Regularization

Victor Kitov
v.v.kitov@yandex.ru

Regularization

Aims of regularization:

- make underdetermined model determined[1]
- improve generalization (performance on train may decrease)
  - by encoding prior domain knowledge
  - by solving bias-variance trade-off
    - reduces variance
    - at the expense of small bias increase
  - this can useful when
    - model space is large and complex ($\downarrow$variance)
    - model space can approximate well the true model (bias is low)
    - example: decision trees, neural nets.

[1] examples: linear regression estimated with LS, logistic regression

Types of regularization

- add restrictions on parameters
- add penalty to objective function (soft restriction)
- ensemble learning

Soft regularization

- Modified loss:
$$\tilde{J}(\theta) = J(\theta) + \alpha R(\theta)$$
- Specifics of neural networks:
  - On layer h: $i^{h+1} = \beta_0 + \sum \beta_k o_k^h$
  - bias term β_0 is usually not included in regularization
    - there are comparatively few bias terms
    - model will stay unbiased
  - we may use different α_h for different layers $h = 1, 2, ...H$.

L_2 regularization («weight decay»)

$$\tilde{J}(w, X, Y) = \frac{\alpha}{2} w^T w + J(w, X, Y)$$

$$\nabla_w \tilde{J}(w, X, Y) = \alpha w + \nabla_w J(w, X, Y)$$

Stochastic gradient descent step:

$$w \leftarrow (1 - \varepsilon\alpha) w - \varepsilon \nabla_w J(w, X, Y)$$

Weights are shrunk towards zero.

Analysis of L_2-regularized solution

- Write $\tilde{J}(w)$ for Taylor 2nd order approximation around $w^* = \arg\min_w J(w)$:

$$\widehat{J}(w) = J(w^*) + \frac{1}{2}(w - w^*)^T H(w - w^*) + \frac{\alpha}{2} w^T w$$

  where $H = \nabla_w^2 J(w^*) \succeq 0$ and $\nabla_w J(w^*)^T (w - w^*) = 0$, because in minimum $\nabla_w J(w^*) = 0$.
- This expansion is precise for quadratic loss $J(w)$ (e.g. MSE).
- Minimum is achieved when $\nabla \widehat{J}(\tilde{w}) = 0$:

$$\begin{aligned} H(\tilde{w} - w^*) + \alpha \tilde{w} &= 0 \\ (H + \alpha I)\tilde{w} &= Hw^* \\ \tilde{w} &= (H + \alpha I)^{-1} Hw^* \end{aligned} \tag{1}$$

- When $\alpha = 0$ $\tilde{w} = w^*$.

Analysis of L_2-regularized solution

- $H = Q\Lambda Q^T$ (spectral decomposition), where
  - Q is orthonormal basis of eigenvectors
  - Λ - diagonal matrix with eigenvalues

Analysis of L_2-regularized solution

- $H = Q\Lambda Q^T$ (spectral decomposition), where
  - Q is orthonormal basis of eigenvectors
  - Λ - diagonal matrix with eigenvalues
- Substituting spectral decomposition into(1), we obtain:

$$\begin{aligned}\tilde{w} &= (Q\Lambda Q^T + \alpha I)^{-1} Q\Lambda Q^T w^* \\ &= \left[Q(\Lambda + \alpha I)Q^T\right]^{-1} Q\Lambda Q^T w^* \\ &= Q(\Lambda + \alpha I)^{-1}\Lambda Q^T w^*\end{aligned}$$

