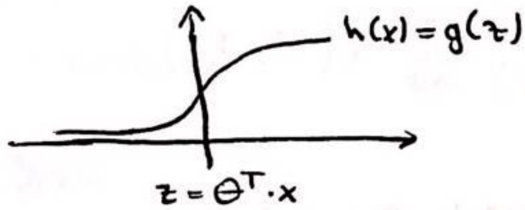
W7 - SVM Intro I -

- There is one more powerful and very widely used learning algorithm: Support Vector Machine (SVM): compared to both logistic regression and neural networks, SVM sometimes gives a cleaner and more powerful way of learning complex non-linear functions.

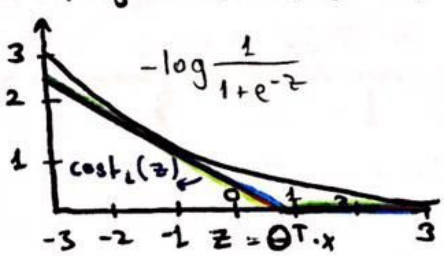
- Logistic Regression ile başlayıp değişiklikler ile SVM'i anlayacağız.

  - $h(x) = \dfrac{1}{1+e^{-\theta^T \cdot x}}$
  - 

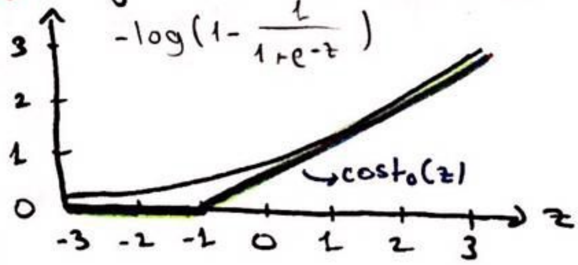
- If $y=1$, we want $h(x) \approx 1$, $\theta^T \cdot x \gg 0$ ki hipotez doğru çalışsın.
- If $y=0$, we want $h(x) \approx 0$, $\theta^T \cdot x \ll 0$

- Cost function tüm training ex.s ile hesaplanıyordu. Tek bir training ex.'in etkisine (cost'una) bakarsak:
  - $-(y \log h(x) + (1-y)\log(1-h(x)))$
  - $-y \log \dfrac{1}{1+e^{-\theta^T \cdot x}} - (1-y)\log\left(1-\dfrac{1}{1+e^{-\theta^T \cdot x}}\right)$

- If $y=1$ (want $\theta^T \cdot x \gg 0$)



- If $y=0$ (want $\theta^T x \ll 0$):



- $y=1$ ise z büyüdükçe ilgili training ex. cost'u düşüyor üstelik $h(x)$ grafiği de aynı şeyi söyler. z büyüdükçe $h(x) \approx 1$ olur. Cost düşer.

- SVM için başka bir cost function kullanacağız. Yukarıda mavi-kırmızı ile çizildi. It is a pretty close approximation of the cost function used by logistic regression. Slope çok önemli değil. Bu yeni cost function SVM'e computational advantages verir. Give us an easier optimization problem.

W7 - SVM Intro - II -

- Cost function for logistic regression:

$$\underbrace{\min_{\theta} \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}.\underbrace{\left(-\log h(x^{(i)})\right)}_{\substack{\text{SVM'in burası,}\\ cost_1(\Theta^T x^{(i)})}} + (1-y^{(i)})\underbrace{\left(\left(-\log(1-h(x^{(i)}))\right)\right)}_{\text{Burası da } cost_0(\Theta^T x^{(i)})}\right]}_{\text{Term A}} + \underbrace{\frac{\lambda}{2m}\sum_{j=1}^{n}\Theta_j^2}_{\text{Term B}}$$

- Cost Function for Support Vector Machine:

$$\min_{\theta} \cancel{\frac{1}{m}} \sum_{i=1}^{m} y^{(i)}.cost_1(\Theta^T.x^{(i)}) + (1-y^{(i)}).cost_0(\Theta^T.x^{(i)}) + \frac{\lambda}{2\cancel{m}}\sum_{j=1}^{n}\Theta_j^2$$

By convention we write things slightly different

- get rid of the $1/m$ terms. Sonuçta bu bir constant min Θ değişmez.
- $A + \lambda B$: Burada λ ile underfitting ($\lambda\uparrow$) vs overfitting ($\lambda\downarrow$) trade off'u kontrol ediliyordu. Şimdi convention olarak λ yerine C kullanılacak ve: $CA + B$ kontrol edilecek. λ ile ters etki yapacak ama mantık aynı! $C = \frac{1}{\lambda}$ gibi düşünebilirsin. → ise aynı min Θ bulunur!

