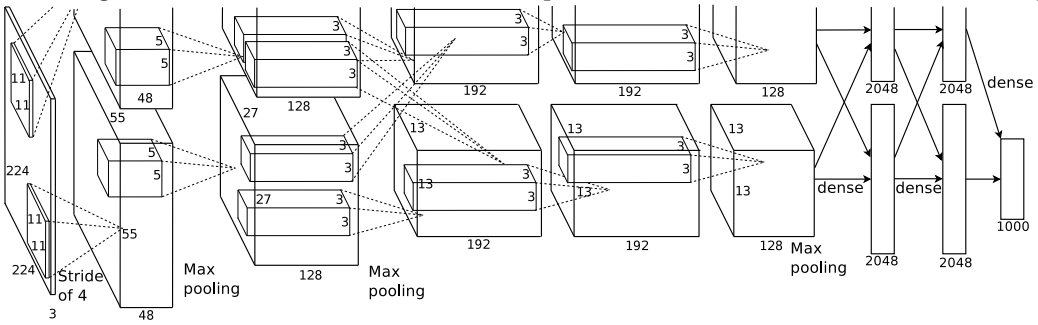
# Deep Learning – NPFL114 – Exam Topics

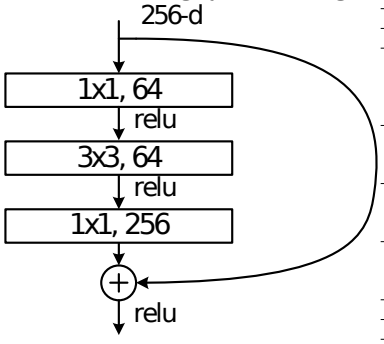
Generally, only the topics covered on the lecture are part of the exam *(i.e., you should be able to tell me what I told you)*. The references are to Deep Learning Book, unless stated otherwise.

- Computation model of neural networks
  - acyclic graph with nodes and edges
  - evaluation (forward propagation) *(Algorithm 6.1)*
  - activation functions *(*tanh *and* ReLU*, including equations)*
  - output functions *(*$\sigma$ *and* softmax*, including equations (3.30 and 6.29); you should also know how* softmax *is implemented to avoid overflows)*
- Backpropagation algorithm *(Algorithm 6.2) (Algorithms 6.5 and 6.6 are used in practise, i.e., during* `tf.train.Optimizer.compute_gradients`*, so you should understand the idea behind them, but you do not have to understand the notation of* `op.bprop` *etc.)*
- Gradient descent and stochastic gradient descent algorithm *(Section 5.9)*
- Maximum likelihood estimation (MLE) principle *(Section 5.5, excluding 5.5.2)*
  - negative log likelihood as a loss derived by MLE
  - mean square error loss derived by MLE from Gaussian prior *(Equations (5.64)-(5.66))*
- *In addition to have theoretical knowledge of the above, you should be able to perform all of it on practical examples – i.e., if you get a network with one hidden layer, a loss and a learning rate, you should perform the forward propagation, compute the loss, perform backpropagation and update weights using SGD. In order to do so, you should be able to derivate softmax with NLL, sigmoid with NLL and linear output with MSE.*
- Stochastic gradient descent algorithm improvements *(you should be able to write the algorithms down and ideally understand motivations behind them)*:
  - learning-rate decay
  - SGD with momentum *(Section 8.3.2 and Algorithm 8.2)*
  - SGD with Nestorov Momentum *(and how it is different from normal momentum) (Section 8.3.3 and Algorithm 8.3)*
  - AdaGrad *(Section 8.5.1 and Algorithm 8.4)*
  - RMSProp *(and why is it a generalization of AdaGrad) (Section 8.5.2 and Algorithm 8.5)*
  - Adam *(and why the bias-correction terms* $(1 - \beta^t)$ *are there) (Section 8.5.3 and Algorithm 8.7)*
- Gradient clipping *(Section 10.11.1)*
- Regularization methods:
  - Early stopping *(Section 7.8, without the* ***How early stopping acts as a regularizer*** *part)*
  - L2 regularization *(First paragraph of 7.1.1 and Equation (7.5))*
  - L1 regularization *(Section 7.1.2 up to Equation (7.20))*
  - Dropout *(just the description of the algorithm)*
  - Batch normalization *(Section 8.7.1)*

