# Feed-Forward Networks

Vladimir Feinberg

July 7, 2017

## 1 Introduction

Many-layer feed-forward networks, or deep neural networks (DNNs), are the simplest type of artificial neural network. For a given network architecture, a feed-forward network forms a family of possibly nonlinear functions. This family defines a hypothesis class for empirical risk minimization. These networks have been incredibly successful in supervised ML settings.

These notes follow Goodfellow's Deep Learning Book. As mentioned there, there are two essential components to DNNs:

1. Expressiveness through depth: iterative compositions of linear functions followed by activations can efficiently model mostly continuous functions (specifics below).

2. Tractable learning: compositions of simple functions as a graph allow for linear-time evaluation, and, critically, linear-time gradients via back-propagation.

Interestingly, these two frequently-cited facts give rise to two perplexing questions about DNNs, which don't seem to yet have a good answer:

1. If DNNs are so expressive, then isn't their VC dimension extremely large? Wouldn't this make them extremely prone to overfitting, to the point that they would be unusable? Perhaps DNNs can model "real distributions" well, so the distribution-independent analysis of VC-based statistical learning theory is insufficient. How can we justify the previous sentence? (Zhang et al 2017, Krueger et al 2017)

2. Why exactly is learning so tractable? Efficient derivatives aren't good enough in a non-convex enviornment: we know we'll reach a local min, but what makes a local min good? (Lee et al 2016, Jin et al 2017, Choromanska et al 2014)

## 2 Definition

A vanilla multi-layer perceptron (MLP) $f$ of depth $d$ is a a composition of $d$ vector-valued functions:

$$f(\mathbf{x}) = f^{(d)} \circ f^{(d-1)} \circ \cdots \circ f^{(2)} \circ f^{(1)} (\mathbf{x})$$

Note that the input and output dimension of each $f^{(i)}$ may differ; the maximum such dimension is the width of the network. For MLPs, each $f$ is the composition of a vectorized activation function $g$ and an affine function. The dimensionality of the affine functions and the choice of activation are hyperparameters.

$$f^{(i)}(\mathbf{x}) = g\left(W^{(i)}\mathbf{x} + \mathbf{b}^{(i)}\right)$$

$W^{(i)}, \mathbf{b}^{(i)}$ define parameters for $f$. The activation $g$ is usually a parameter-free (but perhaps not hyperparameter-free) nonlinearity. The MLP differs from other DNNs in that it is fully connected (all $W^{(i)}$ are unconstrained).

A network whose $W^{(i)}$ matrices correspond to convolutions is a CNN, for instance, and refers to a much more restricted family of functions.

A cost function $J(\boldsymbol{\theta})$ must be defined over the vector of aggregate parameters $\boldsymbol{\theta}$. MLE informs the shape of $J$ in most cases: for $(\mathbf{x}, \mathbf{y})$ distributed according to some data distribution, the log likelihood is the cross entropy $\mathbb{E}\, p_{\text{model}}(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})$. With $p_{\text{model}}(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) = N(f_{\boldsymbol{\theta}}(\mathbf{x}), I)$, for instance, MLE is equivalent to minimizing MSE. This kind of approach has the DNN specify the parameters for an output distribution, instead of predicting the output directly: this enables us to encode additional statistical knowledge about the inputs.

Training is typically performed with first-order optimization of $J(\boldsymbol{\theta})$; this leverages back-propagation for efficient derivatives.

## 3 Activations

**ReLU** (Glorot et al 2011). The ReLU $g(\mathbf{x}) = \max(\mathbf{x}, \mathbf{0})$ induces activation sparsity, which is more biologically plausible. Their use improves accuracy when compared to smoother activations in practice, and their linear regime keeps magnitude scaling behavior reasonable as we traverse the net, which is a boon to optimization.

**Maxout** (Goodfellow et al 2013, Goodfellow et al). A maxout layer is a generalized ReLU $g$ with input dimension $n$ and fixed $p$ and $k$ as hyperparameters has fixed subsets $G^{(i)} = \{ik + j \mid j \in [p]\}$ such that $g(\mathbf{z})_i = \max_{j \in G^{(i)}} z_j$. In addition to being more theoretically expressive (allowing for fewer units in the next layer), maxout enables units to be driven by several “filters” (activations in the previous layer), preventing catastrophic forgetting. Moreover, maxout is complimentary to dropout.

