

Deep Learning

Sem 02 - Aula 02

Renato Assunção - DCC - UFMG



Funções de ativação precisam ser não-lineares

- Em redes neurais, é essencial ter uma função de ativação não-linear
- Função linear de entradas x_1, x_2, x_3, x_4 :
 - Qualquer função da forma $f(x_1, x_2, x_3, x_4) = w_0 + w_1 * x_1 + w_2 * x_2 + w_3 * x_3 + w_4 * x_4$
 - ou da forma $f(x) = Wx + w_0$ onde w_0 e W são matrizes de constantes e x é vetor-coluna
- O que acontece se as funções de ativação numa rede neural são lineares?

E se forem lineares?

- Considere uma rede neural com L camadas sucessivas
- Se as funções de ativação em cada camada forem todas lineares:
 - as L camadas podem ser reduzidas a uma única camada
 - a relação entre os inputs e a saída será uma função linear
 - a rede vai representar bem apenas funções aproximadamente lineares

$$z^{[2]} = W^{[2]}a^{[1]} \quad (2.8)$$

$$= W^{[2]}g(z^{[1]}) \quad \text{by definition} \quad (2.9)$$

$$= W^{[2]}z^{[1]} \quad \text{since } g(z) = z \quad (2.10)$$

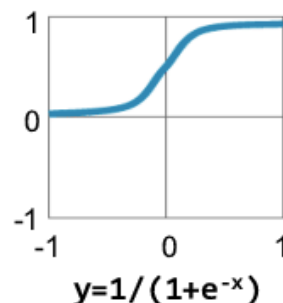
$$= W^{[2]}W^{[1]}x \quad \text{from Equation (2.4)} \quad (2.11)$$

$$= \tilde{W}x \quad \text{where } \tilde{W} = W^{[2]}W^{[1]} \quad (2.12)$$

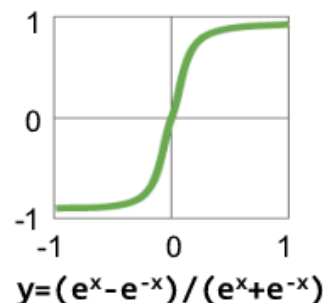
Funções de ativação

Traditional Non-Linear Activation Functions

Sigmoid

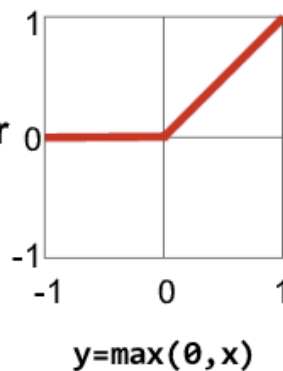


Hyperbolic Tangent

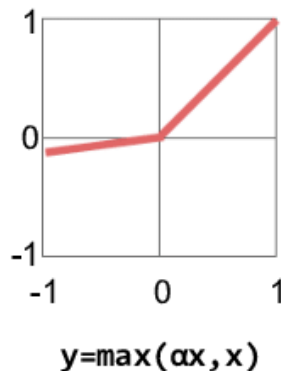


Modern Non-Linear Activation Functions

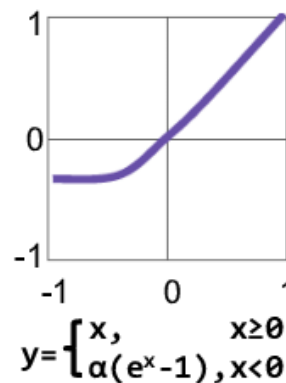
**Rectified Linear Unit
(ReLU)**



Leaky ReLU



Exponential LU



$\alpha = \text{small const. (e.g. 0.1)}$

- Uma descoberta importante recente:
 - sigmóides e tangente hiperbólica não são boas funções de ativação quando temos muitas camadas na rede.
 - Em geral, usamos sigmóide ou tangente hiperbólica na última camada (na saída) quando o problema é de classificação
 - Em outras camadas mais internas costumamos usar outras funções de ativação como a ReLU ou outras mais especializadas para imagens (mais tarde, em redes convolucionais)
- Qual o problema com sigmóides e tangentes hiperbólicas?

Gradientes em funções compostas

- Rede neural é uma grande sequência de composição de funções.
- Alternamos combinações lineares das saídas do passo anterior com uma função de ativação.
- Fazemos isto diversas vezes.
- Na regra da cadeia vai aparecer a DERIVADA da função de ativação várias vezes, como fatores multiplicativos.
- Suponha que todas as funções de ativação sejam sigmóides.
- No backpropagation, teremos derivadas de funções lineares vezes a derivada da logística VÁRIAS VEZES (= número de camadas)

Composição e o produto de derivadas da função logística

- Função de custo ou de perda é uma composição de camadas (funções):

$$\begin{aligned}\mathcal{L}(w_1, w_2, w_3, w_4, \dots) &= \sum_{i=1}^m \mathcal{L}^{(i)} \\ &= \sum_{i=1}^m (\sigma_1 \circ T_1 \circ \sigma_2 \circ T_2 \circ \sigma_3 \circ T_3)(x^{(i)}, w_1, w_2, w_3, w_4, \dots)\end{aligned}$$

- Derivada parcial com respeito a w_2 :

$$\begin{aligned}\frac{\partial}{\partial w_2} \mathcal{L}(w_1, w_2, w_3, w_4, \dots) &= \sum_{i=1}^m \frac{\partial}{\partial w_2} \mathcal{L}^{(i)} \\ &= \sum_{i=1}^m \frac{\partial}{\partial w_2} (\sigma_1 \circ T_1 \circ \sigma_2 \circ T_2 \circ \sigma_3 \circ T_3)(x^{(i)}, w_1, w_2, w_3, w_4, \dots) \\ &= \sum_{i=1}^m \sigma'_1 \times \partial T_1 \times \sigma'_2 \times \partial T_2 \times \sigma'_3 \times \partial T_3\end{aligned}$$

Composição e o produto de derivadas da função logística

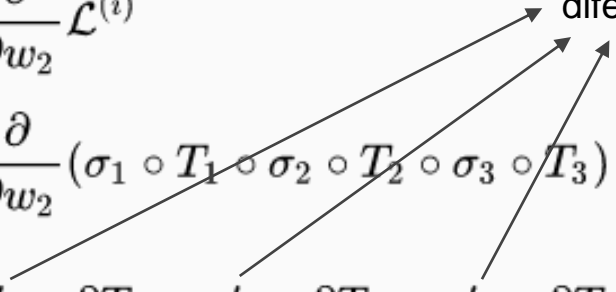
- Função de custo ou de perda é uma composição de camadas (funções):

$$\begin{aligned}\mathcal{L}(w_1, w_2, w_3, w_4, \dots) &= \sum_{i=1}^m \mathcal{L}^{(i)} \\ &= \sum_{i=1}^m (\sigma_1 \circ T_1 \circ \sigma_2 \circ T_2 \circ \sigma_3 \circ T_3)(x^{(i)}, w_1, w_2, w_3, w_4, \dots)\end{aligned}$$

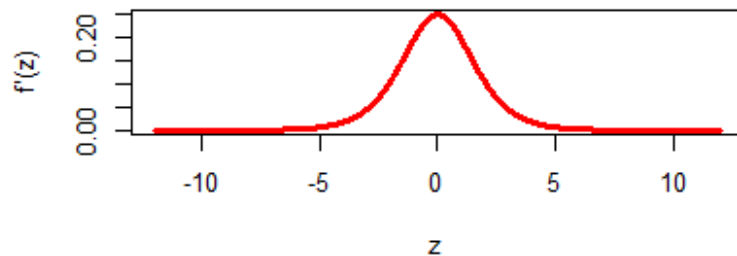
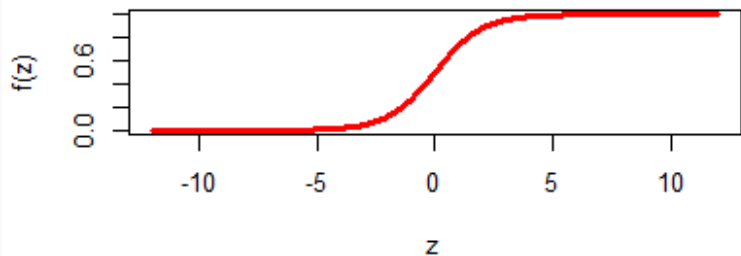
- Derivada parcial com respeito a w_2 :

$$\begin{aligned}\frac{\partial}{\partial w_2} \mathcal{L}(w_1, w_2, w_3, w_4, \dots) &= \sum_{i=1}^m \frac{\partial}{\partial w_2} \mathcal{L}^{(i)} \\ &= \sum_{i=1}^m \frac{\partial}{\partial w_2} (\sigma_1 \circ T_1 \circ \sigma_2 \circ T_2 \circ \sigma_3 \circ T_3)(x^{(i)}, w_1, w_2, w_3, w_4, \dots) \\ &= \sum_{i=1}^m \sigma'_1 \times \partial T_1 \times \sigma'_2 \times \partial T_2 \times \sigma'_3 \times \partial T_3\end{aligned}$$

produto de derivadas da sigmóide avaliadas em diferentes pontos

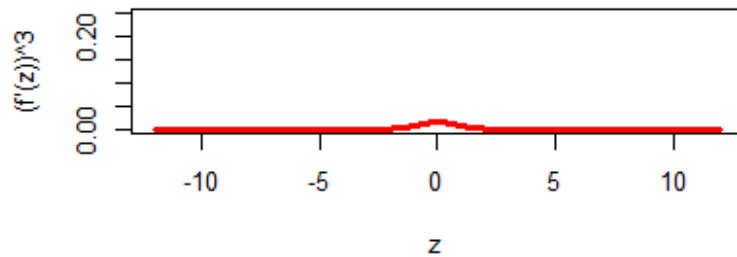
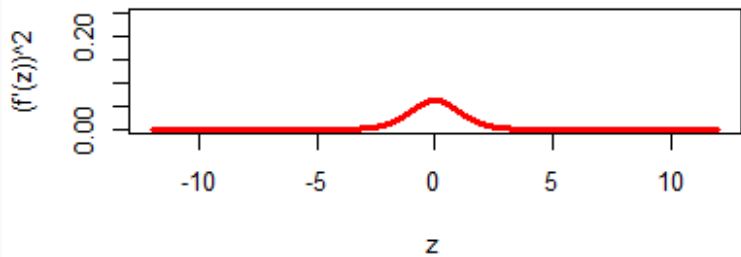


