Regularization for deep-learning models

Ways to adress overfitting

Sebastian Starke

26th September 2018

HZDR

HELMHOLTZ
ZENTRUM DRESDEN
ROSSENDORF

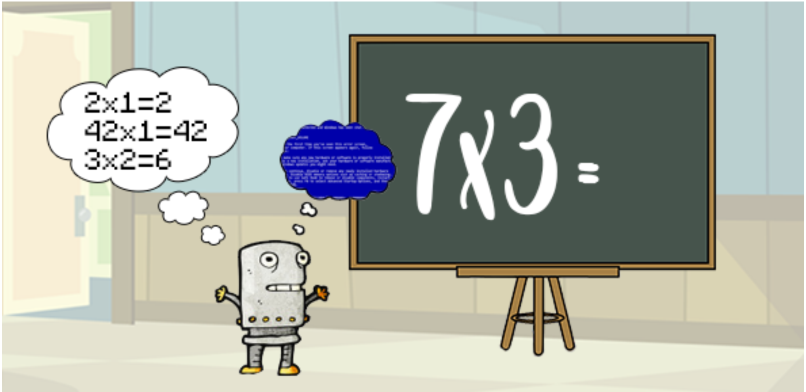

Figure: https://hackernoon.com/
memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-mach

Member of the Helmholtz Association
Sebastian Starke | Department of Information Services and Computing | http//www.hzdr.de

Overfitting

1 Overfitting
- Why does it happen?
- When does it happen?

2 Regularization methods
- Early stopping
- Penalties on the weights
- Dropout
- Data augmentation
- Batch normalization

Overfitting

- the phenomenon of fitting training data too well (learning by heart)
- not capturing general structure but fitting of noise
- loss of ability to generalize to unseen samples

$\Rightarrow$ Tradeoff between capturing training information well enough but not exactly memorize it

Regularization methods aim to reduce overfitting and improve the models ability to generalize to unseen data.

Example of overfitting

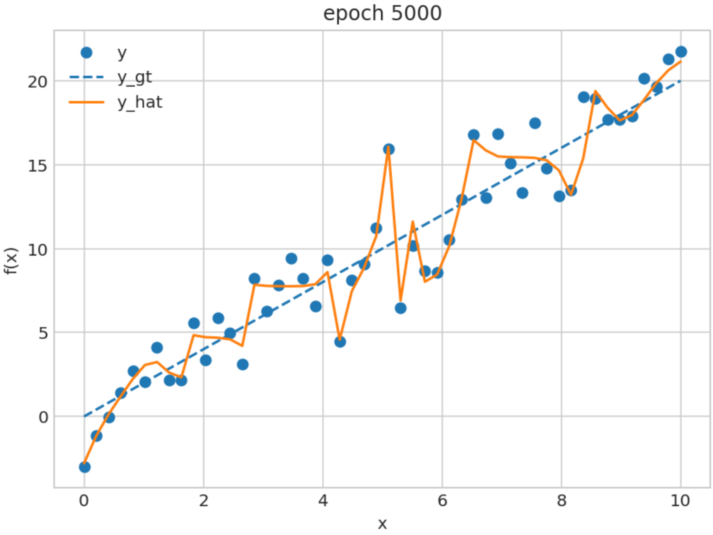

Figure: https://github.com/uschmidt83/keras-intro/blob/master/

Why?

- goal: from representative sample $\Rightarrow$ learn about data-generation mechanism (unknown distribution P)
- model f should minimize the expected error over P $\Rightarrow$ infeasible

$$E_{\boldsymbol{x}\sim P} L(f(\boldsymbol{x}), y)$$

- instead have to minimize over training samples

$$\frac{1}{N}\sum_{i=1}^{N} L\left(f\left(\boldsymbol{x}_i\right), y_i\right) \tag{1}$$

- possible over-adaptation to these N points (which is the mathematical goal, but not what we actually want)

$\Rightarrow$ model might not learn the general concepts

Sebastian Starke | Department of Information Services and Computing | http//www.hzdr.de

When?

- on small datasets
- powerful models ("high capacity model") with many parameters

⇒ **Deep Learning models normally have very high capacity** (especially if you stack many nonlinear layers)

```
In [3]: from keras.applications import VGG16
        print(VGG16().summary())

Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
```

Figure: VGG16.