## 4 Regularization

At a high level, regularization changes the objective from usual loss $J(\boldsymbol{\theta})$ to loss plus weight penalty $J(\boldsymbol{\theta}) + \Omega(\boldsymbol{\theta})$ or loss plus constraints; it is effective in moving overfit models of high capacity to those of appropriate capacity for the problem: lower variance for a bit higher bias.

**Weight Decay**. For all affine transforms let the weight matrix terms (not bias), vectorized, be $\mathbf{w}$. Add $\Omega(\boldsymbol{\theta}) = \frac{\alpha}{2}\|\mathbf{w}\|_2^2$. Intuitively, bias is unrestricted because it only interacts with one variable directly (the output feature map), so it isn't prone to overfitting. For a quadratic approximation of the loss at $J(\boldsymbol{\theta})$, with hessian $H = Q\Lambda Q^\top$, $L^2$ regularization would result in a new optimum $Q(\Lambda + \alpha I)^{-1}\Lambda Q^\top \mathbf{w}_*$ where $\mathbf{w}_*$ is the unconstrained optimum. If $\mathbf{v}_i$ is the $i$-th eigenvector of $H$ and $\lambda_i$ its eigenvalue, then the regularization adjusts the solution by rescaling the $\mathbf{v}_i$-th component of $\mathbf{w}_*$ by a factor of $\frac{\lambda_i}{\lambda_i + \alpha}$. Regularization can also be viewed as constraints by identifying that $\alpha$ is a KKT multiplier: a fixed $\alpha$ corresponds to optimizing $\min J(\boldsymbol{\theta})$ such fthat $\|\mathbf{w}\| \le k$ for some $k$, which is in practice difficult to find given $\alpha$ (Fig. 1). $L^2$ regularization is equivalent to Bayesian MAP if weights have $N(0, \alpha^{-1})$ prior.

**Proximal Regularization**. Similar to weight decay, but $\Omega(\boldsymbol{\theta}) = \alpha\|\mathbf{w}\|_1$. Assuming a (convex) diagonal Hessian approximation to $J(\boldsymbol{\theta})$ (more strict than above), the $i$-th component solution is pulled down to

$$w_i = \begin{cases} \left(|w_i| - \frac{\alpha}{H_{i,i}}\right) \operatorname{sgn} w_i & H_{i,i} > 0, |w_i| > \frac{\alpha}{H_{i,i}} \\ 0 & \text{otherwise} \end{cases}$$

$L^1$ regularization is equivalent to Bayesian MAP if weights have $\text{Laplace}(0, \alpha^{-1})$ prior.

**Dataset Augmentation and Manifold Learning**. Modify supervised pairs $(\mathbf{x}, y)$ to include $(T(\mathbf{x}), y)$ where $T$ is a transformation we expect the classifier to be invariant to. Noise injection into inputs, too can be a form of dataset augmentation. An automatic technique for doing this is tangent propagation

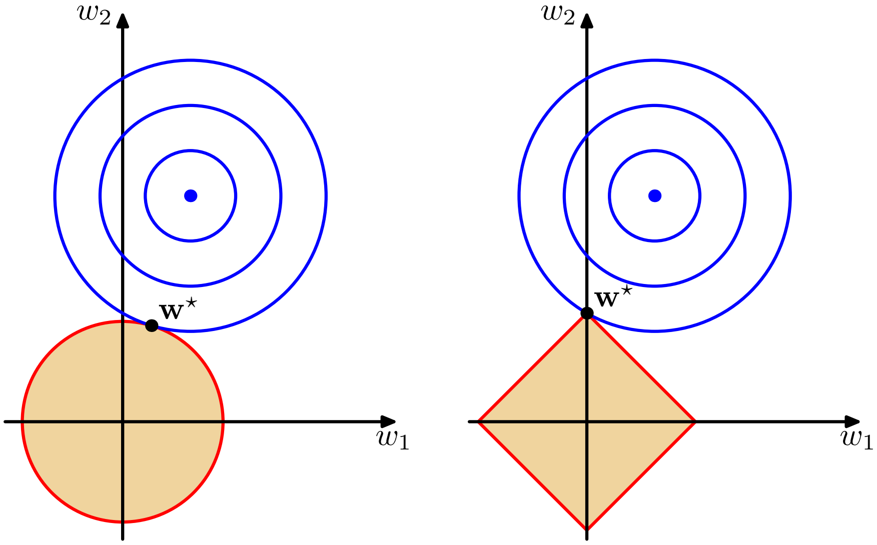


Figure 1: Regularization as constrained optimization: the above shows the contour plots of a quadratic objective, and $L^2, L^1$ regularizations, respectively, as constraints constructed by weighing the objective against the shape of the regularizer's ball. Image from Bishop's Pattern Recognition and Machine Learning, Chapter 3, page 146.

