Bias, variance and regularization

Jan Chorowski
Instytut Informatyki
Wydział Matematyki i Informatyki Uniwersytet Wrocławski

2020

Where are we?

Goal: do well on unsen data

do well
- low error rate
- proper training
- essentially lect 4

on unsen data
- low err rate on test data
- proper regularization
- the difference between ML and OPT.
- goal for today

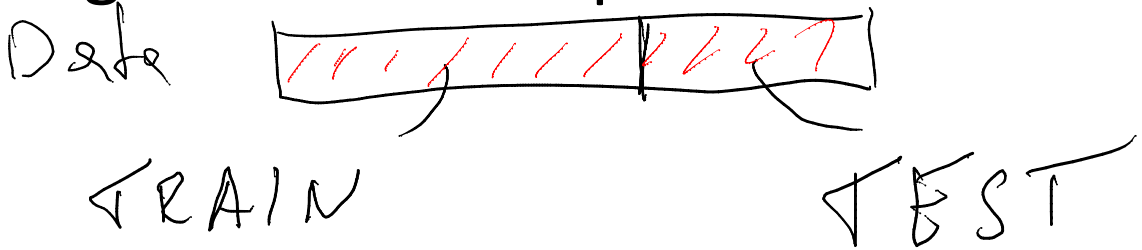
Generalization, the goal of learning

- Problem:
  - We care about the performance on **all the data**
  - We have only a **training sample**
  - **Under-fitting:**
    Doesn’t work even on **training samples**
    Too weak classifier (cannot express the relation in the data), bad training …
  - **Over-fitting**:
    Too powerful classifier will perfectly interpolate the training data (even the noise in it!) and do poorly on **unseen 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

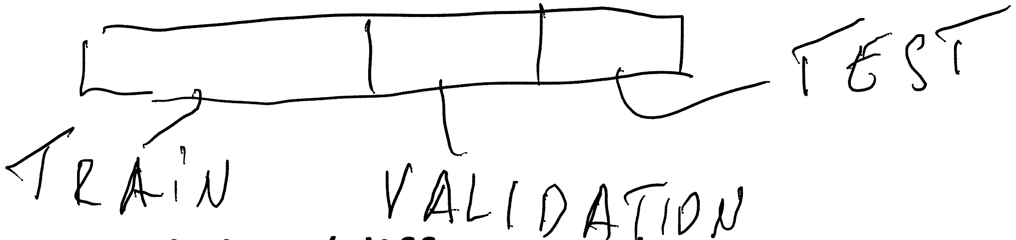
Honest estimates: Hold-out set

Large data case!!!

- Split the training data into two parts:



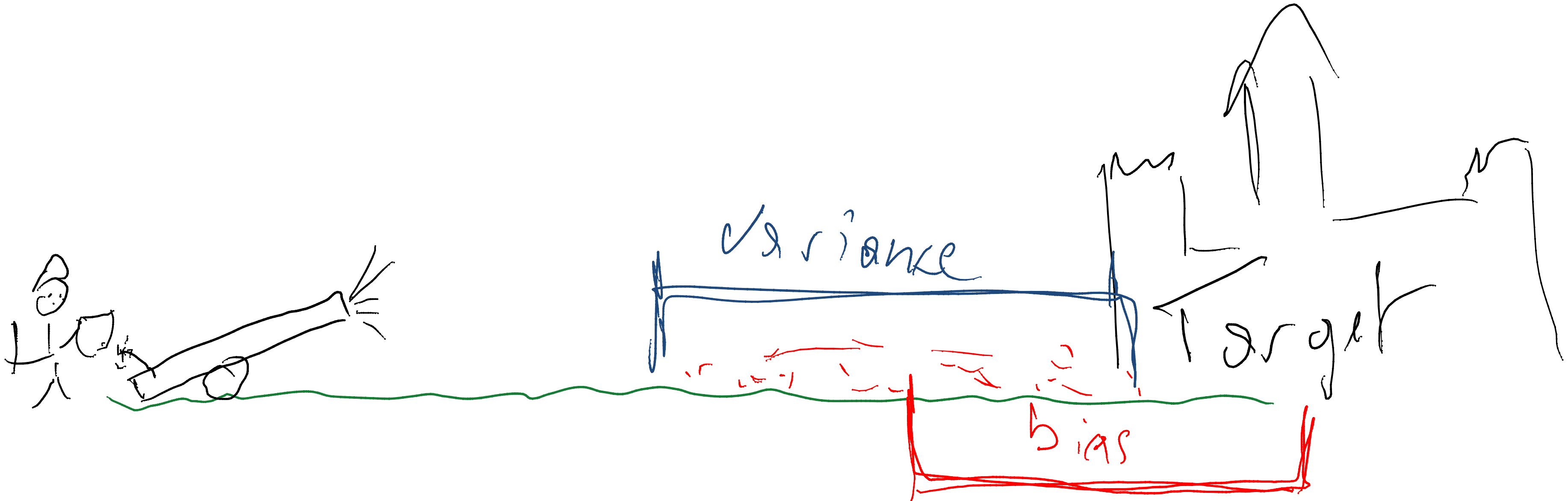
- Train only on training, then test on testing.
- Often we do a three-way split:



- Then:
  - Train many models on training (different algos, parameters)
  - Use validation to choose best model
  - Test on testing
- Other techniques: Cross-Validation and bootstrap

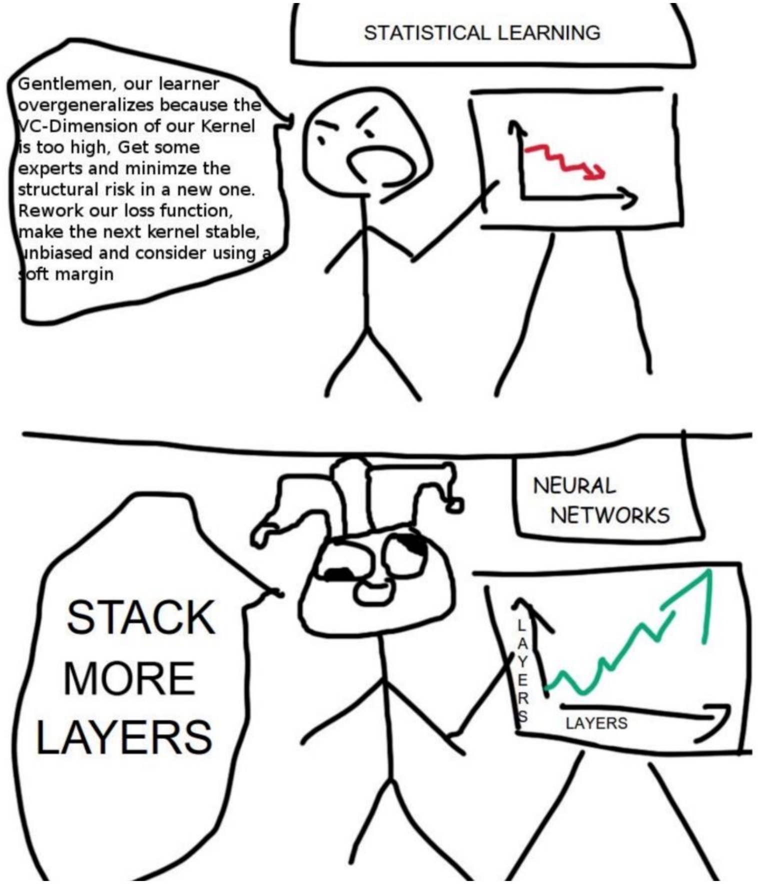
Bias-Variance: two sources of error!

- The **bias** captures how well our family of functions (hypothesis space) matches the data.
  Intuition: systematic error
  Large neural nets have a low bias (universal approximation!)
- The **variance** captures how results of different training runs vary
  Deep Learning is unstable: random init, SGD…



How to lower the bias?

