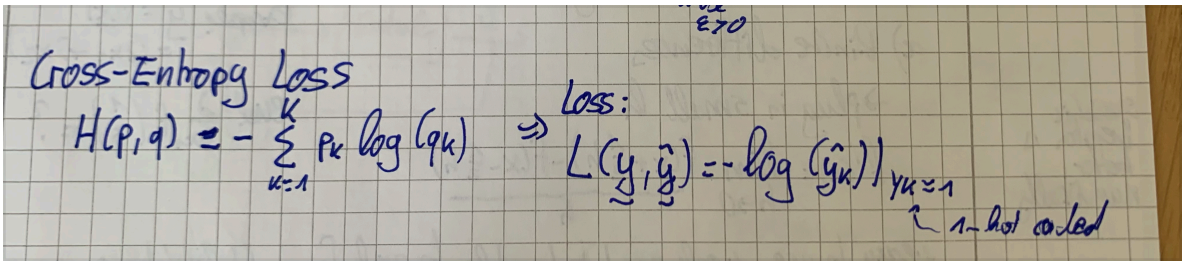
Deep Learning Questions

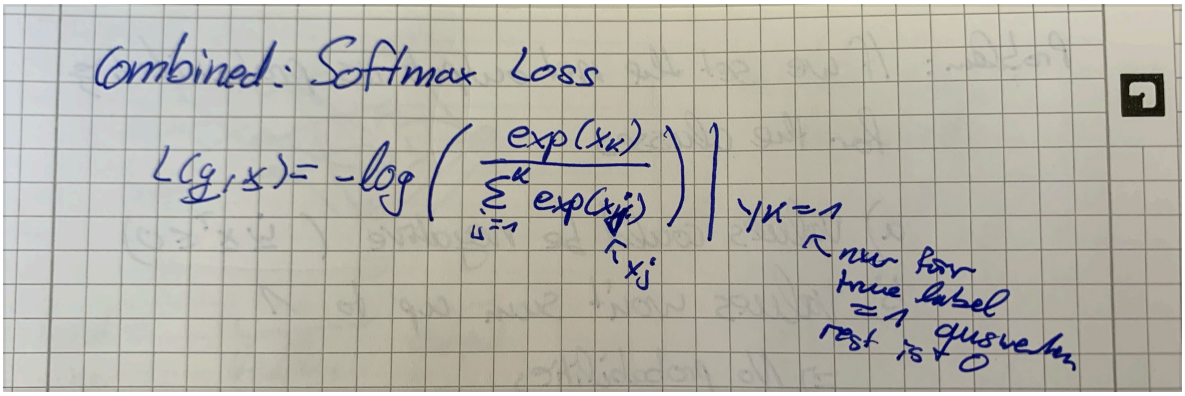
Introduction

- What are the six postulates of pattern recognition?
- What is the Perceptron objective function?
- Can you name three applications successfully tackled by deep learning?
  - Object Classification, Facial Recognition, Language Translation, Picture/Handwriting Style Transfer, Autonomous Driving (ie Lane Detection), writer recognition, MRI reconstruction, Image compression, Image generation, Image captioning, „solving“ Go ….

