I need notes for myself to study for ML, so let me go through the script in chronological order and write a short synopsis of each lecture, catching the essential ideas, together with important formulas, definitions and results. The overall leisurely style of the notes is a reflection of the lectures style. I have tried before to put more structure into notes for this lecture, but the content withstood my attempts unbothered, so I will go with the flow of the lecture. It also makes quite some sense, since it is a collection of tools (algorithms), arising from the concept of dynamically adapting the algorithm, to solve particular problems.

1 Probabilistic inference

Maximum likelihood: maximize the likelihood of our observations. given that we observed some event E (or rather some events E_i) our best guess at a probability distribution for the whole space is one that maximizes the probability of the observed events happening. This is the same as finding

$$f(\theta) \coloneqq p(\mathcal{D} \mid \theta)$$
$$\theta_{MLE} = \arg\max_{\theta \in [0,1]} f(\theta)$$

each evaluation at a θ is a probability distribution, obtained by imposing certain conditions. Our beliefs about theta, i.e. model choices reflecting assumptions about the experiment, are represented by $p(\theta)$. We call $p(\theta \mid \mathcal{D})$ the posterior distribution

$$p(\theta \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \theta) \cdot p(\theta)}{p(\mathcal{D})}$$

it can be interpreted as updating beliefs about θ after observing $\mathcal{D}$. Using Baye's theorem the posterior can be decomposed into $p(\mathcal{D} \mid \theta)$ called the likelihood , $p(\theta)$ called the prior encoding beliefs before we observe any data, $p(\mathcal{D})$ called the evidence, it acts as a normalizing constant.

$$p(\mathcal{D}) = \int p(\mathcal{D}, \theta) d\theta = p(\mathcal{D} \mid \theta) p(\theta) d\theta$$

From above we deduce that maximizing the posterior probability is a good guess at finding the θ governing the unknown distribution, thus we define

$$\begin{aligned} \theta_{MAP} &= \arg\max_{\theta} p(\theta \mid \mathcal{D}) \\ &= \arg\max_{\theta} \frac{p(\mathcal{D} \mid \theta) \cdot p(\theta)}{p(\mathcal{D})} \\ &= \arg\max_{\theta} p(\mathcal{D} \mid \theta) \cdot p(\theta) \end{aligned}$$

which is called maximum a posterior (MAP) estimation. To calculate it we observe that the likelihood $p(\mathcal{D} \mid \theta)$ is known and then the prior $p(\theta)$ is chosen to make calculations easier, for example the Beta distribution for some reason.

If a prior is conjugate for a given likelikhood, then the posterior will be of the same family as the prior.

The last estimation introduced is estimating the posterior distribution $p(\theta \mid \mathcal{D})$, that is not just the theta that maximizises it. This can be done by finding the normalizing constant $p(\mathcal{D})$ and using Bayes theorem again.

Example 1.1. Assume we have a coin that shows heads with probability θ and tails with probability $(1-\theta)$, then we obtain a function $f(\theta) \coloneqq p(\mathcal{D} \mid \theta)$, where $\mathcal{D}$ are some observed coin tosses. We obtain

$$\theta_{MLE} = \frac{|T|}{|T| + |H|},$$

where $|T|$ is the number of tails and $|H|$ the number of heads. Now we have

$$p(\mathcal{D} \mid \theta) = \theta^{|T|}(1-\theta)^{|H|}$$
$$p(\theta) = p(\theta \mid a, b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\theta^{a-1}(1-\theta)^{b-1}$$

put together we obtain

$$p(\theta \mid \mathcal{D}) \propto \theta^{|T|+a-1}(1-\theta)^{|H|+b-1},$$

and the maximum a posteriori estimate turns out to be

$$\theta_{MAP} = \frac{|T| + a - 1}{|H| + |T| + a + b - 2}.$$

To obtain the exact result we need to calculate the normalizing constant, which we can do by noticing the similarity between the unnormalized posterior and a Beta distribution and thus obtain

$$p(\theta \mid \mathcal{D}) = (\theta \mid a + |T|, b + |H|).$$

References for this section are

[3][ch. 3.1 - 3.3]
[4][ch. 4.2, 4.6]
https://seeing-theory.brown.edu/#secondPage

2 Evaluation

1. Train on training set.
2. Evaluate on validation set.

So a problem is apparently leakage of data, we train a model on data, but some test data leaks into the training data, thus our model is overfitting to the particular dataset. This happens also as a macroscopic process, i.e. we train models on these popular benchmark sets, people publish the results and the results are used down the line on models evaluated on the benchmark set again, overfitting to the benchmark set.

We can only ever compare the losses of models with respect to data, never the loss of the model with respect to the true distribution, thus we an only obtain empirical confidence intervals and not true confidence intervals.

Definition 2.1. A $100(1-\alpha)\%$ confidence interval for a parameter θ is any interval $I(\mathcal{D}) = (l(\mathcal{D}), u(\mathcal{D}))$ derived from the dataset $\mathcal{D}$ such that

$$p(\theta \in I(\mathcal{D}) \mid \mathcal{D} \sim \theta) = 1 - \alpha.$$

A confidence interval can be interpreted as repeatedly sampling datasets $\mathcal{D}$ and computing $I(\mathcal{D})$ then $100(1-\alpha)\%$ of the intervals will contain the true parameter θ.

3 linear regression

Let

$$\begin{aligned} f_w(x) &= w_0 + w_1 x_1 + \dots w_d x_d \\ &= w_0 + w^T x \\ &= \tilde{w}^T \tilde{x} \end{aligned}$$

we will omit the Tilde in the future in the last expression with the absorbed bias term. The squared loss is given as

$$\begin{aligned} \mathcal{L}(w) &= \frac{1}{2} \sum_{i=1}^{N} (x_i^T w - y_i)^2 \\ &= \frac{1}{2} (Xw - y)^T (Xw - y) \end{aligned}$$

and the least square loss is the corresponding minimum of the above formulas. The gradient is given by

$$\nabla_w \mathcal{L}(w) = X^T X w - X^T y$$

we obtain

$$\begin{aligned} w^* &= \arg\min_w \frac{1}{2}(Xw - y)^T(Xw - y) \\ &= (X^T X)^{-1} X^T y \end{aligned}$$

Issues with the implementation can be that $X^T X$ is ill-conditioned or singular, thus one can compute the pseudo inverse using SVD (signular value decomposition, default in sklearn), QR (if N >> D), iterative solvers such as (GMRES). More generally instead of the linear basisfunctions (x_i) chosen above, we can choose basis functions ϕ_j and obtain

$$\begin{aligned} f_w(x) &= w_0 + \sum_{j=1}^{M} w_j \phi_j(x) \\ &= w^T \phi(x) \end{aligned}$$

this gives the square loss as

$$\mathcal{L}(w) = \frac{1}{2}(\phi w - y)^T(\phi w - y)$$

with

$$\phi = \begin{pmatrix} \phi_0(x_1) & \phi_1(x_1) & \dots & \phi_M(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_N) & \phi_1(x_N) & \dots & \phi_M(x_N) \end{pmatrix}$$

which is called the design matrix of ϕ. Analogously to above we obtain

$$w^* = (\phi^T \phi)^{-1} \phi^T y = \phi^\dagger y.$$

We can also add a regularization term to the squared loss and obtain ridge regression

$$\mathcal{L}_{\text{ridge}}(w) = \frac{1}{2} \sum_{i=1}^{N} \left[w^T \phi(x_i) - y_i \right]^2 + \frac{\lambda}{2} \|w\|_2^2$$

where $\|w\|_2^2 = w^T w$. Maximizing the likelihood is equivalent to minimizing the least square loss. References for this section are.