Sigmóide: vanishing gradients



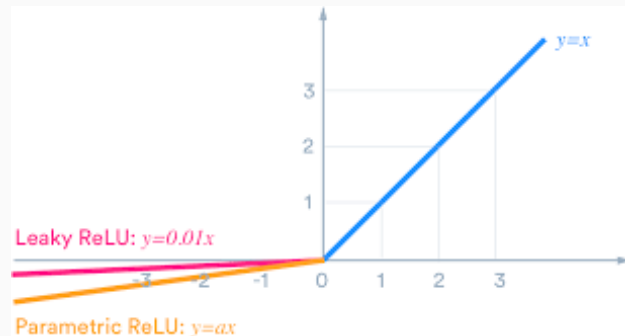
Produto das derivadas da sigmóide no mesmo ponto

Melhor caso:
ponto 0, o
máximo:
dissolve
rapidamente
(vanishes
quickly)



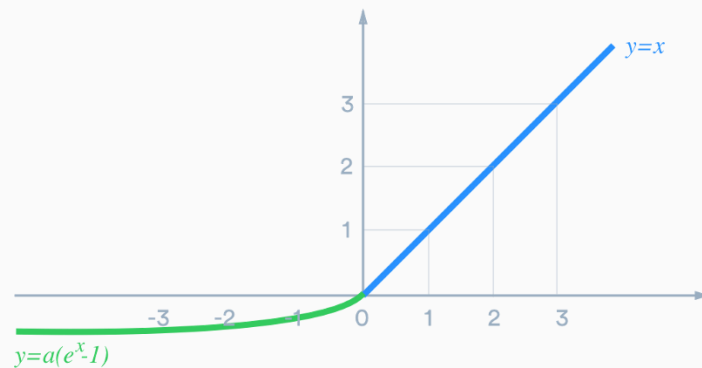
E a ReLU?

- Função $f(x) = \max\{0, x\}$
- Derivada $f'(x) = 0$ ou 1 dependendo de $x < 0$ ou $x > 0$
- Derivadas no backprop:
 - além dos termos de derivadas de funções lineares, teremos produtos de derivadas de ReLU.
 - Elas podem zerar a derivada parcial se uma delas for zero
 - Mas o produto das derivadas não necessariamente $\rightarrow 0$, como nas sigmóides
- Leaky ReLU: criada para evitar a derivada ZERO da ReLU (“dying ReLU”)
- Parametrized ReLU: PReLU
 - introduz um parâmetro a

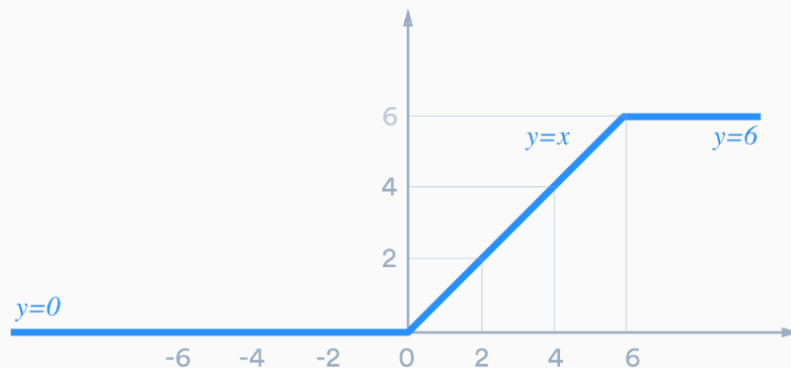


Alternativas a ReLU

- Exponential ReLU: ELU



- ReLU-6

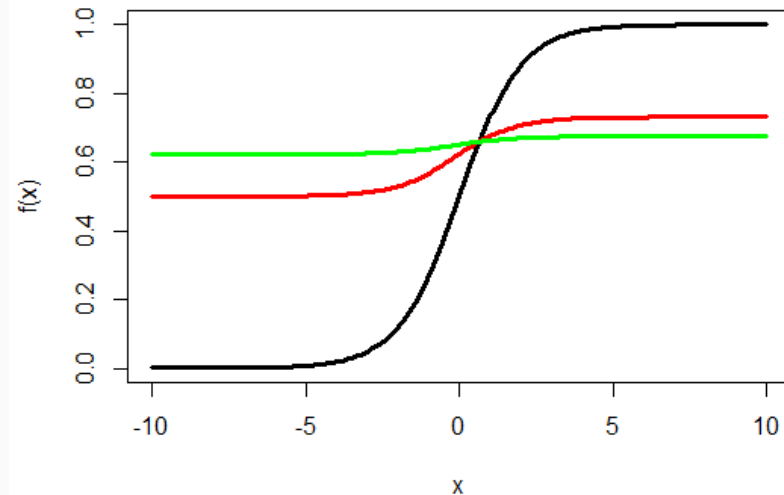


ReLU é idempotente

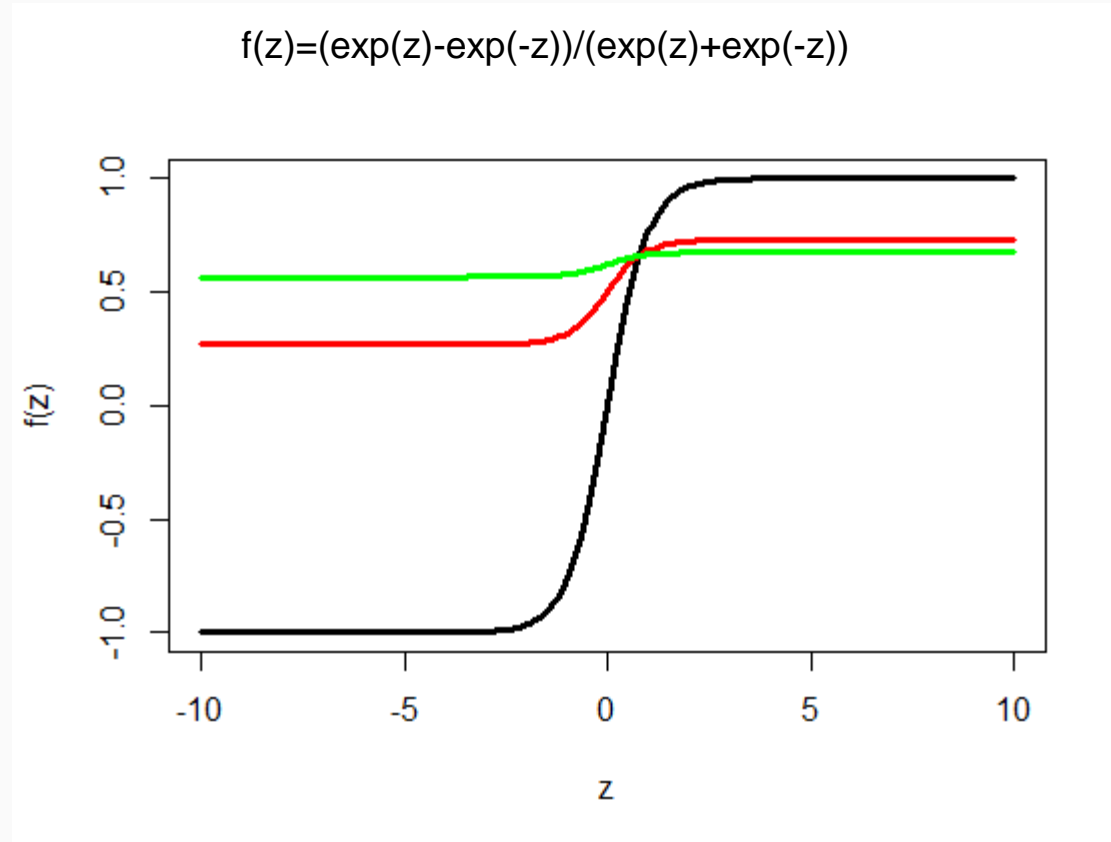
- Uma coisa importante a salientar é que ReLU é uma função idempotente.
- Def: Uma função $f(x)$ é chamada de idempotente se $f(f(x)) = f(x)$
- Exemplo:
 - $f(x) = x$
 - $f(x) = |x|$
 - $f(x) = \max(0, x) = \text{ReLU}$
- Veja que $f(f(x)) = f^2(x) = (f \circ f)(x)$
- Funções que não são idempotentes:
 - $f(x) = x^2$ ou $f(x) = 1/(1+e^{-x}) = \text{logística}$

Outra maneira de ver o vanishing gradient

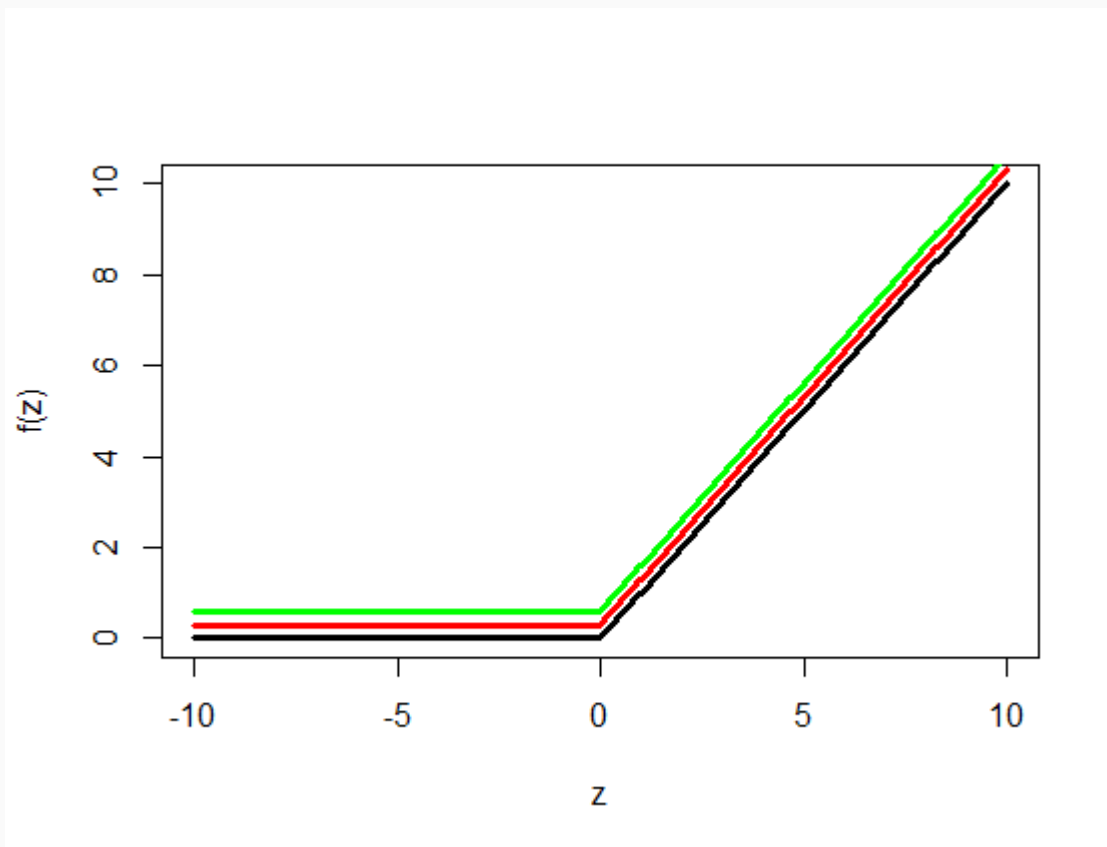
- Se f é idempotente então $f \circ f \circ f \circ \dots \circ f = f$
- Essa propriedade é importante para redes neurais profundas, porque cada camada na rede aplica uma não-linearidade.
- Agora, vamos aplicar duas funções da família sigmóide à mesma entrada repetidamente 1, 2, 3 vezes
- A função sigmóide composta "esmaga" sua entrada, resultando no problema do vanishing gradient: as derivadas vão para zero com no. de composições