Feed Forward Networks

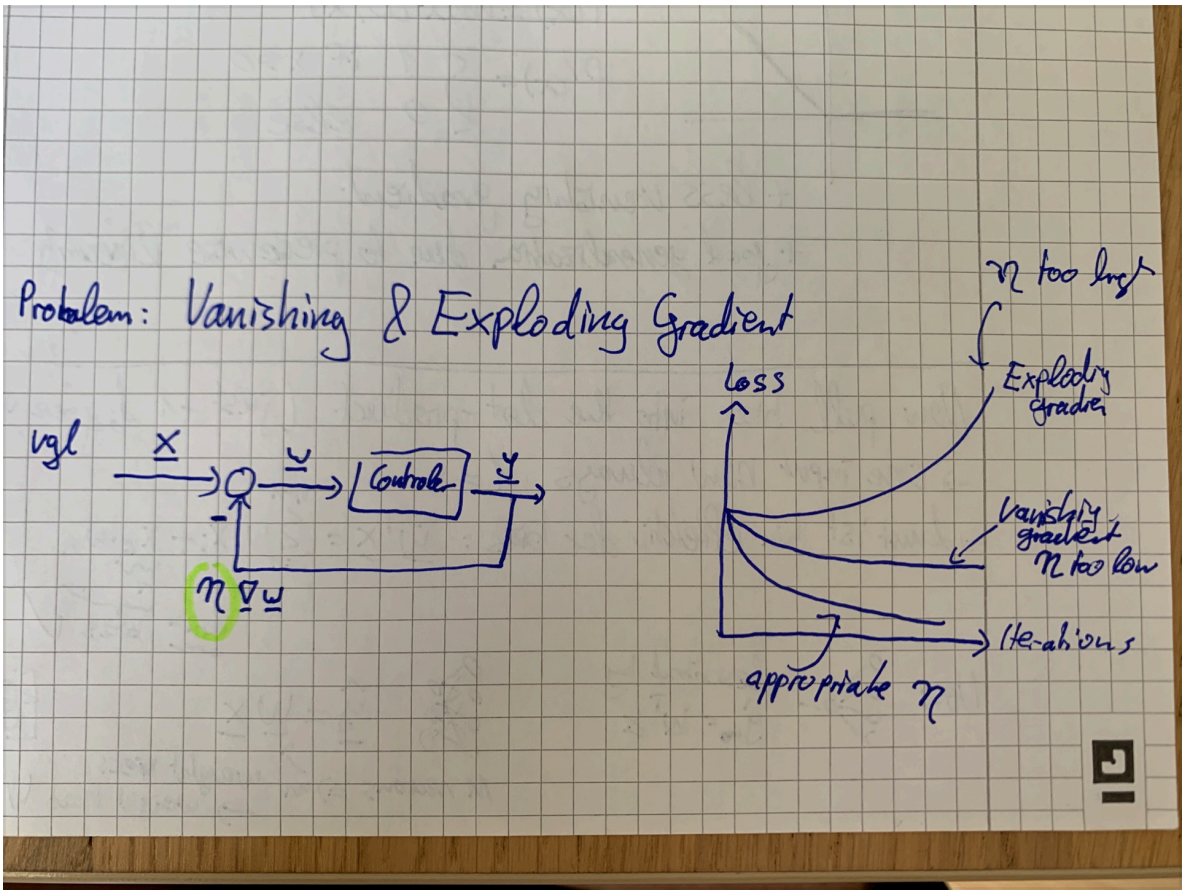
- Name a loss function for multi-class classification in deep learning
  - Ie cross entropy loss. Or often combined with soft max activation (for getting activations into pdf shape: >0 and Sum over label vector = 1) then called softplus Loss:





- Explain how this loss function works
  - Softmax makes label predictions vector into pdf. Cross Entropy gives measure how much we know from the label prediction about the true label, in other words the conveyed information by prediction (reduced uncertainty of) about true label.
- How can you check if the derivative implementation of a loss function is correct?

    - An alternative to the analytic derivative (which is fast and accurate) is the (central) finite differences: f′(x) = lim h→0 f(x+ h/2) - f(x - h/2) which is super easy to implement (no errors). We do only need a functions evaluation capacity, not need to know it's analytics structure (black box) but it is computationally inefficient
- What does backpropagation do?
    - Calculates the gradients wrt input at every neuron & layer
- How does backpropagation work?
    - It repeatedly applies the chain rule of differentiation from the output layer (& its loss) to its inputs. This gradient then gets propagated one layer back and there the same thing happens: The nodes now known output gets differentiated regarding its inputs and gets backpropagated
- Explain the exploding and vanishing gradient problems
    - New weights = old weights - learning rate * gradient wrt weights (average loss)
    - Problem is here like a feedback loop, learning rate controls how strong the feedback is incorporated
    - Learning rate too small: Too slow convergence (vanishing gradient). To high: Divergence (exploding gradient). Finding a good learning rate is critical



- Why is the signum function not used in deep learning?
    - Derivative is 2*delta(x). That means its gradient is zero almost

everywhere. Thus it does not give sensible information how to improve the weights with gradient descent

Loss Functions and Optimization

„Of high relevance: Given some probabilistic model, come up with the loss function.“

- Regression: pdf(label y given observation x) = gaussian -> MaxLikelyhood -> Log -> argmax -> derivative of weights = 0 -> L2 loss function
- Classification: Bernoulli/categorial C(observation y given model probabilities p (eg 0.7 head, 0.3 tails)) -> log likelihood -> cross-Entropy Loss (using one-hot) = Equivalent to Kulback-Leibler Divergence since we can't change the probability model implied by the data through optimizing our weights