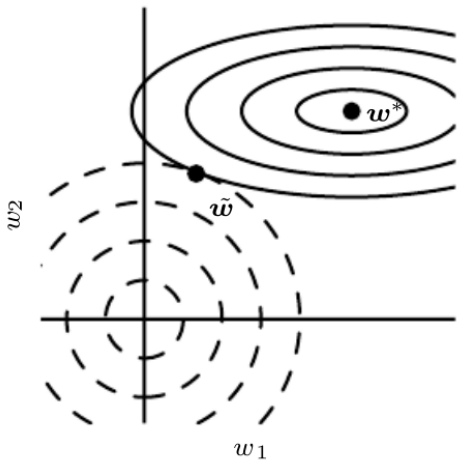
Analysis of L_2-regularized solution

- $H = Q\Lambda Q^T$ (spectral decomposition), where
  - Q is orthonormal basis of eigenvectors
  - Λ - diagonal matrix with eigenvalues
- Substituting spectral decomposition into(1), we obtain:

$$\begin{aligned}\tilde{w} &= (Q\Lambda Q^T + \alpha I)^{-1} Q\Lambda Q^T w^* \\ &= \left[Q(\Lambda + \alpha I)Q^T\right]^{-1} Q\Lambda Q^T w^* \\ &= Q(\Lambda + \alpha I)^{-1}\Lambda Q^T w^*\end{aligned}$$

- $\tilde{w}$ is obtained by rescaling w^* along the eigenvectors.
  - along i-th eigenvector rescaling factor is $\frac{\lambda_i}{\lambda_i+\alpha}$
  - rescaling effect is
    - high for small λ_i
    - insignificant for large λ_i

Illustration of L_2 regularization effect



- Notation
  - Solid: iso-lines of $J(w)$
  - Dashed: iso-lines of $\frac{\alpha}{2} w^T w$
- $\tilde{w}$ - equlibrium point
- Eigenvectors of H:
  - $v_1 = [1, 0]$, λ_1 is small => $|w_1^* - \tilde{w}_1|$ - large
  - $v_2 = [0, 1]$, λ_2 large => $|w_2^* - \tilde{w}_2|$ - small

Linear regression with L_2 regularization

$$y = x^T w$$

$$\widehat{w} = \arg\min_w \sum_{n=1}^{N} \left(x_n^T w - y_n \right)^2 + \frac{\alpha}{2} w^T w$$

Solution:

$$\widehat{w} = \left(X^T X + \alpha I \right)^{-1} X^T Y$$

For centered features:
$X^T X \propto N cov[x, x]$, $X^T Y = N cov[x, y]$
L_2 regularization «adds» α variance to each feature.
this forces estimator to reduce weights (based on cov[x,y])

L_1 norm regularization

$$\tilde{J}(w) = J(w) + \alpha \|w\|_1$$

$$\nabla \tilde{J}(w) = \nabla J(w) + \alpha \, \text{sign}(w)$$

When $\alpha > \sup_{w_i} |\nabla J(w)|$ SGD will force $w_i \to 0$.

Effect of L_1 regularization on solution

- To get analytical solution need to assume that Hessian is diagonal.
- Consider 2nd order Taylor approximation to $\widehat{J}(w)$:

$$\widehat{J}(w) = J(w^*) + \sum_i \left[\frac{1}{2} H_{i,i} (w_i - w_i^*)^2 + \alpha |w_i| \right]$$

- Solution[2]:

$$w_i = \text{sign}(w_i^*) \max \left\{ \left| w_i^* - \frac{\alpha}{H_{i,i}}, 0 \right| \right\}$$

- Analysis:
  - solution is sparse (many w_i may be 0)
  - shift in weights is smaller along directions with high $H_{i,i}$
  - $\frac{\alpha}{H_{i,i}} > w_i^*$: regularizer dominates $J(w)$ improvements.

[2] L_2 regularized solution would be here $w_i = \frac{H_{i,i}}{H_{i,i}+\alpha} w_i^*$

L_1 regularizer: feature selection

- $||w||_1$ regularizer will do feature selection.
- Consider

$$\tilde{J}(w) = J(w) + \alpha \sum_{d=1}^{D} |w_d|$$

- if $\alpha > \sup_w \left| \frac{\partial J(w)}{\partial w_i} \right|$, then it becomes optimal to set $w_i = 0$
- For higher α more weights will become zeroes.