Tangente hiperbólica composta 3 vezes



ReLU composta 3 vezes (jittered para visualizar)



ReLU e variantes estão nos principais frameworks

```
import tensorflow as tf

conv_layer = tf.layers.conv2d(
    inputs=input_layer,
    filters=32,
    kernel_size=[5, 5],
    padding='same',
    activation=tf.nn.relu,
)
```

```
from keras.layers import Activation, Dense

model.add(Dense(64, activation='relu'))
```

```
from torch.nn import RNN

model = nn.Sequential(
    nn.Conv2d(1, 20, 5),
    nn.ReLU(),
    nn.Conv2d(20, 64, 5),
    nn.ReLU()
)
```



Entendendo a necessidade de regularização

- Aprendemos os parâmetros W 's e b 's minimizando a função de perda
- Como a função de perda = $-\log$ -verossimilhança, estamos maximizando \log -verossimilhança
- Fisher mostrou que o MLE é o melhor estimador possível de um parâmetro.
- Então... nada mais a fazer, certo?
- George Box: todos os modelos são falsos; alguns são úteis



- O modelo é verdadeiro: dados observados foram, de fato, gerados de acordo com o modelo probabilístico sob análise.
- Existe um verdadeiro valor do parâmetro θ que gera os dados observados
- Quando a amostra não é pequena, o MLE é:
 - converge para θ quando a amostra cresce
 - aprox não-viciado para estimar θ
 - um estimador não viciado possui variância $\geq \text{Informação de Fisher}^{-1}$
 - o erro de estimação do MLE tem variância \rightarrow inverso da informação de Fisher
 - MLE aprox gaussiano

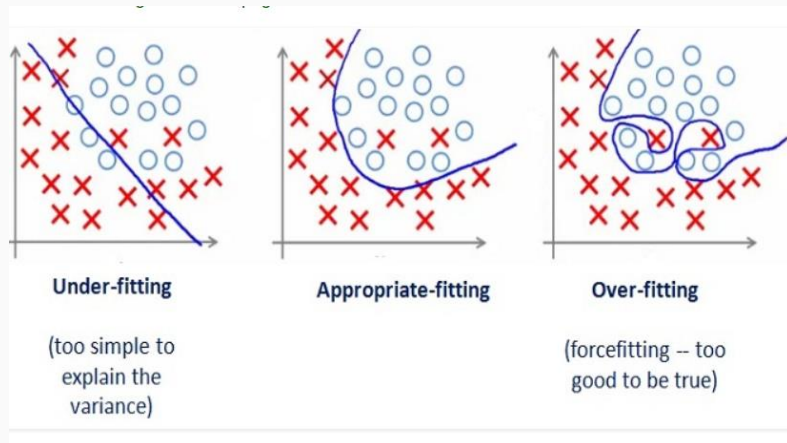
Exemplo muito simples

- Caso gaussiano i.i.d. $Y_1, Y_2, \dots, Y_n \sim N(\mu, \sigma^2)$
- Como estimar μ ? Média aritmética? Mediana? Média dos extremos?
Combinação desses estimadores?
- Suponha que $\hat{\mu}$ seja qualquer estimador de μ
- Um estimador é não-viciado se ele não subestima ou superestima sistematicamente: $\mathbb{E}(\hat{\mu}) = \mu$
- MSE = Erro de estimação ao quadrado $MSE(\hat{\mu}) = \mathbb{E}[(\hat{\mu} - \mu)^2]$
- Para estimadores aprox não-viciados, $MSE(\hat{\mu}) \geq \sigma^2/n$
- O MLE tem seu MSE igual a σ^2/n , menor valor possível

inverso da informação de Fisher
- Nada pode ser melhor

- Em princípio, podemos ter um estimador viesado para μ com variância menor que σ^2/n
- E quando o modelo verdadeiro não for normal?
- Usamos o MLE no modelo falso.
 - O MLE converge? Para onde?
 - Tem variância mínima?
 - O MLE converge para o modelo “falso” mais próximo (sentido de distância de Kullback-Leibler) do modelo verdadeiro desconhecido
 - White (1982). Econometrica,

OK, e daí? Especificação incorreta em redes neurais

- Sem esperanças de especificar um “modelo verdadeiro” em redes neurais
- Em que direção erramos na especificação?
- Costumamos colocar muita redundância de features ou parâmetros.
- Em DL, #parâmetros cresce com #dados
- Risco de overfitting na fase de treinamento (controlado pelo erro no conjunto de teste).

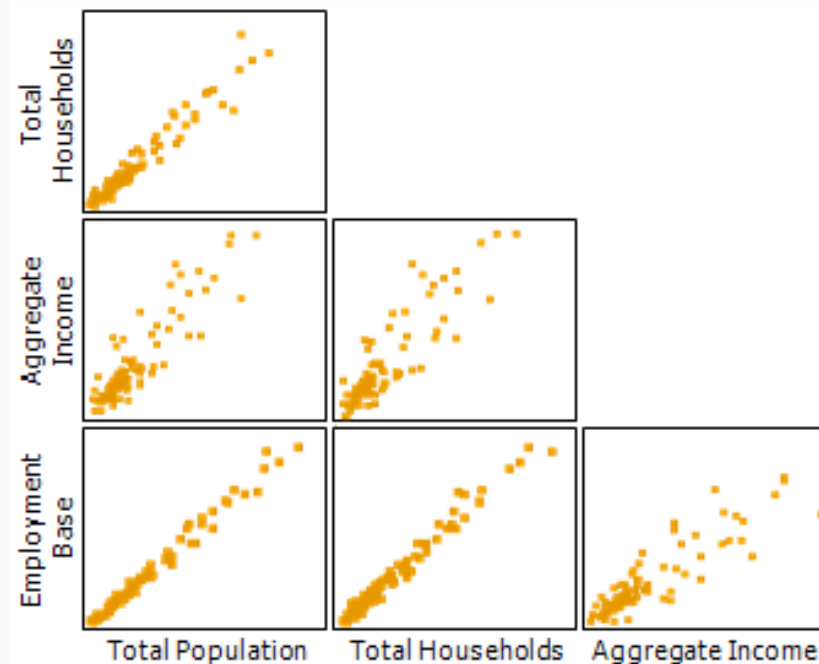


de aulas de Andrew Ng, coursera

- Regressão linear simples → para entender bem o núcleo do problema
- vetor Y com as respostas (tamanho = 1500 exemplos)
- Imagine predizendo resposta Y com base em $n+1$ features
- Preditor é uma combinação linear dos n vetores $1, x_1, x_2, \dots, x_n$
- $n + 1$ colunas de tamanho 1500 cada uma delas.
- Preditor é uma combinação linear desses vetores.
- Imagine que a predição é muito boa, pouco erro de predição.

Muitas features → redundância

- Colocamos muitas features no modelo
- Várias medem coisas similares
- → altamente correlacionadas entre si
- Veja exemplo ao lado
- Dados de setores censitários



Outro exemplo simples

All Greens Franchise

The data (X1, X2, X3, X4, X5, X6) are for each franchise store.