[1][ch., 1.1, 3.1 - 3.6]
[4][ch. 7.2, 7.3, 7.5.1, 7.6.1, 7.6.2]

4 ML-libraries

Tools for loading and handling data:

- For DataFrames Panda, filtering grouping.
- numpy, for numerical operations, e.g. matrix mult., eigenvalue decomp., etc.
- matplotlib for visualisation, making subplots, saving plots, functions to draw basic plots
- seaborn for statistical data visualization. Pair plots, heatmaps, and so on.

Are your dataframes way too large? How to reduce dimensionality you ask? Use correlation to determine relation between feature and the target and delete the features that have the lowest correlation with your target. This can be done via Pandas, DataFrame.corr() in Python.

A machine learning python library is given by scikit-learn. Each algorithm is a class inheriting from **BaseEstimator**. **BaseEstimator** need to be defined:

1. a **fit** method that computes parameters of the model given the training data.
2. a predict method which makes predictions on the test data.

The **fit** function computes the minimum and the maximum of the training data. The **transform** function scales the given data to range $[0, 1]$

An important step is preprocessing the data, i.e cleaning, normalizing (account for varying scales of different categories), encoding categorical data (Label encoding, one-hot encoding: that is a binary number with only 1 nonzero value, each category is assigned one of those).

Pipelines are useful.

We need/want to find the hyperparameters of our model or best hyperparameters for our model, this is called hyperparameter search. **Estimator** has a **getparams** method that returns hyperparameters of the model, can be accesed via name, e.g. **scalerfeaturerange**. (Demo in script) What else

- Pytorch: Python library for numerical computing and bulding neural network models
- JAX: NumPy+ autodiff + GPU/TPU speed-up
- Optuna: automatic hyper-param search.
- Weights & biases: logs models, evaluations for comparison

5 Linear Classification

Given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ the total number of misclassified examples is

$$\mathcal{L}(f) = \sum_{i=1}^{N} l_{01}(y, \hat{y}) = \sum_{i=1}^{n} \mathbb{I}(\hat{y}_i \neq y_i)$$

where $\hat{y} = f(x)$ is our prediction. Now how can we choose a good $f(\cdot)$?

Assume we have two classes, then we can try to use a hyperplane $(w^T x + x_0)$ to seperate the datapoints corresponding to the 2 classes, we call a data set linearly seperable if there exists a hyperplane that seperates the data.

Lemma 5.1. *The perceptron algorithm is one of the roots of this field, the prediction is given by*

$$\hat{y} = f(x) = \mathbb{I}[w^T x + x_0 > 0]$$

where

$$H(a) = \mathbb{I}[a > 0] = \begin{cases} 1 & \text{if } a > 0 \\ 0 & \text{otherwise} \end{cases}$$

Now the algorithm is given as

$w, w_0 \leftarrow 0, 0$
while $\mathcal{L}(f) > \epsilon$ ***do***
  for $i \in \{1, \dots, n\}$ ***do***
    if $f(x_i) \neq w^T x + w_0$ ***then***

$$w \leftarrow \begin{cases} w + x_i & \text{if } y_i = 1, \\ w - x_i & \text{if } y_i = 0. \end{cases}$$

$$w_0 \leftarrow \begin{cases} w_0 + 1 & \text{if } y_i = 1, \\ w_0 - 1 & \text{if } y_i = 0. \end{cases}$$

    end if
    $i \leftarrow i + 1$
  end for
end while

If the data is linearly seperable, we can set $\epsilon = 0$ and it converges to an optimum after a finite number of steps.

This idea can be extended to multiple classes by associating to each hyperplane a class and obtain that one side of the hyperplane corresponds to the class and the other side corresponds to not the class.

Sometimes it is also very helpful to apply a transformation $\phi\colon \mathbb{R}^D \to \mathbb{R}^M$ that maps the samples to a space where they are linearly seperable, a standard example for that is changing the coordinates from or to spherical ones.

Another approach is trying to model the distribution of the class labels, i.e. a stochastic approach.

Definition 5.2. Generative model
The model consists of

- class prior - a priori probability of a point belonging to class c,
- class conditional - probability of observing x, given that it belongs to class c.

$$\text{class conditional } p(x \mid y = c, \psi)$$
$$\text{class prior } p(y = c \mid \theta)$$

then use maximum likelihood to obtain ψ_{MLE}, θ_{MLE}. We then obtain as a prediction

$$p(y_{new} = c \mid x_{new}, \psi_{MLE}, \theta_{MLE}) \propto p(x_{new} \mid y_{new} = c, \psi_{MLE}) \cdot p(y_{new} = c \mid \theta_{MLE})$$

where the proportionality comes from the fact that we are omitting a factor $p(x_{new})^{-1}$ after applying Bayes rule to the left side.

So how do we choose class prior and class conditional ? Take first the class prior $p(y = c)$. If $p(y = c) = \theta_c$, then we obtain the MLE as

$$\hat{\theta}_c = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(y_i = c).$$

For the class conditionals, given the features $x \in \mathbb{R}^D$ are continuous, we can use a multivariate normal for each class, i.e.

$$p(x \mid y = c) = \mathcal{N}(x \mid \mu_c, \Sigma) =$$
$$\frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} \exp(-1/2(x - \mu_c)^T \Sigma^{-1}(x - \mu_c))$$

How do we do predictions now ? If we have two classes $y \in \{0, 1\}$ and Gaussian class conditionals, we obtain

$$p(y = 1 \mid x) = \sigma(w^T x + w_0) = \frac{1}{1 + \exp(-(w^T x + w_0))}$$

where the parameters w and w_0 can be calculated from the Gaussian distribution. In general we obtain

$$p(y = c \mid x) = \frac{\exp(w_c^T x + w_{c0})}{\sum_{c'=1}^{C} \exp(w_{c'}^T x + w_{c'0})}$$

with

$$w_c = \Sigma^{-1}$$
$$w_{c0} = -\frac{1}{2}\mu_c^T \Sigma^{-1} \mu_c + \log p(y = c)$$

We are going to turn the general idea of the function above into a definition, due to the special interest we have in simplices.

