EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

MATHEMATISCH-
NATURWISSENSCHAFTLICHE FAKULTÄT

Deep Neural Networks

Chapter 7: Regularization for Deep Learning

7. Regularization for Deep Learning

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Regularization: “any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.”
- Most regularization strategies are based on regularizing estimators. Regularization of an estimator works by trading increased bias for much reduced variance.
- An effective regularizer is one that reduces variance significantly while not overly increasing the bias.
- In practical deep learning scenarios we almost always find that the best fitting model (minimizing generalization error) is a large model that has been regularized appropriately.

7.1 Parameter Norm Penalties

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Many regularization approaches are based on limiting the capacity of models by adding a parameter norm penalty $\Omega(\boldsymbol{\theta})$ to the objective function J:

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

- $\alpha \in [0,\infty)$ is a hyperparameter that weights the relative contribution of the norm penalty term, Ω, relative to the standard objective function J.
- For neural networks, we typically use a parameter norm penalty Ω that *penalizes only the weights* w of the affine transformation at each layer and *leaves the biases unregularized*. Regularizing the bias parameters can introduce a significant amount of underfitting.

7.1.1 L^2 Parameter Regularization

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- L^2 parameter norm penalty is known as weight decay. This regularization strategy drives the weights closer to the origin by adding a regularization $\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$ to the objective function.
- L^2 regularization is also known as ridge regression or Tikhonov regularization.
- Assuming no bias parameter, so θ is just w, we write:

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

with the corresponding parameter gradient

$$\nabla_w \tilde{J}(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) = \alpha\boldsymbol{w} + \nabla_w J(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y})$$

- To take a single gradient step we update the weights

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon\left(\alpha\boldsymbol{w} + \nabla_w J(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y})\right)$$

7.1.1 L^2 Parameter Regularization

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Written another way, the update is:

$$\boldsymbol{w} \leftarrow (1-\epsilon\alpha)\boldsymbol{w} - \epsilon\nabla_{w} J(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y})$$

- The modified learning rule multiplicatively shrinks the weight vector by a constant factor on each step, just before performing the usual gradient update.
- We further simplify the analysis by making a quadratic approximation assumption to the objective function in the neighborhood of the value of the weights that obtains minimal unregularized training cost, $\boldsymbol{w}^* = \arg\min_w J(\boldsymbol{w})$. This approximation is given as

$$\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{w}^*) + \tfrac{1}{2}(\boldsymbol{w}-\boldsymbol{w}^*)^T \boldsymbol{H}(\boldsymbol{w}-\boldsymbol{w}^*)$$

where H is the Hessian matrix of J with respect to $\boldsymbol{w}$ evaluated at $\boldsymbol{w}^*$.

7.1.1 L^2 Parameter Regularization

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- As $\boldsymbol{w}^*$ is the location of a minimum of J, H is positive semidefinite.
- The minimum of $\hat{J}$ occurs where its gradient

$$\nabla_w \hat{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

  is equal to 0.
- To study the effect of weight decay, we modify this equation by adding the weight decay gradient and solve for the minimum of the regularized version of $\hat{J}$.
  $\tilde{\boldsymbol{w}}$ represents the location of the minimum.

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

$$(\boldsymbol{H} + \alpha\boldsymbol{I})\tilde{\boldsymbol{w}} = \boldsymbol{H}\boldsymbol{w}^*$$

$$\tilde{\boldsymbol{w}} = (\boldsymbol{H} + \alpha\boldsymbol{I})^{-1}\boldsymbol{H}\boldsymbol{w}^*$$

7.1.1 L^2 Parameter Regularization

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- As $\alpha \to 0$, the regularized solution $\tilde{\boldsymbol{w}} \to \boldsymbol{w}^*$.
- But what happens as α grows? Because $\boldsymbol{H}$ is real and symmetric, we can decompose it into a diagonal matrix Λ and an orthonormal basis of eigenvectors, $\boldsymbol{Q}$, such that $\boldsymbol{H} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$.
- Applying this decomposition to $\tilde{\boldsymbol{w}} = \left(\boldsymbol{H} + \alpha \boldsymbol{I}\right)^{-1} \boldsymbol{H}\boldsymbol{w}^*$:

$$\tilde{\boldsymbol{w}} = \left(\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T + \alpha \boldsymbol{I}\right)^{-1} \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T \boldsymbol{w}^*$$

$$= \left[\boldsymbol{Q}\left(\boldsymbol{\Lambda} + \alpha \boldsymbol{I}\right)\boldsymbol{Q}^T\right]^{-1} \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T \boldsymbol{w}^*$$

$$= \boldsymbol{Q}\left(\boldsymbol{\Lambda} + \alpha \boldsymbol{I}\right)^{-1} \boldsymbol{\Lambda}\boldsymbol{Q}^T \boldsymbol{w}^*$$

doesn't have to remember function, just conclusion

- Weight decay rescales $\boldsymbol{w}^*$ along the axes defined by the eigenvectors of $\boldsymbol{H}$.

7.1.1 L^2 Parameter Regularization

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Specifically, the component of $\boldsymbol{w}^*$ that is aligned with the i-th eigenvector of $\boldsymbol{H}$ is rescaled by a factor of $\frac{\lambda_i}{\lambda_i + \alpha}$.

- Along the directions where the eigenvalues of $\boldsymbol{H}$ are relatively large, for example, where $\lambda_i \gg \alpha$, the effect of regularization is relatively small. However, components with $\lambda_i \ll \alpha$ will be shrunk to have nearly zero magnitude. This effect is illustrated in fig. 7.1.
- Only directions along which the parameters contribute significantly to reducing the objective function are preserved relatively intact. Directions that do not contribute to reducing the objective function are decayed away by regularization.

