Optimization on Deep Learning

Santiago VELASCO-FORERO
http://cmm.ensmp.fr/ velasco/

MINES ParisTech
PSL Research University
Center for Mathematical Morphology

Contents

Contents

Contents

Stochastic Gradient Descent (SGD) update

- **Require**: Learning rate ϵ (or a learning rate schedule)
- **Initialization**: $k = 1$, θ_1 some random value.
- **while** stopping criterion not met **do**
- **Sample** a value from the training set $(x^{(0)}, y^{(0)})$
- **Gradient** $\hat{g}(\theta_i) = \frac{\partial L(f(x^{(0)}, \theta_i), y^{(0)})}{\partial \theta}$
- **Update** $\theta_{i+1} = \theta_i - \epsilon \hat{g}(\theta_i)$
- $k = k + 1$

MiniBatch Stochastic Gradient Descent (MSGD) update

- **Require**: Learning rate ϵ (or a learning rate schedule)
- **Initialization**: $k = 1$, θ_1 some random value.
- **while** stopping criterion not met **do**
- **Sample** m examples from the training set $\{(x^{(0)}, y^{(0)}), (x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$
- **Gradient** $\hat{g}(\theta_i) = \frac{1}{m} \frac{\partial \sum_{i=0}^{m} L(f(x^{(i)}, \theta_i), y^{(i)})}{\partial \theta}$
- **Update** $\theta_{i+1} = \theta_i - \epsilon \hat{g}(\theta_i)$
- $k = k + 1$

Of $m = n$ then we get the BSGD.

Learning Rate

INCLUDE FIGURE

Contents

Table: Optimisers

Method	Update
SGD	$\theta_{i+1} = \theta_i - \eta \hat{g}(\theta_i)$
Momentum	$\theta_{i+1} = \theta_i + v_i,\ v_i = \alpha v_{i-1} - \eta \hat{g}(\theta_i)$
Nesterov Mom.	$\theta_{i+1} = \theta_i + v_i\ ,\ v_i = \alpha v_{i-1} - \eta \hat{g}(\theta_i + \alpha v_{i-1})$
Adagrad	$\theta_{i+1} = \theta_i - \frac{\eta}{\sqrt{G_{ii}+\epsilon}} \hat{g}(\theta_i)$

where G_{ii} is the sum of the squares of the gradients of θ_i up to time step i while ϵ is a smoothing term that avoids division by zero.

From Adagrad to Adadelta

In the Adagrad formulation,

$$\theta_{i+1} = \theta_i - \frac{\eta}{\sqrt{G_{ii} + \epsilon}} \hat{g}(\theta_i) \tag{1}$$

Accordingly, RMSprop (G. Hinton) proposes a rule update model's parameter by

$$\theta_{i+1} = \theta_i - \frac{\eta}{\sqrt{\mathtt{RMS}(g)_i + \epsilon}} \hat{g}(\theta_i) \tag{2}$$

However, the unit in the learning rate don't correspondent with denominator. Thus, Adadelta optimiser update by:

$$\theta_{i+1} = \theta_i - \frac{\mathtt{RMS}(\partial\theta)_{i-1}}{\sqrt{\mathtt{RMS}(g)_i + \epsilon}} \hat{g}(\theta_i) \tag{3}$$

Table: Optimisers

Method	Update
SGD	$\theta_{i+1} = \theta_i - \eta \hat{g}(\theta_i)$
Momentum	$\theta_{i+1} = \theta_i + v_i,\ v_i = \alpha v_{i-1} - \eta \hat{g}(\theta_i)$
Nesterov Mom.	$\theta_{i+1} = \theta_i + v_i\ ,\ v_i = \alpha v_{i-1} - \eta \hat{g}(\theta_i + \alpha v_{i-1})$
Adagrad	$\theta_{i+1} = \theta_i - \frac{\eta}{\sqrt{G_{ii}+\epsilon}} \hat{g}(\theta_i)$
RMSprop	$\theta_{i+1} = \theta_i - \frac{\eta}{\sqrt{\mathtt{RMS}(g)_i+\epsilon}} \hat{g}(\theta_i)$
Adadelta	$\theta_{i+1} = \theta_i - \frac{\mathtt{RMS}(\partial\theta)_{i-1}}{\sqrt{\mathtt{RMS}(g)_i+\epsilon}} \hat{g}(\theta_i)$

where G_{ii} is the sum of the squares of the gradients of θ_i up to time step i while ϵ is a smoothing term that avoids division by zero. $\mathtt{RMS}$ is the root mean squared error (RMS) of gradient for $\mathtt{RMS}(g)_i$ or of parameter updates for $\mathtt{RMS}(\partial\theta)_i$

Momentum

The momentum algorithm accumulates the exponentially decaying moving average of past gradients (called as velocity) and uses it as the direction in which to take the next step. Momentum is given by mass times velocity, which is equal to velocity if we're using unit mass. The momentum update is given by:

Nesterov Momemtum

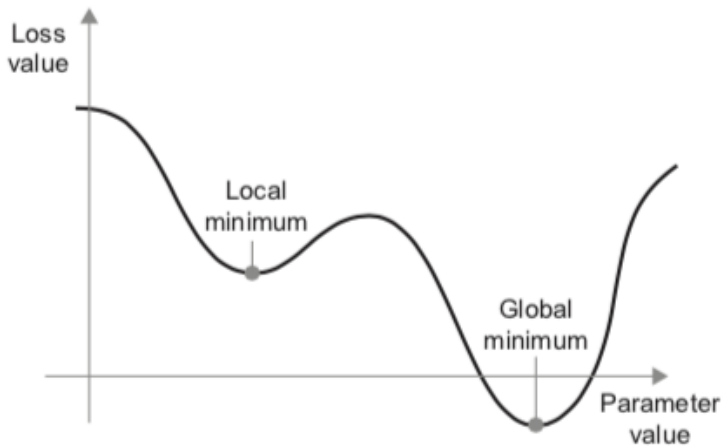
Thus, it might be better to compute the gradient from that point onward

