Regularization

Tamás Grósz

Overfitting

Although it's often possible to achieve high accuracy on the training data, our real goal is to develop models that generalize well to data they haven't seen before.

Overfitting

Although it's often possible to achieve high accuracy on the training data, our real goal is to develop models that generalize well to data they haven't seen before.

Possible causes

- Training the model too long
- Too large model
- Small training set
- The network learns patterns from the training data that don't generalize well (are not present in the test set)

Underfitting

- The opposite of overfitting.
- There is still room for improvement on the test data.

Possible causes

- If the model is not powerful enough -> more neurons and layers
- We over-regularize the model -> less regularization
- The network was not trained long enough -> more epoch

Underfitting

- The opposite of overfitting.
- There is still room for improvement on the test data.

Possible causes

- If the model is not powerful enough -> more neurons and layers
- We over-regularize the model -> less regularization
- The network was not trained long enough -> more epoch

Solution

The solution for bot problem is to use more training data.

Early stopping

- A very simple regularization technique
- Utilizes the validation set simulating the test set during training

Early stopping

- A very simple regularization technique
- Utilizes the validation set simulating the test set during training
- After every epoch, the model is evaluated on the validation data

Early stopping

- A very simple regularization technique
- Utilizes the validation set simulating the test set during training
- After every epoch, the model is evaluated on the validation data
- If we see no improvement on the validation set in the last N epochs, then we stop the training.
- We always back up the best model (on the validation set), and in the end, we restore its weights.

Parameter regularization

Basic principle: we need to prevent the network from having too large weights

Parameter regularization

Basic principle: we need to prevent the network from having too large weights

Weight decay

After each update, the weights are multiplied by a factor slightly less than 1. This prevents the weights from growing too large

Another solution is to add a new penalty term to the loss function.

L1, L2 regularization

We add a penalty term tot the optimized loss function

$$LossRegularized = Loss + \lambda Penalty$$

L1, L2 regularization

We add a penalty term tot the optimized loss function

$$LossRegularized = Loss + \lambda Penalty$$

L1

Penalty term: $\sum_i |w_i|$

L1, L2 regularization

We add a penalty term tot the optimized loss function

$$LossRegularized = Loss + \lambda Penalty$$

L1

Penalty term: $\sum_i |w_i|$

L2

Penalty term: $\sum_i (w_i)^2$

L1, L2 regularization

We add a penalty term tot the optimized loss function

$$LossRegularized = Loss + \lambda Penalty$$

L1

Penalty term: $\sum_i |w_i|$

L2

Penalty term: $\sum_i (w_i)^2$

The key difference between these techniques is that L1 shrinks the less important feature's coefficient to zero thus, removing some feature altogether.
This works well for feature selection in case we have a huge number of features, but reduces the models capacity.

Sparse networks

- The new principle is that the larger and deeper a network is, the better it can perform.
- As a result, computational resources have become a key limiting factor in achieving better performance.

Sparse networks

- The new principle is that the larger and deeper a network is, the better it can perform.
- As a result, computational resources have become a key limiting factor in achieving better performance.
- Forcing sparsity: after each epoch, the small parameters are pruned (changed to 0)

Sparse networks

- The new principle is that the larger and deeper a network is, the better it can perform.
- As a result, computational resources have become a key limiting factor in achieving better performance.
- Forcing sparsity: after each epoch, the small parameters are pruned (changed to 0)
- This way we decrease the computations required to evaluate the net

Ensemble learning

Ensemble learning

Ensemble learning is the process by which multiple models, such as classifiers or experts, are strategically generated and combined to solve a particular computational intelligence problem.

Ensemble learning

Ensemble learning

Ensemble learning is the process by which multiple models, such as classifiers or experts, are strategically generated and combined to solve a particular computational intelligence problem.

- Train multiple small DNNs, then combine their predictions

Ensemble learning

Ensemble learning

Ensemble learning is the process by which multiple models, such as classifiers or experts, are strategically generated and combined to solve a particular computational intelligence problem.

- Train multiple small DNNs, then combine their predictions
- Voting: use majority vote to determine the final prediction

Ensemble learning

Ensemble learning

Ensemble learning is the process by which multiple models, such as classifiers or experts, are strategically generated and combined to solve a particular computational intelligence problem.