X1 = annual net sales/\$1000

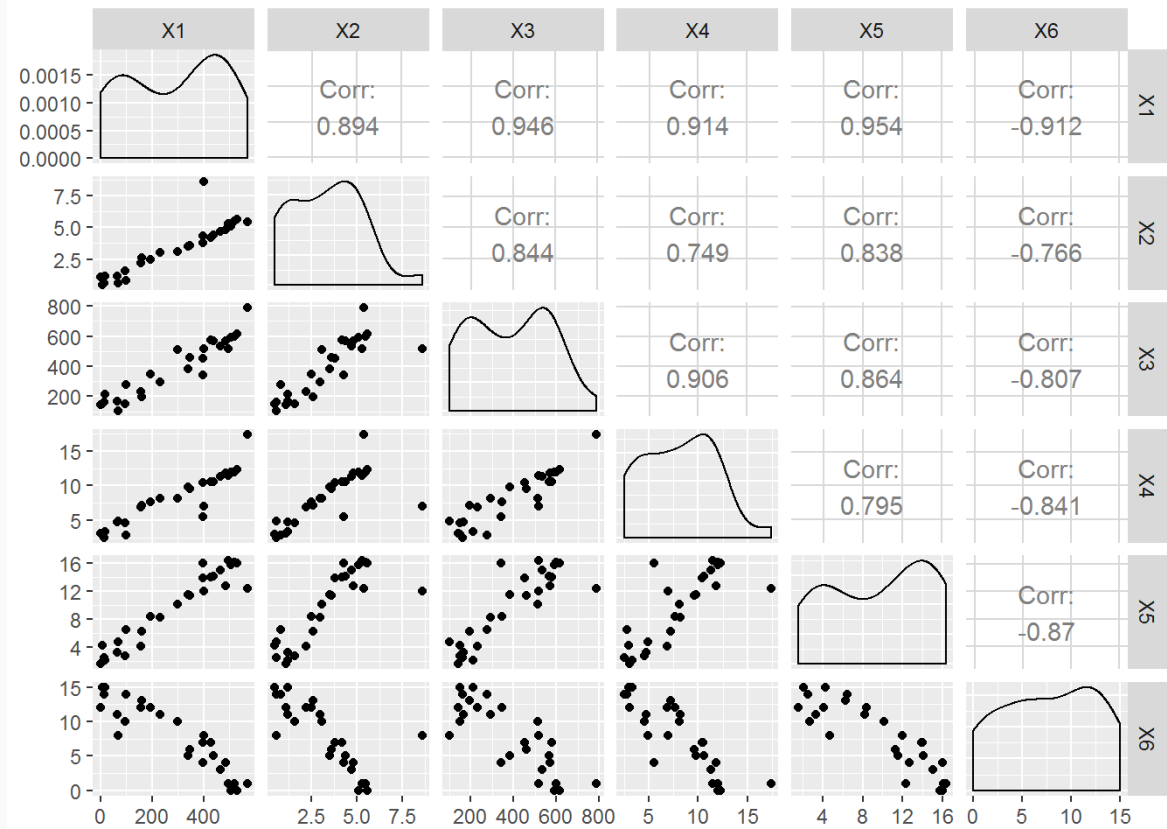
X2 = number sq. ft./1000

X3 = inventory/\$1000

X4 = amount spent on advertising/\$1000

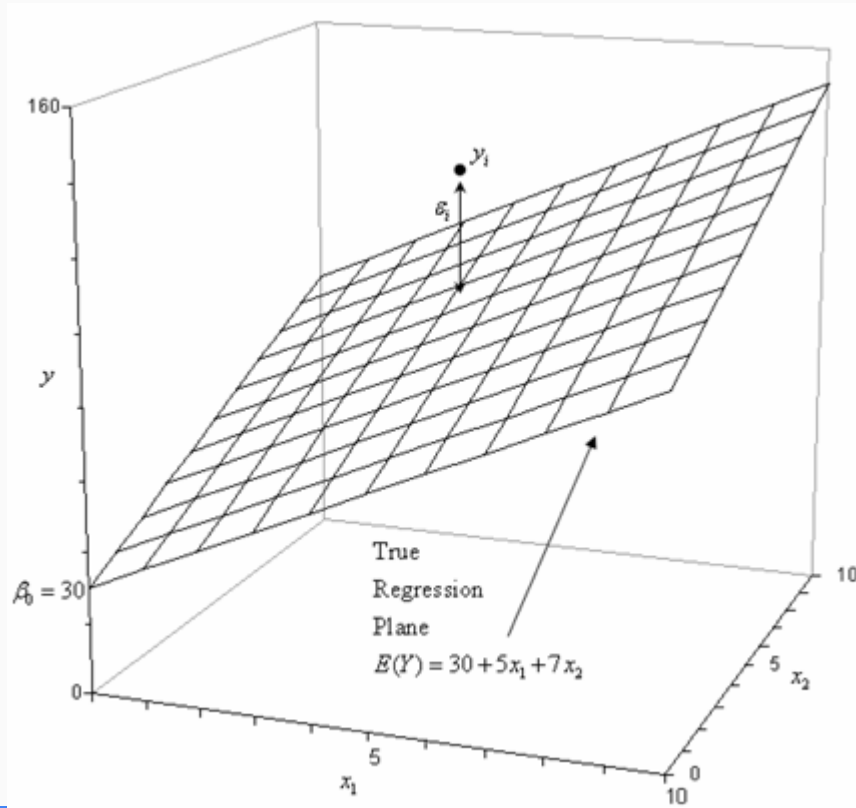
X5 = size of sales district/1000 families

X6 = number of competing stores in district

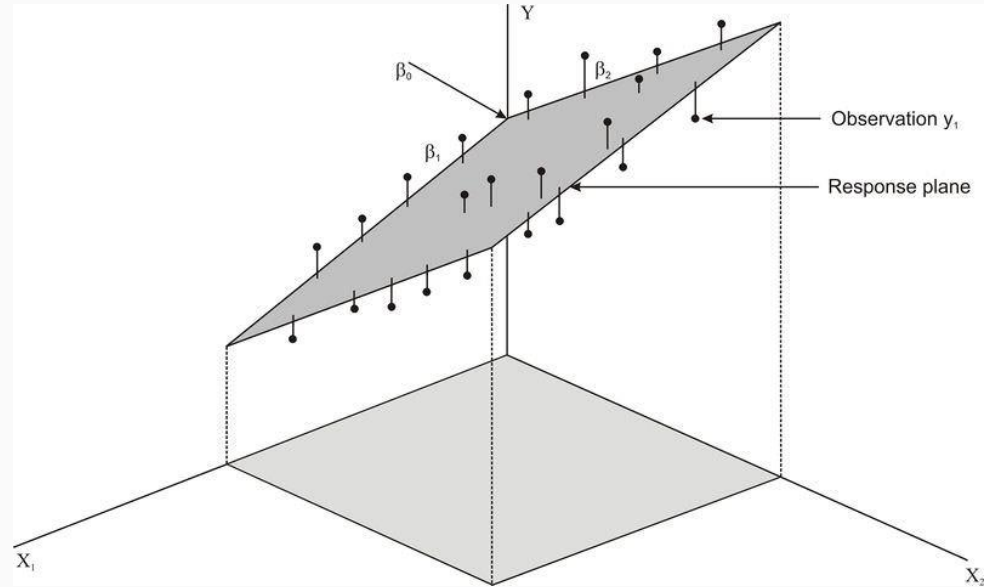


A regressão múltipla mais simples: duas features

O modelo gerador dos dados

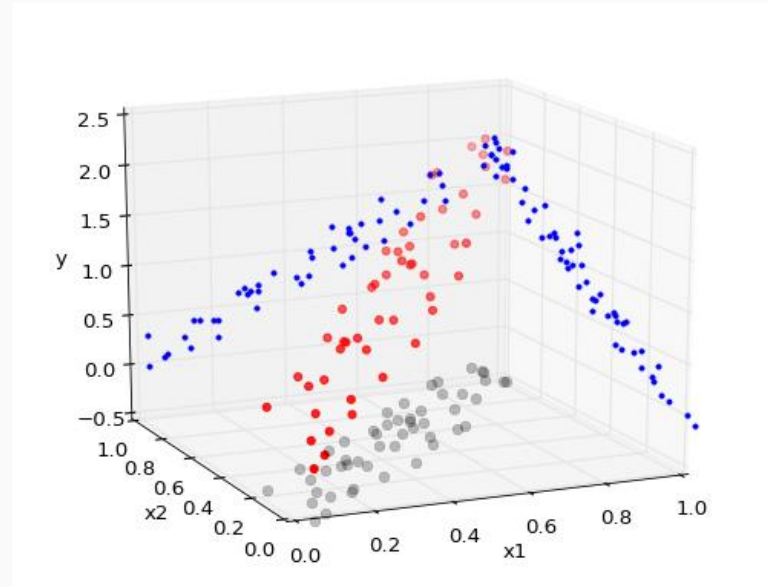
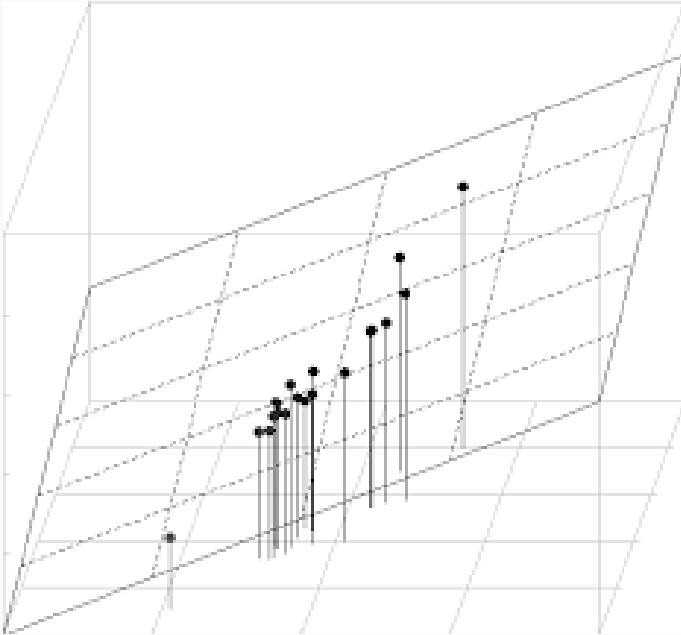


O modelo e os dados gerados



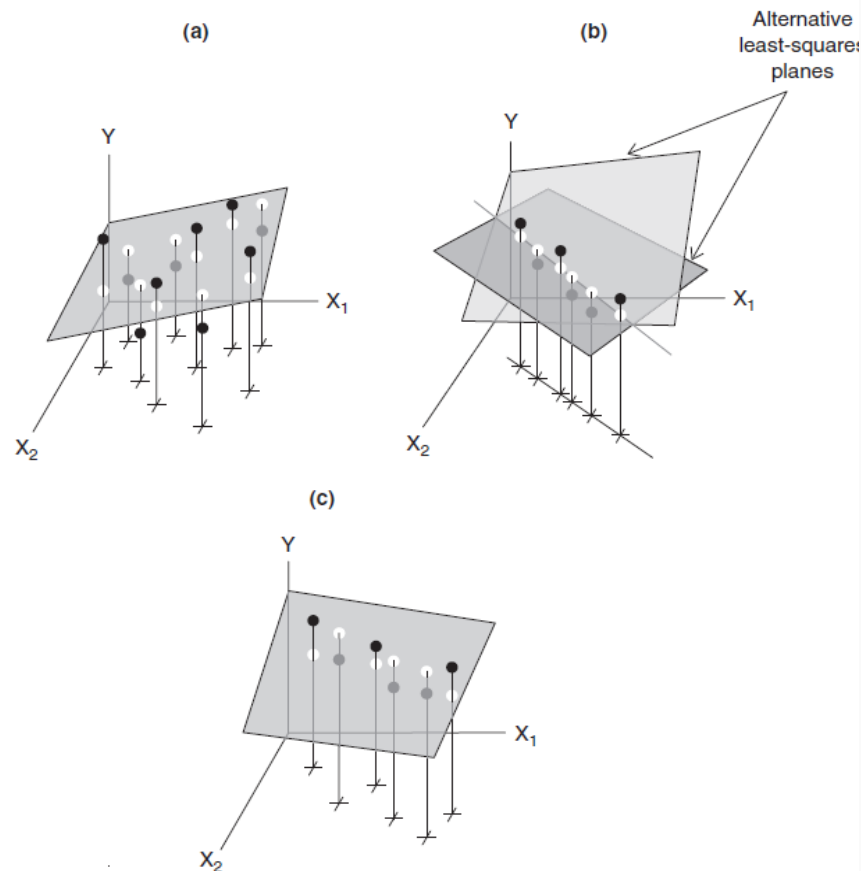
Duas features quase colineares

- <https://www.youtube.com/watch?v=gAlVp3mt8Bw>



Instabilidade dos coeficientes

- Predição: OK, quase não muda
- Coeficientes da regressão:
 - altamente instáveis
 - Quer dizer: alta variância
 - Pequena mudança nos dados, e teremos coeficientes completamente diferentes (mas quase as mesmas predições)
- Modelo pode omitir certas features sem prejuízo da predição