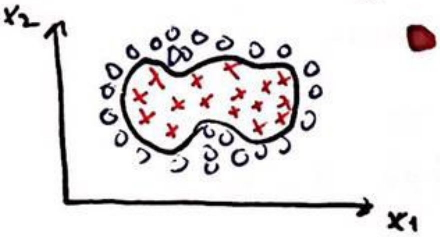
$$\boxed{\min_{\theta} C\sum_{i=1}^{m}\left[y^{(i)}.cost_1(\Theta^T.x^{(i)}) + (1-y^{(i)}).cost_0(\Theta^T.x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\Theta_j^2}$$

Unlike logistic regression, the SVM doesn't output a probability. It just makes a prediction of y being equal to one or zero, directly.

$$\boxed{h(x)\begin{cases} 1 & \text{if } \Theta^T.x \geq 0 \\ 0 & \text{otherwise}\end{cases}}$$

- We will start adapting SVMs in order to develop complex nonlinear classifiers. The main technique for doing that is something called kernels.

- If we have a training set as below and we have to find a nonlinear decision boundary to distinguish the positive and negative examples.



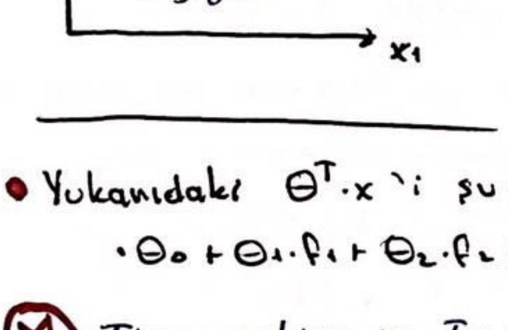
- One way to do so is complex polynomial features.
  - Predict $y=1$ if
  - $\Theta_0 + \Theta_1 x_1 + \Theta_2 x_2 + \Theta_3 x_1 x_2 + \Theta_4 x_1^2 + \Theta_5 x_2^2 + \dots \geq 0$
  - Predict $y=0$ otherwise

- Yukarıdaki $\Theta^T \cdot x$'i şu şekilde de yazabilirim:
  - $\Theta_0 + \Theta_1 \cdot f_1 + \Theta_2 \cdot f_2 + \Theta_3 \cdot f_3 + \dots$ where $f_1 = x_1, f_2 = x_2, f_3 = x_1 \cdot x_2, \dots$

The question is: Is there a better choice of features than high order polynomials? Because it is not clear that these high order polynomials are what we want, and we have seen that when n (feature #) is large (for ex for a computer vision problem) using high order polynomials become very computationally expensive. There will be too many features.

- So is there a better choice of these features $f_1, f_2, f_3, \dots$?

Here is a new Idea: KERNEL. Let's define only 3 new features:



- Given the training example x, compute the new feature depending on proximity to the related landmark $\ell^{(i)}$. $\|x - \ell^{(1)}\|$ → Eucledian distance b/w point x and $\ell^{(1)}$

→ $\boxed{\text{Given } x: f_1 = \text{similarity}(x, \ell^{(1)}) = \exp\left(-\frac{\|x-\ell^{(1)}\|^2}{2\sigma^2}\right)}$

→ $\boxed{\text{Given } x: f_2 = \text{similarity}(x, \ell^{(2)}) = \exp\left(-\frac{\|x-\ell^{(2)}\|^2}{2\sigma^2}\right)}$

→ $\boxed{\text{Given } x: f_3 = \text{similarity}(x, \ell^{(3)}) = \exp(\dots)}$