7.1.1 L^2 Parameter Regularization

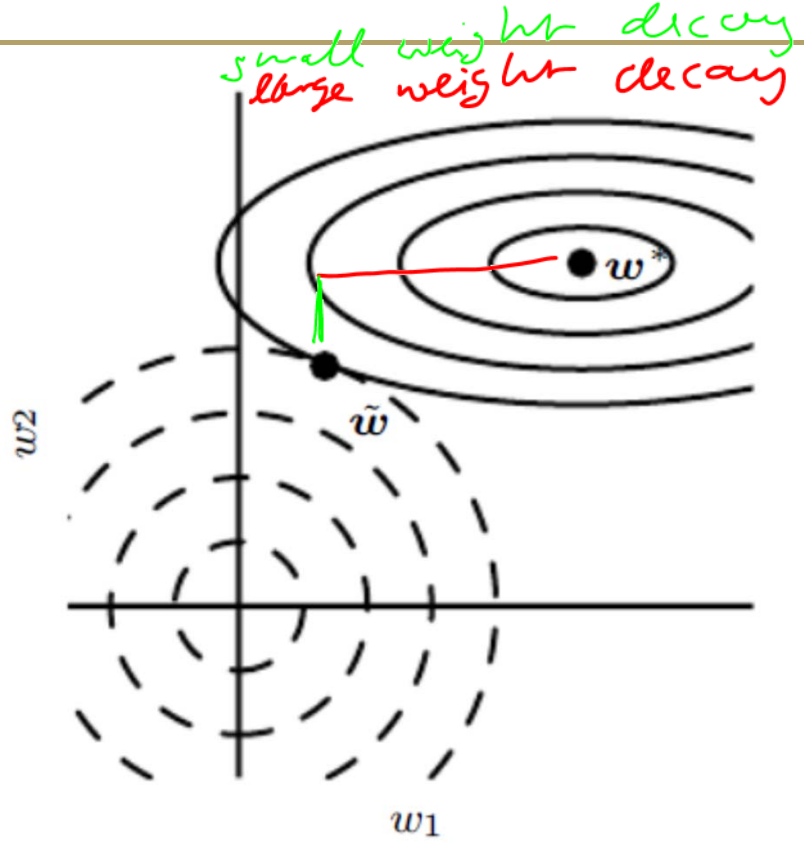


Fig. 7.1: The effect of L^2 regularization on the value of the optimal w. The solid ellipses represent contours of equal value of the unregularized objective. The dotted circles represent contours of equal value of the L^2 regularizer. At the point $\tilde{\boldsymbol{w}}$, these competing objectives reach an equilibrium. In the first dimension w_1, the eigenvalue of the Hessian of J is small. The objective function does not increase much when moving horizontally away from $\boldsymbol{w}^*$. Because the objective function does not express a strong preference along this direction, the regularizer has a strong effect on this axis. The regularizer pulls w_1 close to zero. In the second dimension, the objective function is very sensitive to movements away from $\boldsymbol{w}^*$. Its eigenvalue is large, indicating high curvature. As a result, weight decay affects the position of w_2 relatively little.

7.1.2 L^1 Parameter Regularization

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- L^1 regularization on the model parameter w is defined as:

$$\Omega(\boldsymbol{\theta}) = \|\boldsymbol{w}\|_1 = \sum_i |w_i|$$

- Thus, the regularized objective function $\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ is given by

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \|w\|_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

with the corresponding gradient (actually, sub-gradient):

$$\nabla_w \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \cdot sign(\boldsymbol{w}) + \nabla_w J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

- Now the regularization contribution to the gradient no longer scales linearly with each w_i; instead it is a constant factor with a sign equal to sign(w_i).
- In comparison to L^2 regularization, L^1 regularization results in a solution that is more sparse.

7.1.2 L^1 Parameter Regularization

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- The sparsity property induced by L^1 regularization has been used extensively as a feature selection mechanism.
- Feature selection simplifies a machine learning problem by choosing which subset of the available features should be used. In particular, the well known LASSO (least absolute shrinkage and selection operator) model (Tibshirani, 1995) integrates an L^1 penalty with a linear model and a least squares cost function.
- The L^1 penalty causes a subset of the weights to become zero, suggesting that the corresponding features may safely be discarded.

7.2 Norm Penalties as Constrained Optimization

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- We skip this section

7.3 Regularization and Under-Constrained Problems

too many weights for training examples

- In some cases, regularization is necessary for machine learning problems to be properly defined.
- Many linear models in machine learning depend on inverting the matrix $\boldsymbol{X}^T\boldsymbol{X}$. This is not possible whenever $\boldsymbol{X}^T\boldsymbol{X}$ is singular. This matrix can be singular whenever the data generating distribution has no variance in some direction
- In this case, many forms of regularization correspond to inverting $\boldsymbol{X}^T\boldsymbol{X}+\alpha\boldsymbol{I}$ instead. This regularized matrix is guaranteed to be invertible.
- We can solve underdetermined linear equations using the Moore-Penrose pseudoinverse

$$\boldsymbol{X}^+ = \lim_{\alpha \searrow 0} \left(\boldsymbol{X}^T\boldsymbol{X} + \alpha\boldsymbol{I} \right)^{-1} \boldsymbol{X}^T$$

7.4 Dataset Augmentation

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- The best way to make a machine learning model generalize better is to train it on more data.
- In practice, the amount of data we have is limited.
- One way to get around this problem is to create fake data and add it to the training set.
- This approach is easiest for classification. A classifier needs to take a complicated, high dimensional input $\boldsymbol{x}$ and summarize it with a single category identity y.
- This means that a major task of a classifier is to be invariant to a wide variety of transformations.
- We can generate new $(\boldsymbol{x}, y)$ pairs easily just by transforming the $\boldsymbol{x}$ inputs in our training set, keeping y.