(Rifai et al 2011), which adds a penalty if the output is sensitive to changes along learned manifold vectors $\mathbf{v}_i$: $\Omega(f) = \sum_i \left(\mathbf{v}_i^\top \nabla f\right)^2$.

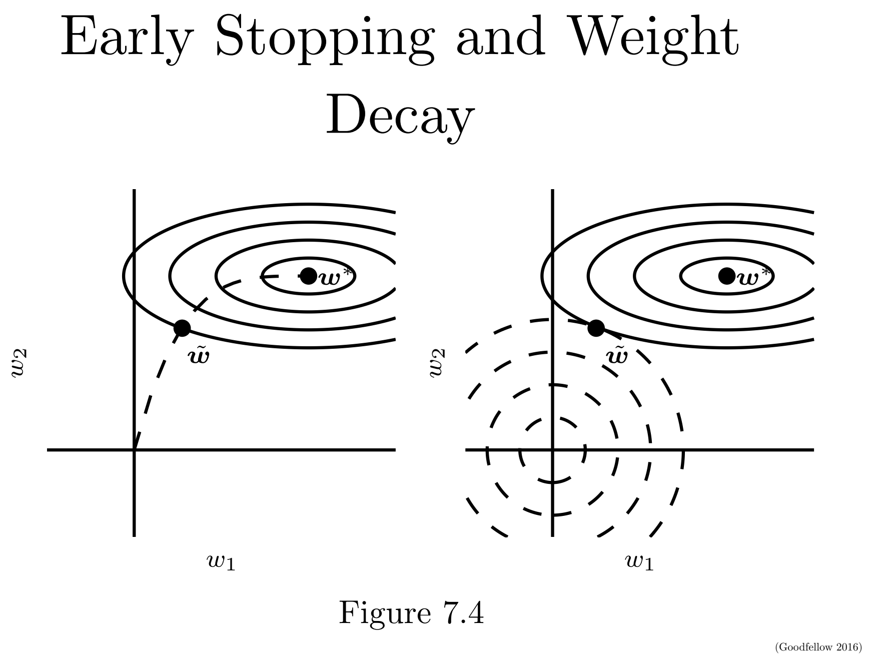
**Noise Robustness** (Graves 2011). Inject noise into weights during training, forcing learned weights to be robust to small variations.

**Early Stopping**. Stop training if validation increases; even if training is unconverged. If validation set is large, you miss out on a lot of data. More complicated algorithms are necessary to find an applicable early stopping time when re-training to include validation set (see Goodfellow's DL book, Algorithm 7.3). Early stopping is similar to $L^2$ regularization, and for quadratic loss it is just that (Fig. 2).

**Parameter Sharing**. Reuse part of a network for another task, encoding prior knowledge that the feature maps must be the same between two tasks. CNNs, below, are a form of parameter sharing, since the linear matrix $W$ is restricted to a convolution transform.

**Ensemble Methods**. Bagging uses votes of several models: $k$ uncorrelated models will tend to have $^1/_k$-th generalization error when bagged. Training data can be resampled to create more uncorrelated models.

**Dropout** (Hinton et al 2012, Wang et al 2013, Srivastava 2014, Gal et al 2015). Dropout is a cheap, practical approximation to averaging exponentially many models (exponential in the number of dropout layers). Dropout at a certain hidden or input layer wp $p$ drops an activation by setting it to zero during training and multiplying all activations (or the weights) by $p$ with no drops during testing (the weight scaling inference rule); theoretically this does not guarantee a proper output activation averaging, only individual hidden unit output expectation is preserved, but this works well in practice (Fig. 3). Weight scaling actually corresponds to taking a geometric means of the votes, and has been showed to work better than Monte Carlo sub-network averaging (which is intractable). First, this prevents feature co-adaptation: since a given activation might



Early Stopping and Weight Decay

Figure 7.4

(Goodfellow 2016)

Figure 2: From Dr. Goodfellow's DL book lecture, Chapter 7.

