Regularizing neural networks

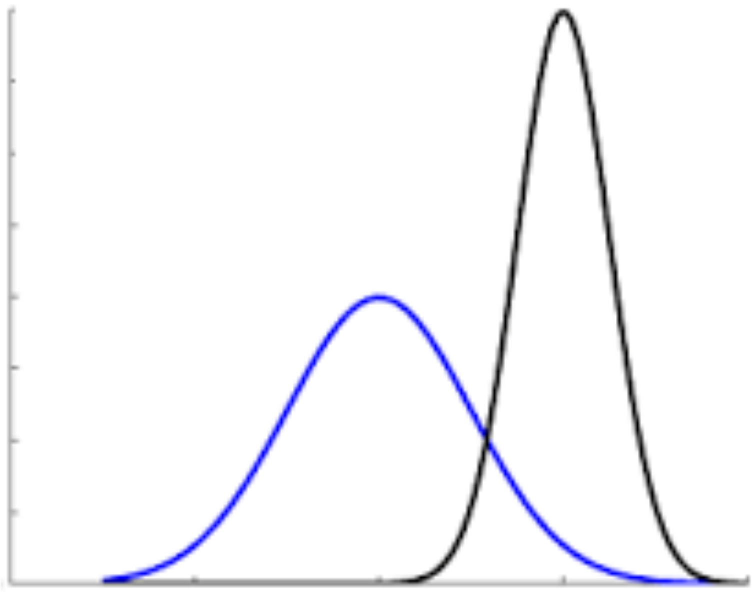
Evgeny Sokolov

Senior lecturer at HSE

Lead data scientist at Yandex.Zen

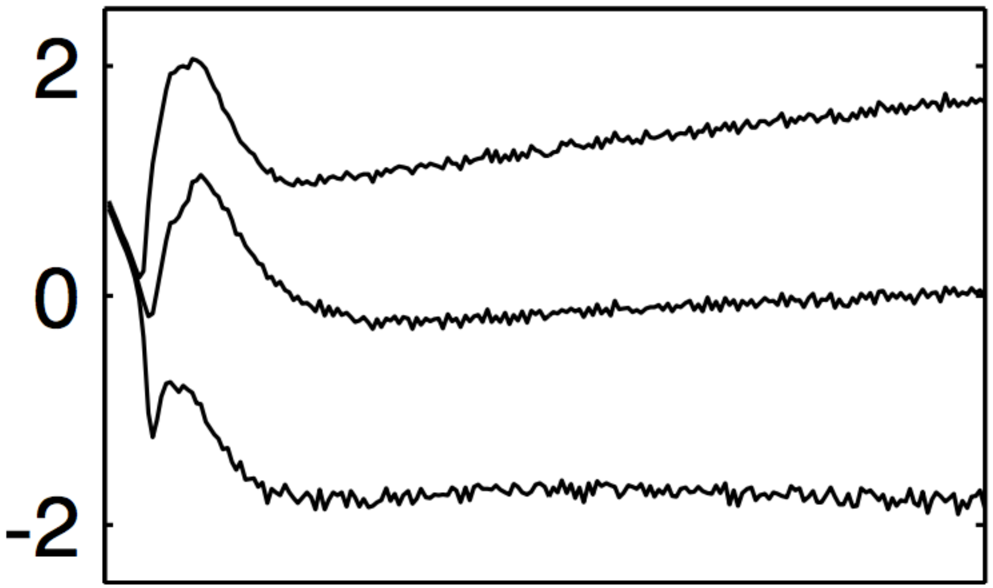
Deep|Bayes

Covariate shift



Input distribution in neural network

- Input distribution changes over the course of training
- Example: 3 fully connected layers, 100 activations each
- Strong internal covariate shift



Simple normalization

- First idea: whitening output of a layer
- $f(x; b) = x + b - \mathbb{E}(x + b)$
- If the backprop ignores dependence of sample average on b, then after step $b + g$:
$$f(x; b) = x + b + g - \mathbb{E}(x + b + g) = x + b - \mathbb{E}(x + b)$$
- Parameter b can grow indefinitely
- Gradients should take normalization into account!