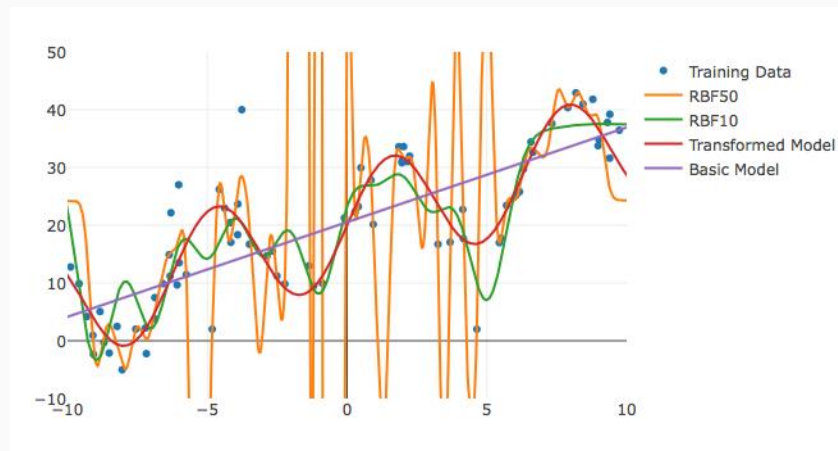
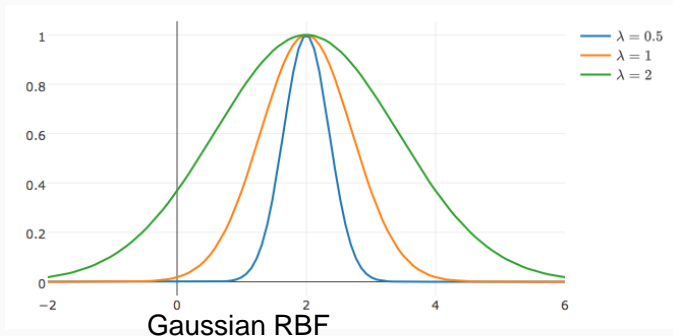


Outra maneira (mais usual) de ver a necessidade de regularização

- **Generic Features:** *increase model expressivity*

- **Gaussian Radial Basis Functions:**

$$\phi_{\lambda_i, \mu_i}(x) = \exp\left(-\frac{\|x - \mu_i\|_2^2}{\lambda_i}\right)$$

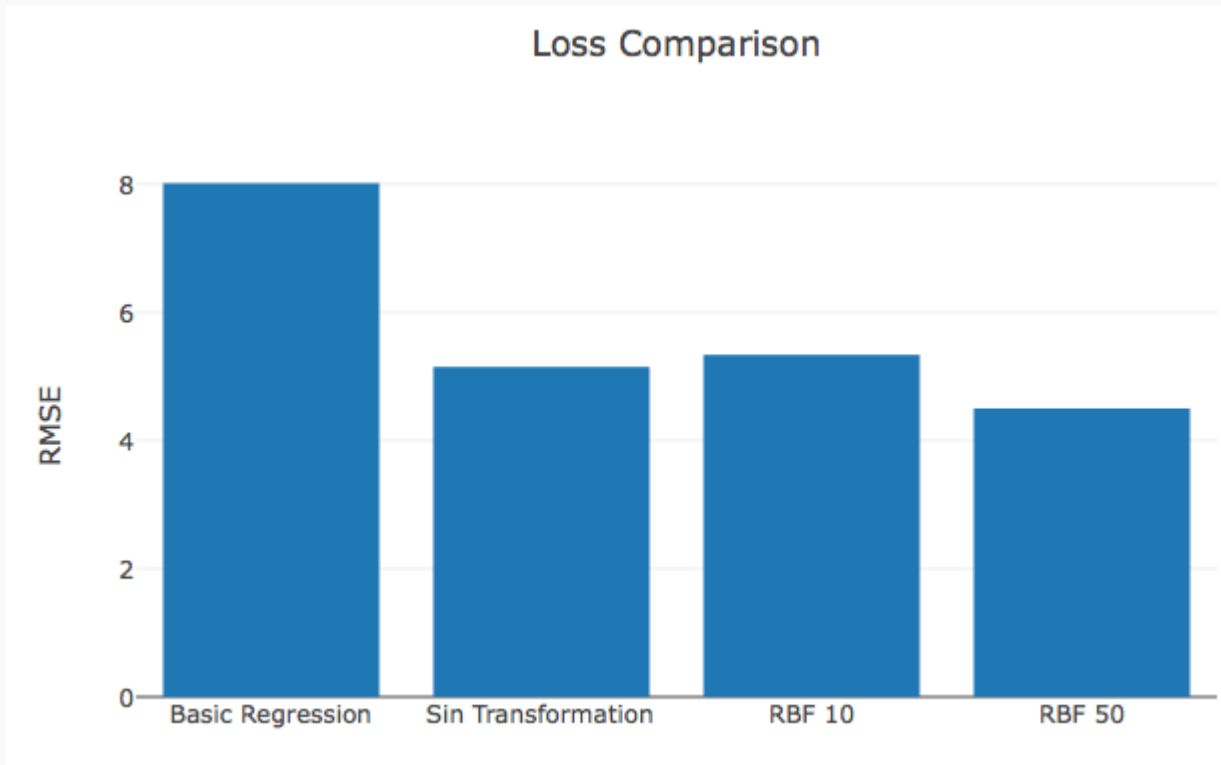


Este e os próximos 7 slides vieram de aula
preparada por

Joseph E. Gonzalez jegonzal@cs.berkeley.edu

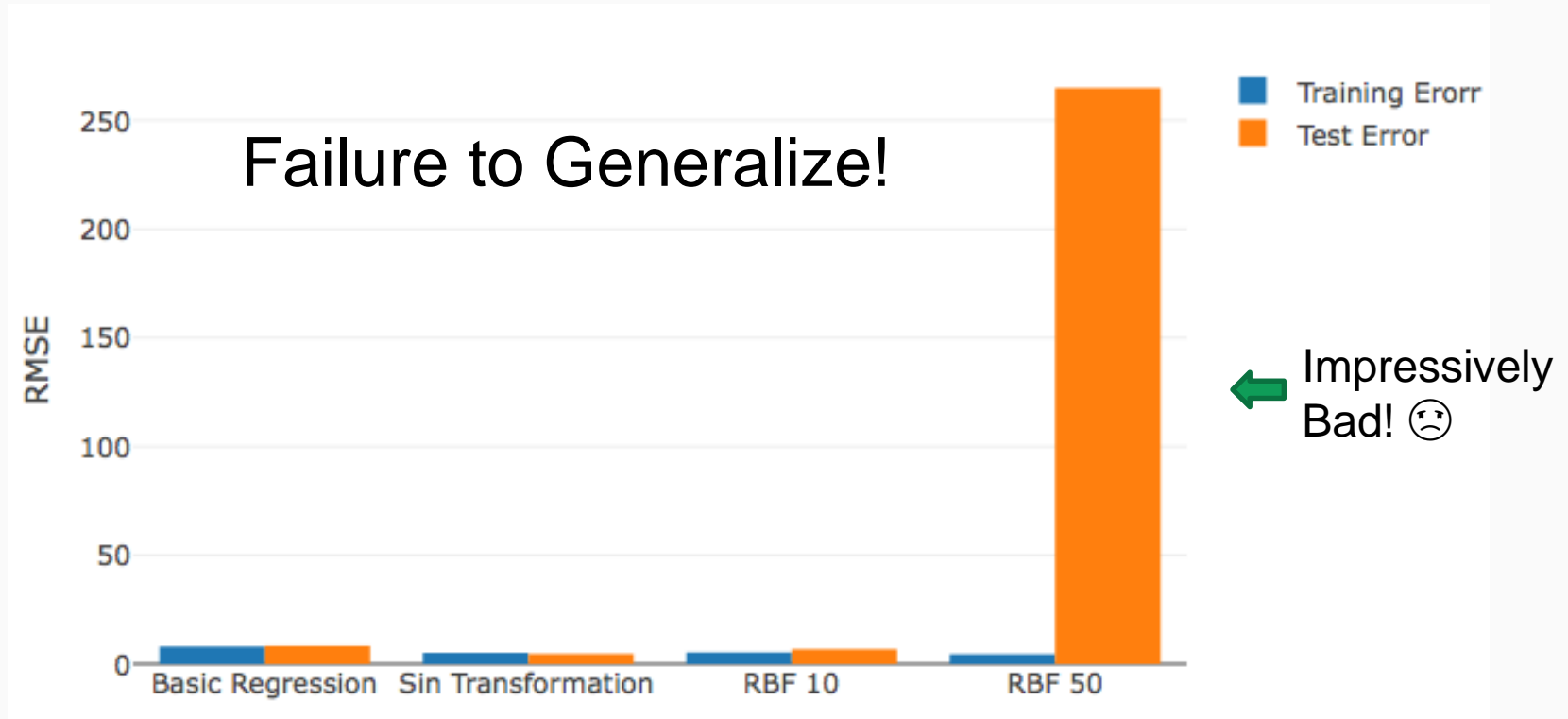
Fernando Perez fernando.perez@berkeley.edu

