Deep Learning (BEV033DLE)
Lecture 7. Regularization

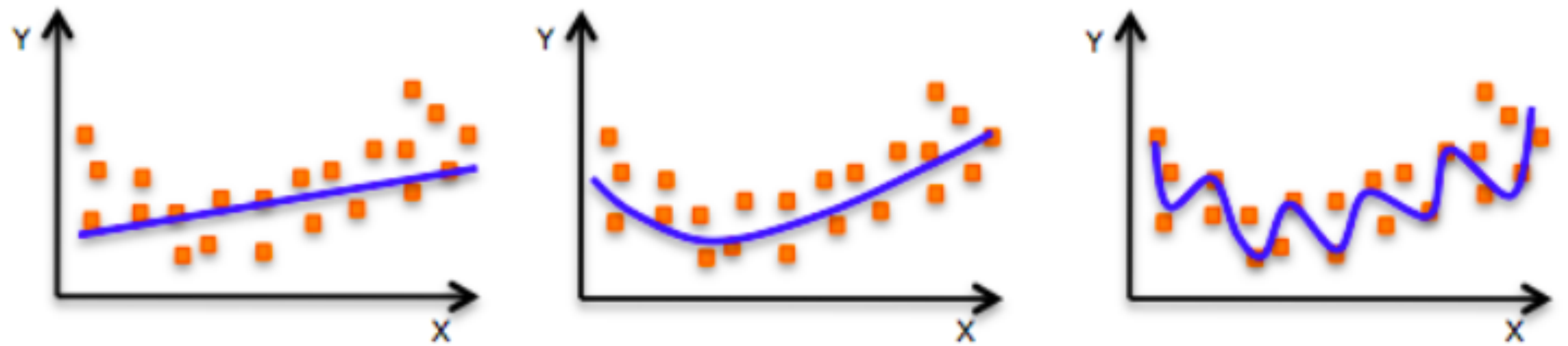
Czech Technical University in Prague

- Recap of Overfitting Issues
- L2 regularization (Weight Decay)
- Dropout
- Implicit Regularization and Other Methods

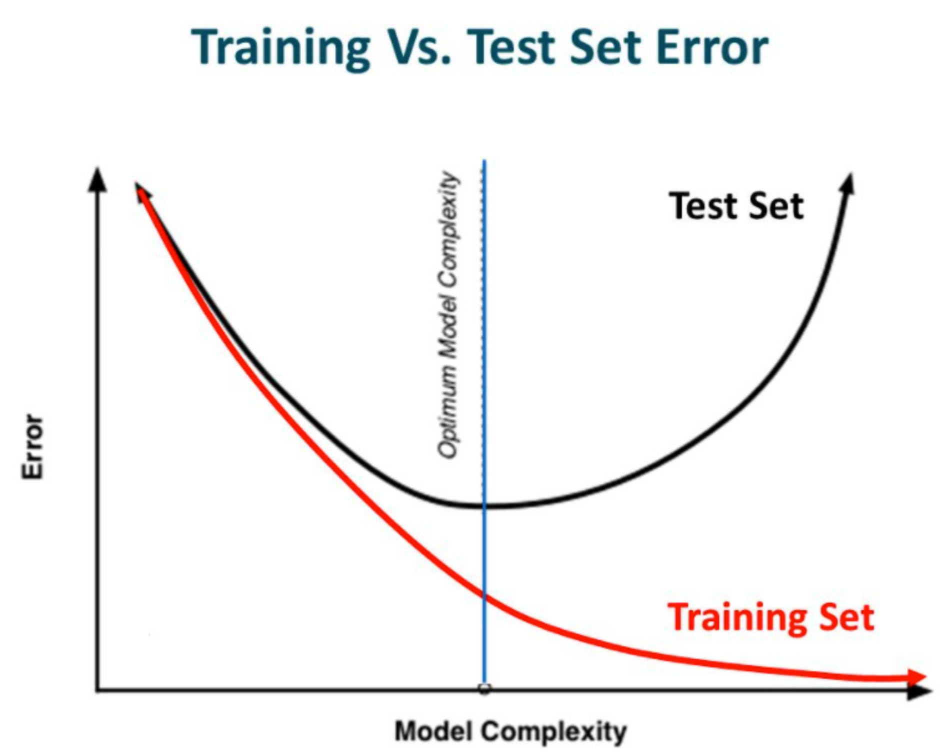
Overfitting in Deep Learning (Recall)

Underfitting and Overfitting

- Classical view in ML:



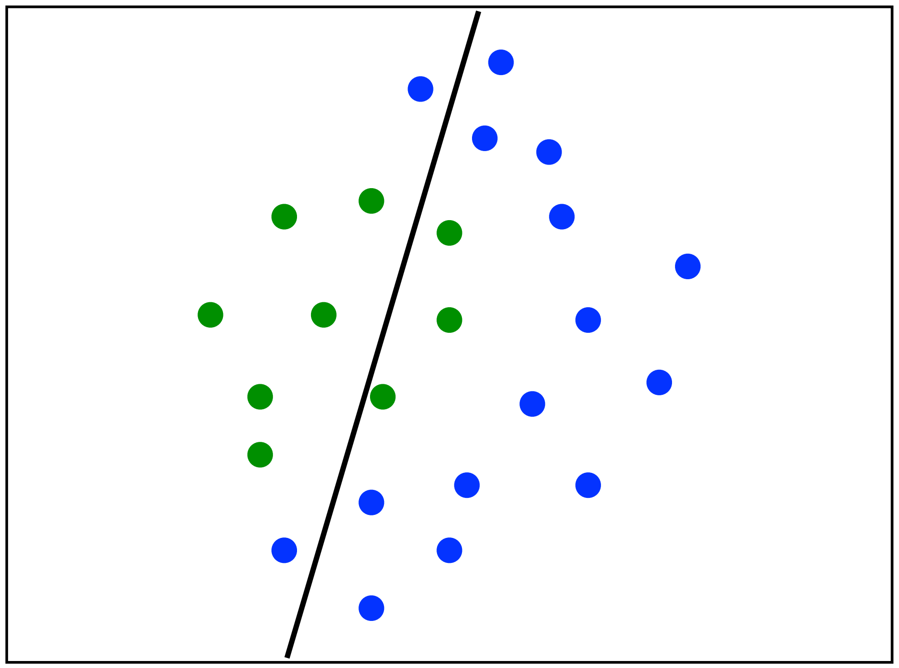
Underfitting — capacity too low | Just right | Overfitting — capacity to high



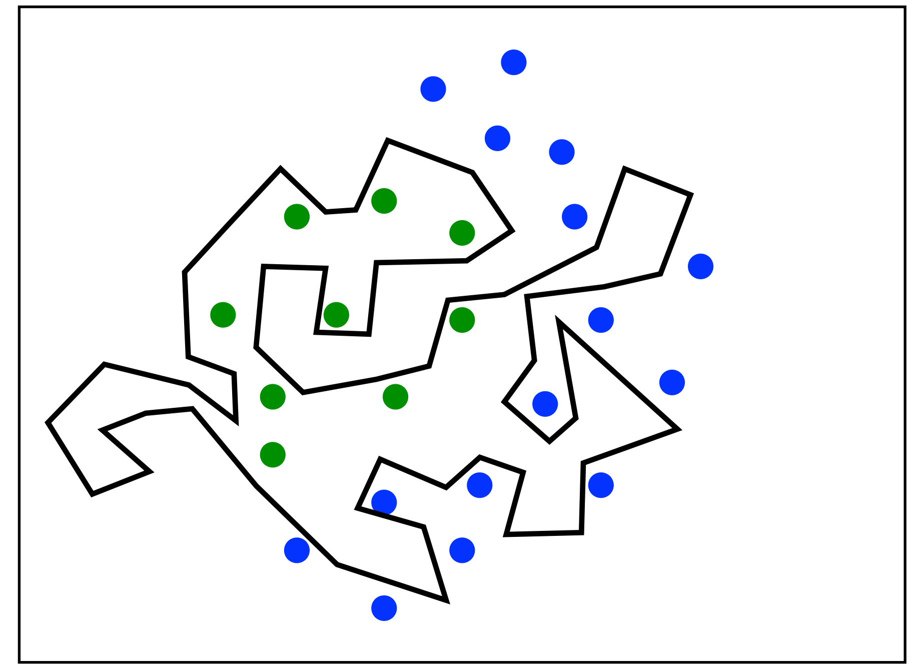
- Control model capacity (prefer simpler models, regularize) to prevent overfitting
  - In this example: limit the number of parameters to avoid fitting the noise

Underfitting and Overfitting

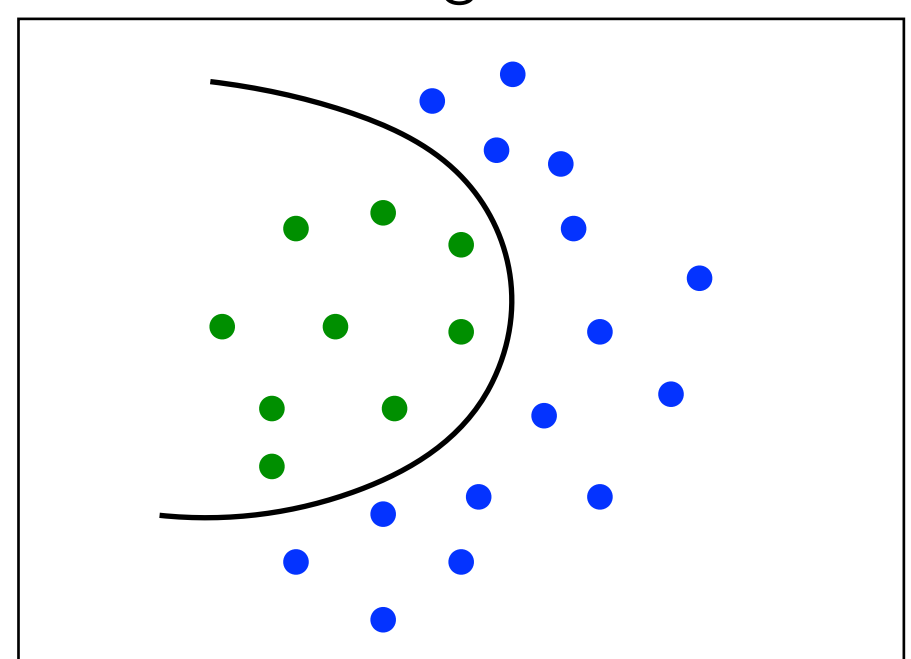
Underfitting — model capacity too low



Overfitting — model capacity too high



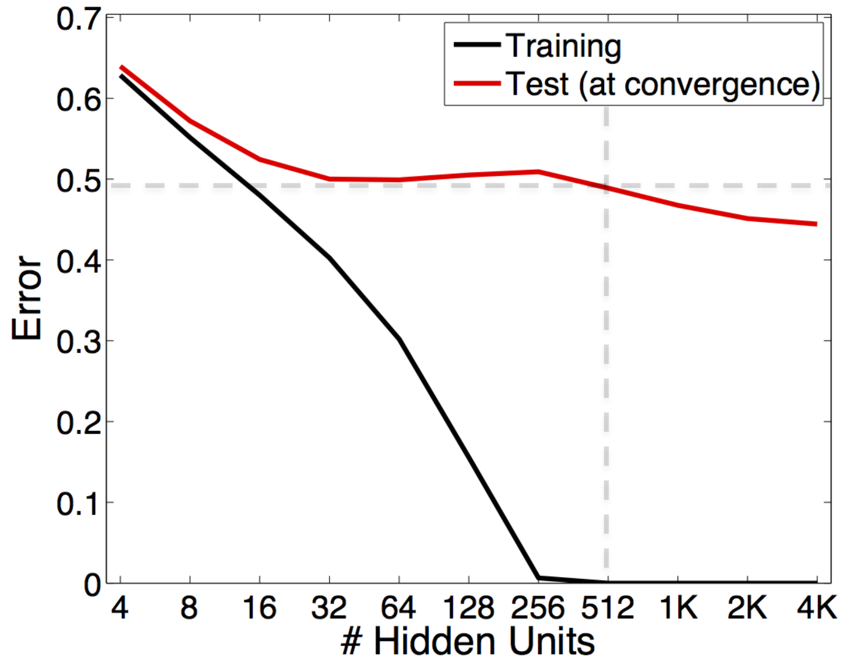
Good overfitting?