- What are standard loss functions for classification and regression? What assumptions do our standard loss functions imply?
  - L2 (L1) Norm for regression. Implied data model: conditional probability density function of true label y given observed data x is IID gaussian (with unit variance). Then L2 is ML estimate of underlying function. Actually Gaussian was found by Gauss the other way round: Given data points that were fitted with least squares to a linear model (line) the distribution on the errors ML estimation (label - observation) were gaussian distributed. Gaussian is distribution with highest entropy given fixed mean and variance
  - Cross Entropy Loss for classification. Implied data model: Bernoulli / Categorial (Multi-dim, multi-label) distribution C(observation y given model probabilities p). Then ML estimation.

- What is a subdifferential at a point x0?
  - Convex functions are classified by the fact that every tangent is a lower bound on the function. A subdifferntial is essentially that: A line touching f(x) at the point x0 and being a lower bound for it. There may be many of such sub gradients
- How can we optimize a non-smooth convex function?
  - By looking at the (set of) sub gradients and picking one that we use for the optimization. All are equally good (only the steepness has an indirect equivalent to rescaling the learning rate)
- What if somebody tells you, to use an SVM because it's superior to Deep Learning / Neural Networks?

    - SVMs are a nice easy to use tool. Essentially there we compute a separating hyperplane with the biggest possible margin + slack variables and some regularization constraints. We can reformulate this and see that this Is equivalent to training the weights of our network if we use the Hinge Loss function which is a convex approximation of the signum function of the perceptron. If we modify it we can incorporate even more constraints into our network. So SVM is just a subset of deep learning tools
- What is Nesterov Momentum?
    - If we set the learning rate to small then we train too slowly and if we set it too high we may not reach optimum (no convergence). Now one way to deal with it is to use learning rate decay which says that we eg half the learning rate every x epochs. Another approach is to introduce momentum. We add to the current gradient mu times the old gradient so there is some memory (one delay element that is fed back) The effect is that if we repeatedly go into the same direction while updating then our steps become bigger and bigger. However if we change direction (because we may have shot over the goal) then our steps get smaller. Nesterov builds on the same idea but tries to implement momentum in a look ahead fashion in that the momentum term is added before and not after computing the next gradient so we effectively try to approximate the next steps momentum. This reduces oscillations of the gradient that can occur with standard momentum
- Describe Adam
    - Has momentum inbuilt plus it assigns individual learning rates to parameters (ie if they are only activated every now and then or if their gradients are big / small) by taking the squares of the gradient vectors elements and multiplying the learning rate with this terms square root inverse. The goal of this to correct for the variances of the parameters change in the updates. But since this might lead to too aggressive learning rate decay a dampening factor rho is introduced that tones this a bit down. Finally a term for bias correction of momentum and learning rate variances is introduced. It's quite robust but it's loss curves are harder to interpret Page 42

Activation functions and CNNs

- Name 5 activation functions & discuss those
    - Sigmoid: 1/(1+e^-x).

    - Advantages:
        - Easy, continous derivative f'(x) = f(x)(1-f(x)).
        - Follows biological intuition & is „continuous perceptron“
        - Output are [0,1] and can be interpreted as probabilities
    - Disadvantages:
        - Vanishing gradient problem (saturation in very high/low values)
- TanH (= shifted sigmoid) = 2*sigmoid(2x)
    - Advantages:
        - Zero-Centered
    - Disadvantages:
        - Vanishing Gradient
        - Output ranges -1 to +1 so no probabilities
- ReLU: max(0,x)
    - Advantages:
        - Fast, easy derivative (0 or 1)
        - Good generalization due to piece-wise linearity
        - Gets rid of the vanishing gradients (non zero / meaningful gradient in large areas)
    - Disadvantages:
        - Dying ReLU: Can get stuck in negative inputs = zero gradient. No learning anymore
        - Outputs are no probabilities
        - Not zero-centered
- Leaky Relu / Parametric ReLU: max(alpha*x, x). Alpha ie 0.01
    - Advantages:
        - ReLU + Fixes Dying Relu issue
        - Parametric ReLU: Learns how leaky ReLU should be (alpha)
        - GO TO SOLUTION
    - Disadvantages
        - Outputs are no probabilities
        - Not zero-centered
- Exponential LU / Scaled ELU
    - Advantages:
        - Output is zero-centered
        - Scaled: Special parameters of ELU that make output zero mean and unit variance (alternative to batch normalization)
    - Disadvantages
        - ??
- Radial Basis Functions
- Maxout
- Softplus (empirically worse than ReLU)

