INTRODUCTION TO DEEP LEARNING FOR COMPUTER VISION

DAY 4 – BAG OF TRICKS TO START CNNS

SEBASTIAN HOUBEN

EVALUATION

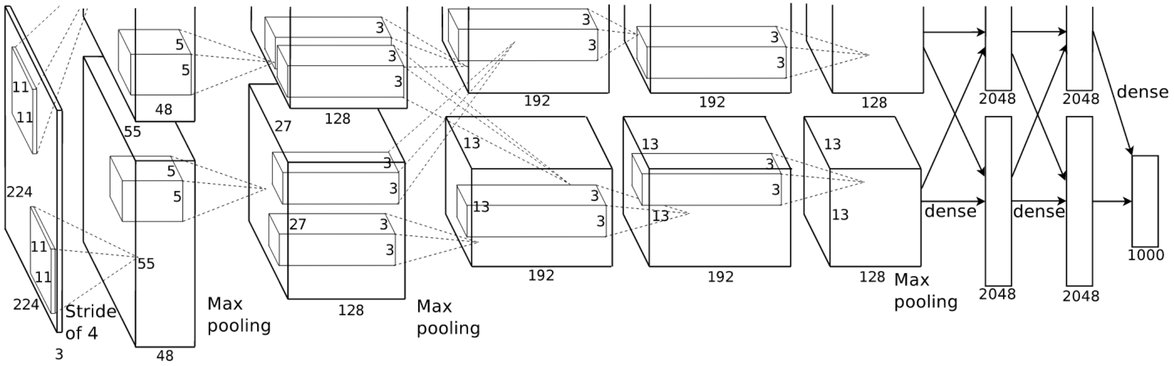
Schedule

Today

- Initialization - *how to start*
- Data Augmentation - *how to prepare*
- Plot Interpretation - *how to train*
- Regularization - *how to fit*
- *Inspection – how to understand*

Convolutional Neural Nets

- Stack repeating elements
  - Small convolutions
  - ReLU
  - MaxPool
  - Fully connected layers



Alex Krizhevsky et al.

Initialization

- Avoid that all weights receive the same gradient
  - Symmetry breaking
  - $w_i \sim \mathcal{N}\left(0, 0.001\right)$
- Avoid that gradient grows (or shrinks) over multiple layers
  - Vanishing gradient
  - Exploding gradient
- Xavier initialization

$$\frac{\partial}{\partial w} L(x, y, w, u) = \sum_{i=1}^{n} 2\left(\sigma\left(u\sigma\left(wx^{(i)}\right)\right) - y\right) \cdot \sigma'\left(u\sigma\left(wx^{(i)}\right)\right) \cdot u\sigma'\left(wx^{(i)}\right) \cdot x^{(i)}$$

Xavier Initialization

$$w_1 x_1 + w_2 x_2 + \cdots + w_d x_d$$

$$\underbrace{Var\left[w_1 x_1 + w_2 x_2 + \cdots + w_d x_d\right]}_{\text{var of output}} = \underbrace{d \cdot Var\left[w_i\right]}_{\overset{!}{=}1} \underbrace{Var\left[x_i\right]}_{\text{var of input}}$$

$$Var\left[w_i\right] = \frac{1}{d_{\text{in}}}$$

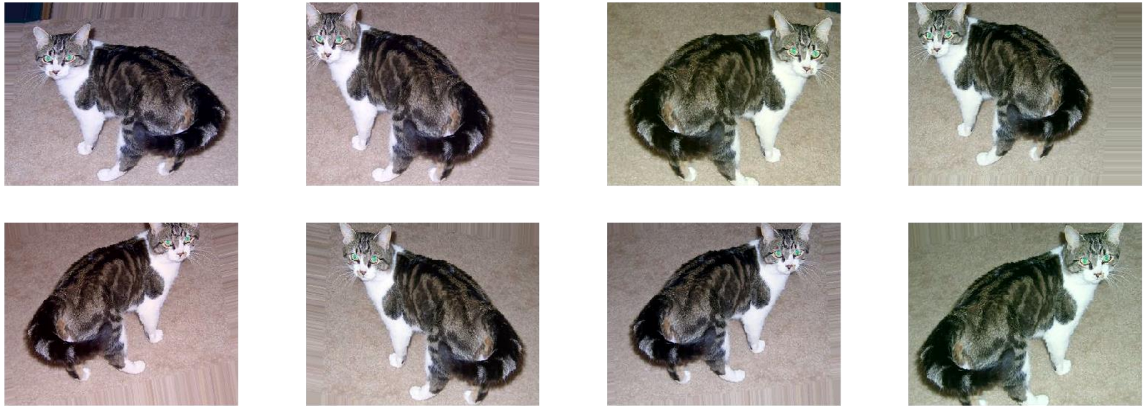
$$Var\left[w_i x_i\right] = Var\left[w_i\right] Var\left[x_i\right]$$

