A Gentle Introduction to Machine Learning

First Lecture

Originally created by Olov Andersson
Revised and lectured by Yang Liu

LINKÖPINGS UNIVERSITET

1

Outline of Machine Learning Lectures

- Introduction to machine learning (two lectures)
  - Supervised learning, unsupervised learning (very brief)
  - Reinforcement learning
- Recent Advances: Deep learning (one lecture)
  - Applied to both SL and RL above
  - Examples

2

What is Machine Learning about?

- To enable machines to *learn and adapt* without programming them
- Our only frame of reference for learning is from biology
  - …but brains are hideously complex, the result of ages of evolution
- Like much of AI, Machine Learning mainly takes an **engineering approach**[1]
  - Remember, humanity didn't master flight by just imitating birds! ☺

1. Although there is occasional biological inspiration

3

Theoretical Foundations

Mathematical foundations borrowing from several areas

- Statistics (theories of how to **learn from data**)
- Optimization (how to **solve** such learning problems)
- Computer Science (efficient **algorithms** for this)

This intro lecture will focus more on **intuitions** than mathematical details

ML also **overlaps** with multiple areas of engineering, e.g.

- Computer vision
- Natural language processing (e.g. machine translation)
- Robotics, signal processing and control theory

...but traditionally differs by focusing more on **data-driven** models and AI

4

Why Machine Learning

- Difficulty in **manually programming** agents for every possible situation
- The world is ever **changing**, if an agent cannot adapt, it will fail
- Many argue learning is required for Artificial **General** Intelligence (AGI)
- We are still far from human-level general learning ability…
  - …but the algorithms we have so far have shown themselves to be useful in a wide range of applications!
  - Using just data, recent "deep learning" approaches can come *near* human performance on many problems, but *near* may not always be sufficient

2021-09-23 5

5

When Is Machine Learning Useful Today?

- While **not as data-efficient** as human learning, once an AI is "good enough", it can be cheaply duplicated
- Computers work **24/7** and you can usually **scale** throughput by piling on more of them

Software Agents (Apps and web services)

- Companies collect ever more data and processing power is cheap ("*Big data*")
- Can **let an AI learn how to improve business**, e.g. smarter product recommendations, search engine results, ad serving, to decision support
- Can **sell services that traditionally required human work**, e.g. translation, image categorization, mail filtering, content generation…?

Hardware Agents (Robotics)

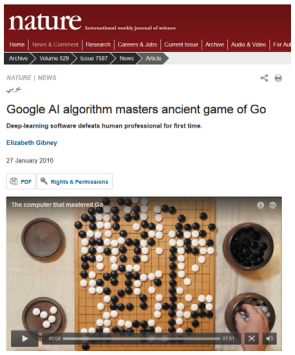
- Although data is more extensive, many capabilities that humans take for granted like **locomotion, grasping, recognizing objects**, **speech** have turned out to be **difficult to manually construct rules** for.

2021-09-23 6

6

Example – Google Deepmind's Go Agent

However, in **narrow applications** machine learning can rival or beat human performance.



2021-09-23 7

7

Example – Stanford Helicopter Acrobatics

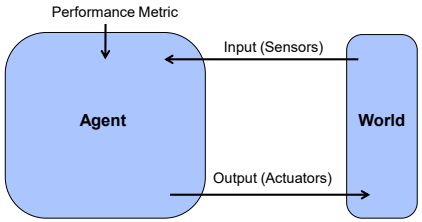
However, in **narrow applications** machine learning can even rival or beat human performance. This work was done over a decade old but still astonishing.

8

To Define Machine Learning

Given a task, mathematically encoded via some performance metric, a machine can improve its performance by learning from experience (data)

From the agent perspective:



To Define Machine Learning

- Arthur Samuel (1959). Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.
- Tom Mitchell (1998) Well-posed Learning Problem: A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.
- Suppose your email program watches which emails you do or do not mark as spam, and based on that learns how to better filter spam.
  - Experience E is Watching you label emails as spam or not spam.
  - Task T is Classifying emails as spam or not spam.
  - Performance P is The number (or fraction) of emails correctly classified as spam/not spam.

