Bias, variance and regularization

Jan Chorowski
Instytut Informatyki
Wydział Matematyki i Informatyki Uniwersytet Wrocławski

2017

Generalization, the goal of learning

- Problem:
  - We care about the performance on **all the data**
  - We have only a **training sample**
  - **Over-fitting**:
    Too powerful classifier will perfectly interpolate the training data (even the noise in it!) and do poorly on **unseen samples**
  - **Under-fitting:**
    Too weak classifier cannot express the relation in the data, even on **training samples**
- Questions:
  - How to estimate the **generalization** (performance on all data)? -> Honest estimates
  - How to control the capacity of a model?
  - Can we provably ensure good generalization performance -> Learning Theory

Honest estimates: Hold-out set

- Large data case!!!
- Split the training data into two parts:

- Train only on training, then test on testing.
- Often we do a three-way split:

- Then:
  - Train many models on training (different algos, parameters)
  - Use validation to choose best model
  - Test on testing

Cross-validation

- Small data case!!
- Hold-out set makes inefficient data use
- Idea:
  - Divide the data into k sets (~5,10)

  For i=1..k

  Train on all but the i-th set

  may further split to choose the model…

  Test on the i-th set

  Finally:

  take the answers on the testing sets and use them to compute the performance measures
- Extreme case: leave-one-out (jackknife) – always use all but one sample to train!

Bootstrap

- Small data case!!
- Sample with replacement m samples
  - About 37% will not be selected
- Train on the selected samples
- Test on the remaining ones
- Optionally repeat.

Bias-Variance: two sources of error!

- The **bias** captures how well our family of functions (hypothesis space) matches the data.
- The **variance** captures how the results of training vary with different samples from the training data

How to lower the bias?

- Choose more powerful/better models:
  - Understand the data and choose a matching model
  - Describe the data with more attributes

  - **More hidden neurons**
  - Better data transformation

- This usually increases the Hypothesis space

How to lower variance?

- Get more data (or generate synthetic, e.g. rotate and shear pictures)
- Select only the most important inputs
- **Constrain the models:**
  - Simpler models
  - Regularize the models:
    Assign a probability distribution to the models and choose the most probable ones
- **Average the models**
  - Very powerful
  - Also called “ensemble learning”, boosting, bagging
  - Requires that the models make uncorrelated errors
    - You can even INJECT randomness to decrease the correlation, e.g. *Random Forests*

Model regularization

- The intuitions:
  - Start with many weights
  - Choose only the ones that we need. How?
  - Force all the weights to decrease
  - Hope that the necessary ones will remain

  - Subtract a little bit in each training iteration:
    $\Theta \leftarrow \Theta - \alpha(\nabla_{\Theta}(J) + \beta\Theta)$ (weight decay)
  - Note: this minimizes $J(\Theta) + \frac{\beta}{2}\sum_j \left(\Theta_j\right)^2$
  - Note: usually you don't decay the biases

Other ideas for NNet regularization

- Choose proper architecture – add or remove neurons or whole layers
- Choose weight decay constants
  - Can also use 1st norm, i.e. $\sum\left|\Theta_j\right|$
    Hint: to gradient train approx $|x| = \sqrt{x^2 + \epsilon}, \epsilon \approx 10^{-4}$
- Share weights between neurons
  - Example: convolutional networks
- Early stop training
  - Monitor validation error as training progresses. Stop when it starts to increase
- Use dropout -> randomly remove some neurons
- Use weight noise
- Use gradient noise