Definition 5.3. Softmax σ is a generalization of sigmoid to multiple dimensions

$$\sigma \colon \mathbb{R}^K \to \Delta^{K-1}$$
$$x \mapsto (\sigma(x)_i)_{1 \leqslant i \leqslant K}$$

with $\sigma(x)_i = \frac{\exp(x_i)}{\sum_{k=1}^{K} \exp(x_k)}$, where $\Delta^{K-1} \subset \mathbb{R}^K$ is the standard K simplex.

We can of course also do quadratic instead of linear decision boundaries, which gets a whole lot more complicated in the calculations and frankly currently I do not understand them, this happens when we do not assume a shared covariance Σ anymore, i.e. the different classes have different covariance matrices Σ_0, Σ_1.

A discriminative model is one, that models $p(y \mid x)$ directly. That is w, w_0 are free parameters.

Definition 5.4. Let $w, w_0 \in \mathbb{R}^d$ be free parameters and $x \sim \mathit{Bernoulli}(\sigma(w^T x + w_0))$ be the posterior distribution where $\sigma(a)$ is the sigmoid function. This model is called logistic regression. The likelihood is given as

$$p(y \mid X, w) = \prod_{i=1}^{N} \sigma(w^T x_i)^{y_i} (1 - \sigma(w^T x_i))^{1-y_i}$$

and resultingly the negative log-likelihood (loss) as

$$\mathcal{L}(w) = -\log p(y \mid w, X) = -\sum_{i=1}^{N} (y_i \log \sigma(w^T x_i) + (1 - y_i) \log(1 - \sigma(w^t x_i)))$$

which is called binary cross entropy. Finding the maximum likelihood for w (the parameter w^* that maximizes the probability of our observed data appearing) is given as

$$w^* = \arg\min_w \mathcal{L}(w)$$

Again maximum likelihood estimation can lead to overfitting, we penalize large weights (since large weights make it possible for a single summand (input) to determine the whole model, i.e. overfit w.r.t. that particular input)

$$\mathcal{L}_{reg}(w) = -\log p(y \mid w, X) + \lambda \|w\|_q^q$$

Furthermore the negative log likelihood for multiclass logistic regression can be written as

$$\begin{aligned}\mathcal{L}(w) &= -\log p(Y \mid X, w) \\ &= -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic} \log \frac{\exp(w_c^T x)}{\sum_{c'} \exp(w_{c'}^T x)}\end{aligned}$$

We us a one-hot encoding for the categorical variables $y \in \mathcal{C}^N$ and obtain a binary matrix $Y = (y_{ic})_{1 \leqslant i \leqslant N, 1 \leqslant c \leqslant C}$. This is called cross entropy.

Discriminative models usually tend to achieve better performance when it comes to pure classification tasks. While generative models work reasonably well when their

References for this section are

[1, ch. 4.1.1, 4.1.2, 4.1.7, 4.2, 4.3.0 – 4.3.4] [3, ch.9, 10.1–10.3].

6 ensemble bias variance

$$\text{expected test error} = \text{variance} + \text{bias}^2 + \text{noise}$$

This can be made precise.

$$\begin{aligned}\mathbb{E}_{\mathcal{D}}[(\hat{\theta}_{\mathcal{D}} - \theta^*)^2] &= \mathbb{E}_{\mathcal{D}}[(\hat{\theta}_{\mathcal{D}} + \overline{\theta} - \overline{\theta} - \theta^*)^2] \\ &= \mathbb{E}_{\mathcal{D}}[(\hat{\theta}_{\mathcal{D}} - \overline{\theta})^2] + (\overline{\theta} - \theta^*)^2\end{aligned}$$

where θ^* is the value to be estimated, $\hat{\theta}_{\mathcal{D}}$ is the estimate we obtain from the data set $\mathcal{D}$ and $\overline{\theta}$ the estimate obtained from different datasets $\mathcal{D}$.

Definition 6.1. Let $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$ with $(x_i, y_i) \sim p(x, y)$ and $y_i \in \mathbb{R}$ be a dataset. The expected test error given a model $h_{\mathcal{D}}$ is given by

$$\mathbb{E}_{(x,y)\sim p}[(h_{\mathcal{D}}(x) - y)^2]$$

If we denote $\mathcal{D} \sim p^N$ for N samples the expected classifier $\overline{h}(x)$ is given as

$$\overline{h}(x) = \mathbb{E}_{\mathcal{D}\sim p^N}[h_{\mathcal{D}}(x)].$$

Let now $\overline{y}(x) = \mathbb{E}_{y|x}[y]$ be the expected label, we obtain putting everything together

$$\underbrace{\mathbb{E}_{x,y,\mathcal{D}}[(h_{\mathcal{D}}(x) - y)^2]}_{\text{expected test error}} = \underbrace{\mathbb{E}_{x,\mathcal{D}}[(h_{\mathcal{D}}(x) - \overline{h}(x))^2]}_{\text{variance}} + \underbrace{\mathbb{E}_x[(\overline{h}(x) - \overline{y}(x))^2]}_{\text{bias}^2} + \underbrace{\mathbb{E}_{x,y}[(\overline{y}(x) - y)^2]}_{\text{noise}}$$

where the variance tells us how much the classifier changes if we train on different data. The bias is the inherent error of the classifier even with infinite training data. The Noise is data-intrinsic error.

Remark 6.2. We generally have a bias, variance tradeoff. That is high bias, low variance leads to underfitting and low bias, high variance to overfitting.

We can lower the variance by aggregating several models, that is taking the average of multiple models

$$f(x) = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} f_m(x)$$

There are three major examples of ensemble learning:

1. Stacking: Train a meta-classifier with the base classifiers predictions as features.
2. Bagging: Create new datasets by sampling training set, train separate classifiers on each dataset, combine the predictions, e.g. average or majority vote.
3. Boosting: Incrementally train weak classifiers that correct previous mistakes, focus (give higher weight) on hard (misclassified) examples.

Algorithm 1.