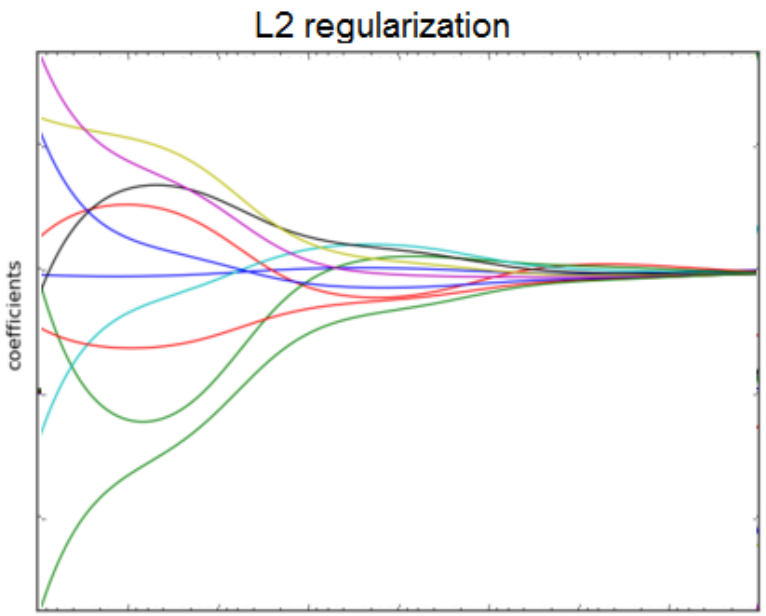
L_2 regularizer: no feature selection

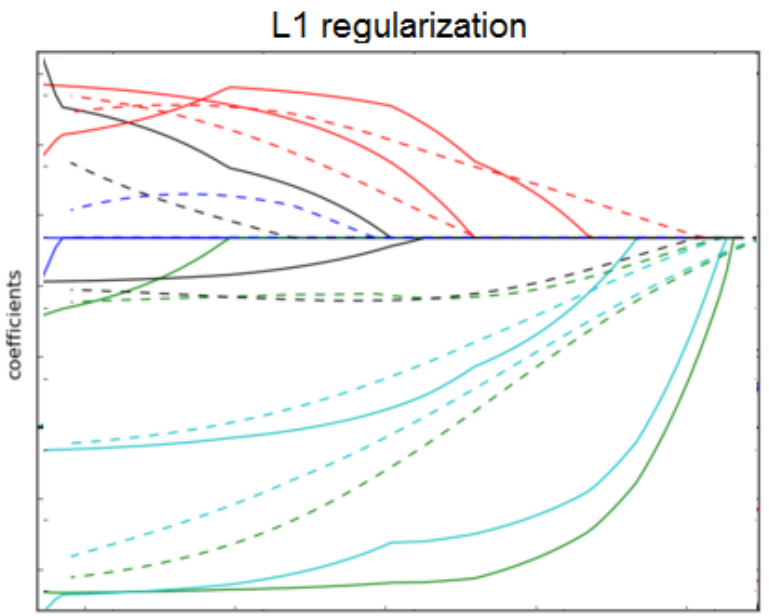
- Consider $R(w) = \frac{\alpha}{2} \|w\|_2^2 = \frac{\alpha}{2} \sum_d w_d^2$

$$\tilde{J}(w) = J(w) + \frac{\alpha}{2} \sum_{d=1}^{D} w_d^2$$

- $\frac{\partial R(w)}{\partial w_i} = \alpha w_i \to 0$ when $w_i \to 0$.

Illustration





Constrained optimization

$$\tilde{J}(\theta) = J(\theta) + \alpha R(\theta) \to \min_{\theta}$$

is equivalent to constrained maximization task for some $\gamma = \gamma(\alpha)$:

$$\begin{cases} J(\theta) \to \min_{\theta} \\ R(\theta) \leq \gamma \end{cases} \tag{2}$$

$\alpha \downarrow \Longleftrightarrow \gamma \uparrow$
To solve (2) repeat:
$\theta \leftarrow \theta - \varepsilon \nabla J(\theta)$ (or any other optimization update)
project θ onto region $\{\theta : R(\theta) \leq \gamma\}$

When to use constrained optimization

- Penalty addition may force algorithm get stuck in local optima around zero:
  - causing «dead units» with very small weights
  - inefficient local solution
- Constrained maximization has no such problem
- Constrained maximization: more stable
  - weights cannot take arbitrary values
  - may use higher learning rate!

