UiO : Department of Informatics
University of Oslo

IN5400 Machine learning for image classification

Summary - Learning goals

02 - Linear models for regression and classification

- Understand linear regression and the loss function
- Be able to compute by hand and implement the gradient descent updates
- Understand logistic regression and the loss function
- Be able to compute by hand and implement the logistic gradient descent updates
- Understand softmax classification
- Cross-entropy loss will be derived in detail next week
- Implement softmax and gradient descents for cross-entropy loss
  - This will come in handy for Mandatory 1
- Theory exercises relevant for exam

03 - Dense Neural Network Classifiers

- Detailed knowledge about the structure of a standard fully-connected classification network
  - Activation in an arbitrary node at an arbitrary layer
  - Sigmoid and ReLU activation functions
  - Softmax function with multiple classes
  - Cross entropy loss function
  - Stochastic gradient descent optimization method
  - Backward propagation
- Justification and derivation of concepts, knowing why they are as they are
  - Cross-entropy loss function
  - Stochastic gradient descent method
  - Backward propagation equations (with cross-entropy loss and softmax)
- Knowledge about making the implementation efficient
  - Derivation of the various vectorized equations

04 - Pytorch

- Know what we mean with a deep learning framework.
- Pytorch
  - Automatic differentiation

05 - Convolutional neural networks

- Understand why we use CNN on image data
- Know what a convolutional layer is, and its hyperparameters
- Understand the difference between the theoretical and the effective receptive field
- Know how we can use dilated convolutions
- Know what a pooling layer is
- Understand why depthwise separable convolutions can be parameter efficient.
- Know the difference between using a dense layer and a convolutional layer as the last layer in the network.

07 - Training and architectures

- Pro’s and con’s for different activation functions.
- How weights should be initialized and scaled given the activation function.
- How batch norm works at training and test time.
- How momentum SGD and ADAM works.
- Know how to optimize the hyperparameters, including scale and sensitivity.
- Know the most characteristic features of central architectures.

08 - Generalization

- You should be familiar with the learning problem.
  - In-sample
  - Out-of-sample
  - Hypothesis set
  - A hypothesis
  - Final hypothesis
  - Target hypothesis
- Model capacity/complexity
  - VC dimention
- The Vapnik-Chervonenkis Inequality
- Learning from a small datasets
  - Regularization L2
  - Dropout
  - Data augmentation
  - Transfer learning
  - Multitask learning

09 - Segmentation and Object Detection

- Performance evaluation metrics
  - Sensitivity
  - Specificity
  - Precision
  - Accuracy
  - Balanced accuracy
  - Jaccard index
  - Mean average precision (familiarity)
- Object detection
  - Label vector
  - Multi-task loss function
  - Network architectures (basic concepts)
- Image segmentation
  - Spatial upsampling (unpooling) techniques
  - Network architectures (basic concepts)
  - Difference between semantic segmentation and instance segmentation

10 - Visualization and adversarial fooling

- Understand the need to be able to visualize the network
- Understand the limitations of visualizing the filters directly
- Understand heatmaps like class activation maps, saliency maps, and know about layerwise relevance propagation
- Know the principles and goals of guided backprop
- Understand how adversarial images are created, and what adversarial training is, and what models are suspect to being fooled.

11 - Recurrent neural network

- Know how to build a recurrent neural network
- Understand why recurrent neural network are susceptible to exploding and vanishing gradients.
- Know why we use backpropagation through time
- Gated Recurrent Unit (GRU)
- Multi-layer Recurrent Neural Networks
- Bidirectional recurrent neural network

12 - Unsupervised Learning

- Background (clustering, PCA) is background, and assumed known (not including derivations)
- SNE
  - Motivation and idea
  - High-dimensional neighbor probability distributions
  - Low-dimensional neighbor probability distributions
  - Probability distribution distance metric
  - Optimization of the distance metric
  - Benefits and drawbacks
- t-SNE
  - Motivation and idea, in relation to SNE
  - Changes from SNE
  - Why it fixes some problems with SNE, and works better