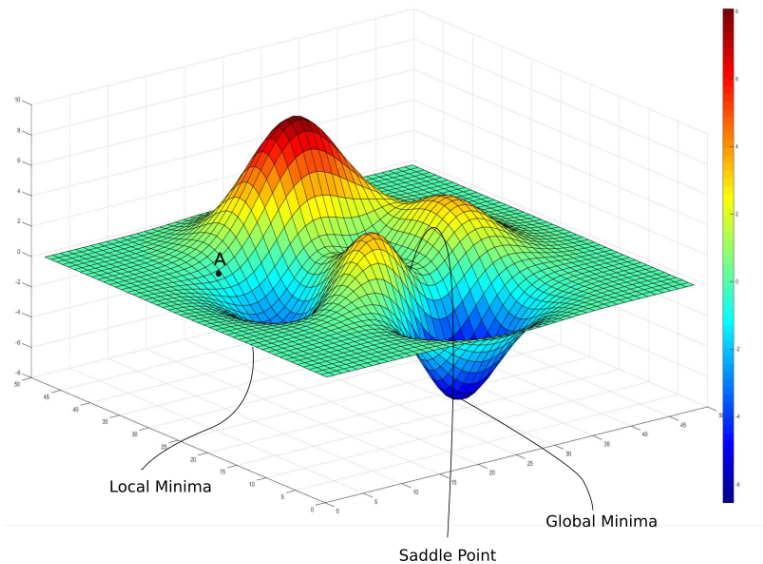
Contents

Difficulties in Deep Network Optimisation

A Local minima / Global minima

B Saddle Point (Plateaus or Flat Regions)





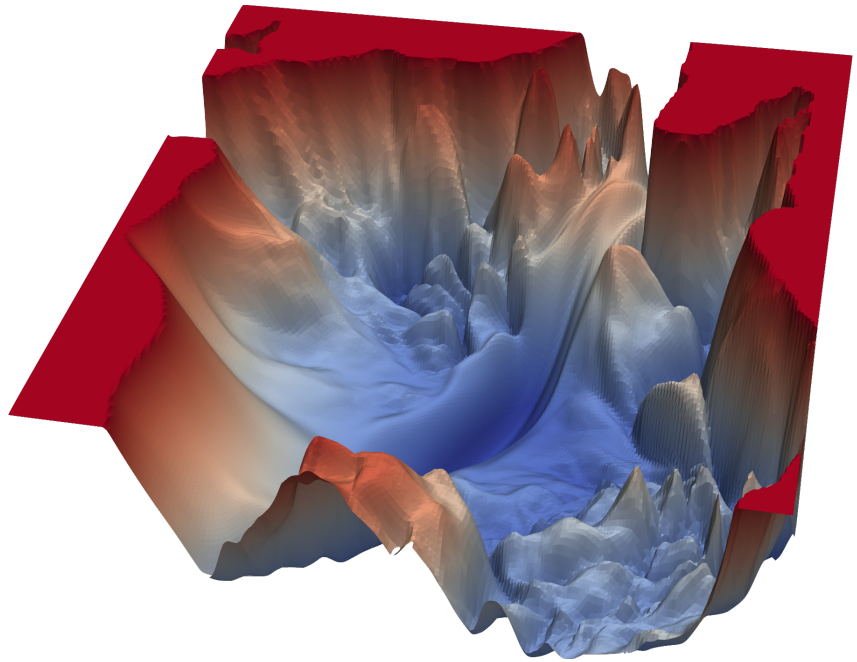


Figure: Loss Function for VGG56

Reference: **Visualizing the Loss Landscape of Neural Net**, Hao Li, Zheng Xu, Gavin Taylor, and Tom Goldstein, 2017

Difficulties in Deep Network Optimisation

- Vanishing Gradient: If feedback signal has to be propagated through a deep stack of layers, the signal may become tenuous or even be lost entirely, rendering the network untrainable. During training, it causes the model's parameter to grow so large so that even a very tiny amount change in the input can cause a great update in later layers' output. The value of layer weights sometimes it overflow and the value becomes NaN.

Fighting against Vanishing Gradient

1. Initialization of Weights: Don't initialize to values that are too large.
2. Gradient clipping: clips parameters gradients during backpropation by a maximum value or maximum norm

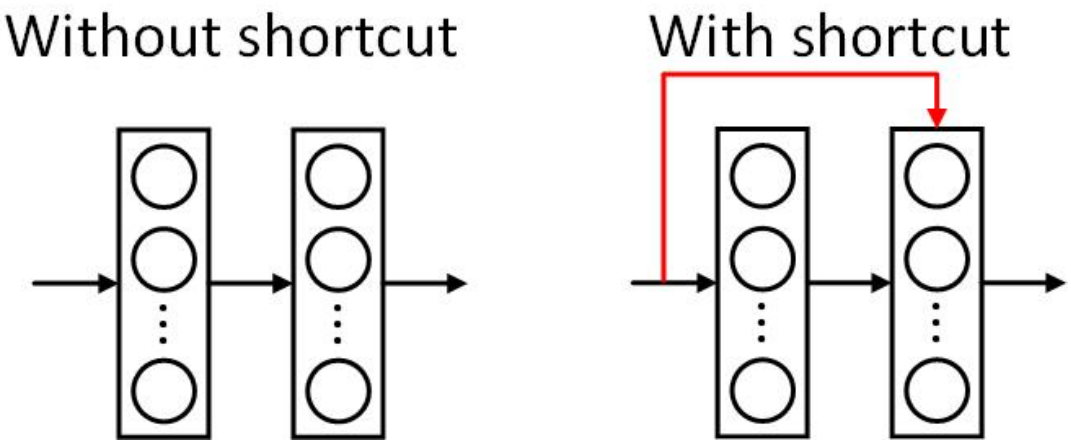
```
from keras import optimizers

# All parameter gradients will be clipped to
# a maximum value of 0.5 and
# a minimum value of -0.5.
sgd = optimizers.SGD(lr=0.01, clipvalue=0.5)

# All parameter gradients will be clipped to
# a maximum norm of 1.
sgd = optimizers.SGD(lr=0.01, clipnorm=1.)
```