$f_0(x) \leftarrow \arg\min_h \sum_{i=1}^N l(h(x_i), y_i)$

for $m \in \{1, \dots, M\}$ **do**

$r_{im} \leftarrow -\left[\frac{\partial l(f(x_i), y_i)}{\partial f(x_i)}\right]$

$f_m \leftarrow \arg\min_h \sum_{i=1}^N (r_{im} - h(x))^2$

$f_m(x) = f_{m-1}(x) + \beta f_m(x)$

end for

return $f(x) = f_M(x)$

For regression we obtain

$$\begin{aligned} r_{im} &= -\left[\frac{\partial l(f(x_i), y_i)}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_1)} \\ &= y_i - f_{m-1}(x_i) \end{aligned}$$

Algorithm 2. This algorithm is called Ada boost, it is used primarily for binary classification tasks.

$\omega_i \leftarrow \frac{1}{N}, \forall i \in \{1, \dots, N\}$

for $m \in \{1, \dots, M\}$ **do**

train f_m with weights ω_i

$\epsilon_m \leftarrow \frac{\sum_{i=1}^N \omega_i \mathbb{I}(f_m(x_i) \neq y_i)}{\sum_{i=1}^N \omega_i}$

$\alpha_m \leftarrow \log\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$

$w_i \leftarrow w_i * \exp(\alpha_m \mathbb{I}(f_m(x_i) \neq y_i)), \forall i \in \{1, \dots, N\}$

end for

return $f(x) = \text{sign}\left[\sum_{m=1}^M \alpha_m f_m(x)\right]$

Remark 6.3. Boosting reduces the bias of weak learners, bagging reduces the variance.

References for this section are

[4, ch. 4.7.6, 5.2.3, 18] https://mateusmaia.shinyapps.io/adaboosting/

7 Unconstrained Optimization

The problem we are dealing with in this chapter is that we want to minimize our loss, i.e. find the minimum of a function, that has possibly local minima that are not global minima.

Definition 7.1. Let $\mathcal{X} \subset \mathbb{R}^d$ be a convex set. A function $f\colon \mathcal{X} \to \mathbb{R}$ is called convex, iff for all $x, y \in \mathcal{X}\colon f(\lambda x + (1-\lambda)y) \leqslant \lambda f(x) + (1-\lambda) f(y)$ for $\lambda \in [0, 1]$.

Theorem 7.2. *Suppose $f\colon \mathcal{X} \to \mathbb{R}$ is a differentiable function and $\mathcal{X}$ is convex. Then f is convex iff for $x, y \in \mathcal{X}$*

$$f(y) \geqslant f(x) + \nabla f(x)^T (y - x).$$

Lemma 7.3. *If f is twice differentiable, then f is convex iff its domain is convex and its Hessian is positive semidefinite, i.e. for all $x\colon (\nabla^2 f(x))_i \geqslant 0$.*

Proposition 7.4. *Let $f_1, f_2 \colon \mathbb{R}^d \to \mathbb{R}$ be convex functions, and $g \colon \mathbb{R}^d \to \mathbb{R}$ be a concave function, then the following hold:*

- $h(x) = f_1(x) + f_2(x)$ *is convex*
- $h(x) = \max\{f_1(x), f_2(x)\}$ *is convex*
- $h(x) = c \cdot f_1(x)$ *is convex if* $c \geqslant 0$
- $h(x) = cg(x)$ *if* $c \leqslant 0$
- $h(x) = f_1(Ax + b)$ *is convex*
- $h(x) = m(f_1(x))$ *is convex if* $m \colon \mathbb{R} \to \mathbb{R}$ *is convex and nondecreasing.*

7.1 Gradient descent

Observe that

$$\mathcal{L}(\theta - \eta \nabla \mathcal{L}(\theta_t)) < \mathcal{L}(\theta)$$

which means the negative gradient is a descent direction. So we can optimize stepwise by setting

$$\theta_{t+1} \leftarrow \theta_t - \eta \cdot \nabla \mathcal{L}(\theta_t)$$

Problem: a bad choice of η leads to the process not converging. More precisely

Algorithm 3. Gradient descent with line search, let ϵ be some small real number and $\mathrm{Dom}(\mathcal{L})$ the domain of $\mathcal{L}$.

$\theta \leftarrow$ pick.Random(Dom($\mathcal{L}$))

$\epsilon \leftarrow \epsilon$

while $\mathcal{L}(\theta) > \epsilon$ **do**

$\Delta\theta \leftarrow -\nabla \mathcal{L}(\theta)$

$t^* \leftarrow \arg\min_{t>0} \mathcal{L}(\theta + t\Delta\theta)$

$\theta \leftarrow \theta + t^* \Delta\theta$

end while

return θ

Solution choose η_t to be a function of the iteration, first large changes, then fine-tuning.

Definition 7.5. The adaptive moment estimation (short: Adam) update rule is given by

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

where

- $m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla\mathcal{L}(\theta_t)$
- $v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla\mathcal{L}(\theta_t))^2$
- $\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \hat{v}_t = \frac{v_t}{1-\beta_2^t}$

Remark 7.6. Nothing stops us from considering information from higher derivatives, as opposed to Gradient descent (first-order optimization). For example the Newton method, which has the update rule

$$\theta_{t+1} \leftarrow \theta_t - [\nabla^2 f(\theta_t)]^{-1}\nabla f(\theta_t)$$

which we can mostly use for low dimensional problems, since it has $O(d^3)$ runtime, but it has nice convergence properties.

Problem: For real world statistical data, even first order methods are too costly. Solution: Stochastic optimization

We can approximate the expectation by a smaller amount of samples:

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} l_i(\theta) \approx \frac{n}{|S|}\sum_{j\in S} l_j(\theta)$$

Algorithm 4. This algorithm is called stochastic gradient descent, let $\mathcal{D}$ be the dataset

$\theta_0 \leftarrow 1$
for $t \in \{1,\dots,N\}$ **do**
  while $\theta_t > \epsilon$ **do**
    $S \leftarrow pick.random(n, \mathcal{D})$
    for $j \in \{1,\dots,n\}$ **do**
      $c_j \leftarrow \nabla l_j(\theta_t)$
    end for
    $\theta_{t+1} \leftarrow \theta_t - \eta \cdot \frac{|\mathcal{D}|}{|S|}\sum_{j\in S} c_i$
  end while
end for
return θ_N =0

References for this section are given in

[4, ch.8.1-8.4], [5, ch.6]

8 Constrained Optimization

Constrained optimization deals with an optimization task, i.e. minimizing some function, subject to additional constraints.