- Choose more powerful/better models:
  - **More hidden neurons**
  - Use better network architecture
  - Understand the data and choose a matching model
  - Describe the data with more attributes
  - 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
- **Constrain the models:**
  – Simpler models
  – Shorter training (early stopping)
  – Regularize the models
  – Select only the most important inputs
- **Average the models (very powerful)**
  – Also called “ensemble learning”, boosting, bagging
  – Requires that the models make uncorrelated errors
  – Some neural net techniques have an ensembling interpretation:
    - Dropout: ensemble of many nets that share weights, but use different neurons
    - ResNets: sum of many shallow nets formed by subsets of layers

Classical Regularization

- Limit the size of parameters
- Ridge regression: $\min_{\Theta} \sum_i \left(\Theta^T x^{(i)} - y^{(i)}\right) + \frac{\lambda}{2} \sum_j \Theta_j^2$
- Interpretation:
  – Unique solution

  – Small $\frac{\partial \Theta^T x}{\partial x}$

  – Use all features to decide (L2 reg), select features (L1 reg)

Weight Decay == Ridge Regression in DL

The intuitions:

- Start with many weights (larger nets train easier!)
- Don't depend on a single net. How?
- Force all the weights to decrease

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

Probabilistic interpretation of Weight Decay

- The gradient step:

$$\Theta \coloneqq \Theta - \alpha(\nabla_{\Theta}(J) - \beta\Theta)$$

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. 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 Data $\mathcal{D}$?

$$p(\Theta|\mathcal{D}) = \frac{p(\mathcal{D}|\Theta)p(\Theta)}{p(\mathcal{D})}$$

How to make predictions? Integrate over all models:

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

Then

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

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

Maximum-a-posteriori

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

$$\Theta_{MAP} = \arg\max_{\Theta} p(\Theta|\mathcal{D})$$

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

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

Gaussian model MAP

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

$$\arg\max_{\Theta} \sum_{(x^{(i)},y^{(i)})\in\mathcal{D}} \log p\left(y^{(i)} \middle| x^{(i)}, \Theta\right) + \log\left(p(\Theta)\right)$$

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:

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

Weight decay and momentum training

Recall that in weight decay:

$$TotLoss = Loss + \lambda \sum_i \Theta_i^2$$

$$\frac{\partial TotLoss}{\partial \Theta} = \frac{\partial Loss}{\partial \Theta} + \frac{\lambda}{2} \sum_i \Theta_i$$

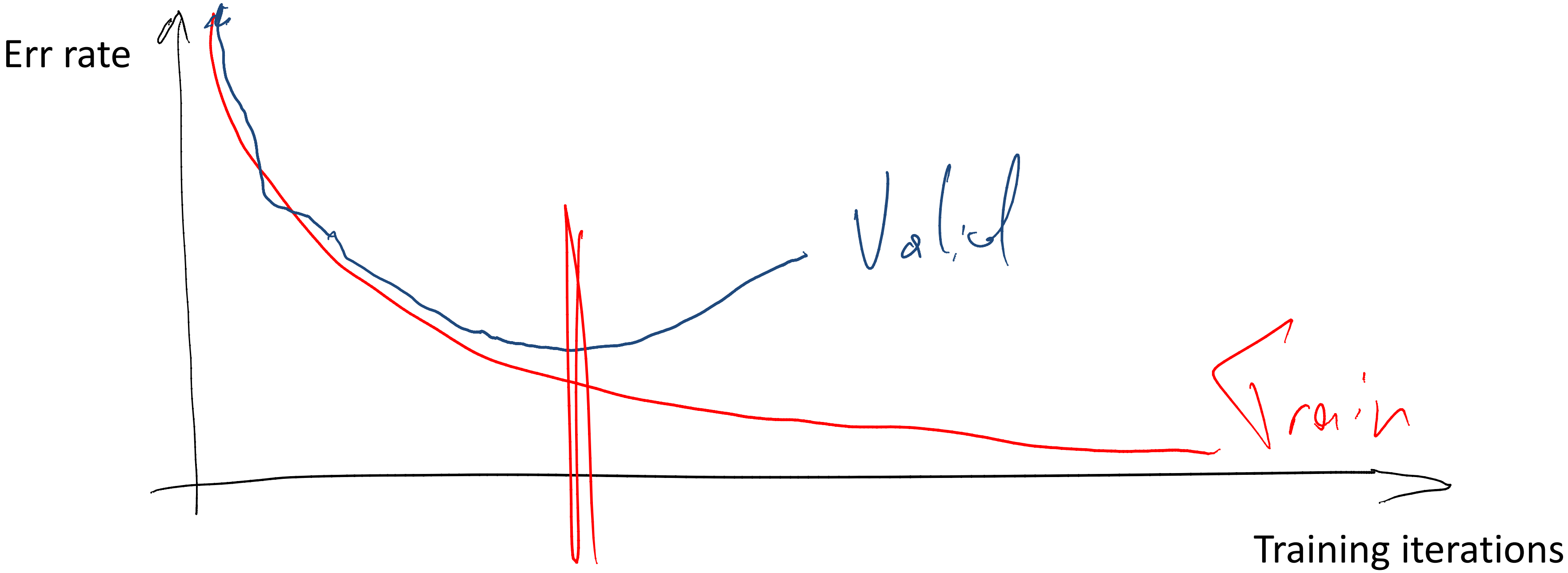
The weight decay term is stable over time -> gets boosted by momentum