Fighting against Vanishing Gradient

3 Skip connections or Shortcuts (Residual Networks):



Fighting against Vanishing Gradient

4 Avoid "stuck states" induced by activation function:

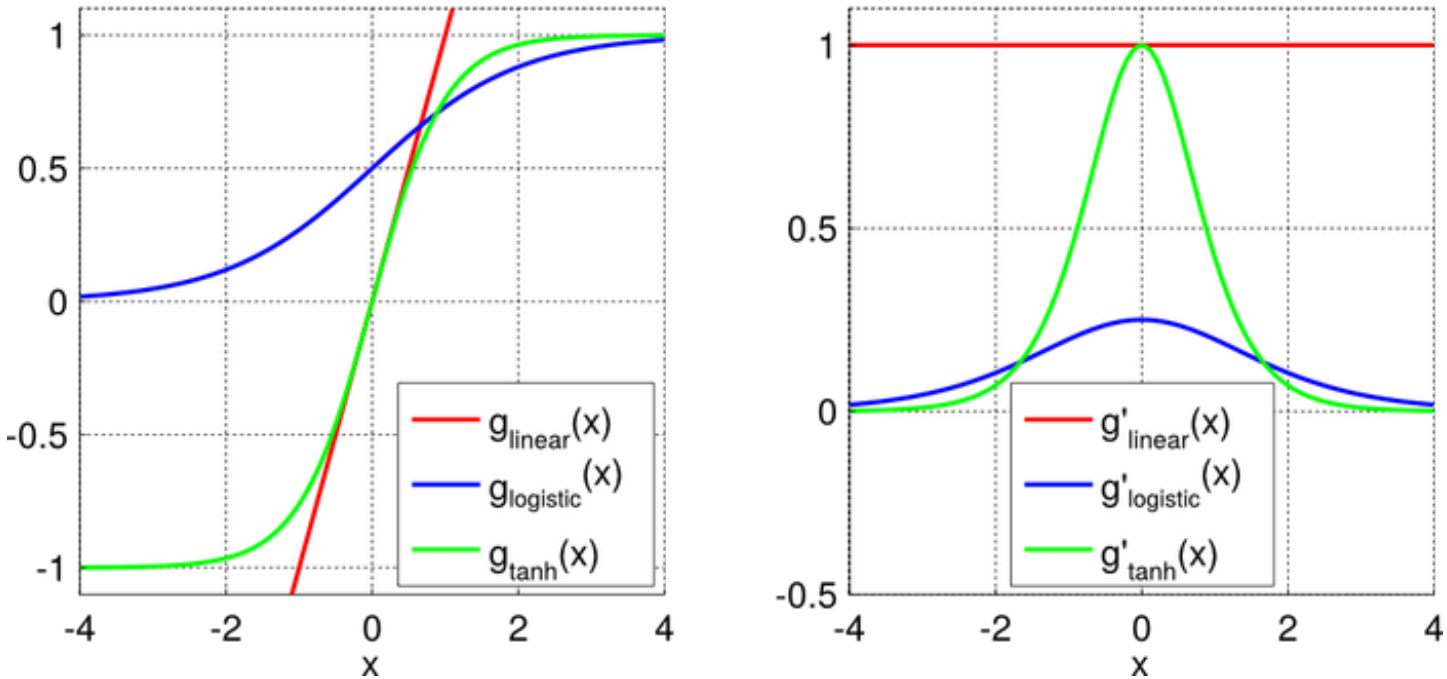


Figure: Left: Three activation function Right: Derivative of activation function.

Fighting against Vanishing Gradient

5 Regularization: L_2 or L_1 norm applies "weight decay" in the cost function of the network. Note that for many activation function, when the activation value is small, that will be almost linear.

```
from keras import regularizers
model.add(Dense(64, input_dim=64,
                kernel_regularizer=regularizers.l2(0.01),
                activity_regularizer=regularizers.l2(0.01)))
```

Batch Normalization

TODO: FORMULA, EXAMPLES

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2}}$$

$$BN_{\gamma,\beta}(x_i) := \gamma \hat{x}_i + \beta$$

where μ_B and σ_B^2 are respectively the mini-batch mean and variance. γ scale and β shift.