12 - Unsupervised Learning

- Autoencoders
  - Detailed knowledge (based on lecture and weekly exercise) about the various variants (compression-, denoising-, and sparse-autoencoder)
- Variational autoencoders
  - Purpose
  - Problems it is trying to solve, and how it solves them
  - Structure and key elements (related to the previous point)
  - How it can be used to generate new images from the training distribution
  - How it can be used to generate an interpolation between two existing images

13 - Generative Adversarial Networks

- General concept
  - Purpose and structure of the generator network
  - Purpose and structure of the discriminator network
  - Training process
  - Advantages and challenges
- Motivation and derivation of generator and discriminator cost functions
  - Minimax-GAN
  - Non-saturating GAN

14 - Reinforcement learning

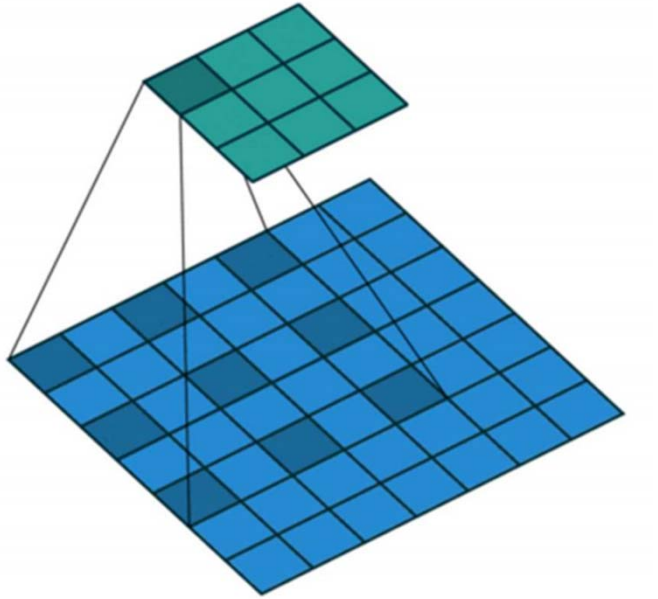
- Be familiar with the RL environment: states, actions, rewards, return
- Value based RL:
  - State-value function and action-value function
  - Understand how we can use the bellman (expectation) equation to update a value function.
  - Q-learning
  - Exploration vs Exploitation
  - Experience replay
- Policy based RL:
  - Policy Gradients
  - Why policy Gradients suffers from high variance

15 – Not just AI/Deep learning

- Be able to reflect upon the challenges of applying deep learning in an application
- Be confident about the current limitations of deep learning and seek to gain insight into the black box.
- Know the need for evaluating the statistical significant for any deep learning application.
- Know the need to get insight into when the system will fail.
- Know the pitfalls of bias in the model and the data
- Be able to reflect upon ethical challenges given an application.
- Know that you as a developer must follow GDPR and other national regulations.

