CSC321 Lecture 13/14
Improving generalization

Roger Grosse and Nitish Srivastava

February 26, 2015

Overview

We know how to train big models to fit the training data.
Just build a big neural net and train it with backprop!
But this will just overfit the training data. What we really want is to get good generalization.

Overview

We know how to train big models to fit the training data.
Just build a big neural net and train it with backprop!
But this will just overfit the training data. What we really want is to get good generalization.
Ways to get that -

- Collect more data.

Overview

We know how to train big models to fit the training data.
Just build a big neural net and train it with backprop!
But this will just overfit the training data. What we really want is to get good generalization.
Ways to get that -

- Collect more data.
- Choose a model with the right capacity.
  - Fewer parameters: fewer hidden units and layers.
  - Weight penalties and constraints.
  - Early stopping.

Overview

We know how to train big models to fit the training data.
Just build a big neural net and train it with backprop!
But this will just overfit the training data. What we really want is to get good generalization.
Ways to get that -

- Collect more data.
- Choose a model with the right capacity.
  - Fewer parameters: fewer hidden units and layers.
  - Weight penalties and constraints.
  - Early stopping.
- Ensemble methods - bagging, boosting, mixture of experts.

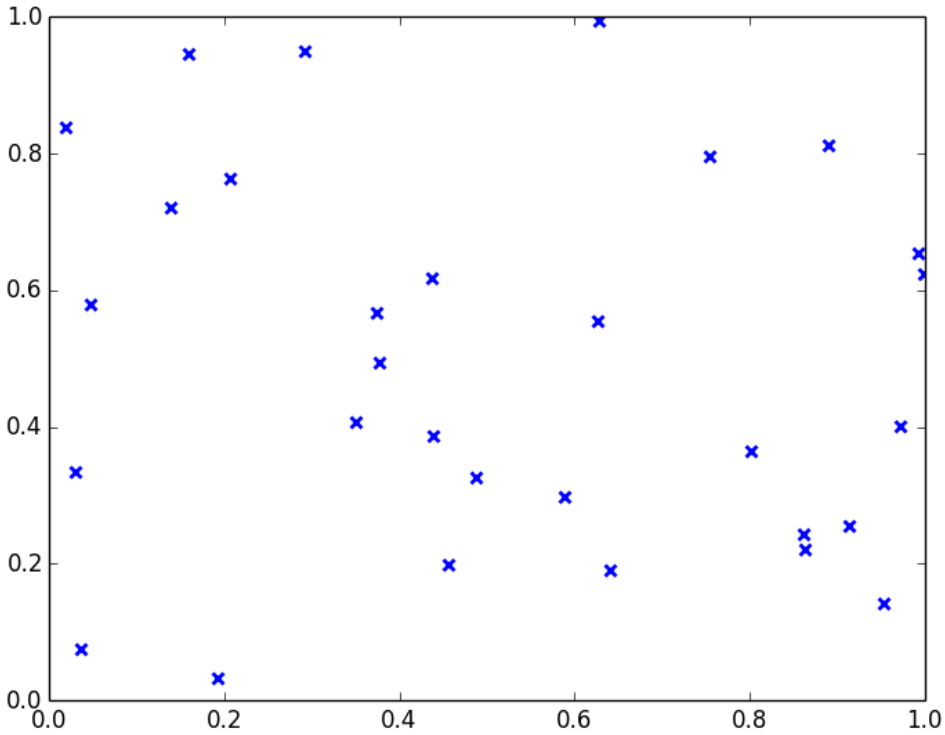
Overview

We know how to train big models to fit the training data.
Just build a big neural net and train it with backprop!
But this will just overfit the training data. What we really want is to get good generalization.
Ways to get that -

- Collect more data.
- Choose a model with the right capacity.
  - Fewer parameters: fewer hidden units and layers.
  - Weight penalties and constraints.
  - Early stopping.
- Ensemble methods - bagging, boosting, mixture of experts.
- Add Noise - Data augmentation, Stochastic neurons, Dropout.