- Train multiple small DNNs, then combine their predictions
- Voting: use majority vote to determine the final prediction
- Pobabilistic voting: average the probabilities of the DNNs then choose the most probable one

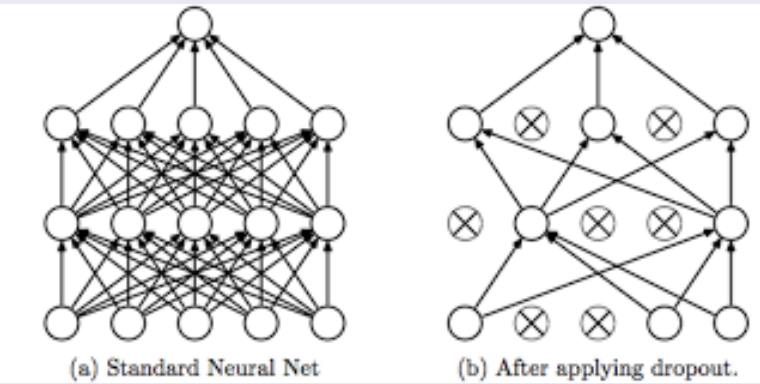
Dropout

- During training we ignore some randomly choosen neurons.
- Technically a random binary mask is generated and applied for each batch and layer.

Dropout

- During training we ignore some randomly choosen neurons.
- Technically a random binary mask is generated and applied for each batch and layer.

Why does it work?



(a) Standard Neural Net (b) After applying dropout.

Dropout forces a neural network to learn more robust features that are useful in conjunction with many different random subsets of the other neurons. One can also say that we train many different small DNNs for each batch (ensemble learning)

Batch normalization

Normalizes the output of a previous hidden layer by subtracting the batch mean and dividing by the batch standard deviation.

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

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

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

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

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

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

Batch normalization

Normalizes the output of a previous hidden layer by subtracting the batch mean and dividing by the batch standard deviation.

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

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

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

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

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

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

- It allows each layer to learn more independently
- We can use higher learning rates because there's no really high or really low activation

Data augmentation

Data augmentation

Data augmentation is another method we can use to avoid overfitting. We simply increase the amount of training data using only original training data.

How do I get more data, if I don't have "more data"?

Data augmentation

Data augmentation

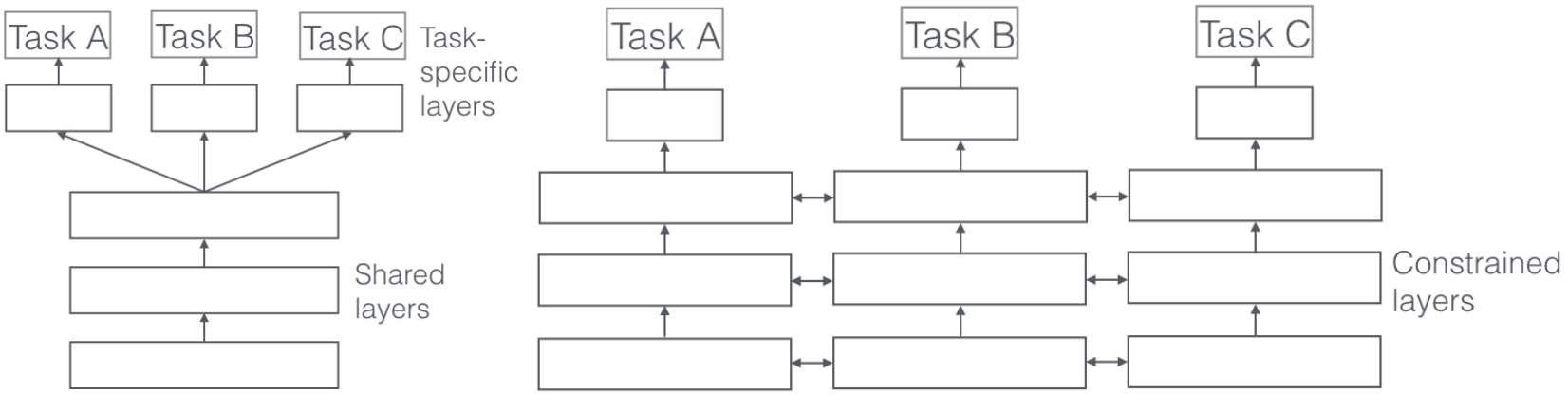
Data augmentation is another method we can use to avoid overfitting. We simply increase the amount of training data using only original training data.

How do I get more data, if I don't have "more data"?
We just need to make minor alterations to our existing dataset:

- Flipping, Cropping, translating, rotating images
- Slowing down or speeding up speech or video
- Adding some noise to the data

Multitask learning

- DNNs usually solve a single problem from a single example
- Sometimes it is beneficial to solve two optimization problems (i.e. two related problems) at the same time
- Weight regularization is basicaly multitask learning



Practice

Python tutorial: practice_06.ipynb