The Three Main Types of Machine Learning

Machine learning is a young science that is still changing, but traditionally algorithms are divided into three types depending on their purpose.

- **Supervised Learning**
- **Reinforcement Learning**
- **Unsupervised Learning**

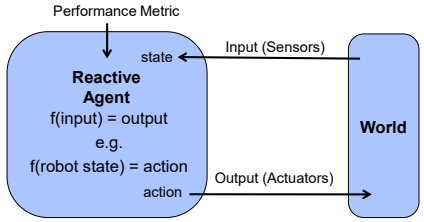
Supervised Learning at a Glance

In **supervised learning**

- Agent has to learn from **examples of *correct*** behavior
- Formally, **learn an unknown function f*(x) = y*** given examples of (x, y)
- Performance metric: **Loss** (difference) between learned function and correct examples
- Typically classified into:
  - Regression: Predict continuous valued output
  - Classification: Discrete valued output

Supervised Learning – Agent Perspective

Representation from agent perspective:



…but it can also be used as a component in other architectures

Supervised Learning is surprisingly powerful and ubiquitous

Some real world examples

- **Spam filter**: f(mail) = spam?
- **Graphics upscaling**: f(pixels) = pixels

2021-09-23 13

13

Supervised Learning of "Super Resolution"

- Learn y=f(x) from examples (x,y),…
  - x = "low-res image", y = "high-res image" (real numbers)
  - Given a new low-res image x' below, predict y':

- Similar technique ships with NVIDIA graphics cards Deep Learning Super Sampling (DLSS)

2021-09-23 14

14

Reinforcement Learning at a Glance

In **reinforcement learning**

- World may have **state** (e.g. position in maze) and be **unknown** (how does an action change the state)
- In each step the agent is only given **current state** and **reward** instead of examples of correct behavior
- Performance metric is sum of **rewards over time**
- Combines learning with a **planning** problem
  - Agent has to **plan a sequence of actions** for good performance
- The agent can **even learn on its own** if the reward signal can be mathematically defined

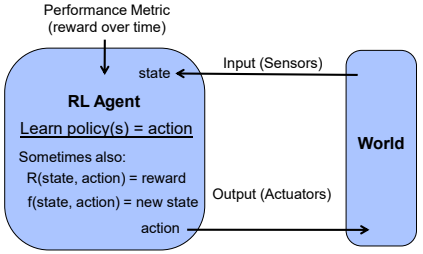
2021-09-23 15

15

Reinforcement Learning at a Glance II

RL is based on a utility (reward) maximizing agent framework

- Agent learns *policy* (plan function) to maximize reward over time
- Either learn intermediate models of the effect of actions (next state, reward) from state s, or use *model-free* approaches



Real world examples – Robot Behavior, Game Playing (AlphaGo…)

2021-09-23 16

16

Demo – Supervised vs. Reinforcement Learning for Robot Behavior

- Learning to flip pancakes, ”supervised” and reinforcement learning (reward not shown).

17

17

Unsupervised Learning at a Glance

In **unsupervised learning**

- Neither a correct answer/output, nor a reward is given
- Task is to **find some structure** in the data
- Performance metric is some **reconstruction** error of patterns compared to the input data distribution

Examples:

- **Clustering** – When the data distribution is confined to lie in a small number of “clusters” we can find these and use them instead of the original representation, e.g. bigger recommender system (news, ads, etc.)
- **Dimensionality Reduction** – Finding a suitable lower dimensional representation while preserving as much information as possible, e.g. image/video compression

Recent trend: Found structure can be used to **generate new data (content)!**

2021-09-23 18

18

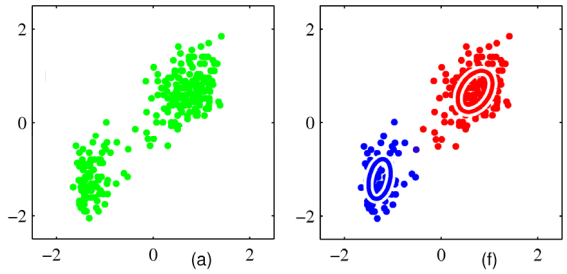
Unsupervised Learning at a glance II

- Not directly applicable to the agent perspective as there is no clear way to encode a goal or behavior
- However, the techniques can be useful as a **preprocessing step** in other learning approaches
  - If fewer dimensions or a few clusters can accurately describe the data, big **computational wins** can be made
- It is also frequently used for **visualization** as smaller representations are easier to visualize on a computer screen
- To keep this brief, we will not go into any further detail on unsupervised learning

2021-09-23 19

19

Unsupervised Learning Example: Clustering – Continuous Data



Two-dimensional continuous input

(Bishop, 2006)

2021-09-23 20

20

Unsupervised example

- Original faces were down sampled to save space but still remain majority features.