- What is the zero Centering problem (= co-variate shift of successive layers)?
  - If input is zero mean and it is passed through say a sigmoid function, then the ouput is [0,1] so it's mean must be higher. This forces successive layers to constantly adapt to this shift in means which will effectively limit the variance of the updates in batch learning.
- Why does ReLU work so much better than tanH?
  - Doesn't have the problem of vanishing gradient which allows deeper networks (since backpropagation multiplies this error with every layer)
- Why are CNNs well suited for image/audio processing?
  - CNNS allow to a) limit the number of features/weights that have to be trained by weight sharing across the image within one feature map which gives translational invariance. Since an image can easily have 512*512 pixels this would give lots of weights in FC layers which would make the network very susceptible to overfitting/high data demand. So they are in a way regularizers. This is possible because pixels are bad features since they have spacial correlations with their neighborhood, are easily changed eg by translation or rotations or rescalings of the same overall object in question on the picture. Also combined with pooling they make for good elements for decomposing difficult tasks hierarchically into more and more abstract features. Also due to Fourier transform and linearity of the convolution/ correlation operation they can be performed efficiently. Essentially they are convolutional filter banks with trainable layers and followed by some non-linearity (ie pooling & activation function). Also if padded correctly and if the classification layer FC is replaced by a 1x1 convolution with flatten and global avg pooling (essentially the same) the architecture becomes independent of input image sizes which is very nice.
- Write down a mathematical description of strides convolution:
  - Today often used to replace convolution + pooling. It's essentially a convolution followed by subsampling and can serve as a means of dimensionality reduction. Other option is dilated convolution (spacings between elements of receptive field). Stride describes how much the kernel middle point jumps when shifting the kernel through the image in the convolution process.
- What is the connection between 1x1 convolutions and fully connected layers?
  - 1x1 = FC if flatten is applied before hand and all features are put into the channel dimension + global avg pooling is applied
- How would you implement a classifier that operates on image

patches?

- What is a pooling layer? Why do we use pooling layers?
  - A Pooling layer (eg Max, Avg, Stochastic etc) serves as a means of dimensionality reduction / aggregation of features for breaking down the complexity of the problem / making the tasks individual layers learn to fulfill layer by layer more abstract. Also can serve as a non-linearity (Max). This also helps reducing the number of trainable parameters (regularization) to avoid overfitting. Max Pooling for example propagates only the max value of a respective area further and only this cell gets the backpropagated error (sub gradient approach). In AVG pooling error is shared. Idea for max: We don't need to know eg where edge is, but only if there is one
- On what data would CNNs perform badly?
  - On data where we wouldn't want translational invariance???
  - On data with no spacial correlation???

Regularization

- What is the bias-variance tradeoff?
  - Bad model performance can come from two sources: Either under fitting (= too high bias ie linear regression) or overfitting (= too high variance = susceptibility to noise, ie nearest neighbor algorithm). With bias variance decomposition we can show that we can decompose the total prediction error into bias and variance induced error terms. We need to carefully trade them off against each other.
- What is model capacity?
  - Model capacity describes the variety of functions the network can approximate ie how flexible it is. It is formally measured by VC (how many points it can separate) and loosely associated with the number of effective parameters/degrees of freedom it has available in the optimization. The capacity of ANNs are usually extremely high (exponentially growing in number of neurons and number of layers). This is why we need to watch out for overfitting = low training error and high test error (memorization which is the perfect way to get the loss to 0 but doesn't lead to generalization) a lot. Opposite: Underfitting (medium training error, medium validation error). So model capacity has to match the size of the training set (would be huge if unregularized)
- Describe three techniques to address overfitting in a neural network
  - All have in common: Enforcing prior knowledge
  - Increase training data variety eg by Augmentation of training data