- no non-linearity
- independent x_i

- with ReLU on average half of the inputs are zero: $Var\left[w_i\right] = \frac{2}{d_{\text{in}}}$

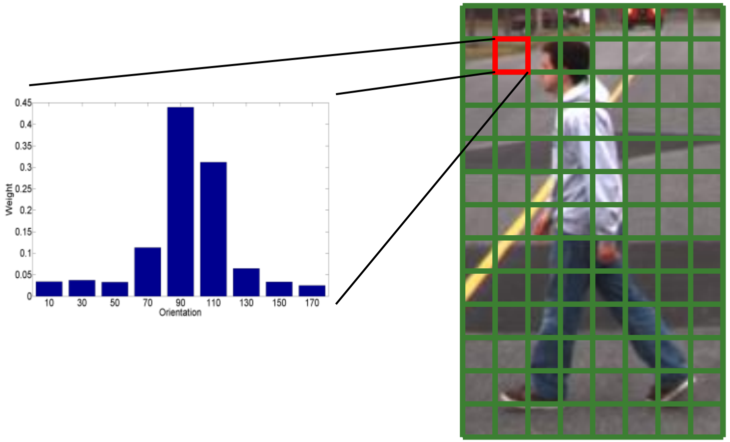
Data Augmentation

- Generate more data (good for most machine learning techniques)
- But Deep Learning in particular
- Train the network on examples with more variance
- Avoid overfitting



Data Augmentation

$$x^{(k)} := \frac{x^{(k)} - \text{mean}(x^{(1)}, ..., x^{(n)})}{\text{std}(x^{(1)}, ..., x^{(n)})}$$



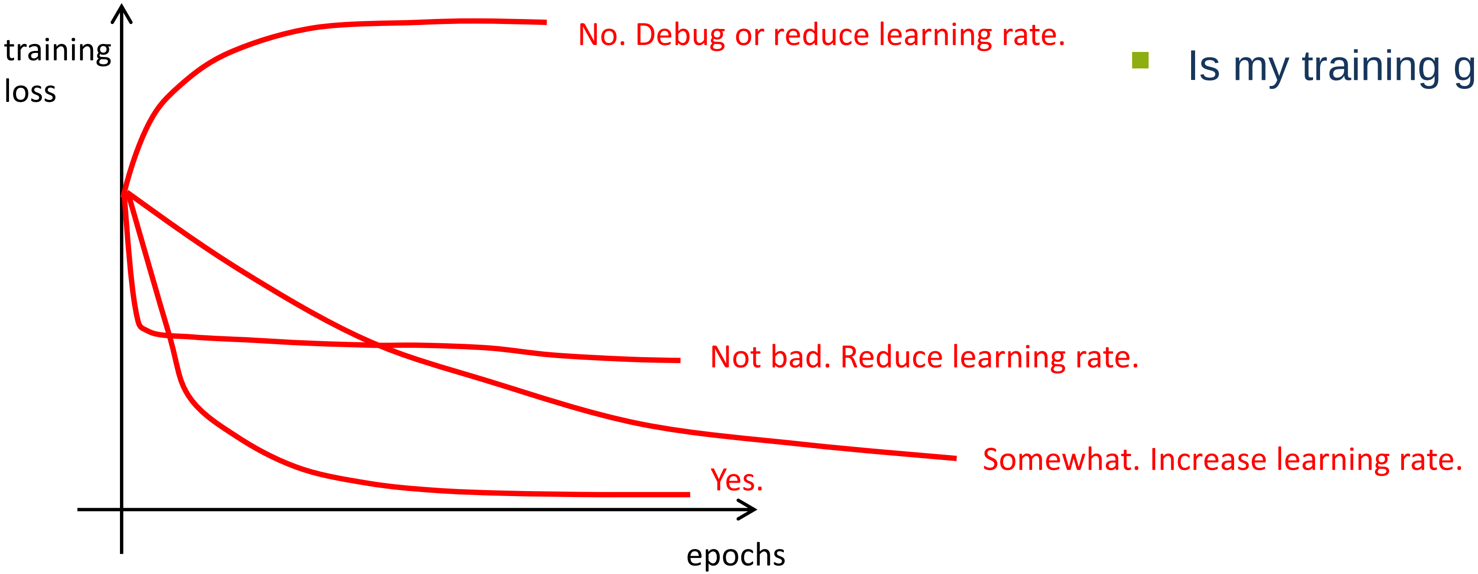
- Recap: Zero-center and normalized range (or standard deviation)
- Transforms the recognition should be invariant to
    - Random crop / shifting
    - Slight rotation
    - Color transform
    - Adding noise (regularization)
    - Horizontal flip
    - Scaling
    - Occlusion
    - Stretching / Shearing

