UNIVERSITÄT BERN

Deep Feedforward Networks - Regularization

Paolo Favaro

Contents

- Regularization in Feedforward Neural Networks
  - Parameters, optimization, dataset augmentation, I/O noise, semi-supervised learning, multi-task learning, early stopping, parameter sharing and sparsity, dropout, adversarial training
- Based on **Chapter 7** of Deep Learning by Goodfellow, Bengio, Courville

Note

Regularization

- A central problem in ML is **generalization**: How do we design an algorithm that can perform well not only on training data but also on new data?
- Regularization aims at reducing the generalization error of an algorithm

Note

Generalization

- Problems with generalization (see also Machine Learning Review slides)
  - Underfitting (large bias but low variance)
  - Overfitting (small bias but high variance)
- Neural networks typically are in the second case and regularization aims at reducing variance

Note

Regularization

- Strategies
  - Constrain model (e.g., restrict model family or parameter space)
  - Add terms to loss function (equivalent to soft constraints to the model) — can encode priors
  - Ensemble methods (combine multiple hypotheses)

Often the best option in deep learning is a large model with good regularization.

Regularization

- Strategies
  - Constrain model (e.g., restrict model family or parameter space)
  - Add terms to loss function (equivalent to soft constraints to the model) — can encode priors
  - Ensemble methods (combine multiple hypotheses)

Regularization

- Regularization has been widely used in optimization and in particular in **inverse problems**
- Inverse problems amount to a gain of information, which can be provided in the form of **a prior**
- Tikhonov defined regularization as the creation of a **parametric family of solutions**

With Tikhonov the parameter is such that when it is zero, then the solution should be the desired one (when there is no noise) and when the parameter is non zero, then the solution would be an approximation of the ideal solution.

The Bayesian Framework

Note

The Bayesian Framework

- A priori information can be given through a probability distribution

The Bayesian Framework

- A priori information can be given through a probability distribution
- Formalizes previous experience or additional assumptions

A Posteriori Density Function

- Suppose that the joint pdf $p(\theta, z)$ is given

Note

A Posteriori Density Function

- Suppose that the joint pdf $p(\theta, z)$ is given
- Obtain the **a priori** density function via marginalization

$$p(\theta) = \int p(\theta, z) dz$$

A Posteriori Density Function

- Suppose that the joint pdf $p(\theta, z)$ is given
- Obtain the **a priori** density function via marginalization

$$p(\theta) = \int p(\theta, z) dz$$

- Relate probabilities via the product rule

$$p(\theta, z) = p(z|\theta)p(\theta)$$

A Posteriori Density Function

- Suppose that the joint pdf $p(\theta, z)$ is given
- Obtain the **a priori** density function via marginalization

$$p(\theta) = \int p(\theta, z) dz$$

- Relate probabilities via the product rule

$$p(\theta, z) = p(z|\theta)p(\theta)$$

- The **posterior** is a conditional probability that describes the data given the target

$$p(\theta|z) = \frac{p(\theta, z)}{p(z)} = \frac{p(z|\theta)p(\theta)}{\int p(z|w)p(w)dw}$$

Note

Bayesian Estimates

- The posterior provides all the information we can have about the object
- The most common point estimates are
  - Conditional mean

$$\tilde{\theta} = \mathrm{E}[\theta|z] = \int \theta p(\theta|z) d\theta$$

  - **Maximum a posteriori**

$$\tilde{\theta} = \arg\max_{\theta} p(\theta|z)$$

These point estimates are obtained by minimizing Bayes risk with two different loss functions (the L2 norm in the first case and the 0-1 loss in the second case). Let us consider Maximum a Posteriori.

Maximum a Posteriori

- Recall the previous formulation

$$\tilde{\theta} = \arg\max_{\theta} p(\theta|z)$$

we can rewrite it as

$$\tilde{\theta} = \arg\max_{\theta} p(z|\theta)p(\theta)$$

or, equivalently, as

$$\tilde{\theta} = \arg\min_{\theta} -\log p(z|\theta) - \log p(\theta)$$

We now want to show how MAP can be written in the classic regularization form

Maximum a Posteriori

- Recall the MAP formulation

$$\tilde{\theta} = \arg\min_{\theta} -\log p(z|\theta) - \log p(\theta)$$

- We can choose, for example, the model and prior

$$-\log p(z|\theta) = \frac{|z-\theta|^2}{2\sigma^2} + \text{const}$$

$$-\log p(\theta) = |\theta|^2 + \text{const}$$

and obtain the following optimization problem

$$\tilde{\theta} = \arg\min_{\theta} |z-\theta|^2 + \lambda|\theta|^2$$

The model and the prior are then combined in an additive manner and lead to the classic optimization form. We typically identify the first term as the data term and the second one as the regularization term.
We can also consider the data term as some initial loss function to which we add an additive regularizer. This view is what we use in the next slides.

Parameter Regularization

- Let us consider an objective function in the form of

$$J(\theta; X, y)$$

  where θ are the neural network parameters

- The regularized objective is then

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta)$$

  where $\Omega(\theta)$ is a norm parameter penalty and $\alpha \in [0, \infty)$

The hyperparameter \alpha determines the amount of regularization (analogous to \lambda in the previous slide).
The norm penalty typically acts only on the linear component and ignores the biases (i.e., in Wx+c it acts only on W and not on c). Leaving the biases out of the regularization does not cause too much overfitting and avoids the risk of underfitting. To avoid making the notation too heavy, we keep using \theta even though we might refer only to a subset of the parameters.

L2 Regularization

- Let us consider $p(\theta) = \mathcal{N}(\theta; 0, 1/\sqrt{\alpha})$ then

$$\Omega(\theta) = \frac{1}{2}|\theta|^2$$

- This choice is also referred to as weight decay, ridge regression, or Tikhonov regularization

- The gradient descent of the regularized loss is then

$$\theta \leftarrow (1 - \epsilon\alpha)\theta - \epsilon\nabla_\theta J(\theta; X, y)$$

The L2 regularization term introduces a reweighing of the previous value of the parameters.

L² Regularization

- Let us assume that locally we have

$$J(\theta; X, y) \simeq J(\theta^*; X, y) + \frac{1}{2}(\theta - \theta^*)^\top H(\theta - \theta^*)$$

where $\theta^* = \arg\min_\theta J(\theta; X, y)$

then we impose $\nabla \tilde{J}(\tilde{\theta}; X, y) \simeq \alpha\tilde{\theta} + H(\tilde{\theta} - \theta^*) = 0$

and obtain

$$\begin{aligned}\tilde{\theta} &= (H + \alpha I)^{-1} H\theta^* \\ &= (Q\Lambda Q^\top + \alpha I)^{-1} Q\Lambda Q^\top \theta^* \\ &= Q(\Lambda + \alpha I)^{-1}\Lambda Q^\top \theta^*\end{aligned}$$

SVD

$H = Q\Lambda Q^\top$

$\frac{\lambda_i}{\lambda_i + \alpha}$

If we approximate the original cost function around its optimum with a quadratic function (so the Hessian H is positive definite), we can better see the effects of the L2 regularization. Notice that the regularizing term sets strong singular values \lambda_i of the Hessian to 1 and sends the small singular values to zero.