- In the original paper, BatchNorm is applied before the applying activation.
- Moving values to zero (activation works better!)
- If we use a high learning rate in a traditional neural network, then the gradients could explode or vanish. Large learning rates can scale the parameters which could amplify the gradients, thus leading to an explosion. But if we do batch normalization, small changes in parameter to one layer do not get propagated to other layers. This makes it possible to use higher learning rates for the optimizers, which otherwise would not have been possible. It also makes gradient propagation in

Supervised Learning (Machine Learning)

- **Data**: N observations $(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}, i = 1, \ldots, N$, **i.i.d.**
- **Model**: $\texttt{Model}(\mathbf{x}) := \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x})$ of features $\boldsymbol{\phi}(\mathbf{x}) \in \mathbb{R}^P$
  Prediction as linear mapping of features
- **Minimization of Regularized Empirical Risk**: We would like to find $\boldsymbol{\theta}^*$ the solution of:

$$\boldsymbol{\theta}^* := \min_{\boldsymbol{\theta} \in \mathbb{R}^P} \frac{1}{N} \sum_{i=1}^{N} L(y_i, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x})) \quad + \quad \alpha \mathcal{R}(\boldsymbol{\theta})$$

$$\texttt{Data fitting} \quad + \quad \texttt{regularizer}$$

where $L(\cdot, \cdot)$ is called the *loss function*.

Other loss functions, other models

1. Support Vector Machine (SVM): "Hinge" Loss

$$L(y, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x})) = \max\{1 - y\boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}), 0\} \tag{4}$$

2. Logistic Regression:

$$L(y, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x})) = \log\big(1 + \exp\big(-y\boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x})\big)\big) \tag{5}$$

3. Mean Squared Regression:

$$L(y, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x})) = \frac{1}{2}(y - \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}))^2 \tag{6}$$

4. Adaboost

$$L(y, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x})) = \exp^{-(y - \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}))} \tag{7}$$

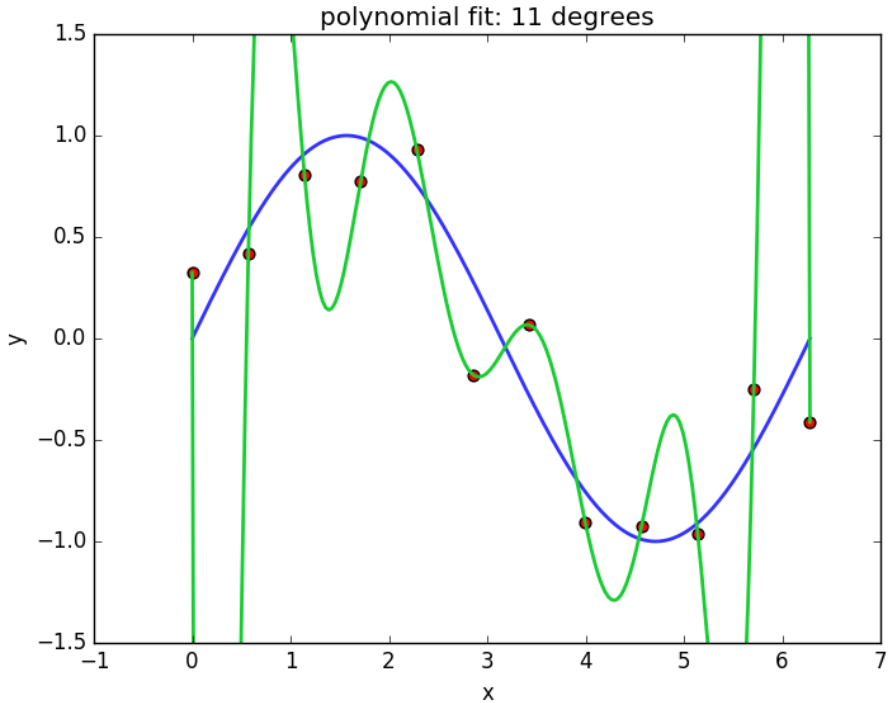
5. Others ...

Minimizing Empirial Risk = Problems!

- Empirical Risk: $\hat{f}(\boldsymbol{\theta}) := \frac{1}{N} \sum_{i=1}^{N} L(y_i, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}_i))$

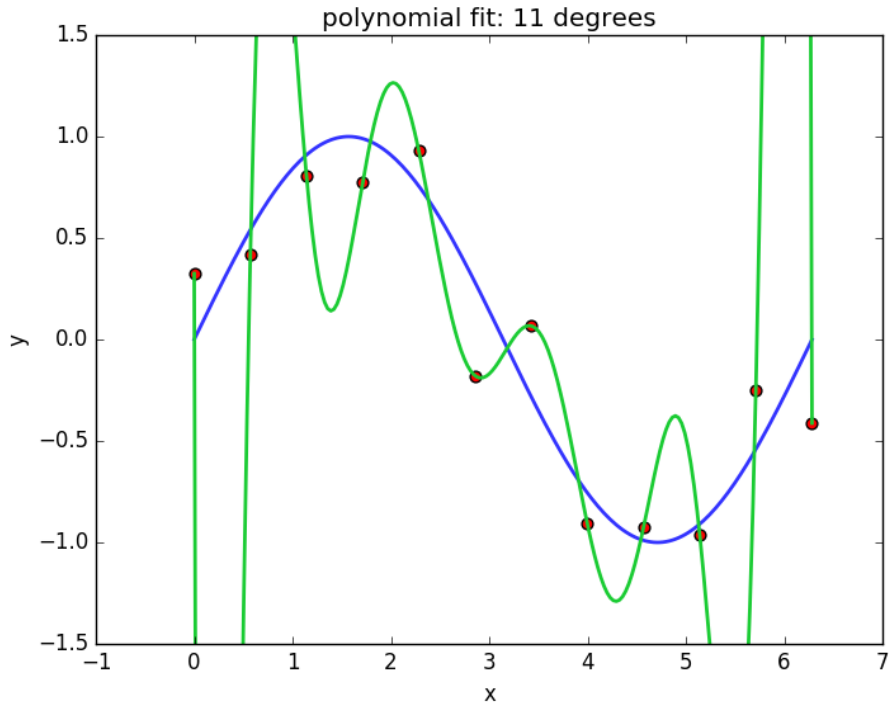
Minimizing Empirial Risk = Problems!