(Deep) Unsupervised Learning – Do AI's dream? ☺

- Generative model ("Dream up" new data) fed e.g. images...
- Can we use them to e.g. fill in scenery in a movie scene?

(Karras et al, 2018) https://youtu.be/G06dEcZ-QTg

(Deep) Unsupervised Learning – Do AI's dream? ☺

- Generative model based on Text-Image data
- Future applications in content generation?

volcano

(Nguyen et al, 2017)
https://youtu.be/ePUIJMtclcY

Outline of Supervised Learning

Today we will focus on **Supervised Learning**

- Definition
- Fundamentals: Features, Models, Loss (or cost) Functions, Training
- Linear Models
- Neural Networks
  - Trend: Deep Learning (more in third ML lecture)
- Pitfalls and Limitations (if time permits)

Formalizing Supervised Learning

Remember, in Supervised Learning:
- Given tuples of ***training data*** consisting of (**x**,y) pairs
- The objective is to learn to **predict** the **output** y' for a new input **x'**

Formalized as **searching** for approximation to **unknown function** y = ***f(x)***, given N examples of **x** and y: $(\mathbf{x}_1,y_1), \ldots ,(\mathbf{x}_n,y_n)$

Two major classes of supervised learning
- **Classification** – Output are **discrete** category labels
  - Example: Detecting disease, y = "healthy" or "ill"
- **Regression** – Output are **numeric** values
  - Example: Predicting temperature, y = 15.3 degrees

In either case, input data $\mathbf{x}_i$ could be **vector valued** and **discrete**, **continuous** or **mixed**. Example: $\mathbf{x}_1$ = (12.5, "cat", true).

2021-09-23 25

25

Classical Supervised Learning in Practice

Can be seen as **searching** for an approximation to unknown function y = ***f(x)*** given N examples of **x** and y: $(\mathbf{x}_1,y_1), \ldots ,(\mathbf{x}_n,y_n)$

Want the algorithm to **generalize** from **training** examples to new inputs **x'**, so that y'=f(**x'**) is "close" to the correct answer

1. An **input "feature" vector** $\mathbf{x}_i$ of examples is constructed by **mathematically encoding** relevant problem data
   - Examples of such $(\mathbf{x}_i, y_i)$ make up the **training set**
2. A *model (or hypothesis)* for *f(x)* is selected with some parameters
3. A *loss* function is selected that defines "closeness" to correct answers
4. The model is **trained** on the examples by searching for its **parameters** that minimize loss on the training set (i.e. are "close" to unknown *f(x)*)

2021-09-23 26

26

Feature Vector Construction

Want to learn ***f(x)*** = *y* given N examples of **x** and y: $(\mathbf{x}_1,y_1), \ldots ,(\mathbf{x}_n,y_n)$

Most standard algorithms work on **real number variables**
- If inputs ***x*** or outputs y contain categorical values like "book" or "car", we need to encode them with numbers
  - With only two classes we get y in {0,1}, called **binary classification**
  - Classification into multiple classes can be reduced to a sequence of binary one-vs-all classifiers
- The variables may also be structured as **text**, **audio**, **image** or **video** data

Finding a suitable feature representation **can be non-trivial**, but there are standard approaches for the common domains
- With sufficient data, features can also be learned (*deep learning,* later…)

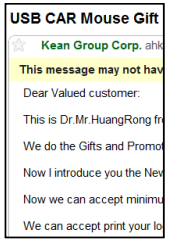
2021-09-23 27

27

Feature Vector Example for Text - Bag of Words

One of the early successes of ML was **learning spam filters**

Spam classification example:



Each mail is an input, some mails are flagged as spam or not spam to create training examples.