7.4 Dataset Augmentation

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Dataset augmentation has been a particularly effective for object recognition.
- Images are high dimensional and include an enormous variety of factors of variation, many of which can be easily simulated.
- Operations like translating the training images a few pixels in each direction can often greatly improve generalization, even if the model has already been designed to be partially translation invariant by using convolution and pooling techniques.
- Many other operations such as rotating the image or scaling the image have also proven quite effective.

7.4 Dataset Augmentation: Noise

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Injecting noise in the input to a neural network (Sietsma and Dow, 1991) can also be seen as a form of data augmentation.
- For many classification and even some regression tasks, the task should still be possible to solve even if small random noise is added to the input.
- Neural networks prove not to be very robust to noise, however (Tang and Eliasmith, 2010).
- One way to improve the robustness of neural networks is to train them with random noise applied to their inputs.
- Dropout, a powerful regularization strategy (sect. 7.12), can be seen as a process of constructing new inputs by *multiplying* by noise.

7.5 Noise Robustness

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Noise injection can be much more powerful than simply shrinking the parameters, especially when the noise is added to the hidden units (see Dropout later).
- Noise may also be added to the weights. This technique has been used primarily in the context of recurrent neural networks (Jim *et al.*, 1996; Graves, 2011).
- This can be interpreted as a stochastic implementation of Bayesian inference over the weights. Model weights are regarded to be uncertain and representable via a probability distribution.
- Adding noise to the weights is a practical, stochastic way to reflect this uncertainty.

7.5.1 Injecting Noise at Output Targets

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Most datasets have some amount of mistakes in the y labels. It can be harmful to maximize $\log p(y \mid \boldsymbol{x})$ when y is a mistake.
- One way to prevent this is to explicitly model the noise on the labels. For example, we can assume that for some small constant ϵ, the training set label y is correct with probability $1-\epsilon$, and otherwise any of the other possible labels might be correct. This assumption is easy to incorporate into the cost function analytically.
- Label smoothing regularizes a model based on a softmax with k output values by replacing the hard 0 / 1 classification targets with targets of $\frac{\epsilon}{k-1}$ and $1-\epsilon$, respectively. The standard cross-entropy loss may then be used with these soft targets.

7.5.1 Label Smoothing

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- While maximum likelihood learning with a softmax classifier and hard targets may actually never converge (the softmax can never predict a probability of exactly 0 or 1, so it will continue to learn larger and larger weights, making more extreme predictions forever, if not prevented by weight decay regularization)
- Label smoothing has the advantage of preventing the pursuit of hard probabilities without discouraging correct classification.
- This strategy has been used since the 1980s and is still used in modern neural networks (Szegedy *et al.*, 2015).

7.6 Semi-Supervised Learning

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- In semi-supervised learning, both unlabeled examples from $P(\boldsymbol{x})$ and labeled examples from $P(\boldsymbol{x}, \boldsymbol{y})$ are used to estimate $(\boldsymbol{y} \mid \boldsymbol{x})$ or predict y from $\boldsymbol{x}$.
- In the context of deep learning, semi-supervised learning usually refers to learning a representation $\boldsymbol{h} = f(\boldsymbol{x})$.
- The goal is to learn a representation so that examples from the same class have similar representations.
- Unsupervised learning can provide useful cues for how to group examples in representation space. Examples that cluster tightly in the input space should be mapped to similar representations. A linear classifier in the new space may achieve better generalization in many cases (Belkin and Niyogi, 2002; Chapelle *et al.*, 2003).

7.6 Semi-Supervised Learning

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- An old variant of this approach is the application of PCA as a pre-processing step before applying a classifier on the projected data.
- Instead of having separate unsupervised and supervised components in the model, one can construct models in which a generative model of either $P(\boldsymbol{x})$ or $P(\boldsymbol{x}, \boldsymbol{y})$ shares parameters with a discriminative model of $P(\boldsymbol{y}|\boldsymbol{x})$.
- One can then trade-off the supervised criterion $-\log P(\boldsymbol{y}|\boldsymbol{x})$ with the unsupervised or generative one (such as $-\log P(\boldsymbol{x})$ or $-\log P(\boldsymbol{x}, \boldsymbol{y})$).
- The generative criterion then expresses a prior belief that the structure of $P(\boldsymbol{x})$ is connected to the structure of $P(\boldsymbol{y}|\boldsymbol{x})$ in a way captured by the shared parametrization.

7.7 Multi-Task Learning

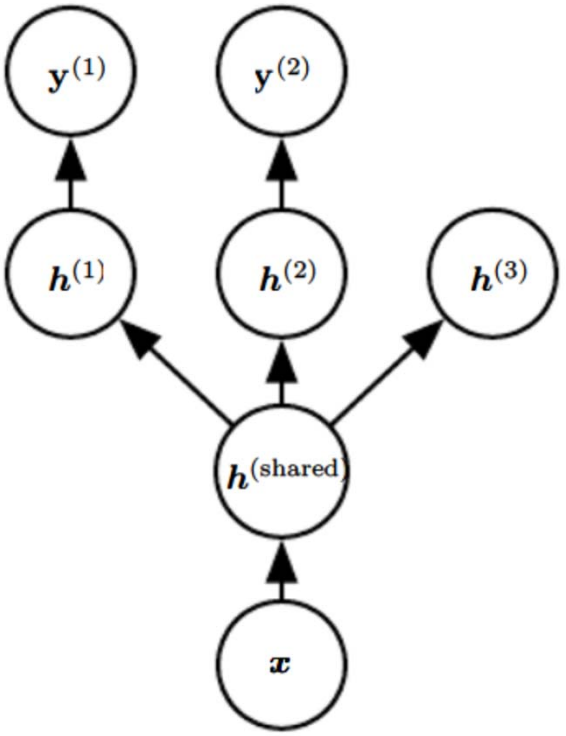
EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Multi-task learning (Caruana, 1993) is a way to improve generalization by pooling the examples arising out of several tasks.
- When part of a model is shared across tasks, that part of the model is more constrained towards good values (assuming the sharing is justified), often yielding better generalization.
- Figure 7.2 (next slide) illustrates a common form of multi-task learning, in which different supervised tasks (predicting $\boldsymbol{y}^{(i)}$ given $\boldsymbol{x}$) share the same input $\boldsymbol{x}$, as well as some intermediate-level representation $\boldsymbol{h}^{(\text{shared})}$ capturing a common pool of factors.