L² Regularization

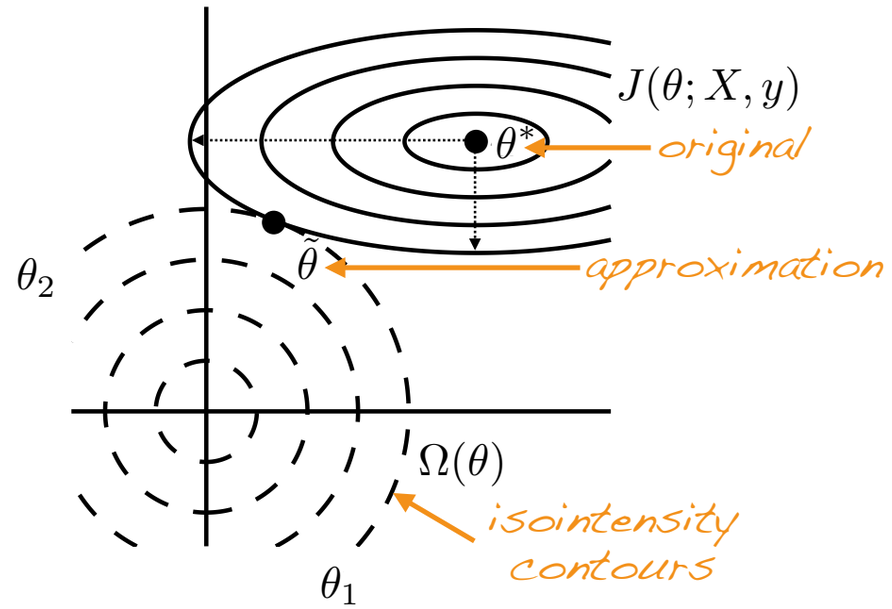


Illustration of the L2 regularization.
The Hessian has a small singular value (small curvature) along the abscissa coordinate.
The Hessian has a large singular value (high curvature) along the ordinate coordinate.
Regularization will affect more the abscissa of w* (indeed it shrinks it more than the ordinate of w*).

L1 Regularization

- Let us consider $p(\theta) = \text{Laplace}(\theta; 0, 1/\alpha)$ then

$$\Omega(\theta) = |\theta|_1$$

- This choice has been found to favor sparsity, i.e., it tends to set a number of the parameters to zero

- The corresponding gradient of the regularized objective function is then

$$\nabla_\theta \tilde{J}(\theta; X, y) = \alpha \ \text{sign}(\theta) + \nabla_\theta J(\theta; X, y)$$

The L1 regularization term introduces a constant penalty that depends on the sign of the parameters (per entry).

L[1] Regularization

- Let us use again the same approximation as before

$$J(\theta; X, y) \simeq J(\theta^*; X, y) + \frac{1}{2}(\theta - \theta^*)^\top H(\theta - \theta^*)$$

but where the Hessian is a diagonal matrix (h_i).
We obtain the following equation for each element

$$\alpha \, \mathrm{sign}(\theta_i) + h_i(\theta_i - \theta_i^*) = 0$$

which gives the **shrinkage-thresholding** function

$$\theta_i = \mathrm{sign}(\theta_i^*) \max\left\{0, |\theta_i^*| - \frac{\alpha}{h_i}\right\}$$

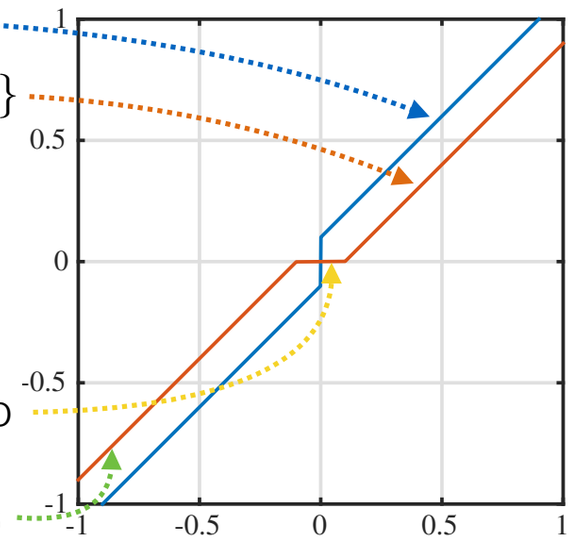
Note

L¹ Regularization

$$\phi(z) = z + \lambda \operatorname{sign}(z)$$

$$\phi^{-1}(z) = \operatorname{sign}(z) \max\{0, |z| - \lambda\}$$

- The inverse function is the shrinkage-thresholding operator
- It maps small values to zero
- It scales down large values



The shrinkage-thresholding function is the inverse of \phi(\theta) = \lambda \sign(\theta)+\theta.
Notice that the gap in both plots is equal to 2\lambda and the slope is 1.
To relate the functions above to the previous slide use
\lambda = \alpha/h

Constrained Optimization

- Hard constraints can be formulated as highly selective priors (i.e., the probability of a set of values is set to zero)

- For example, this prior enforces that the parameters must lie on a hypersphere of radius ρ

$$p(\theta) = \frac{2\pi^{n/2}\rho^{n-1}}{\Gamma(n/2)}\delta(|\theta| - \rho)$$
$$\propto \delta(|\theta| - \rho)$$

\Gamma is the Gamma function.