Bag of Words Feature Vector:

Encode the existence of a fixed set of relevant **key words** in each mail as the **feature vector**.

$\mathbf{x}_i$ = words_i =

Feature	Exists?
"Customer"	1 (Yes)
"Dollar"	0 (No)
"Fund"	0
"Accept"	1
"Bank"	0
....	...

y_i = 1 (spam) or 0 (not spam)

Simply learn f(x)=y using suitable classifier!

2021-09-23 28

28

Selecting Models: Linear **Regression Example**

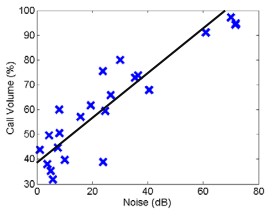
I. Construct a **feature vector** $\mathbf{x}_i$ to be used with examples of y_i
II. Select a model and **train** it on examples (search for a good approximation to the unknown function)

Fictional example: Smartphone app that learns desired ring volume based on examples of volume and background noise level

Feature vector $\mathbf{x}_i$ = (Noise dB), y_i = (Volume %)

- Select the familiy of **linear** functions:
  $$y_i = w_1 \cdot x_i + w_0$$
- **Train** the algorithm by searching for a line that **fits the data well**

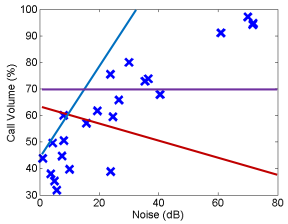
…but how does "training" really work?



Training a Learning Algorithm

Feature vector $\mathbf{x}_i$ = (Noise in dB), outputs y_i = (Volume %)

- Recap: Want to find approximation *h*(x) to the unknown function f(x)
- As an example, let it to be the family of **linear** functions:
  $$y_i = w_1 \cdot x_i + w_0$$
- The model $h_{\mathbf{w}}(x)$ has two **parameters:** $\mathbf{w} = (w_1, w_0)$ (line slope and offset)
- How do we find parameters that result in a **good** approximation h?



Three poor linear hypotheses

Training a Learning Algorithm – Loss Functions

How do we find parameters **w** that result in a **good** approximation $h_{\mathbf{w}}(x)$?

- Need a performance metric for function approximations of unknown *f(x)*
  - **Loss functions** $L(f(x), h_{\mathbf{w}}(x))$
- Minimize deviation against the N example data points from *f(x)*
  - For **regression** one common choice is a **sum square loss** function:
  $$L(f(x), h_{\mathbf{w}}(x)) = (f(x) - h_{\mathbf{w}}(x))^2 = \sum_{i=1}^{N} (y_i - h_{\mathbf{w}}(x_i))^2$$
- Why square loss? Negative difference is as bad as positive
- Search in continuous domains like **w** is known as **optimization**
  - (if unfamiliar, see Ch4.2 Local Search in Continuous Spaces in course book AI: A Modern Approach)

Training a Learning Algorithm – Optimization

How do we find parameters **w** that minimize the loss?

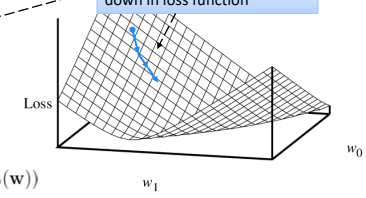
- Optimization approaches iteratively move in the **direction that decreases** the loss function *L(w)*
- Simple and popular approach: **gradient descent**

Initialize **w** to some random point in the parameter space
loop until decrease in loss is **small**
  for each w_j in **w do**

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} L(\mathbf{w})$$

Step size (learning rate)

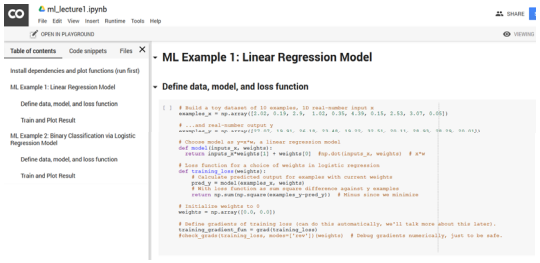
Negative gradient points down in loss function



Note:
$$Gradient = \left(\frac{\partial}{\partial w_0} L(\mathbf{w}), \frac{\partial}{\partial w_1} L(\mathbf{w})\right)$$

Worked Example – Linear Regression

- Google Colab at: http://bit.ly/2maVQKY
  - Run top box to install dependencies (30s), then scroll to ML Example 1
- NOTE: Need to be signed in to a Google account. *Might* need to **save** or download workbook to run it.