I am Ignoring x_0 for the purpose of these slides

The similarity function is called Kernel function and the specific Kernel I'm using here is a Gaussian Kernel. Başka bir kernel da kullanılabilirdi. Bu kullanılan Gaussian kernel. $k(x, \ell^{(i)})$

Her training ex için farklı bir feature vector olur. $(f_1, f_2, f_3, \dots)$

– W7 – Kernels I –

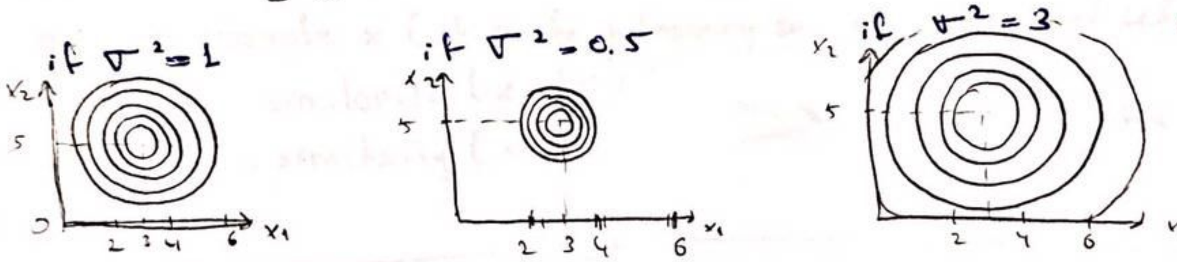
Kernels and Similarity:

- $f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{j=1}^{n}(x_j - l_j^{(1)})^2}{2\sigma^2}\right)$
- If $x \approx l^{(1)}$: $f_1 \approx \exp\left(-\frac{0^2}{2\sigma^2}\right) \approx 1$
- If x is far from $l^{(1)}$: $f_1 \approx \exp\left(-\frac{(\text{large number})^2}{2\sigma^2}\right) \approx 0$

- Each landmark defines a new feature: $l^{(1)} \to f_1$, $l^{(2)} \to f_2$, $l^{(3)} \to f_3$, ---
- Bize bir training ex verilince landmarks sayısı kadar feature bulabiliriz.

Let's plot and try to understand this exponential Gaussian Kernel.

Ex: $l^{(1)} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$, $f_1 = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$

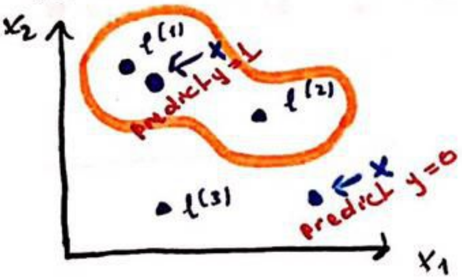


- σ küçüldükçe hızlı sivrileşen dik bir dağ, aksi halde düşük eğimli geniş bir dağ gibi düşün.

Let's see what sorts of hypotheses we can learn:

→ Let's say we have trained our algorithm and our hypothesis: $\boxed{\text{predict } 1}$ when $\boxed{\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0}$ where $\boxed{\theta_0 = -0.5, \theta_1 = 1, \theta_2 = 1, \theta_3 = 0}$

- Verilen training ex. için features f_1, f_2, f_3 hesaplanıp prediction yapılacak ve kabaca decision boundary çizilecek (intuition için)



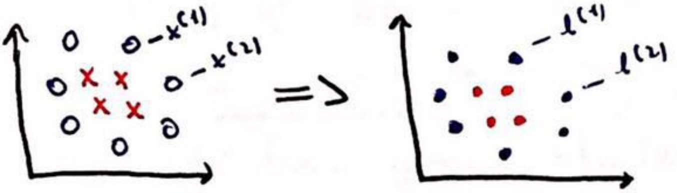
- Mor training ex. x baz alınarak:
  - $f_1 \approx 1$ since x is close to $l^{(1)}$
  - $f_2 \approx 0$, $f_3 \approx 0$ since x is far away from $l^{(2)}$ and $l^{(3)}$
  - $\theta^T \cdot f \Rightarrow \theta_0 + \theta_1 = 0.5 \geq 0$ Predict $y = 1$
- Mavi training ex, x baz alınarak:
  - $f_1 \approx 0$, $f_2 \approx 0$, $f_3 \approx 0$
  - $\theta^T \cdot f \Rightarrow \theta_0 = -0.5 < 0$ thus predict $y = 0$

- Bunu devam ettirirsek görürüz ki $l^{(1)}$ ve $l^{(2)}$'ya yakın training ex. için $y = 1$ tahmini yapılırken uzak olanlar için $y=0$ tahmini yapılır.
- Tırnıncı'ya benzer bir non-linear decision boundary oluşmuştur.