Full normalization

- Normalizations layer:

$$\widehat{\mathrm{x}} = \mathrm{Norm}(\mathrm{x}, \mathcal{X})$$

- We should be able to calculate gradients:

$$\frac{\partial \mathrm{Norm}(\mathrm{x}, \mathcal{X})}{\partial \mathrm{x}} \text{ and } \frac{\partial \mathrm{Norm}(\mathrm{x}, \mathcal{X})}{\partial \mathcal{X}}$$

- Standartization requires inverse square root of covariance matrix:

$$\mathrm{Cov}(x)^{-\frac{1}{2}}(x - \mathbb{E}x)$$

Simplifications

- Normalize each layer activation independently:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

- Estimate mean and variance of input based on current mini-batch

BatchNorm

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

Gradients for BatchNorm layer

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2}(\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}$$

$$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{\sum_{i=1}^{m} -2(x_i - \mu_{\mathcal{B}})}{m}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

BatchNorm during inference

- Average mean and variance estimates over all training batches:

for $k = 1 \ldots K$ **do**

// For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.

Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$

$$\mathrm{Var}[x] \leftarrow \frac{m}{m-1} \mathrm{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

- Use these new estimates for inference:

$$y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$$

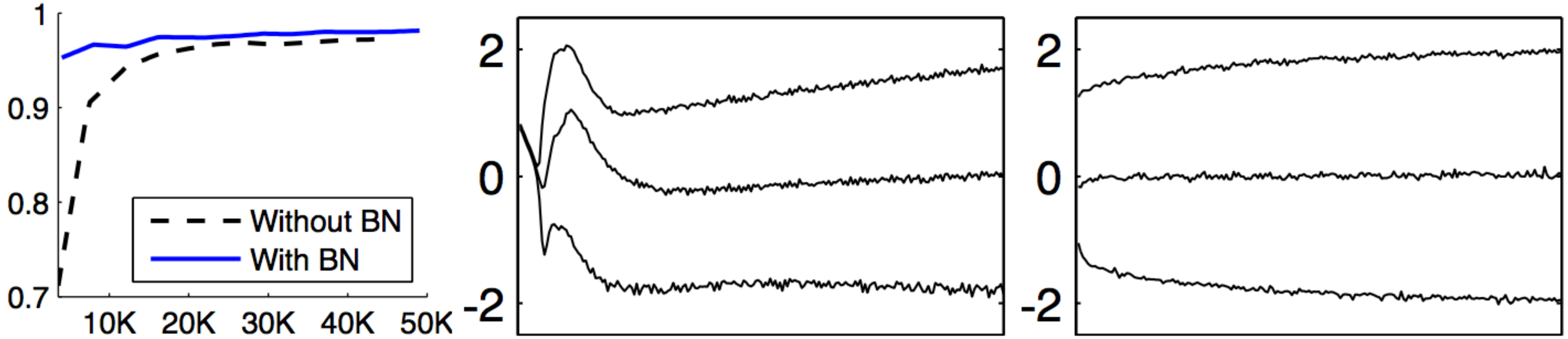
BatchNorm usage

- Usually BatchNorm is inserted before nonlinearity
- Learning rate can be increased
- Optimization becomes robust to parameter scale:

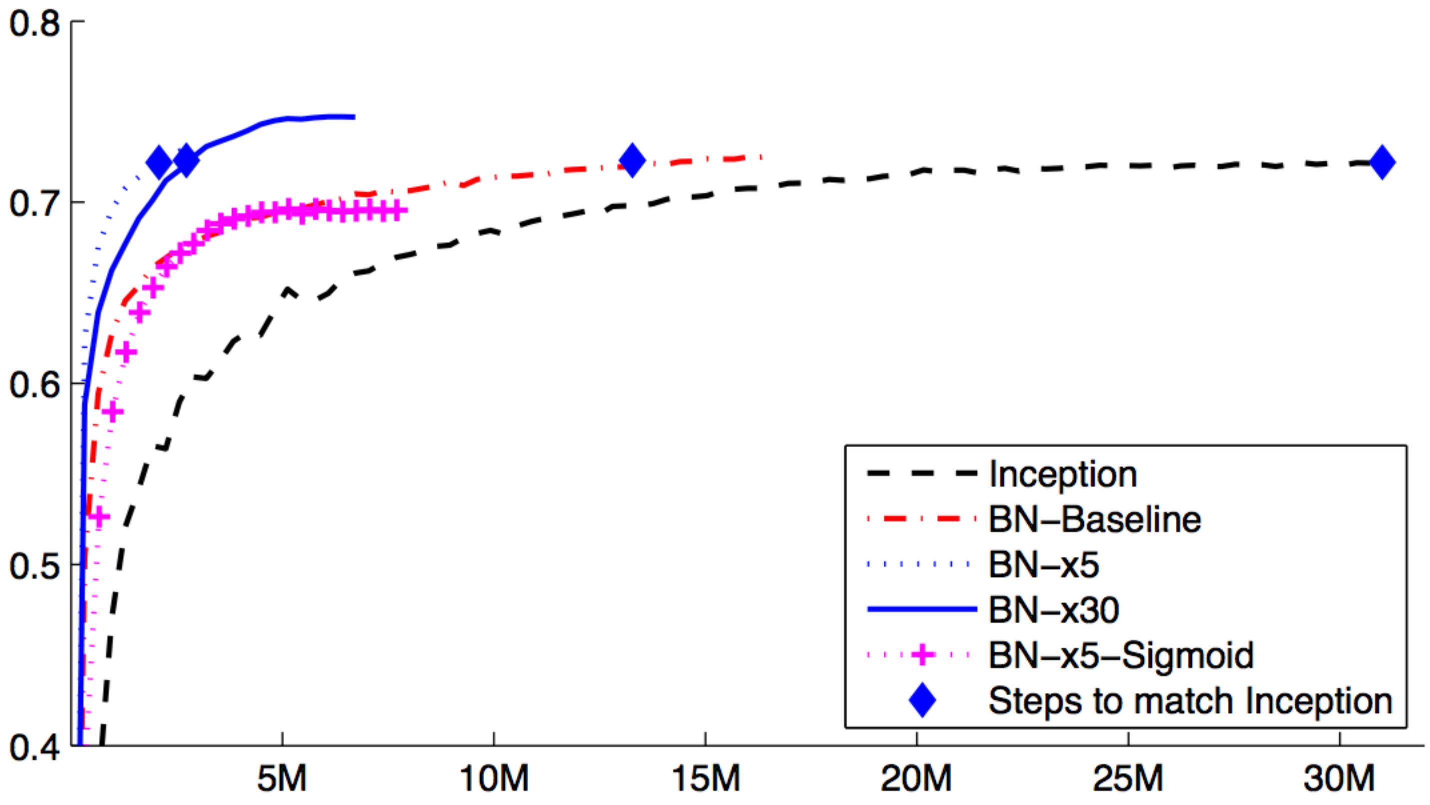
$$\frac{\partial \mathrm{BN}((aW)\mathrm{u})}{\partial \mathrm{u}} = \frac{\partial \mathrm{BN}(W\mathrm{u})}{\partial \mathrm{u}}$$
$$\frac{\partial \mathrm{BN}((aW)\mathrm{u})}{\partial (aW)} = \frac{1}{a} \cdot \frac{\partial \mathrm{BN}(W\mathrm{u})}{\partial W}$$

- Dropout layers can be removed
- Weight regularization can be reduced

BatchNorm example



BatchNorm example



WeightNorm

- Weights reparametrization:

$$\mathbf{w} = \frac{g}{||\mathbf{v}||}\mathbf{v}$$

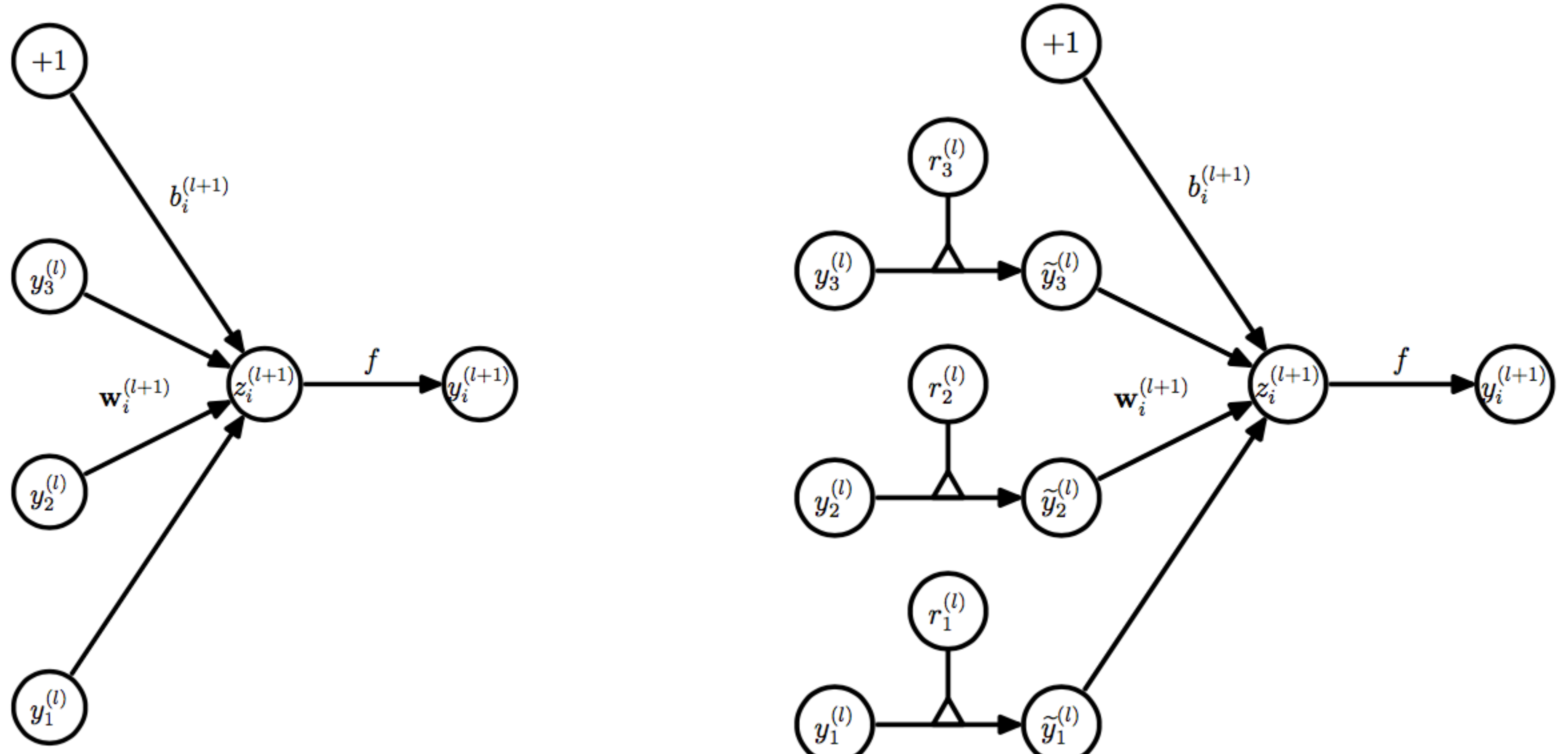
- Fixes the Euclidean norm of w to g
- Gradients:

$$\nabla_g L = \frac{\nabla_{\mathbf{w}} L \cdot \mathbf{v}}{||\mathbf{v}||}, \qquad \nabla_{\mathbf{v}} L = \frac{g}{||\mathbf{v}||}\nabla_{\mathbf{w}} L - \frac{g\nabla_g L}{||\mathbf{v}||^2}\mathbf{v}$$

WeightNorm results on CIFAR-10

Model	Test Error
Maxout [6]	11.68%
Network in Network [17]	10.41%
Deeply Supervised [16]	9.6%
ConvPool-CNN-C [26]	9.31%
ALL-CNN-C [26]	9.08%
our CNN, mean-only B.N.	8.52%
our CNN, weight norm.	8.46%
our CNN, normal param.	8.43%
our CNN, batch norm.	8.05%
ours, W.N. + mean-only B.N.	**7.31%**

Dropout



+1
$b_i^{(l+1)}$
$y_3^{(l)}$
$\mathbf{w}_i^{(l+1)}$
$z_i^{(l+1)}$
f
$y_i^{(l+1)}$
$y_2^{(l)}$
$y_1^{(l)}$
+1
$r_3^{(l)}$
$b_i^{(l+1)}$
$y_3^{(l)}$
$\widetilde{y}_3^{(l)}$
$r_2^{(l)}$
$\mathbf{w}_i^{(l+1)}$
$z_i^{(l+1)}$
f
$y_i^{(l+1)}$
$y_2^{(l)}$
$\widetilde{y}_2^{(l)}$
$r_1^{(l)}$
$y_1^{(l)}$
$\widetilde{y}_1^{(l)}$

Dropout

Randomly disable outputs of a layer:

$$r_{jk} \sim \mathrm{Bernoulli}(p)$$

$$v_{j+1}(x) = f_{j+1}\left(v_j(x) \otimes r_j; w_j\right)$$

Dropout

- Prevents co-adaptation of neurons and makes them more robust to random perturbations
- Similar to training 2^n models with shared weights
- During inference: multiply weights of dropout layer by p
- Works well with max-norm regularization