Data Augmentation – Fancy PCA

- Color distribution over all training images (after normalization)
- PCA to find the main axis of variance of this distribution
- Sample from 3d unit Gaussian and transform sample point to color distribution
- Add sampled color vector to all pixels of an image

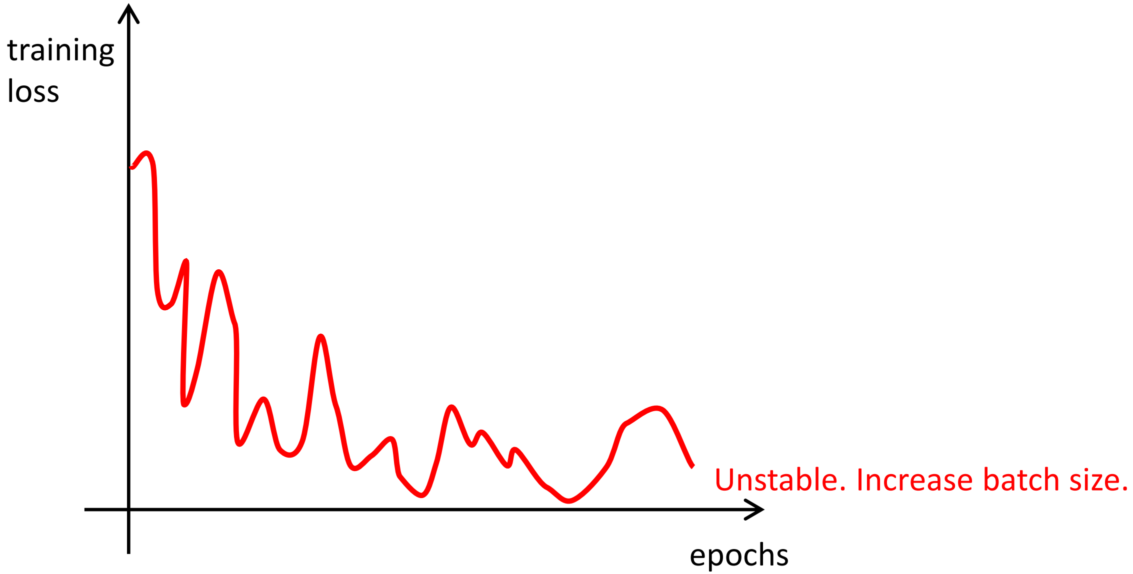
$$\begin{pmatrix} r_1^{(1)} & g_1^{(1)} & b_1^{(1)} \\ r_2^{(1)} & g_2^{(1)} & b_2^{(1)} \\ \vdots & & \\ r_d^{(1)} & g_d^{(1)} & b_d^{(1)} \\ r_1^{(2)} & g_1^{(2)} & b_1^{(2)} \\ \vdots & & \\ r_d^{(n)} & g_d^{(n)} & b_d^{(n)} \end{pmatrix}$$