- Kernels ile ilgili bazı detaylardan bahsedeceğiz. Pratikte nasıl kullanılacağına değineceğiz. SVM için bias-variance trade off'una değineceğiz.

How to place landmarks?

- For every training example we are going to put a landmark as exactly the same locations as the training examples.



- I'm gonna end up with m landmarks $l^{(1)}, l^{(2)}, \dots l^{(m)}$
- Renkler anlaşılsın diye var aslında önemsiz.

SVM with Kernels

- Given $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$
- choose $l^{(1)} = x^{(1)}, \quad l^{(2)} = x^{(2)}, \dots, l^{(m)} = x^{(m)}$

- Given example x (it can be in training set, cv set or test set):

$$f_1 = \text{similarity}(x, \overbrace{l^{(1)}}^{x^{(1)}})$$
$$f_2 = \text{similarity}(x, l^{(2)}) \quad \Rightarrow \quad f = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_m \end{bmatrix} \text{ where } f_0 = 1$$
$$\dots$$

For training example $(x^{(i)}, y^{(i)})$:

$$x^{(i)} \rightarrow \begin{array}{l} f_1^{(i)} = \text{sim}(x^{(i)}, l^{(1)}) \\ f_2^{(i)} = \text{sim}(x^{(i)}, l^{(2)}) \\ \quad \vdots \\ f_m^{(i)} = \text{sim}(x^{(i)}, l^{(m)}) \end{array}$$

$$f_i^{(i)} = \text{sim}(x^{(i)}, \overbrace{l^{(i)}}^{x^{(i)}}) = 1$$

Çünkü training setten gelen bir $x^{(i)}$ için kesinlikle bir $l^{(i)} = x^{(i)}$ olacak bu noktada da $f_i^{(i)} = 1$ olacaktır.

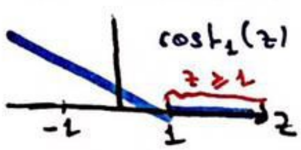
- We can use $f^{(i)} = \begin{bmatrix} f_0^{(i)} \\ f_1^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix}$ instead of $x^{(i)} \in \mathbb{R}^{n+1}$

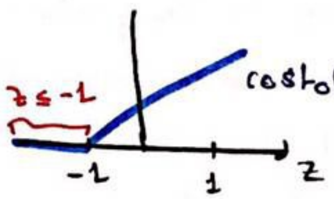
W7 - Large Margin Intuition -

- Sometimes people talk about SVMs as Large Margin Classifiers. Bu derste bundan bahsedeceğiz ve SVM hypothesis nasıl görünür onu konuşacağız

---

- $\min_{\Theta} C \sum_{i=1}^{m} \left[y^{(i)} \cdot cost_1(\Theta^T \cdot x^{(i)}) + (1-y^{(i)}) \cdot cost_0(\Theta^T \cdot x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \Theta_J^2$

- 



If $y=1$, we want $\Theta^T \cdot x \geq 1$ (Not just ≥ 0)
$cost_1(z)$ is 0 only when $z \geq 1$

If $y=0$, we want $\Theta^T \cdot x \leq -1$ (Not just < 0)
$cost_0(z)$ is 0 only when $z \leq 1$

→ $\Theta^T \cdot x \geq 0$ veya < 0 olması yetmiyor. Bunu bir safety factor gibi düşünebiliriz. Bu olay extra safety margin factor build eder

→ Bu durumun sonuçlarına (SVM için) bakalım:

Consider a case that C is very large: Let's say 10 000 (means λ is small. Over-fitting)

- C çok büyükse cost function'ın minimize edilmesi için A termi ≅ 0 olmalı.