Training Error



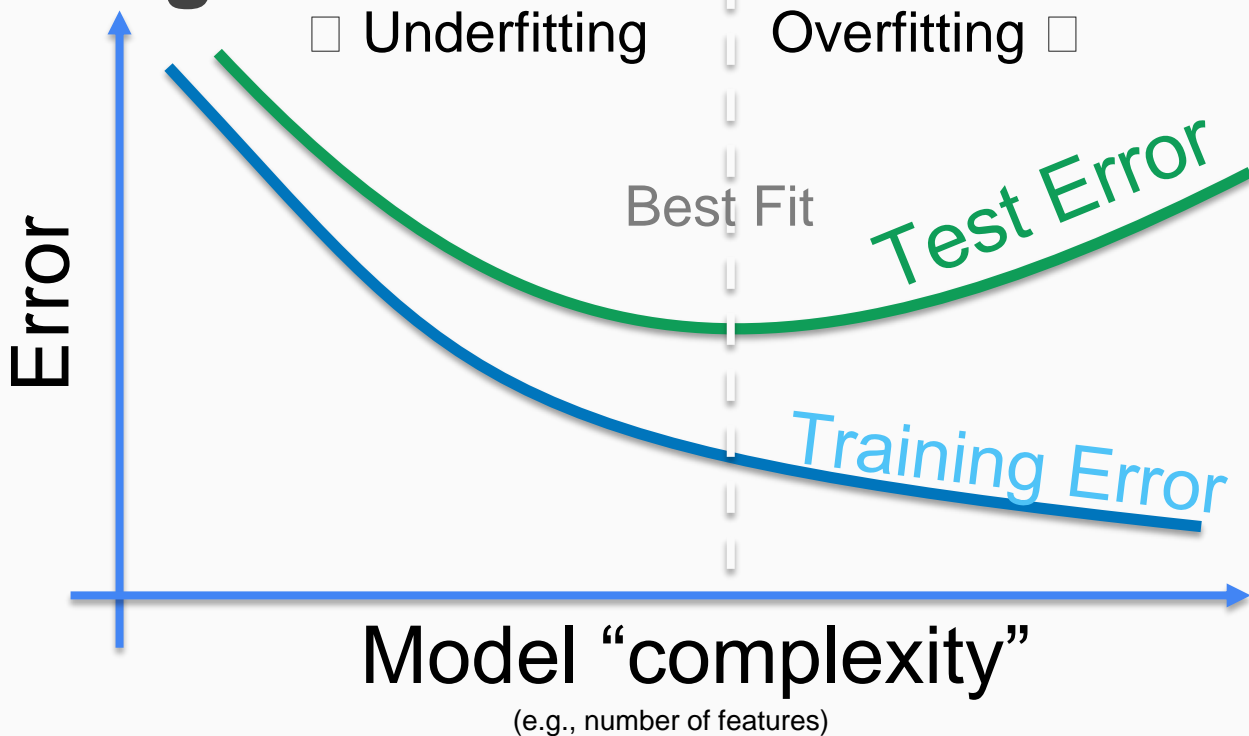
← Impressive!

Training vs Test Error



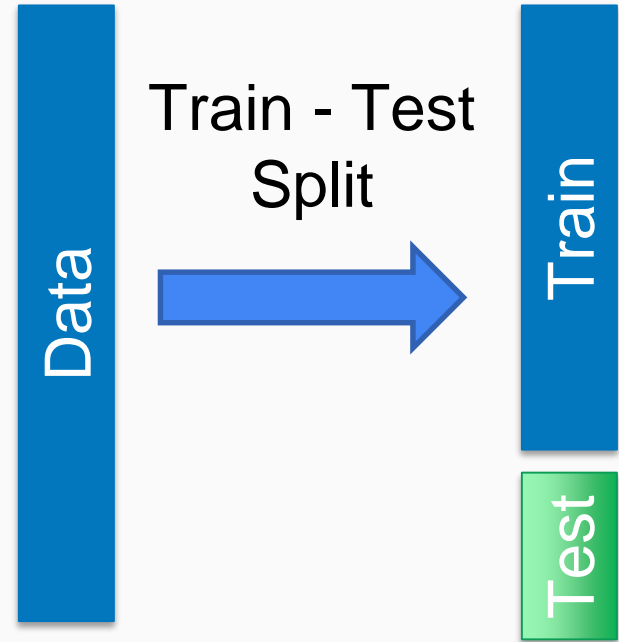
Training error typically
under estimates test error.

Training vs Test Error



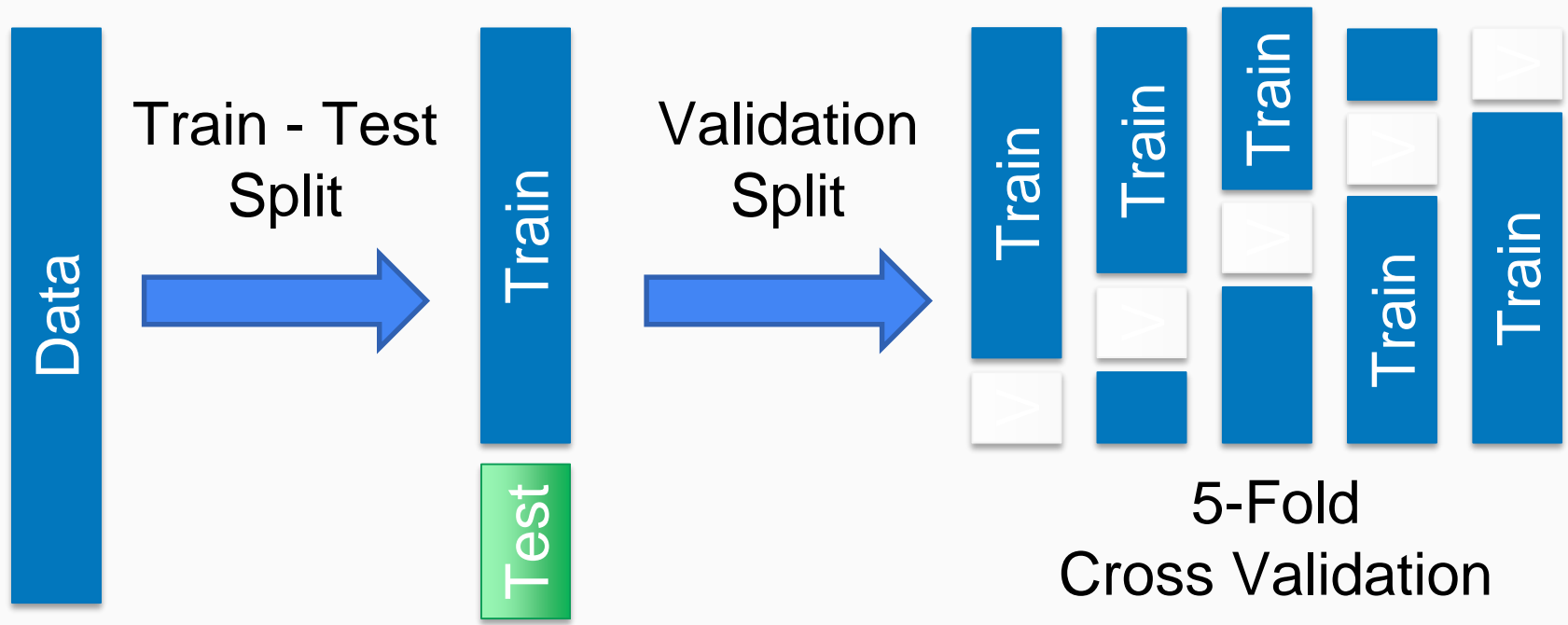
Generalization: *The Train-Test Split*

- **Training Data:** used to fit model
- **Test Data:** check generalization error
- How to split?
 - Randomly, Temporally, Geo...
 - Depends on application (usually randomly)
- What size? (90%-10%)
 - Larger training set \square more complex models
 - Larger test set \square better estimate of generalization error
 - Typically between 75%-25% and 90%-10%



You can only use the test dataset once after deciding on the model

Generalization: *Validation Split*



Cross validation ***simulates multiple train test-splits*** on the training data.

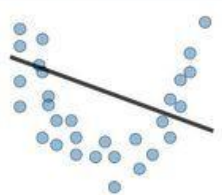


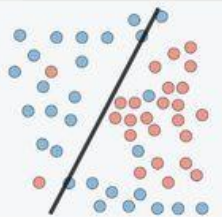
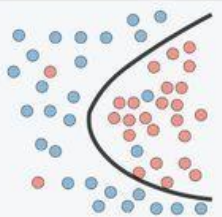
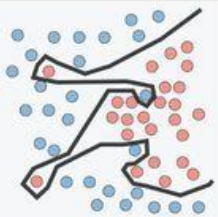



Recipe for Successful Generalization

1. Split your data into **training** and **test** sets (90%, 10%)
2. Use **only the training data** when designing, training, and tuning the model
 - Use **cross validation** to test *generalization* during this phase
 - **Do not look at the test data**
3. Commit to your final model and train once more using **only the training data**.
4. Test the final model using the **test data**. If accuracy is not acceptable return to (2). (*Get more test data if possible.*)
5. Train **on all available data** and ship it!



Bias/variação

- Ideia de bias/variação são fáceis de entender, mas difíceis de dominar.
 - Se o seu modelo underfit (regressão logística com dados não lineares) ele tem um "alto viés"
 - Se o seu modelo overfit, então ele tem uma "alta variação"
 - Seu modelo ficará bem se você equilibrar o bias/variação

	Underfitting	Just right	Overfitting
Symptoms	<ul style="list-style-type: none">• High training error• Training error close to test error• High bias	<ul style="list-style-type: none">• Training error slightly lower than test error	<ul style="list-style-type: none">• Very low training error• Training error much lower than test error• High variance
Regression illustration			
Classification illustration			
Deep learning illustration			
Possible remedies	<ul style="list-style-type: none">• Complexity model• Add more features• Train longer		<ul style="list-style-type: none">• Perform regularization• Get more data

- Se o seu algoritmo tem bias alto:
 - Tente uma RN maior (> hidden layers, > no de unidades)
 - Tente um modelo diferente que seja mais adequado para seus dados.
 - Tente rodar por mais tempo.
 - Tente algoritmos de otimização diferentes.
- Se seu algoritmo tiver uma high variance:
 - Obtenha mais dados.
 - Tente regularização.
 - Tente um modelo diferente que seja mais adequado para seus dados.