Constrained Optimization

- Hard constraints imposed directly on the cost function, but might cause the optimization algorithm to get stuck in local minima
- An alternative: first, the optimization ignores the hard constraints and second, the solution is projected onto the constraint space
- For example, one can impose that the parameters lie on a hypersphere by normalizing the solution

$$\theta \leftarrow \frac{\theta}{|\theta|}$$

Note

Constrained Optimization

- Constrained optimization is also used to stabilize training
- One recommendation is not to normalize the whole weight matrix of a layer, but rather each column separately (corresponding to outputs)
- This prevents any single unit from having very large weights

Note

Under-Constrained Problems

- Problems may be under constrained
- This is the case, for example, when the number of samples is smaller than the dimensionality of the data
- In this case, regularization adds the “missing” equations

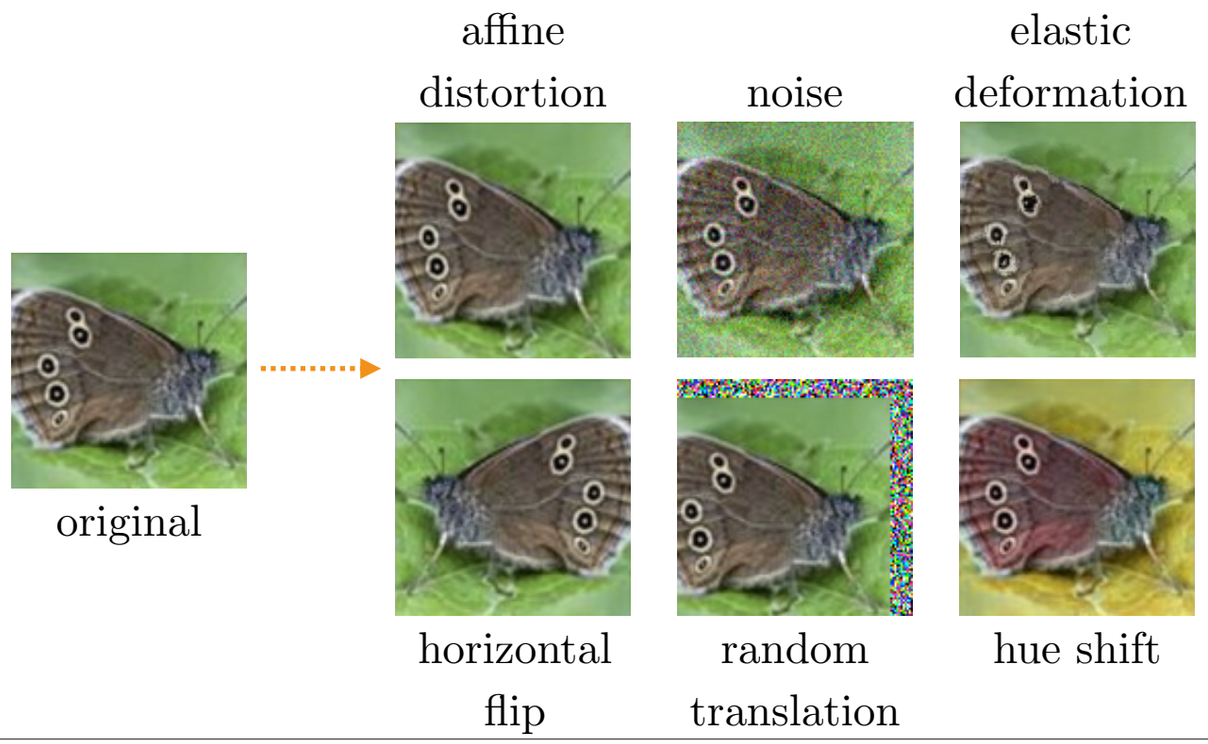
Note

Dataset Augmentation

- The best way to make the model generalize well is to train it on more data
- One way to augment our dataset is to apply a number of realistic transformations to the data we already have and create new synthetic samples, which share the same label
- This process of data manipulation is also called **jittering**

Note

Dataset Augmentation



These are all valid images of a butterfly. Many other transformations can be used (e.g., grayscale mapping, random cropping, contrast jittering). Some transformations are instead desirable but not possible. For example, 3D rotations. Approximations (e.g., homographies) can be used but might introduce artifacts.
Also, depending on the task, one can use jittering to purposefully create unrealistic images.
Note that some transformations may seem not useful, but they can instead help. For example, if the first layer is a convolutional one, random translations may seem unnecessary because the filters are shift-invariant. However, the shifted images help generalize the boundary assumptions and later non shift-invariant hidden layers.

Dataset Augmentation

- A different perspective is to use data augmentation to reduce the effect of artifacts of the data on the model training

- Example: ILSVRC12's chromatic aberration

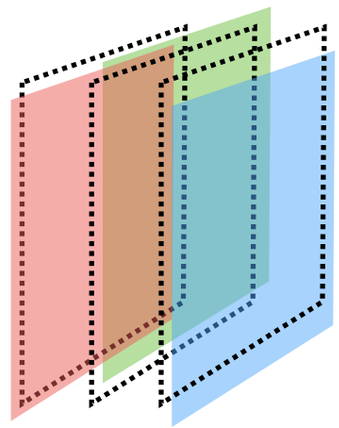
Another important aspect to keep in mind is that jittering can be used to reduce the chances that an NN learns trivial mappings. For example, the ImageNet datasets is known to be made of images with chromatic aberration. If the NN can use this artifact to solve a task, then it might not learn the correct mapping.

Chromatic Aberration

Chromatic aberration introduces a relative shift (or scaling or blur) between the color channels. This artifact changes relative to the image center.

Chromatic Aberration

- To reduce chromatic aberration generate new images by randomly and independently scaling and shifting (by very small amounts) the color channels of each image



- This will randomize chromatic aberration and not allow an NN to learn it

Note

Noise Robustness

- Apply noise to the input data at each iteration
- Apply noise to the inputs of the hidden units (Poole et al 2014)
- Dropout can be seen as multiplicative noise (see later slides)

Notice that noise in data augmentation is fixed on each image once for all.

Noise Robustness

- Apply noise to the weights
  - Model weights as random variables
  - Encourage stability of learned mapping (weights find minima with a flat neighborhood)

Note

Label Smoothing

- Labels might be wrong (remember: it is human annotation)
- Let us model noise in the labels
  - For example, $p(y) = (1-\epsilon)\hat{p}(y) + \epsilon\mathcal{U}[1, K]$

    labeling
  - **Label smoothing** replaces 0s and 1s with

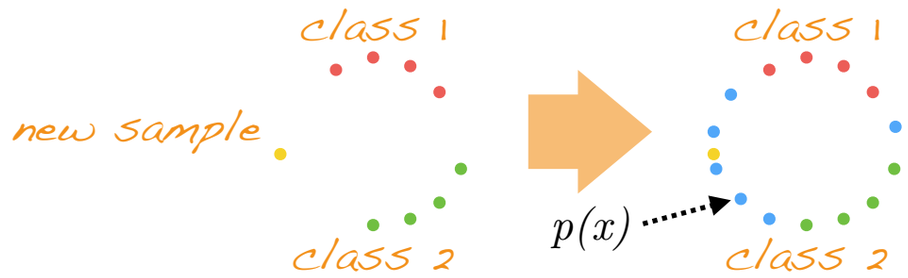
    $\frac{\epsilon}{K-1}$ and $1-\epsilon$ respectively

Where \hat p is the given labeling and U is the uniform distribution. p(y) is the regularized probability distribution. I would however use a different model, where the choice of the label index is correct (1-e)% of the time and a uniform choice e% of the time.