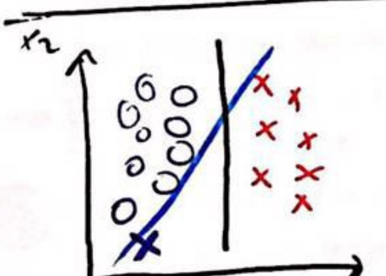
  - whenever $y^{(i)} = 1$: $cost_1(z)$'nin 0 olması için $\Theta^T \cdot x^{(i)} \geq 1$ olmalı!
  - whenever $y^{(i)} = 0$: $cost_0(z)$'nin 0 olması için $\Theta^T \cdot x^{(i)} \leq -1$ olmalı!

Sonuçta eğer Term A 0'a yakın ise bu şu demek her $y^{(i)} = 1$ için $\Theta^T \cdot x^{(i)} \geq 1$ ve her $y^{(i)} = 0$ için $\Theta^T \cdot x^{(i)} \leq -1$ sağlanmış demek. (Term A = 0 ise hepsi sağlanmış demek yakın 0 ise aradan kaçan olabilir) Bu durum ise decision boundary'e şu şekilde yansır:

SVM Decision Boundary: Linearly Seperable Case

- Siyah decision boundary high margin'e sahip. Yani ayırdığı kümelere uzaklığı fazla. İyi ama bu zaten logistic regression için de eğer cost function = 0 ise yaşanmaz mı? Yine large margin olur

---



- C is very large kabul ediyoruz. Yandaki örnekte datalar siyah çizgi ile ayrılabilir.

  - Diyelim ki sol alta ekstra bir positive example eklendi. → Bu aslında overfitting?

- Eğer C çok yüksek ise SVM siyah boundary'den maviye atlar. Ancak bu doğru mu yanlış mı tartışılır.

- C çok yüksek değilse decision boundary yine siyaha benzer olurdu çünkü term A 0'dan biraz büyük olabilir

SVM with Kernels

Hypothesis: Given x, compute features $f \in \mathbb{R}^{m+1}$ ($\Theta \in \mathbb{R}^{m+1}$ now!)

Predict "$y=1$" if $\Theta^T \cdot f \geq 0$ where $\Theta^T \cdot f = \Theta_0 \cdot f_0 + \Theta_1 f_1 + \dots \Theta_m \cdot f_m$

Training the Hypothesis:

$$\min_{\Theta} C \sum_{i=1}^{m} y^{(i)} \cdot cost_1(\Theta^T \cdot f^{(i)}) + (1-y^{(i)}) \cdot cost_0(\Theta^T \cdot f^{(i)}) + \frac{1}{2} \sum_{J=1}^{m} \Theta_J^2$$

→ n=m gibi düşün.

- Yukarıdaki ifade genelde intuition'ı verir ama pratikte regularization term farklıdır:
  - $\sum_{J=1}^{m} \Theta_J^2 = \Theta^T \cdot \Theta$ (ignoring Θ_0) $\Rightarrow \Theta = \begin{bmatrix} \Theta_1 \\ \vdots \\ \Theta_m \end{bmatrix}$
  - $\sum_{J=1}^{m} \Theta_J^2$ yerine $\Theta^T . M . \Theta$ kullanılır yukarıdakine yakın with this modification. It allows to scale to much bigger training sets. Bu sadece matematiksel bir detay.

Can we apply Kernels idea to other algorithms? Like log. reg.

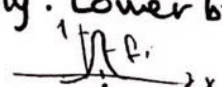
- Uygulanabilir. But computational tricks that apply for SVMs don't generalize well to other algorithms like logistic regression. So using kernels with logistic regression will be very slow.
- SVM and Kernels tend to go particularly well together.

Yukarıdaki cost function'ı minimize etmek için off the shelf algorithms var. Bunları kullanmak daha mantıklı diyor.

SVM Parameters: Bias vs Variance

- C played a role similar to $1/\lambda$ where λ is reg. par. for log. reg.
- Large C: Tend to have a hypothesis with lower bias and higher variance. → more prone to overfitting
- Small C: ↳ "more prone" to underfitting " higher bias and lower variance

- Large σ^2: Features f_i vary more smoothly. Higher bias, lower variance.
- Small σ^2: Features f_i vary less smoothly. Lower bias, higher variance.