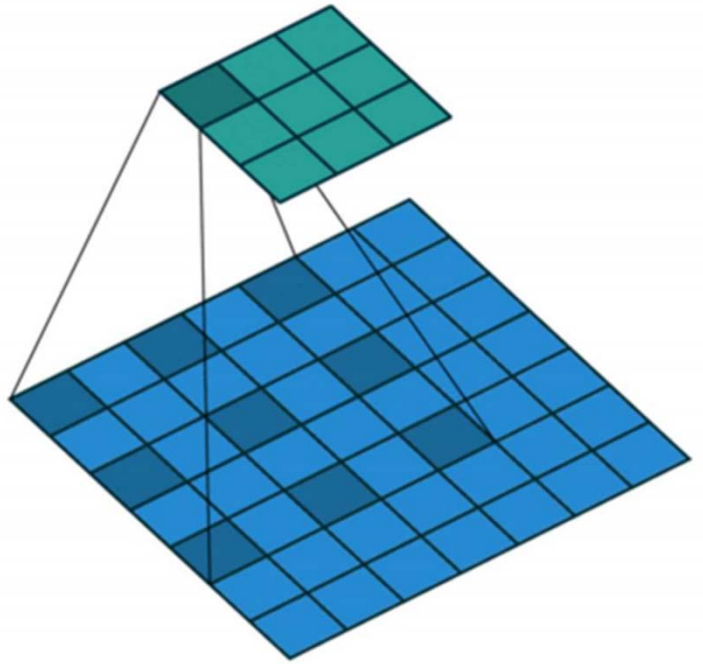
Dilated convolutions

- Larger receptive field, without reducing spatial dimension or increasing the parameters



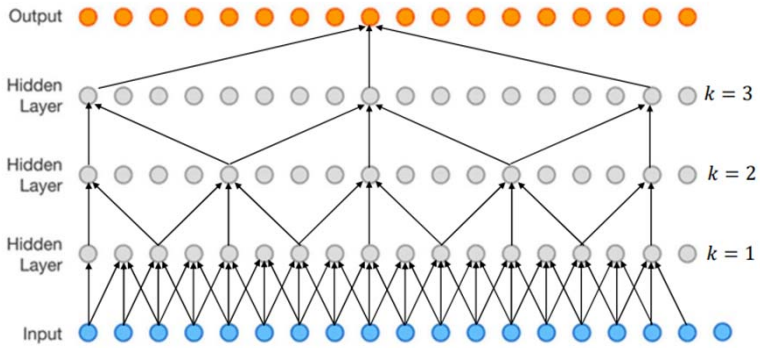
Dilated convolutions

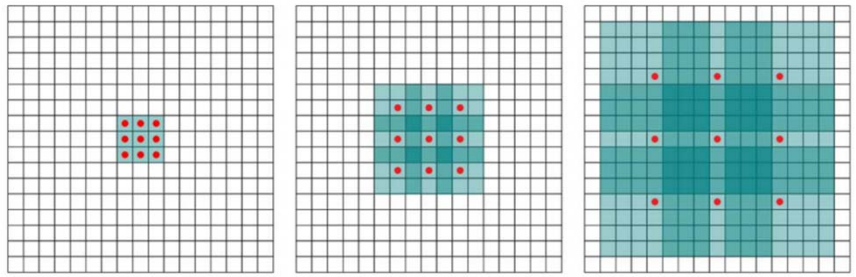
- Skipping values in the kernel
- Same as filling the kernel with every other value as zero
- Still cover all inputs
- Larger kernel with no extra parameters



A growing dilation factor can give similar effect as stride

- With a constant dilation factor you get the similar effect as using a larger kernel
- With growing dilation factor you can get an even larger receptive field, while still covering all inputs