Definition 8.1. A multidimensional constrained optimization problem is given as follows. Given $f_0 \colon \mathbb{R}^d \to \mathbb{R}$ and $f_i \colon \mathbb{R}^d \to \mathbb{R}$,

$$\begin{aligned} &\text{minimize}_\theta \quad f_0(\theta) \\ &\text{subject to} \quad f_i(\theta) \leqslant 0 \text{ for } i = 1, \dots, M \end{aligned}$$

Example 8.2. An example that often appears, is fitting a model, such that the weights are non-negative:

$$\begin{aligned} &\text{minimize}_w \quad \mathcal{L}_{LS}(w) = \frac{1}{2} \sum_{i=1}^{N} (w^T x_i - y_i)^2 \\ &\text{subject to} \quad w_i \geqslant 0, i \in \{1, \dots, d\} \end{aligned}$$

Example 8.3. Linear Programming

$$\begin{aligned} &\text{minimize } c^T \theta \\ &\text{subject to } A\theta - b \leqslant 0 \end{aligned}$$

Geometrically this corresponds to finding the minimum value along some direction inside some convex polytope. The simplex algorithm solves LPs, by moving from vertex to vertex along the edges.

Example 8.4. Quadratic Programming

- minimize $\frac{1}{2}\theta^T Q \theta + c^T \theta$
- subject to $A\theta - b \leqslant 0$

Can we adapt gradient descent to constrained Optimization? Problem: A gradient descent step, θ^{t+1} might be outside of the region $\mathcal{X}$. Solution: Project a new point back to the closest point in the convex set $\mathcal{X}$

$$\theta^{t+1} \leftarrow \pi_{\mathcal{X}}(\theta^t - \tau \nabla f(\theta^t))$$

where $\pi_{\mathcal{X}}(p) = \arg\min_{\theta \in \mathcal{X}} \|\theta - p\|_2^2$ is the projection.

New Problem: Projection itself is a convex optimization problem, costly in praxis. Solution: Standard cases can be solved quickly.

Example 8.5.

- Projection onto box $\mathcal{X} = \{\theta \in \mathbb{R}^d \colon l_i \leqslant \theta_i \leqslant u_i \text{ for all } i = 1, \dots, d\}$

$$(\pi_{\mathcal{X}}(p))_i = \min(\max(l_i, p_i), u_i)$$

- Projection onto L_2-ball $\mathcal{X} = \{\theta \in \mathbb{R}^d \colon \|\theta\|_2 \leqslant c\}$

$$\pi_{\mathcal{X}}(p) = \begin{cases} p & \text{if } \|p\|_2 \leqslant c \\ \frac{c}{\|p\|_2} p & \text{otherwise} \end{cases}$$

- Projection onto L_1-ball can be done with a linear time algorithm.

Example 8.6. Single inequality constraint

- $\text{minimize}_\theta \ -(\theta_1 + \theta_2) = f_0(\theta)$ such that $f_1(\theta) = \theta_1^2 + \theta_2^2 - 1 \leqslant 0$
- let θ^* be the minimzer then

$$-\nabla f_0(\theta^*) = \alpha \nabla f_1(\theta^*)$$

  with $\alpha \geqslant 0$.

In the multidimensional case it holds for the minimizer that

$$-\nabla f_0(\theta^*) = \sum_{i=1}^{M} \alpha_i \nabla f_i(\theta^*), \alpha_i \geqslant 0 \tag{1}$$

Definition 8.7. Given $f_0 \colon \mathbb{R}^d \to \mathbb{R}$ and $f_i \colon \mathbb{R}^d \to \mathbb{R}$,

$$\begin{aligned} &\text{minimize}_\theta && f_0(\theta) \\ &\text{subject to} && f_i(\theta) \leqslant 0 \text{ for } i = 1, \dots, M \end{aligned}$$

we define the Lagrangian $L \colon \mathbb{R}^d \times \mathbb{R}^M \to \mathbb{R}$ associated to the above minimization task

$$L(\theta, \alpha) = f_0(\theta) + \sum_{i=1}^{M} \alpha_i f_i(\theta)$$

where $\alpha_i \geqslant 0$ is the Lagrange multiplier associated with the constraint $f_i(\theta) \leqslant 0$.

Remark 8.8. Setting $\nabla_{\theta^*} L(\theta^*, \alpha) = 0$ recovers (1) for θ^*.

Definition 8.9. We define

$$g(\alpha) = \min_{\theta \in \mathbb{R}^d} L(\theta, \alpha) = \min_{\theta \in \mathbb{R}^d} \left(f_0(\theta) + \sum_{i=1}^{M} \alpha_i f_i(\theta) \right)$$

it is called the Lagrange dual, it is concave.

For each fixed choice of $\alpha \geqslant$ the Lagrange dual $g(\alpha)$ gives a a lower bound on the optimal value p^*.

Definition 8.10. The Lagrange dual problem is given as

$$\begin{aligned} &d^* = \max_{\alpha} g(\alpha) \\ &\text{subject to } \alpha_i \geqslant 0, i = 1, \dots, m \end{aligned}$$

It always holds that $g(\alpha) \leqslant p^*$ thus

$$d^* \leqslant p^*.$$

The difference $p^* - d^*$ is called duality gap, we say that we have strong duality when $p^* = d^*$.

Lemma 8.11. *Let θ^* be a minimizer of the primal problem and α^* maximizer of the dual problem. Assume strong duality holds and $L(\theta, \alpha)$ is convex in θ, then*

$$\theta^* = \arg\min_{\theta} L(\theta, \alpha^*).$$

Proposition 8.12. *Recipe for constrained optimization*

1. *Formulate the Lagrangian $L(\theta, \alpha) = f_0(\theta) + \sum_{i=1}^{M} \alpha_i f_i(\theta)$.*
2. *For each $\alpha \geqslant 0$ obtain the dual function $g(\alpha)$ by solving $g(\alpha) = \min_\theta L(\theta, \alpha)$*
   (a) *Figure out for which α the objective is unbounded*
   (b) *For other compute $g(\alpha)$, e.g. solve $\nabla_\theta L(\theta, \alpha) = 0$ to get $\theta^*(\alpha)$, then $g(\alpha) = L(\theta^*(\alpha), \alpha)$*
3. *Solve dual problem,*

$$\textit{maximize}_\alpha g(\alpha) \quad \textit{s.t.} \quad \alpha_i \geqslant 0 \textit{ for } i = 1, \dots, M$$

given Slater's condition is satisfied, that is there exists some solution x^* that satisfies strict bounds, i.e. $f_i(x^*) < 0$.

Lemma 8.13. *Assume we have constrained optimization problem where all functions are convex. Then θ^* and α^* are optimal solutions if and only if they satisfy the Karush-Kuhn-Tucker (KKT) conditions for $i = 1, \dots, M$:*

$$\begin{aligned} &f_i(\theta^*) \leqslant 0 \qquad \textit{primal feasibility .} \\ &\alpha_i^* \geqslant 0 \qquad \textit{dual feasibility ,} \\ &\alpha_i^* f_i(\theta^*) = 0 \qquad \textit{complementary slackness,} \\ &\nabla_\theta L(\theta^*, \alpha^*) = 0 \qquad \theta^* \textit{ minimizes Lagrangian.} \end{aligned}$$

References for this section are given in

[4, ch.8.5.1-8.5.5, 8.6.1]

9 Support Vector Machines and Kernels

The objective now is to find a hyperplane that separates data-points with maximum margin.