Constrained optimization

- We can impose constraints on:
  - all weights
  - all weights within each layer
  - all incoming weights to each neuron
- Bias weights are usually not constrained.

Dataset augmentation

- More data - more accurate model.
- Using known invariant transformations - can generate more data.
- Example for image classification:
  - translation
  - scaling
  - reflection
    - counterexample: b->d
  - rotation
    - not big, otherwise 6->9, p->d
  - cropping
  - adding small random noise

Adding noise

- Add noise to inputs
  - solution becomes robust to input noise
- Add noise to hidden unit inputs
  - this is dataset augmentation with different levels of abstraction
- Add noise to weights
  - pushes weights to «plateu» regions where small weight changes do not affect output

Add noise to gradient[3]

$$\nabla J(\theta) \leftarrow \nabla J(\theta) + N(0, \sigma_t)$$

Recommended schedule:

$$\sigma_t = \frac{\eta}{(1+t)^\gamma}$$

where $\eta \in \{0.01, 0.3, 1.0\}$, $\gamma = 0.55$.

Improvements obtained:

- for networks with poor initialization (all zeroes)
- for very deep networks
- for memory networks

[3] Neelakantan, Arvind et al. Adding Gradient Noise Improves Learning for Very Deep Networks. 2015.

Add noise to outputs

- When incorrect labels present - overfitting.
- Instead of sampling objects with modified outputs we can
- For (x_n, y_n) replace hard targets with soft targets:

	hard target	soft target
$y = 1$	0	$\frac{\varepsilon}{C}$
$\dots$		
$y = y_n - 1$	0	$\frac{\varepsilon}{C}$
$y = y_n$	1	$1 - \frac{C-1}{C}\varepsilon$
$y = y_n + 1$	0	$\frac{\varepsilon}{C}$
$\dots$		
$y = C$	0	$\frac{\varepsilon}{C}$

- Smoothed likelihood:

$$\prod_{n=1}^{N} \prod_{y \neq y_n} p(y|x_n)^{\frac{\varepsilon}{C}} p(y_y|x_n)^{1-\frac{C-1}{C}\varepsilon} \to \max_{\theta}$$

Semi-supervised learning

- In semi-supervised learning we use:
  - labelled data $(x_1, y_1), ...(x_N, y_N)$
  - unlabelled data $x_{N+1}, ...x_{N+M}$.
- Motivation:
  - labelling is expensive
  - N is small and $M \gg N$.
  - $p(x)$ and $p(y|x)$ have shared parametrization.

Semi-supervised learning - neural nets[4]

$$\mathcal{L}_{hybrid}(X,Y) = \mathcal{L}_{disc}(X,Y) + \gamma \mathcal{L}_{unsup}(X)$$

where

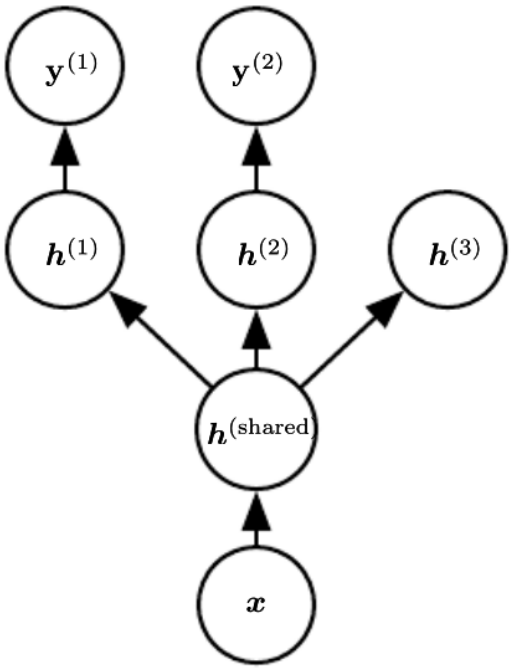
- $\mathcal{L}_{disc}(X,Y) = \sum_{n=1}^{N} \ln p(y_n|x_n)$ - discriminative log-likelihood
- $\mathcal{L}_{unsup}(X,Y) = \sum_{n=1}^{N+M} \ln p(x_n)$ - unsupervised log-likelihood
- γ - trade-off hyperparameter (tuned on validation set)