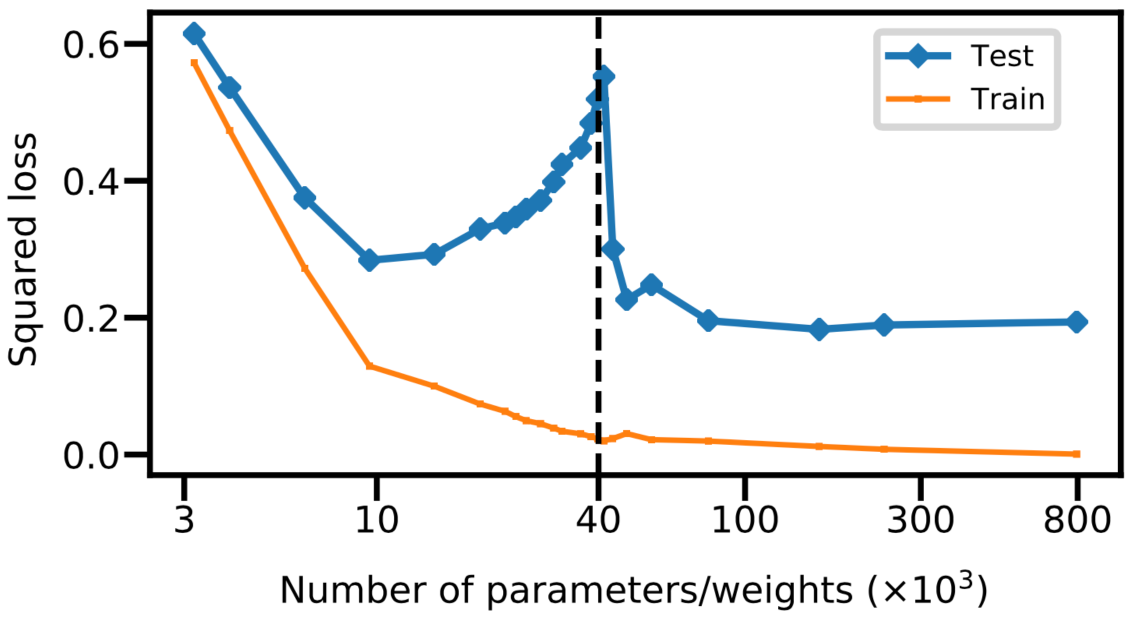


- Models in practice are chosen to perfectly fit training data (overparametrized)
- The boundary may be arbitrary complex as they can fit any labeling

Generalization of Over-Parametrized Models

- Good architecture + SGD generalizes better in the overparametrized regime





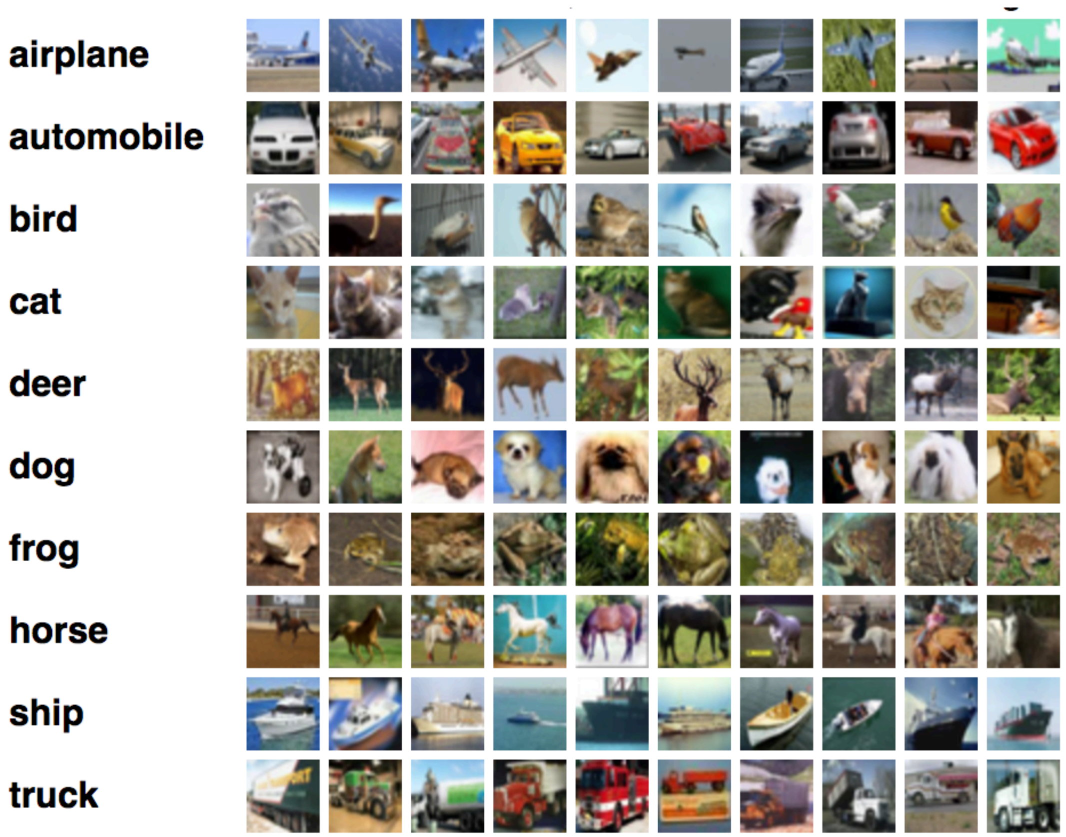
[Neyshabur et al. (2015) In Search of the Real Inductive Bias: On the Role of Implicit Regularization in Deep Learning]

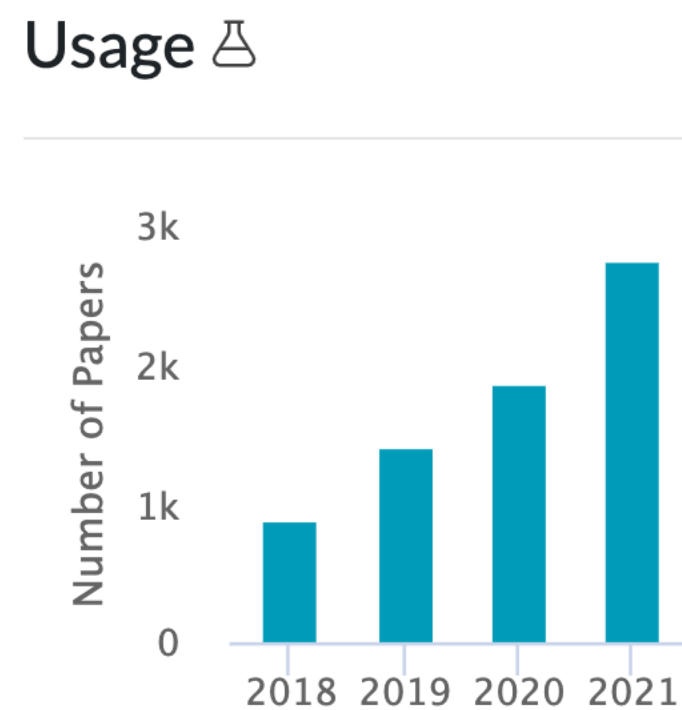
[Belkin et al. (2019) Reconciling modern machine learning practice and the bias-variance trade-off]

- Regularizing by controlling only the number of parameters is not the best option
- Important to regularize by other means:
  1. Good model architecture (putting our knowledge of invariances and useful information processing blocks into the network structure)
  2. Many other components affect implicit regularization properties (optimizer, batch size, normalization etc.)
  3. Explicit regularization

CIFAR10 Example

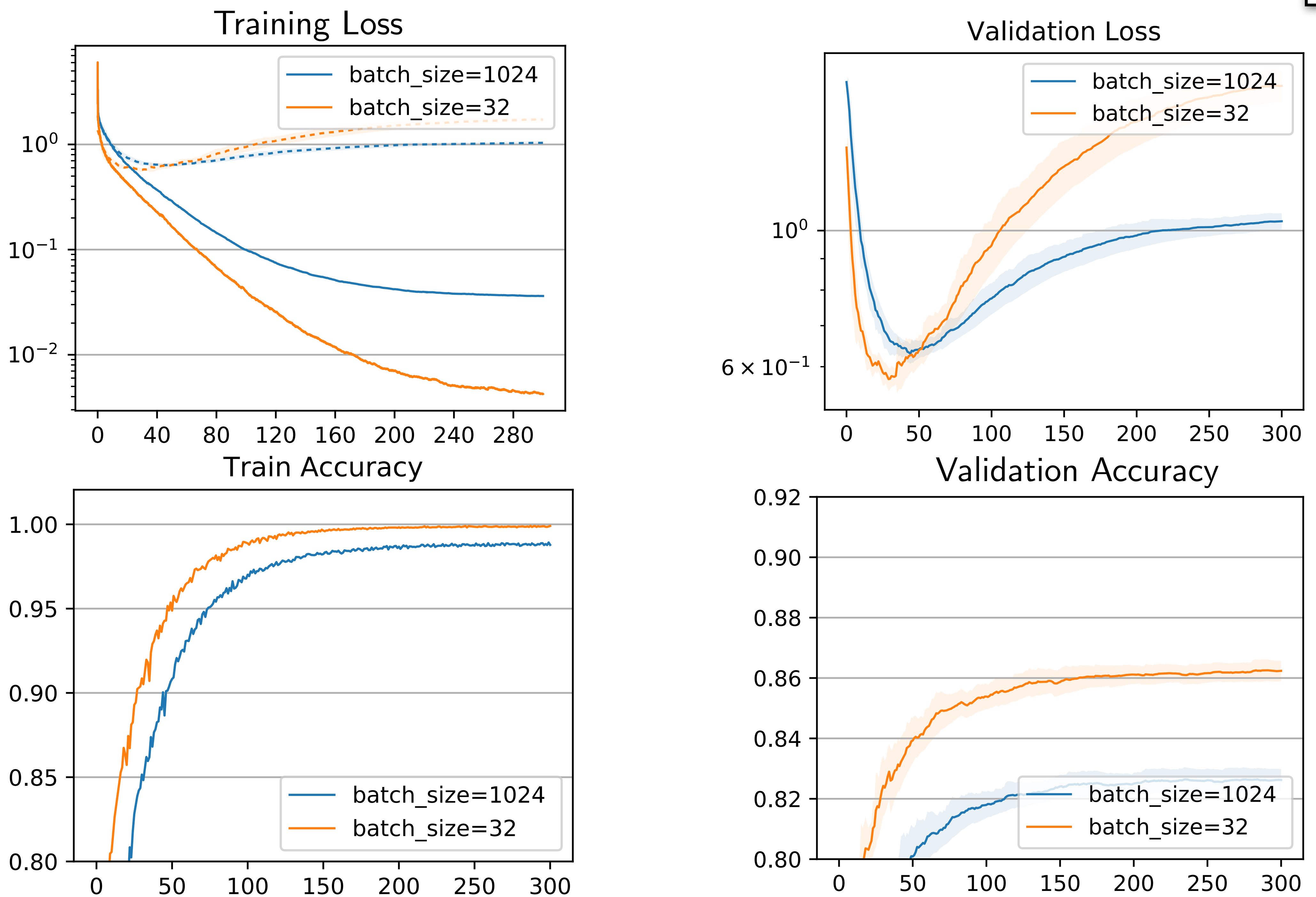
- CIFAR10 dataset
  - 60000 32x32 color images in 10 classes, with 6000 images per class.
  - 50000 training and 10000 test