Problema é mais abstrato em redes neurais

- Em regressão linear, conseguimos visualizar o problema
- Em redes neurais, não é possível ver as coisas de maneira tão simples
- Mas a mesma coisa ocorre: alta variação dos coeficientes
- Solução é chamada de regularização
- Abordagens para regularização:
 - penalização L2
 - penalização L1
 - drop-out
 - data augmentation
 - early stopping

- Evitar a aparição de coeficientes com coeficientes extremamente altos ou extremamente baixos.
- Estes coeficientes são um sinal de redundância das features ou complexidade de explicação/predição
- Considere o vetor $\theta = (W^{[1]}, W^{[2]}, \dots, W^{[L]})$
- Tornamos as matrizes de pesos num longo vetor
- Observe que ele não possui os termos b's de bias

- Seja $\theta = (\theta_1, \theta_2, \dots, \theta_M) = (W^{[1]}, W^{[2]}, \dots, W^{[L]})$
- Ao invés de controlar cada coeficiente θ_i separadamente, usamos uma medida global dos seus “tamanhos”
- Duas normas:
 - Norma $\mathbb{L}_2 = ||\theta||_2^2 = \theta^T \cdot \theta = \sum_i \theta_i^2$
 - Norma $\mathbb{L}_1 = ||\theta||_1 = \sum_i |\theta_i|$
- O objetivo da regularização é que a norma do vetor de pesos estimado não seja grande demais

Regularização via penalização

- A função de custo/perda que queremos minimizar é

$$\begin{aligned} J(\theta, b^{[1]}, b^{[2]}, \dots, b^{[L]}) &= \sum_i \mathcal{L}(y^{(i)}, \hat{y}^{(i)}) \\ &= \sum_i (y^{(i)} - \hat{y}^{(i)})^2 \quad \text{regressao} \\ &= \sum_i \left(y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right) \quad \text{2-classificacao} \\ &= \sum_i \sum_k \left(I[y^{(i)} = k] \log(\hat{y}_k^{(i)}) \right) \quad \text{K-classificacao} \end{aligned}$$

- A função regularizada é

- Caso L1: $J(\theta, b^{[1]}, b^{[2]}, \dots, b^{[L]}) + \lambda ||\theta||_1$ Regularização LASSO (Least Absolute shrinkage and selection operator)

- Caso L2: $J(\theta, b^{[1]}, b^{[2]}, \dots, b^{[L]}) + \lambda ||\theta||_2^2$ Regularização RIDGE

- Ao invés de minimizar a -log-verossimilhança, nós minimizamos a -log-verossimilhança mais um termo de penalização
- Penalizamos vetores de pesos “grandes demais”
- O parâmetro lambda controle a importância do termo de penalização
- É chamado de (um dos) hiper-parâmetro do modelo
- Papel de lambda: Na prática, ele penaliza grandes pesos e limita efetivamente a liberdade de escolher pesos em nossos modelos.

$$J(\theta, b^{[1]}, b^{[2]}, \dots, b^{[L]}) + \lambda ||\theta||_2^2$$

$$J(\theta, b^{[1]}, b^{[2]}, \dots, b^{[L]}) + \lambda ||\theta||_2^2$$

- Se $\lambda = 0$, temos a situação anterior, sem penalização, talvez levando a overfitting, com pesos grandes demais
- Se $\lambda \gg 0$, teremos uma penalização excessiva.
- O segundo termo vai dominar a soma: na prática, vamos minimizar apenas o tamanho do vetor de pesos levando a um vetor muito próximo de zero.
- Acabaremos induzindo um alto bias no nossos modelos ao forçar a maioria dos pesos a ser muito próxima de zero
- Se lambda for OK, ele reduzirá apenas alguns pesos que levam a overfit da

- Backpropagation não muda muito
- Por exemplo, com L2

$$\theta_j^{t+1} = \theta_j^t - \alpha \left[\frac{\partial J}{\partial \theta_j} \Big|_{\theta_j^t} + 2\lambda \theta_j^t \right]$$

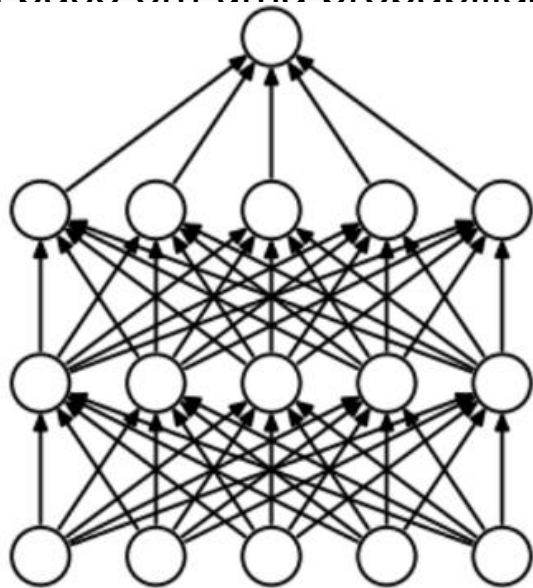
- Com L1:

$$\theta_j^{t+1} = \theta_j^t - \alpha \left[\frac{\partial J}{\partial \theta_i} + \lambda \left(I[\theta_j^t > 0] - I[\theta_j^t < 0] \right) \right]$$

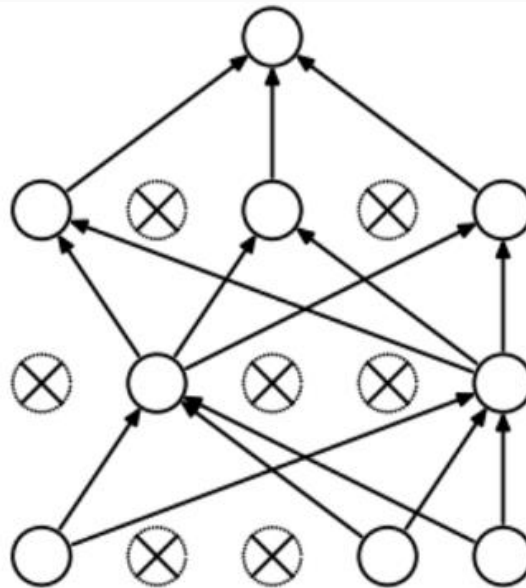
Como escolher λ

- Escala logarítmica:
- Escolha 12 valores para λ \square 0, 0.01, 0.02, 0.04, 0.08, 0.16, 0.32, 0.64, 1.28, 2.56, 5.12, 10
- Rode o conjunto de treinamento para cada lambda
- Calcule a função de custo com o conjunto de validação para checar a qualidade do resultado
- Escolha o valor de lambda que seja o melhor.

- A regularização via dropout elimina alguns neurônios/pesos em cada iteração com base em uma probabilidade.



(a) Standard Neural Net



(b) After applying dropout.

Extraído de

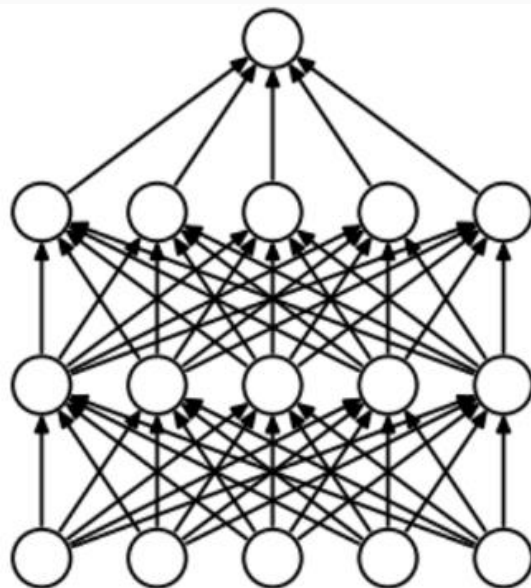
Srivastava,
Hinton,
Krizhevsky,
Sutskever,
Salakhutdinov.

Dropout: a
simple way to
prevent neural
networks from
overfitting.

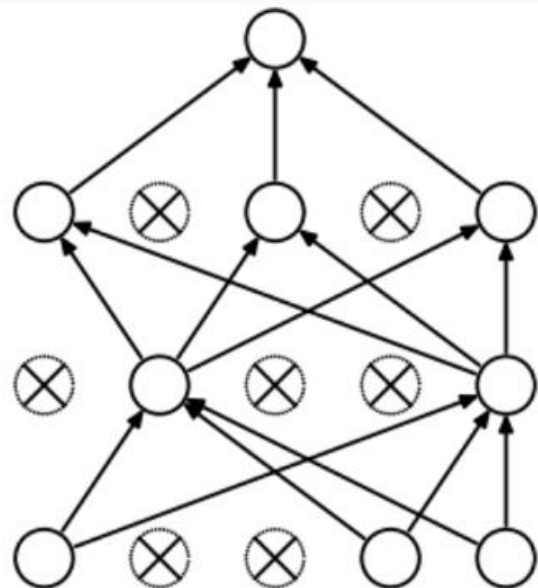
JMLR 2014

Dropout aleatoriamente em cada ciclo de iterações

- Ao descartar um neurônio, removemos a unidade temporariamente da rede, juntamente com todas as suas conexões de entrada e de saída.



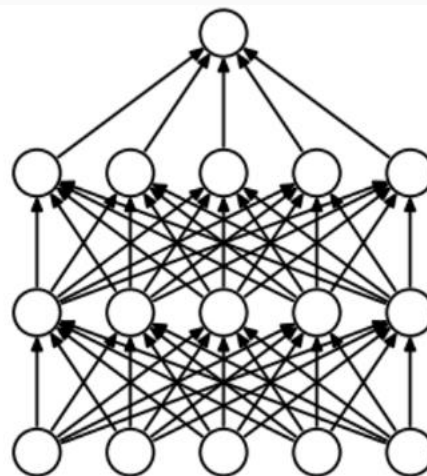
(a) Standard Neural Net



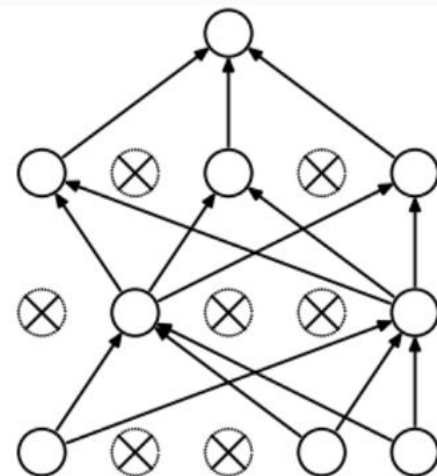
(b) After applying dropout.

Qual a probabilidade usada para descartar unidades?

- Do paper/patente: use probabilidade $p = \frac{1}{2}$ para reter um neurônio interno (escondido) de camadas com muitas unidades.
- Use p aprox 1 para reter neurônios das camadas finais com muito poucos neurônios
- Use probab $p=0.8$ para reter features da camada visível. (Ng recomenda $p=1.0$)



(a) Standard Neural Net



(b) After applying dropout.

Como implementar? Dropout invertido

- Fase de Treinamento:
 - Para cada camada oculta, para cada amostra de treinamento, para cada iteração, ignore (zero out) uma fração aleatória, p , de nós (e as ativações correspondentes).
 - Use "Dropout invertido" para compensar: multiplique os valores das ativações que sobram pelo fator de dropout inverso $1/p$.
- Suponha $p=0.8$ e uma camada com 50 unidades → em média restarão 40
- Ao passar para a próxima camada, o valor z vai usar apenas 40 neurônios/ativados → multiplique-os por $1/0.8 = 1.25$ para manter o valor de z da próxima camada igual (em valor esperado) ao valor SEM dropout.