Regularization methods

1 Overfitting

- Why does it happen?
- When does it happen?

2 Regularization methods

- Early stopping
- Penalties on the weights
- Dropout
- Data augmentation
- Batch normalization

Early stopping

- stop training procedure before model over-adapts

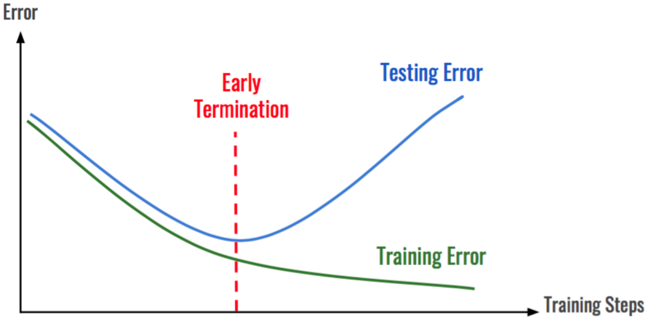

Figure: https://hackernoon.com/
memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-mach

Penalize weights

- constraints on allowed parameters restrict model capacity
- modify the loss function by adding penalty term R ($\lambda > 0$ controls strictness of penalty)

$$\frac{1}{N}\sum_{i=1}^{N} L\left(f(\boldsymbol{x}_i), y_i\right) + \lambda \cdot R(f) \tag{2}$$

- tradeoff between fit and regularization needs to be found by optimizer
- L1 regularization (w_k are weights of the network function f):

$$R(f) = \sum_{k=1}^{M} |w_k| \tag{3}$$

- L2 regularization

$$R(f) = \sum_{k=1}^{M} \|w_k\|^2 \tag{4}$$

Dropout

- randomly knock-out (ignore) neurons of a layer (only during training!)
- implicetly train many sub-networks
- forces the net to distribute its information (all neurons have to be able to do the job)
- might need longer training time

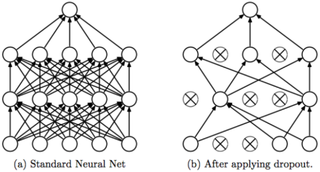

Figure: Subnetwork after randomly dropping some neurons.

Data augmentation

- an easy way to get "more data"
- done by random transformations (rotate, flip, zoom, shift, ...) on training set
- increases variability of your data
- due to randomness, the net can't focus on a small subset ⇒ harder to overfit

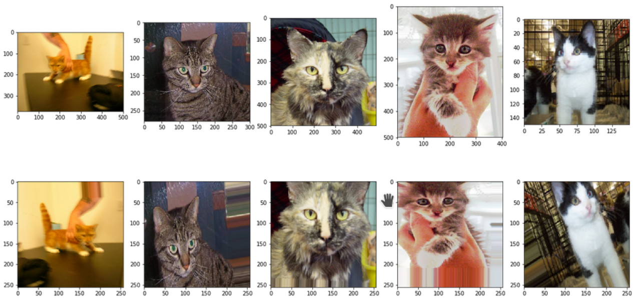

Figure: Original data (top) and augmented data (bottom).

Batch normalization: Motivation

- Motivation: during learning weights change

$\Rightarrow$ neuron outputs change $\Rightarrow$ next layer has to adapt to that change of scale (covariate shift)

- normalize each feature of training batch (zero mean, unit variance for each feature dimension)

$\Rightarrow$ avoids layer inputs to change on orders of magnitude

- inserted before nonlinear activations to avoid saturation (vanishing gradients)
- each input representation influenced by random batch $\Rightarrow$ harder to 'memorize' fixed representation

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
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.

Figure: https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c

- network can focus on learning, not rescaling
- allows higher learning rates
- at test time works differently:
  - can't use batch means and variances
  - use running averages obtained during training

role of γ and β

- last step of algorithm allows rescaling
- parameters are learned during training
- handle cases where normalized data might not be optimal $\Rightarrow$ model learns that

Member of the Helmholtz Association
Sebastian Starke | Department of Information Services and Computing | http//www.hzdr.de

```
day2/notebooks/regularization_cat_dog-mine
```