Semi-Supervised Learning

- Semi-supervised learning uses unlabeled samples from $p(x)$ and labeled samples from $p(x,y)$ to build $p(y|x)$ or directly predict y from x

- The probability density $p(x)$ can be seen as a prior on the input data



The projection of data and the classification (or regression) is easier if we have the data distribution p(x).

Semi-Supervised Learning

- Learn a representation so that samples from the same class have similar representations
- Then a linear classifier may achieve better generalization

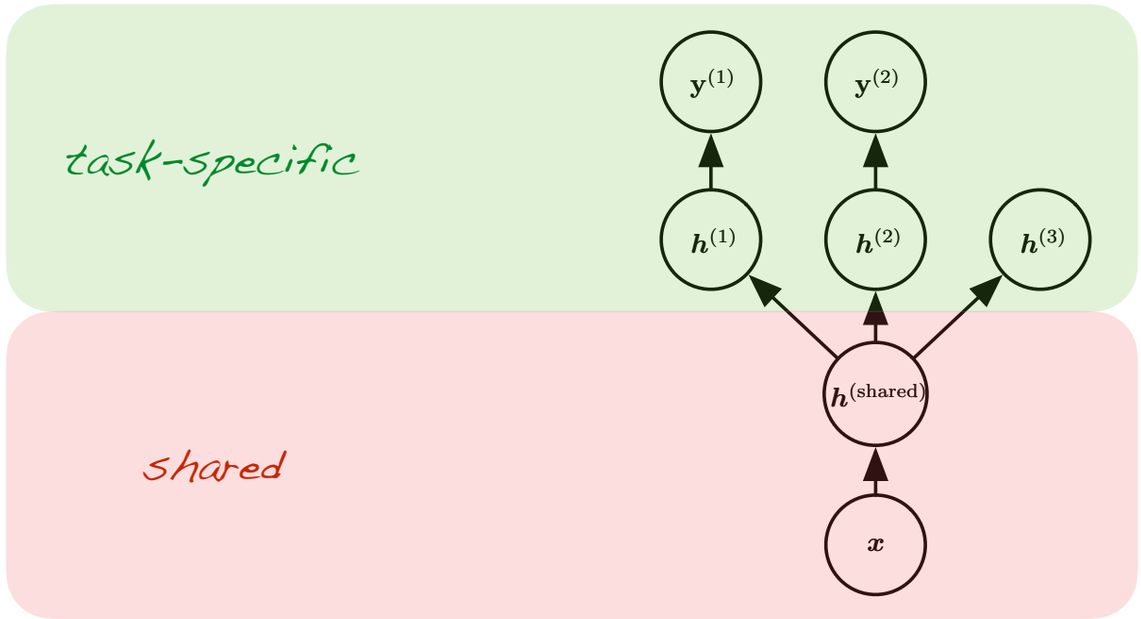
We refer to a Deep Learning context.
One classic example is to compute the PCA on the unlabeled samples and then to learn p(y|x) on the projected data (onto the subspace defined by the truncated PCA).

Multi-Task Learning

- Training on several tasks is a form of regularization as it can also improve generalization
- We assume that: 1) input variations can be explained by a **common pool of factors** and 2) tasks require only a **subset of these factors**
- The model is split into two parts:
  - One that is task-specific
  - Another that is shared across the tasks

Note

Multi-Task Learning



Note

Example

- Consider the tasks of object classification and single image 3D reconstruction

- The two tasks require common factors since the definition of an object relies on a 3D structure and vice versa

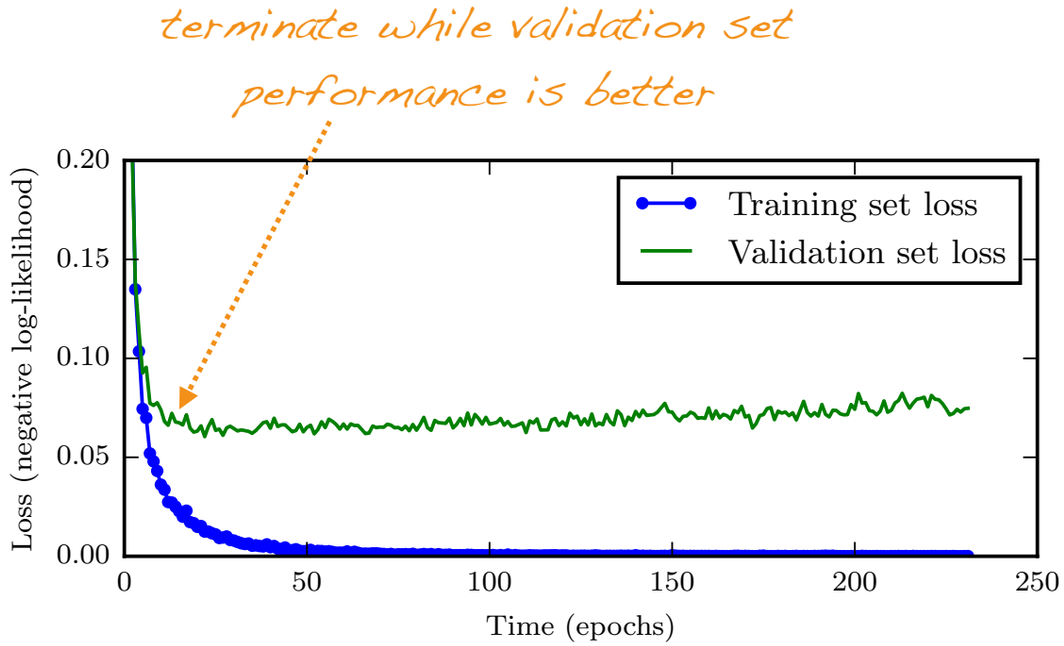
Note

Early Stopping

- Neural networks require iterative algorithms for training (typically a gradient descent-type)
- The larger the number of iterations and the lower the training error
- A technique to increase the generalization of the model is to **limit the number of iterations**

Note

Early Stopping



We return to the parameter setting at the point in time with the lowest validation set error (so hopefully this has also the lowest test set error). Every time the error on the validation set improves, we store a copy of the model parameters. When the training algorithm terminates, we return these parameters, rather than the latest parameters. This is a form of hyperparameter selection.

Most libraries in deep learning have mechanisms to save the parameters of the network periodically so that training can be picked up from where it was left after an interruption. The validation set test is also infrequent and typically done to monitor the performance of the training.