(eg flipping, rotating, adding noise, zooming)
    - Adapt architecture
    - Adapt training process
    - Regularization of loss function eg by L2 norm (small weights) or L1 norm (sparsity in weights / connections)
    - Dropout (during training drop random node with probability 1-p and during test time multiply it with p). This breaks the case where two nodes have same weights and thus always receive the same gradient update and can't develop independent features. So never init weights all 1.
    - Auxiliary task / Multi task learning: training on multiple related tasks simultaneously (ie translating multiple languages)
- Why do we often need a validation set?
    - We train on the training set. The error gets progressively smaller. At some point we get into overfitting. But how do we know when? By the point the validation error is the smallest (the network only queries it eg at the end of each epoch as inference set, but doesn't see it as training set)
- Can we optimize the hyperparameter lambda by gradient descent on the training set?
    -
- What connects the covariate shift problem and the ReLU?
    - Covariate shift problem: ReLU is not zero-centered. That means if input has zero mean, since ReLU will set all negative values to 0, the mean of the output will be higher than the input's one. This forces the layers to constantly adapt to this and thus slows down learning
- How can we address the covariate shift problem?
    - One way would be to use zero-centered activation functions like SELU + specific weight initialization. Originally proposed way was by batch normalization which calculates the mean and variance of the inputs and renormalizes them (element wise, per mini batch) to zero mean and unit variance. Then a linear transform (weight & bias) gamma * x + beta is applied. These parameters are learned by backprop. In validation time / inference some fixed mean and variance for normalization is used. The way it works is probably not due to covariate-shift but due to smoothing the loss landscape
- Which problem do current initialization schemes try to address?
    - We need proper initialization since our problems are non-convex so it matters. Bias can be init as zero. With ReLU we should however go for small but positive due to dying ReLU problem. Weights: Don't init with zero -> zero gradient!, better with Gaussian distribution of small magnitude. The variance of the weights initialization values influences the stability of the training:

The output variance of the neuron grows with the number of connections of the neurons so we divide the weights variances by their mean of in + out weights (Xavier) or (He) by in-nodes.

- What is transfer learning?
  - An AI that composes violin music should have learned a lot of things that are useful (ie rhythm, melody, ...) for composing piano music. Since early layers do more general things we copy the weights from the violin network and set their learning rate to 0 in the early layers. We only train the second part of the network and maybe adapt its architecture a bit.

Common Practices

Architectures

- „Highly relevant": Why do ResNets work so well? (Page 26) / Why can we say that residual networks learn an ensemble of shallow networks
  - Normal Layers learn the input-output mapping F(x). However if we stack up these layers very deeply we get poor propagation properties, covariate-shift issues and vanishing gradient (neurons saturate and early layers don't train anymore). Covariate shift problem we can solve by ELU/SELU/Batchnorm. Vanishing gradient we can solve by ReLU. Additionally Residual elements can help. They learn the residual of a normal layer thet is ouput-input. So input x is forwarded (skip connection) and in a side branch the trainable function h(x) is computed and then added to x. We can apply 1x1 conv as bottleneck layer to reduce the number of features and thus computational burden. If we build multiple resNet block upon each other they all function like a ensemble of shallow networks. We implicitly average over exponentially many networks / there is an exponential number of features (resume). At layer n we have 2^n paths of varying length not all the same. They do not build completely new representation but gradually improve on previous representations. Like in biology we can even drop total layers without completely destroying the predictive power. ResNet can be interpreted as a form of gradient descent procedure
- What are the advantages of deeper networks?
  - New records in challenges have employed increasingly deep networks (empirical results). Generally the model capacity increases with network depth. One reason is that they can

hierarchically decompose tasks into smaller and smaller subtask that give the network in later layers more and more abstract features. Another reason is that they enable exponential feature reuse (eg 2 features put through n layers -> 2^n feature paths through the network)

- How does a bottleneck layer work?
  - A bottleneck layer is a 1x1 convolution (that can eg be used after a resNet block) that does feature compression / dimensionality reduction which works without much degradation since most features are correlated anyhow. The advantage is that we need to do fewer computations.
- What is the standard inception module and how can it be improved?
  - The standard inception module let's the net decide if it wants to do a convolution, a pooling or something else. This is done by having the available options calculate in parallel in one layer and concatenating the output (split-transform-merge strategy). It can be improved by including 1x1 convolution bottleneck layers that limit the amount of computations needed. Actually I think 1x1 is already part of it. The improvement is instead of offering 7x7, 5x5, 3x3 to the network we replace 7x7 with 3x3 followed by 3x3.