- Variational dropout can learn separate dropout weight for each node

Weight regularization

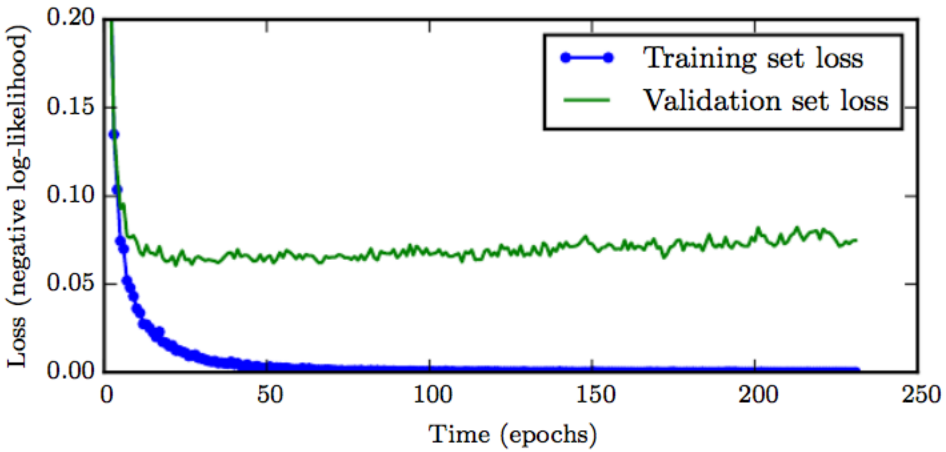
- L_2 regularization: adds $\lambda||w||^2$ to the loss function
- Penalizes peaky weight vectors
- Prevents neurons form focusing on one strong input
- Something similar to dropout

- Wager, Wang, Liang. Dropout Training as Adaptive Regularization:

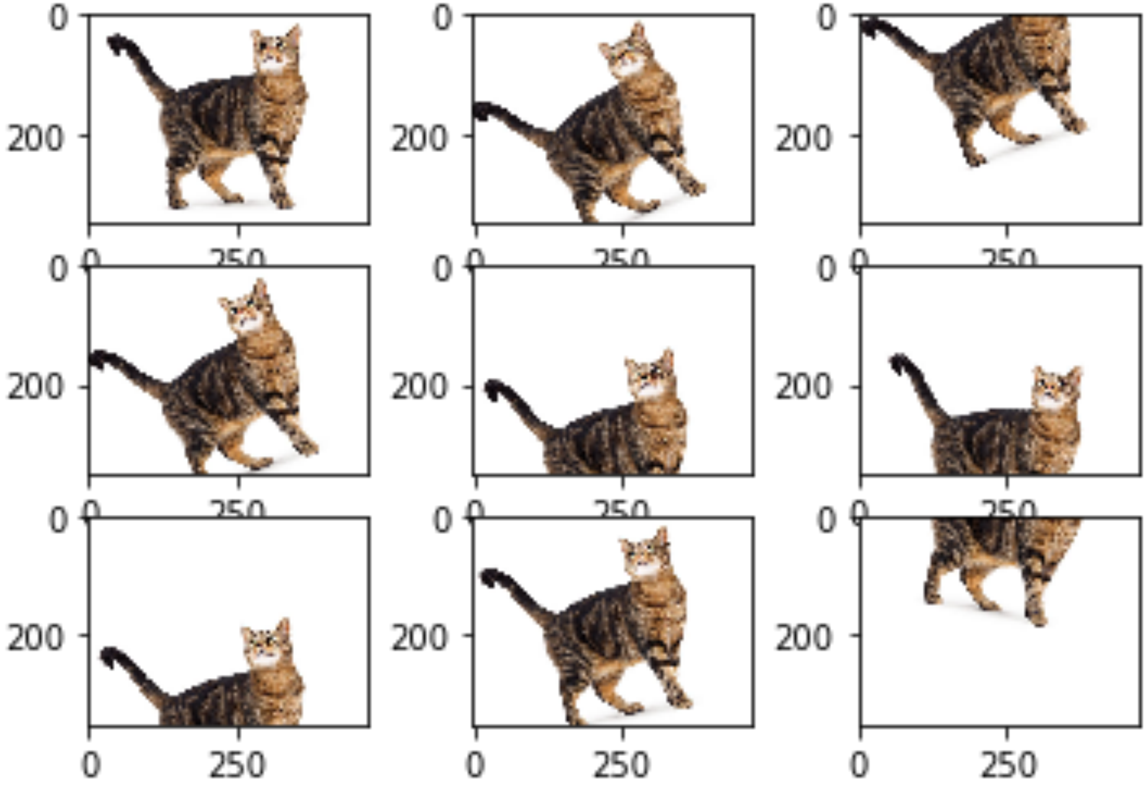
«We show that the dropout regularizer is first-order equivalent to an L2 regularizer applied after scaling the features by an estimate of the inverse diagonal Fisher information matrix»