Hacky trick (promoted by fast.ai): don't include weight decay in loss and gradient computation.

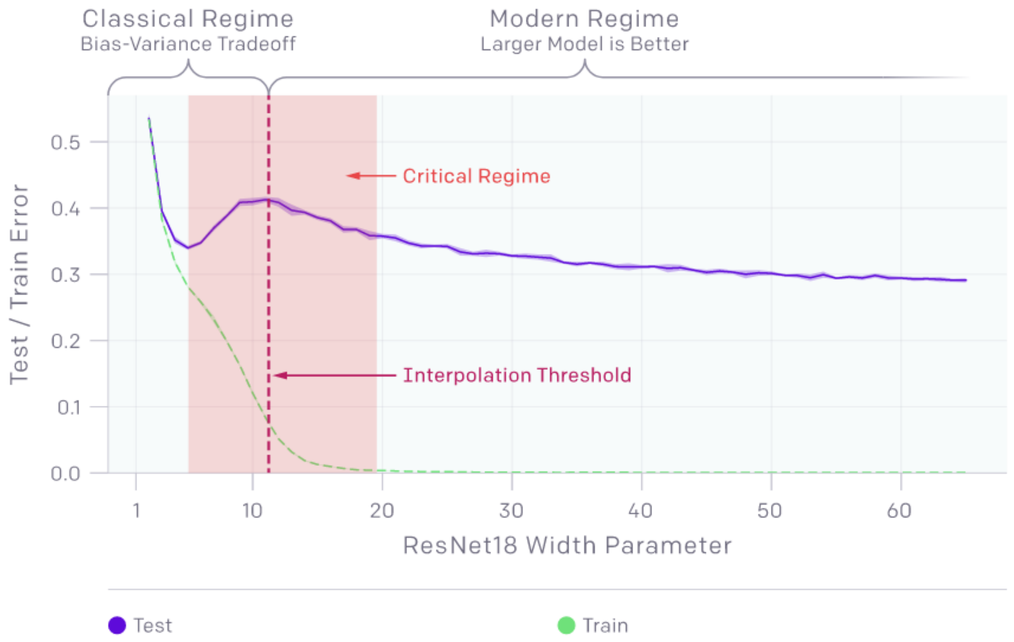
However, apply the weight decay step separately after the main optimizer.

Early stopping

- The net starts with small weights (we initialize it like that)
- As training progresses the weights grow (net specializes)
- At some point, it over-specializes
- Look for that moment, by monitoring a validation error!



With early stoping, large nets don't overfit



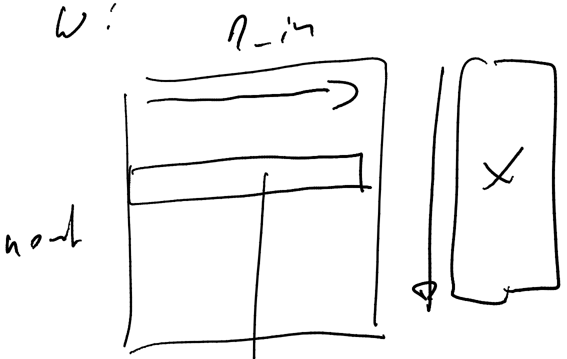
https://openai.com/blog/deep-double-descent/

Weight norm constrains

In Weight Decay we want small weights …and pull all weights to 0!