- Empirical Risk: $\hat{f}(\boldsymbol{\theta}) := \frac{1}{N} \sum_{i=1}^{N} L(y_i, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}_i))$
  Loss in a training set



Minimizing Empirial Risk = Problems!

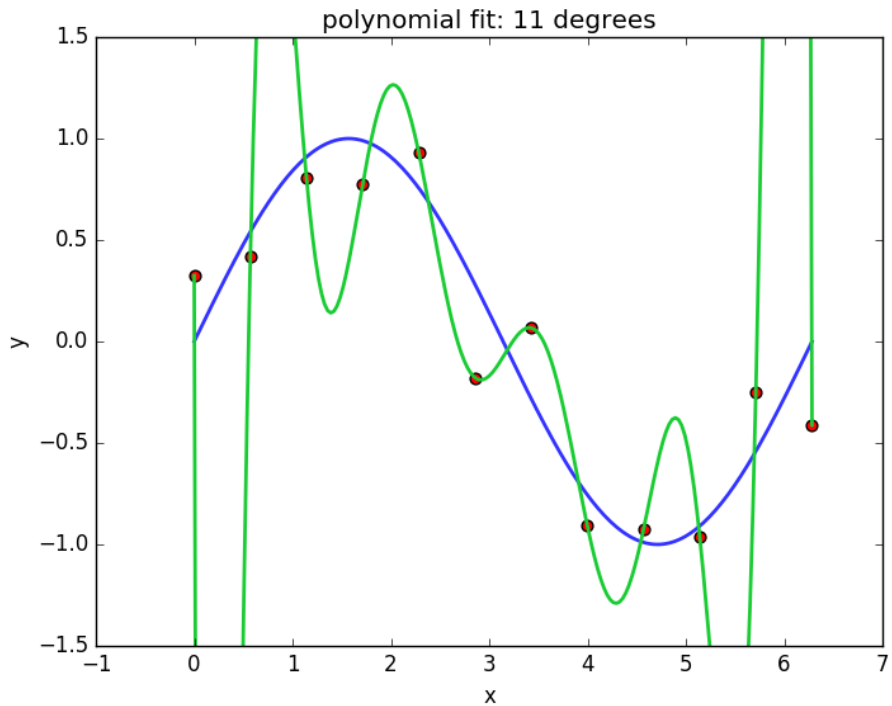
- Empirical Risk: $\hat{f}(\boldsymbol{\theta}) := \frac{1}{N} \sum_{i=1}^{N} L(y_i, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}_i))$
  Loss in a training set



- Expected Risk : $f(\boldsymbol{\theta}) := \mathbb{E}_{(\mathbf{x},y)} L(y, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}))$

Minimizing Empirial Risk = Problems!

- Empirical Risk: $\hat{f}(\boldsymbol{\theta}) := \frac{1}{N} \sum_{i=1}^{N} L(y_i, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}_i))$
  Loss in a training set



- Expected Risk : $f(\boldsymbol{\theta}) := \mathbb{E}_{(\mathbf{x},y)} L(y, \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}))$
  Loss in a testing set

There are infinity minimizers of the empirical risk, but most of them have a large expected risk (overfitting).

Contents

Bias/Variance Tradeoff

Let $\hat{y} := \texttt{Model}(\mathbf{x})$ the prediction of a deterministic model evaluated at $\mathbf{x}$

$$\mathbb{E}_{(\mathbf{x},y)}\left[(y - \texttt{Model}(\mathbf{x}))^2\right] =$$

$$Var\left[y\right] + Var\left[\texttt{Model}(\mathbf{x})\right] + \left(Bias[\texttt{Model}(\mathbf{x})]\right)^2$$