7.7 Multi-Task Learning

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Fig. 7.2: Multi-task learning where the tasks share a common input $\boldsymbol{x}$ but involve different target random variables $\boldsymbol{y}^{(1)}$ and $\boldsymbol{y}^{(2)}$.
- The lower layers of a deep network can be shared across such tasks, while task-specific parameters (associated with the weights into and from $\boldsymbol{h}^{(1)}$ and $\boldsymbol{h}^{(2)}$) can be learned on top of those, yielding a shared representation $\boldsymbol{h}^{(\text{shared})}$.
- The underlying assumption is that there exists a common pool of factors that explain the variations in the input $\boldsymbol{x}$, while each task is associated with a subset of these factors.
- Here, top-level hidden units $\boldsymbol{h}^{(1)}$ and $\boldsymbol{h}^{(2)}$ are specialized to each task (predicting $\boldsymbol{y}^{(1)}$ and $\boldsymbol{y}^{(2)}$) while some lower level representation $\boldsymbol{h}^{(\text{shared})}$ is shared across all tasks.



7.8 Early Stopping

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- When training large models with sufficient capacity to overfit the task, we often observe that the training error decreases steadily over time, but the validation set error begins to rise again. This behavior occurs very reliably.
- This means we can obtain a model with better validation set error by returning to the parameter setting at the point in time with the lowest validation 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.
- This early stopping algorithm terminates when no parameters have improved over the best recorded validation error for some number of iterations.

7.8 Early Stopping

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Early stopping is probably the most commonly used form of regularization in deep learning. Its popularity is due both to its effectiveness and its simplicity.
- We control the effective capacity of the model by testing how many steps it can take to fit the training set.
- The “training time” hyperparameter is unique in that by definition a single run of training tries out many values of the hyperparameter.
- The only significant cost of early stopping is running the validation set evaluation periodically during training. Ideally, this is done in parallel to the training process on a separate machine, separate CPU, or separate GPU from the main training process.

7.8 Early Stopping

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- If such resources are not available, then the cost of these periodic evaluations may be reduced by using a validation set that is small compared to the training set or by evaluating the validation set error less frequently.
- Early stopping is a very unobtrusive regularization, in that it requires almost no change in the underlying training procedure or the objective function.
- Early stopping may be used either alone or in conjunction with other regularization strategies.
- Early stopping requires a validation set, i.e. some training data is not fed to the model. To best exploit this extra data, one can perform extra training after the initial training with early stopping. In the second, extra training step, all of the training data is included.

7.8 Early Stopping

Long derivation, consider jumping to end of p. 30

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- In the case of a simple linear model with a quadratic error function and simple gradient descent we can show that early stopping is equivalent to L^2 regularization.
- To compare with L^2 regularization, we examine a setting where the only parameters are linear weights ($\boldsymbol{\theta} = \boldsymbol{w}$).
- We can model the cost function J with a quadratic approximation in the neighborhood of the empirically optimal value of the weights

$$\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^T \boldsymbol{H} (\boldsymbol{w} - \boldsymbol{w}^*)$$

where $\boldsymbol{H}$ is the Hessian matrix of J with respect to $\boldsymbol{w}$ evaluated at $\boldsymbol{w}^*$. Given the assumption that $\boldsymbol{w}^*$ is a minimum of $J(w)$, we know that $\boldsymbol{H}$ is positive semidefinite.

7.8 Early Stopping

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Under a local Taylor series approximation, the gradient is given by:

$$\nabla_w \hat{J}(\boldsymbol{w}) = \boldsymbol{H}\left(\boldsymbol{w} - \boldsymbol{w}^*\right)$$

- We are going to study the trajectory followed by the parameter vector during training. For simplicity, let us set the initial parameter vector to the origin, $\boldsymbol{w}^{(0)} = 0$.
- Let us study the approximate behavior of gradient descent on J by analyzing gradient descent on $\hat{J}$:

$$\begin{aligned}
\boldsymbol{w}^{(\tau)} &= \boldsymbol{w}^{(\tau-1)} - \epsilon \nabla_w \hat{J}\left(\boldsymbol{w}^{(\tau-1)}\right) \\
&= \boldsymbol{w}^{(\tau-1)} - \epsilon \boldsymbol{H}\left(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*\right) \\
\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^* &= \left(\boldsymbol{I} - \epsilon \boldsymbol{H}\right)\left(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*\right)
\end{aligned}$$

7.8 Early Stopping

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Let us now rewrite this expression in the space of the eigenvectors of $\boldsymbol{H}$, exploiting the eigendecomposition of $\boldsymbol{H} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$, where $\boldsymbol{\Lambda}$ is a diagonal matrix and $\boldsymbol{Q}$ is an orthonormal basis of eigenvectors.

$$\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^* = \left(\boldsymbol{I} - \epsilon \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T\right)\left(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*\right)$$

$$\boldsymbol{Q}^T\left(\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^*\right) = \left(\boldsymbol{I} - \epsilon\boldsymbol{\Lambda}\right)\boldsymbol{Q}^T\left(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*\right)$$