Como implementar? Dropout invertido

- Fase de teste:
 - Use todos os neurônios, não delete nenhum.
 - E os pesos a serem usados?
 - Use o valor mais recente obtido para cada peso quando ele não esteve deletado.
 - Não precisa compensar pela probabilidade p .
 - Isto foi feito na fase de treinamento.
- Paper original era um pouco mais complicado.

Implementing dropout ("Inverted dropout")

Illustrate with layer $l=3$. $\text{keep-prob} = \underline{0.8}$ 0.2

$\rightarrow \underline{d3} = \text{np.random.rand}(a3.\text{shape}[0], a3.\text{shape}[1]) < \underline{\text{keep-prob}}$

$\underline{a3} = \text{np.multiply}(a3, d3)$ # $a3 \times d3$.

$\rightarrow \boxed{a3 /= \text{keep-prob}}$ \leftarrow

50 units \rightarrow 10 units shut off

$$z^{(3)} = w^{(2)} \cdot \underline{a^{(2)}} + b^{(3)}$$

\uparrow \uparrow scaled by 20%.

$\underline{/= 0.8}$

Test

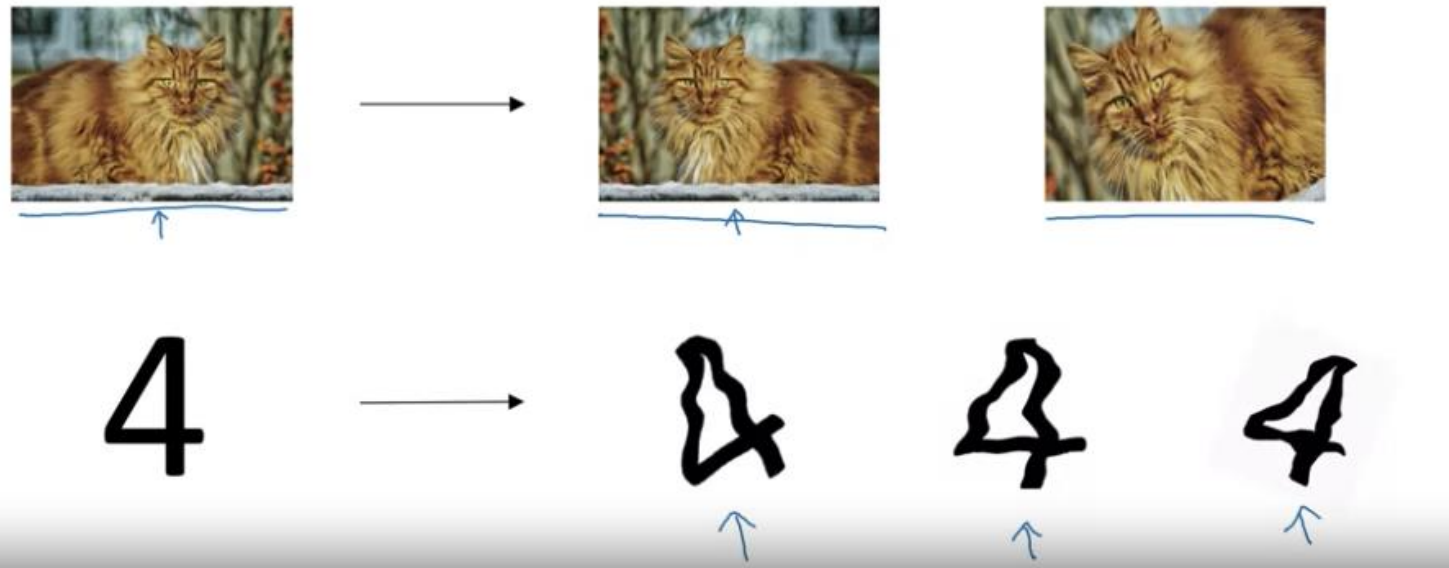
- O dropout força uma rede neural a aprender quais são os neurônios mais robustos.
- São aqueles neurônios que são importantes em muitos subconjuntos aleatórios diferentes de outros neurônios.
- O dropout praticamente dobra o número de iterações necessárias para convergir.
- No entanto, o tempo de treinamento para cada exemplo é menor pois a rede é mais simples.

Data augmentation



Slide de Andrew Ng

Data augmentation

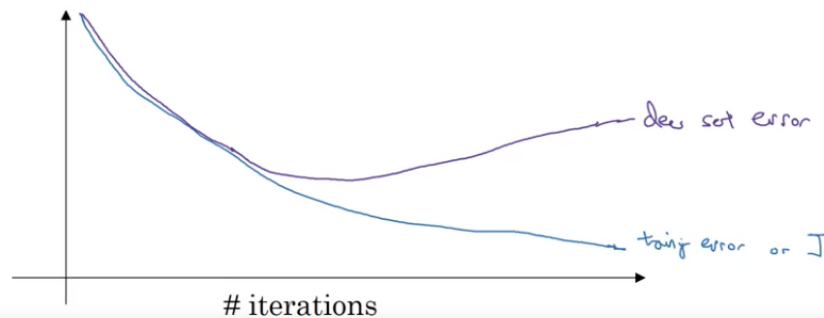


Slide de Andrew Ng

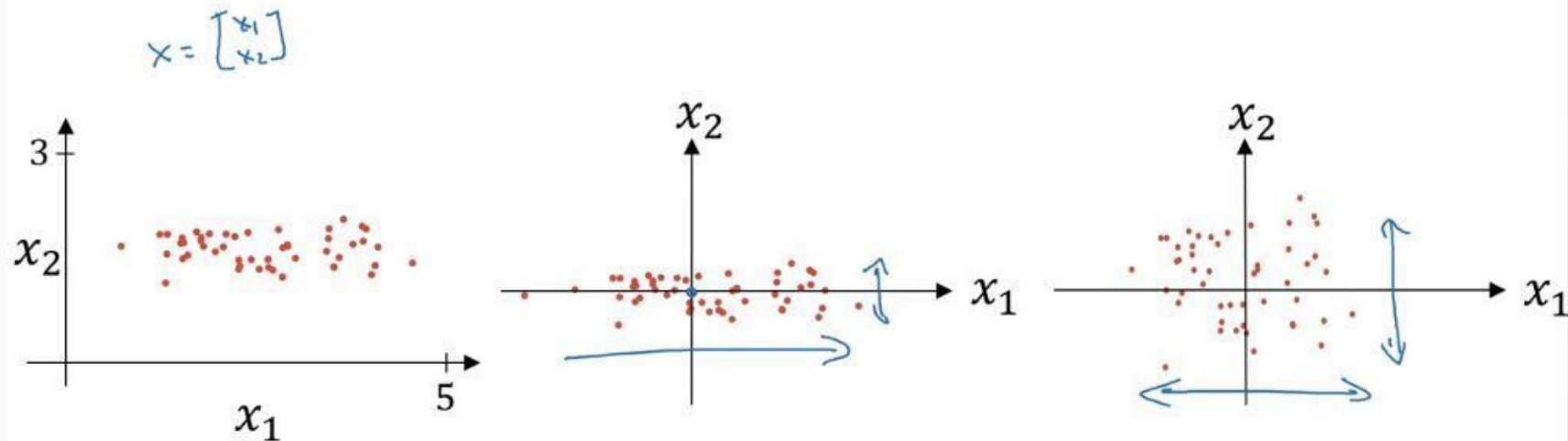
Outras maneiras de regularizar: early stopping

- Divida os dados de treinamento em um conjunto de treinamento e um conjunto de validação.
- Treinar apenas no conjunto de treinamento.
- Avaliar o erro no conjunto de validação de vez em quando (a cada 5 iterações, por exemplo)
- Pare o treinamento assim que o erro no conjunto de validação subir.
- Use os pesos que a rede tinha na etapa anterior como resultado da execução do treinamento.

Early stopping



Normalizando as entradas



Subtract mean:

$$\mu = \frac{1}{n} \sum_{i=1}^m x^{(i)}$$
$$x := x - \mu$$

Normalize variance

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^m x^{(i)} * x^{(i)T}$$

← element-wise

$$x /= \sigma^2$$

Use same μ σ^2 to normalize test set.

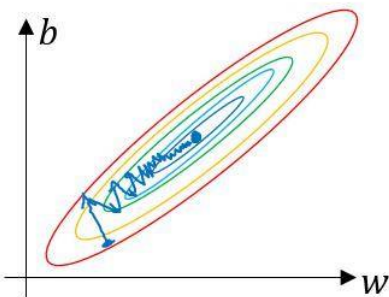
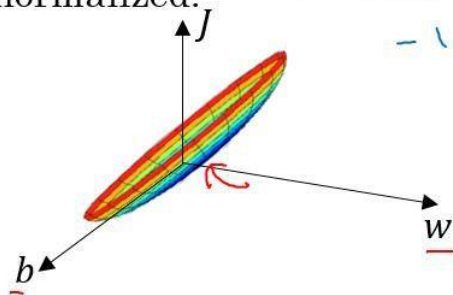
Andrew Ng

Why normalize inputs?

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

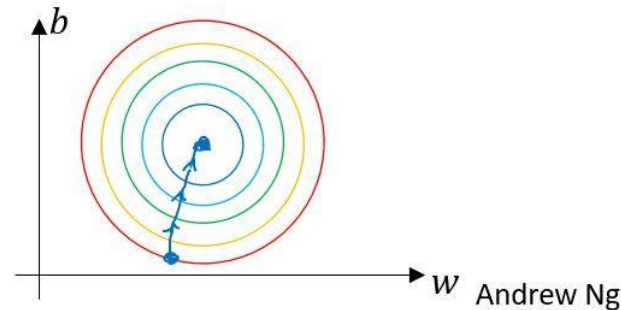
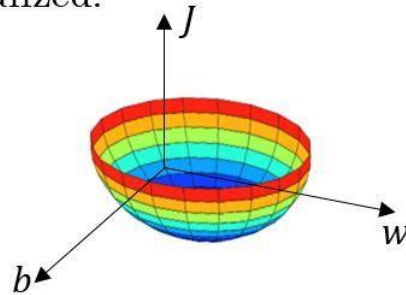
Unnormalized:

$w_1, x_1: 1 \dots 1000 \leftarrow$
 $w_2, x_2: 0 \dots 1 \leftarrow$
 $-1 \dots 1$



$x_1: 0 \dots 1$
 $x_2: -1 \dots 1$
 $x_3: 1 \dots 2$

Normalized:



Andrew Ng