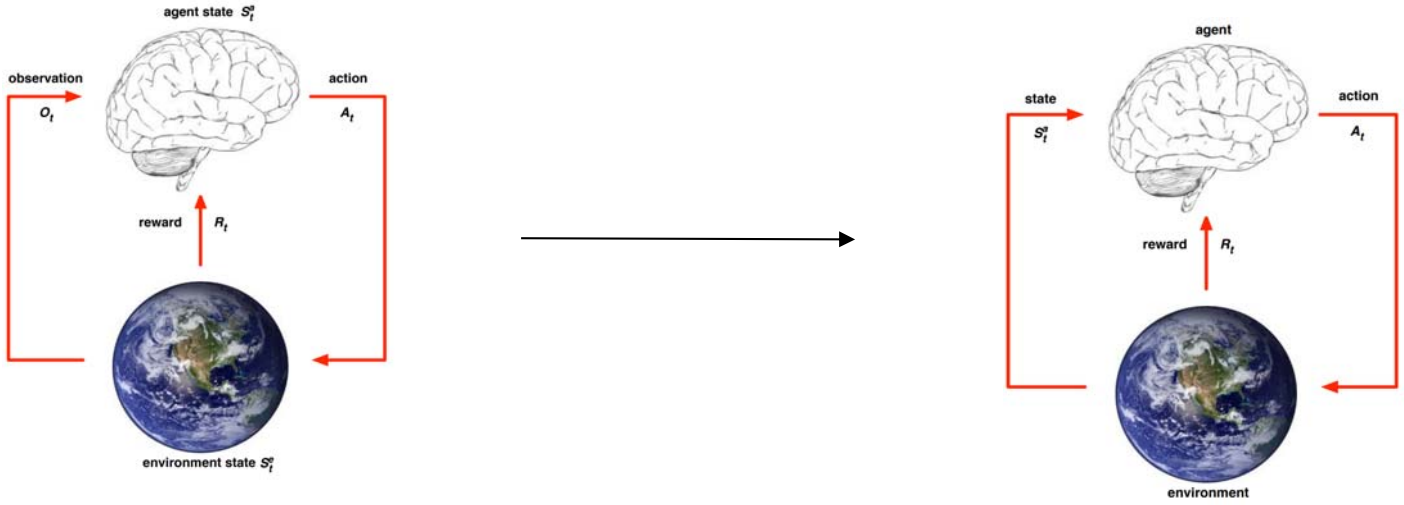




Fisher Yu, Vladlen Koltun (2016) Multi-scale Context Aggregation by Dilated Convolutions

Reinforcement learning

- **History / trajectory :**
  - $H_t = \tau_t = R_1, O_1, A_1,\ R_2, O_2, A_2, \dots,\ R_t, O_t, A_t$
- **State:**
  - $S_t = f(H_t)$
- **Full observatory:**
  - $O_t = S_t^e = S_t^a$



Value based vs policy based RL

- Value function based methods (Q-learning)
- Policy based methods (policy gradients)

- The goal in both methods is to find a policy which maximizes the accumulated reward.

$$G_t = R_t + \gamma R_{t+1} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$$

- In value based methods we try to estimate the goodness of a state (and an action). We then selects actions greedily based on the values function. Example: grid world

- Because estimating a value function can be difficult, policy gradients tries to estimate the best policy directly.

State-value function and action-value function

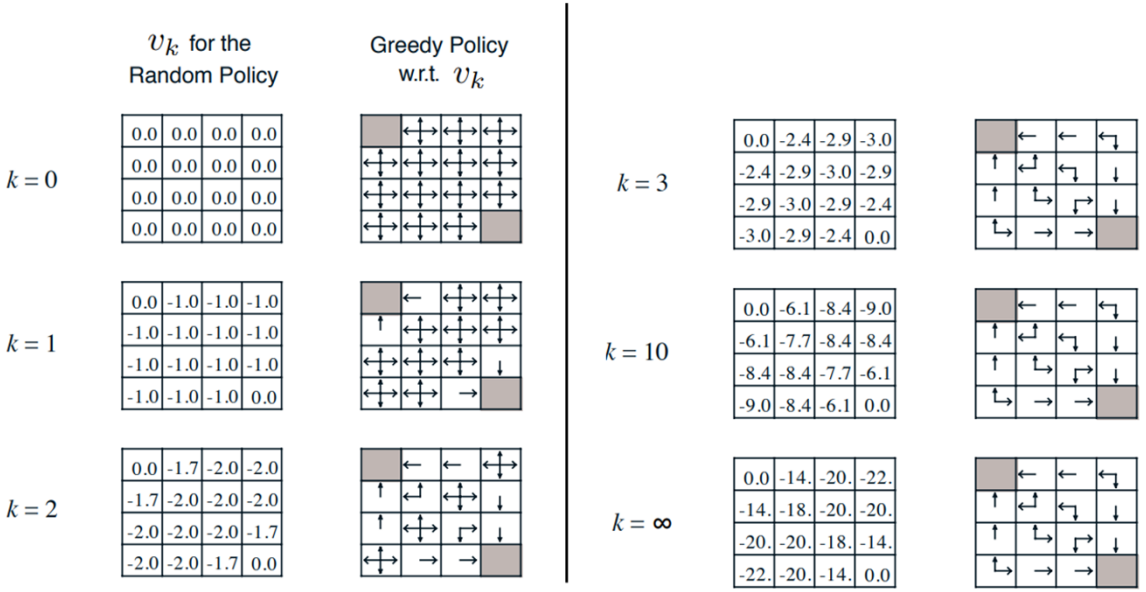
- Definition: a **state-*value function***, $v_\pi(s)$, of an MDP is the expected return starting from state, s, and then following the policy π. In general, how good is it to be in this state.