Similar idea: constrain weights to be small, i.e.

$$\min_{\substack{W: \\ \underbrace{\forall i\ \mathrm{norm}(W_{i:})<\delta}_{\text{constrain all rows.}}}} Wx$$



each row: weights of 1 neuron

How?

1 make an SGD update

2 constrain: W /= max(1, sqrt(sum(W**2, 1)) / δ)

Weight noise

Weights can be large, but have limited precision (thus it doesn't matter if they change a bit).

How? Add noise in SGD:

$$\Theta \leftarrow \frac{\partial Loss(\Theta + \text{noise})}{\partial \Theta}$$

for each minibatch:

$\Theta' = \Theta +$ noise

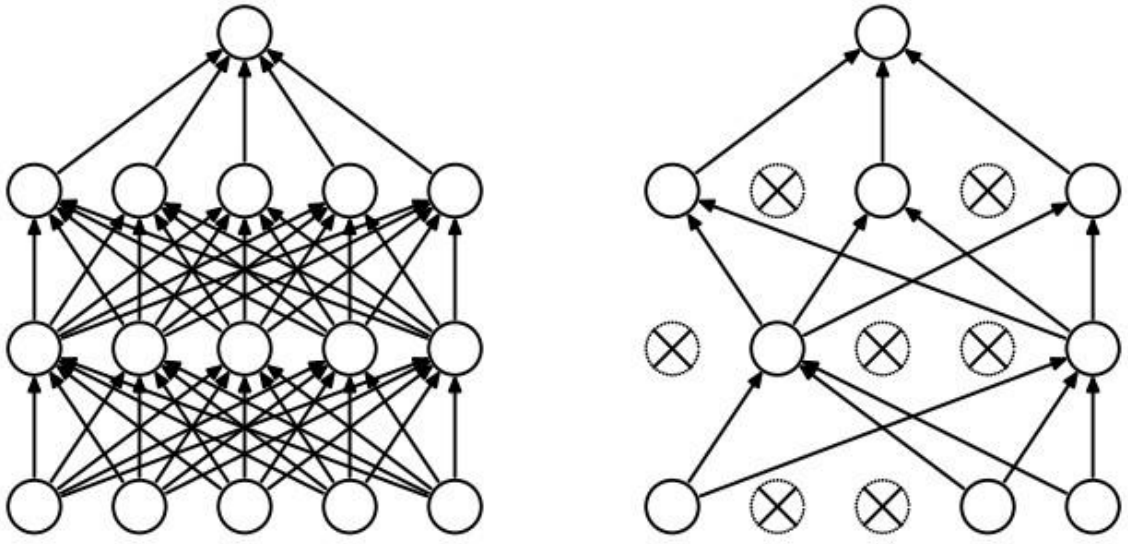
do fprop using Θ'

do bprop

apply gradient.

Dropout

- At train time:
  – For each example, select with probability p which neurons will be used (not dropped out).
  – Multiply the outputs of other neurons by $1/p$ to compensate
- At test time:
  – Use all neurons
- Interpretation:
  – Prevents co-adaptation of neurons,
  it is harder for neurons to cooperate if 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.

Networks are overconfident

- We usually train with cross-entropy loss

$$\mathrm{L} = \sum_{c=1}^{C} -p\left(y^{(i)} = c\right) \log p\left(y = c | x^{(i)}\right)$$

- In the train data the class is certain, loss simplifies to

$$\mathrm{L} = -\log p\left(y = y^{(i)} | x^{(i)}\right)$$

- When model is 99% accurate…
- The only way to reduce loss is to make $p\left(y = y^{(i)} \middle| x^{(i)}\right) = 1$
- This promotes overconfidence p(first guess) >> p(second guess)

Label Smoothing

- Introduced in Inception V2 (arXiv:1512.00567)
- Assume train labels are more ambiguous:

$$\mathrm{L} = \sum_c -p\left(y^{(i)} = c\right) \log p(y = c|x^{(i)})$$

- $p\left(y^{(i)} = c\right)$ is a smoothing distribution, e.g.