Early Stopping

- Since the validation set is not used for training, after early stopping one can either

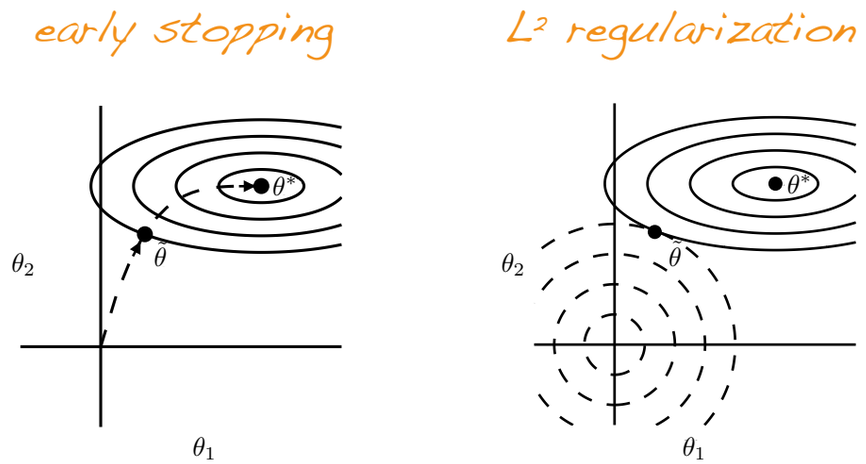
  1) retrain the network on all the data (training + validation sets) and then stop after the same number of steps of the early stopping or

  2) continue training the network on all the data (training + validation sets) and then stop when the loss on the validation set is below the loss on the training set (at the early stopping iteration time)

Both of these techniques however have no guarantee to succeed in further improving the training.

Early Stopping

- The regularizing effect of early stopping can be seen as a limitation to the space of parameters that can be explored (around the initial values)



Early stopping is analogous to previous parameter regularization (e.g., L2 regularization).
On the left we see the trajectory taken by SGD in early stopping. Instead of stopping at the optimal (for the training set loss, but not necessarily for the test set loss) value \theta*, it stops earlier at \tilde\theta.
This is a similar behavior to the case of L2 regularization (shown on the right) where the parameters find a tradeoff between the optimal value and the prior (around the origin in the illustration).

Early Stopping

- Comparison with L^2 regularization
- Consider a linear model and the local approx. loss

$$J(\theta) = J(\theta^*) + \frac{1}{2}(\theta - \theta^*)^\top H (\theta - \theta^*)$$

with a standard gradient descent

$$\begin{aligned}\theta^\tau &= \theta^{\tau-1} - \epsilon \nabla J(\theta^{\tau-1}) \\ &= \theta^{\tau-1} - \epsilon H \left(\theta^{\tau-1} - \theta^*\right)\end{aligned}$$

where $\nabla J(\theta) = H(\theta - \theta^*)$ is also a Taylor expansion

Note that H is the Hessian of J and \theta* is the local minimum of J.
Then H is also positive semidefinite (it is also always symmetric by definition).
The expression for the gradient is also a first order Taylor expansion where the constant term is zero (the gradient at the solution \theta* is zero).

Early Stopping

- With the singular value decomposition $H = Q\Lambda Q^\top$ and the assumption that initially $\theta^0 = 0$, we have

$$\theta^\tau - \theta^* = (I - \epsilon H)\left(\theta^{\tau-1} - \theta^*\right)$$
$$Q^\top \theta^\tau = (I - (I - \epsilon\Lambda)^\tau) Q^\top \theta^*$$

- Recall the optimal L2 regularization condition

$$\alpha\tilde{\theta} + H(\tilde{\theta} - \theta^*) = 0$$

which we can rewrite as

$$Q^\top \tilde{\theta} = (I - (\Lambda + \alpha I)^{-1}\alpha) Q^\top \theta^*$$

Note

Early Stopping

- We compare the two solutions

$$Q^\top \theta^\tau = (I - (I - \epsilon\Lambda)^\tau) Q^\top \theta^*$$

$$Q^\top \tilde{\theta} = (I - (\Lambda + \alpha I)^{-1}\alpha) Q^\top \theta^*$$

and find that $(1 - \epsilon\lambda_i)^\tau = \frac{\alpha}{\alpha + \lambda_i}$, which gives

$$\tau = \frac{\log \frac{\alpha}{\alpha+\lambda_i}}{\log(1 - \epsilon\lambda_i)} = -\frac{\log\left(1 + \frac{\lambda_i}{\alpha}\right)}{\log(1 - \epsilon\lambda_i)} \simeq -\frac{\frac{\lambda_i}{\alpha}}{-\epsilon\lambda_i} = \frac{1}{\epsilon\alpha}$$

if all singular values are sufficiently small

This shows that the number of training iterations \tau is inversely proportional to the L2 regularization parameter. The combination \epsilon\tau fully defines the inverse of the regularization parameter.

Parameter Sharing

- Parameter tying and sharing are ways to express parameter dependencies

- **Example #1**
  - Two models on classification (same classes, but different datasets)
  - We believe that the parameters of the two models should be similar
  - We can add the following penalty to the loss function

$$\Omega(\theta_A - \theta_B) = |\theta_A - \theta_B|^2$$

Parameter regularization penalizes parameters that deviate from a given reference (e.g., zero). In other cases, however, we might not know what the correct reference should be, but we might know interdependencies between parameters.

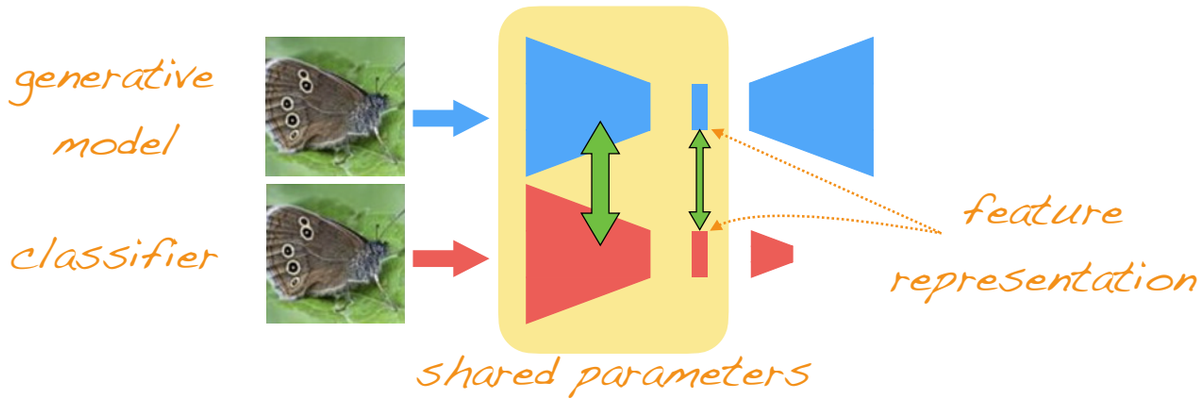
Parameter Sharing

- **Example #2**
  - One model on classification (supervised) and one on a generative model (unsupervised)
  - We believe that some parameters of the two models should be similar
  - We can also add the previous penalty (on a subset of the parameters) to the loss function

Note

Parameter Sharing

- **Example #3**
  - One model on classification (supervised) and one on a generative model (unsupervised)
  - Instead of a penalty (on a subset of the parameters) we can force them to be the same



The parameter sharing case uses the same variables in both networks and thus also saves memory.