[paperswithcode.com]

CIFAR10 Example: Overfitting

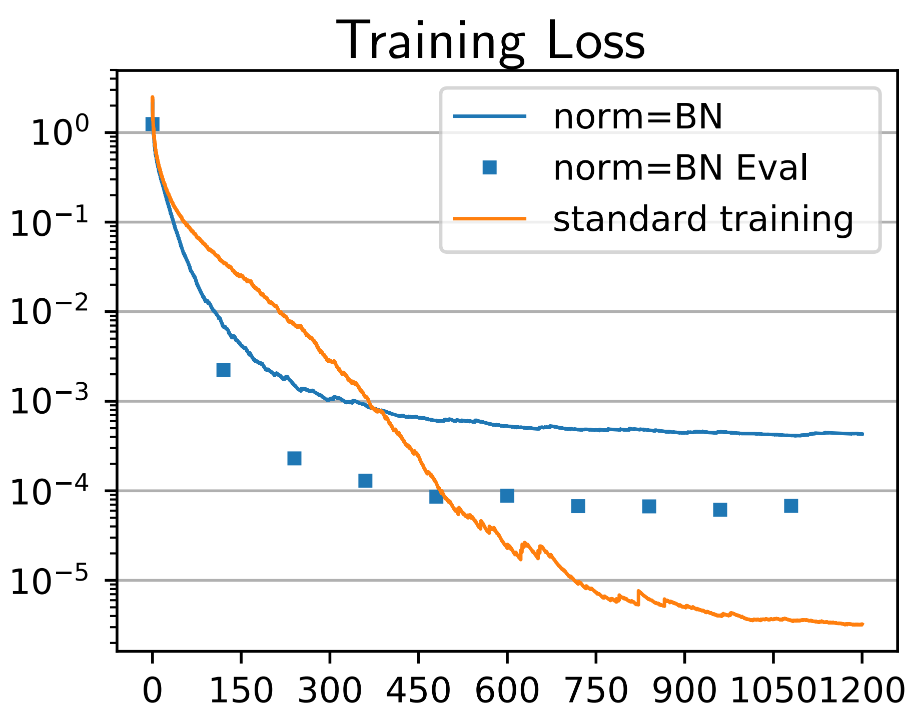


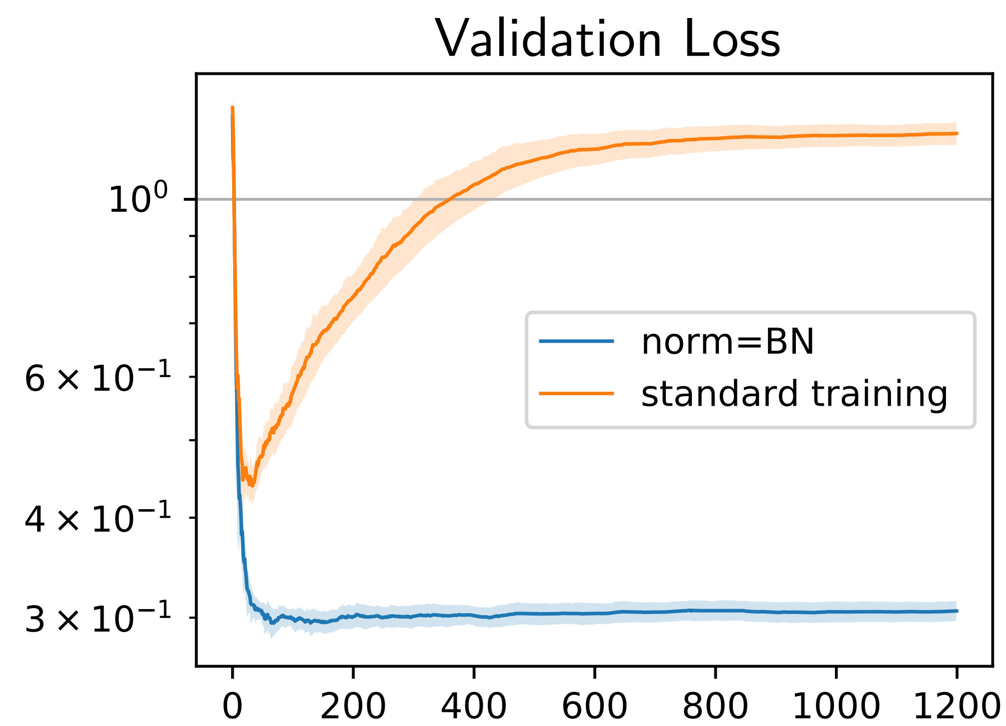
- Training loss approaches 0
- Training accuracy approaches 100%
- Validation loss starts growing
- Validation accuracy may still be improving but the model becomes overconfident

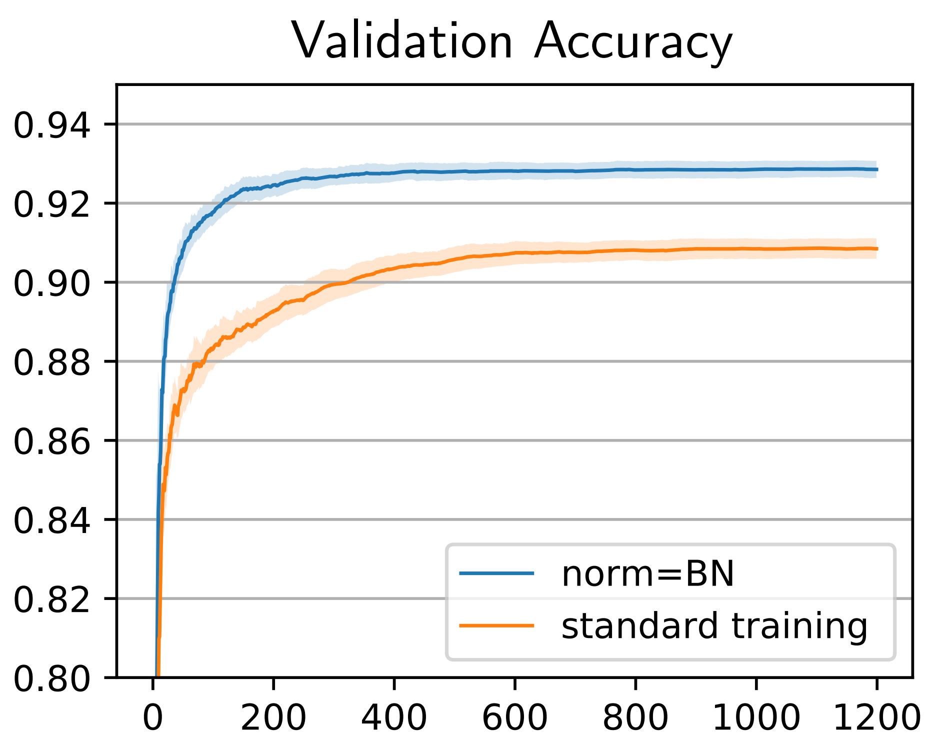
CIFAR10 Example: BN

✦ BN has a strong regularization effect!

- It depends on a randomly formed batch -> injecting specific structured noises
- The normalization bends the parameter space -> different behavior of SGD







L_2 Regularization (Weight Decay)

General Setup

- Regularized training objective:

  $\min_\theta L(\theta) + \lambda R(\theta) = \min_\theta \sum_i l_i(y_i|x_i;\theta) + \lambda R(\theta)$

  - $R(\theta)$ - function not depending on data
  - λ - regularization strength

- Recall connection to maximum a posteriori parameter estimation (MAP):

  $\max_\theta p(D|\theta)p(\theta)$

  - $p(\theta) \propto \exp(-\lambda R(\theta))$ - prior on the model weights
  - $p(D|\theta)$ - likelihood of the data given parameters
  - $p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)}$ - Bayesian posterior over parameters

  RPZ lecture 3:(Parameter Estimation: Maximum a Posteriori (MAP))

- In practice also commonly appears in the form independent of the amount of data:

  $\min_\theta \frac{1}{n} \sum_i l_i(y_i|x_i;\theta) + \lambda R(\theta)$

  - λ is tuned for a given dataset with cross-validation

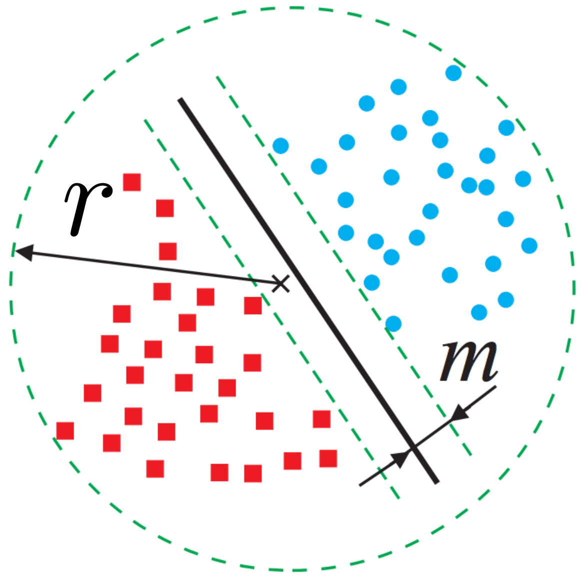
Well-understood in Linear Models

- L_2-regularization (l_2, weight decay):