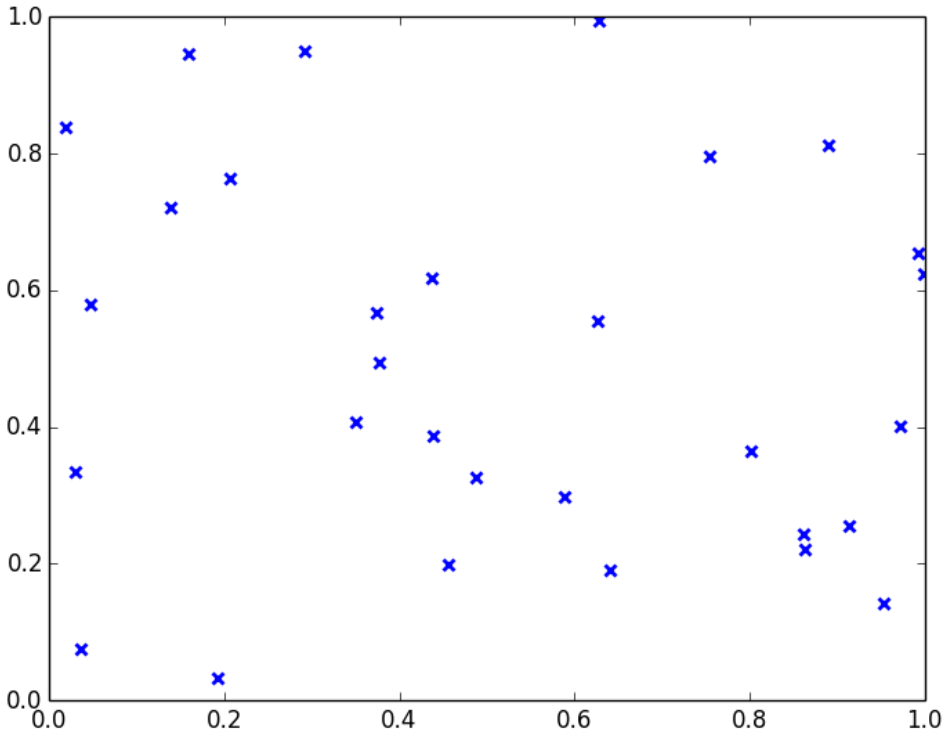
Overview

What patterns do you see in the following data?



Overview

What patterns do you see in the following data?



These points were sampled uniformly at random!

Regularities : Real and Accidental

Consider any finite dataset $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$.

- This set of samples was drawn from some distribution (the "real world").

Regularities : Real and Accidental

Consider any finite dataset $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$.

- This set of samples was drawn from some distribution (the "real world").
- There are some "true" underlying regularities in the input-output mapping that we want to discover.
- But there will also be accidental regularities due to sampling.

Regularities : Real and Accidental

Consider any finite dataset $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$.

- This set of samples was drawn from some distribution (the "real world").
- There are some "true" underlying regularities in the input-output mapping that we want to discover.
- But there will also be accidental regularities due to sampling.
- For example, we want to learn a classifier that separates images of trucks from cars.
  - But all truck pictures are taken on bright and sunny days, and all cars were taken in low-light conditions.

Regularities : Real and Accidental

Consider any finite dataset $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$.

- This set of samples was drawn from some distribution (the "real world").
- There are some "true" underlying regularities in the input-output mapping that we want to discover.
- But there will also be accidental regularities due to sampling.
- For example, we want to learn a classifier that separates images of trucks from cars.
  - But all truck pictures are taken on bright and sunny days, and all cars were taken in low-light conditions.
  - Or, in all truck pictures, pixel (20, 34) has a green value greater than 102, but in all car pictures it is less than 102.

Regularities : Real and Accidental

Consider any finite dataset $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$.

- This set of samples was drawn from some distribution (the "real world").
- There are some "true" underlying regularities in the input-output mapping that we want to discover.
- But there will also be accidental regularities due to sampling.
- For example, we want to learn a classifier that separates images of trucks from cars.
  - But all truck pictures are taken on bright and sunny days, and all cars were taken in low-light conditions.
  - Or, in all truck pictures, pixel (20, 34) has a green value greater than 102, but in all car pictures it is less than 102.
  - Or, some other complicated "weird and unnatural" hidden feature (or group of features) separates the two classes without really understanding the true regularities.

Regularities : Real and Accidental

Consider any finite dataset $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$.

- This set of samples was drawn from some distribution (the "real world").
- There are some "true" underlying regularities in the input-output mapping that we want to discover.
- But there will also be accidental regularities due to sampling.
- For example, we want to learn a classifier that separates images of trucks from cars.
  - But all truck pictures are taken on bright and sunny days, and all cars were taken in low-light conditions.
  - Or, in all truck pictures, pixel (20, 34) has a green value greater than 102, but in all car pictures it is less than 102.
  - Or, some other complicated "weird and unnatural" hidden feature (or group of features) separates the two classes without really understanding the true regularities.

