Let n be the number of steps between evaluations.
Let p be the "patience," the number of times to observe worsening validation set error before giving up.
Let θ_o be the initial parameters.

$\theta \leftarrow \theta_o$
$i \leftarrow 0$
$j \leftarrow 0$
$v \leftarrow \infty$
$\theta^* \leftarrow \theta$
$i^* \leftarrow i$
while $j < p$ **do**
  Update θ by running the training algorithm for n steps.
  $i \leftarrow i + n$
  $v' \leftarrow \text{ValidationSetError}(\theta)$
  if $v' < v$ **then**
    $j \leftarrow 0$
    $\theta^* \leftarrow \theta$
    $i^* \leftarrow i$
    $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are θ^*, best number of training steps is i^*

Algorithm determines the best amount of time to train. The meta algorithm is a general strategy that works well with a variety of training algorithms and ways of quantifying error on the validation set.

7

20

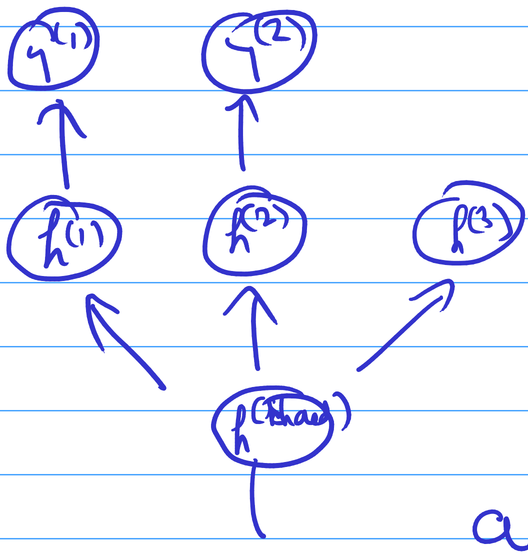
θ_0

T iterations

θ_0 ~~4,000~~

$\boxed{20{,}000}$

Note: SGD does (provides) implicit regularization



no. of steps for early stopping is a hyperparameter

↳ Find this by training a small subset, then use the number to train whole model

(Shut up & TRAINNN)

Flavor #2

Let $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ be the training set.
Split $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ into $(\boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{X}^{(\text{valid})})$ and $(\boldsymbol{y}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{valid})})$ respectively.
Run early stopping (algorithm 7.1) starting from random $\boldsymbol{\theta}$ using $\boldsymbol{X}^{(\text{subtrain})}$ and $\boldsymbol{y}^{(\text{subtrain})}$ for training data and $\boldsymbol{X}^{(\text{valid})}$ and $\boldsymbol{y}^{(\text{valid})}$ for validation data. This updates $\boldsymbol{\theta}$.
$\epsilon \leftarrow J(\boldsymbol{\theta}, \boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{subtrain})})$
while $J(\boldsymbol{\theta}, \boldsymbol{X}^{(\text{valid})}, \boldsymbol{y}^{(\text{valid})}) > \epsilon$ do
  Train on $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ for n steps.
end while

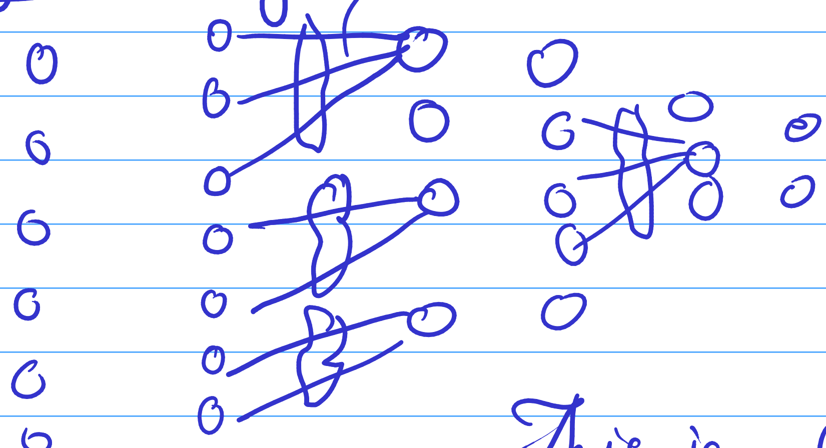
θ^* for this

70%

don't throw θ^* away

Weight sharing:

same



This is also a regularizer
, overfitting doesn't happen

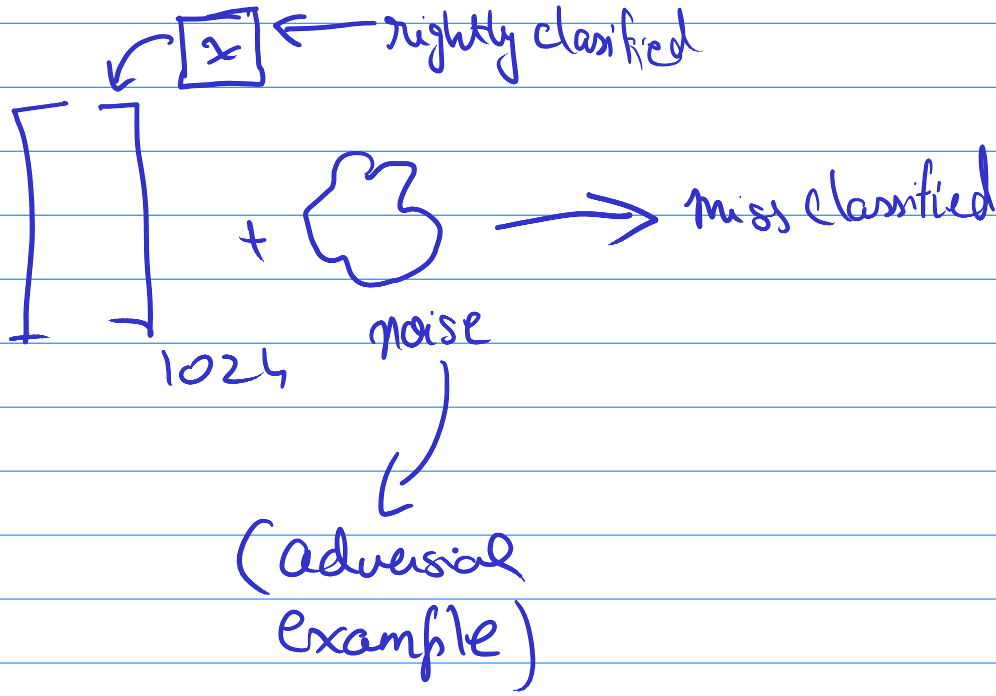
(adversarial)