- Convolutional networks:
  - Basic convolution and cross-correlation operation *(Equations (9.5) and (9.6))*
  - Differences compared to a fully connected layer *(Section 9.2 and Figure 9.6)*
  - Multiple channels in a convolution *(Equation (9.7))*
  - Stride and padding schemes *(Section 9.5 up to page 349, notably Equation (9.8))*
  - Max pooling and average pooling *(Section 9.3)*
  - AlexNet *(general architecture, without knowing specific constants, i.e., the following image which is taken from* ***Alex Krizhevsky et al.: ImageNet Classification with Deep Convolutional Neural Networks*** `https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`)




  - ResNet *(only the important ideas and overall architecture of ResNet 151, without specific constants; the following is taken from* ***Kaiming He et al.: Deep Residual Learning for Image Recognition*** `https://arxiv.org/abs/1512.03385`)




| layer name | output size | 18-layer | 34-layer | 50-layer | 101-layer | 152-layer |
|---|---|---|---|---|---|---|
| conv1 | 112×112 | 7×7, 64, stride 2 | | | | |
| conv2_x | 56×56 | 3×3 max pool, stride 2 | | | | |
| | | 3×3, 64<br>3×3, 64 ×2 | 3×3, 64<br>3×3, 64 ×3 | 1×1, 64<br>3×3, 64 ×3<br>1×1, 256 | 1×1, 64<br>3×3, 64 ×3<br>1×1, 256 | 1×1, 64<br>3×3, 64 ×3<br>1×1, 256 |
| conv3_x | 28×28 | 3×3, 128<br>3×3, 128 ×2 | 3×3, 128<br>3×3, 128 ×4 | 1×1, 128<br>3×3, 128 ×4<br>1×1, 512 | 1×1, 128<br>3×3, 128 ×4<br>1×1, 512 | 1×1, 128<br>3×3, 128 ×8<br>1×1, 512 |
| conv4_x | 14×14 | 3×3, 256<br>3×3, 256 ×2 | 3×3, 256<br>3×3, 256 ×6 | 1×1, 256<br>3×3, 256 ×6<br>1×1, 1024 | 1×1, 256<br>3×3, 256 ×23<br>1×1, 1024 | 1×1, 256<br>3×3, 256 ×36<br>1×1, 1024 |
| conv5_x | 7×7 | 3×3, 512<br>3×3, 512 ×2 | 3×3, 512<br>3×3, 512 ×3 | 1×1, 512<br>3×3, 512 ×3<br>1×1, 2048 | 1×1, 512<br>3×3, 512 ×3<br>1×1, 2048 | 1×1, 512<br>3×3, 512 ×3<br>1×1, 2048 |
| | 1×1 | average pool, 1000-d fc, softmax | | | | |
| FLOPs | | $1.8\times10^9$ | $3.6\times10^9$ | $3.8\times10^9$ | $7.6\times10^9$ | $11.3\times10^9$ |

- Recurrent networks:
  - Using RNNs to represent sequences *(Figure 10.2 with $h$ as output; Chapter 10 and Section 10.1)*
  - Using RNNs to classify every sequence element *(Figure 10.3; details in Section 10.2 excluding Sections 10.2.1-10.2.4)*
  - Bidirectional RNNs *(Section 10.3)*
  - Encoder-decoder sequence-to-sequence RNNs *(Section 10.4; note that you should know how the network is trained and also how it is later used to predict sequences)*
  - Stacked (or multi-layer) LSTM *(Figure 10.13a of Section 10.10.5; more details (not required for the exam) can be found in* ***Alex Graves: Generating Sequences With Recurrent Neural Networks*** `https://arxiv.org/abs/1308.0850`)
  - The problem of vanishing and exploding gradient *(Section 10.7)*
  - Long Shoft-Term Memory (LSTM) *(Section 10.10.1)*
  - Gated Recurrent Unit (GRU) *(Section 10.10.2)*