$$p\left(y^{(i)} = c\right) = \begin{cases} \beta, & \text{on correct class} \\ \dfrac{1-\beta}{C-1}, & \text{otherwise} \end{cases}$$

- Even better: smooth the $1 - \beta$ according to class marginal probabilities

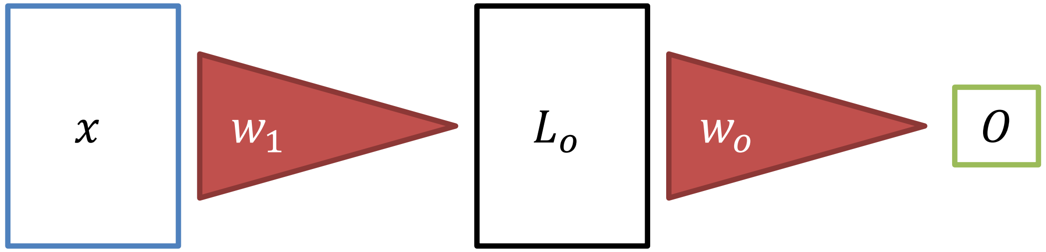
Effects of Label Smoothing

- Reduces overconfidence and regularizes
- Also prevents gradient vanishing:
  - Without smoothing SoftMax derivative is $p_\Theta(Y_i|X_i) - [Y_i = c]$
  - This vanishes when $p_\Theta(Y_i|X_i) \approx 1$
  - Effectively the model stops training on correctly classified instances

Label Smoothing vs Other Regularizers

At a high level, all regularizers want to forbid large changes of output for small changes of input.

- E.g. weight decay



Magnitude of w_o controls the output sensitivity $\frac{\partial O}{\partial L_o} = w_o^T$

- Label smoothing may be easier to use:
  - Easy to say how smooth the output should be
  - Hard to say how large the weights should be

Mixup

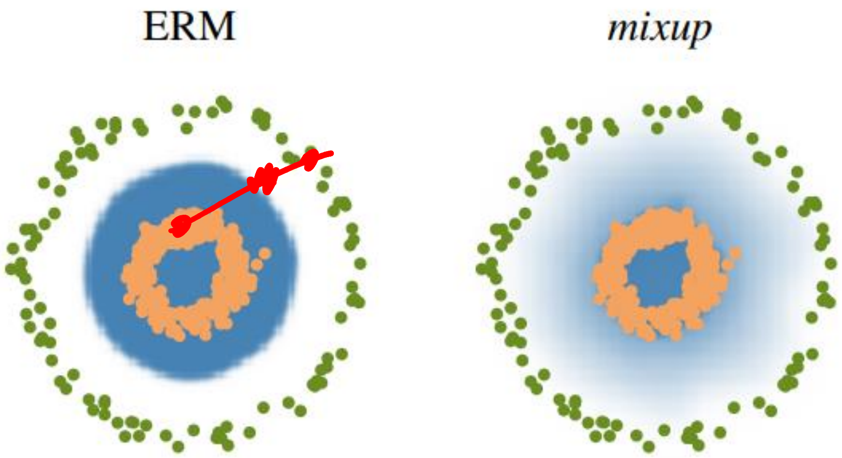
Mixup (https://arxiv.org/pdf/1710.09412.pdf)

- Take two data samples: $\left(x^{(i)}, y^{(i)}\right)$ and $\left(x^{(j)}, y^{(j)}\right)$.
- Make a *new* sample

$$\tilde{x} = \lambda x^{(i)} + (1-\lambda)x^{(j)}$$

$$\tilde{y} = \lambda y^{(i)} + (1-\lambda)y^{(j)}$$

- Intuition: again, this forces the network's output to gradually change



Summary of NNet regularizations

- Choose proper architecture
  add or remove neurons or whole layers
  - Share weights between neurons
    - Example: convolutional networks
- Use weight decay or other weight priors
  L2 (Gaussian), or L1 (Laplace)
- Use Early stopping
  Monitor validation error as training progresses. Stop when it starts to increase
- Use dropout -> randomly remove some neurons
- Use weight noise