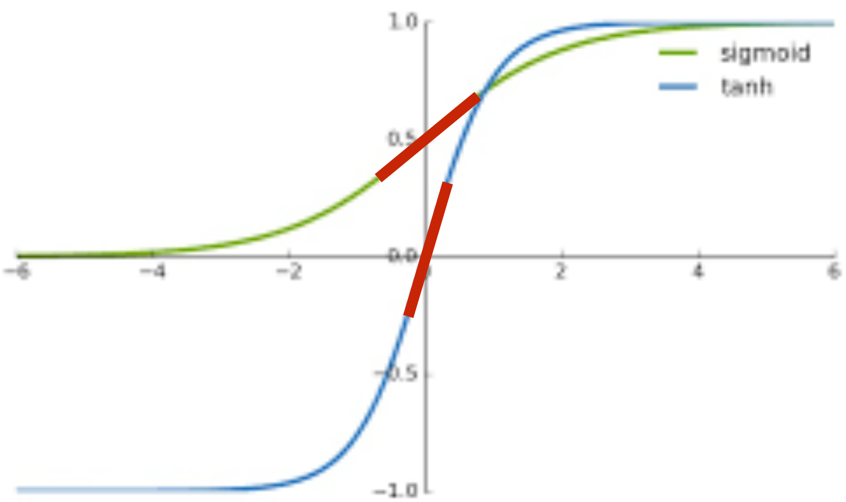
  $R(\theta) = \|\theta\|^2$

- In **linear regression**:
  - Known as ridge regression, Tikhonov regularization
  - Equivalent to using *multiplicative noise* $\mathcal{N}(1, \lambda^2)$ on the input
  - Smoothing effect (reduces the variance of $\hat{\theta}$)

- In **linear classification**:
  - Small $\theta \leftrightarrow$ large margin
  - Generalization bounds independent of dimensionality of the model (roughly): $\mathrm{Risk}(h) \leq O^*\left(\frac{1}{N}\frac{r^2+\|\xi\|^2}{m^2}\right)$, where ξ are slacks

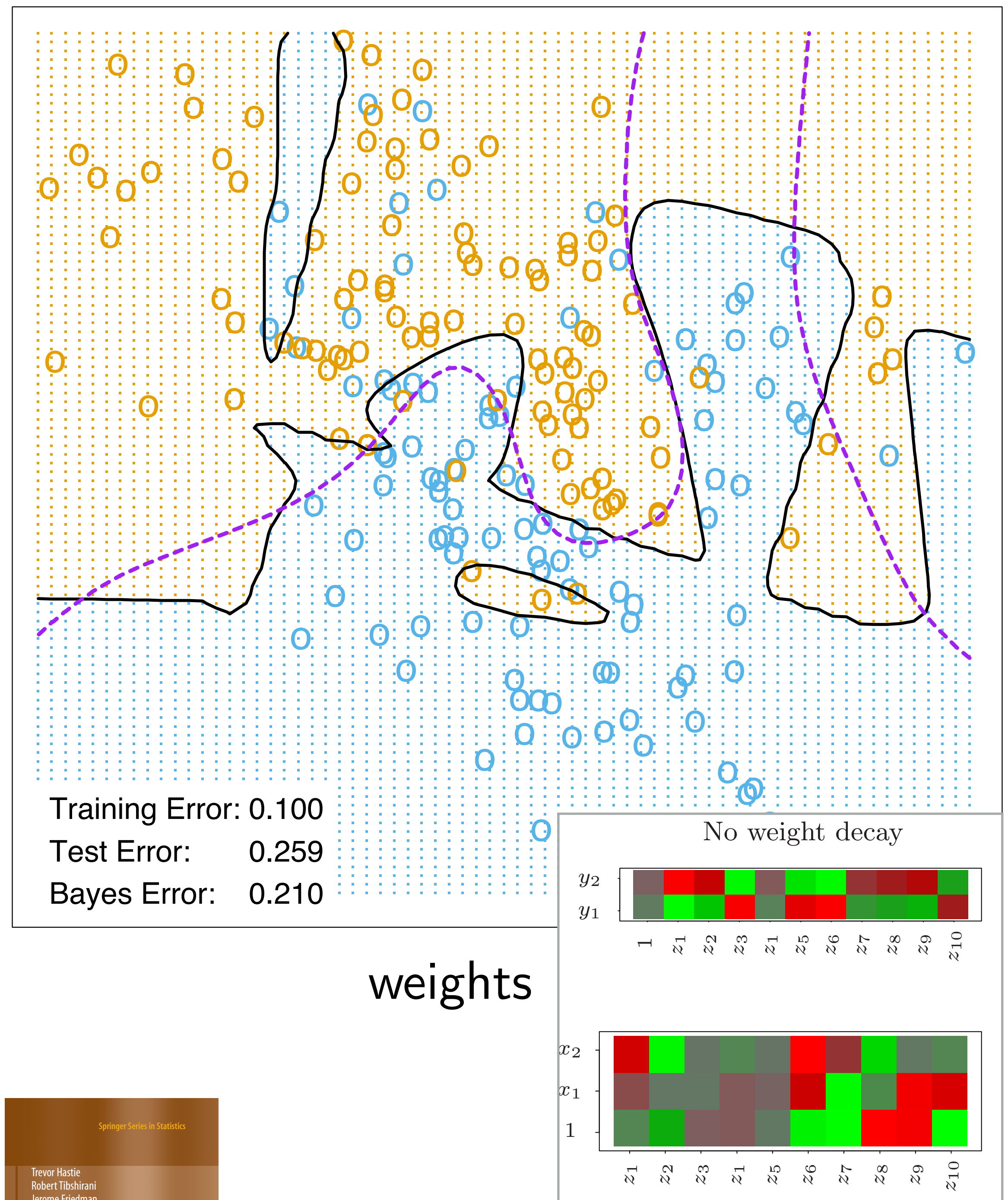


- **Sigmoid NNs**:
  - Small $\theta \rightarrow$ sigmoid outputs are close to linear $\rightarrow$ smoother classification boundary



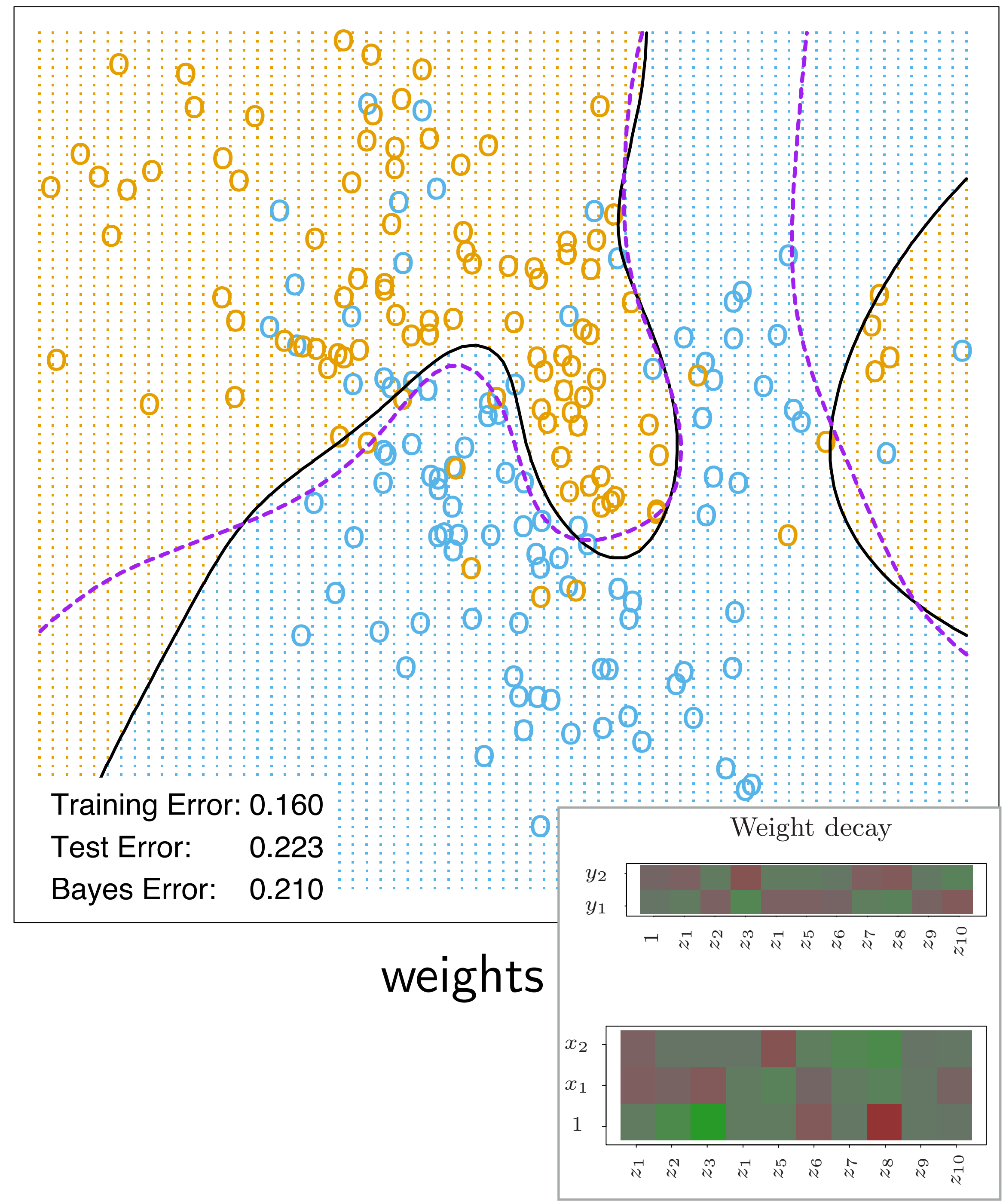
Simulated Data Example

Neural Network - 10 Units, No Weight Decay

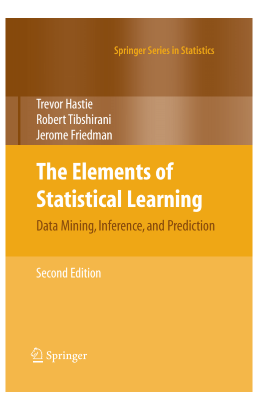


weights

Neural Network - 10 Units, Weight Decay=0.02



weights



Hastie, Tibshirani and Friedman: The Elements of Statistical Learning

https://web.stanford.edu/~hastie/ElemStatLearn/

L_2 Regularization and Batch Normalization

- Consider BN-normalized layer:

  $a = \frac{Wx+b-\mu}{\sigma}\gamma + \beta$

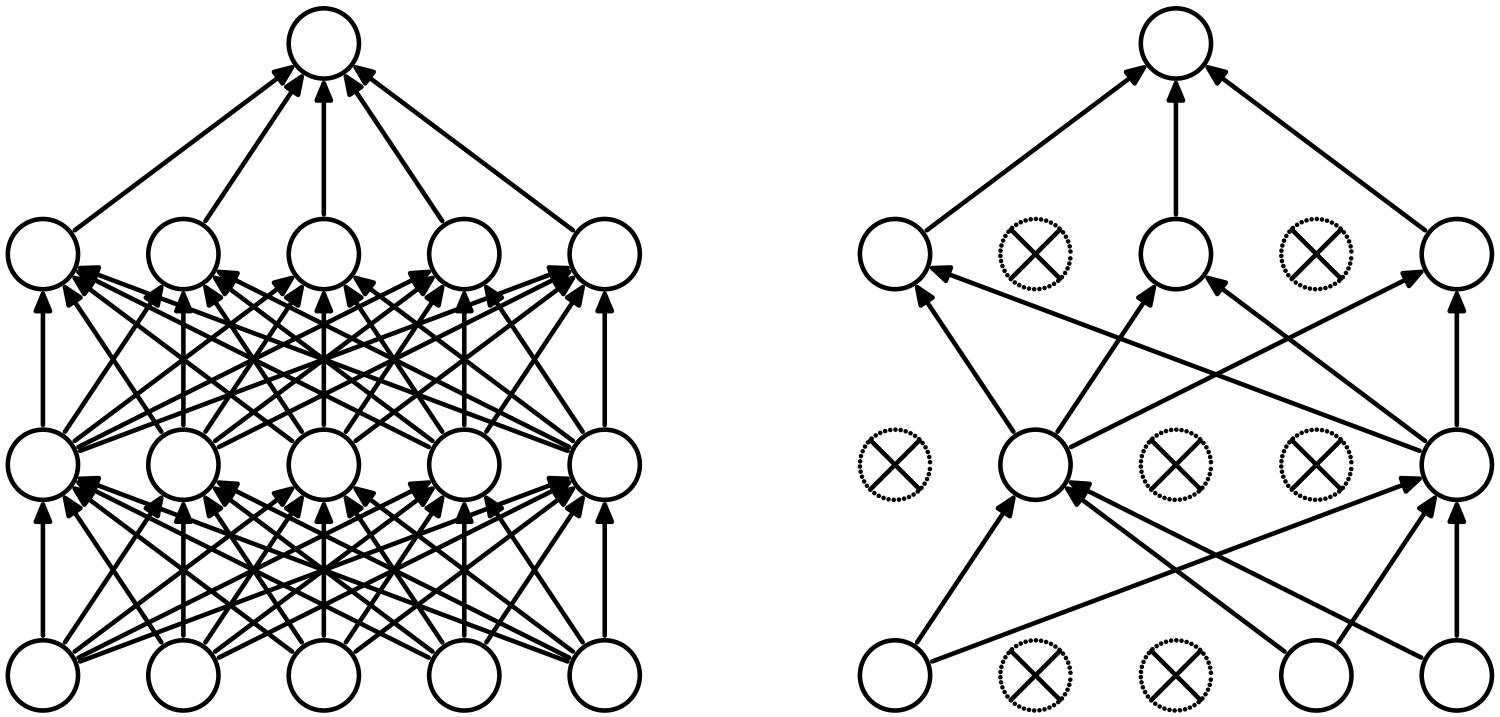
  - $\mu = \frac{1}{M}\sum_i (Wx_i + b) \qquad \sigma^2 = \frac{1}{M}\sum_i (Wx_i + b - \mu)^2$
  - Exercise: the value of a does not depend on the bias b and the scale of the weights $W \to sW$

- What will happen if we try to solve $\min\limits_W L(a(W)) + \|W\|^2$, where $L(a(W))$ is invariant w.r.t. $\|W\|$?

  - Ill-posed: optimum value is approached with $\|W\| \to 0$
  - Still works if you apply it in practice with small weight decay
  - Better to avoid such ill-specified problems

Dropout

Simple Idea



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

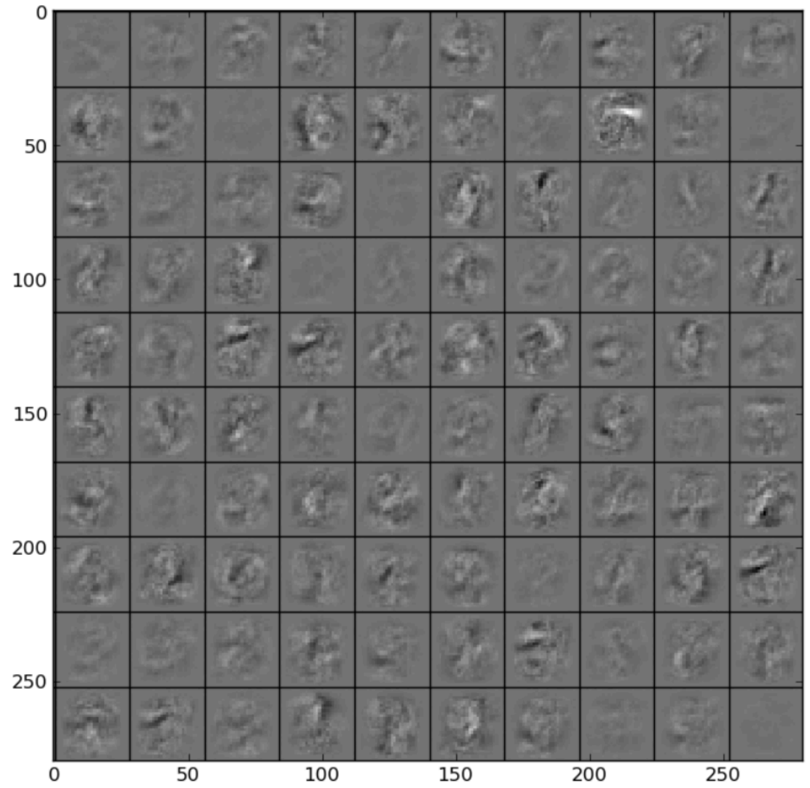
[Hinton et al. (2012) Improving Neural Networks by Preventing Co-adaptation of Feature Detectors]

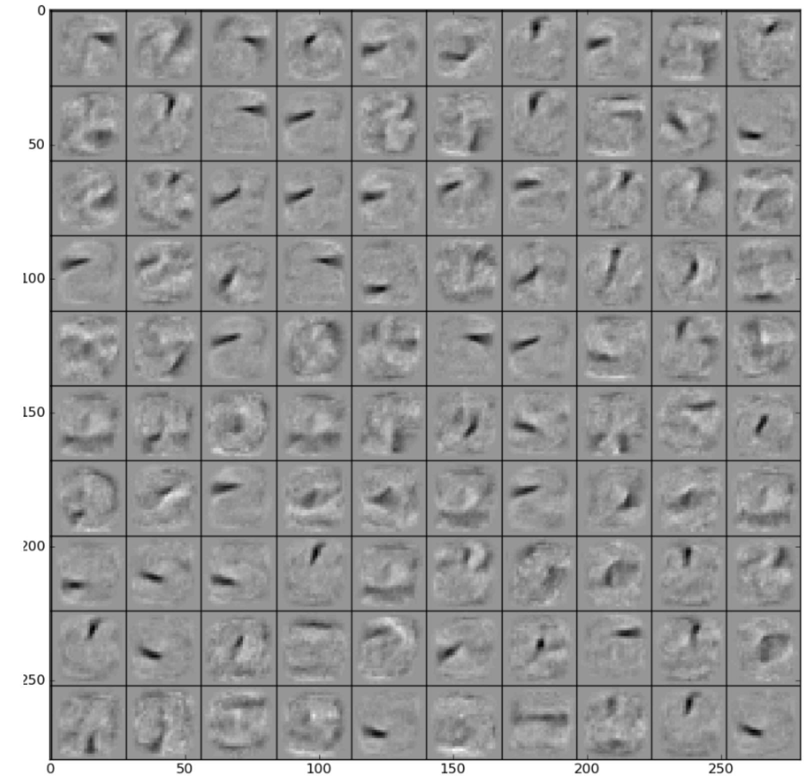
[Srivastava et al. (2014) Dropout: A Simple Way to Prevent Neural Networks from Overfitting]

- During training:
  - Randomly, "drop" some units activities -- set their outputs to zero
  - This results in the associated weights not being used and we obtain a (random) subnetwork
  - When learning, the network develops robustness to units being dropped
- During testing:
  - Use all units

Co-adaptation

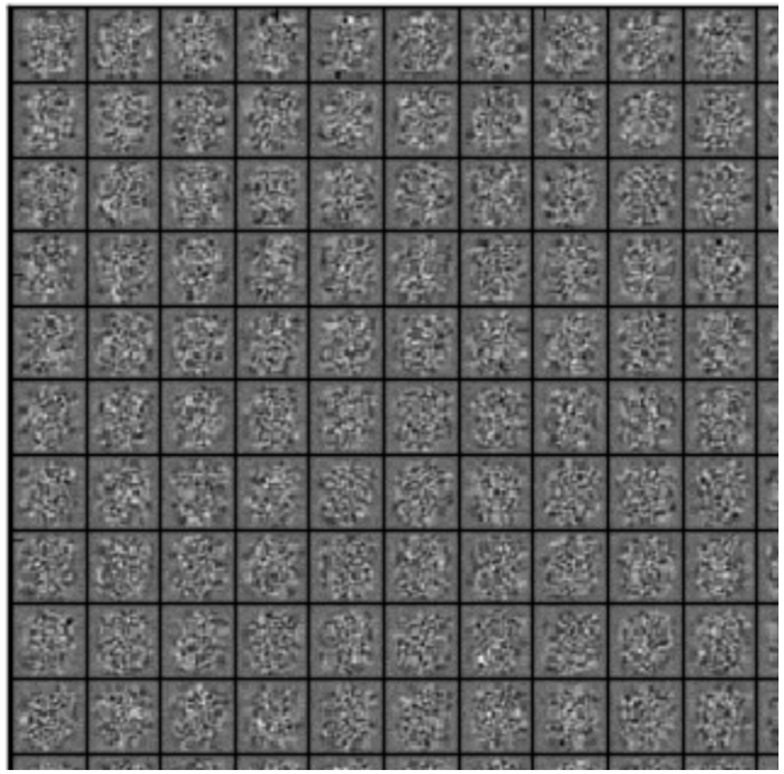
- MNIST 784-500-500 neural network, first layer features