Definition 9.1. Let $w^T x + b = 0$ be a hyperplane in a dataset, and let

$$w^T x + (b - s) > 0$$

for all points x from the class above the original hyperplane, as well as

$$w^T x + (b + s) < 0$$

for all points x from the class below the original hyperplane, be 2 additional hyperplanes. We obtain a margin between the two hyperplanes

$$m = \frac{2s}{\|w\|}$$

w.l.o.g we can set $s = 1$ and obtain

$$m = \frac{2}{\|w\|}.$$

The above problem can be restated as the following constrained convex optimization problem.

Definition 9.2. To find the maximum margin separating hyperplane, find $\{w, b\}$ such that

$$\begin{aligned} &\text{minimize } f_0(w, b) = \frac{1}{2} w^T w, \\ &\text{subject to } f_i(w, b) = y_i(w^T x_i + b) - 1 \geqslant 0, \quad i \in \{1, \dots, N\}. \end{aligned}$$

Example 9.3. Let us apply Proposition 8.12 to finding the maximum margin hyperplane

1. Formulate the Lagrangian

$$L(w, b, \alpha) = \frac{1}{2} w^T w - \sum_{i=1}^{N} \alpha_i [y_i (w^T x_i + b) - 1]$$

2. Minimize $L(w, b, \alpha)$ with respect to w and b.

$$\nabla_w L(w, b, \alpha) = w - \sum_{i=1}^{N} \alpha_i y_i x_i$$

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{N} \alpha_i y_i$$

Thus the weights are a linear combination of the training samples,

$$w = \sum_{i=1}^{N} \alpha_i y_i x_i$$

Substitute both expressions into $L(w, b, \alpha)$. Obtain the dual.

$$\begin{aligned} \text{maximize} \quad & g(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j x_i^T x_j \\ \text{subject to} \quad & \sum_{i=1}^{N} \alpha_i y_i = 0 \\ & \alpha_i \geqslant 0, \qquad \text{for all } i \in \{1, \dots, N\} \end{aligned}$$

Solve this problem instead, notice that

$$g(\alpha) = \frac{1}{2} \alpha^T Q \alpha + \alpha^T 1_N$$

where Q is a symmetric negative (semi-)definite matrix, and the constraints on α are linear. Obtain the original parameters as

$$w = \sum_{i=1}^{N} \alpha_i^* y_i x_i$$

as well as

$$b = y_i - w^T x_i.$$

9.1 Soft margin SVM

Problem: As soon as one data point violates the margins, we do not get a solution. Solution: We relax the constraint, but punish the relaxation of a constraint.

The relaxed conditions are given as

$$y_i(w^T x_i + b) \geqslant 1 - \xi_i, \quad y_i \in \{-1, 1\}, \forall i$$

the new loss is

$$f_0(w, b, \xi) = \frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i.$$

Definition 9.4. To find hyperplane that separates most of the data with maximum margin we solve

$$\begin{aligned} \text{minimize} \quad & f_0(w, b, \xi) = \frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i, \\ \text{subject to} \quad & y_i(w^T x_i + b) - 1 + \xi_i \geqslant 0, \quad i \in \{1, \dots, N\} \\ & \xi_i \geqslant 0. \end{aligned}$$

Lemma 9.5. *The problem of Definition 9.4 yields the following dual problem*

$$\begin{aligned} \textit{maximize} \quad & g(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j x_i^T x_j, \\ \textit{subject to} \quad & \sum_{i=1}^{N} \alpha_i y_i = 0 \\ & 0 \leqslant \alpha_i \leqslant C \quad \{i = 1, \dots, N\}. \end{aligned}$$

The optimal solution for the slack variables is

$$\xi_i = \begin{cases} 1 - y_i(w^T x_i + b), & \text{if } y_i(w^T x_i + b) < 1 \\ 0 & \text{else} \end{cases}$$

This means we can write this as the following unconstrained optimization

$$\min_{w,b} \frac{1}{2} w^T w + C \sum_{i=1}^{N} \max\{0, 1 - y_i(w^T x_i + b)\}$$

this is called the hinge loss formulation, the corresponding hinge loss function $\mathcal{L}_{hinge}(z) = \max\{0, 1 - z\}$ *penalizes the points that lie in the margin.*

9.2 kernels

Definition 9.6. We call a function $k\colon \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$

$$k(x_i, x_j) \coloneqq \phi(x_i)^T \phi(x_j)$$

a kernel function and rewriting the dual

$$g(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j k(x_i, x_j)$$

is called the kernel trick.

Theorem 9.7. *A kernel is valid if it gives rise to a symmetric, positive semidefinite kernel matrix (Gram matrix) $K \in \mathbb{R}^{N \times N}$*

$$K = \begin{pmatrix} k(x_1, x_1) & k(x_1, x_2) & \dots & k(x_1, x_N) \\ k(x_2, x_1) & k(x_2, x_2) & \dots & k(x_2, x_N) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_N, x_1) & k(x_N, x_2) & \dots & k(x_N, x_N) \end{pmatrix}$$

Lemma 9.8. *Let $k_1\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and $k_2\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be kernels, with $\mathcal{X} \subseteq \mathbb{R}^N$. Then the following functions are kernels as well:*

- $k(x_1, x_2) = k_1(x_1, x_2) + k_2(x_1, x_2)$,
- $k(x_1, x_2) = c \cdot k_1(x_1, x_2)$ *with* $c > 0$,
- $k(x_1, x_2) = k_1(x_1, x_2) \cdot k_2(x_1, x_2)$,
- $k(x_1, x_2) = k_3(\phi(x_1), \phi(x_2))$, *with the kernel k_3 on $\mathcal{X}' \subseteq \mathbb{R}^M$ and $\phi\colon \mathcal{X} \to \mathcal{X}'$*,
- $k(x_1, x_2) = x_1^T A x_2$ *with $A \in \mathbb{R}^{N \times N}$ symmetric and positive semidefinite.*

Example 9.9.

- $k(a, b) = (a^T b)^p$ or $k(a, b) = (1 + a^T b)^p$
- $k(a, b) = \exp\left(- \frac{\|a-b\|^2}{2\sigma^2} \right)$ or $k(a, b) = \exp(-\gamma \|a - b\|^2)$
- Sigmoid: $k(a, b) = \tanh(\kappa a^T b - \delta)$ for $\kappa, \delta > 0$.