- W7 - Using a SVM -

- What we actually need to do in order to run an SVM.

- It is not recommended to write a software to solve for the parameters θ. Yani $J(\theta)$'yı minimize edecek algoritmayı bizim yazmamız şart değil. Instead use SVM software package (e.g. liblinear, libsvm, ...) to solve for parameters θ.

- We still need to specify: → Choice of parameter C → Choice of kernel (similarity function):

  - E.g. No kernel ("linear kernel"): Predict "$y=1$" if $\theta^T.x \geq 0$
    - when n is large, m is small: Maybe its better to fit a linear decision boundary and not try to fit a very complicated non-linear function because you might not have enough data, there might be overfitting.
  - Another choice is Gaussian Kernel: $f_i = \exp\left(-\frac{\|x-l^{(i)}\|^2}{2\sigma^2}\right)$, where $l^{(i)} = x^{(i)}$.
    - Need to choose σ^2.
    - when n is small, m is large: We have enough training set to tran a complex nonlinear hypothesis and avoid overfitting. Mesela 2 dimensional training set (x_1, x_2) ve binsenu training data ran complex bir hipotezi overfit etmeden oturtabiliriz.

If we decide to use Gaussian kernel here is what we should do:

- function f = kernel (x_1, x_2) $\quad x^{(i)}, \; l^{(j)} = x^{(j)}$

  $$f = \exp\left(-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right)$$

  return

Do perform feature scaling before using the Gaussian kernel. Aslında feature'lar farklı ise her zaman uygula!

size 1000's   bedrooms 1-5

$$\|x - l^{2}\| = (x_1 - l_1)^2 + (x_2 - l_2)^2 + \dots + (x_n - l_n)^2$$

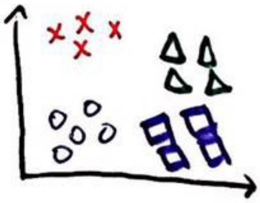
- Feature scaling lazım çünkü size hep çok büyük olacak bu yüzden sonucu domine eder.

Other choices of kernel: Not all similarity (x, l) functions make valid kernels. (Need to satisfy technical condition called "Mercer's Theorem" to make sure SVM packages' optimizations run correctly, and do not diverge).

- Many off-the-shelf kernels available:
  - Polynomial kernel: $k(x, l) = (x^T.l + \text{constant})^{\text{degree}}$
  - More esoteric: String kernel, chi-square kernel, histogram intersection kernel, ...

- W7 - Using a SVM II -

Multi-class Classification



$y \in \{1, 2, 3, \ldots k\}$

- Many SVM packages already have built-in multi-class classification functionality.
- Otherwise, use one vs all method. (Train K SVMs, each distinguish different class
(Exactly same as Logistic Regression case)

Logistic Regression vs. SVMs

- n = # of features ($x \in \mathbb{R}^{n+1}$) • m = # of training examples.

- If n is large (relative to m): (E.g. n=10000, m=10-1000)
  - Use logistic regression, or SVM without a kernel ("linear kernel")
  - Because if we have so many features with smaller training sets a linear function will probably do fine. We don't have enough data to fit a complex non-linear function

- If n is small, m is intermediate (n=1-1000, m=10-10000)
  - Often an SVM with Gaussian kernel will work fine.

- If n is small, m is large (n=1-1000, m=50000+)
  - In this case SVM with Gaussian kernel would be somehow slow to run. Yani m çok büyükse SVM with Gaussian Kernel zorlanır.
  - Manually Create/Add more features, then use logistic regression or SVM without kernel

---

- SVM without kernel ile logistic regression çok benzer. Sadece SVM'in olayı kernel ile efficient çalışabilmesi

---

- For all of these regimes a Neural Network is likely to work well. But maybe slower to train.

- Optimization problem that the SVM has is a convex optimization problem. So a good algorithm will always find the global minimum.

In practice local optimas aren't a huge problem for neural networks neither.