Plot Interpretation



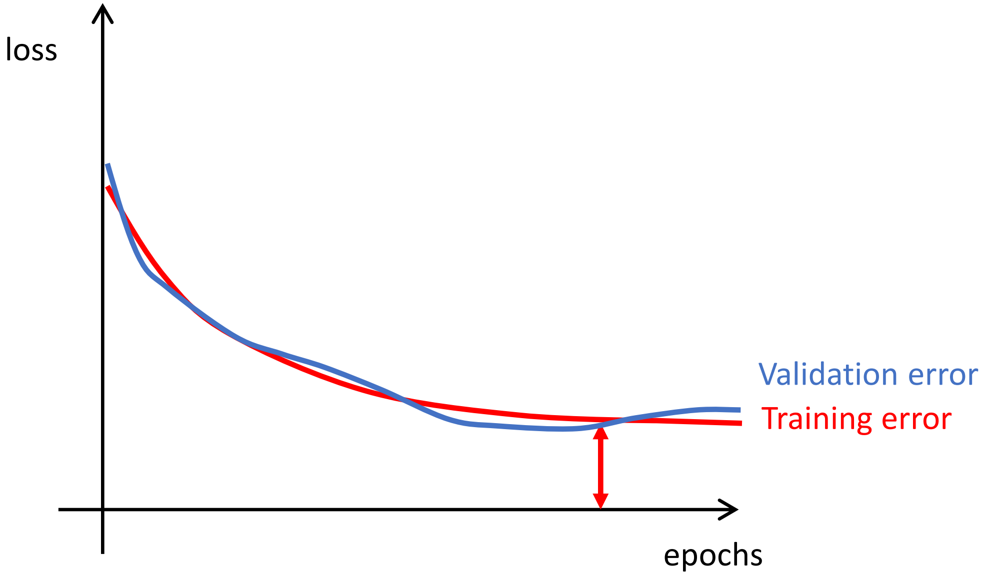
- Is my training going well?

Plot Interpretation



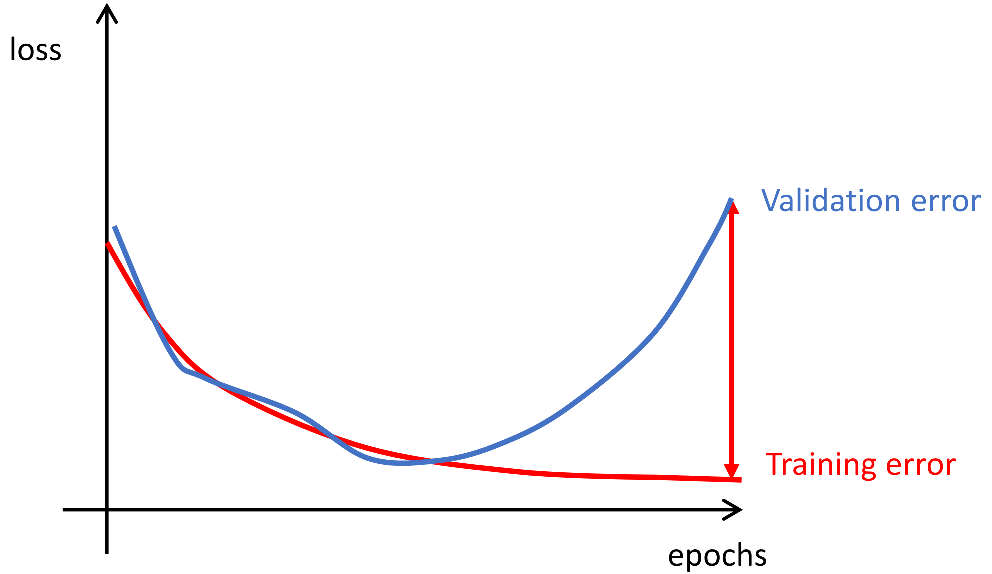
- Is my training going well?