- Assuming that $\boldsymbol{w}^{(0)} = 0$ and that ϵ is chosen to be small enough to guarantee $\left|1 - \epsilon\lambda_i\right| < 1$, the parameter trajectory during training after τ parameter updates is as follows:

$$\boldsymbol{Q}^T\boldsymbol{w}^{(\tau)} = \left[\boldsymbol{I} - \left(\boldsymbol{I} - \epsilon\boldsymbol{\Lambda}\right)^{\tau}\right]\boldsymbol{Q}^T\boldsymbol{w}^*$$

7.8 Early Stopping

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Now, the expression for $\boldsymbol{Q}^T\tilde{\boldsymbol{w}}$ in the equation for L^2 regularization can be rearranged as:

$$\boldsymbol{Q}^T\tilde{\boldsymbol{w}} = \left(\boldsymbol{\Lambda} + \alpha\boldsymbol{I}\right)^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^T\boldsymbol{w}^*$$

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

- Comparing this equation with the last one on previous page, we see that if the hyperparameters ϵ, α, and τ are chosen such that

$$\left(\boldsymbol{I} - \epsilon\boldsymbol{\Lambda}\right)^{\tau} = \left(\boldsymbol{\Lambda} + \alpha\boldsymbol{I}\right)^{-1}\alpha$$

  then L^2 regularization and early stopping can be seen to be equivalent (at least under the quadratic approximation of the objective function).

7.8 Early Stopping

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Going even further, by taking logarithms and using the series expansion for log(1 + x), we can conclude that if all λ_i are small then

$$\tau \approx \frac{1}{\epsilon\alpha}$$

$$\alpha \approx \frac{1}{\tau\epsilon}$$

- That is, under these assumptions, the number of training iterations τ plays a role inversely proportional to the L^2 regularization parameter, and the inverse of $\tau\epsilon$ plays the role of the weight decay coefficient α.

7.9 Parameter Tying and Parameter Sharing

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Sometimes we want to express a dependency that certain parameters should be close to another.
- We have model A with parameters $w^{(A)}$ and model B with parameters $w^{(B)}$. The two models have different, but related outputs: $\hat{y}^{(A)} = f\left(\boldsymbol{w}^{(A)}, \boldsymbol{x}\right)$ and $\hat{y}^{(B)} = g\left(\boldsymbol{w}^{(B)}, \boldsymbol{x}\right)$.
- If we believe the Model parameters should be close to each other, we can use a parameter norm penalty of the form $\Omega\left(\boldsymbol{w}^{(A)}, \boldsymbol{w}^{(B)}\right) = \left\|\boldsymbol{w}^{(A)} - \boldsymbol{w}^{(B)}\right\|_2^2$.
- This is called parameter tying.
- A more stringent way to enforce similarity of parameters is parameter sharing, which forces them to be equal.
- Parameter sharing has the advantage, that only the unique parameters need be stored in memory.

7.9 Parameter Sharing: CNNs

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- By far the most popular and extensive use of parameter sharing occurs in convolutional neural networks (CNNs) applied to computer vision.
- Natural images have many statistical properties that are invariant to translation.
- CNNs take this into account by sharing parameters across multiple image locations.
- Parameter sharing has allowed CNNs to dramatically lower the number of unique model parameters and to significantly increase network sizes without requiring a corresponding increase in training data.

7.10 Sparse Representations

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Another regulation strategy is to place a penalty on the activations of the units in a neural network, encouraging their activations to be sparse.
- Sparsely parameterized linear regression model (sparse weight matrix $\boldsymbol{A}$):

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

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

7.10 Sparse Representations

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Linear regression with a sparse representation $\boldsymbol{h}$ of the data $\boldsymbol{x}$.

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

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

- Norm penalty regularization of representations is performed by adding a norm penalty on the *representation*, denoted $\Omega(\boldsymbol{h})$, to the loss function $\boldsymbol{J}$.

7.10 Sparse Representations

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

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

- Just as an L^1 penalty on the parameters induces parameter sparsity, an L^1 penalty on the elements of the representation induces representational sparsity:

$$\Omega(\boldsymbol{h}) = \|\boldsymbol{h}\|_1 = \sum_i |h_i|$$

7.11 Bagging, Ensemble Methods

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Bagging (short for bootstrap aggregating) is a technique for reducing generalization error by combining several models (Breiman, 1994).
- The idea is to train several different models separately, then have all of the models vote on the output for test examples.
- This is an example of a general strategy in machine learning called model averaging.
- Techniques employing this strategy are known as ensemble methods.
- The reason that model averaging works is that different models will usually not make all the same errors on the test set.

7.11 Bagging, Ensemble Methods

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Consider a set of k regression models. Suppose that each model makes an error ϵ_i on each example, with the errors drawn from a multivariate normal distrib. with zero mean, variances $\mathbb{E}\left[\epsilon_i^2\right] = v$ and covariances $\mathbb{E}\left[\epsilon_i \epsilon_j\right] = c$. Then the error made by the average prediction of all the ensemble models is $\frac{1}{k}\sum_i \epsilon_i$. The expected squared error of the ensemble predictor is

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

- In the case where the errors are perfectly correlated and $c = v$, the mean squared error reduces to v, so the model averaging does not help at all.

7.11 Bagging, Ensemble Methods

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- In the case where the errors are uncorrelated and $c = 0$, the expected squared error of the ensemble is only $\frac{1}{k}v$.
- This means that the expected squared error of the ensemble decreases linearly with the ensemble size.

- Bagging involves constructing k different datasets:
- Each dataset has the same number of examples as the original dataset, but is constructed by sampling with replacement from the original dataset.
- This means that each dataset is missing some of the examples from the original dataset and contains several duplicate examples. Model i is then trained on dataset i.