Parameter Sharing

- **Example #4**
  - Convolutional Neural Networks use convolutional filters, which are translation-invariant
  - A photo of a butterfly remains the same regardless of any translation

  - If we think of these filters as pattern detectors, then we would like them to be shared across different image locations

Note

Sparse Representations

- Instead of imposing sparsity in the weights, we can impose sparsity in the **activations**

$$\begin{bmatrix} 18 \\ 5 \\ 15 \\ -9 \\ -3 \end{bmatrix} = \begin{bmatrix} 4 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & -1 & 0 & 3 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & -4 \\ 1 & 0 & 0 & 0 & -5 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ -2 \\ -5 \\ 1 \\ 4 \end{bmatrix}$$

$\boldsymbol{y} \in \mathbb{R}^m$ $\quad \boldsymbol{A} \in \mathbb{R}^{m \times n}$ $\quad \boldsymbol{x} \in \mathbb{R}^n$

weight sparsity

$$\begin{bmatrix} -14 \\ 1 \\ 19 \\ 2 \\ 23 \end{bmatrix} = \begin{bmatrix} 3 & -1 & 2 & -5 & 4 & 1 \\ 4 & 2 & -3 & -1 & 1 & 3 \\ -1 & 5 & 4 & 2 & -3 & -2 \\ 3 & 1 & 2 & -3 & 0 & -3 \\ -5 & 4 & -2 & 2 & -5 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ -3 \\ 0 \end{bmatrix}$$

$\boldsymbol{y} \in \mathbb{R}^m$ $\quad \boldsymbol{B} \in \mathbb{R}^{m \times n}$ $\quad \boldsymbol{h} \in \mathbb{R}^n$

representation sparsity

Note

Sparse Representations

- Sparsity in the activations can also be achieved via an L^1 penalty on the output of one layer

$$\Omega(h) = |h|_1$$

  or other sparsity inducing norms.

- Other approaches, such as orthogonal matching pursuit, might impose sparsity via greedy methods

Note

Bagging and Ensembles

- **Bagging** (bootstrap aggregating) is the strategy of combining the votes of several methods (trained on different training subsets) to reduce the generalization error
- The combination of several methods leads to an **ensemble**
- This strategy works when the different models make different errors on the test set (which is very often the case)

Note

Bagging and Ensembles

- Simplified analysis
  - Consider k regression models each with zero-mean normal error ϵ_i where

$$E[\epsilon_i^2] = v \qquad\qquad E[\epsilon_i \epsilon_j] = c$$

  - The error of the average prediction is

$$\frac{1}{k} \sum_{i=1}^{k} \epsilon_i$$

Note

Bagging and Ensembles

- Simplified analysis
  - The expected squared error of the ensemble predictor is

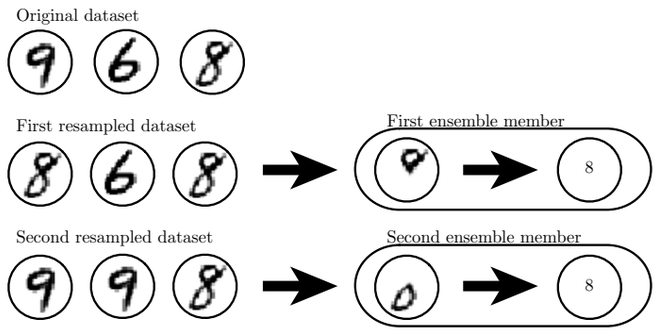
$$E\left[\left(\frac{1}{k}\sum_{i=1}^{k}\epsilon_i\right)^2\right] = \frac{1}{k^2}E\left[\sum_{i=1}^{k}\left(\epsilon_i^2 + \sum_{j\neq i}\epsilon_i\epsilon_j\right)^2\right]$$
$$= \frac{1}{k}v + \frac{k-1}{k}c$$

If the errors are all perfectly correlated (the models make exactly the same mistakes), then the variance is still v and the ensemble does not help at all. However, when the correlation c between the models is very low, then the averaging can reduce the variance of each model proportionally to the number of models.
In general the ensemble will not worsen the result. However, the computational cost will increase.

Bagging and Ensembles

- While one could combine different models, it is also possible to combine the same model several times
- One approach is to build k different datasets by sampling with replacement from the same dataset (i.e., replicas are possible and samples might be missing)

training of an 8 detector



Model i is trained on set i and because of the sample differences it will make different errors.
The first ensemble member learns that an 8 is a loop on the top.
The second ensemble member learns that an 8 is a loop on the bottom.
The combination is a more robust detector of an 8 than each single member.

Bagging and Ensembles

- In the case of deep learning it might be impractical to use bagging with many neural networks
- Training each network might take days or weeks
- Evaluation at test time might also become impractical

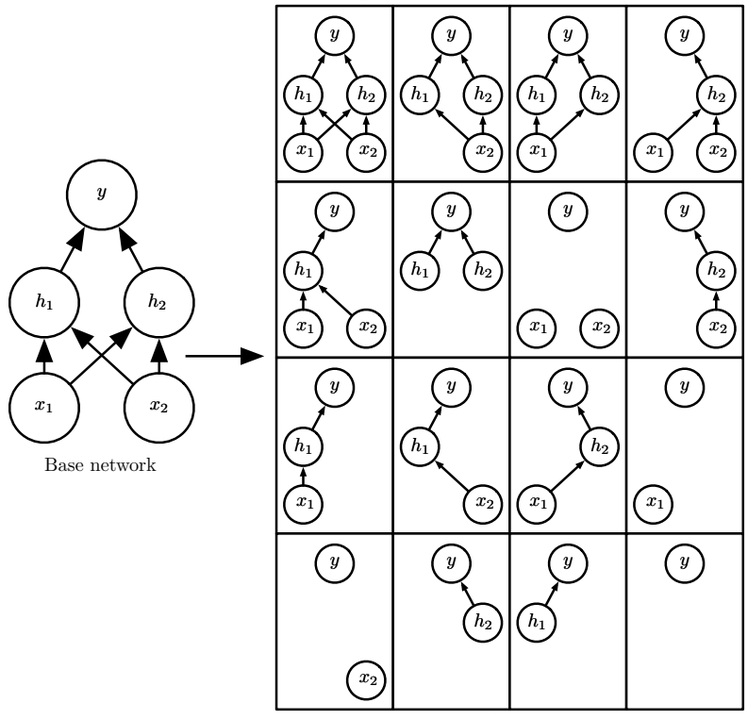
Note

Dropout

- Dropout consists of randomly setting to zero a subset of the hidden layer outputs at each iteration (excluded the output layer)
- Training with dropout can be seen as an efficient approximation to bagging
- Dropout trains the ensemble of all sub-networks obtained by removing non-output units