- Innovations by layers?
  - LeNet: Convolution for spatial features
  - AlexNet: GPUs, ReLU, Dropout, Dataaugmentation, mini-batch SGD momentum with L2 weight decay
  - Network in Network: 1x1 convolution + Global average pooling = FC but flexible for every input image size
  - VGG: Small kernel sizes 3x3, gradually increased channel dimension while decreasing spacial dimension
  - Google Net: Inception Modules
  - ResNet: Residual Units (skip connections that allow the network to choose the data path. Effectively builds layered shallow networks. The layers do not compute new representations but gradually improved the previous ones. Allows easier/faster training.)
  - Reinforcement learning of architectures: Very computationally complex.

Recurrent Neural Networks

Very important chapter. Eg know how to sketch a LSTM, Elman Cell, GRU
Advantages: Alleviate the vanishing gradient (thus allowing for longer

memory) by the linear transformations on the cell state. In the hidden state we have a program/finite state machine that operates on some memory and learns which information to store/delete/load. Network designs seem to gradually approach computer architecture

- What is the strength of RNNs compared to feed-forward networks
  - Output for every input sample (real time)
  - Time dependent signals / temporal correlations (eg Speech, Language, Videos, Time-Series, Sensor Data…)
- What role does the hidden state play in RNNs?
  - The hidden state relays memory from one time step to the next. This allows us to capture sample sequence correlations
- How do you train RNNs? What are the challenges?
  - We calculate the forward pass for the full sequence & unravel the network/cells for that. At the last layer we compute the loss. Then we propagate the loss back through every layer (we need to do this since the hidden state output of first step influences the last one so we need to propagate the error all the way back)
- What is the main idea behind LSTMs and what is the main difference to simple RNN units?
  - The Elman Cell completely overwrites the old hidden state with a new one. This makes it difficult to keep a memory that is relevant for later prediction steps (long range correlations, eg if every step produces a new word then the context from the previous sentence probably already vanished). Also gradient propagation through many layers leads to vanishing gradient since they are related by multiplications.
- What are the differences between LSTMs and GRUs?
  - Page 48-52
- In which scenarios would LSTMs be beneficial compared to GRUs?
  -

- Name three applications where many-to-one and one-to-many RNNs would be beneficial
  - Sequential relationship of input samples vs architecture / outputs:
  - One to one: Image classification (=classic feed forward) (sequence of unconnected pictures: One. Architecture: Feed forward: One)
  - One to many: Image captioning (sequence of unconnected, individual pictures as input: One. Architecture: Outputs multiple linked words: Many)
  - Many to one: Sentiment analysis (Multiple words as input that are sequentially linked (context matters): Many. Architecture: Outputs one word)
  - Many to many: Video classification (Input images are sequentially

linked: Many. Architecture: Multiple outputs)
    - Music compasition? Impulse response estimation? Speaker recognition, Speech synthesis? Speech direction estimation? Time series next sample prediction?

Visualization and attention mechanisms

- Why is visualization important?
    - End to end training is a major benefit of deep learning (ie no need to design features) but it makes the whole process appear as a black box. But if there are problems in training (eg dying ReLUs) then we would want to identify them. Also we would like to gain insight why / what the network learns to make the algorithm explainable both to other researchers as well as the government or a boss or customers. Also we want to be able to identify faulty training/test data (that eg lead to confounders vgl Dumbell class learns arm that holds it)
- What can visualization help with? What can't it help with?
    - There are three big areas that we want to visualize:
    - Architecture (& the priors it imposes): Node-link diagrams, Block diagrams, fancy stuff
    - Training: Loss (Training & test) over time, sample predictions.... (Ie Tensor Board)
    - Parameters/weights/Representations of data in network:
- Why are confounds a problem?
    - DL networks are really good at learning the most easy connection between the training data and their labels to minimize the loss. The training data can however include some biases that make the network learn something else than the intended relationship as a proxy. For example in speaker recognition if there are audio samples recorded in different studios/microphones/distances that are individual for certain speakers the network might have a way easier job to learn the microphone characteristics than the speakers voice patterns. Another example is in medical data: Patients with an illness in treatment go to hospital A. Patients with no illness go to hospitals B and C. Then illness detection might learn just the noise characteristics of sensor A. This Is very bad overfitting and might only occur to us in real life applications and not with the test/validation data! A solution is to look at the representations of the network and see eg if the dumbbell class learns an arm that holds it or to work with occlusion techniques.
- Why could adversarial examples pose a security problem?