7.12 Dropout

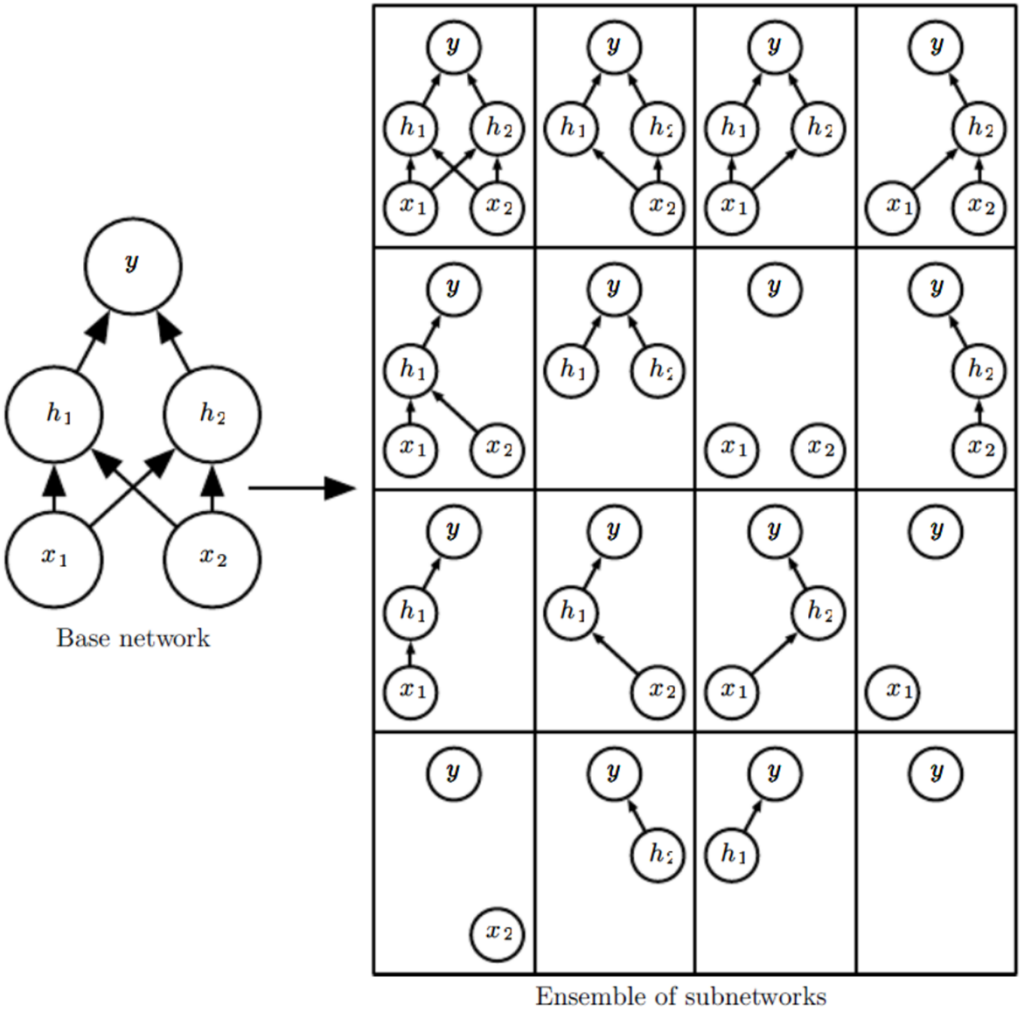
EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Dropout (Srivastava *et al.*, 2014) is a computationally inexpensive but powerful method of regularizing models.
- It can be thought of as a method of making bagging practical for ensembles of many large neural networks.
- Bagging involves training multiple models, and evaluating multiple models on each test example. This is impractical when each model is a large neural network.
- Dropout provides an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks.
- Specifically, dropout trains the ensemble consisting of all sub-networks that can be formed by removing non-output units from an underlying base network.

7.12 Dropout

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Fig. 7.6: Dropout trains an ensemble consisting of all sub-networks that can be constructed by removing non-output units from an underlying base network.
- Here, we begin with a base network with two visible units and two hidden units.
- There are sixteen possible subsets of these four units. We show all sixteen subnetworks that may be formed by dropping out different subsets of units from the original network.



7.12 Dropout

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- In most modern NNs we can effectively remove a unit from a network by multiplying its output value by zero.
- To train with dropout, we use a minibatch-based learning algorithm that makes small steps, such as stochastic gradient descent.
- Each time we load an example into a minibatch, we randomly sample a different binary mask to apply to all of the input and hidden units in the network.
- The mask for each unit is sampled independently from all of the others. The probability of sampling a mask value of one is a hyperparameter fixed before training begins.
- Typically, an input unit is included with probability 0.8 and a hidden unit is included with probability 0.5.
- We then run forward propagation, back-propagation, and the learning update as usual.

7.12 Dropout

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- More formally, suppose that a mask vector μ specifies which units to include, and $J(\boldsymbol{\theta}, \boldsymbol{\mu})$ defines the cost of the model defined by parameters $\boldsymbol{\theta}$ and mask $\boldsymbol{\mu}$. Then dropout training consists in minimizing $\mathbb{E}_{\mu} J(\boldsymbol{\theta}, \boldsymbol{\mu})$. The expectation contains exponentially many terms but we can obtain an unbiased estimate of its gradient by sampling values of $\boldsymbol{\mu}$.
- Dropout training is not quite the same as bagging training. In the case of bagging, the models are all independent. In the case of dropout, the models share parameters, with each model inheriting a different subset of parameters from the parent neural network.