The accidental regularities may not happen at test time.

Regularization

Models with high capacity/complexity will be able to learn weird features. How to prevent this?

Regularization

Models with high capacity/complexity will be able to learn weird features. How to prevent this?

> Idea: come up with a function R which is large when the features are weird and unnatural. Then try to keep R small.

Regularization

Models with high capacity/complexity will be able to learn weird features. How to prevent this?

> Idea: come up with a function R which is large when the features are weird and unnatural. Then try to keep R small.

New optimization problem:

$$C_{\text{reg}}(\mathbf{w}) = L(\mathbf{w}) + R(\mathbf{w}),$$

where $L(\mathbf{w})$ is the loss function (e.g. cross-entropy) and $R(\mathbf{w})$ is a regularizer.

Regularization

Models with high capacity/complexity will be able to learn weird features. How to prevent this?

> Idea: come up with a function R which is large when the features are weird and unnatural. Then try to keep R small.

New optimization problem:

$$C_{\text{reg}}(\mathbf{w}) = L(\mathbf{w}) + R(\mathbf{w}),$$

where $L(\mathbf{w})$ is the loss function (e.g. cross-entropy) and $R(\mathbf{w})$ is a regularizer.

Intuition: small R should correspond to "simpler" models. (But this is hard to make precise.)

Weight penalties

How can we measure a model's capacity / complexity ?
Equivalently, how do we pick what $R(\mathbf{w})$ should be ?

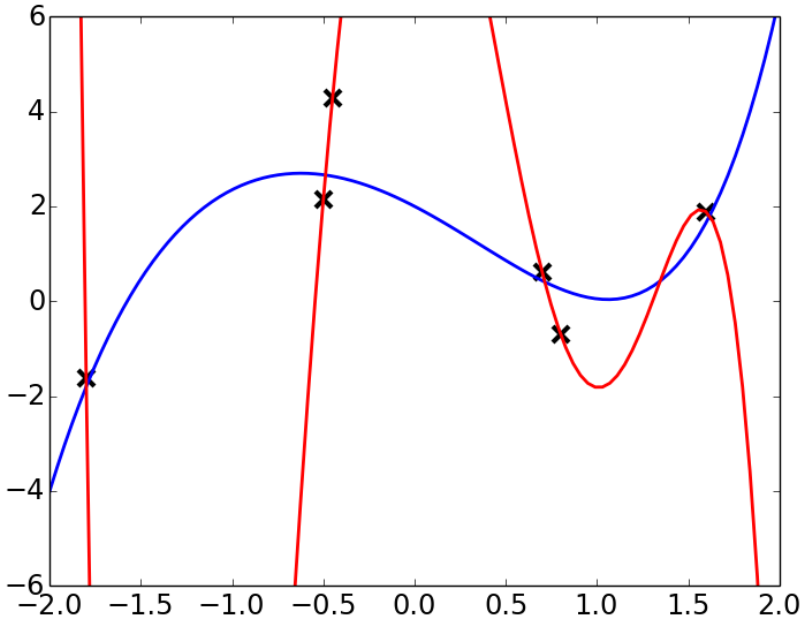
Weight penalties

How can we measure a model's capacity / complexity ?
Equivalently, how do we pick what $R(\mathbf{w})$ should be ?
In general, this depends on the model.
But there are some basic requirements, for example smoothness.
For any model,

$$y = f_{\mathbf{w}}(\mathbf{x})$$

if y varies a lot over small changes in $\mathbf{x}$, then it is more complicated.
So, if we make $\mathbf{w}$ small, then (usually) y won't vary a lot for small changes in $\mathbf{x}$.