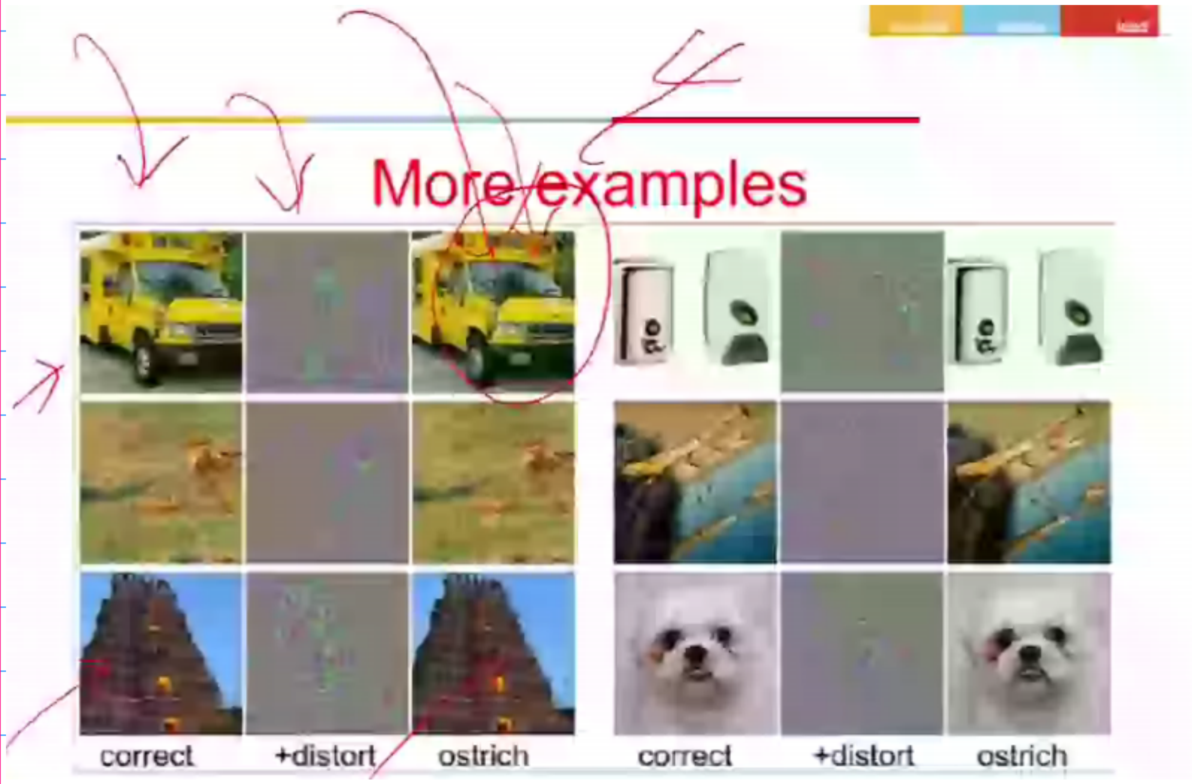
Adversian Training

Szegedy

2014

$\boxed{x}$ ← rightly classified





goodfellow 2014, explaining adversial -- a composition of functions
But relu type functions composed can be incredibly linear most of the time
a heavily linear function will look like a lin. comb. of weights

a close neibourhood can be W.(x + epsilon) (x,w and e are vectors)

$f(x+\varepsilon) = \left(\frac{1}{x+\varepsilon}\right)$

$\frac{1}{x+\varepsilon} \simeq \frac{1}{x}$

if f is quadratic, ε will become ε^2 etc. and vanish

Every layer

$$\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{30000} \end{bmatrix} \left(\begin{bmatrix} x_1 \\ \vdots \\ x_{30000} \end{bmatrix} + \underbrace{\begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_{30000} \end{bmatrix}}_{\varepsilon \text{ is non linear}} \right)$$

$\Rightarrow$ x becomes high non linear

$$f\left(\begin{bmatrix} x_{11} \\ x_{12} \\ \vdots \\ x_{1100} \end{bmatrix} + \begin{bmatrix} \epsilon \\ \epsilon \\ \epsilon \\ \vdots \\ \epsilon \end{bmatrix} \right)$$ changes quite alot

$$f(x) - f(x+\varepsilon)$$

apply 'f_1' on this

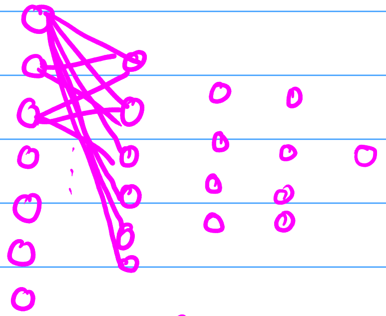
$\vdots$

$$g = f_0 \circ f_1 \circ f_2 \cdots f_n(x)$$

$g(y) - g(y+\varepsilon)$ will be way more than

$f(x) - f(x+\varepsilon)$

$$f = \omega_1 x_1 + \omega_2 x_2 + \cdots + \omega_{100} x_{100}$$



$$f \simeq \omega_1 x_1 + \omega_2 x_2 + \cdots + \omega_{100} x_{100}$$

Adding noise

Whitebox attack

$$\omega^T x + \omega^T \delta$$

$$\omega^T x + \omega^T \underbrace{\begin{bmatrix} \delta_1 \\ \delta_2 \\ \vdots \\ \delta_{100} \end{bmatrix}}_{\text{the quantity}}$$

Let δ_1 be $+\epsilon$ if $\omega \geq 0$

δ_1 be $-\epsilon$ if $\omega < 0$

$x = x + \delta$

$\delta = \epsilon\,[\text{sign}(\text{grad})]$

hyperparam

$$\omega^T x + \epsilon(\|\omega\|_{L1}) = \omega^T x + \epsilon(|\omega_1| + |\omega_2| + \cdots |\omega_{100}|)$$

$x \to (x + \delta)$

$f(x) \to f(x + \delta)$

Since f is linear, even if x is close to x'

$f(x')$ can be completely different

BUT WHY?? (This seems contradictory)

You can't ask this lol

Basically, this is an explanation, not a proof for adversarial examples

It works for not all examples, f is normally piecewise linear

We are also NOT comparing with polynomial & exponential; we are just saying linearity itself makes values grow! Compared to sigmoid etc.

SVMs are also linear but have large decision regions, so they don't face this problems.

⇒ Training on adversarial examples makes model more robust