7.12 Dropout

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- This parameter sharing makes it possible to represent an exponential number of models with available memory.
- In the case of bagging, each model is trained to convergence on its respective training set.
- In the case of dropout, a tiny fraction of the possible sub-networks are each trained for a single step, and the parameter sharing causes the remaining sub-networks to arrive at good settings of the parameters.
- These are the only differences. Beyond these, dropout follows the bagging algorithm. For example, the training set encountered by each sub-network is indeed a subset of the original training set sampled with replacement.

7.12 Dropout

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- To make a prediction, a bagged ensemble must accumulate votes from all of its members. We refer to this process as inference in this context.
- Now, we assume that the model's role is to output a probability distribution. In the case of bagging, each model i produces a probability distribution $p^{(i)}\left(y \mid \boldsymbol{x}\right)$.
- The prediction of the ensemble is given by the arithmetic mean of all of these distributions, $\frac{1}{k}\sum_{i=1}^{k} p^{(i)}\left(y \mid \boldsymbol{x}\right)$.
- In the case of dropout, each sub-model defined by mask vector $\boldsymbol{\mu}$ defines a probability distribution $p\left(y \mid \boldsymbol{x}, \boldsymbol{\mu}\right)$.
- The arithmetic mean over all masks is given by $\sum_{\mu} p\left(\boldsymbol{\mu}\right) p\left(y \mid \boldsymbol{x}, \boldsymbol{\mu}\right)$.

7.12 Dropout

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Because this sum includes an exponential number of terms, it is intractable to evaluate.
- Instead, we can approximate the inference with sampling, by averaging the output from many masks.
- Even 10-20 masks are often sufficient to obtain good performance.
- However, there is an even better approach, that allows us to obtain a good approximation to the predictions of the entire ensemble, with only one forward propagation.
- To do so, we change to using the *geometric mean* rather than the arithmetic mean of the ensemble members' predicted distributions.

7.12 Dropout

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- The geometric mean of multiple probability distributions is not guaranteed to be a probability distribution.
- To guarantee that the result is a probability distribution, we impose the requirement that none of the sub-models assigns probability 0 to any event, and we renormalize the resulting distribution.
- The unnormalized probability distribution defined directly by the geometric mean is given by

$$\tilde{p}_{ensemble}\left(y \mid \boldsymbol{x}\right) = \sqrt[2^d]{\prod_{\boldsymbol{\mu}} p\left(y \mid \boldsymbol{x}, \boldsymbol{\mu}\right)}$$

where d is the number of units that may be dropped.

7.12 Dropout

- To make predictions we must re-normalize the ensemble

$$p_{ensemble}\left(y \mid \boldsymbol{x}\right) = \frac{\tilde{p}_{ensemble}\left(y \mid \boldsymbol{x}\right)}{\sum_{y'} \tilde{p}_{ensemble}\left(y' \mid \boldsymbol{x}\right)}$$

- A key insight (Hinton *et al.*, 2012c) in dropout is that we can approximate $p_{ensemble}$ by evaluating $p(y \mid x)$ in the model with all units, but with the weights going out of unit i multiplied by the probability of including unit i.
- The motivation for this modification is to capture the right expected value of the output from that unit. We call this approach the weight scaling inference rule.
- There is not yet a theoretical argument for the accuracy of this approximate inference rule in deep nonlinear networks, but empirically it performs very well.

7.12 Dropout

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Because we usually use an inclusion probability of 1/2, the weight scaling rule amounts to dividing the weights by 2 at the end of training, and then using the model as usual.
- Another way to achieve the same result is to multiply the states of the units by 2 during training.
- Either way, the goal is to make sure that the expected total input to a unit at test time is roughly the same as the expected total input to that unit at train time, although half the units at train time are missing on average.
- For many classes of models that do not have nonlinear hidden units, the weight scaling inference rule is exact.

7.12 Dropout for Linear Softmax Class.

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- As example, consider a softmax regression classifier with n input variables represented by the vector v:

$$P(\mathrm{y} = y \mid \boldsymbol{v}) = \mathrm{softmax}\left(\boldsymbol{W}^T \boldsymbol{v} + \boldsymbol{b}\right)_y$$

- We can index into the family of sub-models by element-wise multiplication of the input with a binary vector $\boldsymbol{d}$:

$$P(\mathrm{y} = y \mid \boldsymbol{v}; \boldsymbol{d}) = \mathrm{softmax}\left(\boldsymbol{W}^T \left(\boldsymbol{d} \odot \boldsymbol{v}\right) + \boldsymbol{b}\right)_y$$

- The ensemble predictor is defined by re-normalizing the geom. mean over all ensemble members' predictions:

$$P_{ensemble}(\mathrm{y} = y \mid \boldsymbol{x}) = \frac{\tilde{p}_{ensemble}(\mathrm{y} = y \mid \boldsymbol{x})}{\sum_{y'} \tilde{p}_{ensemble}(\mathrm{y} = y' \mid \boldsymbol{x})}$$

where

$$\tilde{P}_{ensemble}(\mathrm{y} = y \mid \boldsymbol{v}) = \sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} P(\mathrm{y} = y \mid \boldsymbol{v}; \boldsymbol{d})}$$