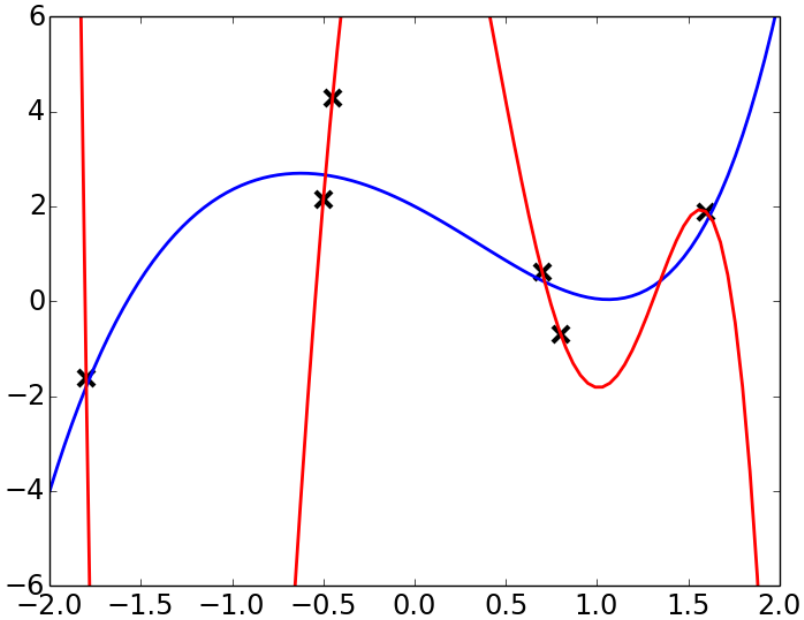
Weight penalties



$$y = 0.1x^5 + 0.2x^4 + 0.75x^3 - x^2 - 2x + 2$$

$$y = -7.2x^5 + 10.4x^4 + 24.5x^3 - 37.9x^2 - 3.6x + 12$$

Weight penalties



$$y = 0.1x^5 + 0.2x^4 + 0.75x^3 - x^2 - 2x + 2$$

$$y = -7.2x^5 + 10.4x^4 + 24.5x^3 - 37.9x^2 - 3.6x + 12$$

The red one overfits. Notice it has really large coefficients.

Weight penalties

The key idea

A network with small weights is weaker/simpler/smoother than a network with big weights. If a network has small weights then it can only do simple things.

Since the model can only do simple things, it cannot learn weird and unnatural features.

Weight penalties

The key idea

A network with small weights is weaker/simpler/smoother than a network with big weights. If a network has small weights then it can only do simple things.

Since the model can only do simple things, it cannot learn weird and unnatural features.
Note that small/big refers only to the magnitude of the weight. The sign can be positive or negative.

Weight penalties

The key idea

A network with small weights is weaker/simpler/smoother than a network with big weights. If a network has small weights then it can only do simple things.

Since the model can only do simple things, it cannot learn weird and unnatural features.
Note that small/big refers only to the magnitude of the weight. The sign can be positive or negative.
Also note that we assumed that the real regularities are simpler than the accidental ones.

Weight penalties

L_2 penalty (a.k.a L_2 decay)

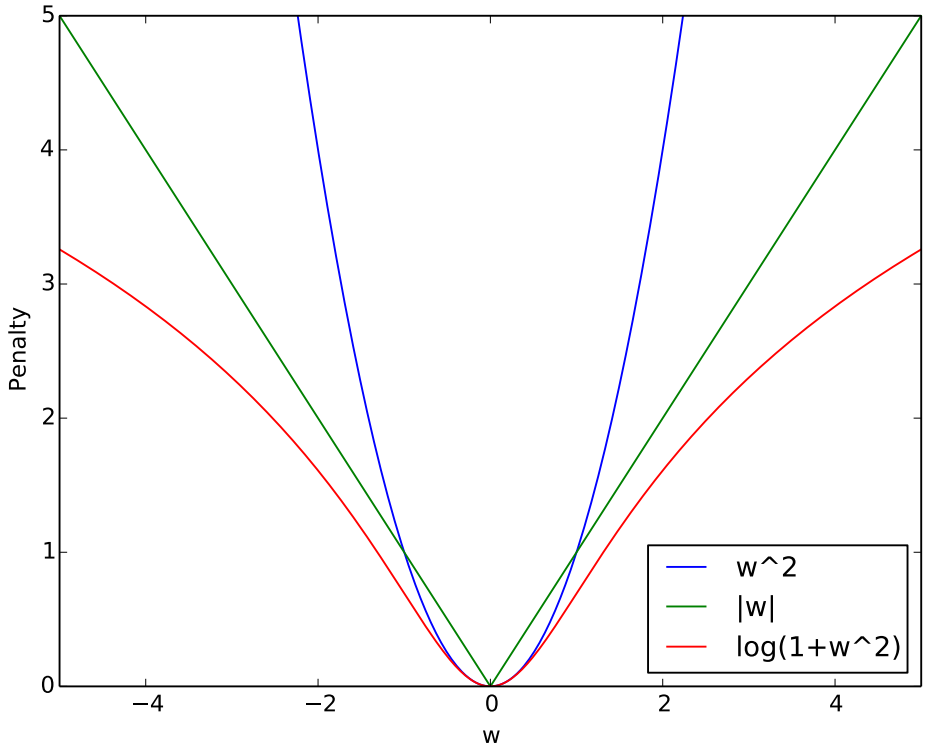
$$R(\mathbf{w}) \propto \sum_i w_i^2$$

L_1 penalty (a.k.a lasso)

$$R(\mathbf{w}) \propto \sum_i |w_i|$$

Other forms of penalty:

$$R(\mathbf{w}) \propto \sum_i \log(1 + w_i^2)$$



All these regularizers would be happiest if w_i's were all zero.
They differ in how unhappy they become as w_i's increase.