Plot Interpretation



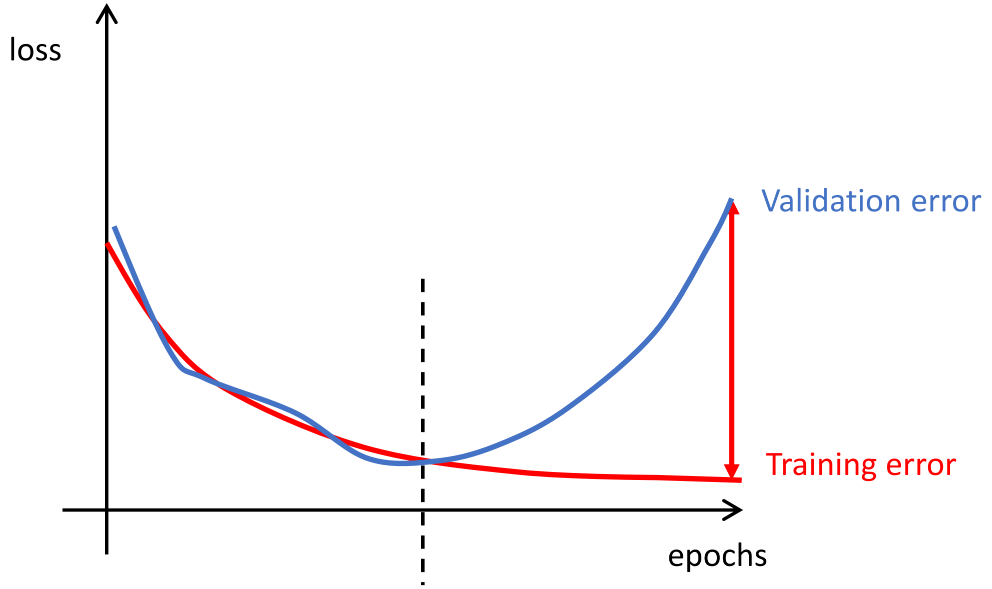
- Underfitting
- Increase model complexity

Plot Interpretation



- Overfitting
- Regularization

Regularization - Early Stopping



- Watch validation error
- Stop training when validation error increases

- Add validation set to training set
  - Restart and stop after same number of epochs
  - Continue until same error level is reached
- Popular, cheap and easy

Regularization – “Weight control“

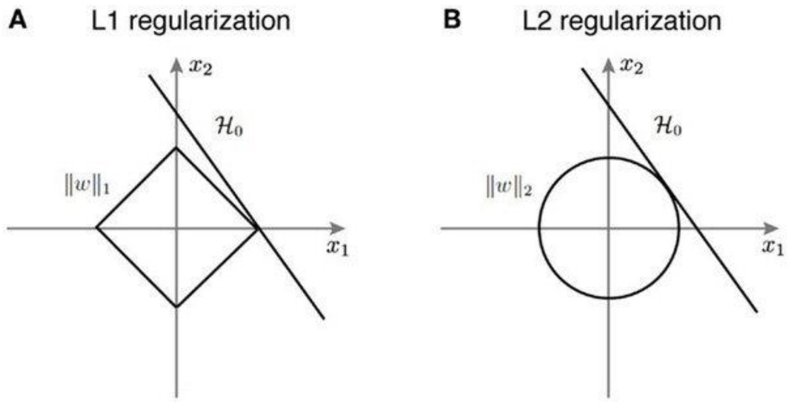
- Overfitting goes along with
    - Large absolute weights
    - Zero weight: „Ignore the feature“
- Only important features should be used
- Ignore as many features as possible

- Also: Large weights result in exploding gradient

- Renormalize weight vectors after update
  (max norm constraints)