Early stopping

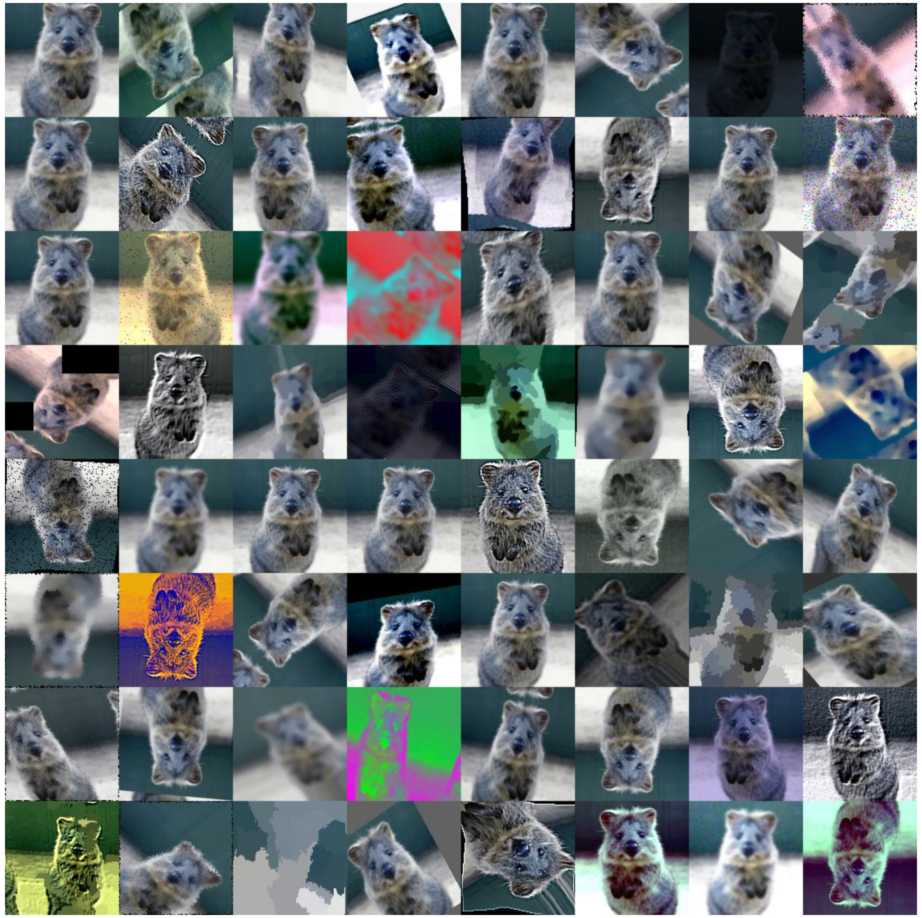
- Select number of optimization steps based on validation quality
- When the optimal number of steps is known, one can retrain the model both on training and validation data
- For a linear model with MSE loss and SGD optimizer, early stopping is equivalent to L_2 regularization



Data augmentation



Data augmentation



https://github.com/aleju/imgaug

Data augmentation for images

- Shift
- Zooming in/out
- Rotation
- Flip
- Distortion
- Shade

Data augmentation

- You can even use style transfer!

Data augmentation for audio

- Time stretching
- Pitch shifting
- Dynamic Range Compression
- Background noise
- …

Data augmentation

- Improves robustness of the model
- Useful for small samples
- Improves results on large datasets too

Conclusions

- Normalizations improve training both in terms speed and quality
- Regularizations (dropout, max-norm, L_2) prevent co-adaptation
- Data augmentation is useful to improve generalization
- Many other regularization techniques: multi-task learning, adversarial samples, sparsification etc.