-
- How does occlusion work?
- What is the difference between deconvolution, backpropagation and guided backpropagation regarding feature visualization?
- What is a saliency map?
- What is the idea behind attention?
- What is the difference between normal and self-attention?
- How does the transformer architecture avoid recurrence and convolutions?

Reinforcement Learning

- What is a policy?
  - A policy pi (a=action) is a mechanism of choosing the action to be played (eg as a pdf) from them available moves as to maximize the overall reward
- What are value functions?
  - Value functions tell us how valuable (that is how big the expected / future reward) for either taking a specific action and/or obtaining a specific game state is. Either we just accumulate and average the rewards or we actively discount them into the future. If we have the optimum value functions (that is we know which actions in which state bring the highest rewards) then we can easily find the best policy by greedy optimization (therefore if we update the maximum value function then we simultaneously implicitly update the best policy). This process is called policy iteration. If we do this until no further improvement happens we know this is guranteed to work. If we update the policy directly instead of updating the value function, this is called value iteration.
- Explain the exploration vs exploitation dilemma
  - Compare to a company: On the one hand we just want to exploit the profit making opportunities of existing products like a big company like McDonalds where we have a proven concept and a cashcow. It's just the one thing we know that gives us the biggest profits. On the other hand we need to find these cash cows in the first place or want to know if there are better ones attainable. This is why we need to do previously unknown moves and explore. This is the dilemma: We would like to to as much exploitation as we can but for that to be effective we need to have done enough exploration.

- Describe typical solutions to the dilemma
  - Epsilon greedy proposes to use the best action known with probability 1-epsilon and then unformly randomly picks on of the remaining actions if not the best is played. Eg 90% exploitation, 10% exploration
  - Softmax: Parameter tau to decrease exploration over time. Softmax is a smoother version of greedy in the sense that if there are two very good actions but one is slightly better than in greedy only the very best would be played as exploitation move and in softmax their porbability of being played would be proportional to their rewards.
- What is the difference of a multi armed bandit problem to the full reinforcement learning problem?
  - Multi-armed bandit is sequential decision making problem where we the reward only depends on the played action as pdf: p(r=reward given a=action). We want to pick the argmax a maximizing the expected value of this pdf.
  - Full reinforcement learning problem: Actions also influence the state of the world. PDF(reward given state & action), captured by state transition pdf (new state given action and old state)
- Describe a Markov decision process
  - We can choose an action, this action changes the state according to a state transition pdf (new state given action and old state). Also the action produces a reward according to the pdf (reward given action and state). The action itself is chosen by a policy pi (action given state). If all those sets are finite we call this a finite Markov decision process (eg there is a finite number of states and actions possible. No continuous choice. Squential decision making
- Is an optimal policy necessarily unique?
  - In a finite Markov decision process there is only one optimal policy
- What do the Bellman equations represent?
  - The Bellman equations are consistency conditions for our state-value / action-value function. They give us a means of iteratively (after every action taking and reward over serving cycle) updating the value functions
- Describe policy iteration
  - Take action according to greedy action selection on value-function (choose max reward). Observe the reward. Update value-function by applying the bellman update equations. Iterate until policy stops changing.
- Why does policy iteration work?
  - Since we only select greedily we only change our action if the expected reward is higher. This means that our payoffs only increase or stay the same at worst.

- How can you beat your friends at every Atari game?
  - By using deep reinforcement learning. We train a net on recognizing the game states eg understanding the displayed image frames (CNN!) and one on learning the rewards given the states and actions. The outputs are the actions. This is deep Q learning
- How can one master the game of Go?
  - For Go we have combinatorial explosion of full search and value-functions that are not constructible even with the help of human experts. So we do a Monte Carlo tree search (with some probabilities extend the search tree down in promision variants of the game). This rollout policy is controlled by a neural network. Another neural network is examining the valuation of positions (trained by self play). And the third net finally is the policy network which suggests the next moves in leaf nodes for extension.

Unsupervised Deep Learning

- What is the basic idea of contrastive divergence?
- What is the defining characteristic of an auto encoder?
- How do denoting auto encoders work?
- What does an optimal discriminator for GANs learn?
- What are the advantages of GANs in comparison to other generative models?
- What is mode collars?
- Explain feature matching / perceptual loss
- How do variational auto encoders work?
- How can we use backpropagation with random sampling layers?
- How do cycle GANs work?

Segmentation and Object Detection

Stanford Cs230exam win19 soln - cs231n exam as a reference mögliche Vorlage