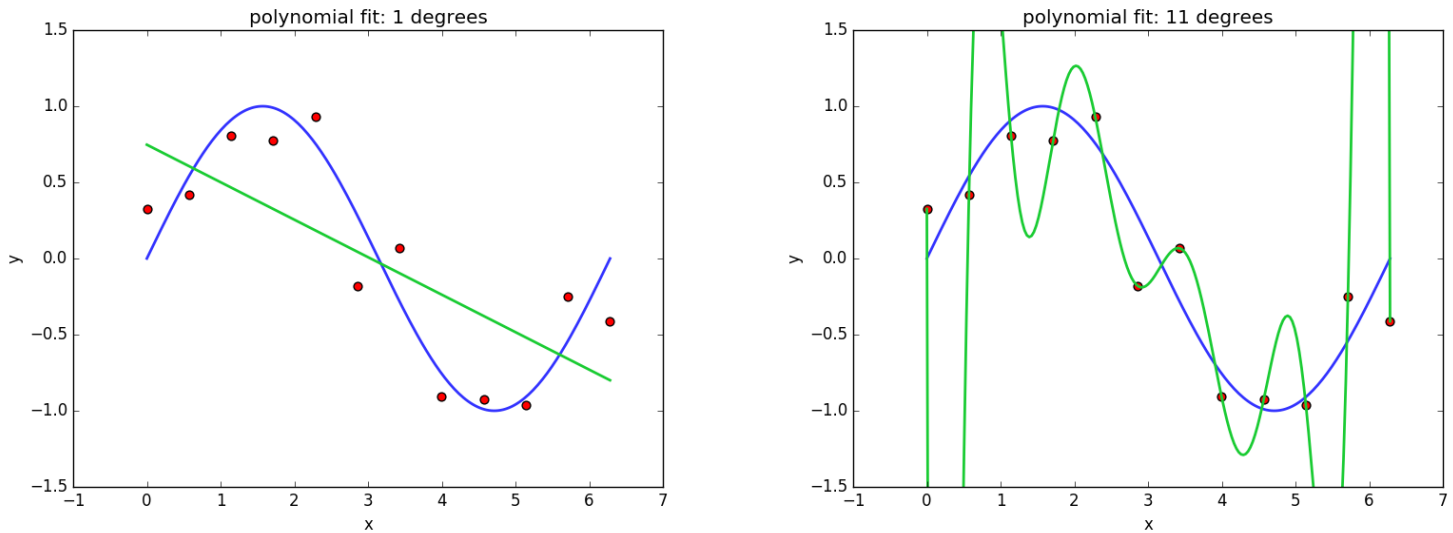
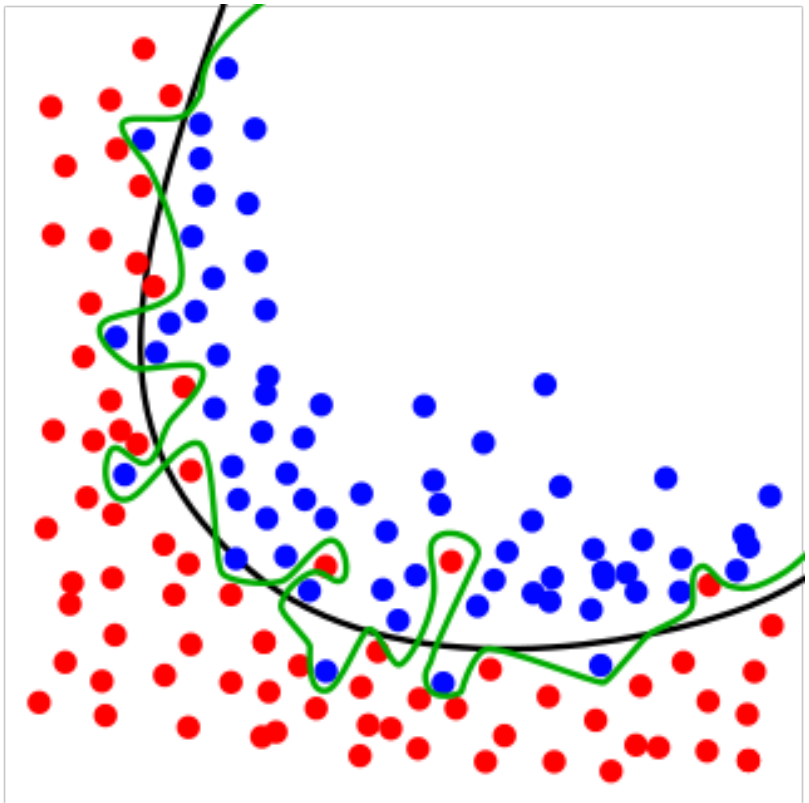


Figure: Underfitting / Overfitting

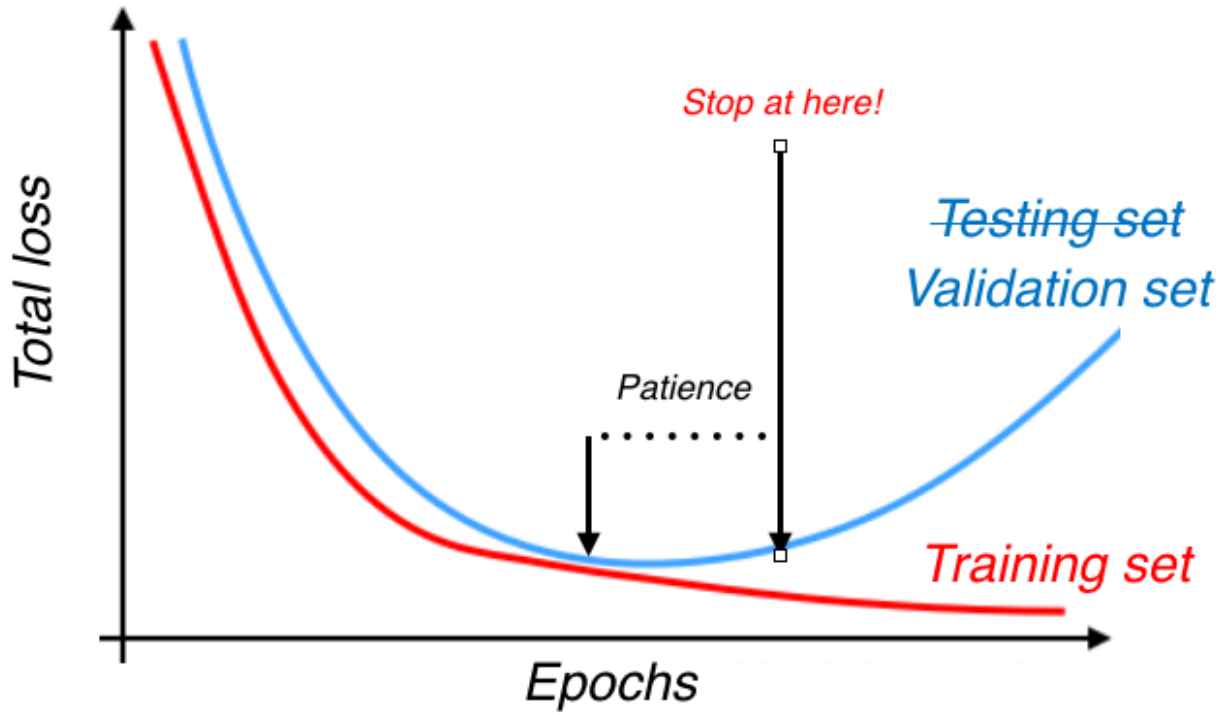
Underfitting : Prediction with less variance but more bias
Overfitting : Prediction with more variance but less bias

How to judge if a deep machine learning model is overfitting or not?