  $v_\pi(s) = \mathbb{E}_\pi[\, G_t \mid \mathrm{S_t} = \mathrm{s}]$

- Definition: an ***action-value (q-value) function***, $q_\pi(s,a)$, is the expected return starting from state, s, taking action, a, and following policy, π. In general, how good it is to take this action.

  $q_\pi(s,a) = \mathbb{E}_\pi[\, G_t \mid A_t = a, S_t = s]$

Grid world – Bellman equation



$$v_{k+1}(s) = \mathbb{E}_\pi \left[R_t + \gamma v_k(S_{t+1}) \mid S_t = s \right]$$

$$v_{k=1}(s[1,1]) = -1 + 0.25 \cdot (\underbrace{v_{k=0}(s[0,1])}_{0} + \underbrace{v_{k=0}(s[2,1])}_{0} + \underbrace{v_{k=0}(s[1,0])}_{0} + \underbrace{v_{k=0}(s[1,2])}_{0}) = -1.0$$

$$v_{k=2}(s[1,1]) = -1 + 0.25 \cdot (\underbrace{v_{k=1}(s[0,1])}_{-1} + \underbrace{v_{k=1}(s[2,1])}_{-1} + \underbrace{v_{k=1}(s[1,0])}_{-1} + \underbrace{v_{k=1}(s[1,2])}_{-1}) = -2.0$$

Q-learning

- **Goal**: Find a Q-function satisfying the Bellman (optimality) equation.
- We parameterize our action-value function using a neural network.
- Since we get training data collected by the agent itself only, we want to balance exploration and exploitation.
  - D_i is your dataset with state action pairs $s_t, s_{t+1,} a_t, r_t$

- $Q_*(s_t, a_t, \theta_i) = \mathbb{E}\left[R_t + \gamma \max_{a_{t+1}} Q_*(s_{t+1}, a_{t+1}, \theta_{i-1}) \mid A_t = a_t, S_t = s_t \right]$

- Reference:

$$y_i = \mathbb{E}\left[R_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}, \theta_{i-1}) \mid A_t = a_t, S_t = s_t \right]$$

- Loss:

$$L_i(\theta_i) = \mathbb{E}_{s_t, s_{t+1,}\, a_t, r_t \sim D_i}\left[\left(y_i - Q(s_t, a_t, \theta_i) \right)^2 \right]$$

Policy based methods

- **Goal:**
  - The goal is to use experience/samples to try to make a policy better.
  - Maximize the objective function: $\mathcal{J}(\theta) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi_\theta\right]$

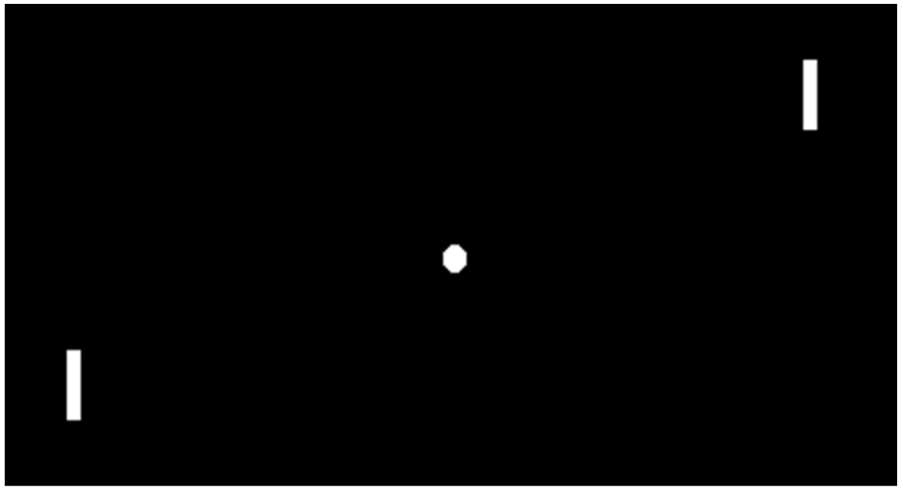
- **Idea:**
  - If a trajectory achieves a high reward, the actions were good
  - If a trajectory achieves a low reward, the actions were bad
  - We will use gradients to enforce more of the good actions and less of the bad actions. Hence the method is called Policy Gradients.