Note

Dropout



Base network

Ensemble of subnetworks

Note

Dropout

- Typical practice
  - Stochastic gradient descent (on minibatches)
  - At each iteration generate a random binary mask (each entry is an independent sample)
  - Apply mask to inputs and to hidden layer outputs
  - Define probability of one unit to be active

Typical dropout probability settings are 0.8 for inputs and 0.5 for hidden layers.

Dropout

- Formally, let μ be the mask defining which parameters θ are active and which are not
- Dropout consists in minimizing

$$E_{\mu}[J(\theta, \mu)]$$

- In contrast to bagging, here the sampled models (via the mask) are not independent and only trained for one gradient step

Note

Dropout

- Suppose that in the case of bagging each model outputs

$$p^i(y|x)$$

- Then, the prediction of the ensemble is the average

$$\frac{1}{k}\sum_{i=1}^{k} p^i(y|x)$$

- In the case of dropout we have instead

$$\sum_{\mu} p(\mu)p(y|x,\mu)$$

Because of the exponential number of masks, the averaging in the case of dropout is infeasible.

Dropout

- To obtain a good approximation of the predictions of the entire ensemble we can use the geometric mean rather than the arithmetic mean
- Let us assume that no probability maps values to zero

$$\tilde{p}_{\text{ensemble}}(y|x) = \sqrt[2^d]{\prod_{\mu} p(\mu)p(y|x,\mu)}$$

- Then we can make predictions by normalization

$$p_{\text{ensemble}}(y|x) = \frac{\tilde{p}_{\text{ensemble}}(y|x)}{\sum_{\bar{y}} \tilde{p}_{\text{ensemble}}(\bar{y}|x)}$$

Notice that the geometric mean is always a lower bound of the arithmetic mean.
d is the number of units that may have dropped.

Dropout

- **Key idea**: We can approximate p_{ensemble} by evaluating $p(y|x)$ in one model with all units active, but with weights of layer i multiplied by the probability of including unit i
- This is an approximation of the expected value of each unit
- In practice, we divide the weights by the probability of dropout after training

Note

Dropout

- Example: Softmax regression

$$p(y|v) = \mathrm{softmax}(W^\top v + b)_y$$

- We consider the masking

$$p(y|v,\mu) = \mathrm{softmax}(W^\top(\mu \odot v) + b)_y$$

and obtain the ensemble predictor

$$p_{\mathrm{ensemble}}(y|v) = \frac{\tilde{p}_{\mathrm{ensemble}}(y|v)}{\sum_{\bar{y}} \tilde{p}_{\mathrm{ensemble}}(\bar{y}|v)}$$

$$\tilde{p}_{\mathrm{ensemble}}(y|v) = \sqrt[2^n]{\prod_{\mu \in \{0,1\}^n} p(\mu) p(y|v,\mu)}$$

Note

Dropout

$$
\begin{aligned}
\tilde{p}_{\text{ensemble}}(y|v) &= \sqrt[2^n]{\prod_{\mu \in \{0,1\}^n} p(\mu) p(y|v,\mu)} \\
&\propto \sqrt[2^n]{\prod_{\mu \in \{0,1\}^n} \text{softmax}(W^\top(\mu \odot v) + b)_y} \\
&\propto \sqrt[2^n]{\prod_{\mu \in \{0,1\}^n} \exp(W_{y,\cdot}^\top(\mu \odot v) + b_y)} \\
&= \exp\left(\frac{1}{2^n} \sum_{\mu \in \{0,1\}^n} W_{y,\cdot}^\top(\mu \odot v) + b_y\right) \\
&= \exp\left(\frac{1}{2} W_{y,\cdot}^\top v + b_y\right)
\end{aligned}
$$

We assume that the probability of a mask p(\mu) is uniform.
Notice that the normalized probability will be a softmax with a reweighed matrix W (by 1/2, the probability of dropout).
The second to last row has two terms: the first is summing all 1s (and there are 1/2 2^n), the second is summing all b_y-s (and there are 2^n of those).

Dropout

$$\tilde{p}_{\text{ensemble}}(y|v) = \sqrt[2^n]{\prod_{\mu \in \{0,1\}^n} p(\mu) p(y|v,\mu)}$$

$$\propto \sqrt[2^n]{\prod_{\mu \in \{0,1\}^n} \text{softmax}(W^\top (\mu \odot v) + b)_y}$$

$$\propto \sqrt[2^n]{\prod_{\mu \in \{0,1\}^n} \exp(W^\top_{y,.} (\mu \odot v) + b_y)}$$

$$= \exp\left(\frac{1}{2^n} \sum_{\mu \in \{0,1\}^n} W^\top_{y,.} (\mu \odot v) + b_y \right)$$

$$= \exp\left(\frac{1}{2} W^\top_{y,.} v + b_y \right)$$

Dropout

- Advantages
  - It is computationally cheap (it only needs to generate, store and compute n random masks)
  - It is not limited to a type of model or training procedure

Note

Dropout

- General observations
  - Dropout is a regularizer as it reduces the effective capacity of the network
  - Dropout is less effective with few labeled examples for training (Bayesian Neural Networks do better here)
  - May interfere with batch normalization (see later slides)

Note

Dropout

- General observations
  - There is no requirement that the mask must be binary or map to a finite set
  - Srivastava et al 2014 showed that multiplying the weights by $\mu \sim \mathcal{N}(1, I)$ can outperform a binary mask (notice that the mean is 1 and so there is no need for post-scaling)

Note

Dropout

- Dropout can also be seen as a form of data augmentation
- We randomly create occlusions of high level features (e.g., we drop a face detector) and make later detectors robust to that
- It is difficult to create directly such an effect on the data without introducing more unrealistic artifacts