- Training Set / Testing Set
- Cross-Validation
- $||\boldsymbol{\theta}||_p$ is large

1. Early Stopping



TODO: Include KERAS CODE (BACKEND)

2. We need more data!

- Additive Gaussian noise.
- Data augmentation.
- Adversarial Training.

Expected Risk (again)

The expected risk

$$\mathbb{E}_{(\mathbf{x},y)} L(y, \texttt{Model}) = \int L(y, \texttt{Model}(\mathbf{x})) dP(\mathbf{x}, y) := R(\texttt{Model})$$

Expected Risk (again)

The expected risk

$$\mathbb{E}_{(\mathbf{x},y)} L(y, \texttt{Model}) = \int L(y, \texttt{Model}(\mathbf{x})) dP(\mathbf{x}, y) := R(\texttt{Model})$$

But the distribution P is unknown in most practical situations.

Expected Risk (again)

The expected risk

$$\mathbb{E}_{(\mathbf{x},y)} L(y, \texttt{Model}) = \int L(y, \texttt{Model}(\mathbf{x})) dP(\mathbf{x}, y) := R(\texttt{Model})$$

But the distribution P is unknown in most practical situations. We usually have access to a set of training data $T = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $(\mathbf{x}_i, y_i) \sim P$, for all $i = 1, \ldots, N$. Thus, we may approximate P by the *empirical distribution*:

$$P_\delta(\mathbf{x}, y) = \frac{1}{N} \sum_{i=1}^{N} \delta(\mathbf{x} = \mathbf{x}_i, y = y_i)$$

Empirical Risk (again)

Using the empirical distribution P_δ, we can now approximated the expected risk, by the called *empirical risk*

$$R_\delta(\texttt{Model}) = \int L(y, \texttt{Model}(\mathbf{x}))dP_\delta(\mathbf{x}, y) = \frac{1}{N}\sum_{i=1}^{N} L(\texttt{Model}(\mathbf{x}_i), y_i) \tag{8}$$

Learning the function f by minimizing (**??**) is known as the Empirical Risk Minimization (ERM) principle [**?**] (Vapnik, 1998). If the number of parameters are comparable to N, one trivial way to minimize (**??**) is to memorize the whole set of training data (overfitting).

Vicinal Risk Minimization (VRM)

P_δ is only one of the possibility to approximate the true distribution P. (Chapelle et al., 2000) proposed to approximate P by:

$$P_v(\mathbf{x}, y) = \frac{1}{N} \sum_{i=1}^{N} v(\tilde{\mathbf{x}}, \tilde{y}, |\mathbf{x}_i, y_i)$$

where v is *vicinity distribution* that measure the probability for a "virtual" pair $(\tilde{\mathbf{x}}, \tilde{y})$ to be in the *vicinity* of the training pair $(\mathbf{x}, y)$.

Vicinal Risk Minimization (VRM)

P_δ is only one of the possibility to approximate the true distribution P. (Chapelle et al., 2000) proposed to approximate P by:

$$P_v(\mathbf{x}, y) = \frac{1}{N} \sum_{i=1}^{N} v(\tilde{\mathbf{x}}, \tilde{y}, |\mathbf{x}_i, y_i)$$

where v is *vicinity distribution* that measure the probability for a "virtual" pair $(\tilde{\mathbf{x}}, \tilde{y})$ to be in the *vicinity* of the training pair $(\mathbf{x}, y)$.

1. Gaussian vicinities: $v(\tilde{\mathbf{x}}, \tilde{y}, |\mathbf{x}_i, y_i) = \mathcal{N}(\tilde{\mathbf{x}} - \mathbf{x}, \sigma^2 \mathbf{I}) \delta(\tilde{y} = y)$

Vicinal Risk Minimization (VRM)

P_δ is only one of the possibility to approximate the true distribution P. (Chapelle et al., 2000) proposed to approximate P by:

$$P_v(\mathbf{x}, y) = \frac{1}{N} \sum_{i=1}^{N} v(\tilde{\mathbf{x}}, \tilde{y}, |\mathbf{x}_i, y_i)$$

where v is *vicinity distribution* that measure the probability for a "virtual" pair $(\tilde{\mathbf{x}}, \tilde{y})$ to be in the *vicinity* of the training pair $(\mathbf{x}, y)$.

1. Gaussian vicinities: $v(\tilde{\mathbf{x}}, \tilde{y}, |\mathbf{x}_i, y_i) = \mathcal{N}(\tilde{\mathbf{x}} - \mathbf{x}, \sigma^2 \mathbf{I}) \delta(\tilde{y} = y)$

which is equivalent to augmenting the training data with additive Gaussian noise

TODO include KERAS CODE OF GAUSSIAN LAYER!

Why Data-augmentation?

2 Data-augmentation based vicinities:
$P_{agg}(\mathbf{x}, y) = \frac{1}{N} \sum_{i=1}^{N} \delta(\tilde{\mathbf{x}}, y_i)$, where $\tilde{\mathbf{x}}$ is a random transformation applied $\mathbf{x}$

Figure: Example of a set of image produce by random transformations (translations, rotations, zooming, ...)

TODO include KERAS CODE OF Data Augmentation!