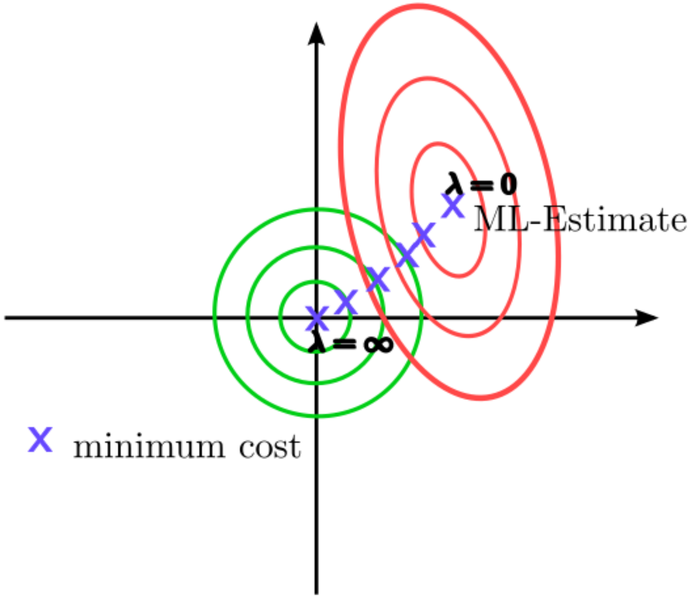
$$L^*(x, y, W, U, \lambda) = L(x, y, W, U, \lambda) + \lambda ||w||_2^2$$