- Word representations *(in all cases, you should be able to describe the algorithm for computing the embedding, and how the backpropagation works [there is usually nothing special, but if I ask what happens if a word occurs multiple time in a sentence, you should be able answer])*:
  - The `word2vec` word embeddings
    - CBOW and Skip-gram models *(**Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean: Efficient Estimation of Word Representations in Vector Space** `https://arxiv.org/abs/1301.3781`)*
    - Hierarchical softmax *(Section 12.4.3.2, or Section 2.1 of the following paper)*
    - Negative sampling *(Section 2.2 of **Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean: Distributed Representations of Words and Phrases and their Compositionality** `https://arxiv.org/abs/1310.4546`; note that negative sampling is a simplification of Importance sampling described in Section 12.4.3.3, with $w_i = 1$; the proposal distribution in `word2vec` being unigram distribution to the power of 3/4)*
  - Character-level embeddings using RNNs *(C2W model from **Wang Ling, Tiago Lus, Lus Marujo, Ramn Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W. Black, Isabel Trancoso: Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation** `http://arxiv.org/abs/1508.02096`)*
  - Character-level embeddings using CNNs *(CharCNN from **Yoon Kim, Yacine Jernite, David Sontag, Alexander M. Rush: Character-Aware Neural Language Models** `https://arxiv.org/abs/1508.06615`)*
  - Character-level embeddings using character n-grams *(described simultaneously in several papers, read for example **Piotr Bojanowski, Edouard Grave, Armand Joulin, Tomas Mikolov: Enriching Word Vectors with Subword Information** `https://arxiv.org/abs/1607.04606`)*
- Machine Translation
  - Translation using encoder-decoder (also called sequence-to-sequence) architecture *(Sections 10.4 and Section 12.4.5)*
  - Attention mechanism in NMT *(Section 12.4.5.1, but you should also know the equations for the attention, notably Equations (4), (5), (6) and (A.1.2) of **Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio: Neural Machine Translation by Jointly Learning to Align and Translate** `https://arxiv.org/abs/1409.0473`)*
  - Subword units *(The BPE algorithm from Section 3.2 of **Rico Sennrich, Barry Haddow, Alexandra Birch: Neural Machine Translation of Rare Words with Subword Units** `https://arxiv.org/abs/1508.07909`)*

- Reinforcement learning *(note that proofs are not required for reinforcement learning; all references are to the Sep 2016 draft of second edition of* ***Reinforcement Learning: An Introduction by Richar S. Sutton*** `http://ufal.mff.cuni.cz/~straka/courses/npfl114/2016/sutton-bookdraft2016sep.pdf`)
  - Multi-arm bandits *(Chapter 2, Sections 2.1-2.3)*
  - General setting of reinforcement learning *(agent-environment, action-state-reward, return; Chapter 3, Sections 3.1-3.3)*
  - Monte Carlo reinforcement learning algorithm *(Algorithm in Section 5.4)*
  - Q-Learning *(Algorithm in Section 6.5; you should also understand Eq. (6.1) and (6.2))*
  - Policy gradient methods *(representing policy by the network, using softmax, Section 13.1)*
    * Policy gradient theorem. Use the following formulation:
      Let $\pi(a|s)$ be a policy (computes probability of an action in a given state), $v_\pi(s)$ value function (computes expected return in a given state) and $q_\pi(s,a)$ action-value function (computes expected return for a given state and action; note that $v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a)$). Then

      $$\frac{\partial}{\partial\theta} v_\pi(s_0) = E_{s,a\sim\pi}\left[\frac{\partial \log \pi(a|s)}{\partial\theta} q_\pi(s,a)\right],$$

      where the expectation is computed according to probability that state $s$ and action $a$ will be visited from $s_0$ following policy $\pi$.
    * REINFORCE algorithm *(Equation (13.6) and Algorithm in 13.3, but note that the $\gamma^t$ in (13.6) and in the Algorithm show not be there)*
    * REINFORCE with baseline algorithm *(Equation (13.9) and Algorithm in 13.4, but note that the $\gamma^t$ in (13.9) and in the Algorithm show not be there)*
    * Actor-critic algorithm *(Equation (13.11) and One-step Actor-Critic Algorithm in 13.5, but note that the $\gamma$ on the last but one line of the Algorithm should not be there)*
- Deep generative models using differentiable generator nets *(Section 20.10.2)*:
  - Variational autoencoders *(Section 20.10.3 up to page 698 (excluding), together with Reparametrization trick from Section 20.9 (excluding Section 20.9.1))*
    * Regular autoencoders *(undercomplete AE – Section 14.1, sparse AE – first two paragraphs of Section 14.2.1, denoising AE – Section 14.2.2)*
  - Generative Adversarial Networks *(Section 20.10.4 up to page 702 (excluding))*