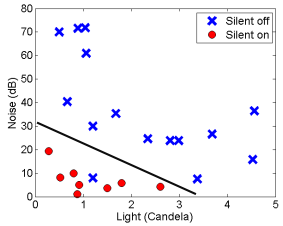
What about categorical outputs (classification)?

I. Construct a **feature vector x_i** to be used with examples of y_i
II. Select a model and **train** it on examples (search for a good approximation to the unknown function)

Fictional example: Smartphone app that learns if silent mode should be on/off at different levels of **background noise** and **light**

Feature vector **x_i = (Noise, Light level)**, y_i = {"silent on", "silent off"}

- Again, can select the familiy of **linear** functions. However, now outputs y have to be **transformed** to the interval [0,1]
- Can classify new inputs according to how close output is to 0 or 1.
- For linear models, the decision boundary will still be a straight line.



Classifier Training – Loss Functions II

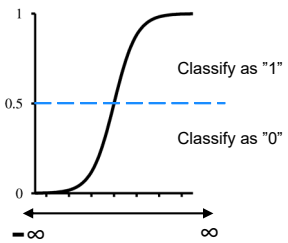
- How to transform standard models to classification?
  - Squared error does not make sense when target output discrete set {0,1}
- Could use custom loss functions for classification
  - Minimize number of missclassifications (unsmooth w.r.t. parameter changes)
  - Maximize information gain (used in decision trees, see book)
  - **However**, requires specialized parameter search methods
- Instead: Make outputs **probabilities [0,1]** by **squashing** predicted **numeric outputs** via sigmoid ("S")

Sigmoid functions allow us to do use **any** regression model with binary classification by def. Pr(y="1"|X) = g(model(x))

Where g is **"logistic" sigmoid** :

$$g(x) = \frac{1}{1+e^{-x}}$$

For >2 classes, use **soft-max** (see book)



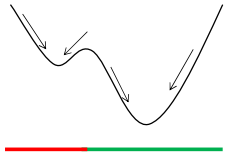
Worked Example – Binary Classification via Linear Logistic Regression

- Same Google Colab as before: http://bit.ly/2maVQKY
  - Run top box to install dependencies (30s), then scroll down to ML Example 2
- NOTE: May need to be signed in to a Google account

Training a Learning Algorithm – Limitations

Limitations

- Local optimization of loss is greedy – Gets stuck in *local* minima unless the loss function is **convex** w.r.t. **w**, i.e. there is **only one minima**.
- **Linear models are convex**, however most more advanced models are **vulnerable** to getting stuck in local minina.
- Care should be taken when training such models by using for example **random restarts** and picking the least bad minima.



If we happen to start in red area, optimization will get stuck in a bad local minima!

Linear Models in Summary

Advantages

- Linear algorithms are **simple and computationally efficient**
  - For both regression and classification
- Training them is a **convex** optimization problem, i.e. one is guaranteed to find the **best** hypothesis in the space of linear hypothesis
- Can be extended by non-linear feature transformations

Disadvantages

- The hypothesis space is very restricted, it cannot handle non-linear relations well

Still **widely used** in applications

- Recommender Systems – Initial Netflix Cinematch was a linear regression, before their **$1 million** competition to improve it. Rather simple and are appropriate for small systems.
- Often a good place to start...
- At the core of many big internet services. Ad systems at Twitter, Facebook, Google etc...

What about models with uncertainty?