Results:

- In article Bolzmann machines were used
- Significant reduction of error-rate on MNIST, 20 newsgroups.

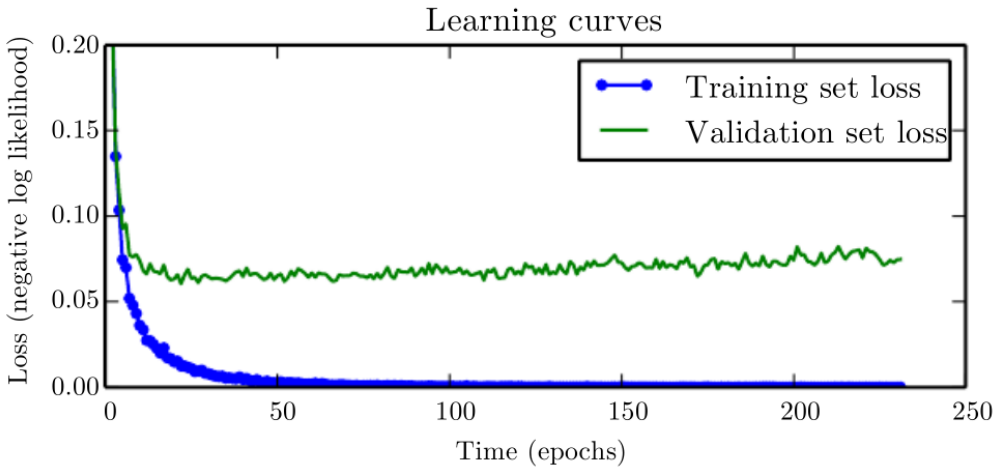
[4] Larochelle, H. and Bengio, Y. (2008). Classification using discriminative restricted Boltzmann machines. In ICML'2008.

Multi-task learning



- Applicable when several tasks have shared factors.
- Statistical benefit - more accurate estimation

Early stopping



- Is similar to weight decay. Needs separate validation set.
- Parameters:
  - period of steps when validation performance is reevaluated
    - smaller period - more accurate, but more computationally intensive
  - after how many «bad» evaluations (quality didn't improve) set to stop
    - if small - may stop too early due to noisy performance estimation.

Early stopping - utilizing validation set

Early stopping returned:

- optimal number of steps i^*
- optimal parameters θ^*
- performance on validation P_{val} and train P_{train}

Two approaches how to utilize validation set:

1. reinitialize NN and run i^* steps using training+validation set.
   - use the same number of passes through objects or dataset (epochs)?
2. continue training NN with initialization θ^* on the validation set until quality on validation reaches P_{train}.
   - may not reach

Sparse representation

- Suppose
  - θ is a vector of estimated model parameters
  - h is inner representation:
- Optimized criterion in sparse representation becomes:

$$\tilde{J}(\theta) = J(\theta) + \alpha R(h(\theta)) \rightarrow \min_{\theta}$$

where $R(h)$ is sparsity provoking prior such as $R(h) = \sum_i |h_i|$.

Example of sparse representation: sparse coding

- Definitions:
  - $X \in \mathbb{R}^{N \times D}$ - design matrix
  - $D \in \mathbb{R}$ - dictionary matrix (rows-code words)
  - $W \in \mathbb{R}$ - representation matrix (rows-object representations)
- Sparse coding is found with optimization task:

$$\|X - WD\|_2^2 + \|W\|_1 \to \min_{D,W} \tag{3}$$

where $\|A\|_2^2 := \sum_{i,j} a_{i,j}^2$ and $\|A\|_1 := \sum_{i,j} |a_{i,j}|$.