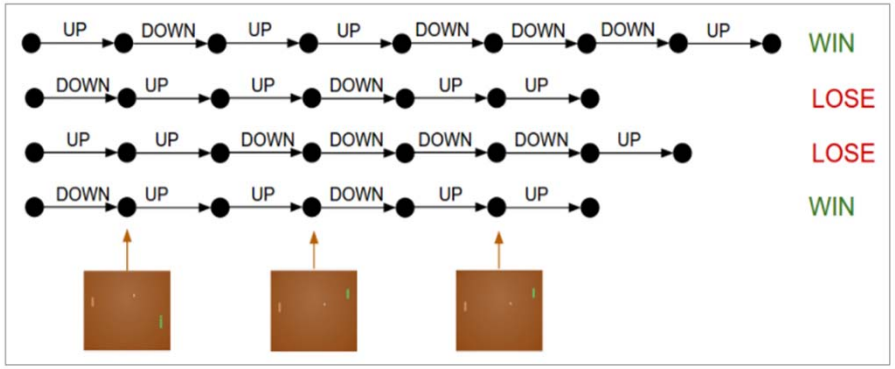
- Reinforce algorithm: We can sample a trajectory to get an estimate of the gradient.

$$\nabla_\theta \mathcal{J}(\theta) \approx \sum_{t\geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Policy gradients: High variance, example game of Pong

- A challenge is that you don't know which action caused the victory?
- In a winning series there may be many non-optimal actions
- In a losing series there may be good actions
- The **true** effect is found by averaging out the noise, as winnings series tend to have more good action and visa versa





May 22, 2019

About the exam

- Inspera, only in English, but your can answer in Norwegian or English
- Around 20-25 questions about variouse topics in the course.
- Calculator allowed (and available in Inspera), but you should not really need it.
- Same exam for master and PhD, additional questions marked “PhD only” to be answered only by PhD students
- Include partial results for computation to get some score even if the answer is incorrect.
- Score from 0-10 on every subtask.
- Exam rounds around 1-1.5 hours after the start.
- If you lack information, assume something or state your assumption. Ask us if you want.
- Please submit “FUI Kurskritikk” after the exam and give us feedback on how to improve the course.
- Consider being a group teacher next spring.

Batch normalization algorithm - training

- For a given node and a given minibatch, compute the mean μ_k and σ_k.
- First, create zero mean, unit variance: $\hat{z}_k = (z_k - \mu_k)/\sigma_k$
- Experiments have shown that we should allow a scaling to avoid limiting what the node can express:

$$\tilde{z_k} = \gamma \hat{z}_k + \beta$$

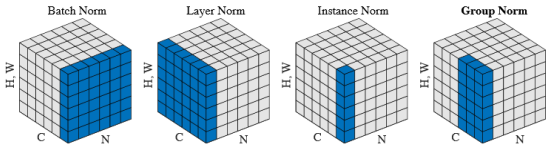
- γ andβ are learned using backpropagation, and they are specific to the layer.
- Using this scaling normally speeds up the convergence by getting more effective gradients.
- **Batch normalization significantly speeds up gradient descent and even improves performance, try it!**
- Store γ and β.