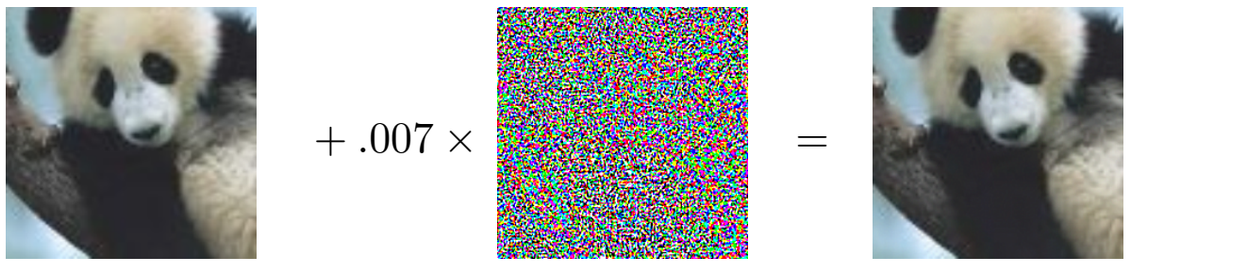
Note

Adversarial Training

- A technique to analyze a classification model is to search for the misclassified examples
- Another technique is to construct artificial examples that are misclassified although close to correctly classifier examples
- This last technique can be implemented via optimization and the computed examples are called **virtual adversarial examples**

These examples might be so close to the “good” examples, that a human observer cannot tell the difference. See next slide.

Adversarial Examples



from Goodfellow et al Explaining and Harnessing Adversarial Examples (2015)

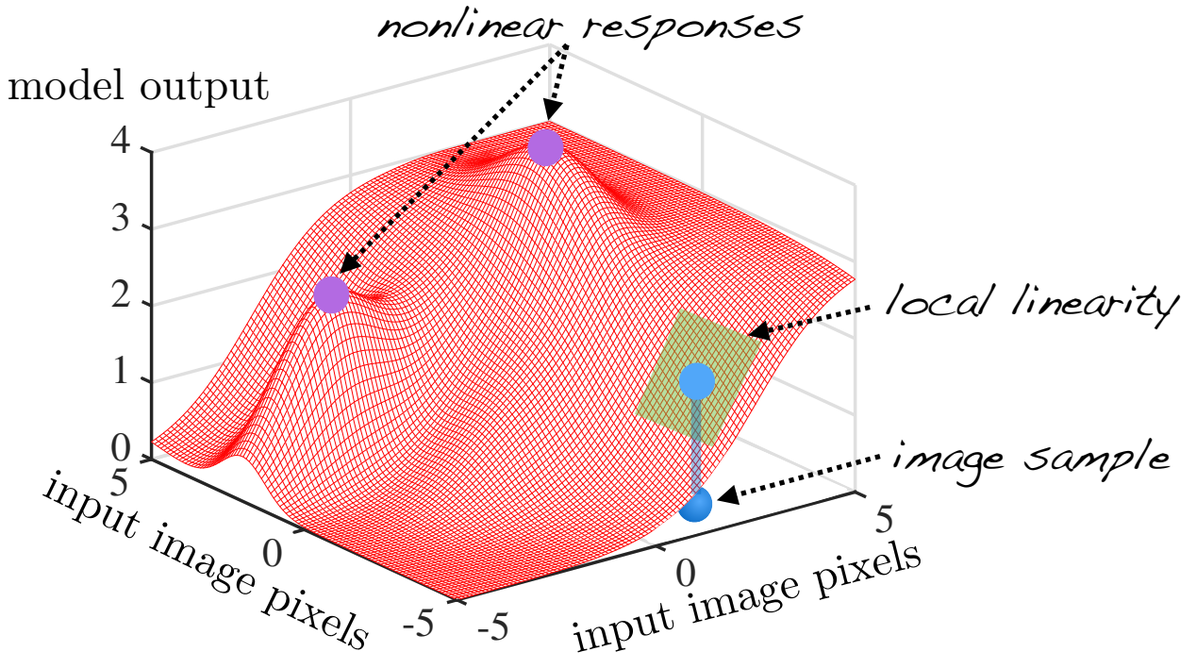
The example above shows how a pre-trained GoogLeNet can easily misclassify examples affected by small variations. This shows that the model is overfitting the data. The astonishing aspect of this example is that the modified image is indistinguishable from the original to the naked eye. Yet, the trained network cannot help by grossly misclassify the image with high confidence. We'll see how to turn this problem into a form of regularization. First, however, we need to understand why this is happening.

Adversarial Examples

- The underlining problem of adversarial examples is the (local) linearity of the neural networks
- Typical layers are locally or fully linear functions (e.g., convolutions, ReLUs, fully connected layers)
- This makes training easier, but also makes the function locally linear (a local subspace)
- The problem of adversarial examples applies also to other models (including linear classifiers!)

Note

Geometrical Interpretation



Variations in the input image (translations on the plane) will generate large variations in the output at a locally linear location. At locations where the response is more nonlinear (e.g., at one of the peaks in the graph) the output is more robust to variations in the input.

Toy example (from Andrej Karpathy)

- Consider a classification with the logistic function

$$p(y=1|x;w,b)=\sigma(w^\top x+b) \qquad \sigma(z)=\frac{1}{1+e^{-z}}$$

- Let $b=0$ and suppose that the weights are $w=[-1,-1,1,-1,1,-1,1,1,-1,1]$ and a sample is $x=[2,-1,3,-2,2,2,1,-4,5,1]$

- Then $w^\top x=-3$ and the probability of the class 1 is $\frac{1}{1+e^{-(-3)}}\simeq 5\%$

Note

Toy example (from Andrej Karpathy)

- Now create an adversarial example as

$$x_{\mathrm{adv}} = x + 0.5w$$
$$= [1.5, -1.5, 3.5, -2.5, 2.5, 1.5, 1.5, -3.5, 4.5, 1.5]$$

- Then, we have $w^\top x_{\mathrm{adv}} = 2$ and the probability of class 1 is now $\dfrac{1}{1+e^{-2}} \simeq 88\%$!

Note

Toy example (from Andrej Karpathy)

- The adversarial example $x_{\text{adv}} = x + \epsilon w$ with $\epsilon = 0.5$

  led to $\dfrac{1}{1+e^{-(w^\top x_{\text{adv}})}} = \dfrac{1}{1+e^{-(w^\top x - \epsilon |w|^2)}}$

- With higher dimensions (as in images) the norm $|w|^2$ can grow very quickly and allows using very small ϵ (as in the earlier example)

Note

Adversarial Training

- A simple solution is to create and add such examples to the training set (data augmentation)
- Training on adversarial examples is mostly for security, but can sometimes provide generic regularization too
- It forces the model to have a flat response around the dataset samples
- So far, there is no effective way to remove completely this behavior

Note