Lemma 9.10. *We can define a classifier called kernelized SVM. Let $\mathcal{S}$ be the set of support vectors: point x_i such that $0 < \alpha_i \leqslant C$. A new point x can be classified as*

$$h(x) = \text{sign}\left(\sum_{\{j|x_j \in \mathcal{S}\}} \alpha_j y_j k(x_j, x) + b \right)$$

Note that a standard SVM cannot handle multiclass data. Two approaches:

- One-vs-rest: Train C SVM models for C classes, where each SVM is being trained for classification of one class against all the remaining ones. The winner is then the class, where the distance from the hyperplane is maximal.
- One-vs-one: Train $\binom{C}{2}$ classifiers (all possible pairings) and evaluate all of them. The winner is the class with the weighted majority vote, weighted according to the distance from the margin.

For regression we use the ϵ-sensitive loss function

$$l_\epsilon(y, \hat{y}) = \begin{cases} 0 & \text{if } |y - \hat{y}| \leqslant \epsilon \\ |y - \hat{y}| - \epsilon & \text{otherwise} \end{cases}$$

to define the entire loss

$$\mathcal{L} = \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{N} l_\epsilon(\hat{y}_i, y_i)$$

we can solve this using constrained optimization and slack variables.

The references for this section are given in

[4, ch. 17.1, 17.3] [1, ch. 7.1.0 - 7.1.2]

10 Neural Networks - Multi-Layer Perceptron

Our classical linear regression model or logistic regression model are build from the composition of some linear function and some activation function, which can be visualized via a neural network.

Popular activation functions are given by

- $\text{ReLU}(z) = \max(0, z)$
- $\text{sigmoid}(z) = \frac{1}{1+e^{-z}}$

- $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

A deep neural network is one with multiple hidden layers, examples can be obtained by stacking shallow neural networks, resulting in one refining the other.

Lemma 10.1. *We obtain all piecewise linear functions as a neural network with one hidden layer and* ReLU *activation function.*

Definition 10.2. A multi-layer preceptron (MLP) is a function $f\colon \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$ defined as:

$$\begin{aligned} h^{(1)} &= \text{act}(W^{(1)}x + b^{(1)}) \\ h^{(2)} &= \text{act}(W^{(2)}h^{(1)} + b^{(2)}) \\ &\vdots \\ y &= W^{(L)}h^{(L-1)} + b^{(L)} \end{aligned}$$

hyperparameters are depth: Number of Layers L and width and Number of hidden units per layer.

One can apply the universal approximation theorem to deep NNs.

Example 10.3. With a deep NN we can solve the classification task for some non-linearly seperable datasets (XOR dataset). Take the model

$$f(x, w) = \sigma(w^T\phi(x) + b) = \sigma\left(b + \sum_{j=1}^{M} w_j\phi_j(x)\right)$$

where ϕ_j are nonlinear basis functions.

Remark 10.4. How do we determine the loss function for a deep neural network, common choices are:

target	$p(y \mid x)$	Final layer	Loss function
Binary	Bernoulli	Sigmoid	Binary Cross entropy
Discrete	Categorical	Softmax	Cross-entropy
Continuous	Gaussian	Identity	Mean squared error

Since we are free to choose our loss function, many model types differ only in their choice of loss.

The main references for this chapter are given by

[5, ch. 3, 4, 5]

11 Convolutional Neural Networks

Definition 11.1. A function f is **invariant** to a transformation T if for all x it holds

$$f(T(x)) = f(x)$$

and it is called **equivariant** if for all x

$$f(T(x)) = T(f(x)).$$

For example image segmentation should be invariant w.r.t. translation, rotation and scaling.

Definition 11.2. Convolution is given by taking the weighted sum of inputs x as output z, we have the following parameters

- Filter size: Number of nearby inputs. (often odd to have center)
- Stride: How many steps filter is moved as we slide.
- Dilation: How many steps we skip when applying the filter.
- Padding: How we handle the edges of the inputs.

Let act be an activation function and b be a bias. Let n be the Filter size (i.e. we have waits $\omega_i, 1 \leqslant i \leqslant n$, s the stride and N the number of inputs. We obtain the filter by setting

$$X = \begin{pmatrix} x_{-\lfloor n/2 \rfloor+1} & x_{-\lfloor n/2 \rfloor+1+s} & \cdots & x_{-\lfloor n/2 \rfloor+1+js} & \cdots & x_{-\lfloor n/2 \rfloor+1+\lfloor N/s \rfloor s} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{\lceil n/2 \rceil} & x_{\lceil n/2 \rceil+s} & \cdots & x_{\lceil n/2 \rceil+js} & \cdots & x_{\lceil n/2 \rceil+\lfloor N/s \rfloor s} \end{pmatrix},$$

where the values x_i for $i < 0$ and $i > N$ are chosen according to the padding, so $X_{k,l} = (x_{-\lfloor n/2 \rfloor+1+k+ls})$ for $0 \leqslant k < n$ and $0 \leqslant l \leqslant \lfloor N/s \rfloor$. The output z_i is then defined as

$$z_i = \mathrm{act}(b + ((\omega_1, \ldots, \omega_n)X)_{1,i}).$$

We can also have 2d-convolutional layers. The convolutional kernel/ filter is now a matrix, for example let $W \in \mathbb{R}^{3\times 3}$ be a weight matrix then

$$h_{ij} = \mathrm{act}(b + \sum_{m=1}^{3}\sum_{n=1}^{3} w_{mn} x_{i+m-2,j+n-2})$$

is the output of the neural network. The terminology is equivalent to the one-dimensional case.

11.1 downsampling and upsampling

Reducing for example the dimensions of a dataset (f.e. an image), can be done by pooling. That is dividing the dataset into patches and then taking maximum (resp. mean) for maximum pooling (resp. mean pooling).

Increasing the dimensions of the dataset can be done by repeating a datapoint, that is every data point, is turned into a subset containing only that datapoint. One can interpolate the values between the pixels.

The overall architecture is now stacking convolutional and pooling layers, and using an MLP at the end.

References for this section are

[5, ch.10]

12 training networks

The main idea of this section is backpropagation, that is

1. Write your function (model) as a composition of modules,
2. work out the local derivatives,
3. do a forward pass for some x, i.e. compute the function $f(x)$ and remember intermediate values,
4. compute the local derivatives for x,
5. obtain the global derivative by multiplying the local derivatives.

If a function has multiple inputs, we compute the derivative along each of the paths. If the computational graph contains multiple paths from some vertex x to some vertex c then we sum along each of the paths.