Early stopping

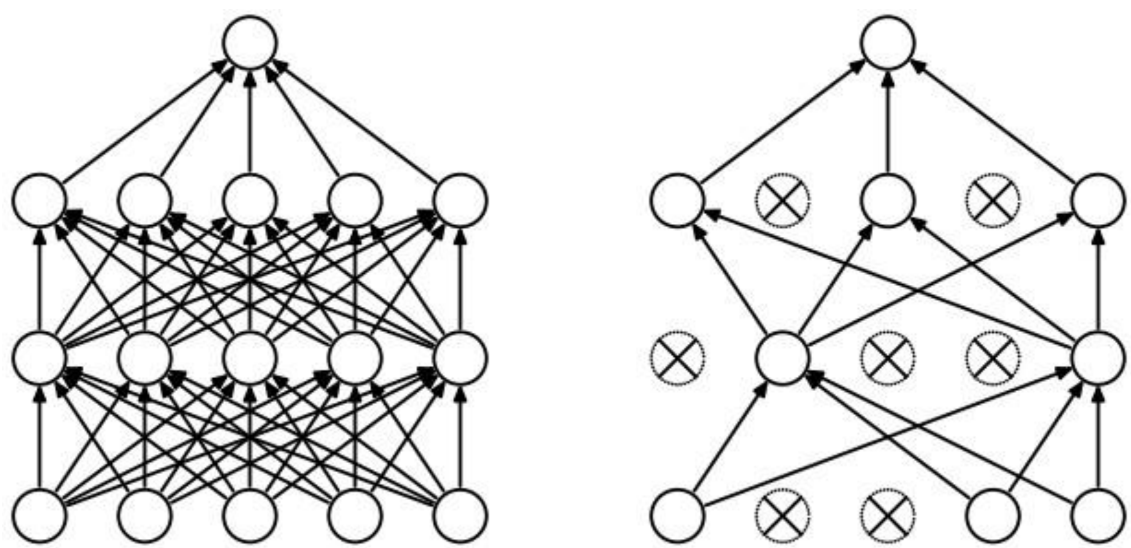
- The net starts with small weights (we initialize it like that)
- Thus it is somewhat linear (all nonlinearities are in the linear range)
- As trainng progresses the net specializes
- At some point, it over-specializes
- Look for that moment, by monitoring a validation error!

Err rate

Training iterations

Dropout

- For each example, select with probability p which neurons will not be used (dropped out).
- Multiply the outputs of other neurons by $1/p$ to compensate
- Enjoy!
- Interpretation:
  - Prevents co-adaptation of neurons, as it is harder for neurons to cooperate is any can be dropped-out
  - Trains infinitely many networks, each sharing selected neurons with the other ones



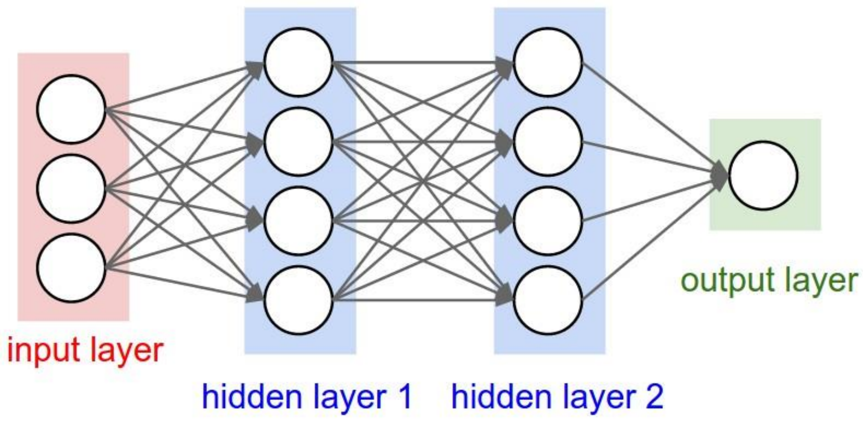
(a) Standard Neural Net (b) After applying dropout.

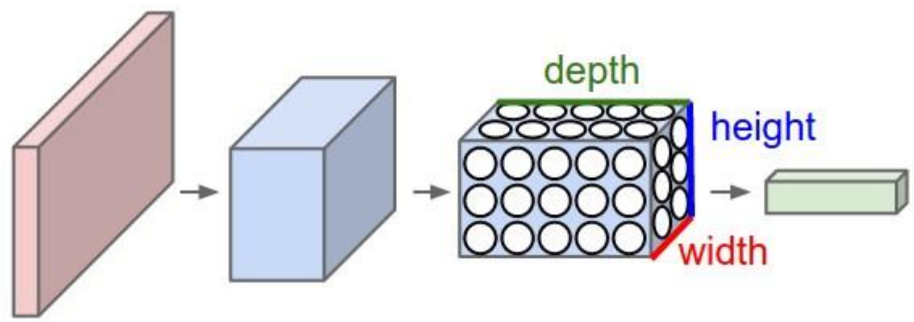
Sharing neurons - convolutions

Note: material from http://cs231n.github.io/convolutional-networks/

In a conv net we use a different connection pattern between layers:

- Typically we use an all-to-all scheme
- Ina conv-net we use local connectivity!





Probabilistic interpretation of Weight Decay

- The gradient step:

$$\Theta := \Theta - \alpha(\nabla_\Theta(J) - \beta\Theta)$$

Corresponds to minimizing:

$$J(\Theta) + \frac{\beta}{2}\sum_j (\Theta_j)^2$$

Now try to find a probabilistic interpretation!