Weight constraints

Instead of just discouraging big weights, we can completely rule them out. I.e., we can add a hard constraint that $||\mathbf{w}|| \leq L$.

- This keeps the weights inside a ball of radius L.

We can also view this as a regularization term:

$$R(\mathbf{w}) = \begin{cases} 0 & ||\mathbf{w}|| \leq L \\ +\infty & \text{otherwise} \end{cases}$$

Question 1

Suppose we are fitting a linear regression model

$$y = wx$$

We just have one data point (x, t) , $x \neq 0$. Consider these three loss functions

$$L_1(w) = \frac{1}{2}(y - t)^2$$
$$L_2(w) = \frac{1}{2}(y - t)^2 + \frac{\lambda}{2}w^2$$
$$L_3(w) = \frac{1}{2}(y - t)^2 + \lambda|w|$$

Minimize each loss function to solve for the optimal w in terms of x, t, λ.

Note: for L_3, you need to check 3 cases, corresponding to $w < 0, w = 0, w > 0$

Ensemble methods

The main intuition

Suppose you ask 100 people Barack Obama's height. There will be lots of variability in people's responses, but that variability washes out when we take the average. The group will be more accurate than most of the individuals.

On the other hand, if there's systematic bias in people's answers, that won't go away when we average.

Ensemble methods

The main intuition

Suppose you ask 100 people Barack Obama's height. There will be lots of variability in people's responses, but that variability washes out when we take the average. The group will be more accurate than most of the individuals.

On the other hand, if there's systematic bias in people's answers, that won't go away when we average.

Ensemble methods

The main intuition

Suppose you ask 100 people Barack Obama's height. There will be lots of variability in people's responses, but that variability washes out when we take the average. The group will be more accurate than most of the individuals.

On the other hand, if there's systematic bias in people's answers, that won't go away when we average.

- This suggests training lots of models and averaging their predictions.

Ensemble methods

The main intuition

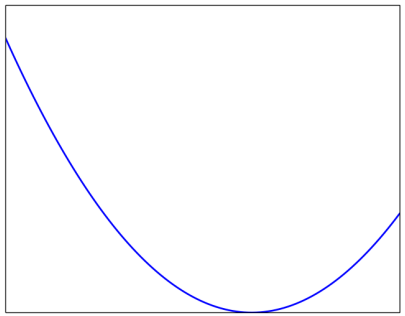
Suppose you ask 100 people Barack Obama's height. There will be lots of variability in people's responses, but that variability washes out when we take the average. The group will be more accurate than most of the individuals.

On the other hand, if there's systematic bias in people's answers, that won't go away when we average.

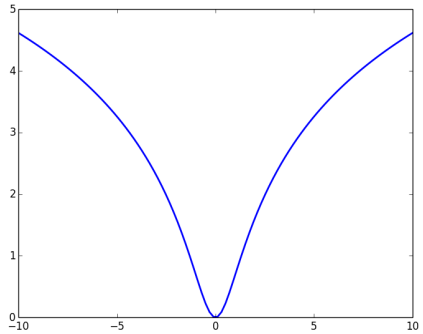
- This suggests training lots of models and averaging their predictions.
- Two general strategies:
  - Train independently and then weight equally.
    - This reduces variability.
    - Examples: bagging, random forests
  - Train jointly and learn the weights.
    - This can result in a more powerful model.
    - Examples: boosting, mixtures of experts

Question 2: When does averaging help?

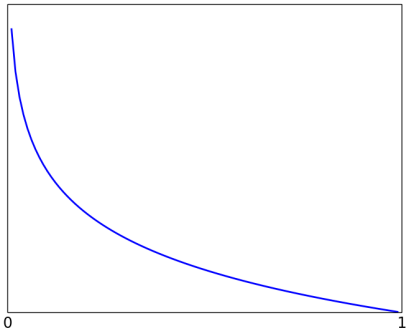
For some of these cost functions, averaging is *guaranteed* to help. Which ones?