Hyperparameters: Art or Science

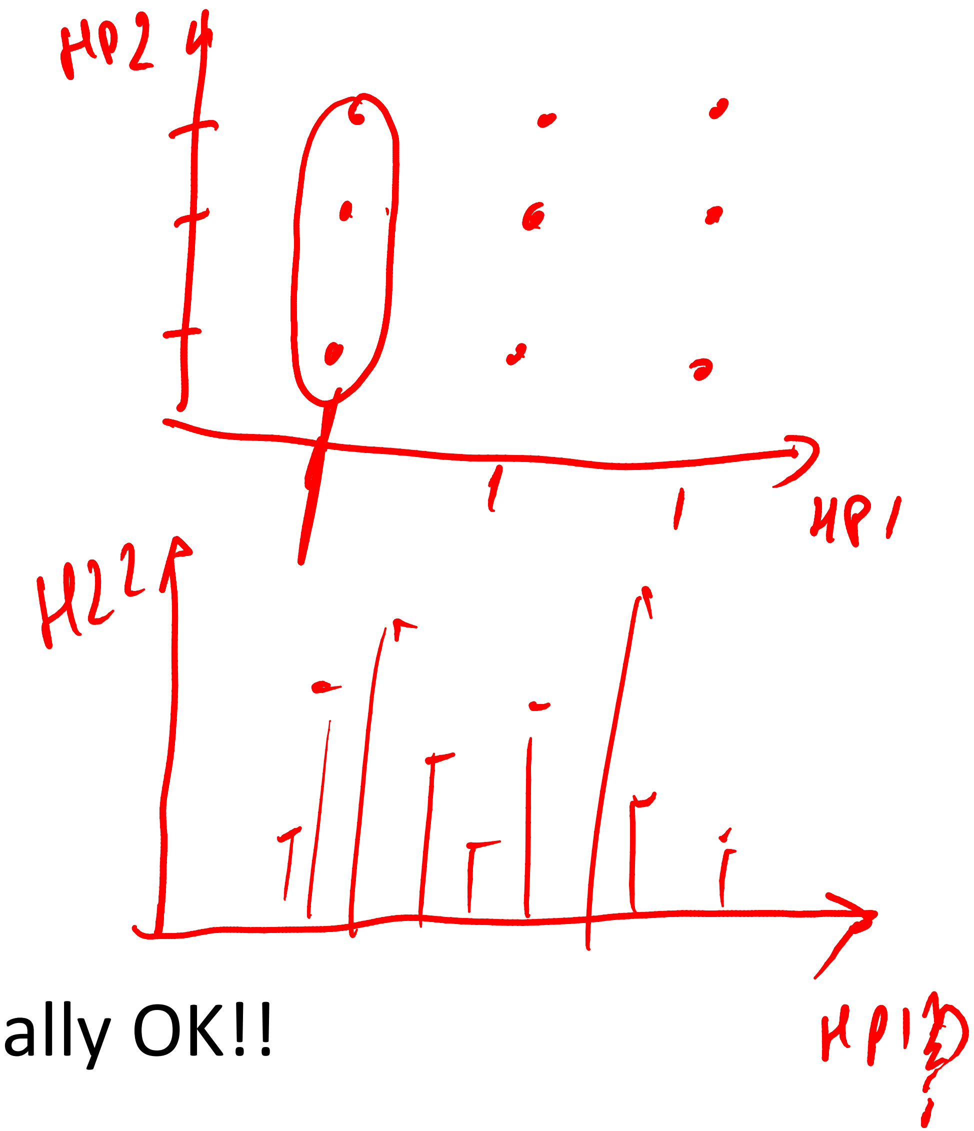
Training a net requires many decisions:

- How many layers, how many neurons
- What weight decay
- What learnign rate, what momentum
- What dropout
- Use Batch norm?

We choose them on the validation set!

Q: how to organize search?

Random hyperparam search is easy and usually OK!!



Hyperparameters: smarter approaches

The random search is surprisingly good.

However, more sophisticated appraches exist.

- https://github.com/JasperSnoek/spearmint provides a method that uses Gaussian Processes to estimate hyperparameter impact
- Many cloud providers have their own tools, e.g. https://cloud.google.com/ai-platform/training/docs/using-hyperparameter-tuning