Example of sparse representation: sparse coding

- Definitions:
  - $X \in \mathbb{R}^{N \times D}$ - design matrix
  - $D \in \mathbb{R}$ - dictionary matrix (rows-code words)
  - $W \in \mathbb{R}$ - representation matrix (rows-object representations)
- Sparse coding is found with optimization task:

$$\|X - WD\|_2^2 + \|W\|_1 \to \min_{D,W} \tag{3}$$

  where $\|A\|_2^2 := \sum_{i,j} a_{i,j}^2$ and $\|A\|_1 := \sum_{i,j} |a_{i,j}|$.
- Task (3) is not convex with respect to D, W but is convex with respect to D or W only (holding another matrix fixed).

Sparse coding: algorithm

INPUT: design matrix X

initialize D randomly
while stop condition not met:
$W = \arg\min_W \|X - WD\|_2^2 + \|W\|_1$
$D = \arg\min_D \|X - WD\|_2^2 + \|W\|_1$

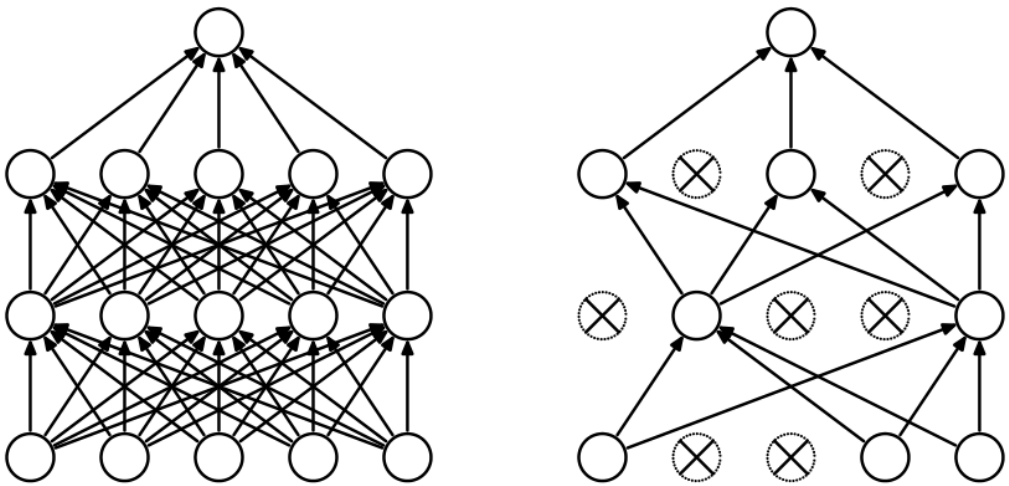
OUTPUT: dictionary D and sparse representation W

Table of Contents

Dropout idea

Each node in the neural network is removed with probability $1 - p$ independently from decisions about other nodes:

Comparison neural net without/with dropout



(a) Standard Neural Net

(b) After applying dropout.

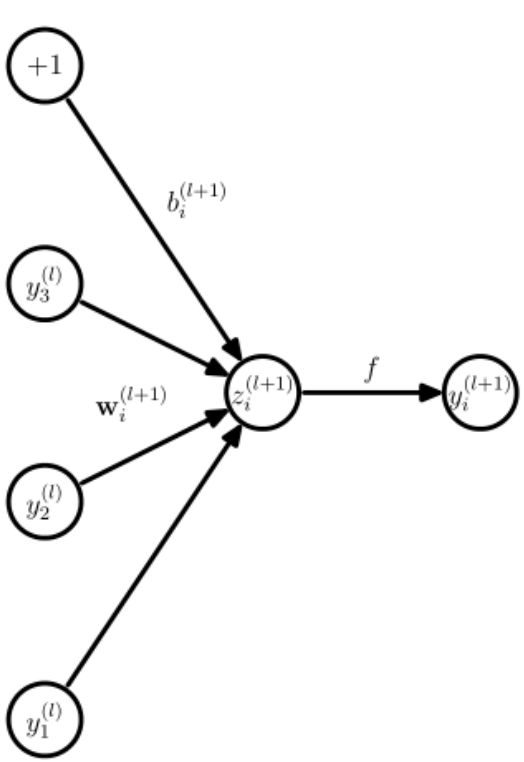
- Output layer nodes are never removed.
- Recommended parameters:
  - $p = 0.5$ for inner layer nodes
  - $p = 0.8$ for input layer nodes (feature subsampling)