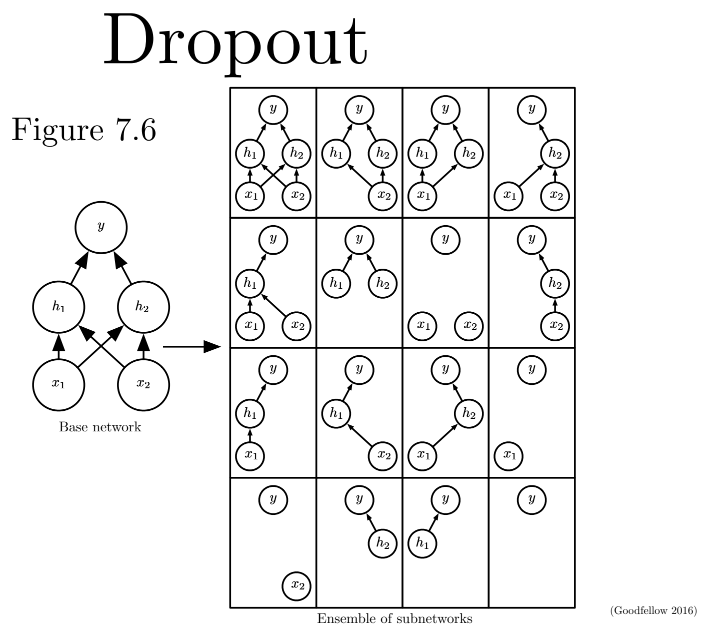
vanish at any time during training, the net is forced to learn those activations at each layer which independently contribute to the loss minimization task. Note that this conflicts with locality-focused tasks which rely on specific co-adaptations, like convolutional layers, but might be used at higher levels in CNNs to force the networks to identify more than one underlying feature for their classification (though lower levels can be locally co-adapted). Naive Bayes is an extreme form of dropout which demonstrates the co-adaptation avoidance.

**Adversarial Training** (Goodfellow 2014). Tiny changes in input can result in large differences in output, showing that the learned manifold for a classification category is too sensitive to perturbations in directions that it shouldn't be. Adversarial training forces nets to overcome this.

## 5 Theory

**Universal Approximation Theorem**. A feed-forward network with a linear output layer, at least one hidden layer, and the logistic sigmoid activation can approximate any Borel measurable function and its derivatives, where they exist, arbitrarily well, given sufficient hidden units.

**Deep Rectifier Network Efficiency Theorem** (Montúfar et al 2014). A deep rectifier network of constant width $w$, input dimension $i$ and depth $d$ may have up to $\Omega\left((w/i)^{di}\, i^i\right)$ distinct linear regions and a maxout network with $k$ filters per unit may have up to $\Omega(k^{i+l-1})$. Visually demonstrated in Figure 4.



Dropout

Figure 7.6

(Goodfellow 2016)

Figure 3: From Dr. Goodfellow's DL book lecture, Chapter 7.

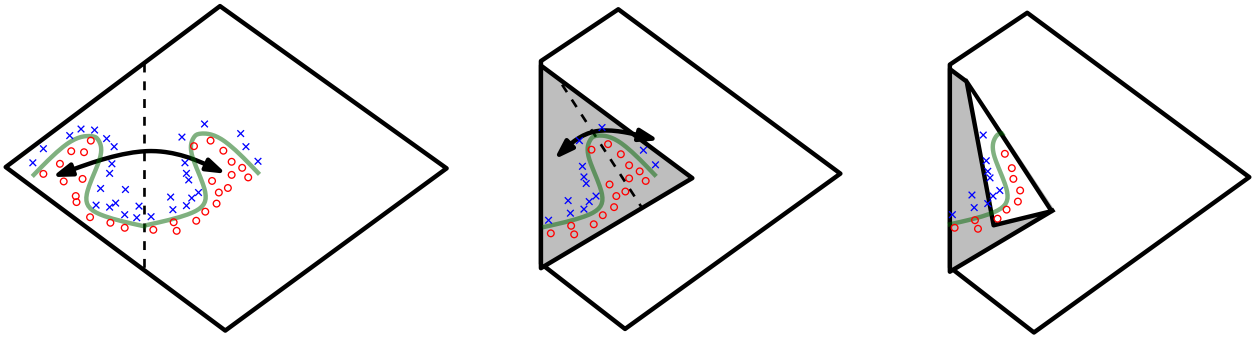


Figure 4: Visual demonstration of how composing an absolute value rectifier, which in effect allows for a higher-level function to be mirrored over some hyperplane in the input space, can result in exponentially many different linear regions by relying on (learned) symmetry.