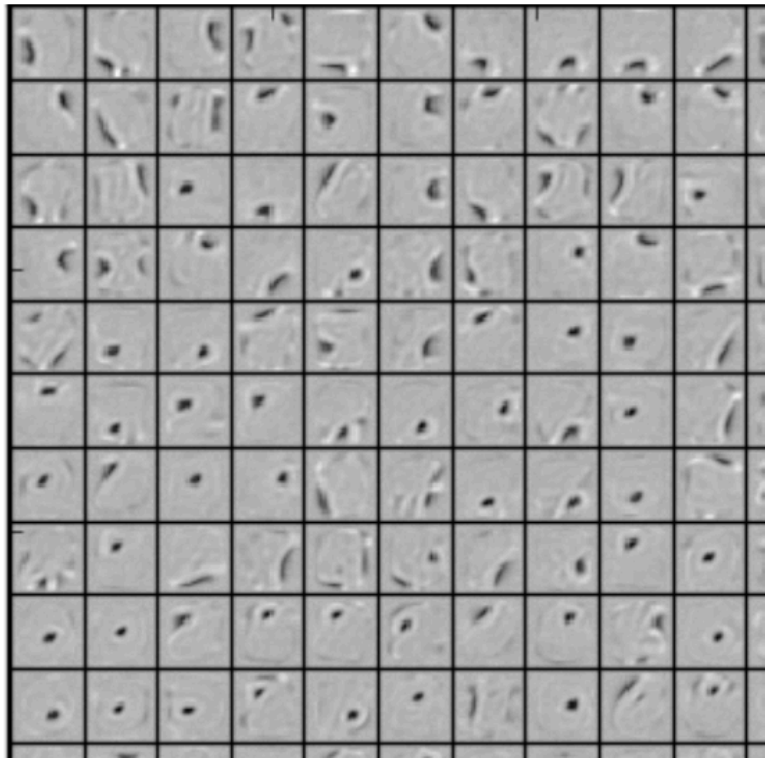




50% dropout

- MNIST autoencoder (non-variational) with 256 hidden units

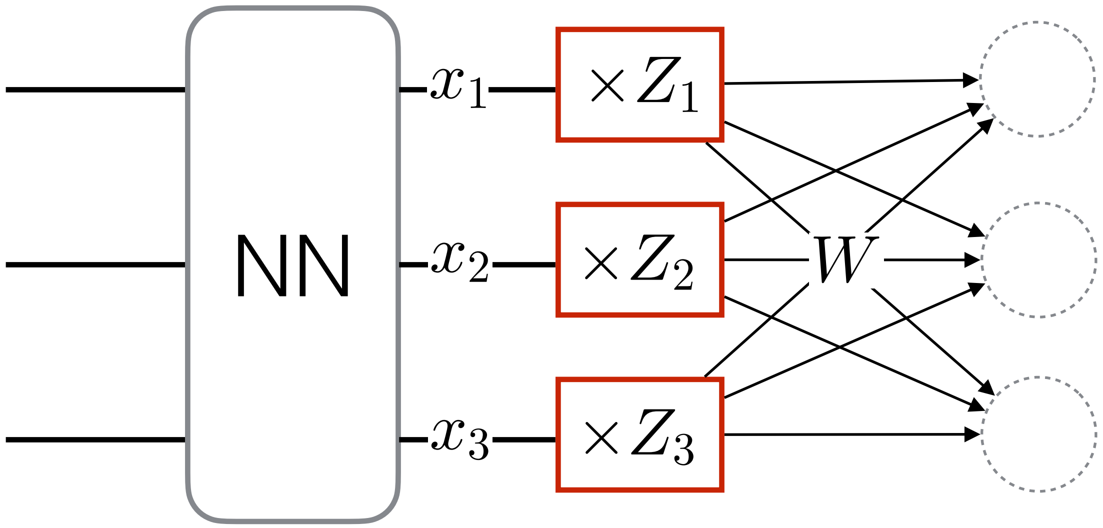




50% dropout

Mathematical Model

- What does it mean mathematically?
  - Introduce random Bernoulli variables $Z_i = \begin{cases} 1, & \text{with probability } p, \\ 0, & \text{with probability } 1-p, \end{cases}$ multiplying outputs of the preceding layer
  - Can interpret outputs multiplied with 0 as dropped
  - Drop probability $q = 1 - p$
  - Next layer activations: $a = W(x \odot Z)$
  - Gaussian multiplicative $\mathcal{N}(1, \sigma^2)$ noises work as well (Gaussian Dropout)



$Z_i \sim$ Bernoulli(0.3)

- Prediction is random now?
  - Denote the network output as $f(x, Z; \theta)$
  - We have two choices how to make predictions:
    - **Randomized predictor:** $p(y|x, Z) = f(x, Z; \theta)$
    - **Ensemble:** $p(y|x) = \mathbb{E}_Z[f(x, Z; \theta)] = \sum_Z p(z) f(x, Z; \theta)$

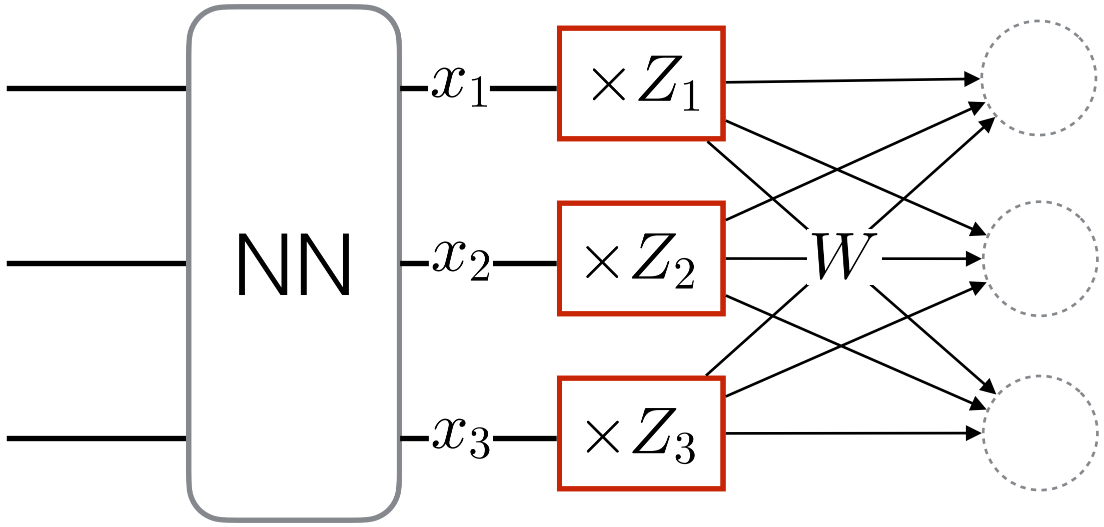
Training

- Loss of randomized predictor:
  - Double expectation in noises and date: $\mathbb{E}_Z\Big[\mathbb{E}_{(x,y)\sim\text{data}}\Big[l(y, f(x, Z; \theta))\Big]\Big]$
  - Same as: $\mathbb{E}_{Z\sim\text{Bernoulli}(q),\ (x,y)\sim\text{data}}\Big[l(y, f(x, Z; \theta))\Big]$
  - Unbiased loss estimate using a batch of size M: $\frac{1}{M}\sum_{i=1}^{M} l(y_i, f(x_i, z_i; \theta))$
- What it means practically:
  - Draw a batch of data
  - For each data point i independently sample noises z
  - Compute forward and backward pass as usual
  - Will have increased variance of the stochastic gradient

Testing

- Use approximation (common default):
  - $\mathbb{E}_Z\Big[f(x,Z;\theta)\Big] \approx f(x,\mathbb{E}_Z[Z];\theta)$
  - Since $\mathbb{E}_Z[Z]=p$, we have

    $a = W(x \odot \mathbb{E}[Z]) = (pW)x$
  - i.e. need to scale down the weights



$Z_i \sim$ Bernoulli(0.3)

$E[Z]=p$

- Use sampling:
  - $\mathbb{E}_Z\Big[f(x,Z;\theta)\Big] \approx \frac{1}{M}\sum_{i=1}^{M} f(x_i,z_i;\theta)$
  - Generalizes slightly better than the above
  - Can be used to also estimate model uncertainty

- Both variants achieve a "comity" or "ensembling" effect

averaging of many well fitting models:

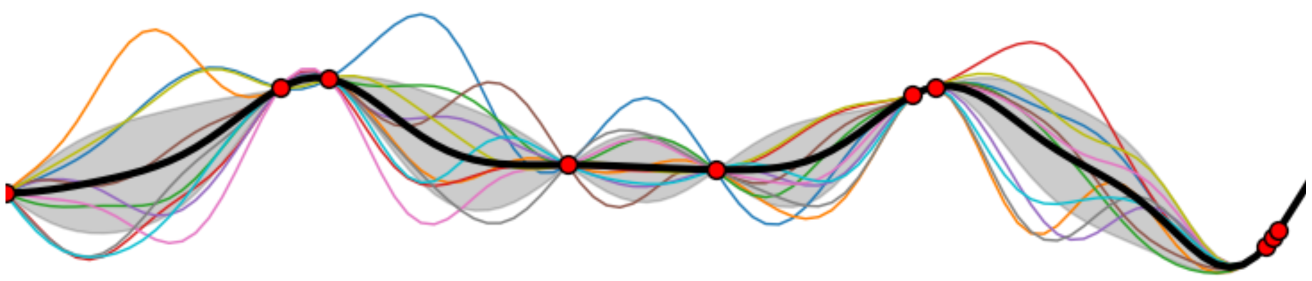
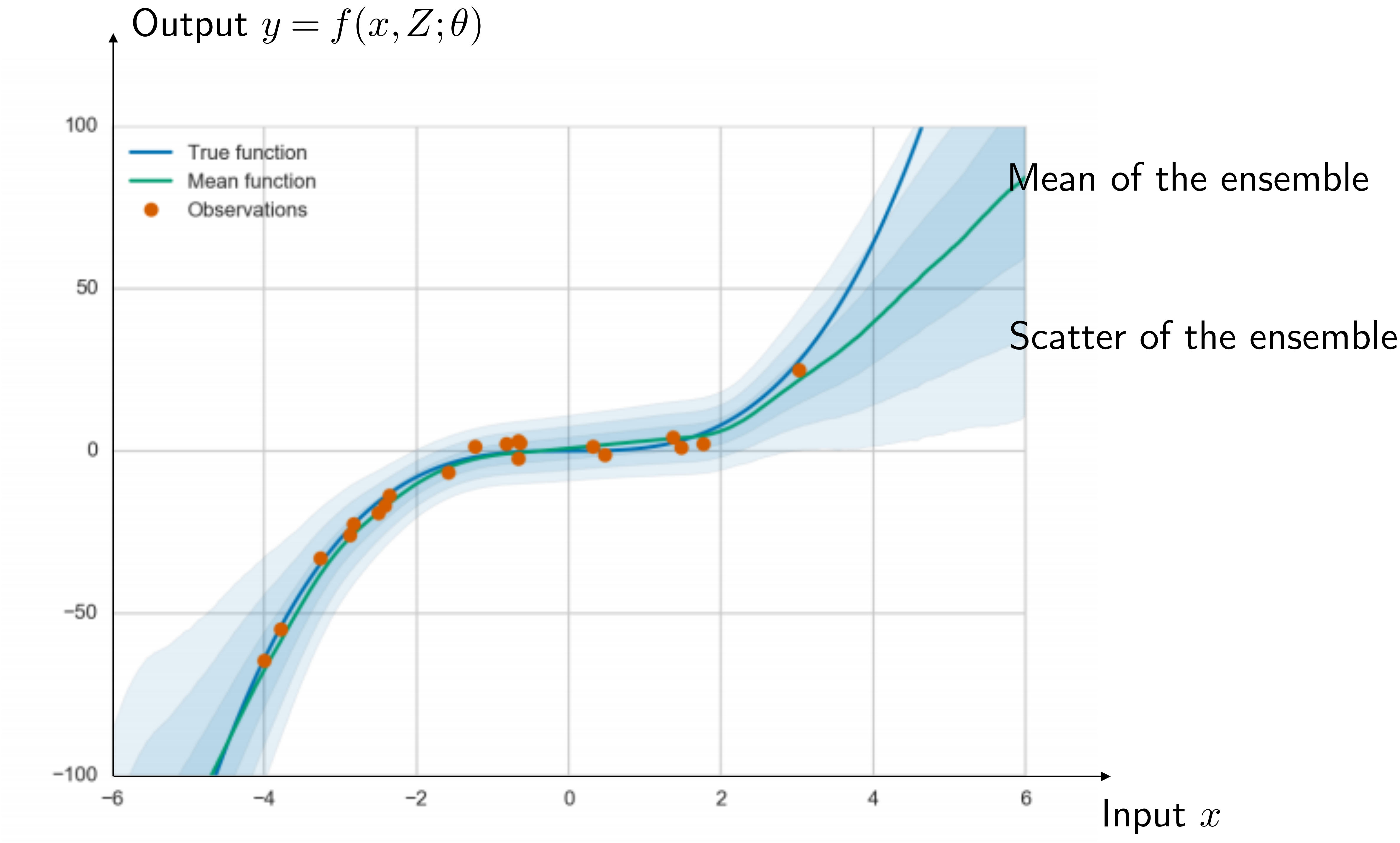