Bayesian approach

1. Make some models more probable than others
2. Set a **prior** probability distribution over Θ
3. For example:
   - weights are normally distributed $p(\Theta_i) \sim \mathcal{N}(0, \sigma_\Theta)$
4. Previously we have assumed: $P(Y|X;\Theta)$ i.e. y depends on x, with a fixed, but unknown Θ
5. Now we will treat Θ as a random variable too $P(Y|X,\Theta)$ i.e. y depends on x and Θ which is randomly sampled too

Bayes theorem

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}$$

Interpretation: how our estimate of A changes after seeing B.

Why?

$$p(A, B) = p(A|B)p(B) = p(B|A)p(A)$$

Then divide by $p(B)$

Bayesian approach to ML

- What is the model probability after seeing the examples in set S?

$$p(\Theta|S) = \frac{p(S|\Theta)p(\Theta)}{p(S)}$$

How to make predictions? Integrate over all models:

$$p(y|x,S) = \int_{\Theta} p(y|x,\Theta)p(\Theta|S)d\Theta$$

Then

$$E[y|x,S] = \int_{y} yp(y|x,S)dy$$

But computing $p(y|x,S)$ is often intractable :(

Maximum-a-posteriori

- Instead of predicting integrating over all Θ
- Use the maximally probable Θ:

$$\Theta_{MAP} = \arg\max_{\Theta} p(\Theta|S)$$
$$= \arg\max_{\Theta} \left(\prod_{i=1}^{m} p\left(y^{(i)} \middle| x^{(i)}, \Theta\right) \right) p(\Theta)$$

- It’s like Max. Likelihood with the extra term.

Gaussian model MAP

$$\arg\max_{\Theta} \prod_{i=1}^{m} p\left(y^{(i)} \middle| x^{(i)}, \Theta\right) p(\Theta) =$$

$$\arg\max_{\Theta} \sum_{i=1}^{m} \log p\left(y^{(i)} \middle| x^{(i)}, \Theta\right) + \log\left(p(\Theta)\right)$$

Now if Θ_j are Gaussian with zero-mean:

$$p\left(\Theta_{\mathrm{j}}\right) = \frac{1}{\sigma_{\Theta}\sqrt{2\pi}} e^{-\frac{\Theta_j^2}{2\sigma_{\Theta}^2}}$$

Then we recover the weight decay term:

$$-\log p\left(\Theta_j\right) \propto \Theta_j^2$$

Special case of Gaussian errors

Assume Gaussian noise on the outputs: $p(y|x,\Theta) = \mathcal{N}(\mathrm{f}(\mathrm{x},\Theta),\sigma_y)$. Then:

$$-\log p(\Theta|S) = -\log p(\mathrm{S}|\Theta) - \log p(\Theta) + \log p(S)$$

$$J^*(\Theta) = \frac{1}{2\sigma_y^2}\sum\left(y^{(i)} - f\left(x^{(i)},\Theta\right)\right)^2 + \frac{1}{2\sigma_\Theta^2}\sum_j \Theta_j^2$$

The weight decay constant is exactly $\sigma_y^2/\sigma_\Theta^2$:

$$J(\Theta) = \sum\left(y^{(i)} - f\left(x^{(i)},\Theta\right)\right)^2 + \frac{\sigma_y^2}{\sigma_\Theta^2}\sum_j \Theta_j^2$$

Quick & dirty weight decay constant estimation method

Hinton describes the following method due to MacKay (http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec9.pdf):

1. Start with an initial guess for weight decay constant
2. Train the net for a while
3. Estimate σ_y looking at the distribution of residuals
4. Estimate σ_Θ looking at the distribution of weights
5. Set the weight decay constant to $\sigma_y^2/\sigma_\Theta^2$
6. Go to step 2.

Weight decay interpretation

- Weight decay corresponds to adding the weights' sum of squares to the optimization function.
- It can be interpreted as MAP criterion with a prior assumption of a Gaussian weight distribution!
- Other penalties:
  - Sum of absolute values (norm 1) (Lasso penalty), makes weights sparse (many are exactly 0)
  - Mixture of norm 1 and norm 2 (elastic net penalty)

L2 vs L1 weight regularization intuitions

We can apply two kinds of penalty terms to weights:

- L2 (sum of squares) makes all weights small
- L1 (sum of absolute values) makes weights sparse, i.e. some weights exactly zero, and other larger