Supervised Learning:

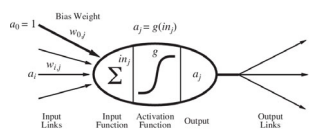
Mathematically, can be seen as finding an approximation to an unknown function y = ***f(x)*** given N examples of **x** and y

Two perspectives:

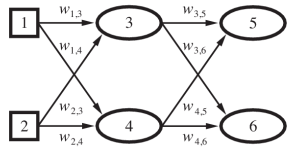
- **Deterministic Models**
  - Search for a suitable function y = h(x)
  - What we have looked at so far, the most common approach
  - Example: In classification something may be either A or B, never in-between, regression gives an exact answer like 15.3
- **Probabilistic Models**
  - Search for a suitable ***probability distribution*** like P(Y|X)
  - When we also want to predict the **uncertainty**
  - Example: P(Y="Healthy"|X) = 0.7 and P(Y="Cancer"|X) = 0.3
  - In a spam filter we might prefer to get a spam too many than to trash that important mail from your boss…

Beyond Linear Models – Artificial Neural Networks

- One **non-linear model** that has captivated people for decades is **Artificial Neural Networks (ANNs)**
- These draw upon inspiration from the physical structure of the brain as an interconnected network of "**neurons**", emitting electrical "spikes" when excited by inputs (represented by non-linear "**activation functions**")



The Neuron



The Network

Artificial Neural Networks – The Neuron

- In (one input) linear regression we used the model:

$$y_i = w_1 \cdot x_i + w_0$$

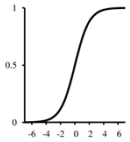
- Each **neuron** in an ANN is a linear model of **all** the inputs passed through a **non-linear** *activation function* g, representing the "spiking" behavior.

$$y = g(\sum_{i=1}^{k} w_i x_i + w_0)$$

- The activation function is traditionally a **sigmoid**, but other options exist

$$g(x) = \frac{1}{1+e^{-x}}$$

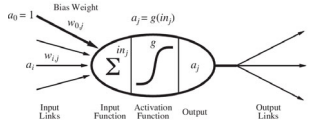
- ANNs generalize logistic linear regression!



Artificial Neural Networks – The Neuron II

- However, there is not just one neuron, but a **network** of neurons!
- Each neuron gets inputs from all neurons in the previous layer.
- We rewrite our neuron definition using a_i for the input, a_j for the output and $w_{i,j}$ for the weight parameters:

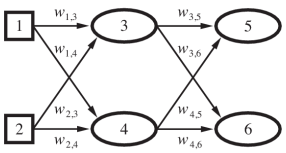
$$a_j = g(\sum_{i=1}^{k} w_{i,j} a_i + w_0)$$



Artificial Neural Networks – The Network

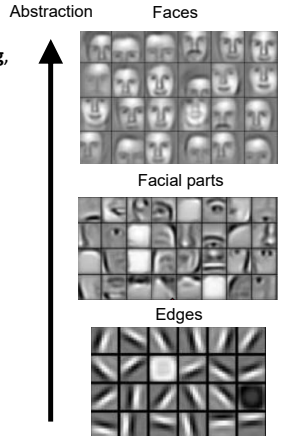
- The networks are composed into **layers**
- In a traditional **feed-forward** and **fully-connected** ANN, all neurons in a layer are connected to all neurons in the next layer, but not to each other
- Expanding the output of a second layer neuron (5) we get

$$a_5 = g(w_{0,5} + w_{3,5}a_3 + w_{4,5}a_4)$$
$$= g(w_{0,5} + w_{3,5}g(w_{0,3} + w_{1,3}x_1 + w_{2,3}x_2) + w_{4,5}g(w_{0,4} + w_{1,4}x_1 + w_{2,4}x_2)))$$



Why Multi-layer Neural Networks?

- Recent surge of successes with **deep learning**, using multi-layer models like ANNs to better capture **layers of abstractions** in data.
- Some tasks are uniquely suited to this like vision, text and speech recognition, where they hold state-of-the-art results.
  - Specialized architectures. More on this later.
- Already used by Google, Microsoft etc.
- These require **large amounts of data and computation** to train, although some techniques can reduce need for data.