Dropout motivation

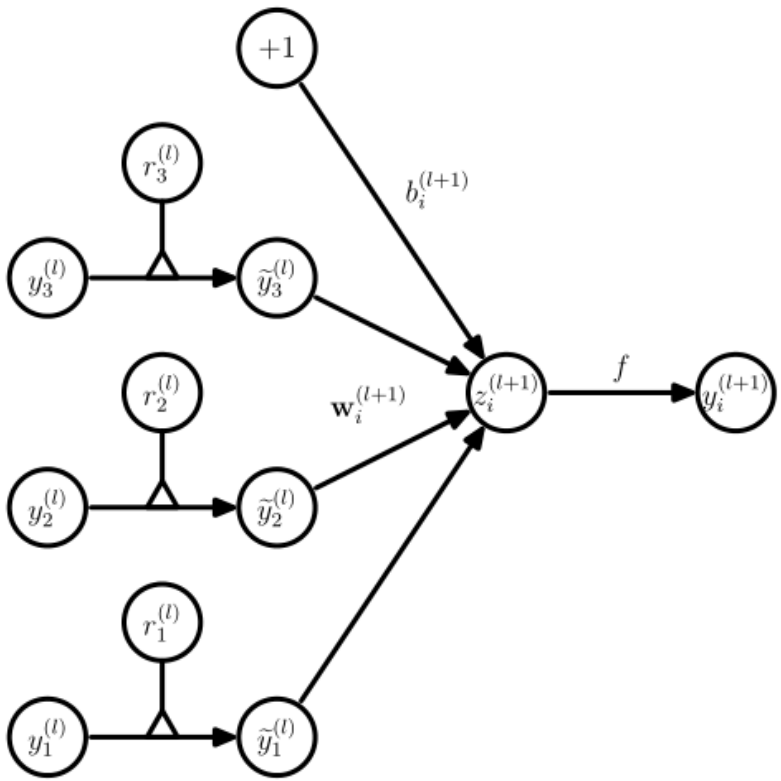
- Motivation from genetic theory of evolution:
  - sexual reproduction involves taking half the genes of one parent and half of the other.
  - best fit genes get mixed with 0.5 probabilities
  - best genes should learn "by themselves", not relying on complex outer gene structure
    - less ovefitting
- In dropout network:
  - nodes rely less on outputs of other nodes
  - try more to learn something by themselves
  - behave in a more robust way
  - resulting network becomes less overfitted.

Dropout algorithm

Comparison of usual and dropout network for one node



(a) Standard network



(b) Dropout network

Definitions

Define:

- $f(x)$ - an activation function.
- y^l - vector of outputs at layer l
- z^l - vector of inputs to layer l
- $a * b$ defines element-wise product of elements.
- L - number of layers in neural network
- $y^{(0)} = x$ - input feature vector
- $Bernoulli(p)$ returns a vector of independent Bernoulli random variables with parameter p.

Forward propagation algorithm

We need to repeat forward propagation recurrently for $l = 0, 1, ...L-1$.