Long derivation, consider jumping to p. 53

7.12 Dropout for Linear Softmax Class.

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- We can simplify this:

$$\tilde{P}_{ensemble}\left(\mathrm{y}=y\,|\,\boldsymbol{v}\right)=\sqrt[2^n]{\prod_{\boldsymbol{d}\in\{0,1\}^n}P\left(\mathrm{y}=y\,|\,\boldsymbol{v};\boldsymbol{d}\right)}$$

$$=\sqrt[2^n]{\prod_{\boldsymbol{d}\in\{0,1\}^n}\mathrm{softmax}\left(\boldsymbol{W}^T\left(\boldsymbol{d}\odot\boldsymbol{v}\right)+\boldsymbol{b}\right)_y}$$

$$=\sqrt[2^n]{\prod_{\boldsymbol{d}\in\{0,1\}^n}\frac{\exp\left(\boldsymbol{W}_{y,:}{}^T\left(\boldsymbol{d}\odot\boldsymbol{v}\right)+b_y\right)}{\sum_{y'}\exp\left(\boldsymbol{W}_{y',:}{}^T\left(\boldsymbol{d}\odot\boldsymbol{v}\right)+b_{y'}\right)}}$$

$$=\frac{\sqrt[2^n]{\prod_{\boldsymbol{d}\in\{0,1\}^n}\exp\left(\boldsymbol{W}_{y,:}{}^T\left(\boldsymbol{d}\odot\boldsymbol{v}\right)+b_y\right)}}{\sqrt[2^n]{\prod_{\boldsymbol{d}\in\{0,1\}^n}\sum_{y'}\exp\left(\boldsymbol{W}_{y',:}{}^T\left(\boldsymbol{d}\odot\boldsymbol{v}\right)+b_{y'}\right)}}$$

7.12 Dropout for Linear Softmax Class.

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Because $\tilde{P}$ will be normalized, we can safely ignore multiplication by factors that are constant with resp. to y, dropping the denominator:

$$\tilde{P}_{ensemble}(\mathrm{y} = y \mid \boldsymbol{v}) \propto \sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} \exp\left(\boldsymbol{W}_{y,:}^T (\boldsymbol{d} \odot \boldsymbol{v}) + b_y\right)}$$

$$= \exp\left(\frac{1}{2^n} \sum_{\boldsymbol{d} \in \{0,1\}^n} \boldsymbol{W}_{y,:}^T (\boldsymbol{d} \odot \boldsymbol{v}) + b_y\right)$$

$$= \exp\left(\frac{1}{2} \boldsymbol{W}_{y,:}^T \boldsymbol{v} + b_y\right)$$

- Substituting this back into the equation for $p_{ensemble}$ we obtain a softmax classifier with weights $\frac{1}{2}\boldsymbol{W}$.

7.12 Dropout

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

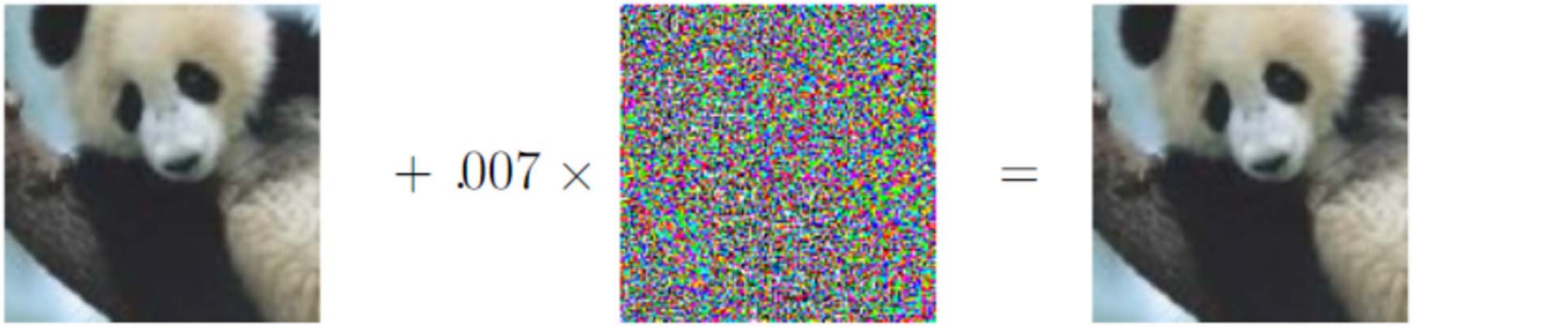
- The weight scaling rule is only an approximation for deep models that have nonlinearities.
- Though the approximation has not been theoretically characterized, it often works well, empirically.
- Srivastava *et al.* (2014) showed that dropout is more effective than weight decay, filter norm constraints and sparse activity regularization.
- Dropout does not significantly limit the type of model or training procedure that can be used.
- For very large datasets, dropout may not be needed.
- A large portion of the power of dropout arises from the fact that the masking noise is applied to the hidden units, not only to the inputs.

7.13 Adversarial Training

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- In order to probe the level of understanding a network has of the underlying task, we can search for examples that the model misclassifies.
- Szegedy *et al.* (2014b) found that even neural networks that perform at human level accuracy have a high error rate on examples that are constructed by using an optimizer to search for an input $\boldsymbol{x}'$ near a data point $\boldsymbol{x}$ such that the model output is very different at $\boldsymbol{x}$'.
- In many cases, $\boldsymbol{x}$' can be so similar to $\boldsymbol{x}$ that a human observer cannot tell the difference between the original example and the adversarial example, but the network can make highly different predictions.

7.13 Adversarial Training



- Fig. 7.8: Adversarial example generation applied to GoogLeNet on ImageNet data. By adding a very small vector we can change the classification of the image.

7.13 Adversarial Training

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Goodfellow *et al.* (2014b) showed that one of the primary causes of adversarial examples is excessive linearity.
- NNs are built out of primarily linear building blocks. These linear functions are easy to optimize.
- Unfortunately, the value of a linear function can change very rapidly if it has many inputs. If we change each input by ϵ, then a linear function with weights $\boldsymbol{w}$ can change by as much as $\epsilon \|\boldsymbol{w}\|_1$, which can be very large, if $\boldsymbol{w}$ is high-dimensional.
- Adversarial training discourages this highly sensitive locally linear behavior by encouraging the network to be locally constant in the neighborhood of the training data.

7.14 Tangent Distance, Tangent Prop, and Manifold Tangent Classifier

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Will not be covered in this lecture