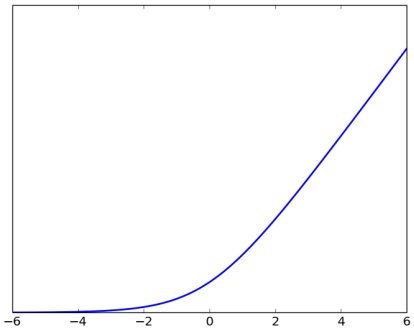
Squared error



$\log\left(1 + (y - t)^2\right)$



Cross-entropy (where we average the predicted probability)



Cross-entropy (where we average the input z to the logistic)

Model Diversity

We want the models to be different so that they make different errors. If all models make the same error, then the average would too.

How can we make 10 different models ?

Model Diversity

We want the models to be different so that they make different errors. If all models make the same error, then the average would too.

How can we make 10 different models ?

- Train different kinds of models - neural nets, decision trees.

Model Diversity

We want the models to be different so that they make different errors. If all models make the same error, then the average would too.

How can we make 10 different models ?

- Train different kinds of models - neural nets, decision trees.
- Train different architectures, different layer sizes, different hyperparameters, different random seeds.

Model Diversity

We want the models to be different so that they make different errors. If all models make the same error, then the average would too.

How can we make 10 different models ?

- Train different kinds of models - neural nets, decision trees.
- Train different architectures, different layer sizes, different hyperparameters, different random seeds.
- Train on different subsets of the data : Bagging

Noise as a regularizer

Data Augmentation

- Having more data is always better.
- If we don't have more data, lets create some using what we have!

Noise as a regularizer

Data Augmentation

- Having more data is always better.
- If we don't have more data, lets create some using what we have!
- For images :
  - Translate: Train on random crops of the image.
  - Flip horizontally.
  - Add noise along eigen vectors of the color space.
  - Add Distortions : Stretch or squash the image.

Noise as a regularizer

Data Augmentation

- Having more data is always better.
- If we don't have more data, lets create some using what we have!
- For images :
  - Translate: Train on random crops of the image.
  - Flip horizontally.
  - Add noise along eigen vectors of the color space.
  - Add Distortions : Stretch or squash the image.
- For speech : VTLP (Vocal Tract Length Perturbation).