(Honglak Lee, 2009)

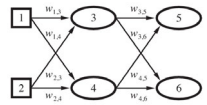
Some Guidence for Training Neural Networks

- One hidden layer is enough for most purposes
- The initial weights should be chosen randomly close to zero
- Typically a rather large number of neurons is used (hidden layer) so that the model is probably over-parameterized. But beware, too many and it will become really slow to train!
- A mechanism known as **early stopping** is often used during training to stop adapting the weights if it starts overfitting, which a neural network is very prone to.
- One thus alternates feeding it batches of training data and evaluating it's training and generalization error
- Stop when the validation error significantly increases
- Do multiple random re-starts to avoid local minima!
- Or preferrably, use a software that has all this built-in (regularization/weight decay etc)

Artificial Neural Networks – Training

- How do we train an ANN to find the best parameters $w_{i,j}$ for each layer?
- Like before, by **optimization**, minimizing a **loss function**
- What is the **computational complexity** of ANN gradients?
- Just evaluting network prediction for ANN with p parameters is O(p)

Predict output on training set



- Naive symbolic/numerical differentiation needs O(p) evaluations
  - This means computational complexity of $O(p^2)$!
- Deep learning networks often have >1M parameters. Can we do better?

Artificial Neural Networks – Backpropagation

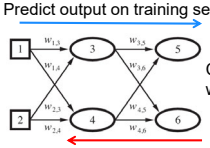
Some intuitions:

- Consider the chain rule of differentiation
  - E.g. assume f(x) = g(h(i(x))), then f(x)' = g'(h(i(x)))h'(i(x))i'(x)
- ANN layers are just compositions of sums and non-linear functions g()

$$a_5 = g(w_{0,5} + w_{3,5}a_3 + w_{4,5}a_4)$$
$$= g(w_{0,5} + w_{3,5}g(w_{0,3} + w_{1,3}x_1 + w_{2,3}x_2) + w_{4,5}g(w_{0,4} + w_{1,4}x_1 + w_{2,4}x_2)))$$

- ANN derivatives can be computed layerwise backwards, and terms are shared across parameter derivatives!
- Caching these terms gives rise to **backpropagation,** a famous **O(p)** algorithm for computing gradients

Predict output on training set



Compute errors w.r.t. a loss function

Propagate backwards and compute derivatives of weights in all layers

Artificial Neural Networks - Demo

- See interactive examples of ANN training
  http://playground.tensorflow.org/
  - 2D input x -> 1D y (binary classification or regression)
- You can try playing with
  - Different data sets vs. network size
  - Deeper neurons can capture more complex patterns
  - Classification vs. Regression
  - Learning rate (Scaling of gradient descent step)

Artificial Neural Networks – Summary

Advantages

- Under some conditions it is a **universal approximator** to any function **f(x)**
  - E.g. It is very flexible, a large "hypothesis space" in book terminology
- Some biological justification (real NNs more complex)
- Can be layered to capture abstraction (**deep learning**)
  - Used for speech, object and text recognition at Google, Microsoft etc.
  - For best results use architectures tailored to input type (see DL lecture)
  - Often using millions of neurons/parameters and GPU acceleration.
- Modern **GPU-accelerated tools** for large models and Big Data
  - Tensorflow (Google), PyTorch (Facebook), Theano etc.

Disadvantages

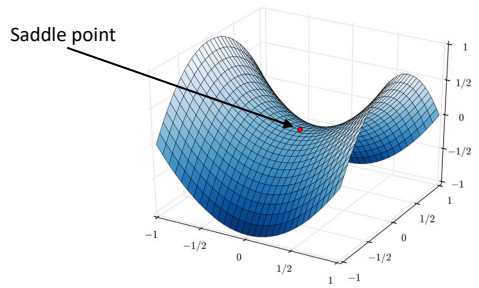
- **Many tuning parameters** (number of neurons, layers, starting weights, gradient scaling...)
- **Difficult to interpret** or debug weights in the network
- Training is a **non-convex** problem with **saddle points** and **local minima**

2021-09-23 49

49

What Was a Saddle Point Again?

- Gradient is zero, but not a minima
  - Loss could be decreased, but gradient descent is stuck
- Believed to be a more common problem than local minima for ANN



2021-09-23 50

50

Cited figures from…

C M Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

Hastie, T., Tibshirani, R. and Friedman, J.H. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Second Edition, Springer, 2009.

Which are two good (but fairly advanced) books on the topic.

2021-09-23 51

51

2021-09-23 52

52