

Both add parameters, but what about the actual functions they can approximate?
Benefits of Depth

Deep Networks are much more effective at approximating complex functions.
Deep Learning

The assumption is that similar to us, effective machine-learned reasoning goes through many layers of abstraction.
Deep Learning

We do see this, but care is needed to not overinterpret this...

ML system has 99% confidence that this is a magpie... an actual magpie
Gradients of Deep Programs
Gradient Descent needs... Gradients!

As neural networks become bigger & deeper, need to find a way to compute them efficiently

\[ f(x) = (t \circ h \circ g)(x) = t(h(g(x))) : \mathbb{R}^4 \rightarrow \mathbb{R}^3 \]
Gradient Descent needs… Gradients!

We want to compute the Jacobian of the deep composition of functions. **But a naive approach scales badly**

\[
\frac{\partial f}{\partial x} = \frac{\partial t}{\partial g} \frac{\partial g}{\partial h} \frac{\partial h}{\partial x}
\]
Matrix-Free Computation

Instead of Matrix-Matrix products, we can compute more cheap vector-Matrix products and compute a row at a time.
Matrix-Free Computation

Instead of Matrix-Matrix products, we can compute more cheap vector-Matrix products and compute a row at a time
Matrix-Free Computation

Instead of Matrix-Matrix products, we can compute more cheap vector-Matrix products and compute a row at a time.
Matrix-Free Computation

Instead of Matrix-Matrix products, we can compute more cheap vector-Matrix products and compute a row at a time.
Matrix-Free Computation

Instead of Matrix-Matrix products, we can compute more cheap vector-Matrix products and compute a row at a time.
Matrix-Free Computation

Instead of Matrix-Matrix products, we can compute more cheap vector-Matrix products and compute a row at a time.
Automatic Differentiation

The approach can be generalized to arbitrary computational graphs: **The Backpropagation Algorithm**

\[
\begin{align*}
    g^z & \quad J_{y \rightarrow z} = \frac{\partial z_i}{\partial y_j} \\
    g^y & \quad J_{x \rightarrow y} = \frac{\partial y_i}{\partial x_j} \\
    g^y & \quad J_{y \rightarrow z} = \frac{\partial z_i}{\partial y_j} \\
    g^y & \quad J_{x \rightarrow y} = \frac{\partial y_i}{\partial x_j} \\
    g^y & \quad \frac{\partial L}{\partial y} = \sum_{c \in \text{children}(y_i)} g^z \frac{\partial z_c}{\partial z_i}
\end{align*}
\]
Putting it all Together
ML Frameworks like PyTorch, Tensorflow, JAX put a lot of the pieces together to provide a performance setup.
A full training Loop

Data $s \sim p(s)$

```
def learn(samples):
    features, labels = samples
    model = MyModel()
    loss_func = torch.nn.BCELoss()
    opt = torch.optim.Adam(
        model.parameters(), lr = 1e-3
    )

    for i in range(steps):
        predictions = model(samples)
        loss = loss_func(predictions, labels)
        loss.backward()
        opt.step()
        opt.zero_grad()

    return model
```
Inductive Bias & Architectures
Beyond Depth

Can we push this further, should we move away from universal function approximators?

- bias variance tradeoff: reduce $\mathcal{H}$ as much as you can

General Idea: $\mathcal{H}$ should match data modality & task
Inductive Bias

If we can throw out irrelevant functions, which we know can’t be the solution, we **bias** our inductive process towards good solution (here: bias is good)
The Architecture Zoo

CNN

GNN

Transformer

Deep Sets

RNN
Convolutional Neural Nets

Convolutional Neural Networks are (approximately) translationally invariant

Is there a cat in the picture? How about now?

One of the early successes of deep learning in the 80s
Convolutions

Two key ideas lead to convolutions as a building block

• local connectivity and weight sharing

Standard Linear Layer

\[ y_i = \sum_{j=0}^{N} w_{ij} x_j \]

Convolution

\[ y_i = \sum_{o=-1,0,1} w_o x_{i+o} \]
Convolutions

Equivalent to a filter that slides across the input
2D Convolutions

We can extend this idea to higher dimensions:
2D Convolutions

We can extend this idea to higher dimensions:
2D Convolutions

We can extend this idea to higher dimensions:
2D Convolutions

We can extend this idea to higher dimensions:
Local Pattern Detectors

The filters are like mini-neural nets extracting features for a local patch: e.g. edges, curves, texture, …
Convolutional Layers

To build up networks, we can extract many features with multiple kernels:

-1 0 1 0 1 0
-1 0 1 1 1 1
-1 0 1 0 1 0
-1 -1 -1 0 0 0
0 0 0 0 -1 0
1 1 1 0 0 0

Input

Output

input channels: 1
output channels: 4
kernel size = 3
stride = 1
padding = 1
Pooling

Role of convolutions is to **extract local features**. Pooling summarizes a local patch in terms of those features.

**Average Pooling**

\[ y = \frac{1}{N} \sum_{v \in \text{view}(y)} x_v \]

**Maximum Pooling**

\[ y = \arg\max_{v \in \text{view}(y)} x_v \]
Building CNNs

The full CNN then implements the Deep Learning idea: learned feature extraction, followed by simple MLP head
Graph Neural Nets

CNNs excel at data that “lives” on a regular grid. Feature extraction by combining information from neighborhood.