Definition 12.1. Let $f\colon \mathbb{R}^n \to \mathbb{R}^m$ be a function and let $a = f(x)$. The Jacobian is an $m \times n$ matrix of partial derivatives

$$\frac{\partial a}{\partial x} = \begin{pmatrix} \frac{\partial a_1}{\partial x_1} & \cdots & \frac{\partial a_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial a_m}{\partial x_1} & \cdots & \frac{\partial a_m}{\partial x_n} \end{pmatrix}$$

Let $g\colon \mathbb{R}^m \to \mathbb{R}$ and let $c = g(a)$. The gradient $\nabla_a c \in \mathbb{R}^m$ is the transpose of the Jacobian $\frac{\partial c}{\partial a} \in \mathbb{R}^{1\times m}$.

$$\nabla_a c = \left(\frac{\partial c}{\partial a}\right)^T$$

We can write the chain rule thus as

$$\frac{\partial c}{\partial x_j} = \sum_{i=1}^{m} \frac{\partial c}{\partial a_i}\frac{\partial a_i}{\partial x_j}$$

in matrix form this becomes

$$\frac{\partial c}{\partial x} = \frac{\partial c}{\partial a}\frac{\partial a}{\partial x}$$

or equivalently

$$\partial_x c = \left(\frac{\partial a}{\partial x}\right)^T \nabla_a c.$$

Problem for linear regression, we have $\widehat{y} = V\sigma(Wx+b)+c$ and $\mathcal{L} = (\widehat{y}-y)^2$, i.e. $a = Wx+b, h = \sigma(a), \widehat{y} = Vh+c$ and $\mathcal{L} = (\widehat{y}-y)^2$. In order to optimize W with gradient descent, need to compute

$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial \widehat{y}}\frac{\partial \widehat{y}}{\partial h}\frac{\partial h}{\partial a}\frac{\partial a}{\partial W}.$$

So what is the derivative w.r.t. a matrix? We can calculate

$$\frac{\partial \mathcal{L}}{\partial W_{kl}} = \frac{\partial \mathcal{L}}{\partial a}\frac{\partial a}{\partial W_{kl}} = \frac{\partial \mathcal{L}}{\partial a_k}x_l.$$

We can write this as a matrix and obtain

$$\frac{\partial \mathcal{L}}{\partial W} = \left(\frac{\partial \mathcal{L}}{\partial a}\right)^T x^T.$$

Algorithm 5. An affine layer is defined as

$W \leftarrow W$
$b \leftarrow b$
$y \leftarrow Wx+b$
$\left(\frac{\partial \mathcal{L}}{\partial W}\right) \leftarrow x\frac{\partial \mathcal{L}}{\partial a}$
$\frac{\partial \mathcal{L}}{\partial x} \leftarrow \frac{\partial \mathcal{L}}{\partial a}W$
$\frac{\partial \mathcal{L}}{\partial b} \leftarrow \frac{\partial \mathcal{L}}{\partial a}$

The first operation is the **forward pass** and computes the output of the module, the other three are called **backward pass** and accumulate the total gradient.

12.1 Initialization

We want to initialize our weights in some smart manner. For example we need to initialize all weights differently, otherwise they will behave the same way through the whole learning process. Thus we we initialize with small random values for exmple $\mathcal{N}(0, \sigma^2)$ distributed. Furthermore we need to initialize W such that the variance does not explode or vanish as we pass inputs through the layers, we initialize the cariance such that

$$\sigma^2 = \frac{2}{\text{fan-in} + \text{fan-out}} = \frac{2}{D_{in} + D_{out}}.$$

Alternatively we can use

$$W \sim Uniform\left(-\sqrt{\frac{6}{\text{fan-in} + \text{fan-out}}}, \sqrt{\frac{6}{\text{fan-in} + \text{fan-out}}} \right).$$

12.2 Hyperparameter optimization

On a NN, we can tune

- number of hidden layer
- number of hidden units
- type of activation function
- optimizer (SGD, ADAM, ADADELTA, etc.)
- learning rate schedule
- data preprocessing
- type of regularization

plan is to potimize programmatically with random search, Bayesian optimization, meta learning (backpropagate through the training procedure),

12.3 shattered gradients, skip connections and batch normalization

To reduce earlier layers influence on the later ones (all derivatives in later layers are determined in terms of earlier ones) one can introduce residual blocks, where the input is added to the ouput of some layer of the neural network.

$$h^{(k+1)} = f(h^{(k)}) + h^{(k)}$$

Batch-normalization stabilizes the distribution of each layer's activations to be zero mean and unit varience using mini-batch statistics, this results in a linear increase in variance for skip-connections.

$$h \leftarrow \frac{h - \mathbb{E}_{\mathcal{B}}[h]}{\sqrt{\text{var}_{\mathcal{B}}[h] + \epsilon}}$$

One can also introduce a parameter γ and offset β and set

$$h \leftarrow \gamma h + \beta.$$

To summarize

1. Initialization is crucial for training deep neural networks.
2. Skip connections prevent shattered gradients in deep NNs.
3. We use regularization methods like dropout and batch normalization.

References for this section are

[5, ch.7, 11],[2, ch.7, 11]

13 PCA

We would like to do dimensionality reduction on our data, computationally more efficient, possibly enables visualization, denoising.

Definition 13.1. Let $X \in \mathbb{R}^{n \times m}$ be a dataset-matrix a **feature selection** is a matrix $W \in \mathbb{R}^{m \times l}$ with $l < m$ such that every row and every column of W have at most one nonzero entry, that is equal to 1. Then XW has the selection of the features given by nonzero rows of W.

This only work if the features are uncorrelated. To get around this problem we want to do **PCA** principal component analysis:

1. Change to a coordinate system such that the data is linearly uncorrelated,
2. drop low-variance dimensions.

Definition 13.2. Let $x_1, \dots, x_n \in \mathbb{R}^D$ and let $W \in \mathbb{R}^{D \times L}$ be a feature selection. Let

$$z = \text{encode}(x) = W^T x, x \in \mathbb{R}^D$$
$$\widehat{x} = \text{decode}(z) = Wz, z \in \mathbb{R}^L$$

then the **reconstruction error** is defined as

$$\mathcal{L}(W) = \frac{1}{N} \sum_{n=1}^{N} \|x_n - \text{decode}(\text{encode}(x_n, W), W)\|_2^2. = \frac{1}{N} \|X - W^T X W^T\|_2^2$$

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738. eprint: https://www.microsoft.com/en-us/research/wp-content/uploads/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[3] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN: 0262018020. URL: https://doc.lagout.org/science/Artificial%20Intelligence/Machine%20learning/Machine%20Learning_%20A%20Probabilistic%20Perspective%20%5BMurphy%202012-08-24%5D.pdf.

[4] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. URL: http://probml.github.io/book1.

[5] Simon J.D. Prince. *Understanding Deep Learning*. The MIT Press, 2023. URL: http://udlbook.com.