Batch normalization - what to do at test time

- At test time: we need mean and standard deviation should we use to normalize.
- The best is to use the mean and standard deviation over the entire training data set.
- This can be efficiently computed using moving average estimates over the mini batches, apply this during traning and store μ_k andσ_k.

Other types of normalization

- Normalization is important for efficient gradient flow
- Batch norm normalize across a batch, but we can also normalize across channels, instances, or groups of channels.



- A good reference is https://arxiv.org/pdf/1803.08494.pdf

GRADIENT DESCENT UPDATES

Parameter-specific update schemes

- SGD and SGD with momentum updates all weight/parameters using the same scheme
- Other methods scale the update by the size of the weights/parameters:
- We will look at ADAM, but other choices are AdaGrad or RMSprop.
  - AdaGrad http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf
    - Accumulates weight gradients, these can build up and is not so often used.
  - RMSprop
    - Introduce a cache of moving average of the gradients of each weight
  - ADAM https://arxiv.org/abs/1412.6980
    - Combines both momentum and a moving average of the gradients of each weight

ADAM - MAIN IDEA WITHOUT FIX FOR THE FIRST ITERATIONS

ADAM update, all variables are vectors

1. Set $\rho_1 = 0.9$, $\rho_2 = 0.999$, $\epsilon = 1e-8$.
2. Update the mean (first order moment) $\mu_{\partial w}$ and the non-centered variance (second order moment) $var_{\partial w}$ of ∂w.

$$\mu_{\partial w} = \rho_1 \mu_{\partial w} + (1 - \rho_1)\partial w$$
$$var_{\partial w} = \rho_2 var_{\partial w} + (1 - \rho_2)(\partial w)^2$$

3. Take a scaled step:

$$w = w - \lambda \frac{\mu_{\partial w}}{(\sqrt{var_{\partial w}} + \epsilon)}$$

ADAM - ALGORITHM

ADAM update with bias correction for the first iterations:

- Set $\rho_1 = 0.9$, $\rho_2 = 0.999$, $\epsilon = 1e-8$.
- For $t = 1 : maxiter$

$$\mu_{\partial w} = \rho_1 \mu_{\partial w} + (1 - \rho_1)\partial w$$
$$\mu_t = \mu_{\partial w}/(1 - \rho_1^t)$$
$$var_{\partial w} = \rho_2 var_{\partial w} + (1 - \rho_2)(\partial w)^2$$
$$v_t = var_{\partial w}/(1 - \rho_2^t)$$
$$w = w - \lambda \frac{\mu_t}{(\sqrt{v_t} + \epsilon)}$$

Recommendations for weigth updates

- SGD with momentum, try learning rate decay too.
  - Also used weigth decay - covered in next lecture
- ADAM also works well
  - If using weigth decay, be aware that this a currently a matter of discussion https://www.fast.ai/2018/07/02/adam-weight-decay/

REPETITION LECTURE

IN5400 — Machine Learning for Image Analysis

Ole-Johan Skrede

22.05.2019

University of Oslo

General

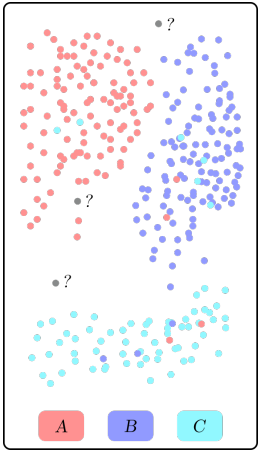
- Many specific networks was briefly covered (segmentation, object detection, adversarial domain adaptation, etc.). In those cases, no detailed knowledge beyond what was lectured is required.
- Important results was derived “from scratch” to enhance understanding. If you want detailed knowledge about a certain thing, you should know where it comes from and why.

t-distributed Stochastic Neighbour Embedding (t-SNE)