$$L^*(x, y, W, U, \lambda) = L(x, y, W, U, \lambda) + \lambda ||w||_1^2$$

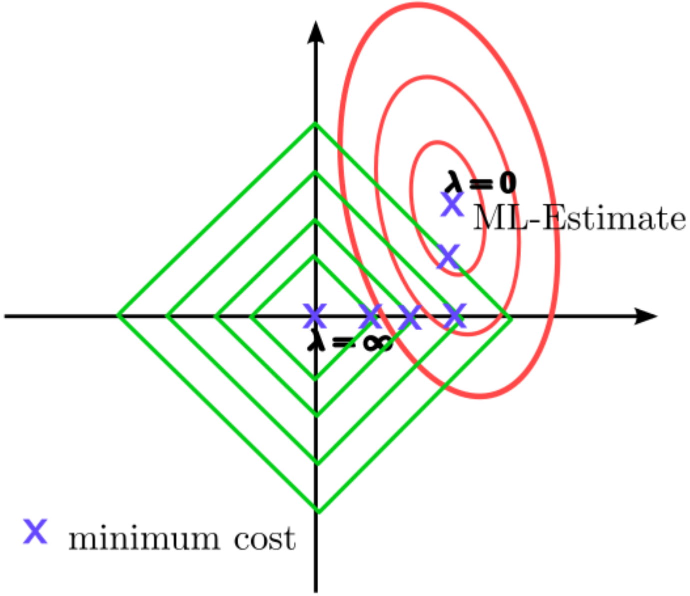


Regularization – “Weight control“

$$L^*(x,y,W,U,\lambda) = L(x,y,W,U,\lambda) + \lambda||w||_2^2$$

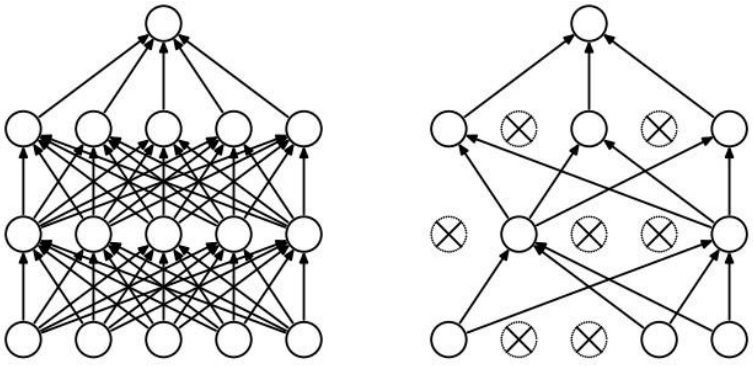


$$L^*(x,y,W,U,\lambda) = L(x,y,W,U,\lambda) + \lambda||w||_1^2$$



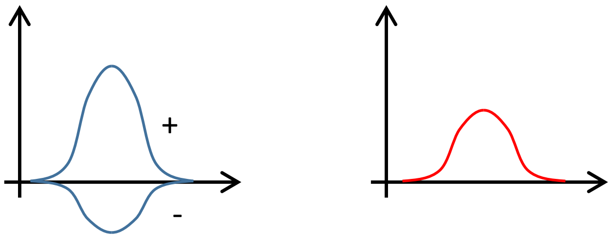
Regularization – Dropout

- Neurons should use each input independently
    - Correlating inputs might be mixed together
    - Waste of resources
    - Should also work if only one of the inputs is present
    - Randomly select *100p%* units and set their output to zero (during training and forward pass)
    - But: Sum of input weights gets lower
        - Increase output by $\frac{1}{p}$
    - During test time activate all neurons (no dropout)



(a) Standard Neural Net

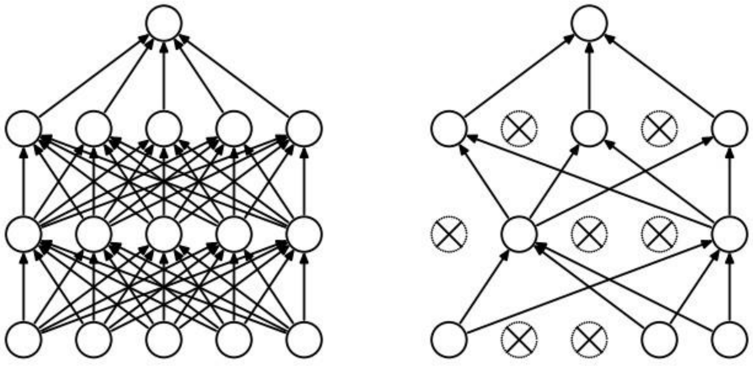
(b) After applying dropout.



RUHR UNIVERSITÄT BOCHUM RUB

Regularization – Dropout

- Extension of Data Augmentation
  - Add noise to input
  - Add noise to intermediate activations
- Alternative interpretation
  - Train 2^n nets at the same time and average their result
  - tensorflow.dropout



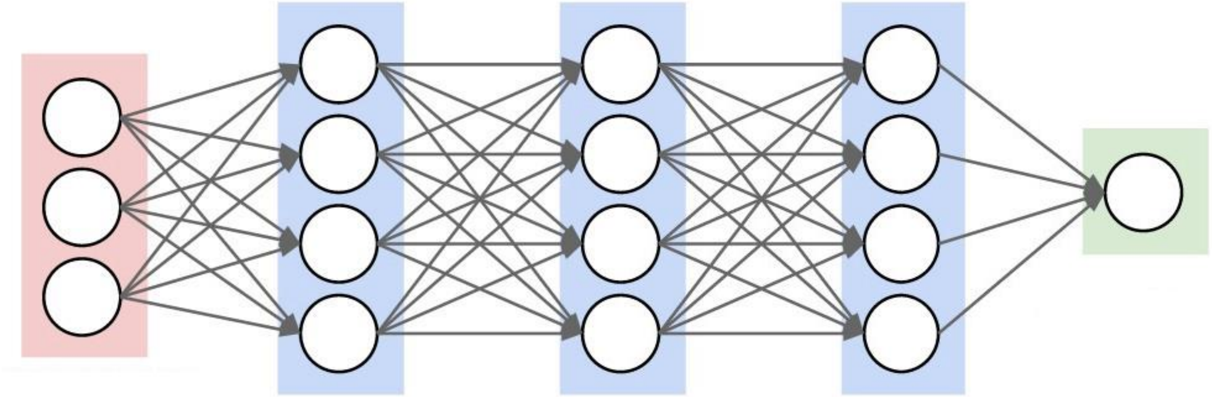
(a) Standard Neural Net

(b) After applying dropout.