Illustration: Gaussian Process

- More accurate analytic approximations than the first option are possible

Model Uncertainty with Dropout

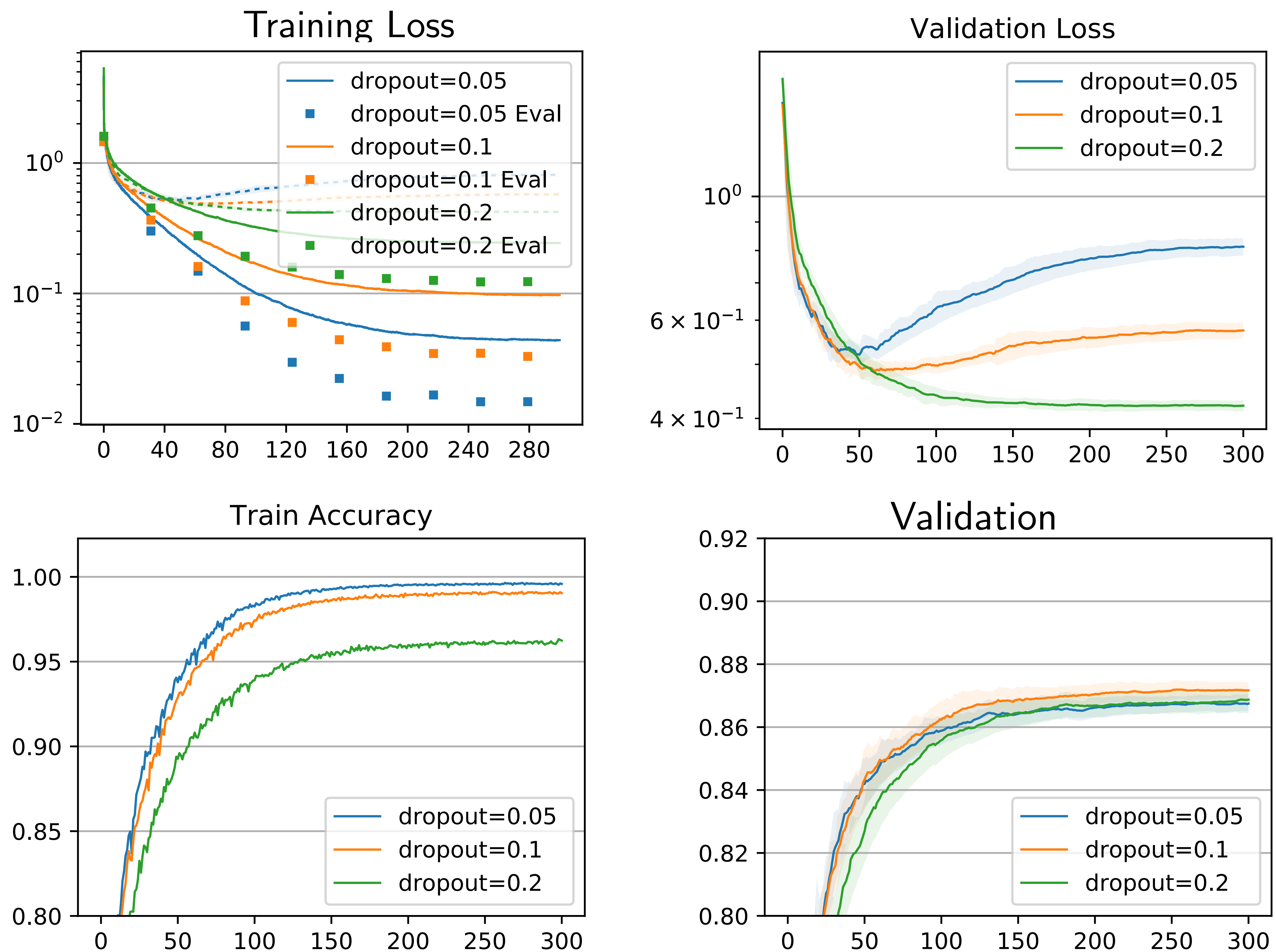
- Toy example of uncertainty estimation with dropout for regression:



[Louizos and Welling 2017: Multiplicative Normalizing Flows for Variational Bayesian Neural Networks]