Regularisation

1. "Weight Decay": Again?
2. Early Stopping
3. More data! : Data Augmentation
4. More data! : Adversarial Examples
5. Summing-up: Dropout

How can we reduce the variance of ?

- Remember: given N i.i.d. obsevations $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ each of them with variance equal to σ^2.
- What is the variance of $\bar{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i$?

How can we reduce the variance of ?

- Remember: given N i.i.d. obsevations $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ each of them with variance equal to σ^2.
- What is the variance of $\bar{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i$? $\frac{\sigma^2}{N}$
- Hint: Train model on different training sets, and use the mean of the prediction as final model of prediction.

Dropout

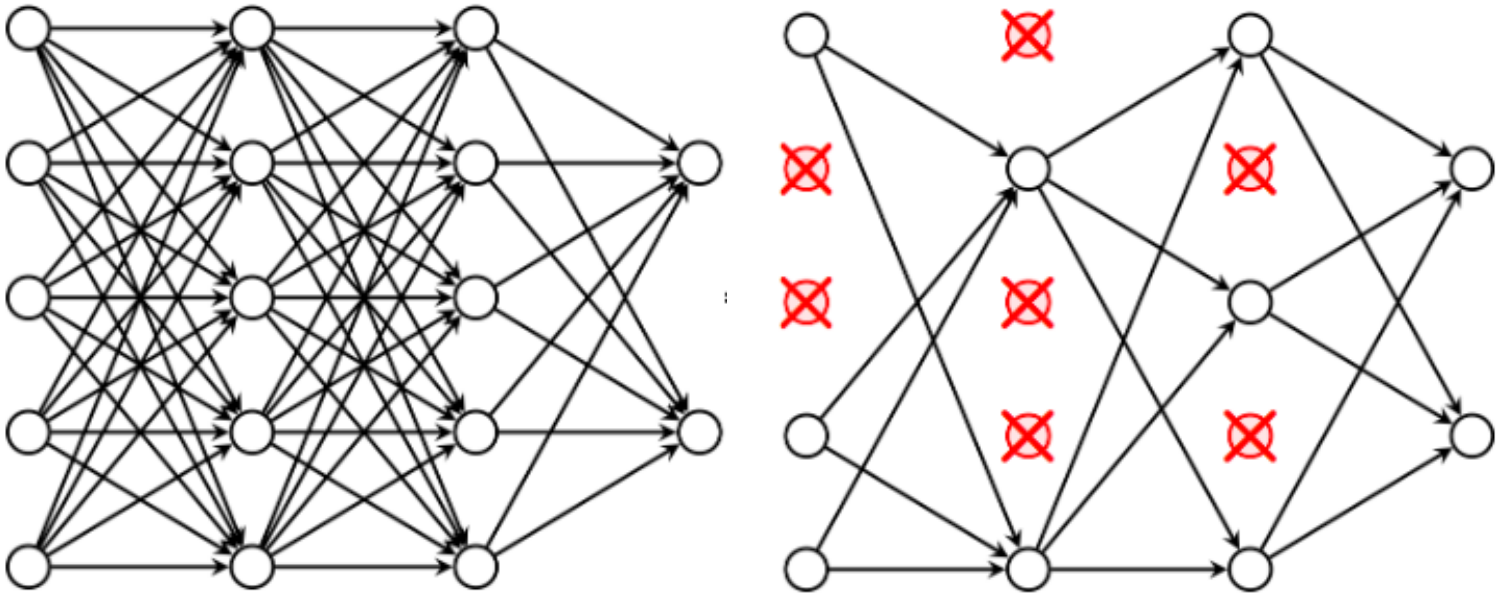


Figure: Left: No Dropout Right: Dropout

- Since dropout can be seen as a stochastic regularization technique
- Avoid memorization!
- Dropout forces to learn more robust features that are useful in conjunction with many different random subsets.
- With H hidden units, each of which can be dropped, we have 2^H possible models!

Dropout

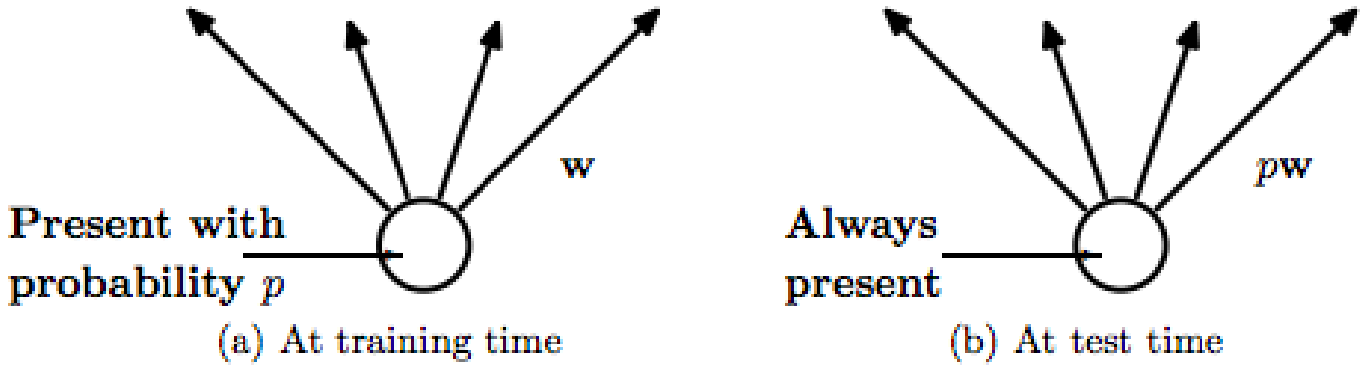


Figure: Left: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. Right: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

TODO include KERAS CODE OF Dropout!