Srivastava et al. (2014)

Learning Stability – Batch Normalization

- Good to have features of similar range
- Input / hidden distributions change during learning (covariate shift)
- At least: control mean and variance



Learning Stability – Batch Normalization

- Good to have features of similar range
- Input / hidden distributions change during learning (covariate shift)
- At least: control mean and variance
- For each mini-batch (because it's cheap)
- Also: estimated mean $\mu_{\mathcal{B}}$ and variance $\sigma_{\mathcal{B}}^2$ are a bit noisy (regularization similar to dropout)
- γ, β are parameters (and may counteract the normalization) that must be trained

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Ioffe and Szegedy (2015)

Learning Stability – Batch Normalization

- Also: estimated mean $\mu_{\mathcal{B}}$ and variance $\sigma_{\mathcal{B}}^2$ are a bit noisy (regularization similar to dropout)
- γ, β are parameters (and may counteract the normalization) that must be trained
- Exponentially smooth mean and variance with each batch:
  - $\bar{\mu} \leftarrow \alpha\bar{\mu} + (1-\alpha)\,\mu_{\mathcal{B}}$
  - $\bar{\sigma}^2 \leftarrow \alpha\bar{\sigma}^2 + (1-\alpha)\,\sigma_{\mathcal{B}}^2$
- During test (only one example):
  - Normalize with $\bar{\mu}, \bar{\sigma}^2$
- Higher learning rates might be possible now!

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

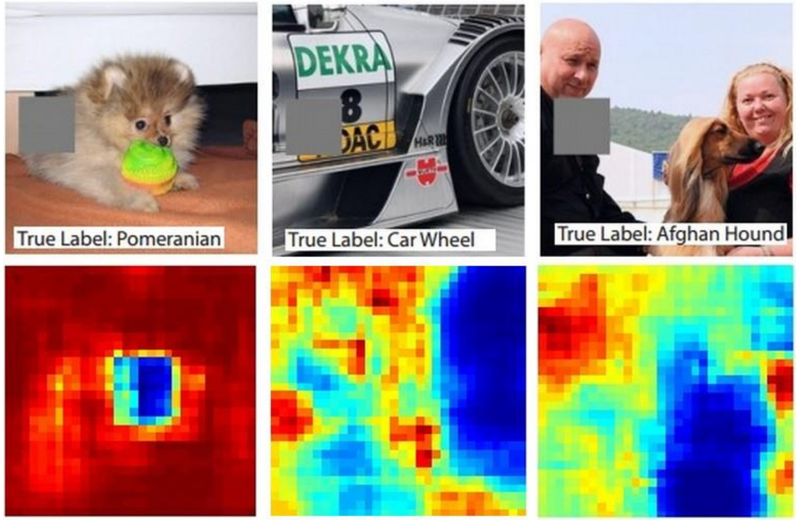
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

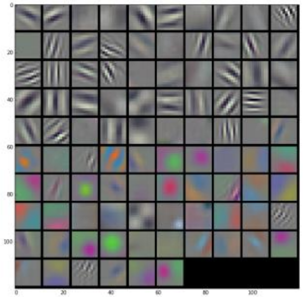
Ioffe and Szegedy (2015)

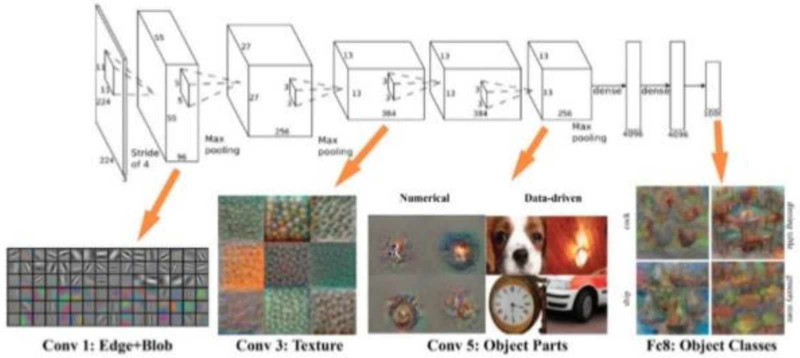
Inspecting a Deep Neural Net



- Occlusion experiments

Inspecting a Deep Neural Net





From: mNeuron: A Matlab Plugin to Visualize Neurons from Deep Models, Donglai Wei et. al.

- Occlusion experiments
- Inspect the first layer of convolutions
- Pick a neuron and look what sample it likes most
- Feed random noise into the network
- Look for Dead ReLUs

QUESTIONS?

EXERCISES.