1. Usual feed-forward neural network:

$$z_i^{(l+1)} = w_i^{(l+1)} y^l + b_i^{(l+1)}$$
$$y_i^{(l+1)} = f(z_i^{(l+1)})$$

2. Feed-forward network with dropout:

$$r_j^{(l)} \sim Bernoulli(p)$$
$$\tilde{y}^l = r^{(l)} * y^{(l)}$$
$$z_i^{(l+1)} = w_i^{(l+1)} \tilde{y}^l + b_i^{(l+1)}$$
$$y_i^{(l+1)} = f(z_i^{(l+1)})$$

Application of dropout

- **Learning**
  - while weights not converge:
    1. sample random subnetwork ("thinned network") with dropout
    2. apply one step of stochastic gradient descent to thinned network

Comment: due to weights sharing across all thinned networks the number of parameters is the same as in original network.

Application of dropout

- **Prediction**
  - use full networks with all nodes, but multiply each weight by p[5].
  - such scaling will yield the same output as average thinned network.

[5] precise for networks without non-linearities. With non-linearities Monte-Carlo sampling may work better.

Complexity

- $O(W)$ operations during each step to generate binary mask.
- $O(W)$ memory to store the mask
- Complexity of forward and backward pass - the same
- BUT: total number of steps until convergence may increase
  - dropout shrinks model capacity
  - to offset this, need to increase the network, make more optimization steps

Modifications

- Additive Gaussian noise:
  - $h_i \leftarrow h_i * N(1,1)$
  - at test time: no scaling needed
- Dropconnect

Conclusion

- Dropout behaves similar to generating 2^W networks and taking weighted average of their predictions (W is the number of weights in the original neural network).
- Dropout performes intelligent high-level information destruction
  - model becomes more robust (at high levels of abstraction as well)
- Properties:
  - number of parameters is the same
  - training complexity is reduced
  - complexity of prediction is the same
- Dropout provides accuracy improvement in many domains.
- More details in: *"Dropout: A Simple Way to Prevent Neural Networks from Overfitting". Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. Journal of Machine Learning Research 15 (2014) 1929-1958.*

Table of Contents

Batch normalization[6]

- Learning by minibatches
  - more accurate gradient
  - faster by using parallelizm
- Problems of deep networks:
  - all parameters change simultaneously
  - this change gets amplified in deep networks
  - for each neuron its input distribution changes
  - neuron such as sigmoid may saturate

[6] Sergey Ioffe, Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2015.

Batch normalization

- Standardizes outputs
- Gradient becomes scale invariant
- Can ensure staying away from neuron saturation regions
- May use higher learning rates
- Approach has beaten state-of-the-art ImageNet model (inception network)

Batch normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Normalization propagation

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

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

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

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

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

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

Batch normalization: algorithm

Input: Network N with trainable parameters Θ;
subset of activations $\{x^{(k)}\}_{k=1}^{K}$

Output: Batch-normalized network for inference, $N_{\text{BN}}^{\text{inf}}$

1: $N_{\text{BN}}^{\text{tr}} \leftarrow N$ // Training BN network
2: **for** $k = 1 \ldots K$ **do**
3: Add transformation $y^{(k)} = \text{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\text{BN}}^{\text{tr}}$ (Alg. 1)
4: Modify each layer in $N_{\text{BN}}^{\text{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
5: **end for**
6: Train $N_{\text{BN}}^{\text{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
7: $N_{\text{BN}}^{\text{inf}} \leftarrow N_{\text{BN}}^{\text{tr}}$ // Inference BN network with frozen // parameters

Batch normalization: algorithm

8: **for** $k = 1 \dots K$ **do**

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

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

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

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

11: In $N_{\mathrm{BN}}^{\mathrm{inf}}$, replace the transform $y = \mathrm{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$

12: **end for**

Algorithm 2: Training a Batch-Normalized Network

-

Weights initialization

- random with distribution $w_i \sim F(0, \sigma^2)$, having
  - zero mean
  - varaince equal to $\frac{1}{n_{in}}$ or $\frac{2}{n_{in}+n_{out}}$ where
    - n_{in} is the number of incoming connections for neuron i.
    - n_{out} is the number of outgoing connections for neuron i.
- Unsupervised pretraining
  - obtain initial weights from solving data-representation problem with autoencoder