- Transforms high-dimensional (hd) data points to low-dimensional (ld) data ponts
- Aims to preserve neighbourhood identity between data points
- imilar (close) hd points should also be similar (close) in the ld representation
- For each point i, we define two distributions:
  - $P_i(x_j)$: Describes the probability that hd point j is the "neighbour" of hd point i, given its location x_i
  - $Q_i(y_j)$: Describes the probability that ld point j is the "neighbour" of ld point i, given its location y_i
- For hd points, we use symmetric gaussian distributions
- For ld points, we use symmetric student-t distributions
- The aim is to make the distributions similar
- We do this by minimizing the KL-divergence between the two
- The KL-divergence is minimized by adjusting the ld points y with gradient descent

Problems with autoencoders for signal generation

- An autoencoder works great if you want to reconstruct a *replica* of the input
- Not well suited for generating new signal
- The reason for this is an “unintuitive” latent variable space
- The latent space might be discontinuous
- Random sampling from an “unseen” region of the latent space produces unpredictable results
- No reasonable way to interpolate between categories in the latent space

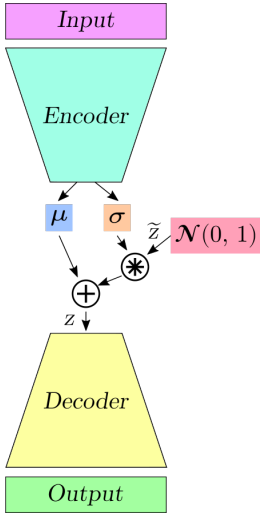


Variational autoencoders

- A variational autoencoder is designed to have a continuous latent space
- This makes them ideal for random sampling and interpolation
- It achieve this by forcing the encoder g to generate Gaussian representations, $z \sim \mathcal{N}(\mu, \sigma^2)$
- More precisely, for one input, the encoder generates a mean μ and a variance σ^2
- We then sample a zero-mean, unit-variance Gaussian $\tilde{z} \sim \mathcal{N}(0, 1)$
- Construct the input z to the decoder from this

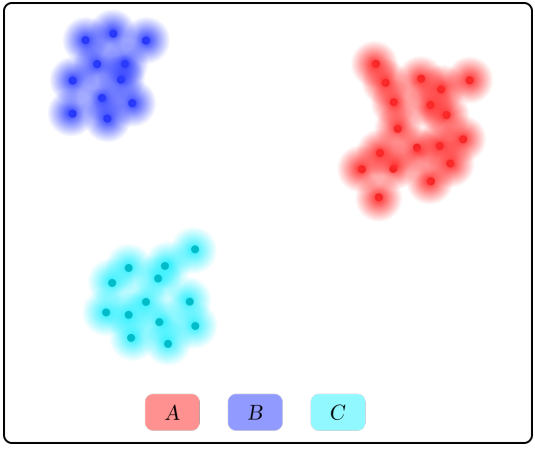
$$z = \mu + \tilde{z} \cdot \sigma$$

- With this, z is sampled from $q = \mathcal{N}(\mu, \sigma^2)$



Problem

- No restriction on μ or σ^2
- Realisticly, clusters of different classes can be placed far apart
- Leaves “empty space” in between with unknown sampling features



Guiding the generative distribution

- We can guide the solutions by restricting the generative distribution q
- We do this by making it approximate some distribution p
- In that way, the latent vectors, even for different categories, will be relatively close
- The desired distribution used in variational autoencoders is the standard normal $p = \mathcal{N}(0, 1)$
- We use the familiar KL-divergence between the desired and the generated distribution as a regularizer in the loss function
- With this, the total loss for an example x_i is something like

$$L(x_i) = ||x^{(i)} - f(x^{(i)})|| + D_{KL}(p||q_{\mu_i,\sigma_i})$$

- That is, the sum of what we call the *reconstruction loss* and the *latent loss*
- The latent loss for a single input $x^{(i)}$ can be shown to be equal to

$$D_{KL}(p||q_{\mu_i,\sigma_i}) = \frac{1}{2}(\mu_i^2 + \sigma_i^2 - \log \sigma_i^2 - 1)$$