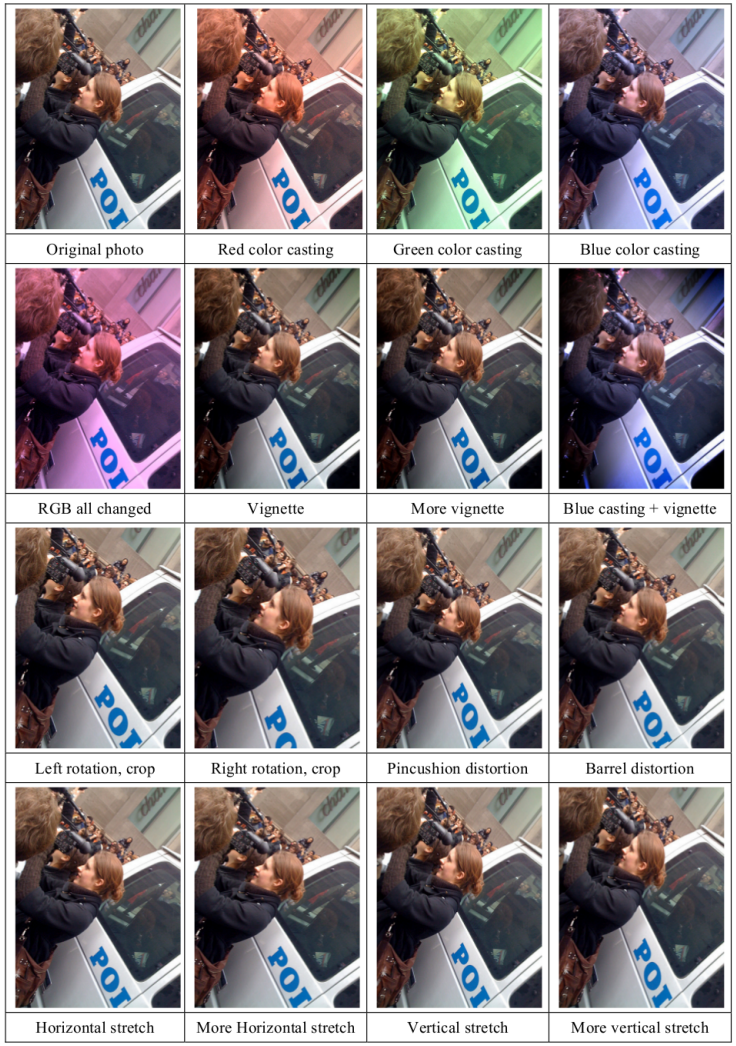
Noise as a regularizer

Data Augmentation

- Having more data is always better.
- If we don't have more data, lets create some using what we have!
- For images :
  - Translate: Train on random crops of the image.
  - Flip horizontally.
  - Add noise along eigen vectors of the color space.
  - Add Distortions : Stretch or squash the image.
- For speech : VTLP (Vocal Tract Length Perturbation).
- Just add plain Gaussian Noise or salt-and-pepper noise.

This data will not be as good as new data, but it's better than nothing.

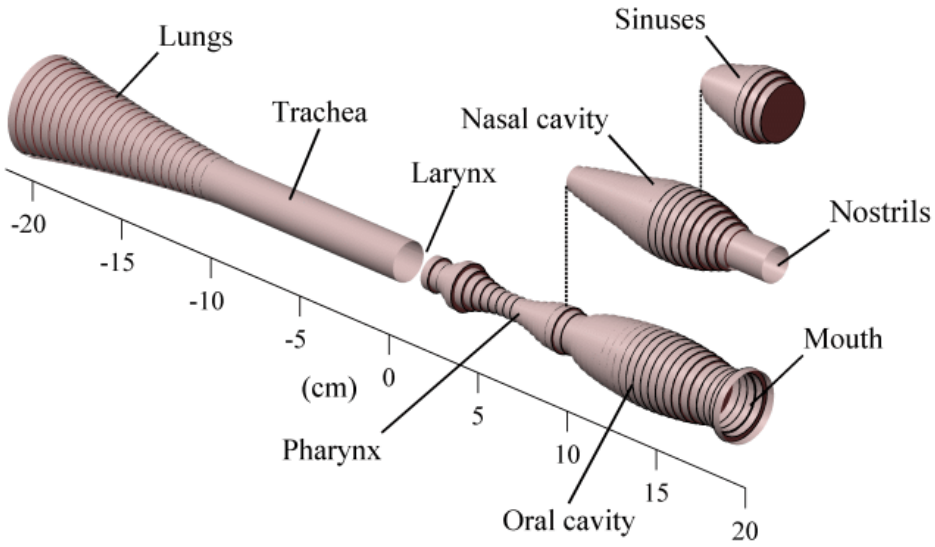
Image Distortions



Deep Image: Scaling up Image Recognition: Wu et al. arXiv 2015 (Baidu)

Speech Distortions

Speech signal $\rightarrow$ vocal track parameters $\overset{\text{noise}}{\rightarrow}$ Perturbed vocal track parameters $\rightarrow$ Distorted speech



Noise as a regularizer

Why just add noise to the input layer ?
Lets add noise to the hidden layers too!