But a lot of data is more irregular.

A local neighborhood defined more by relationships than a grid.
Graphs can still be represented by Matrices, but the processing of graphs must not rely on (arbitrary) order.
Graph Data

Graphs can still be represented by Matrices, but the processing of graphs must not rely on (arbitrary) order

\[ X \in \mathbb{R}^{n \times f} \]

Permutation Invariance
Graph Convolutions generalize CNN convolutions to pass messages from neighbors as defined in the graph.

\[
\begin{align*}
S &= XW
\end{align*}
\]
GNNs

As in CNNs, we can stack Graph processing into a stack of feature extraction, and then follow up high-level “head”
Dynamic Networks

Neural Nets are functions of the input & network parameters

\[ f(x, \phi) \]

In their most basic form, both inputs are static, but some of the most powerful architectures allow them to be dynamic
Often data is variable in size. Recurrent neural networks deal with it like a computer:

Consume data **step-by-step** and keeping the information within a memory component.
Recurrent Neural Networks

RNNs learn an update function for a memory vector, which can be applied many times until the inputs are exhausted.
Active Neurons and Gates

RNNs interact with memory like a “soft” computer reading and writing to memory via “gates”, i.e. multiplicative masks.

Hard Read

Soft Read
Recurrent Neural Networks

Can be applied to arbitrary length sequences
RNN in a time before ChatGPT

Cell sensitive to position in line:

The sole importance of the crossing of the Berezina lies in the fact
that it plainly and indubitably proved the fallacy of all the plans for
cutting off the enemy's retreat and the soundness of the only possible
line of action--the one Kutuzov and the general mass of the army
demanded--namely, simply to follow the enemy up. The French crowd fled
at a continually increasing speed and all its energy was directed to
reaching its goal. It fled like a wounded animal and it was impossible
to block its path. This was shown not so much by the arrangements it
made for crossing as by what took place at the bridges. When the bridges
broke down, unarmed soldiers, people from Moscow and women with children
who were with the French transport, all--carried on by vis inertiae--
pressed forward into boats and into the ice-covered water and
surrendered.

Cell that turns on inside quotes:

"You mean to imply that I have nothing to eat out of... on the
contrary, I can supply you with everything even if you want it for
dinner parties," warmly replied Chichagov, who tried by every
spoke to prove his own rectitude and therefore imagined Kutuzov
animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle per
smile: "I meant merely to say what I said."

Cell that robustly activates inside if statements:

```c
static int _dequeue_signal(struct sigpending *pending, sigset_t *info)
{
    int sig = next_signal(pending, mask);
    if (sig == 0) {
        if(current->notifier) {
            if(sigsismember(current->notifier_mask, sig)) {
                if((current->notifier)(current->notifier_data)) {
                    clear_thread_flag(TIF_SIGPENDING);
                    return 0;
                }
            }
        }
    }
    collect_signal(sig, pending, info);
    return sig;
}
```

The Unreasonable Effectiveness of Recurrent Neural Networks

May 21, 2015

There’s something magical about Recurrent Neural Networks (RNNs). I still remember when I trained my first recurrent network for Image Captioning. Within a few dozen minutes of training my first baby model (with rather arbitrarily-chosen hyperparameters) started to generate very nice looking descriptions of images that were on the edge of making sense. Sometimes the ratio of how simple your model is to the quality of the results you get out of it blows past your expectations, and this was one of those times. What made this result so shocking at the time was that the common wisdom was that RNNs were supposed to be difficult to train (with more experience I’ve in fact reached the opposite conclusion). Fast forward a year: I’m training RNNs all the time and I’ve witnessed their power and robustness many times, and yet their magical outputs still find ways of amusing me. This post is about sharing some of that magic with you.

We’ll train RNNs to generate text character by character and ponder the question “how is that even possible?”

By the way, together with this post I am also releasing code on GitHub that allows you to train character-level

http://karpathy.github.io/2015/05/21/rnn-effectiveness/
Attention Mechanism

The notion of gating / dynamically controlling the flow of information is also key to one of the most impactful ideas in Deep Learning: **Attention**
Attention Mechanism

Standard Neural Nets, have globally fixed data processing
Attention Mechanisms add a **data-dependent** processing

\[ y = Wx \quad \rightarrow \quad y = A(x)x \]

**Standard Neural Net**
- weights are fixed by the training data
- input is just passed through

**Network with attention**
- input data influences the weights at the time of processing
Attention in Language

Example: when representing words with added context, decide dynamically which other words are relevant.
Attention through Constraints

Attention uses a constraint to force focus. Attention weights must add up to unity.

\[ A_{ij}(x) = \text{softmax} W_{ij}(x) \]

“Jack of all trades” - some attention everywhere

decide to drop focus at edges, focus attention at the center

Focus does not mean saying yes, it means saying no.

- Steve Jobs
Attention is *the* key idea in transformer networks that have e.g. largely replaced RNN-based models for text.
Physicists were quick to add their own symmetries to inject physics inductive bias to neural networks.