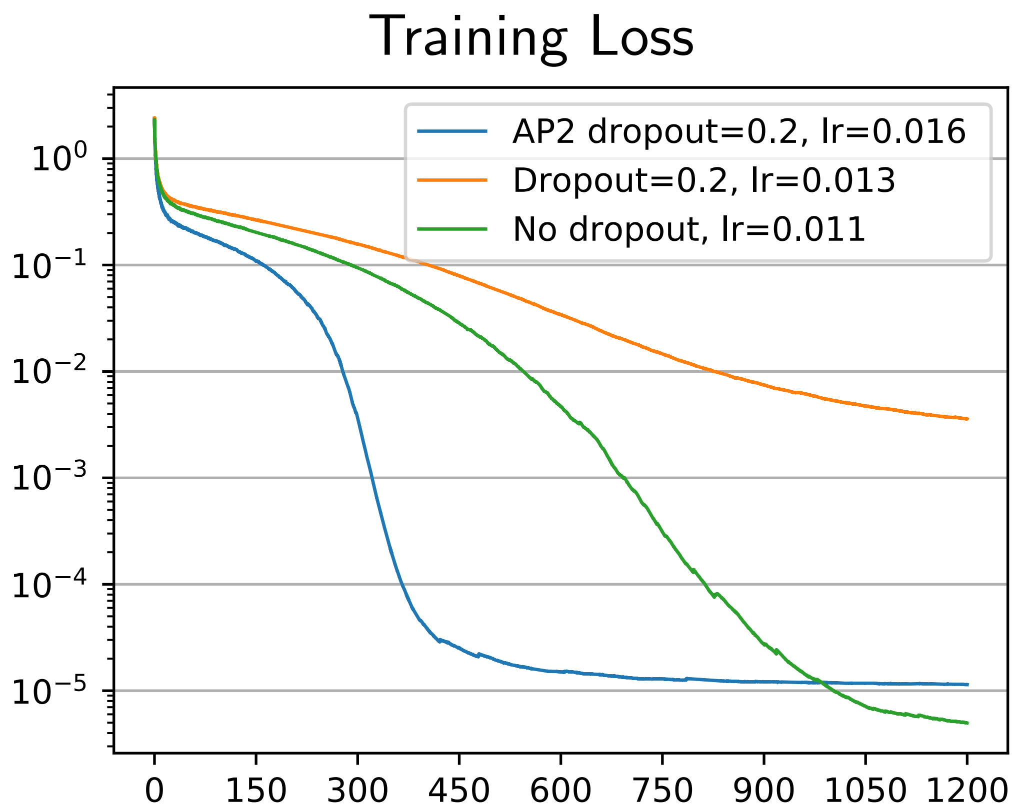
CIFAR10 Example: Dropout

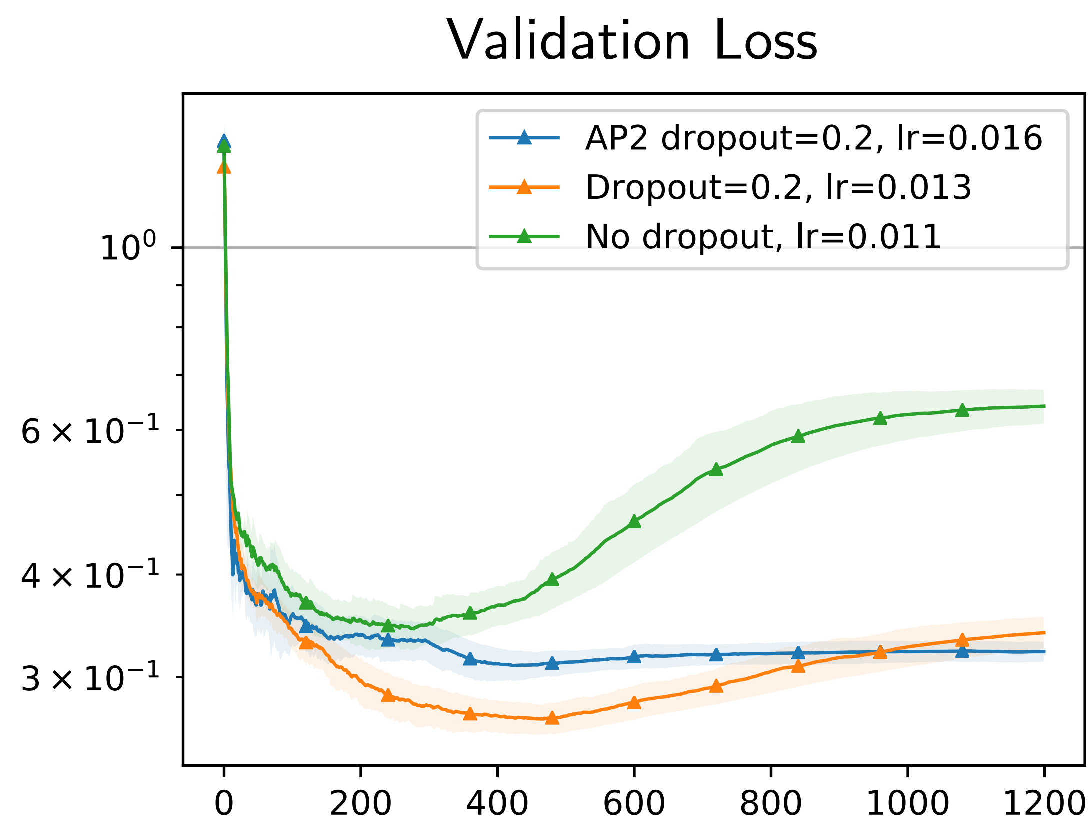
- Here BN is also used in all cases



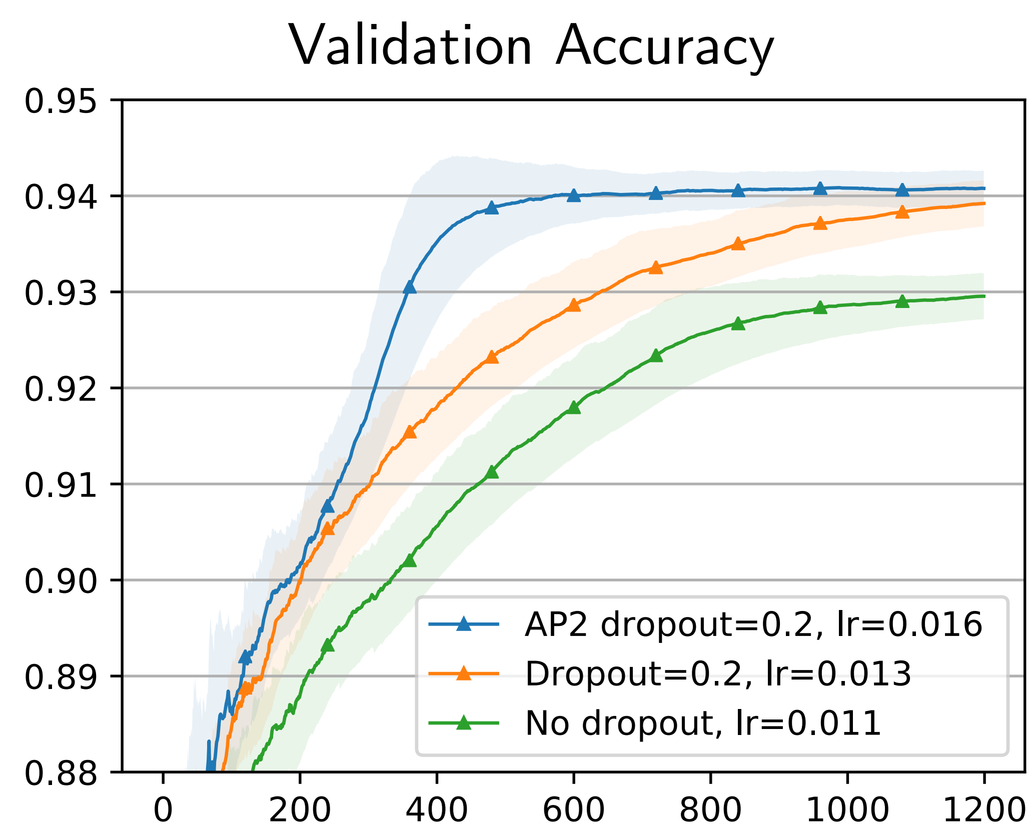
- Looks like dropout does not help for the validation accuracy, but see the next slide

CIFAR10 Example: Dropout





- Change the learning setup:
  - train longer with a slower learning rate decay
- Now it works!
  - Analytic approximations:
    Fast Dropout, Analytic Dropout (AP2)
    less gradient noise -> faster



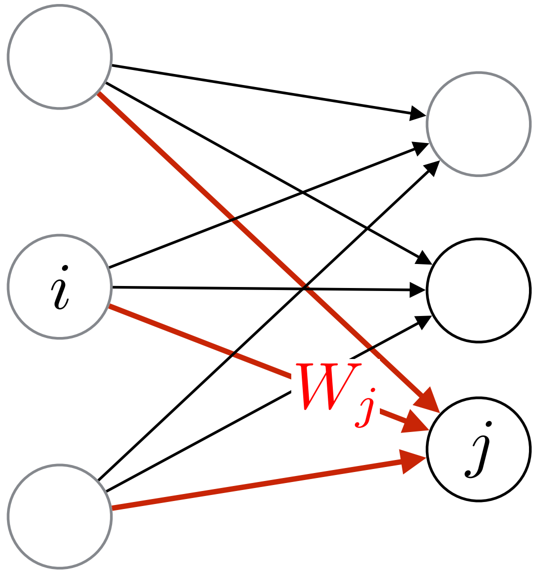
Beyond L_2 and Dropout

Other Norms

- $\mathbf{L}_1$ regularization: $R(W) = \|W\|_1 = \sum_{ij} |W_{ij}|$
  - Promotes sparsity
  - For better generalization we typically do not want sparsity (= less parameters)
- **Constrained** optimization form instead of penalty:

  $\min_W L(W)$ s.t. $R(W) \leq s$
  - Does not makes weights small, but prevents them from growing high
  - Can use projected SGD to solve
  - In particular L_2 norm on each row: $R(W) = \max_j \|W_j\|_2$

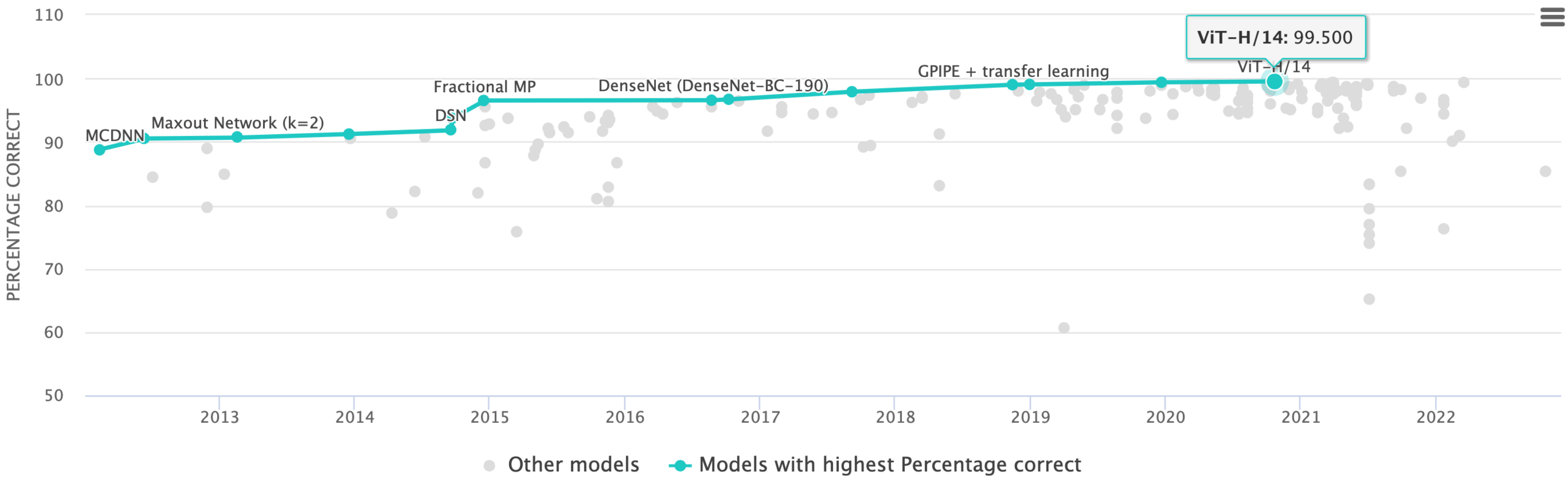
    called **max-norm** appears useful



- **Generalizations**:
  - Flat L_p norm: $R(W) = \left(\sum_{ij} W_{ij}^p\right)^{\frac{1}{p}}$
  - **Group-norm**: $R(W) = \left(\sum_j \left(\sum_i W_{ij}^p\right)^{\frac{q}{p}}\right)^{\frac{1}{q}}$
  - Above variants are special cases
  - Different generalization bounds derived measuring complexity with group norm

CIFAR10: State of the Art

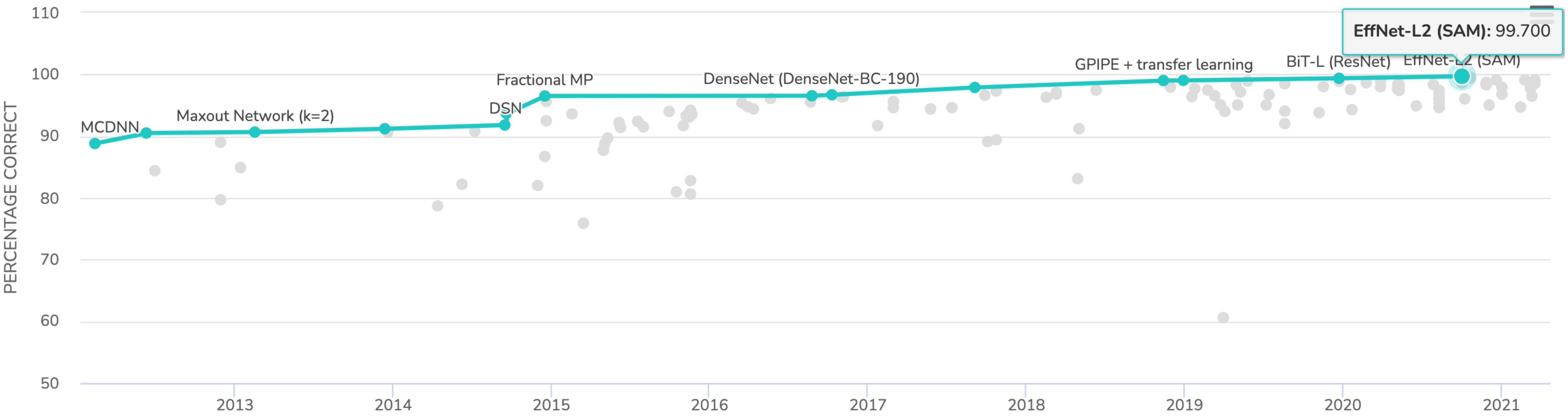
✦ CIFAR10 classification progress [paperswithcode.com]



- Architecture improvements
- Simple regularization techniques (dropout, BN, weight or activation regularization)
- Optimizers, e.g. finding stable local minima (e.g. Sharpness-Aware Minimization)
- Ensembles
- Data augmentation
- Feature Transfer (start from pertained on ImageNet)
- Auxiliary tasks (reconstruct input or its part, etc.)

State of the Art

- CIFAR10 classification progress [paperswithcode.com]



- What are the methods:
  - Architecture improvements
  - Data augmentation
  - Feature Transfer (start from pertained on ImageNet)
  - Simple regularization techniques (dropout, BN, weight or activation regularization)
  - More advanced regularization techniques: SAM = Sharpness-Aware Minimization. In Lecture 8 we will consider adversarially robust training.
  - Ensembles. More generally Bayesian neural networks is a big research topic.