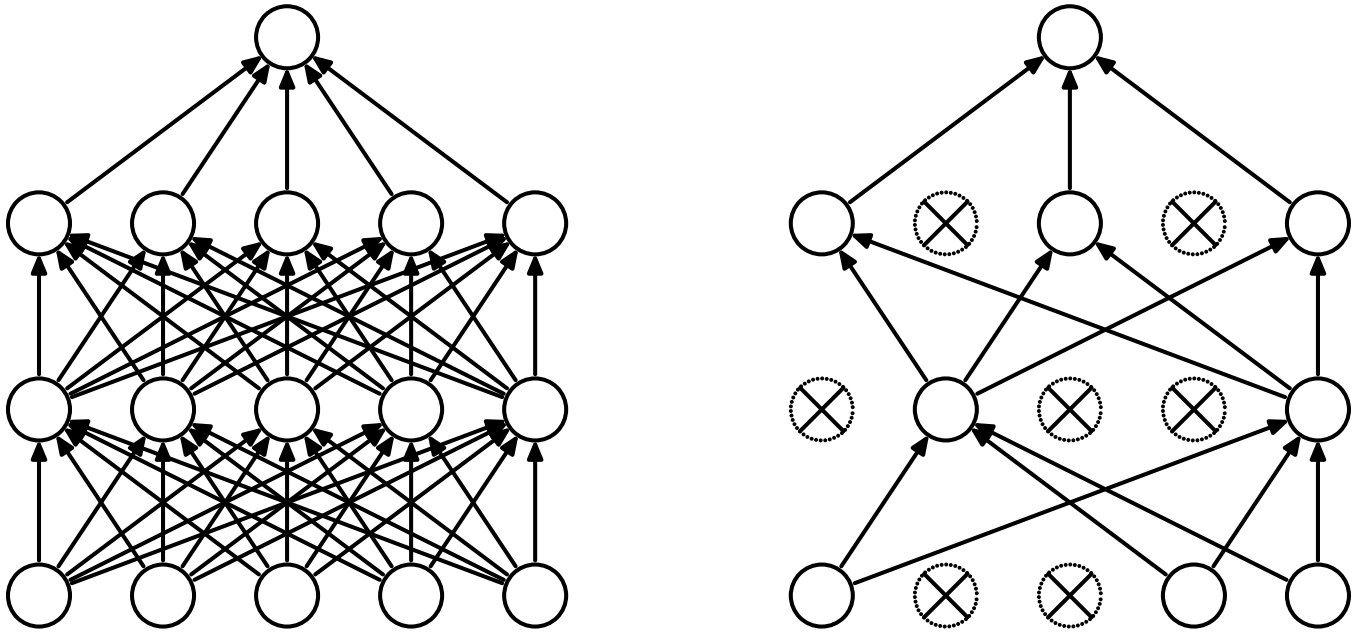
Noise as a regularizer

Why just add noise to the input layer ?
Lets add noise to the hidden layers too!
Stochastic Neurons

- Logistic hidden units have an activation p between 0 and 1.
- Set the activation to 1 with probabilty p, 0 otherwise.
- The same expected activation, but now we have some stochasticity.

Noise as a regularizer

Just randomly drop units : Set the activation to zero with some arbitrarily chosen probabilty p, say 0.5.



A different sampling for each training case in each mini-batch.

Noise as a regularizer

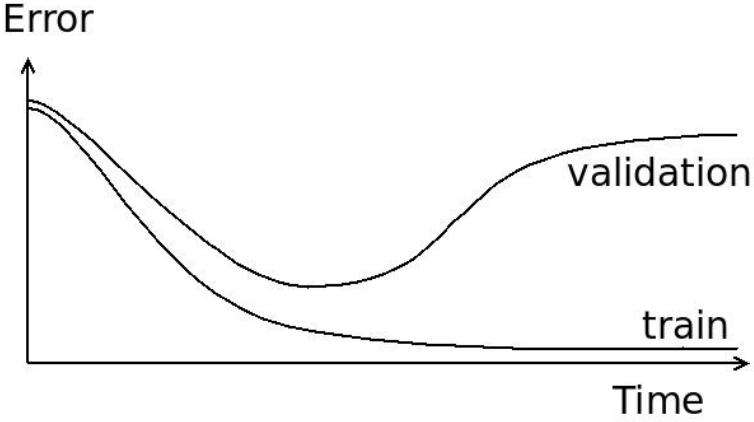
Neural Nets : A co-adaptation conspiracy.

Dropout : Preventing co-adaptation.

"Ten conspiracies each involving five people is probably a better way to create havoc than one big conspiracy that requires fifty people to all play their parts correctly. If conditions do not change and there is plenty of time for rehearsal, a big conspiracy can work well, but with non-stationary conditions, the smaller the conspiracy the greater its chance of still working. Complex co-adaptations can be trained to work well on a training set, but on novel test data they are far more likely to fail than multiple simpler co-adaptations that achieve the same thing". *-Geoff Hinton*

Early stopping

Hold out a part of the training set. This is called a validation set.
Stop training when performance on the validation set starts getting worse.



Picture credits: Ilya Sutskever `https://theneural.wordpress.com/`

Use this to

- Prevent overfitting.
- Make decisions about network architectures and tuning parameters : Pick the model that gets the lowest validation error.

Cross Validation

A problem with having a validation set-

- We never get to train on it.
- If we have only a small amount of precious labelled data, it is not ok to just leave out part of it.

Cross Validation

A problem with having a validation set-

- We never get to train on it.
- If we have only a small amount of precious labelled data, it is not ok to just leave out part of it.

Cross Validation

- Split the dataset into N chunks.
- Train N different models.
- For each model pick a different chunk for validation, use the other $N - 1$ chunks for training.
- So, we get to train on everything, but not in the same model.
- Use the average validation performance to make decisions about hyperparameters.